

# Linear Regression

## Gradient Descent

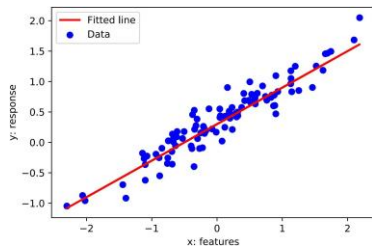
# What is Regression?

- Supervised learning algorithm.
- Consists of one or more feature values and a corresponding single output value.
- Output is a continuous value.
- Learns the line, plane or hyperplane that best fits the training samples.

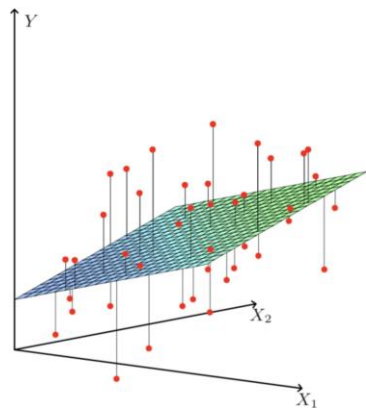
# Linear Regression

- **Linear Model:** assumes linear relationship between input variables ( $x$ ) and a single output variable ( $y$ ).
- **Simple Linear Regression:** single input variable ( $x$ ).
- **Multiple Linear Regression:** multiple input variables ( $x_1, x_2, \dots, x_n$ ).

# What is Linear? 1 Feature vs. D Features



- If we have only 1 feature:  
 $y = wx + b$  where  $w, x, b \in \mathbb{R}$ .
- $y$  is linear in  $x$ .



- If we have  $D$  features:  
 $y = \mathbf{w}^\top \mathbf{x} + b$  where  $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D$ ,  
 $b \in \mathbb{R}$
- $y$  is linear in  $\mathbf{x}$ .

Relation between the prediction  $y$  and inputs  $\mathbf{x}$  is linear in both cases.

# Linear Regression

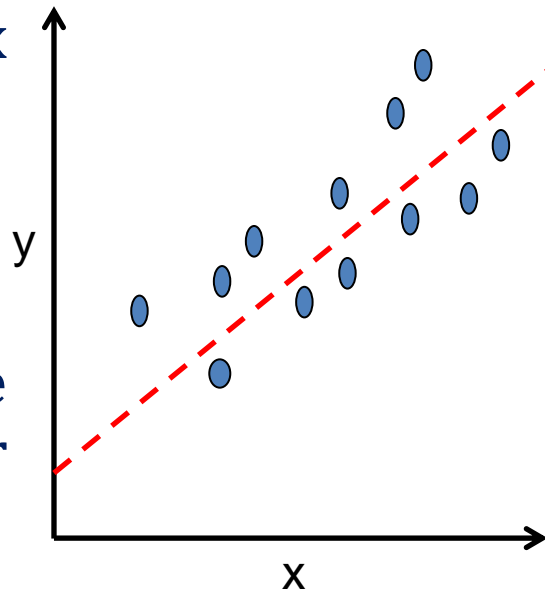
- In linear regression we assume that  $y$  and  $x$  are related with the following equation:

What we are  
trying to predict

$$y = b_0 + b_1 \cdot x + \epsilon$$

Feature values

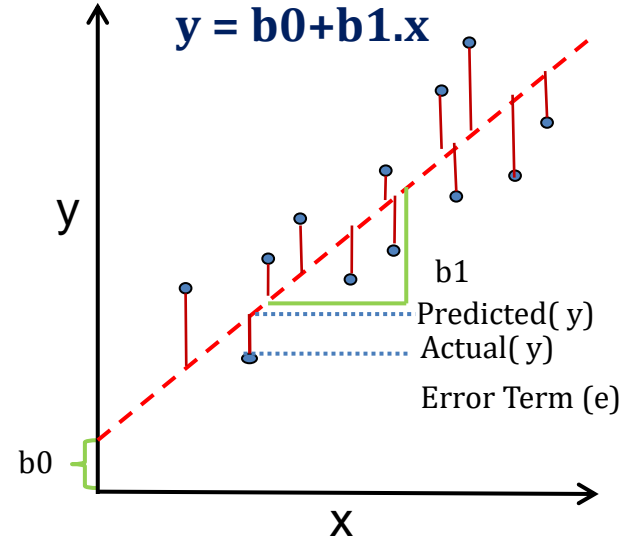
where ' $b_0$ ' is a intercept at  $y$ -axis, ' $b_1$ ' is slope of line, ' $\epsilon$ ' represents error in measurement or other noise



