ECS 171 Machine Learning

Lecture2: Linear Regression, Cross Validation, Curve-Fitting, RSS, OLS, GD

Instructor: Setareh Rafatirad

Parametric and Non-parametric Models in Machine Learning

Supervised Learning Algorithms

Decision Trees

RBF kernel SVMs

kNN

Non-parametric Learners

Parametric Learners

Linear models (linear regression, logistic regression, and linear SVM).

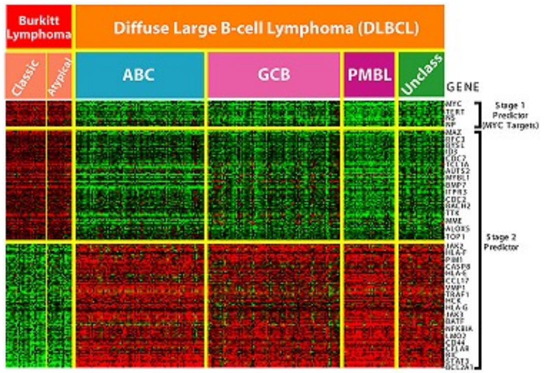
- (potentially) infinite number of parameters
- Lazy learners
- We need to store the training data
- does not estimate the parameters of a model during a training phase
- Can predict immediately
- Costly prediction

- finite number of parameters (fixed structure)
- Eager learners
- Once parameters (weights) of the model are learned, we no longer keep the training data.
- Training is computationally costly
- Inexpensive prediction

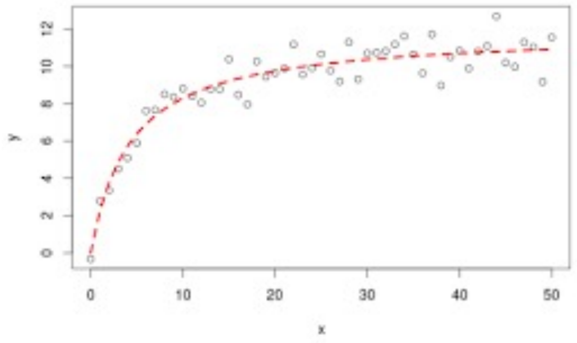
Basic Concepts in ML

- Linearity
- Dataset description
- Independent variables (Feature Vector)
- High-dimensional data
- Feature selection
- Overfitting , underfitting
- Error – variance trade-off
- Evaluation

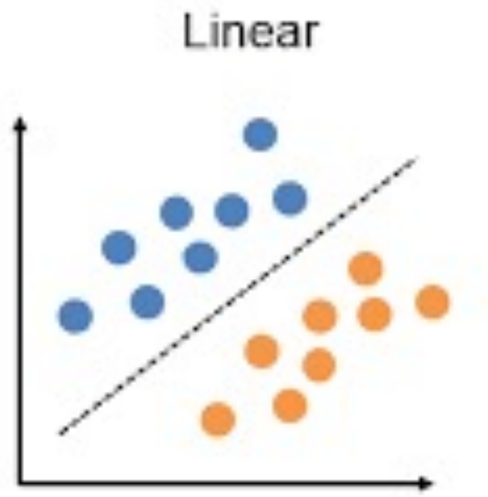
High dimensional Data

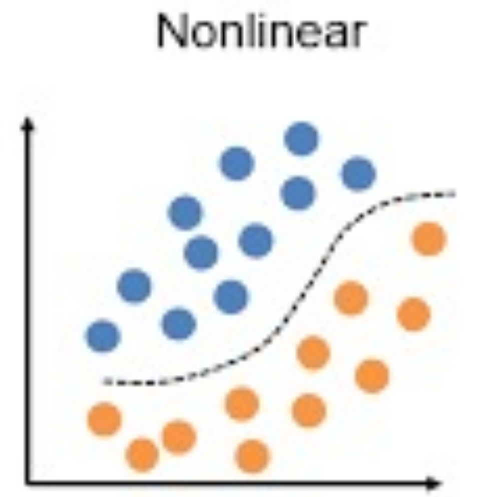


often in linear ML algorithms, we have a high bias but a low variance, or in Nonlinear ML algos, we often have a low bias but a high variance.



