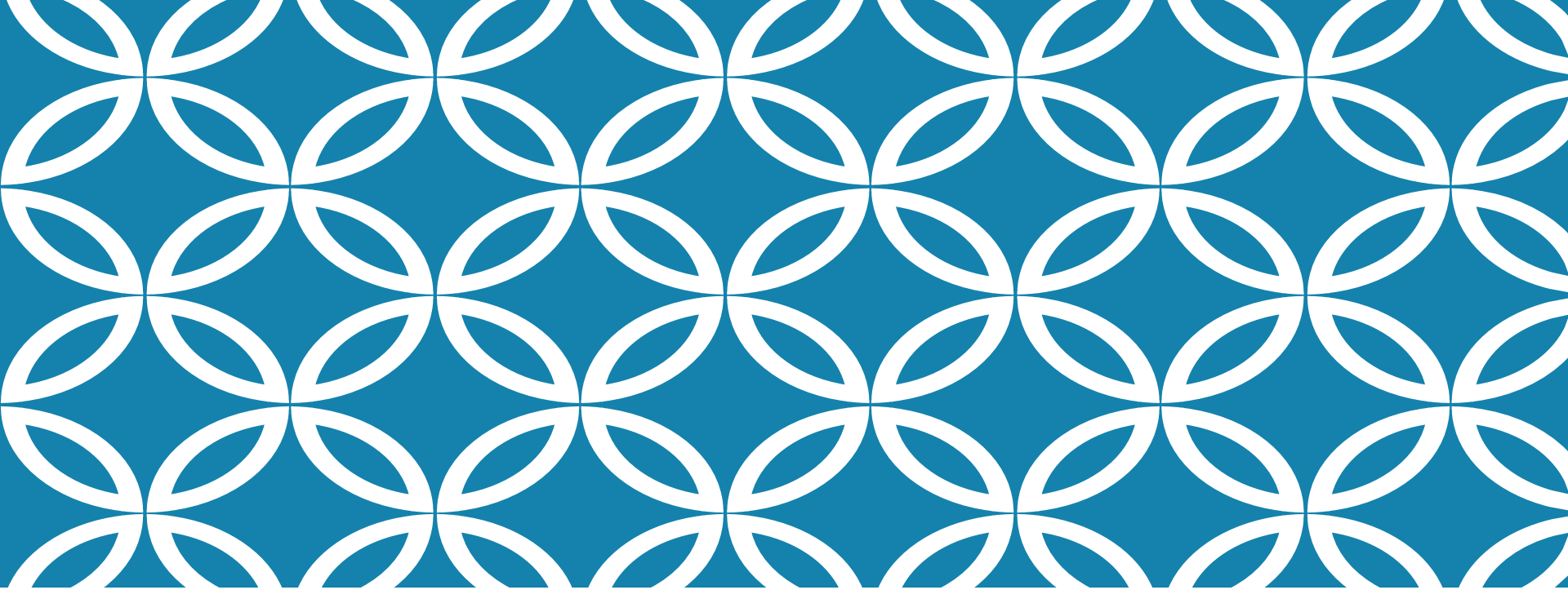
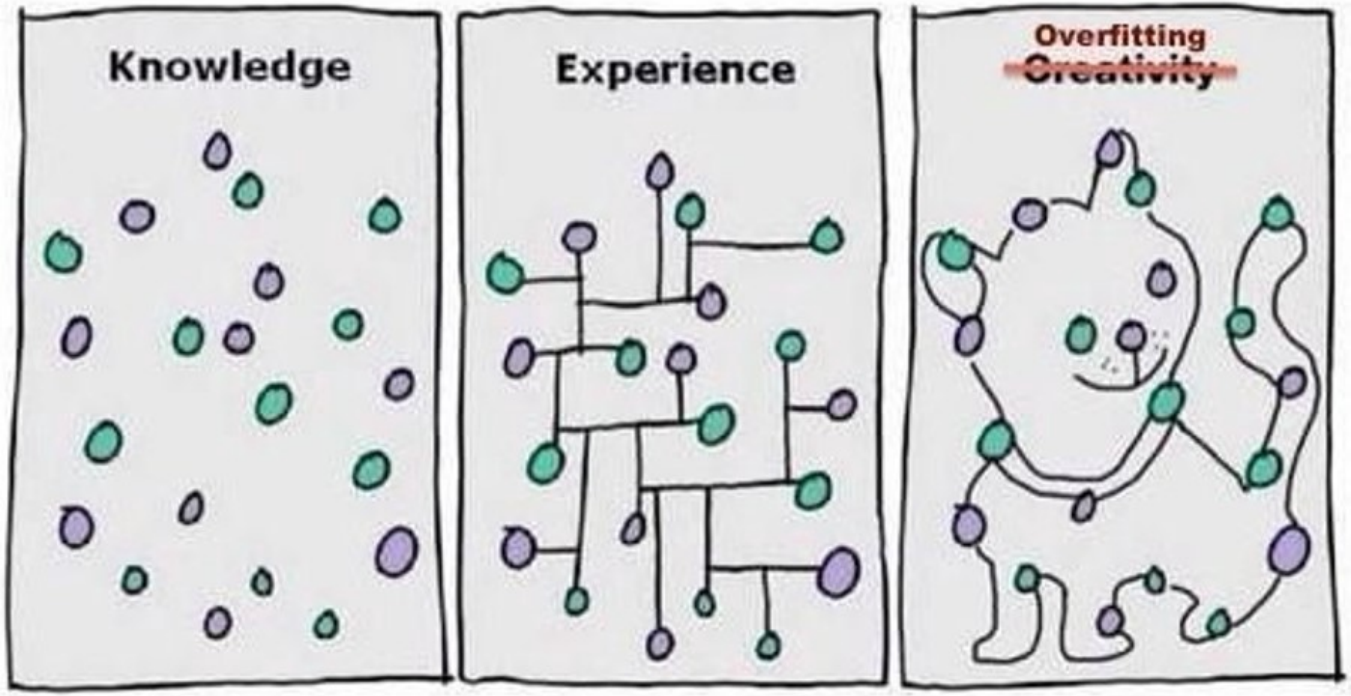


MACHINE LEARNING BASICS

Prashant Sahu


Knowledge
Experience
Overfitting
Creativity

LINEAR ALGORITHMS

- Gradient descent optimization procedure that may be used in the heart of many machine learning algorithms.
- Linear Regression for predicting real values with two tutorials to make sure it really sinks in.
- Logistic regression for classification on problems with two categories.
- Linear discriminant analysis for classification on problems with more than two categories.

NONLINEAR ALGORITHMS

These are techniques that make fewer assumptions about your problem and are able to learn a large variety of problem types. But this power needs to be used carefully because they can learn too well and overfit your training data.

You will discover the following nonlinear algorithms:

- **Classification and regression trees** the staple decision tree algorithm.
- **Naive Bayes** using probability for classification
- **K-Nearest Neighbors** that do not require any model at all other than your dataset.
- **Learning Vector Quantization** which extends K-Nearest Neighbors by learning to compress your training dataset down in size.
- **Support vector machines** which are perhaps one of the most popular and powerful out of the box algorithms.

PARAMETRIC AND NONPARAMETRIC MACHINE LEARNING ALGORITHMS

- Parametric machine learning algorithms simply the mapping to a know functional form.
- Nonparametric algorithms can learn any mapping from inputs to outputs.
- All algorithms can be organized into parametric or nonparametric groups.

PARAMETRIC MACHINE LEARNING ALGORITHMS

A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model.

No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.

The algorithms involve two steps:
1. Select a form for the function.
2. Learn the coefficients for the function from the training data.

PARAMETRIC MACHINE LEARNING ALGORITHMS

Some more examples of parametric machine learning algorithms include:

- Logistic Regression
- Linear Discriminant Analysis
- Perceptron

PARAMETRIC MACHINE LEARNING ALGORITHMS

Benefits of Parametric Machine Learning Algorithms:

- **Simpler**
- **Speed**
- **Less Data**

Limitations of Parametric Machine Learning Algorithms:

- ✓**Constrained**
- ✓**Limited Complexity**
- ✓**Poor Fit**

NONPARAMETRIC MACHINE LEARNING ALGORITHMS

Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features.

Some more examples of popular nonparametric machine learning algorithms are:

- Decision Trees like CART and C4.5
- Naive Bayes
- Support Vector Machines
- Neural Networks

NONPARAMETRIC MACHINE LEARNING ALGORITHMS

Benefits of Nonparametric Machine Learning Algorithms:

- Flexibility
- Power
- Performance

Limitations of Nonparametric Machine Learning Algorithms:

- More data
- Slower
- Overfitting

BIAS – VARIANCE TRADE-OFF

- **Bias** is the simplifying assumptions made by the model to make the target function easier to approximate.
- Hence Bias is the **error** from erroneous assumptions in the **learning** algorithm.
- **Variance** is the amount that the estimate of the target function will change given different training data.
- **Trade-off** is tension between the error introduced by the bias and the variance.

BIAS – VARIANCE TRADE-OFF

The prediction error for any machine learning algorithm can be broken down into three parts:

- Bias Error
- Variance Error
- Irreducible Error

- The irreducible error cannot be reduced regardless of what algorithm is used.
- It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

BIAS ERROR

Bias are the simplifying assumptions made by a model to make the target function easier to learn.

- **Low Bias:** Suggests **less** assumptions about the form of the target function.
- **High-Bias:** Suggests **more** assumptions about the form of the target function.
- High Bias can cause an algorithm to miss the relevant relations between features and target outputs **(underfitting).**

Generally **parametric algorithms** have a high bias making them fast to learn and easier to understand but generally less flexible.

In turn they are have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

BIAS ERROR

Examples of **low-bias** machine learning algorithms include:

- Decision Trees,
- k-Nearest Neighbors
- Support Vector Machines.

Examples of **high-bias** machine learning algorithms include:

- Linear Regression
- Linear Discriminant Analysis and
- Logistic Regression.

VARIANCE ERROR

Variance is the amount that the estimate of the target function will change if different training data was used.

Ideally, the target function should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.

- **Low Variance:** Suggests small changes to the estimate of the target function with changes to the training dataset.
- **High Variance:** Suggests large changes to the estimate of the target function with changes to the training dataset.
- High Variance can cause **overfitting**; modelling the random noise in the training data, rather the the intended outputs.