# Linear Regression

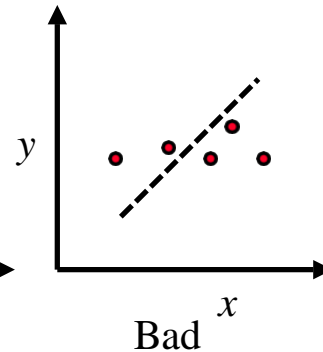
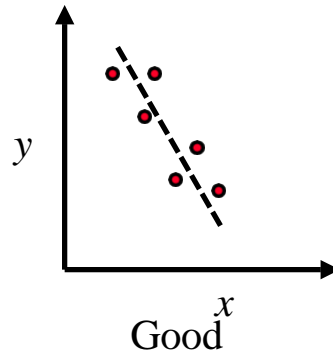
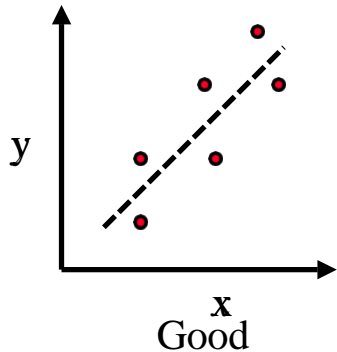
- **Goal:** to estimate  $w$  from a training data of  $\langle x_i, y_i \rangle$  pairs
- **Optimization goal:** minimize squared error (least squares)

$$\arg \min_w \sum_i (y_i - (b_0 + b_1 \cdot x_i))^2$$



# Good Model

- Regression models attempt to minimize the distance measured vertically between the observation point and the model line (or curve).
- The length of the line segment is called residual, modeling error, or simply error.
- The negative and positive errors should cancel out  
 $\Rightarrow$  Zero overall error  
Many lines will satisfy this criterion.



# Good Model (Cont)

- Choose the line that minimizes the sum of squares of the errors.

$$\hat{y} = b_0 + b_1 x$$

where,  $\hat{y}$  is the predicted response when the predictor variable is  $x$ . The parameter  $b_0$  and  $b_1$  are fixed regression parameters to be determined from the data.

- Given  $n$  observation pairs  $\{(x_1, y_1), \dots, (x_n, y_n)\}$ , the estimated response  $\hat{y}_i$  for the  $i$ th observation is:

$$\hat{y}_i = b_0 + b_1 x_i$$

- The error is:

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



# Good Model (Cont)

- The best linear model minimizes the sum of squared errors (SSE):

$$\sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - b_0 - b_1 x_i)^2$$

subject to the constraint that the mean error is zero:

$$\sum_{i=1}^n e_i = \sum_{i=1}^n (y_i - b_0 - b_1 x_i) = 0$$

- This is equivalent to minimizing the variance of errors (see Exercise).

# Estimation of Model Parameters

- Regression parameters that give minimum error variance are:

$$b_1 = \frac{\Sigma xy - n\bar{x}\bar{y}}{\Sigma x^2 - n\bar{x}^2} \quad \text{and} \quad b_0 = \bar{y} - b_1\bar{x}$$

- where,

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

$$\Sigma xy = \sum_{i=1}^n x_i y_i \quad \Sigma x^2 = \sum_{i=1}^n x_i^2$$

# Example

- The number of disk I/O's and processor times of seven programs were measured as: (14, 2), (16, 5), (27, 7), (42, 9), (39, 10), (50, 13), (83, 20)
- For this data:  $n=7$ ,  $\Sigma xy=3375$ ,  $\Sigma x=271$ ,  $\Sigma x^2=13,855$ ,  $\Sigma y=66$ ,  $\Sigma y^2=828$ ,  $\bar{x}= 38.71$ ,  $\bar{y}= 9.43$ . Therefore,

$$b_1 = \frac{\Sigma xy - n\bar{x}\bar{y}}{\Sigma x^2 - n(\bar{x})^2} = \frac{3375 - 7 \times 38.71 \times 9.43}{13,855 - 7 \times (38.71)^2} = 0.2438$$