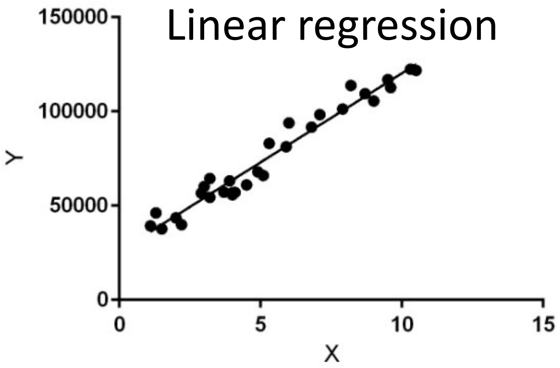
Nonlinear Regression



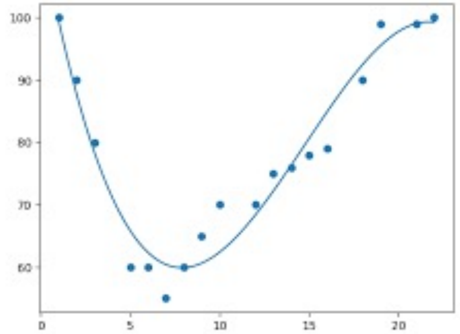


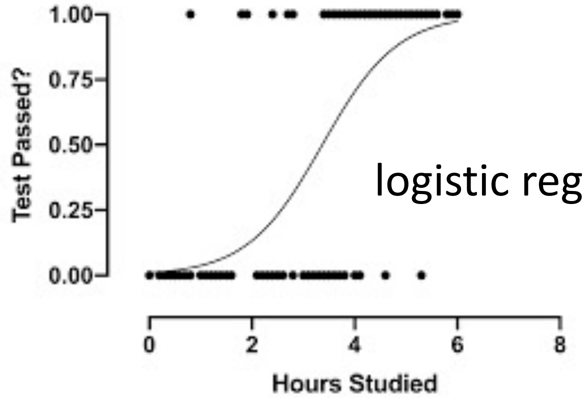
Regression Problem Setting

- Predicting sales for a particular produ
- Data set Description
  - Attribute(s) of the data set (X) includes
    - advertising budget (dollar value)
  - Output Y i.e., the class attribute
    - sales in thousands of units



polynomial regression





logistic regression

Linear regression finds out a linear relationship between X (input) and Y(output).

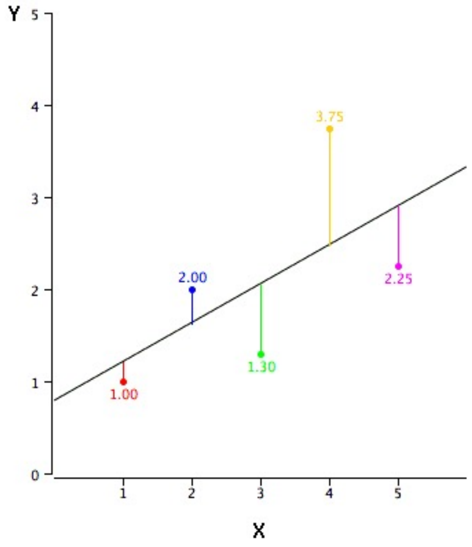
Output sales (dependent variable)

Goal: find f(X)=Y

Advertisement budget (independent variable)

Linear Regression Model

- Supervised learning
- Popular statistical learning method
- Predicts a quantitative response Y from predictive attribute X
- Linear relationship between X and Y



$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

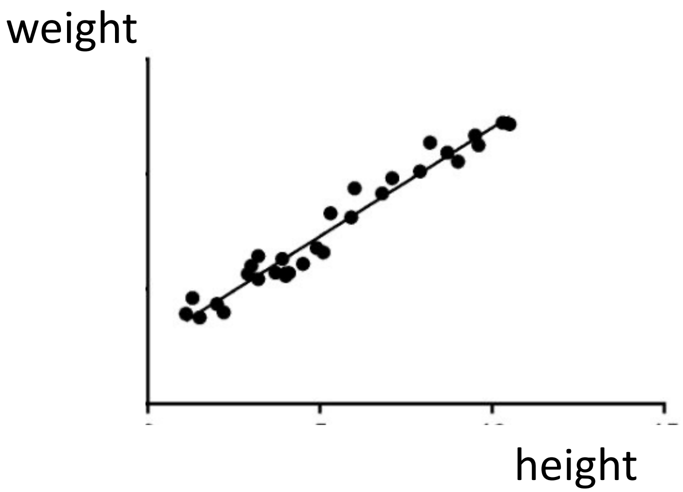
Output

intercept

model coefficients (model parameters)

When training the model – it fits the best line to predict the value of Y (output) for a given value of X (features). The model gets the best regression fit line by finding the best coefficient values.