VARIANCE ERROR

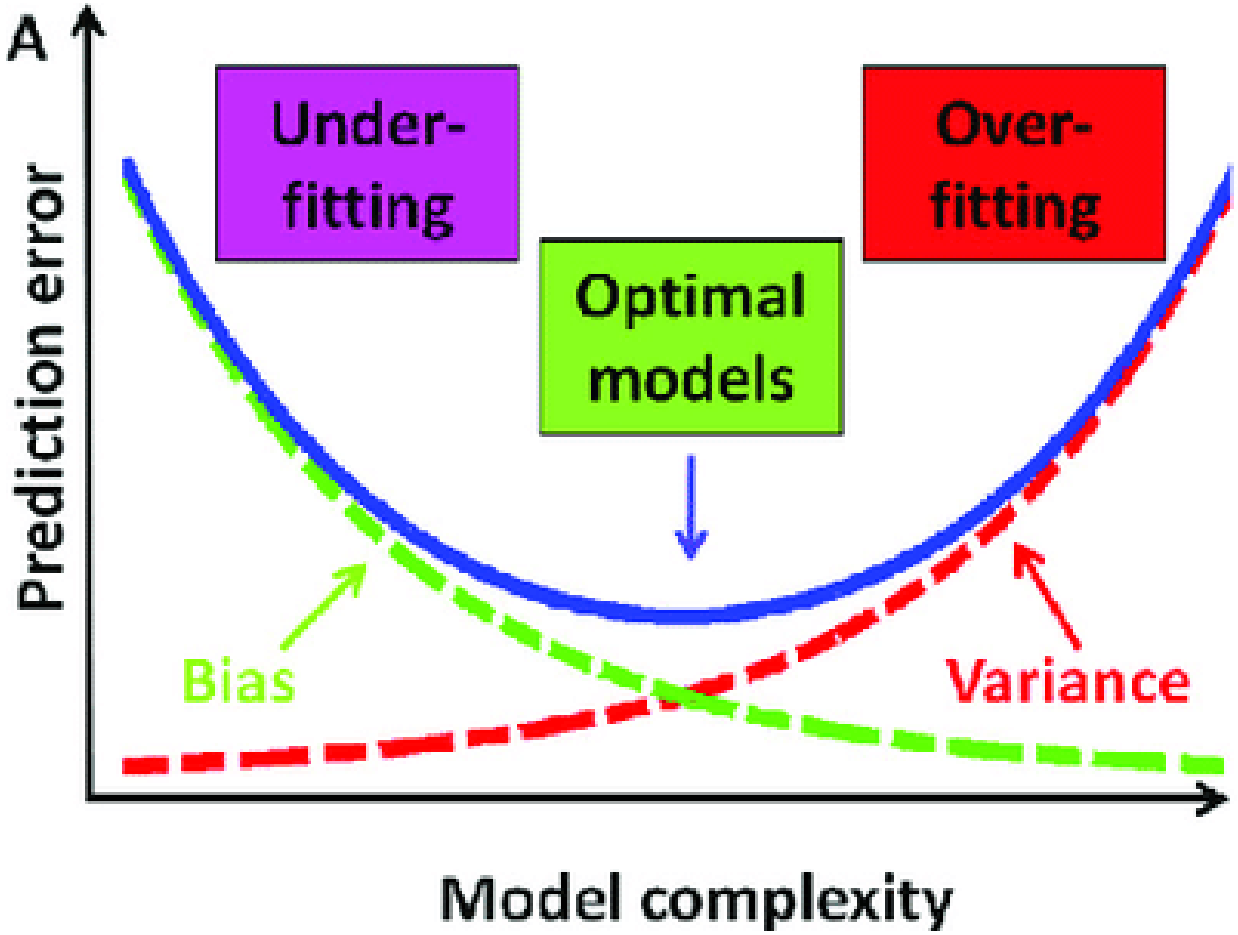
Generally nonparametric machine learning algorithms that have a lot of flexibility have a high variance. For example decision trees have a high variance, that is even higher if the trees are not pruned before use.

Examples of **low-variance machine learning** algorithms include:

- Linear Regression,
- Linear Discriminant Analysis and
- Logistic Regression.

Examples of **high-variance machine learning** algorithms include:

- Decision Trees,
- k-Nearest Neighbors and
- Support Vector Machines.


A
Prediction error
Under-fitting
Optimal models
Over-fitting
Bias
Variance
Model complexity

BIAS-VARIANCE TRADE-OFF

The goal of any supervised machine learning algorithm is to achieve low bias and low variance to achieve good prediction performance.

Parametric or linear machine learning algorithms often have a high bias but a low variance.

Nonparametric or nonlinear machine learning algorithms often have a low bias but a high variance.

The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

BIAS-VARIANCE TRADE-OFF

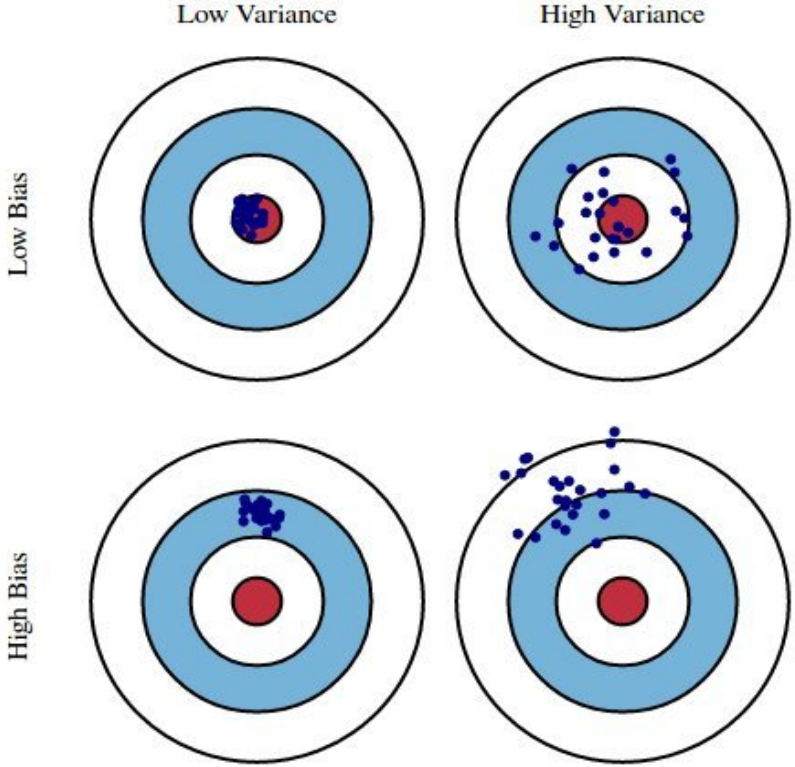
There is no escaping the relationship between bias and variance in machine learning.

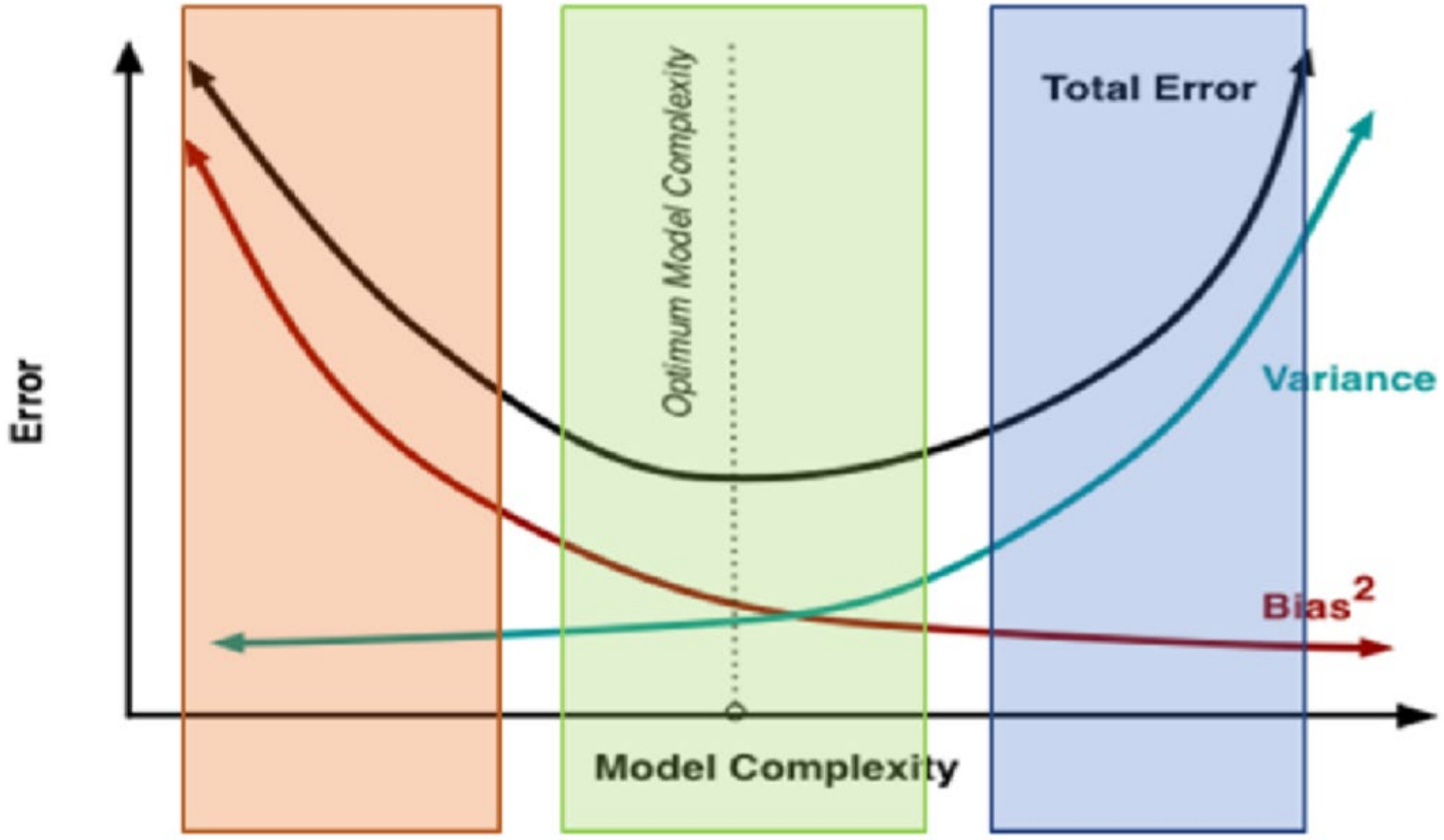
- **Increasing the bias will decrease the variance.**
- **Increasing the variance will decrease the bias.**

There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem.

In reality we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function.

BIAS – VARIANCE TRADE-OFF



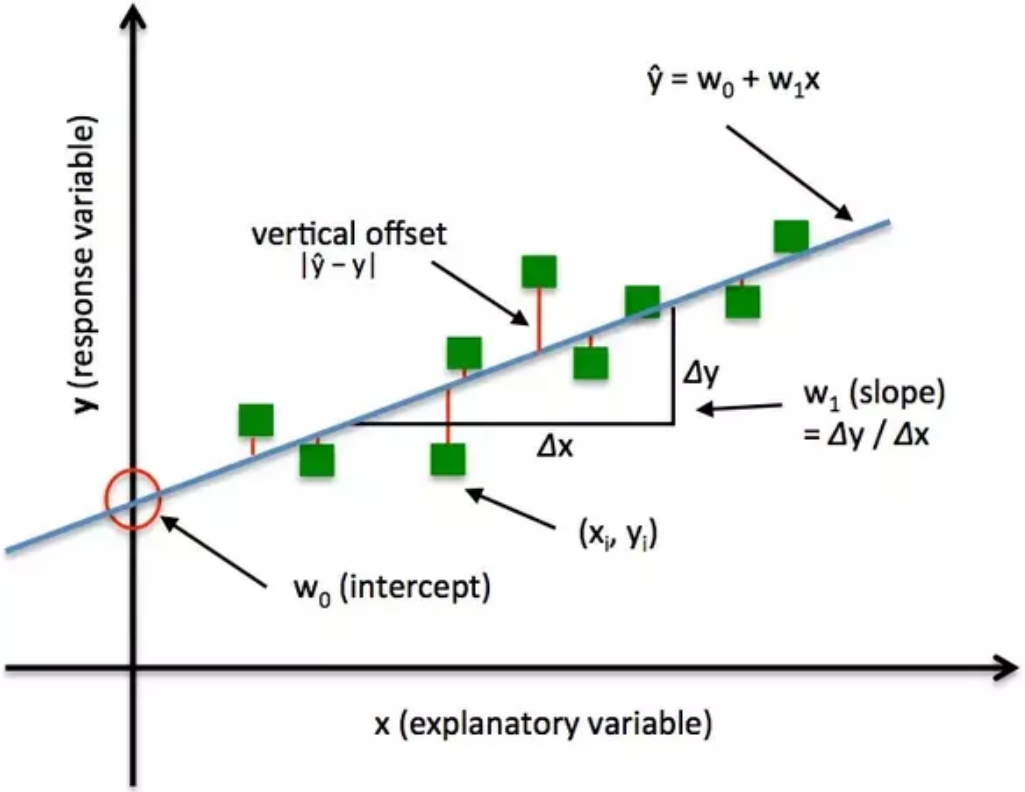


Region of High Bias & Low Variance Models (E.g. Logistic Regression)