$$b_0 = \bar{y} - b_1\bar{x} = 9.43 - 0.2438 \times 38.71 = -0.0083$$

- The desired linear model is:

$$\text{CPU time} = -0.0083 + 0.2438(\text{Number of Disk I/O's})$$

# Example (Cont)

## □ Error Computation

Disk I/O's	CPU Time	Estimate	Error	Error <sup>2</sup>
$x_i$	$y_i$	$\hat{y}_i = b_0 + b_1 x_i$	$e_i = y_i - \hat{y}_i$	$e_i^2$
14	2	3.4043	-1.4043	1.9721
16	5	3.8918	1.1082	1.2281
27	7	6.5731	0.4269	0.1822
42	9	10.2295	-1.2295	1.5116
39	10	9.4982	0.5018	0.2518
50	13	12.1795	0.8205	0.6732
83	20	20.2235	-0.2235	0.0500
$\Sigma$	271	66	0.00	5.8690

# Derivation of Regression Parameters

- The error in the  $i$ th observation is:

$$e_i = y_i - \hat{y}_i = y_i - (b_0 + b_1 x_i)$$

- For a sample of  $n$  observations, the mean error is:

$$\begin{aligned}\bar{e} &= \frac{1}{n} \sum_i e_i = \frac{1}{n} \sum_i \{y_i - (b_0 + b_1 x_i)\} \\ &= \bar{y} - b_0 - b_1 \bar{x}\end{aligned}$$

- Setting mean error to zero, we obtain:

$$b_0 = \bar{y} - b_1 \bar{x}$$

- Substituting  $b_0$  in the error expression, we get:

$$e_i = y_i - \bar{y} + b_1 \bar{x} - b_1 x_i = (y_i - \bar{y}) - b_1 (x_i - \bar{x})$$

## Derivation of Regression Parameters (Cont)

□ The sum of squared errors SSE is:

$$\begin{aligned}\text{SSE} &= \sum_{i=1}^n e_i^2 \\&= \sum_{i=1}^n \left\{ (y_i - \bar{y})^2 + 2b_1 (y_i - \bar{y}) (x_i - \bar{x}) + b_1^2 (x_i - \bar{x})^2 \right\} \\&= \frac{1}{n-1} \sum_{i=1}^n (y_i - \bar{y})^2 - 2b_1 \frac{1}{n-1} \sum_{i=1}^n (y_i - \bar{y}) (x_i - \bar{x}) \\&\quad + b_1^2 \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \\&= s_y^2 - 2b_1 s_{xy}^2 + b_1^2 s_x^2\end{aligned}$$

# Derivation (Cont)

- Differentiating this equation with respect to  $b_1$  and equating the result to zero:

$$\frac{d(\text{SSE})}{db_1} = -2s_{xy}^2 + 2b_1s_x^2 = 0$$

- That is,

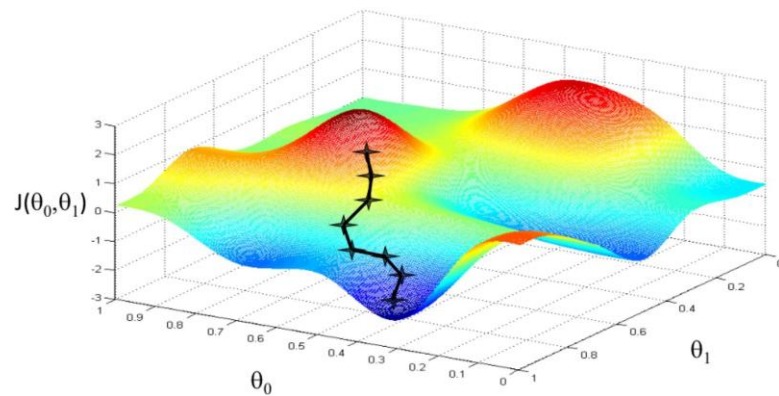
$$b_1 = \frac{s_{xy}^2}{s_x^2} = \frac{\Sigma xy - n\bar{x}\bar{y}}{\Sigma x^2 - n(\bar{x})^2}$$

# Gradient Descent

- Method of updating  $b_0$  and  $b_1$  values, to reduce the RMSE.
- Gradient Descent Algorithm:
  1. Start with initial guess of coefficients.
  2. Keep changing the coefficients little bit to try and reduce the cost function  $J(b_0, b_1)$ .
  3. Each time the parameters are changed the gradient is chosen which reduces  $J(b_0, b_1)$  the most.
  4. Repeat
  5. Keep doing till no improvement is made.