Linear Regression Categories



Simple LR

Independent variables (X_i)

Target Variable (Y)

Temperature	Humidity	Yield
50	57	112
53	54	118
54	54	128
55	60	121
56	66	125
59	59	136
62	61	144
65	58	142
67	59	149
71	64	161
72	56	167
74	66	168
75	52	162
76	68	171
79	52	175
80	62	182

Multiple LR

	red-meat	fish	cholesterol	blood pressure		weight
	X_1	X_2	Y_1	Y_2	$\ldots$	Y_m
$\boldsymbol{x}_1$	5.0	4.5	1	1		0
$\boldsymbol{x}_2$	2.0	2.5	0	1		0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{x}_n$	3.0	3.5	0	1		1
$\boldsymbol{x}$	4.0	2.5	?	?		?

For a given x, predict the vector

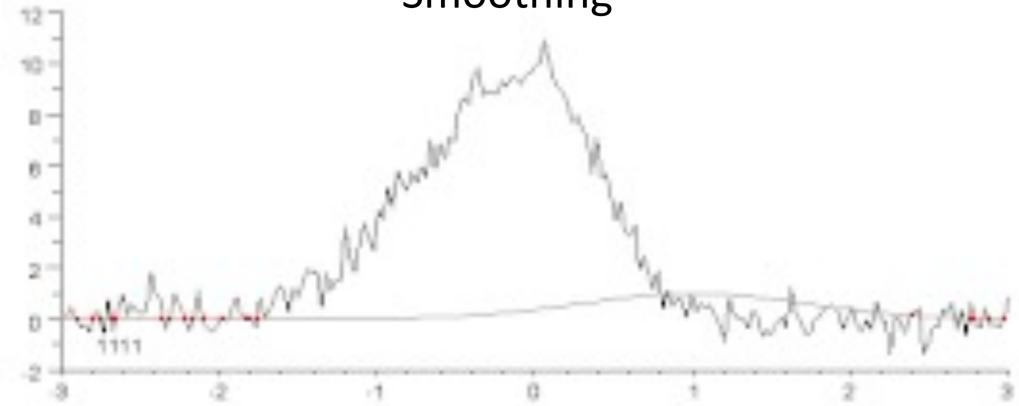
$$Y = (Y_1, Y_2, \ldots, Y_m)$$

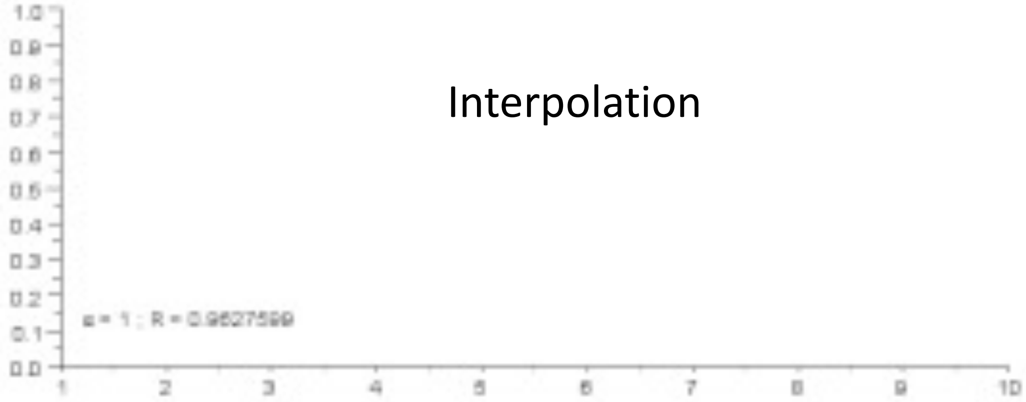
Multivariate LR aka General LR

Curve Fitting

- In regression analysis, curve fitting is the process of finding a model that produces the best fit with the lowest error to the relationships between the variables of a dataset.
- Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points.
  - **Interpolation** where an exact **fit** to **the** data is required
  - **Smoothing** in which a "smooth" function is constructed that approximately fits the data.