Region of Ideal Model with both Low Bias & Low Variance

Region of Low Bias & High Variance Models (E.g. Decision Tree)

LINEAR REGRESSION



Assumptions of linear regression

Linear regression has the following **assumptions,** failing which the linear regression model does not hold true:

1. The dependent variable should be a linear combination of independent variables
2. No autocorrelation in error terms
3. Errors should have zero mean and be normally distributed
4. No or little multi-collinearity
5. Error terms should be homoscedastic

Optimization in Machine Learning

Types of Optimization Problems, in general:

(A) Unconstrained Optimization

$$\min f(x), \text{ where } f(x) = 2x^2 - 3x + 1$$

$$\underset{x}{\operatorname{argmin}} f(x), \text{ where } f(x) = 2x^2 - 3x + 1$$

(B) Constrained Optimization:

a. Linear or non-linear constraints

b. Equality or inequality constraints

$$\min (x_1 - 2)^2 + (x_2 - 1)^2 \qquad \text{subject to} \begin{cases} x_1^2 - x_2 & \leq 0, \\ x_1 + x_2 & \leq 2. \end{cases}$$

Possible Loss (Cost) functions in ML (Regression)

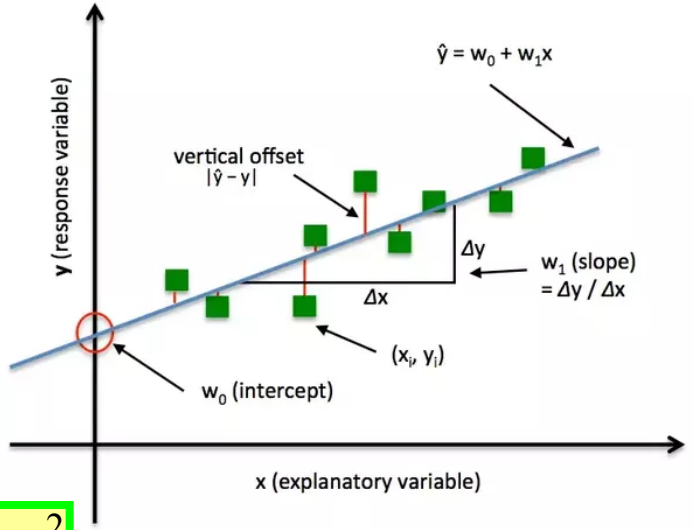
(1) Sum of errors (SE): $L = \sum_{i=1}^{N} \left(\hat{Y}_i - Y_i \right)$

(2) Sum of Absolute Errors (SAE): $L = \sum_{i=1}^{N} \left| \hat{Y}_i - Y_i \right|$

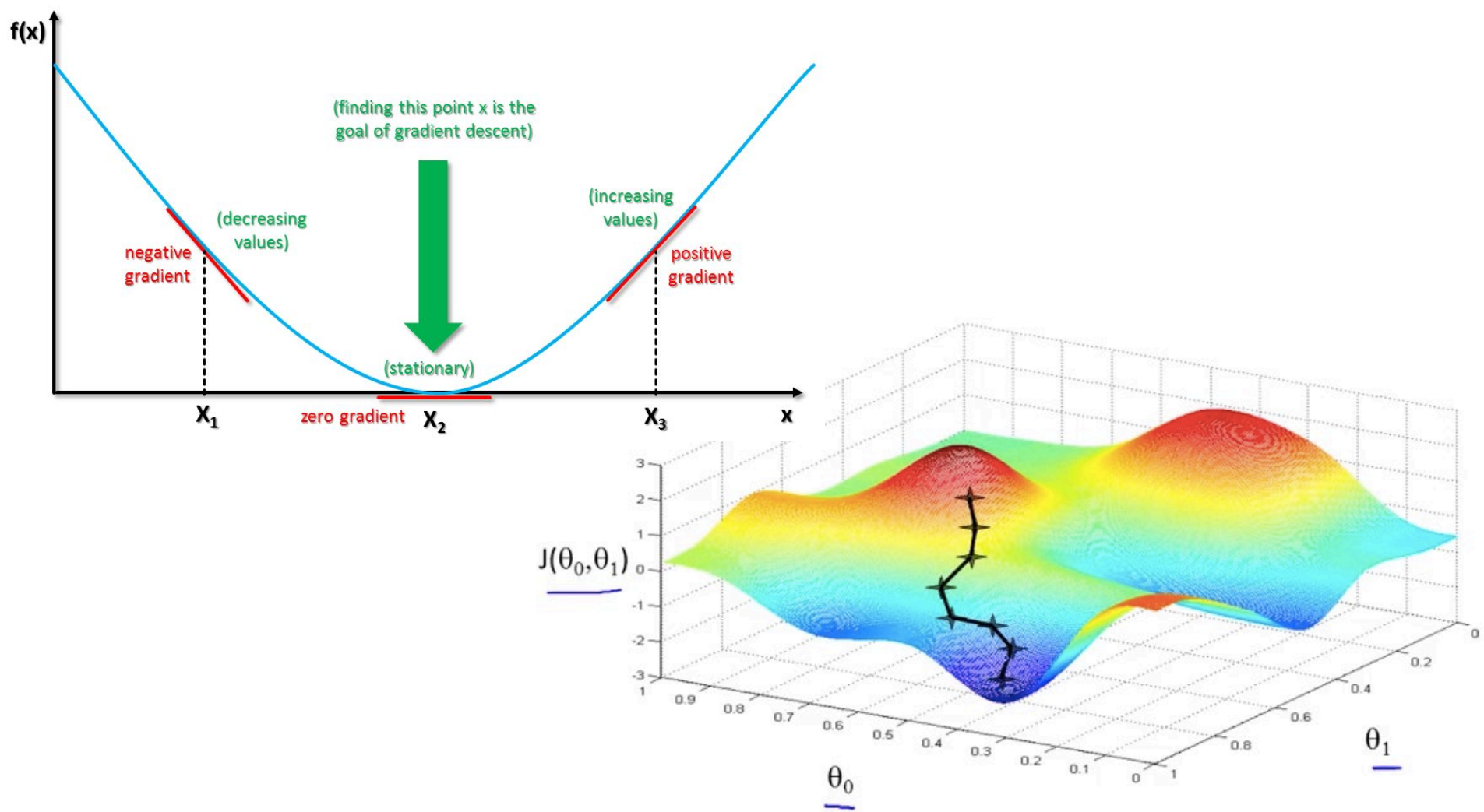
(3) Sum of Squares of Errors (SSE): $L = \sum_{i=1}^{N} \left(\hat{Y}_i - Y_i \right)^2$

(4) Mean of Squares of Errors (MSE): $L = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{Y}_i - Y_i \right)^2$

(5) Root Mean of Squares of Errors (RMSE): $L = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\hat{Y}_i - Y_i \right)^2}$



X1	Y
1	4.8
3	11.4
5	17.5
..	…


f(x)
(finding this point x is the goal of gradient descent)
(decreasing values)
(increasing values)
negative gradient
positive gradient
(stationary)
zero gradient
X1
X2
X3
x
J(θ0,θ1)
θ0
θ1

Linear Regression Problem Formulation:

$$Objective\ Fn: MSE:\ L = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(error)^2$$

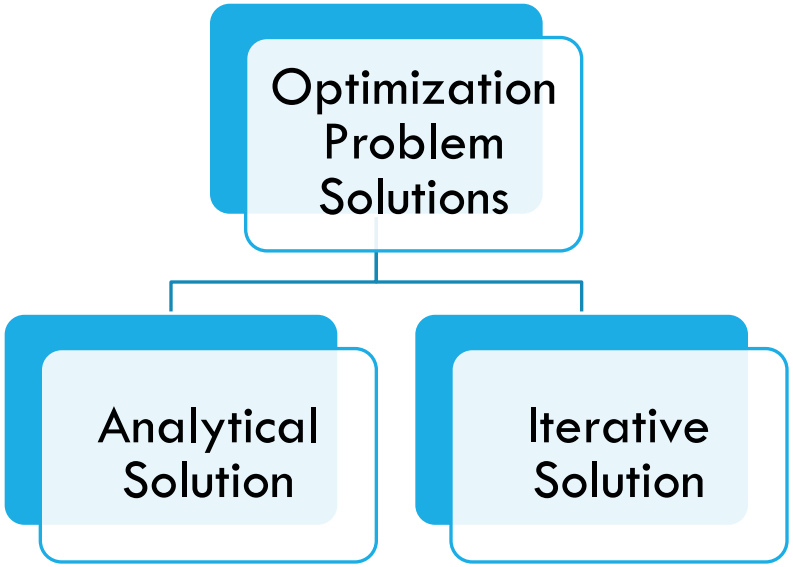
$$\arg\min_{w_j} L(w_j \mid X, Y) \qquad where,\ \ \hat{Y} = w_0 + w_1 X_1$$

X1	Y
1	4.8
3	11.4
5	17.5
..	…

Says, Find the **optimal weights (w_j)** for which the **MSE**

Loss function has **min value**, for a GIVEN X,Y data.

Optimization Problem Solution Methods:



- Theoretical Solution, which gives the "exact" solution to the problem, provided the optim. problem has a "closed-form solution

-Approximate Solution to the optim. problem, based on some iterative algorithm

- Can solve all types of optim. problems.

Analytical Solution for Unconstrained Optimization:

FOC: Necessary Condition: Sates that the first (odd) derivative (gradient) of the objective function must vanish at the optimal points (points of maxima/minima)

$$e.g.\ f(x) = 2x^2 - 3x + 1 \qquad \frac{df}{dx} = 4x - 3 = 0 \qquad \Rightarrow x^* = 0.75$$

SOC: Sufficiency Condition: Sates that the second (even) derivative (gradient) of the objective function **evaluated at the optimal points**, must be:

- **Positive >> for minima**
- Negative >> for maxima

$$\left.\frac{d^2 f}{dx^2}\right|_{x^*=0.75} = 4 \ \ (positive)$$

which means minima occurs at $x^* = 0.75$

Linear Regression – Analytical Solution:

$$MSE: L = \frac{1}{N}\sum\left(\hat{Y} - Y\right)^2 \qquad where, \quad \hat{Y} = w_0 + w_1 X_1$$

X1	Y
1	4.8
3	11.4
5	17.5

$$\arg\min_{w_j} L(w_j \mid X, Y)$$

Find the **optimal weights (wj)** for which the MSE Loss function has min value, for GIVEN X,Y data.