# Gradient Descent



# Gradient Descent

- Observe:
  - ▶ if  $\partial\mathcal{J}/\partial w_j > 0$ , then increasing  $w_j$  increases  $\mathcal{J}$ .
  - ▶ if  $\partial\mathcal{J}/\partial w_j < 0$ , then increasing  $w_j$  decreases  $\mathcal{J}$ .
- The following update decreases the cost function:

$$\begin{aligned}w_j &\leftarrow w_j - \alpha \frac{\partial\mathcal{J}}{\partial w_j} \\&= w_j - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) x_j^{(i)}\end{aligned}$$

- $\alpha$  is a **learning rate**. The larger it is, the faster  $\mathbf{w}$  changes.
  - ▶ We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001

# Gradient Descent

- This gets its name from the [gradient](#):

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

- This is the direction of fastest increase in  $\mathcal{J}$ .
- Update rule in vector form:

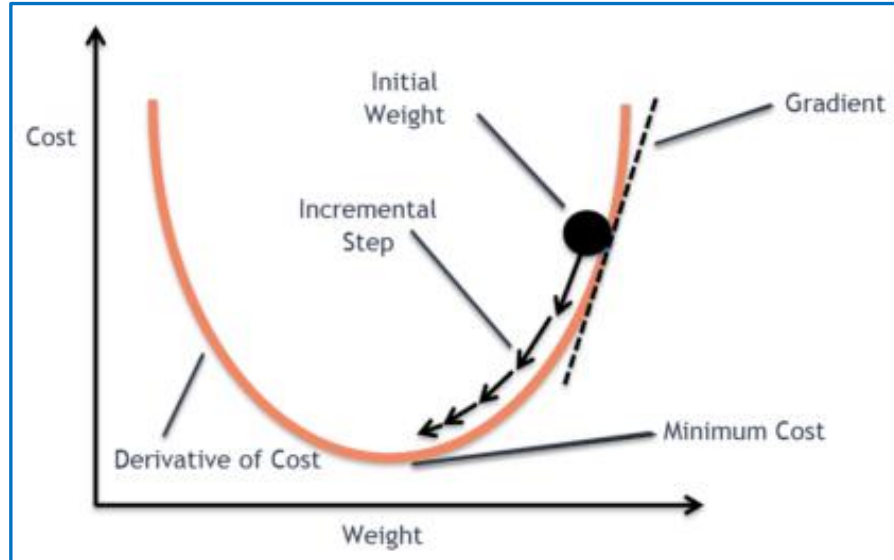
$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ &= \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)} \end{aligned}$$

- Hence, gradient descent updates the weights in the direction of fastest *decrease*.
- Observe that once it converges, we get a critical point, i.e.  $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = 0$ .