Smoothing

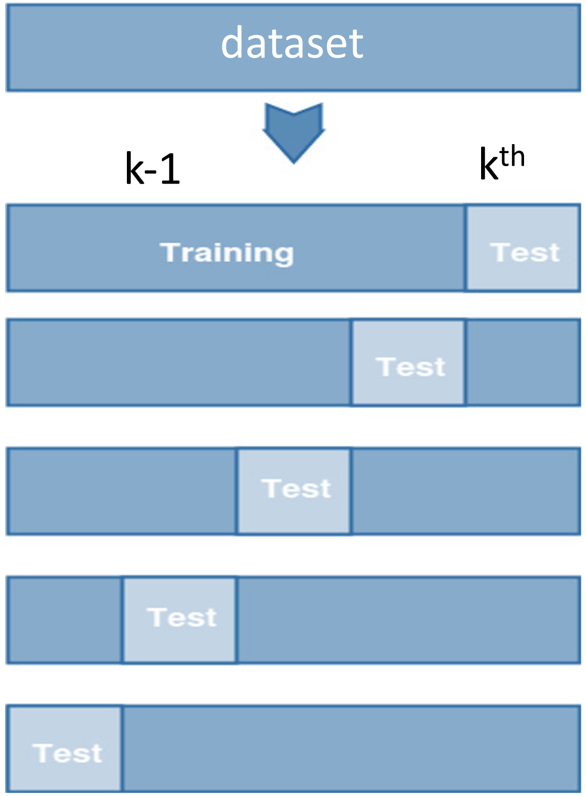




Interpolation

Cross Validation

- Cross validation is method to avoid producing biased models
  - A resampling procedure to help the model to generalize well
  - Has a single parameter called k for the number of partitions
  - k-fold cross validation
  - Procedure for k-fold cross validation:
    1. Randomize the dataset and create k equal size partitions
    2. Use k-1 partitions for training the model
    3. Use the k^{th} partition for testing and evaluating the model
    4. iterate k times with a different subset reserved for testing purpose each time.
  - Some commonly used variations on cross-validation are stratified and repeated are available in scikit-learn.




```
from sklearn import cross_validation

# value of K is 10.
data =
cross_validation.KFold(len(train_set)
, n_folds=10, indices=False)
```

Example for using Cross Validation with Linear Regression

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

boston= datasets.load_boston()
X = boston.data
y = boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

reg = LinearRegression()
'''LinearRegression will take in its fit method arrays X, y and will store -
   the coefficients w of the linear model in its coef_ member:'''
regmodel= reg.fit(X_train, y_train)
# The coefficients
print('Coefficients: \n', regmodel.coef_)
reg_predictions=reg.predict(X_test)

# The mean squared error
print('Mean squared error: %.2f' % mean_squared_error(y_test, reg_predictions))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f' % r2_score(y_test, reg_predictions))

print('Cross Validation: ')
# Array of scores of the estimator for each run of the cross validation.
# cv: Determines the cross-validation splitting strategy
scores = cross_val_score(LinearRegression(), X, y, cv=7)
print(scores)
# report performance
print('Accuracy: %.3f (%.3f)' % (scores.mean(), scores.std()))
print('Mean squared error: %.3f' % (np.mean(np.abs(scores))))
```