STEP – 1: Get the Gradients:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{Y}} \times \frac{\partial \hat{Y}}{\partial w_j}$$

$$\frac{\partial L}{\partial w_j} = \frac{2}{N}\sum\left(\hat{Y} - Y\right) \times \frac{\partial \hat{Y}}{\partial w_j}$$

FINAL GRADIENTS

$$\frac{\partial L}{\partial w_0} = \frac{2}{N}\sum\left(\hat{Y} - Y\right) \times 1$$

$$\frac{\partial L}{\partial w_1} = \frac{2}{N}\sum\left(\hat{Y} - Y\right) \times X_1$$

Linear Regression – Analytical Solution:

STEP – 2: Equate the Gradients to zero and solve:

$$\frac{2}{N}\sum\left(\hat{Y}-Y\right)\times 1 \;=0 \qquad --\ (1)$$

$$\frac{2}{N}\sum\left(\hat{Y}-Y\right)\times X_1 \;=0 \qquad --\ (2)$$

FINAL SOLUTION ⟸ **Ordinary Least Squares (OLS) !**

$$w_1^* = \frac{\left[\overline{XY}-\overline{X}\,\overline{Y}\right]}{\left[\left(\overline{X^2}\right)-\left(\overline{X}\right)^2\right]}$$

$$\hat{Y} = w_0 + w_1 X_1$$

$$\overline{Y} = w_0 + w_1 \overline{X_1}$$

Intercept / Bias Term:

$$w_0^* = \overline{Y} - \left(w_1^* \cdot \overline{X}\right)$$

Loss (Cost) Function in Vector Form:

w0 w1 ..... w13

1 X1 ..... X13 Y (TARGET)

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
1	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
1	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
1	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
1	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

X = N x 14

$$\hat{Y} = w_0 \cdot 1 + w_1 X_1 + w_2 X_2 + \ldots\ldots + w_{13} X_{13}$$

$$W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \ldots\ldots \\ w_{13} \end{bmatrix}_{14\times 1}$$

Y = N x 1

W = 14 x 1

$$\hat{Y}_{[N\times 1]} = X_{[N\times 14]} W_{[14\times 1]}$$

OLS in Vector Form:

$$L = \frac{1}{N}\sum\left(\hat{Y}_i - Y_i\right)^2 = \frac{1}{N}\left[\left(\hat{Y} - Y\right)^T\left(\hat{Y} - Y\right)\right]$$

$$L = \frac{1}{N}\left[(XW)^T(XW) - 2(XW)^T Y + Y^T Y\right]$$

Loss Function in Vector Notation

$$\frac{\partial L}{\partial W} = \frac{1}{N}\left[(X^T X)(2W) - 2(X^T Y) + 0\right] = 0$$

$$2(XW)^T Y = 2(X^T Y)W^T$$

$$(X^T X)W = X^T Y$$

$$Ax = B$$

$$A^{-1}(Ax) = A^{-1}B$$

$$Ix = A^{-1}B \Rightarrow x = A^{-1}B$$

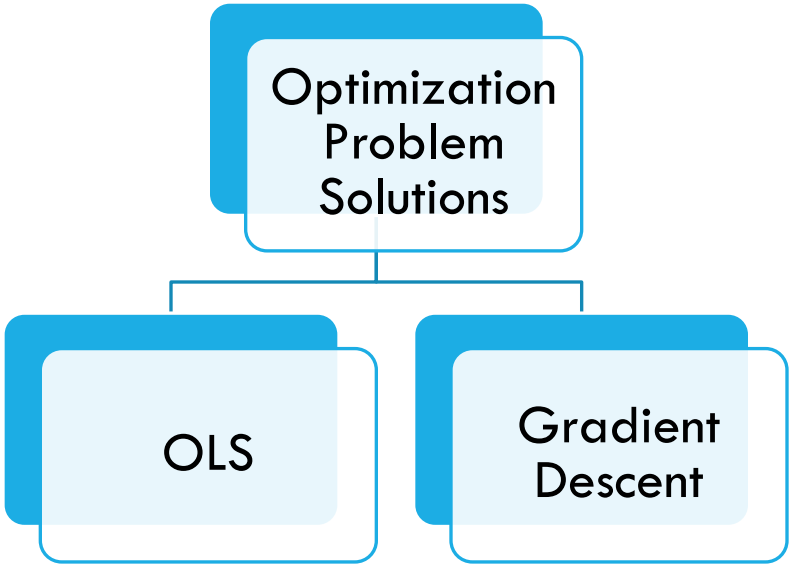
$$W = (X^T X)^{-1} X^T Y$$

OLS Solution (Analytical Solution) !

$$W = (X^T_{[14\times N]} X_{[N\times 14]})^{-1}_{[14\times 14]} X^T_{[14\times N]} Y_{[N\times 1]} = [14\times 1]$$

$$\frac{\partial^2 L}{\partial W^2} = \frac{1}{N}\left[(X^T X)(2)\right] = +ve \text{ (minima)}$$

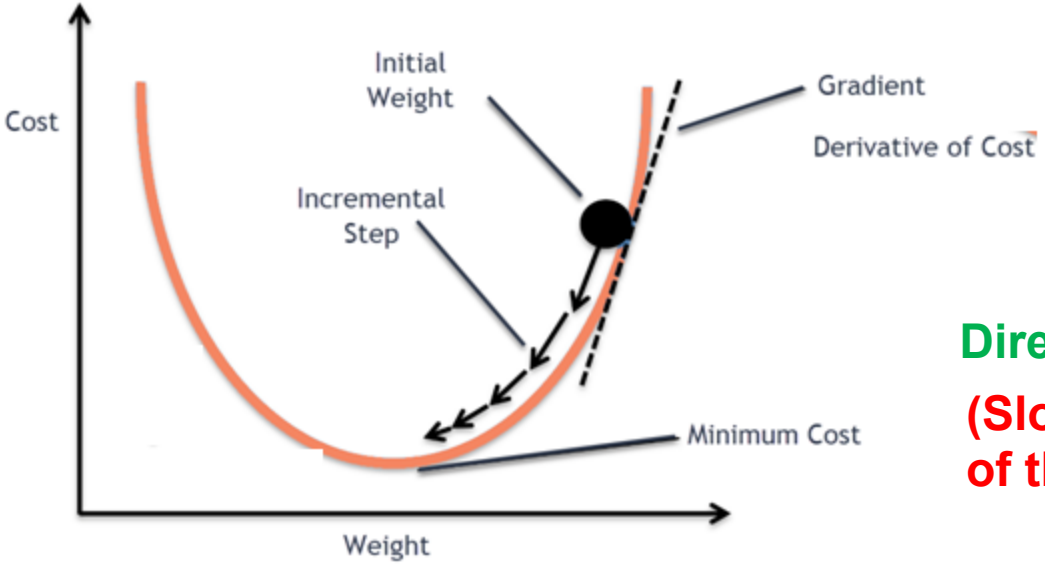
Optimization Problem Solution Methods:



- Analytical Solution, which gives the "exact" solution to the problem, provided the optim. problem has a "closed-form solution

-Approximate Solution (iterative) to the optim. problem, based on some iterative algorithm
- Can solve all types of optim. problems.

Gradient Descent Algorithm:



Gradient Descent Update Rule:

$$w_j^{k+1} = w_j^k - \Delta w_j$$

Direction of Update

(Slope / Gradient of the Loss Fn)

$$\frac{\partial L}{\partial w_j}$$

Amount of Update

Step Size: α or Learning Rate

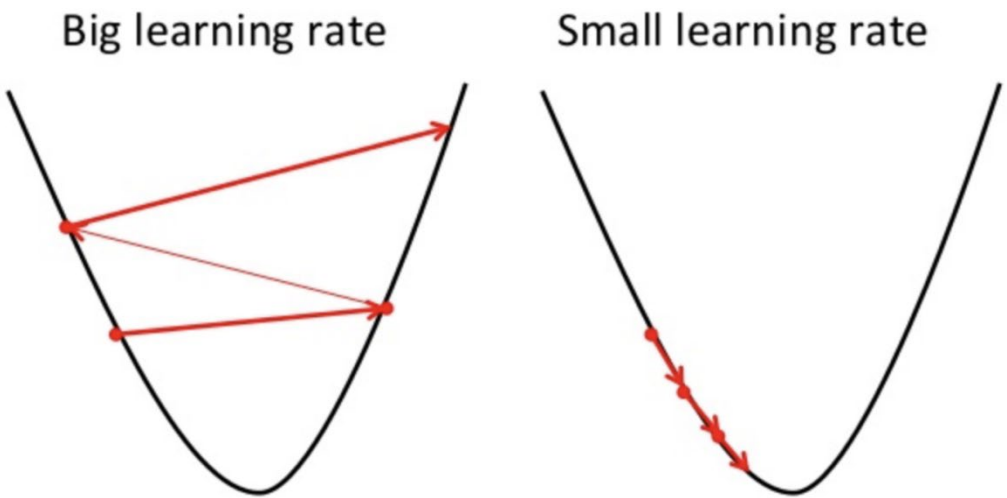
Final Gradient Descent Update Rule:

$$w_j^{k+1} = w_j^k - \left(\alpha \frac{\partial L}{\partial w_j} \right)$$

$$\frac{\partial L}{\partial w_0} = \sum \left(\hat{Y} - Y \right) \times 1$$

$$\frac{\partial L}{\partial w_1} = \sum \left(\hat{Y} - Y \right) \times X_1$$

Effect of Learning Rate:



Alpha is a **Hyper-parameter** that **YOU** have to decide (based on your exp & domain knowledge)… trail & error. HP are to be specified/decided before the start of the iterations start.

Model coefficients/weights (W) >> **Model Parameters >> these are "learnt" by the algo from the DATA.** You don't specify this.

Gradient Descent Steps:

1. Initialize the algo with random values of alpha, and weights (w0, w1)
2. Cal predictions Y^ = w0*1 + w1X1 + …
3. Cal Error terms & L:
   (Y^ – Y), (Y^ – Y)X1, Loss function
4. **Update your weights:**
   w0_1 = w0 – (alpha * (Y^ – Y))
   w1_1 = w1 – (alpha * (Y^ – Y)X1)
5. Repeat 2-4, until convergence..