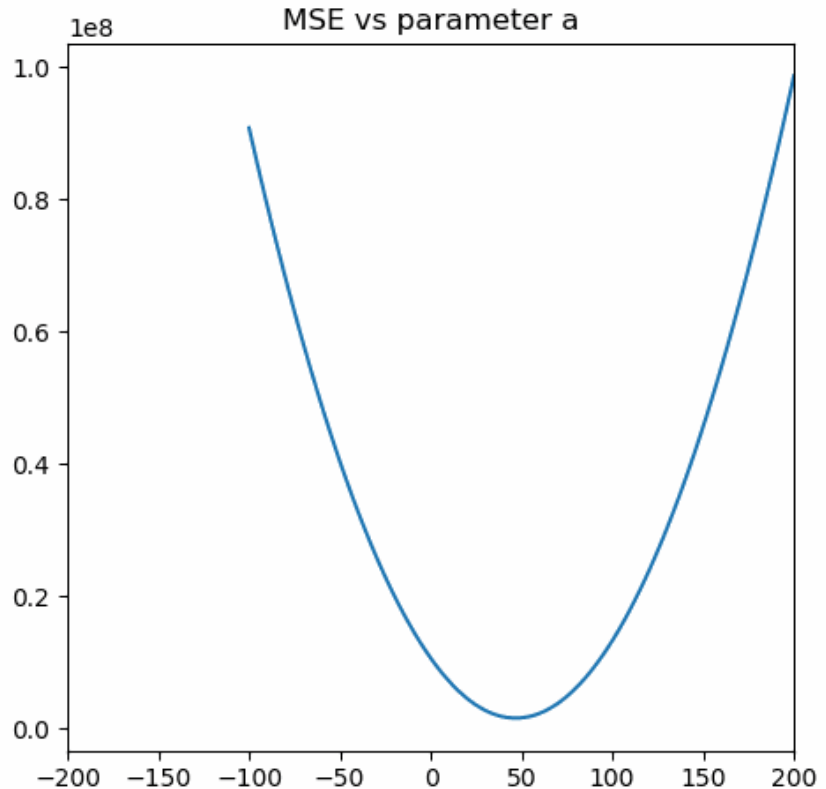
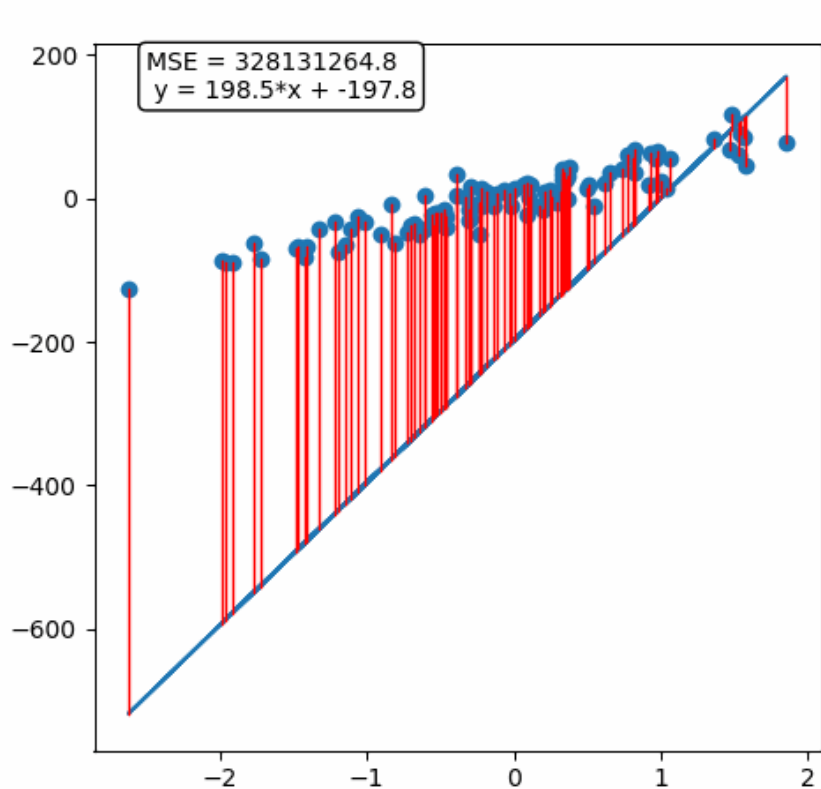
# Gradient Descent for Linear regression

- Even for linear regression, where there is a direct solution, we sometimes need to use GD.
- Why gradient descent, if we can find the optimum directly?
  - ▶ GD can be applied to a much broader set of models
  - ▶ GD can be easier to implement than direct solutions
  - ▶ For regression in high-dimensional spaces, GD is more efficient than direct solution
    - ▶ Linear regression solution:  $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$
    - ▶ matrix inversion is an  $\mathcal{O}(D^3)$  algorithm
    - ▶ each GD update costs  $\mathcal{O}(ND)$
    - ▶ Huge difference if  $D \gg 1$