```
Coefficients:
 [-1.17735289e-01  4.40174969e-02 -5.76814314e-03  2.39341594e+00
 -1.55894211e+01  3.76896770e+00 -7.03517828e-03 -1.43495641e+00
  2.40081086e-01 -1.12972810e-02 -9.85546732e-01  8.44443453e-03
 -4.99116797e-01]
Mean squared error: 29.78
Coefficient of determination: 0.64
Cross Validation:
[ 0.6534446   0.59631685  0.67865382  0.61193393  0.51940514 -0.30577687
  0.4001835 ]
Accuracy: 0.451 (0.321)
Mean squared error: 0.538
```

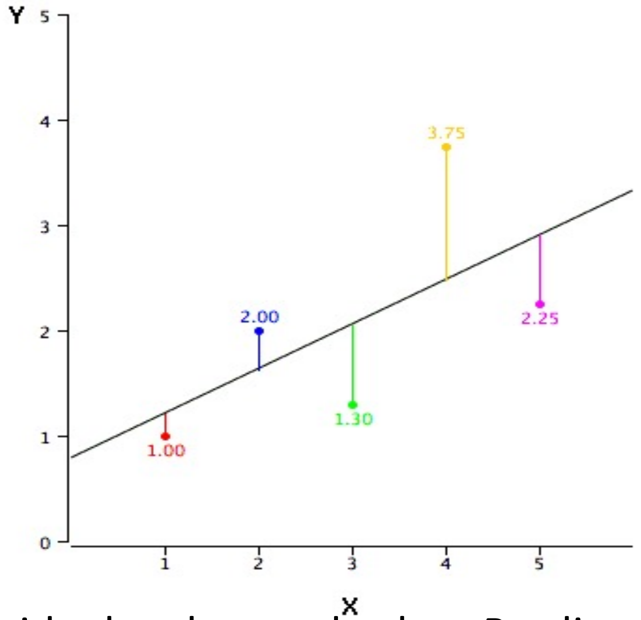
Cost Function

- When training the model, the goal is to minimize the error and update the model coefficients to achieve the best fit line.
- Error is the difference between predicted value (Y) generated by the model and the class attribute value.
- Cost function L is used to measure the error:

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

Observed value Predicted value

Minimize the Loss!



Residual = observed value- Predicted value

Method 1: Ordinary Least Squares (OLS)

Method 2: Gradient Descent (GD)

Linear Regression: Formulation

Given a dataset $\boldsymbol{D}$ with $\boldsymbol{m}$ observations: $D = \{\,(x^i, y^i)\,; 1 \le i \le m\}$

where $x^i = \underbrace{\{x_1^i, x_2^i, \ldots, x_n^i\}}_{\text{n= number of attributes}}$

$$\begin{array}{cc} \text{Independent variables} & \text{Dependent variable} \\ \boldsymbol{D} = \left\{ \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_n^{(2)} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}, \right. & \left. \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(m)} \end{bmatrix} \right\} \end{array}$$

$$w_0 x_0^{(1)} + w_1 x_1^{(1)} + w_2 x_2^{(1)} + \ldots + w_n x_n^{(1)} = \sum_{j=0}^{n} w_j x_j^{(1)}$$

Linear Regression: Formulation cont.

$D = \{ (x^i, y^i) ; 1 \le i \le m\}$

$$\begin{array}{cc} m \text{ x } (n+1) & m \text{ x } 1 \\ \end{array}$$

$$\boldsymbol{D} = \left\{ \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_n^{(2)} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}, \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(m)} \end{bmatrix} \right\}$$

if $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \cdots \\ x_n^{(i)} \end{bmatrix}$ is the i^th observation then $D = \left\{ \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \cdots \\ (x^m)^T \end{bmatrix}, \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(m)} \end{bmatrix} \right\}$ = $\{X, Y\}$

Transpose Operator Overview

$$A = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \Rightarrow A^T = [a\ b\ c\ d]$$

1 x 4

4x1

$$(A+B)^T = A^T + B^T$$
$$(AB)^T = B^T A^T$$
$$A^T . A = \sum_i a_i^2$$
$$(cA)^T = cA^T$$