X	Y	Y^	Y^-Y	(Y^-Y)*X
1	4.8	16.02	11.22	11.22 x1
3	11.4	41.32	29.92	29.92 x 3
5	17.5	66.62	49.12	49.12x 5
		Sum of values	90.26	346.58

Alpha = 0.1, w0, w1 = 0
W0_1 = W0_0 – [alpha*error]
W0_1 = 0 – [0.1*(-33.7)] = +3.37

W1_1 = W1_0 – [alpha*error]
W1_1 = 0 – [0.1*(-126.5)] = +12.65

Alpha = 0.1, w0 = +3.37, w1 = +12.65
W0_2 = W0_1 – [alpha*error]
W0_2 = 3.37 – [0.1*(90.26)] = -5.65

W1_2 = W1_1 – [alpha*error] =
W1_2 = +12.65 – [0.1*(346.58)] = -22.008

Gradient Descent Variants:

- Batch GD (Vanilla GD)
- **Stochastic GD (SGD)**
- Mini Batch GD

X	Y	Y^	Y^-Y	(Y^-Y)*X
1	4.8	0	-4.8	-4.8 x 1
3	11.4	1.92	-9.48	-9.48*3 = -28.44
5	17.5	18.048	0.548	2.74

Alpha = 0.1, w0, w1 = 0,0

W0_1 = 0 – (0.1*(-4.8)) = 0.48
W1_1 = 0 – (0.1*(-4.8*1)) = 0.48

Y2 = 0.48 + (0.48*3) = 1.92
W0_2 = 0.48 – (0.1*(-9.48)) = 1.428
W1_2 = 0.48 – (0.1*(-28.44)) = 3.324

W0_3 = 1.428 – (0.1*(0.548)) = 1.3732
W1_3 = 3.324 – (0.1*(2.74)) = 3.05

1 Epoch = one “pass” of the entire dataset

1	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
2	79545.5	5.68	7.01	4.09	23086.8	1059034	208
3	61287.1	5.87	8.51	5.13	36882.16	1058988	9127
4	63345.2	7.19	5.59	3.26	34310.24	1260617	USS
5	59982.2	5.04	7.84	4.23	26354.11	630943	USNS
6	80175.8	4.99	6.1	4.04	26748.43	1068138	06039
7	64698.5	6.03	8.15	3.41	60828.25	1502056	4759
8	78394.3	6.99	6.62	2.42	36516.36	1573937	972
9	59927.7	5.36	6.39	2.3	29387.4	798870	USS
10	81885.9	4.42	8.17	6.1	40149.97	1545155	Unit
11	80527.5	8.09	5.04	4.1	47224.36	1707046	6368
12	50593.7	4.5	7.47	4.49	34343.99	663732	911
13	39033.8	7.67	7.25	3.1	39220.36	1042814	209
14	73163.7	6.92	5.99	2.27	32326.12	1291332	829
15	69391.4	5.34	8.41	4.37	35521.29	1402818	PSC
16	73091.9	5.44	8.52	4.01	23929.52	1306675	2278
17	79707	5.07	8.22	3.12	39717.81	1556787	064
18	61929.1	4.79	5.1	4.3	24595.9	528485	5498
19	63508.2	5.95	7.19	5.12	35719.65	1019426	Unit
20	62085.3	5.74	7.09	5.49	44922.11	1030591	19696
21	86295	6.63	8.01	4.07	47560.78	2146925	030
22	60835.1	5.55	6.52	2.1	45574.74	929248	USNS
23	64490.7	4.21	5.48	4.31	40358.96	718887	95198
24	60697.4	6.17	7.15	6.34	28140.97	744000	9003 Jay
25	59748.9	5.34	7.75	4.23	27809.99	895737	24282
26	56974.5	8.29	7.31	4.33	40694.87	1453975	61938

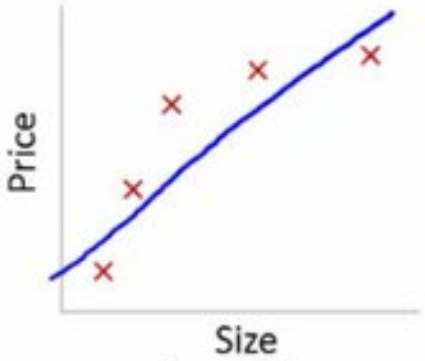
$$\frac{\partial L}{\partial w_0} = \sum \left(\hat{Y} - Y \right) \times 1$$

$$\frac{\partial L}{\partial w_1} = \sum \left(\hat{Y} - Y \right) \times X$$

Steps for GD:

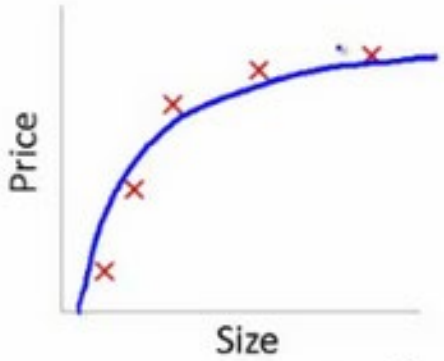
1. Intialise the algo with random values of alpha, and weights
2. Cal Y_pred = w0 + w1X1 + …
3. Cal Error terms (Y_pred – Y), (Y_pred – Y)X1
4. Update your weights:
   w0_1 = w0 – (alpha* (Y_pred – Y))
   w1_1 = w0 – (alpha* (Y_pred – Y)X1)
5. Repeat 2-4, until convergence..

OVERFITTING (REGRESSION)



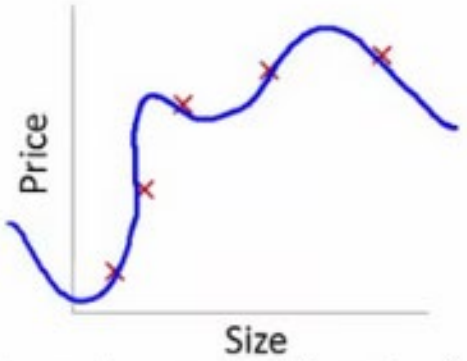
$\theta_0 + \theta_1 x$

High bias
(underfit)



$\theta_0 + \theta_1 x + \theta_2 x^2$

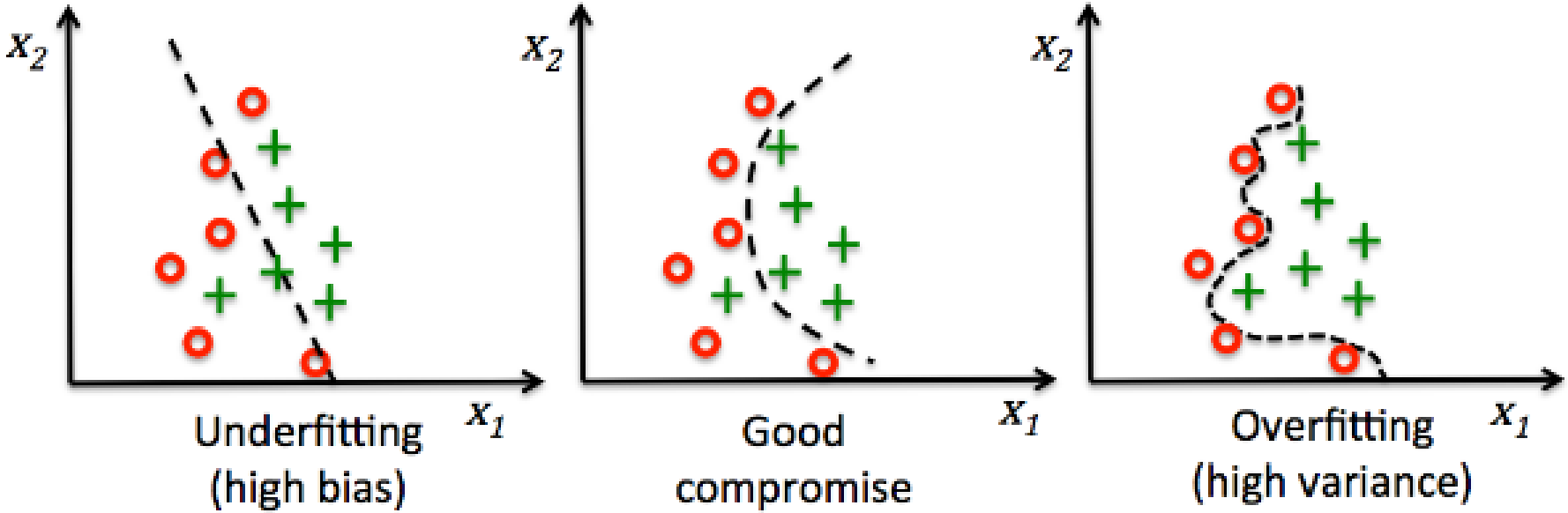
"Just right"



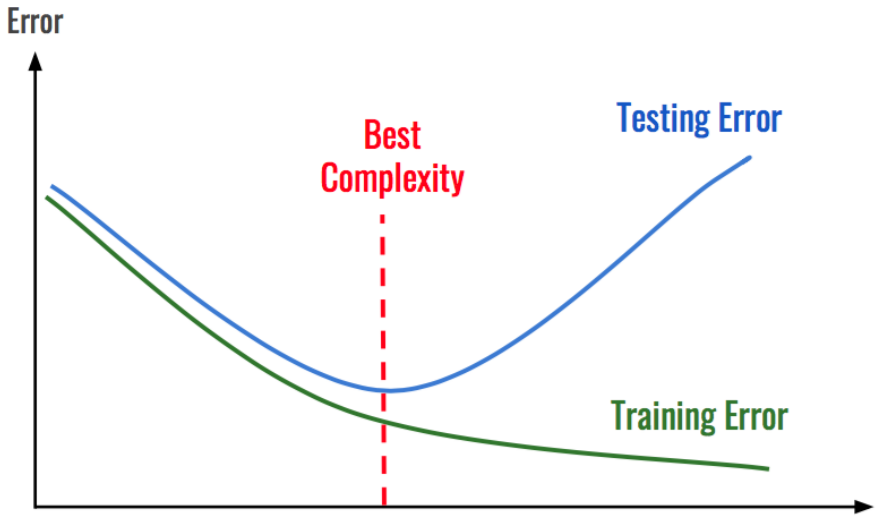
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

OVERFITTING (CLASSIFICATION)



Back to Overfitting



- What is overfitting?
- How do I know my algo is overfitting??
- **How do I check (stop/prevent) my algo from overfitting??**

1. Early Stopping
2. Regularisation
3. Pruning (for Tree-based algo) >> Decision Trees, Random Forests, etc..
4. Dimensionality reduction
5. Feature Selection
6. Ensembling (Bagging algorithms)

Concept of Regularization

Regularizations techniques are used to reduce the error by fitting a function appropriately on the given training set to avoid overfitting. Let's build an intuition with the help of an example

- Let's say a parent is very cautious about the future of his children
- He wants them to be successful in life without being strict with them. He takes a decision about how much flexibility should be given to his children during their upbringing.
- *Too much restriction may suppress their development of character*

The overfitting behaviour occurs when basis functions overlap:

- The coefficients of adjacent basis functions grow large and cancel each other out.
- We need to limit such spikes explicitly in the model by penalizing large values of the model parameters (the thetas of variables)
- Such a penalty is known as regularization.

It can be done in three ways -

- L1 Regularization (also called as Lasso Penalization/Regression)
- L2 Regularization (also called as Ridge Penalization/Regression)
- Elastic-net

Regularization in Machine Learning

$$MSE: \; L = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2$$

$$p-Norm\ (L_p) = \left\|w_j\right\|_p = \left(\sum\left|w_j\right|^p\right)^{1/p}$$

$$Ridge: \; L = \{MSE\} + \lambda\left\|w_j\right\|_2^2 = \left\{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2\right\} + \lambda(w_1^2 + w_2^2 + \ldots + w_p^2)$$

$$LASSO: \; L = \{MSE\} + \lambda\left\|w_j\right\|_1 = \left\{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2\right\} + \lambda\left(\left|w_1\right| + \left|w_2\right| + \ldots + \left|w_p\right|\right)$$

$$ElasticNet: \; L = \{MSE\} + \lambda_1\left\|w_j\right\|_1 + \lambda_2\left\|w_j\right\|_2^2$$

where $j = 1, 2 \ldots\ldots p$ number of features

L1 Regularization (also called as LASSO penalisation)

Involves penalising sum of absolute values (1-norms) of regression coefficients

$$LASSO: \ L = \left\{ \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2 \right\} + \lambda \left\| w_j \right\|_1 = \left\{ \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2 \right\} + \lambda \left(|w_0| + |w_1| + \ldots + |w_j| \right)$$

- Here we are familiar with the First half of the Cost Function.
- By adding all weights to the cost function, which we want to minimize, we're adding further restrictions on these parameters
- Typically intercepts are not penalised.
- The lambda parameter in Lasso tunes the strength of the penalty, and should be determined via cross-validation.

But what if ?