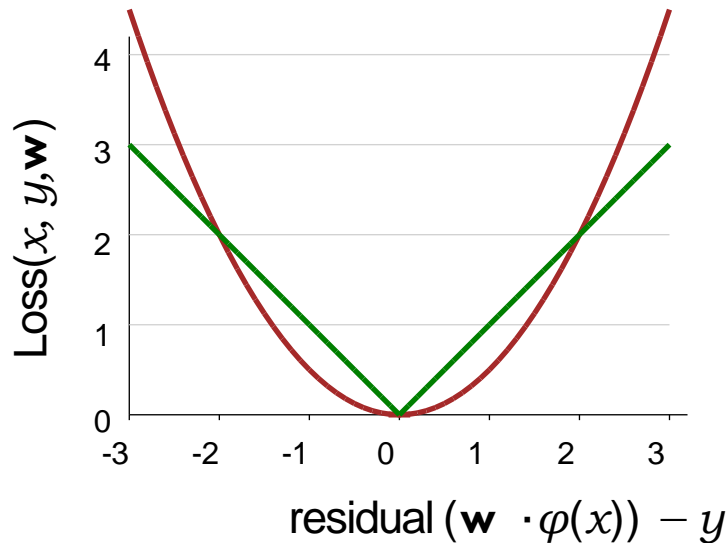
# Gradient Descent



# Gradient Descent



# Regression loss functions

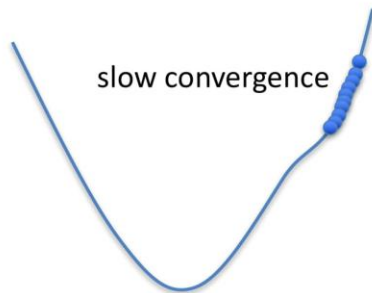


$$\text{LOSS}_{\text{squared}}(x, y, \mathbf{w}) = (\mathbf{w} \cdot \varphi(x) - y)^2$$

$$\text{LOSS}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} \cdot \varphi(x) - y|$$

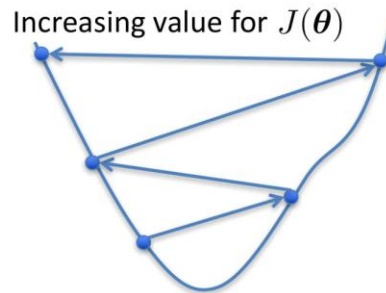