Linear Regression Objective

- Goal: construct a mapping function f(X) : X → Y
- To predict a quantitative response Y for unseen input instances

$$w_0 x_0^{(i)} + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_n x_n^{(i)} = \sum_{j=0}^{n} w_j x_j^{(i)}$$

optimal parameter values

The weights are represented as a n-dimensional vector w

$$W = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{bmatrix}$$

Objective: how to find w_j?

f(X; W) : X → Y

Linear Regression: what else?

- What is the form of the target function?
- What is the relationship between input (X) and output (Y)?

$$y^{(i)} = w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots + w_n x_n^{(i)} = \sum_{j=0}^{n} w_j x_j^{(i)} = w^T x^{(i)} = f(x^{(i)}; w)$$

$$W = \begin{bmatrix} w_0 \\ w_1 \\ \ldots \\ w_n \end{bmatrix}$$

Weight vector

$$x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \ldots \\ x_n^{(i)} \end{bmatrix}$$

Input vector for the i[th] sample

Linear Regression: Problem Formulation

In an imperfect world, we may not find the weight vector *W* such that the output is exactly expressed as the linear function of the input for each sample *(i)* !

Residual error $\epsilon^{(i)}$

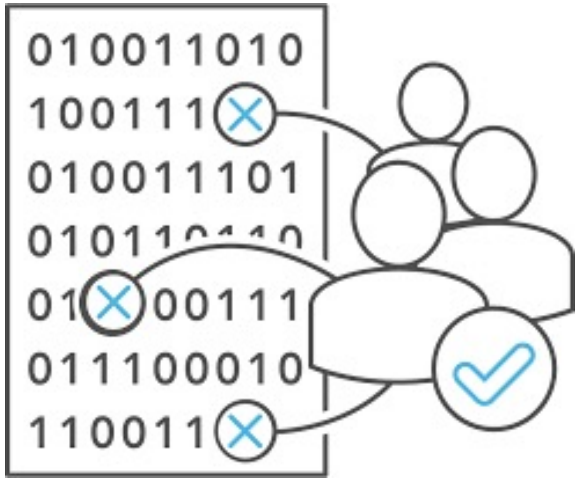
$$y^{(i)} = f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) + \epsilon^{(i)} = \boldsymbol{w}^T \boldsymbol{x}^{(i)} + \epsilon^{(i)} = \sum_{j=0}^{n} w_j x_j^{(i)} + \epsilon^{(i)}$$

- $\epsilon^{(i)} = (\text{real value of } y^{(i)}) - (\text{predicted } y^{(i)} \text{ value}) \rightarrow$

$$\epsilon^{(i)} = y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}$$

$$\underline{RSS} = \sum_{i=1}^{m} (\epsilon^{(i)})^2 = \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2$$

Residual Sum of Squares



Minimizing RSS in LR: Optimization Problem

- In Linear Regression, the basic assumption is that with minimizing the RSS, the relationship between input and output can be outlined in the best possible way.

$$\boldsymbol{w} \triangleq \underset{\boldsymbol{w}}{\operatorname{argmin}}\, RSS = \underset{\boldsymbol{w}}{\operatorname{argmin}}\left(\sum_{i=1}^{m}\left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2\right)$$

- How to minimize the RSS?
    1. **Ordinary Lease Squares (OLS) : Method 1 – Analytical approach**
    2. **Gradient Descent (GD) : Method 2 – Numerical approach**

Method1 : Ordinary Least Squares

$$RSS = \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2$$