- Taking an example, our model has 100 coefficients but only 10 of them have non-zero coefficients, this is effectively saying that “ the other 90 predictors are useless in predicting the target values ”.
- Though this is conceptually very similar to ridge regression(we will see in later slides), the results can differ surprisingly. For example, due to geometric reasons **lasso regression tends to favor sparse models**
- A sparse matrix is a matrix with a **LOT OF 0's**
- That is, it preferentially sets model coefficients to exactly zero!

L2 Regularization (also called as Ridge Penalisation)

This proceeds by penalising the sum of squares (2-norms) of the model coefficients

$$Ridge:\ L = \left\{ \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2 \right\} + \lambda \left\| w_j \right\|_2^2 = \left\{ \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2 \right\} + \lambda (w_0^2 + w_1^2 + \ldots + w_j^2)$$

- The L2 regularization will force the parameters to be relatively small, the bigger the penalization, the smaller (and the more robust to overfitting) the coefficients are.

- Here we are considering every feature, but we are penalizing the coefficients based on how significant the feature is.

α is a hyper-parameter that controls the strength of the penalty.

The α parameter controls complexity of the resulting model.

- In the limit $\alpha \to 0$, we recover the standard linear regression result.
- In the limit $\alpha \to \infty$, all model responses will be suppressed.

L2 regularization	L1 regularization
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases
Non-sparse outputs	Sparse outputs
No feature selection	Built-in feature selection

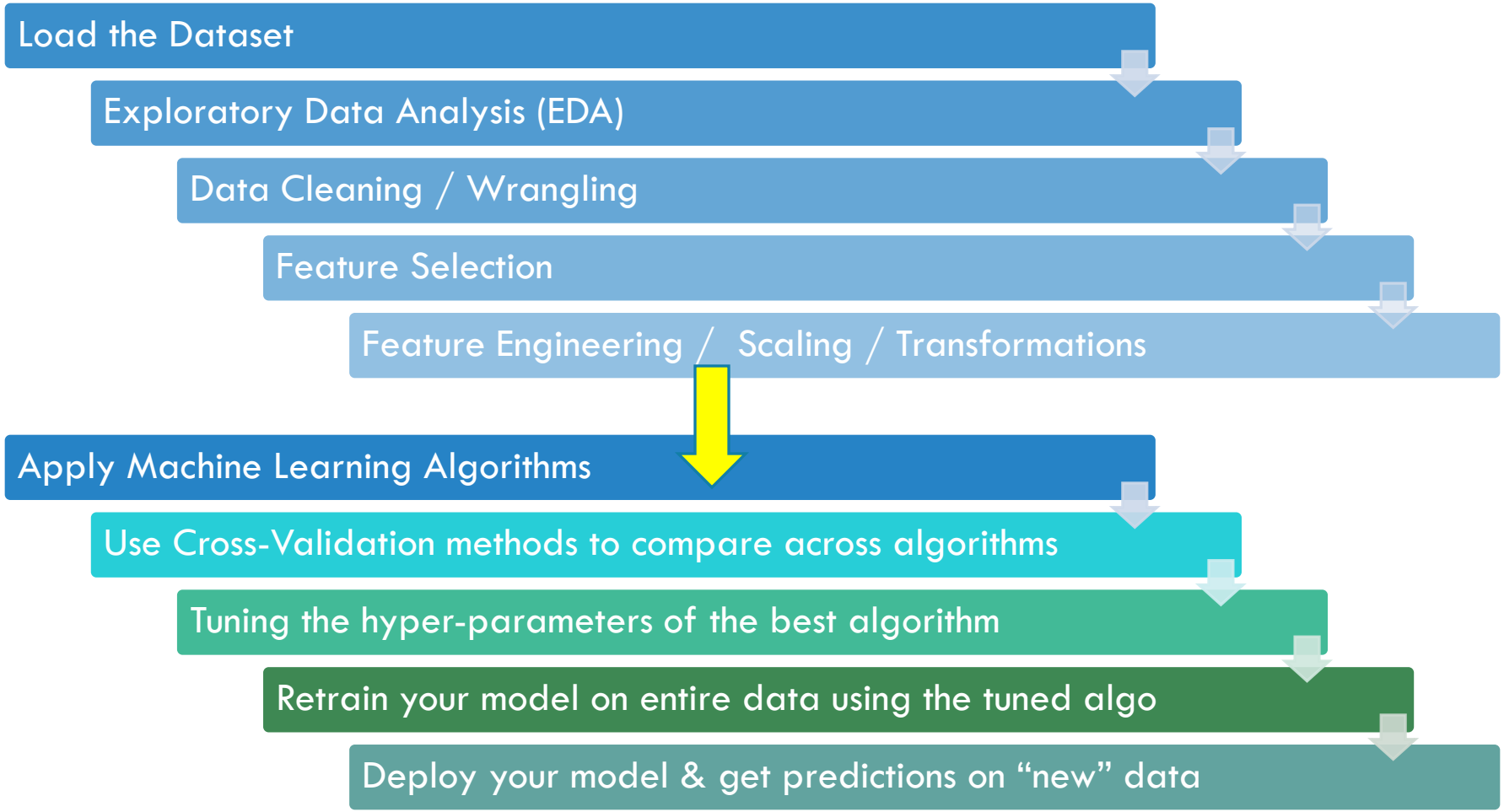
Let's say we have a large dataset which has 10,000 features.
And some of the independent features are correlated with other independent features.

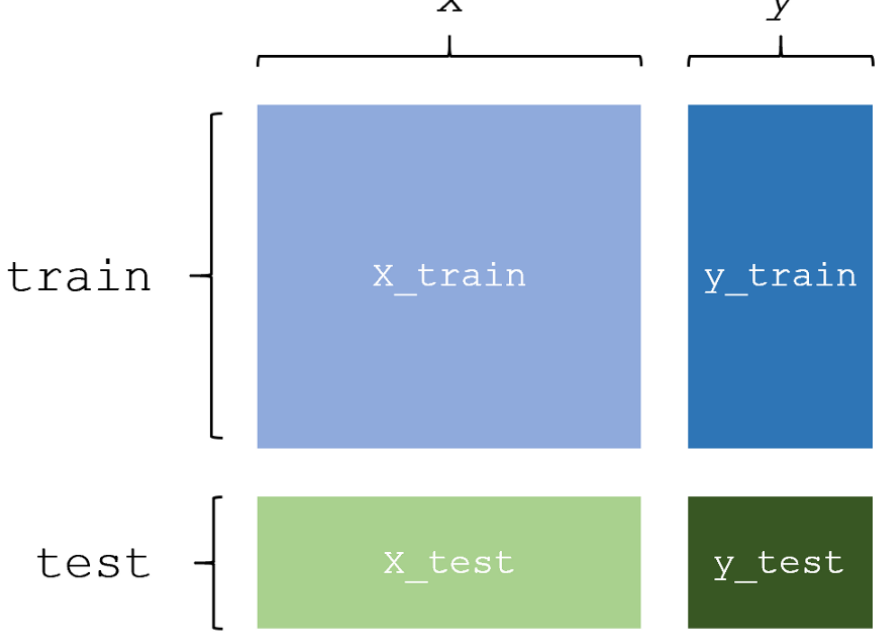
Which one would suit better, Rigde or Lasso?

- If we apply **ridge regression** to it, it will retain all of the features but will **shrink the coefficients**. Still the problem is that model will remain complex as there are 10,000 features, thus may lead to poor model performance.
- If we apply **lasso regression** to this problem, the main problem will be when we have correlated variables, it would retain only one variable and set other correlated variables to zero.
- That will possibly lead to some loss of information resulting in lower accuracy in our model.

TECHNIQUES FOR FEATURE SELECTION:

1. LASSO
2. SelectKBest(scoring_fn, k) >> uses Chi-square test
3. RFE(estimator, K)
4. ExtraTreeClassifier >> Feature Importances


Load the Dataset
Exploratory Data Analysis (EDA)
Data Cleaning / Wrangling
Feature Selection
Feature Engineering / Scaling / Transformations
Apply Machine Learning Algorithms
Use Cross-Validation methods to compare across algorithms
Tuning the hyper-parameters of the best algorithm
Retrain your model on entire data using the tuned algo
Deploy your model & get predictions on "new" data

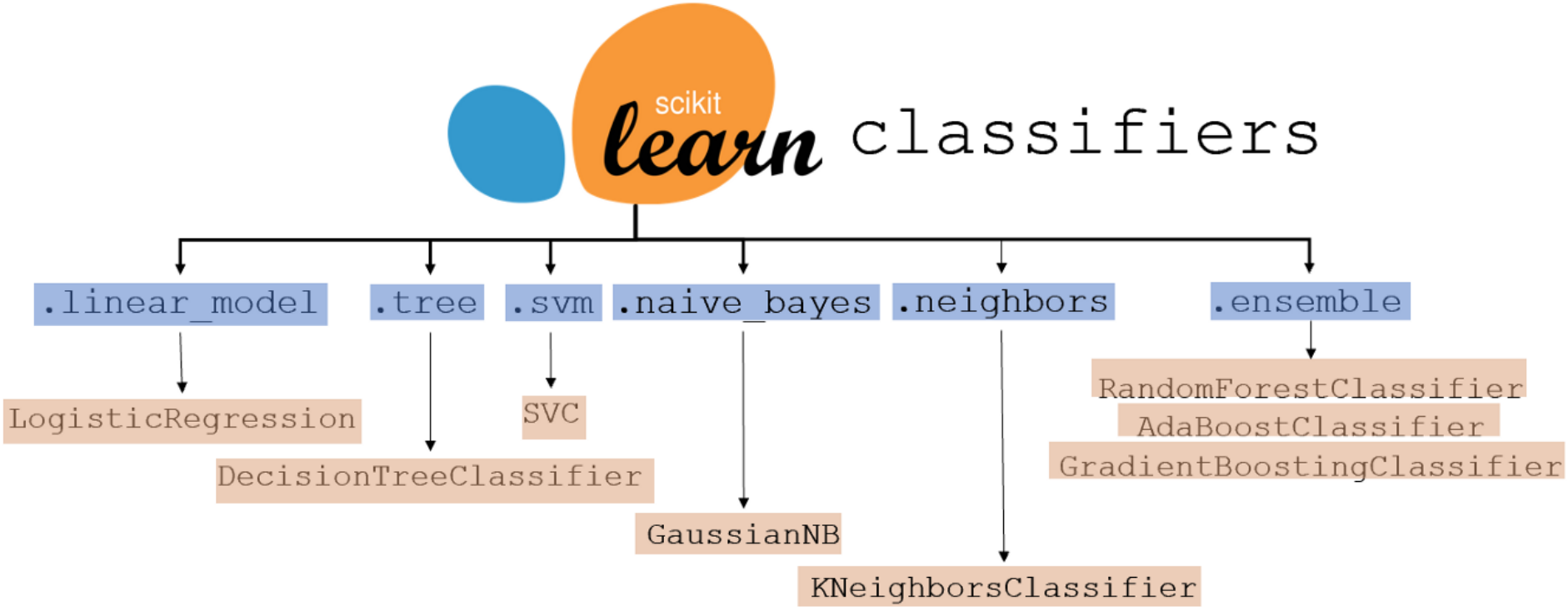


85% +/- 3%

1. TTS
2. ShuffleSplit(5) CV
3. K-Fold CV
4. LOOCV

Mean acc +/- std()
80% +/- 4%

Acc when the model was trained & tested 5 times !!!!!!!!!!
78%
79.5%
83.2%
81.5%
77.7%


scikit
learn
classifiers
.linear_model
.tree
.svm
.naive_bayes
.neighbors
.ensemble
LogisticRegression
DecisionTreeClassifier
SVC
GaussianNB
KNeighborsClassifier
RandomForestClassifier
AdaBoostClassifier
GradientBoostingClassifier