# Choosing $\alpha$

$\alpha$  too small



slow convergence

$\alpha$  too large



Increasing value for  $J(\theta)$

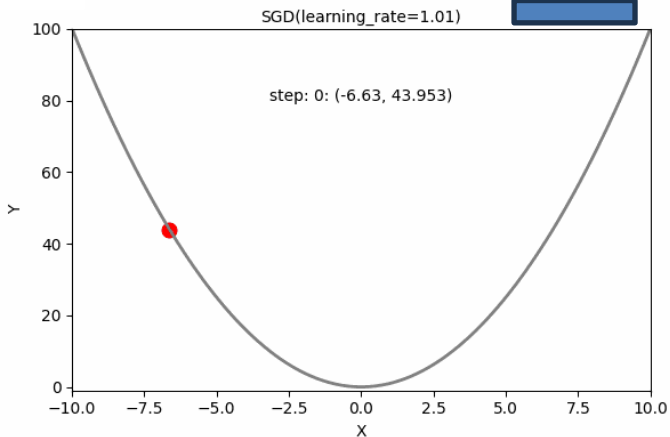
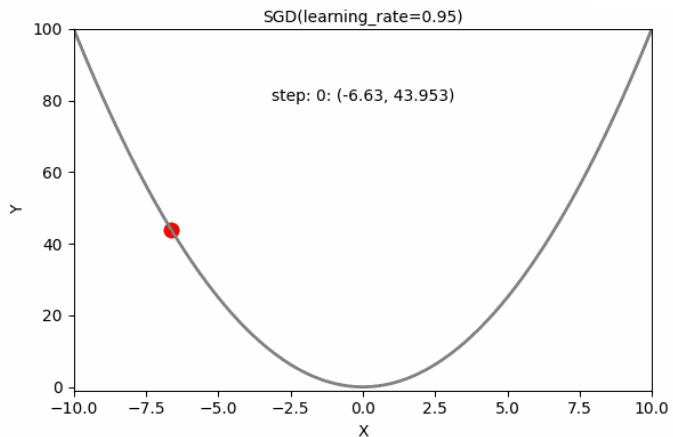
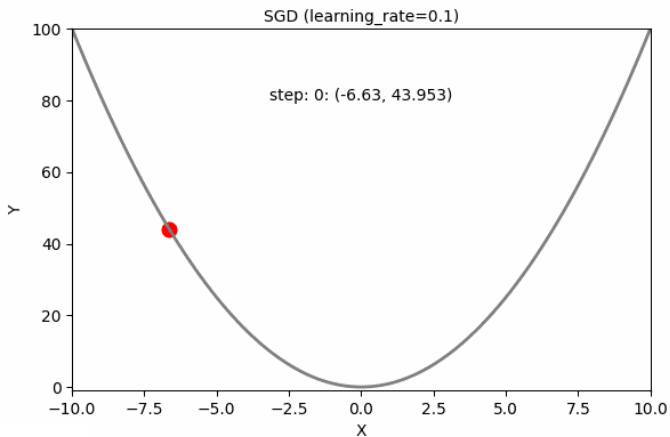
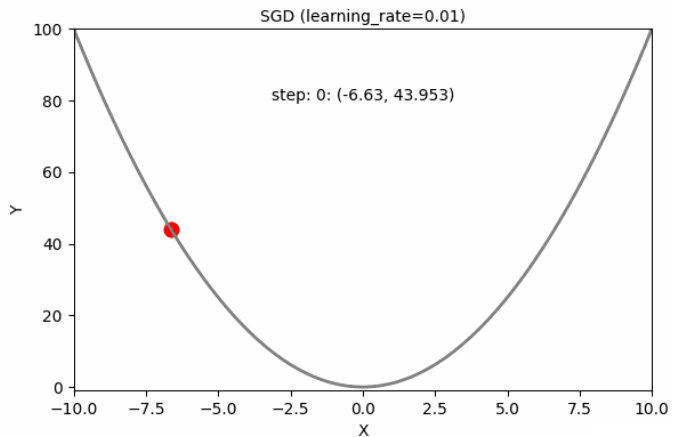
- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out  $J(\theta)$  each iteration