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix} - \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & \dots & x_n^{m} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{bmatrix} = \begin{bmatrix} y^{(1)} - \sum_{j=0}^{n} w_j \, x_j^{(1)} \\ y^{(2)} - \sum_{j=0}^{n} w_j \, x_j^{(2)} \\ \dots \\ y^{(m)} - \sum_{j=0}^{n} w_j \, x_j^{(m)} \end{bmatrix} = \begin{bmatrix} y^{(1)} - w^T x^{(1)} \\ y^{(2)} - w^T x^{(2)} \\ \dots \\ y^{(m)} - w^T x^{(m)} \end{bmatrix} = (Y - Xw)$$

Minimize RSS: OLS cont.

1) $(A+B)^T = A^T + B^T$
2) $(AB)^T = B^T A^T$
3) $A^T . A = \sum_i a_i^2$
4) $(cA)^T = cA^T$

$$(Y - Xw) = \overbrace{\begin{bmatrix} y^{(1)} - w^T x^{(1)} \\ y^{(2)} - w^T x^{(2)} \\ \dots \\ y^{(m)} - w^T x^{(m)} \end{bmatrix}}^{\text{A}}$$

$$RSS = \sum_{i=1}^{m} (y^{(i)} - w^T x^i)^2 = (Y - Xw)^T (Y - Xw)$$

Write your verification of rule (3) using the residual matrix (A).

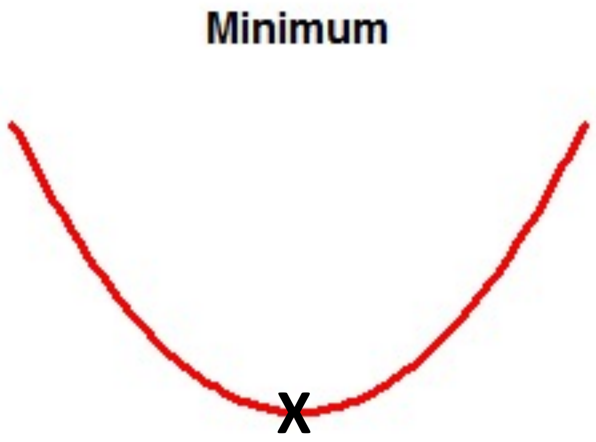
OLS Solution to Minimize RSS for LR

- Differentiation finds the point at which a function (like f(x) i.e., RSS) is minimum.
- Similarly, we differentiate the RSS w.r.t. w, and set it to 0.

$$\frac{\partial RSS}{\partial w} = 0$$

$$\nabla_w[(Y - Xw)^T(Y - Xw)] = 0 \Rightarrow$$
$$2X^T(Y - Xw) = 0 \Rightarrow$$

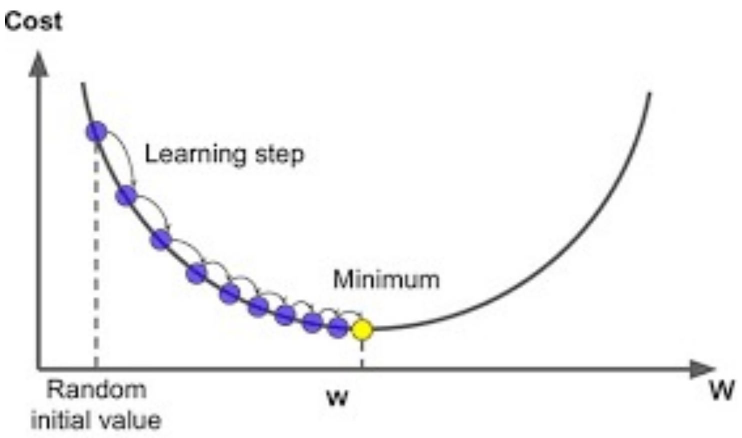
$$w = (X^TX)^{-1}X^TY = \frac{X^TY}{X^TX}$$



$f'(x) = 0$

Method2: Gradient Descent (GD)

Loss function has the shape of a bowl.



- Gradient Descent is an iterative algorithm used as another way to find the weights that minimize RSS.
- learning rate hyperparameter is an important parameter in Gradient Descent, which is the size of the steps.
- Pros
  - No need to know matrix algebra
  - Easy to understand
  - Heuristic approach (stochastic optimization)
- Cons
  - Can take many cycles before converging
  - No guarantee to give optimal solution

How GD works?

- Start with some random values of w (i.e., model parameters)
- Keep updating the model parameters iteratively to reduce the *RSS* until achieving the minimum cost.

Method2: Gradient Descent (GD)

- How GD works?
  - Start wit some random values of w (i.e., model parameters)
  - Keep updating the model parameters iteratively to reduce the *RSS* until achieving the minimum cost.

Moving w towards the direction that will **minimize the RSS**

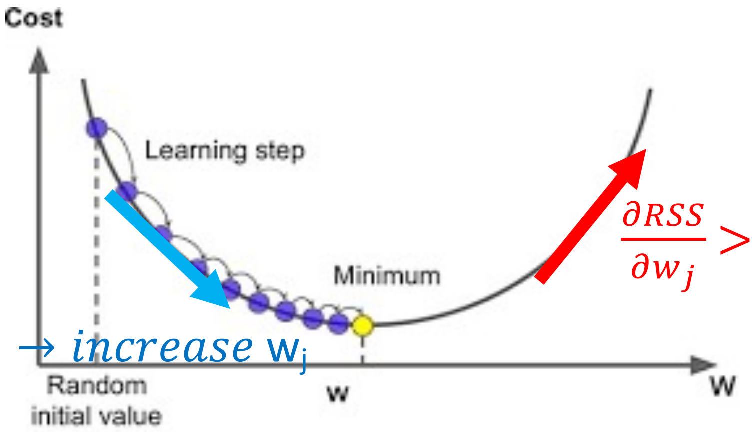
- if $\frac{\partial RSS}{\partial w_j} > 0 \Rightarrow \ decrease\ w$
- if $\frac{\partial RSS}{\partial w_j} < 0 \Rightarrow increase\ w$

Learning rate

Updated model parameter

$$w_j = w_j - \alpha \frac{\partial RSS}{\partial w_j}$$

Current model parameter



$\frac{\partial RSS}{\partial w_j} < 0 \rightarrow increase$ w_j

$\frac{\partial RSS}{\partial w_j} > 0 \rightarrow decrease$ w_j

Overview of Differentiation Rules