regressors

- .linear_model
  - LinearRegression
  - Lasso
  - LassoLars
  - Ridge
  - BayesianRidge
- .tree
  - DecisionTreeRegressor
- .svm
  - LinearSVR
- .neighbors
  - KNeighborsRegressor
- .ensemble
  - RandomForestRegressor
  - AdaBoostRegressor
  - GradientBoostingRegressor

metrics

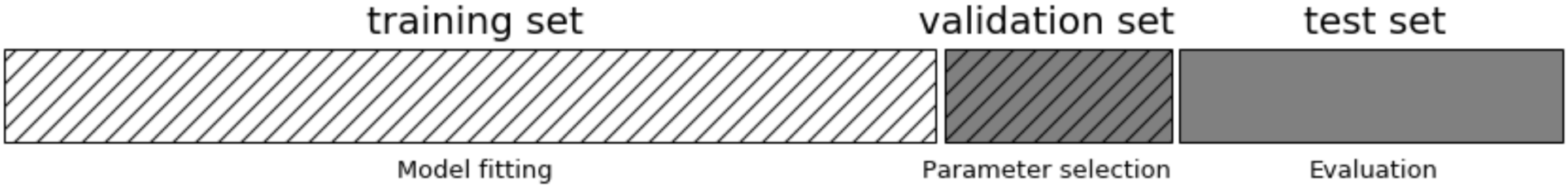
commonly used metrics marked in green

classification

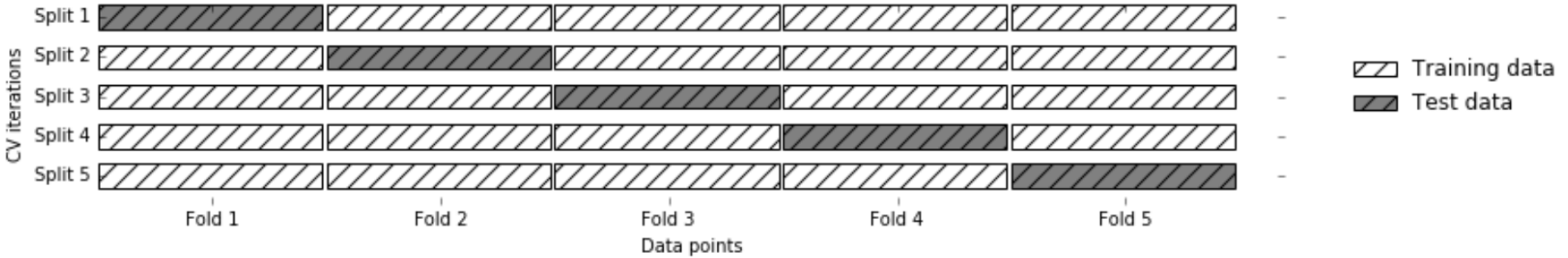
- accuracy_score
- balanced_accuracy_score
- average_precision_score
- brier_score_loss
- f1_score
- log_loss
- precision_score
- recall_score
- jaccard_score
- roc_auc_score

regression

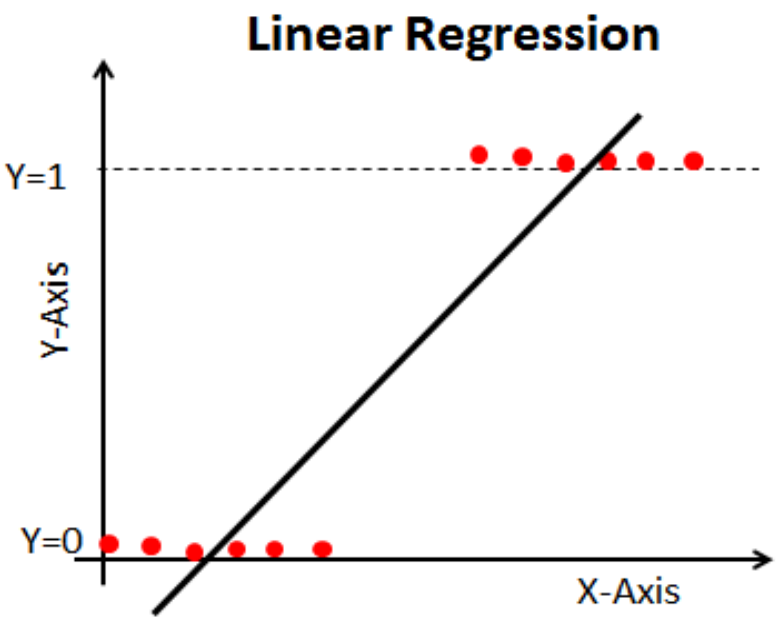
- explained_variance_score
- max_error
- mean_absolute_error
- mean_squared_error
- mean_squared_log_error
- median_absolute_error
- r2_score
- mean_poisson_deviance
- mean_gamma_deviance

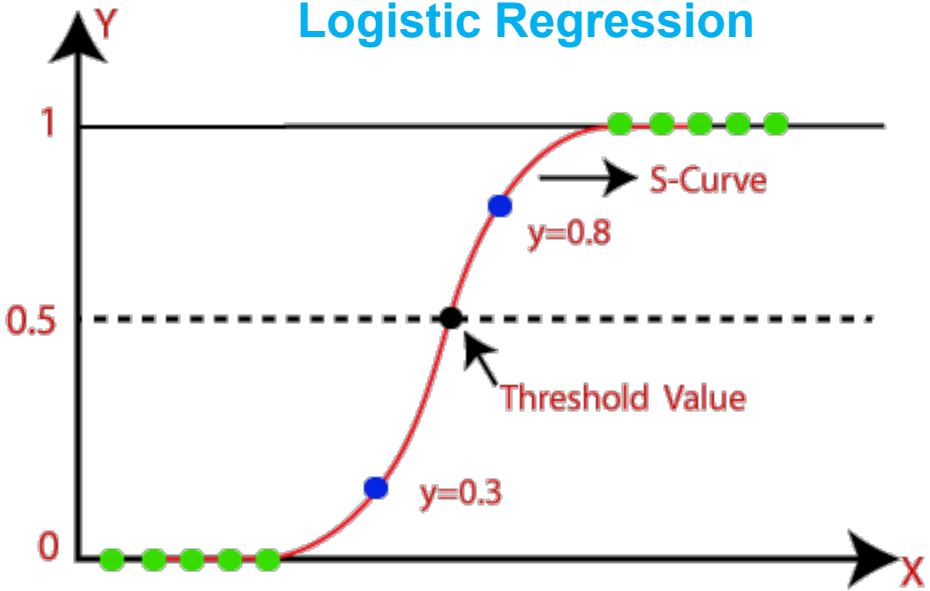


ShuffleSplit()
Kfold
LeaveOneOut()

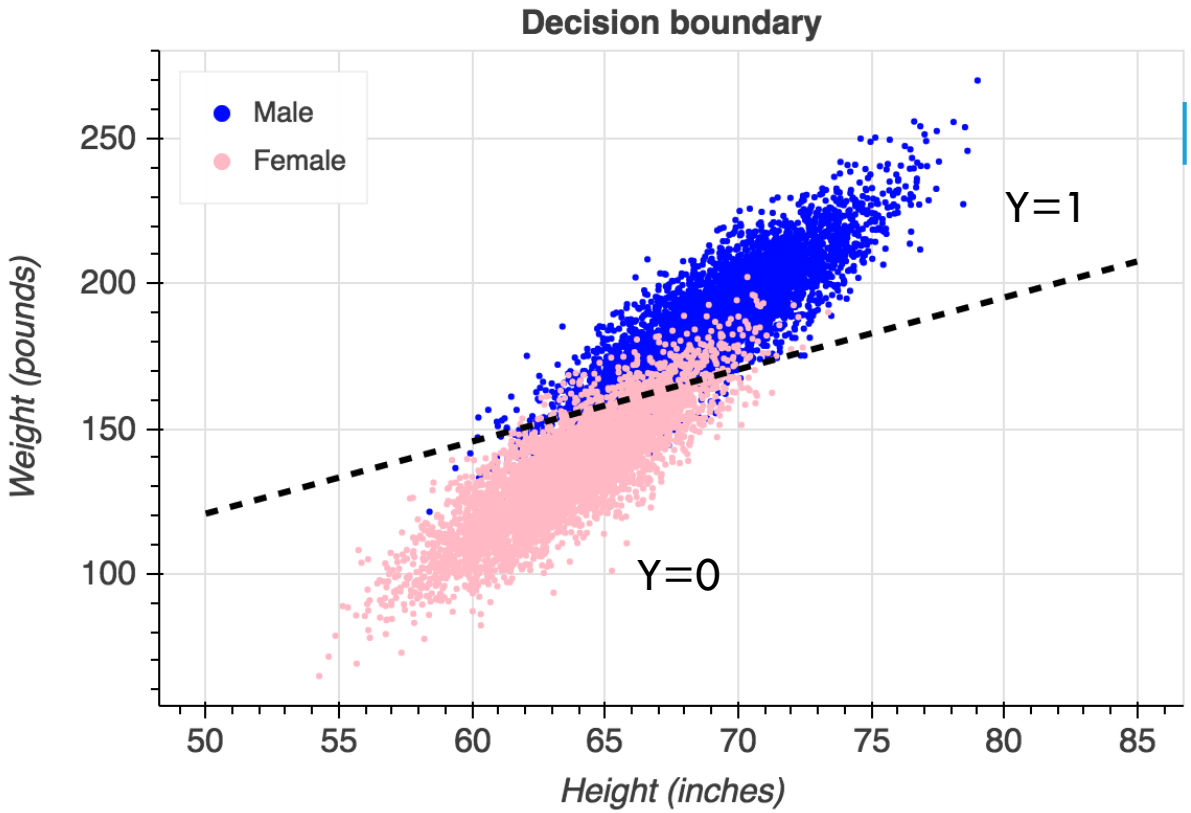


Logistic Regression: Intro





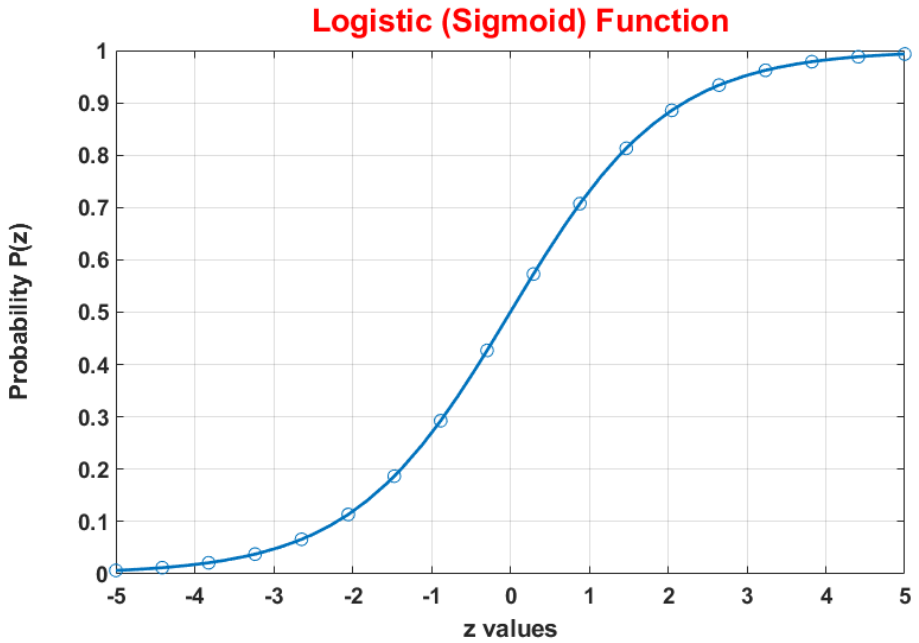
Logistic Regression: Intro



Logistic Regression: Details

Logistic Regression finds the probability of a given instance to belong to a default class (Y=1)