Differentiation Rule	f(x)	f'(x)
constant rule	$y = 5$	$\frac{\partial y}{\partial x} = 0$
power rule	$y = x^5$	$\frac{\partial y}{\partial x} = 5x^4$
constant multiple rule	$y = 4x^3$	$\frac{\partial y}{\partial x} = 12x^2$
sum rule	$y = x^6 + x^3$	$\frac{\partial y}{\partial x} = 6x^5 + 3x^2$
product rule	$y = e^{3x} sinx$	$\frac{\partial y}{\partial x} = e^{3x}(3sinx + cosx)$

https://www.mathsisfun.com/calculus/derivatives-rules.html

Application of GD Update Rule for 1 sample

$$w_j = w_j - a\frac{\partial RSS}{\partial w_j} \Rightarrow$$

$$w_j = w_j - a\frac{\partial (y^{(i)} - w^T x^{(i)})^2}{\partial w_j} \Rightarrow$$

$$w_j = w_j - a\frac{\partial (y^{(i)} - \sum_{k=0}^{n} w_k x_k^{(i)})^2}{\partial w_j} \Rightarrow$$

Update proportional to error

Next wj

constant

$$w_j = w_j + 2a\boxed{(y^{(i)} - \sum_{k=0}^{n} w_k x_k^{(i)})\ x_j^{(i)}}$$

Current wj

GD Update Rule for 1 sample cont.

- Also called the **Least Mean Squares (LMS) update rule** (or **Widrow-Hoff** learning rule).

$$w_j \;=\; w_j + 2a\,\left(y^{(i)} - \sum_{k=0}^{n} w_k x_k^{(i)}\right)\; x_j^{(i)}$$

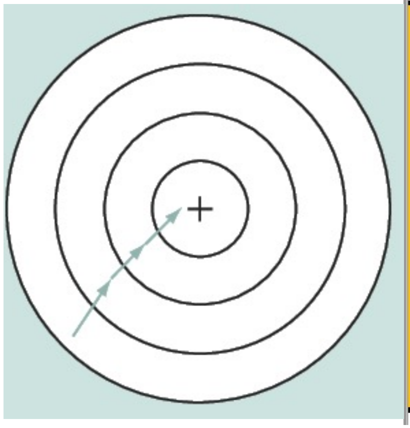
$$\Downarrow$$

$$w_j \;=\; w_j + a\,\left(y^{(i)} - w^T x^{(i)}\right) x_j^{(i)}$$

GD Update Rule for *m* samples

- There are 2 ways to deal with m samples:

Batch gradient descent



Repeat until convergence:

{ *for j=1 to n*

$$w_j := w_j + a \sum_{i=1}^{m} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}) x_j^{(i)}$$

}

Always converges

Stochastic gradient descent

Repeat until convergence:

{ *for i= 1 to m*

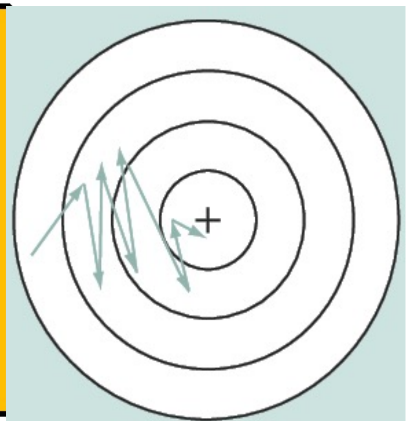
{ *for j=1 to n*

$$w_j := w_j + a (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}) x_j^{(i)}$$

}

}

For 1 epoch



Can take many cycles to converge, or never converge.

Optional Activity Outline

- Stochastic Gradient Descent
  - Create the log table containing these columns and fill out the table for 1 epoch for the dataset provided in the example below. The Python code for multiple epochs is provided in the sample code for Activity_Stochastic GD on Canvas.
  - Source: https://towardsdatascience.com/step-by-step-tutorial-on-linear-regression-with-stochastic-gradient-descent-1d35b088a843
- Batch Gradient Descent
  - Use the coding sample for Activity-Stochastic GD posted on Canvas and change it to implement Batch Gradient Descent.
  - For Batch Gradient Descent, add samples to the dummy dataset and use 3 for batch size.
  - Report the loss.

x1	x2	y	$\hat{y}$	$loss$	y-$\hat{y}$	w_1	w_2	b
4	1	2	-0.116	4.48	2.116	-0.017	-0.048	0.000
2	8	-14						
1	0	1						
…								