Logistic Regression uses the Logistic Function model the Probabilities



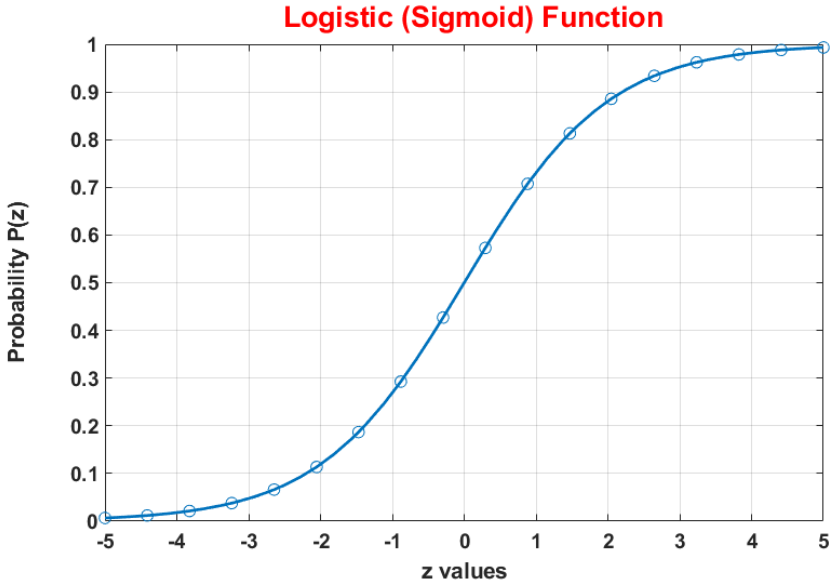
$$P(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

here, $z = w_0 + w_1 X_1 + w_2 X_2$

w_j are the "optimal" model weights

X_1 & X_2 are features..

Logistic Regression: More Details



$$P(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

here, $z = w_0 + w_1 X_1 + w_2 X_2$

w_j are the model weights

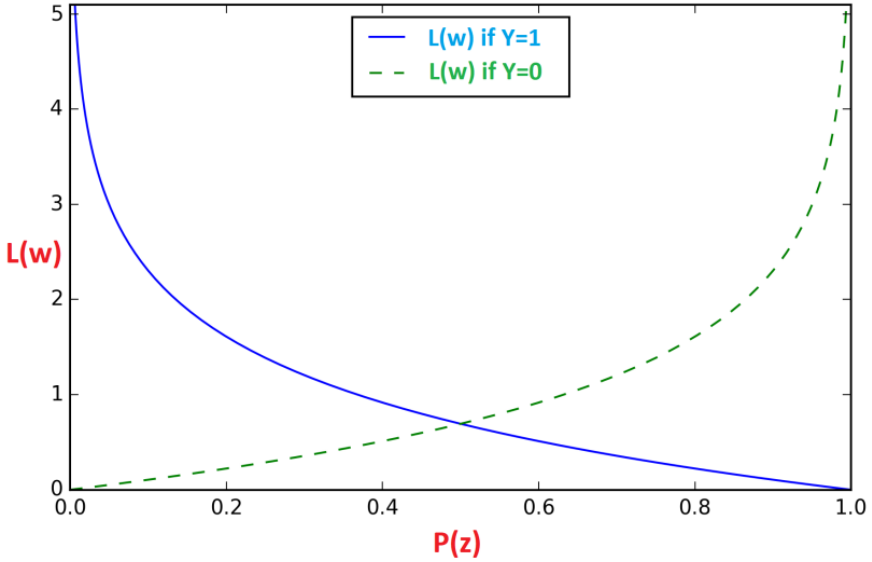
X_1 & X_2 are features..

The actual **Predicted class** is decided on the basis of a **"threshold" (which is hard-coded as 0.5)**

$$Y\_pred = \begin{cases} 0 \text{ if } P(z) < 0.5 \\ 1 \text{ if } P(z) \geq 0.5 \end{cases}$$

P(z) = P(Y=1 | Xi)

Logistic Regression: Loss Function



Intuition:

$$\text{cost} = \begin{cases} -Y \log_2 P & : \text{when } Y = 1 \\ -(1-Y)\log_2(1-P) & : \text{when } Y = 0 \end{cases}$$

Binary CrossEntropy Loss function:

$$L = \frac{1}{N}\sum\left[-Y \log_2 P - (1-Y)\log_2(1-P)\right]$$

Logistic Regression: Problem Formulation

$$\text{Given}: P(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

where, $z = w_0 + w_1 X_1 + w_2 X_2$

w_j are the model weights

X_1 & X_2 are features..

$$\text{Log of odds}: \log_e\left(\frac{P}{1-P}\right)$$

$= z \Rightarrow$ the Log of odds of X belonging to Y=1 class is a Linear Model

Binary CrossEntropy Loss function:

$$L = \frac{1}{N}\sum\left[-Y\log_2 P - (1-Y)\log_2(1-P)\right]$$

Derivative of the Loss function:

$$\frac{dL}{dw_j} = \frac{dL}{dP} \times \frac{dP}{dz} \times \frac{dz}{dw_j}$$

$$\frac{dL}{dw_j} = \frac{1}{N}\sum\left[(P-Y)\times X_j\right]$$

confusion matrix

		Predicted Class	
		0	**1**
Actual Class	**0**	151 (TN)	12 (FP)
	1	35 (FN)	69 (TP)

12 times the model predicts 0 as 1

35 times the model predicts 1 as 0

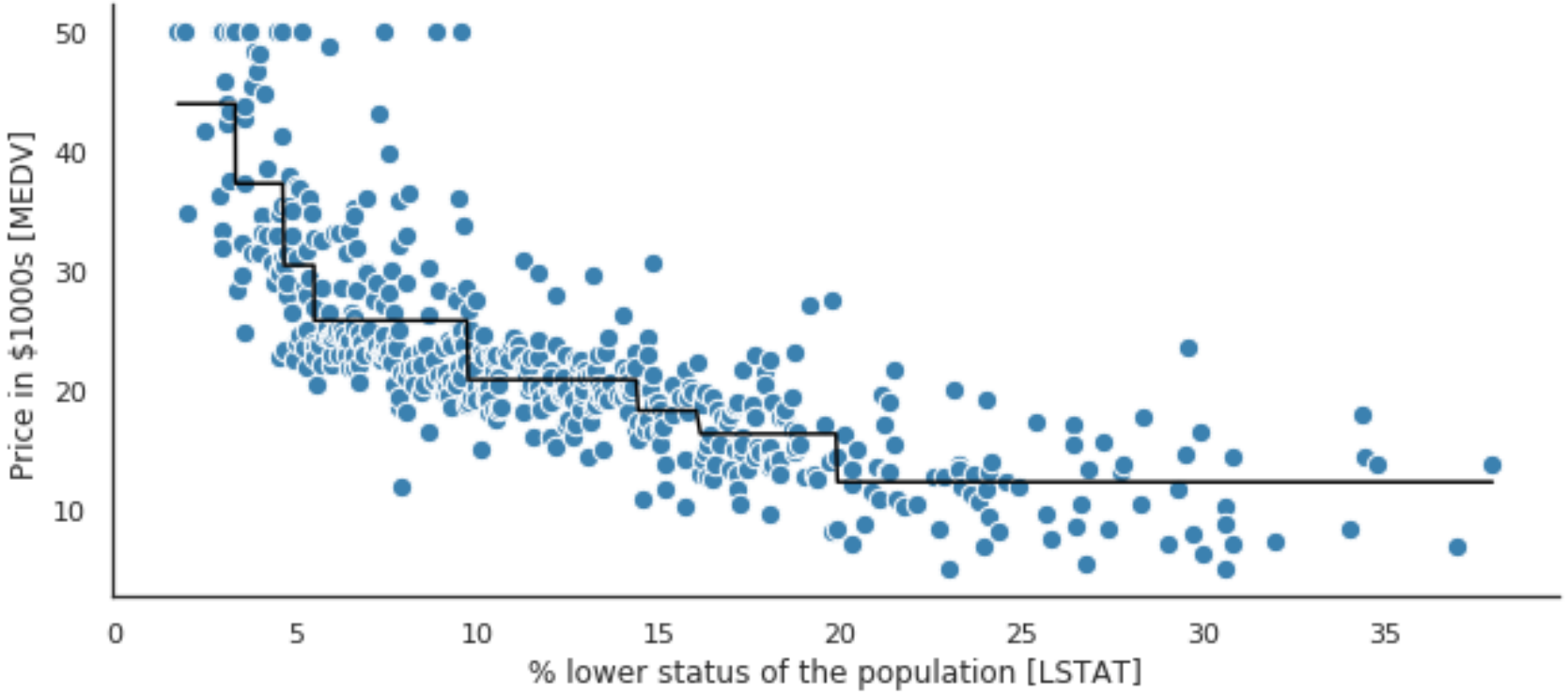
Considering "Positive" as "Survived"

F1 score is the harmonic mean of Pr & Recall

$$\frac{1}{F1} = \frac{1}{2}\left(\frac{1}{Prec} + \frac{1}{Recall}\right)$$

$$F1 = \frac{2 * Pr * Recall}{(Pr + Rec)}$$

Precision for 0	Actual zeros	151		
	predicted zeros	151+35	186	0.81183
Precision for 1	Actual ones	69		
	Predicted ones	12+69	81	0.85185
Recall for 0	predicted 0	151		
	Actual zeros	151+12	163	0.92638
Recall for ones	predicted 1	69		
	Actual ones	35+69	104	0.66346


Price in $1000s [MEDV]
50
40
30
20
10
0
5
10
15
20
25
30
35
% lower status of the population [LSTAT]

Unsupervised Machine Learning

In Unsupervised Learning, we train the models on similar sorts of data except for the fact that this dataset does not contain any label or outcome/target column.

Essentially, we train the model on data without any right answers.

In Unsupervised Learning, the machine tries to find hidden patterns and useful signals in the data that can be later used for other applications.

One of the uses is to find patterns within customer data and group the customers into different clusters that represent some of the properties.

Table 2-2. *Customer Details*

Customer ID	Song Genre
AS12	Romantic
BX54	Hip Hop
BX54	Rock
AS12	Rock
CH87	Hip Hop
CH87	Classical
AS12	Rock

In this data, we have customers and the kinds of music they prefer without any target or output column, simply the customers and their music preference data.

We can use unsupervised learning and group these customers into
meaningful clusters to know more about their group preference and act accordingly.

Cluster A can belong to customers who prefer only Rock and Cluster B can be of people preferring Romantic & Classical music, and the cluster C might be of Hip Hop and Rock lovers.

There are many applications that use unsupervised learning settings such as
Case 1: What are different groups within the total customer base?
Case 2: Is this transaction an anomaly or normal?

The algorithms used in unsupervised learning are
1. Clustering Algorithms (K-Means, Hierarchical)
2. Dimensionality Reduction Techniques
3. Topic Modeling
4. Association Rule Mining

The whole idea of Unsupervised learning is to discover and find out the patterns rather than making predictions.