- The value should decrease at each iteration
- If it doesn't, adjust  $\alpha$

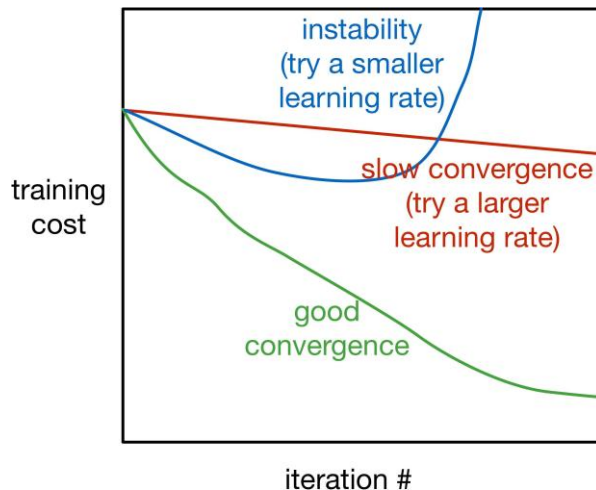


# Choosing $\alpha$



# Training Curves

- To diagnose optimization problems, it's useful to look at **training curves**: plot the training cost as a function of iteration.



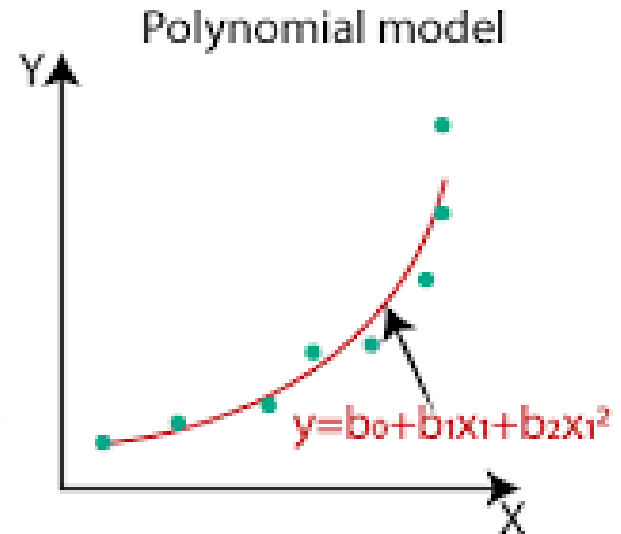
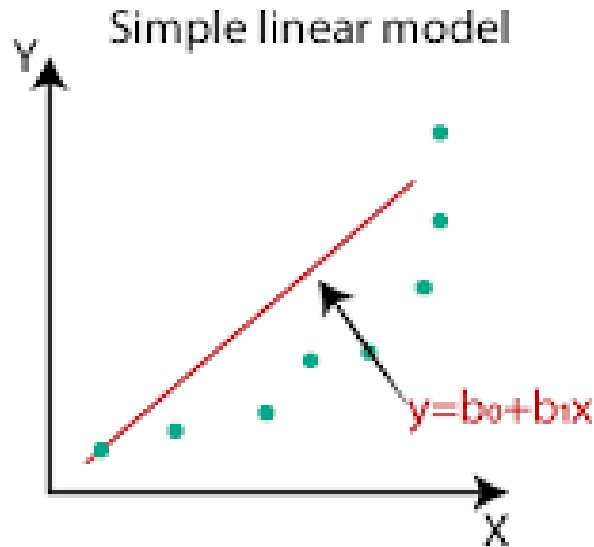
- Warning: it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

# Polynomial Regression

# Polynomial Regression

## Goal of this Session:

- Understand how to extend simple OLS linear regression in a flexible way using polynomial regression.



# Moving Beyond Linearity:

- The linearity assumption is good in many machine learning problems.
- However, there are other methods that offer a lot of flexibility, without losing the ease and interpretability of linear models
  - Polynomial regression

# Polynomial Regression:

- Replace the standard linear model with a polynomial function:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \dots \beta_d x_i^d + \epsilon_i$$

# Polynomial Regression:

- Second order polynomial in one variable:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

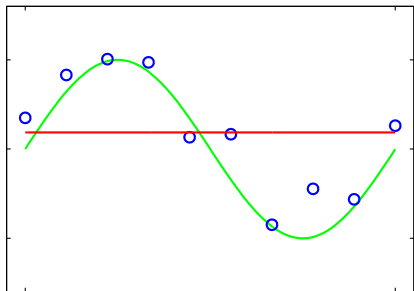
- Second order polynomial in two variables:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_{11} x_{i1}^2 + \beta_{22} x_{i2}^2 + \beta_{12} x_{i1} x_{i2} + \epsilon_i$$

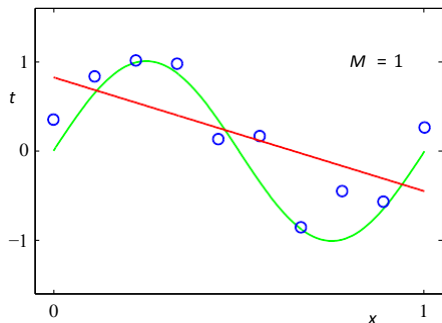
# Order of the Polynomial Model:

- The order ( $k$ ) should be as low as possible.
- The high-order polynomials ( $k > 2$ ) should be avoided unless they can be justified for reasons outside the data.
- Complex models can result in “*model overfitting*”

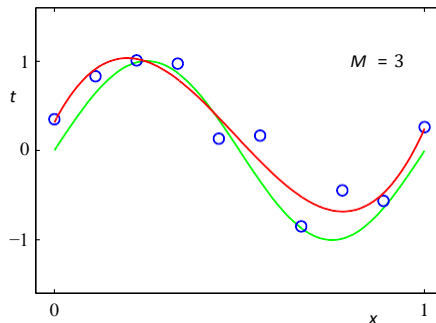
$$y = w_0$$



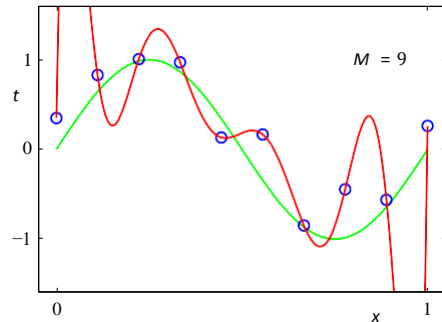
$$y = w_0 + w_1x$$



$$y = w_0 + w_1x + w_2x^2 + w_3x^3$$

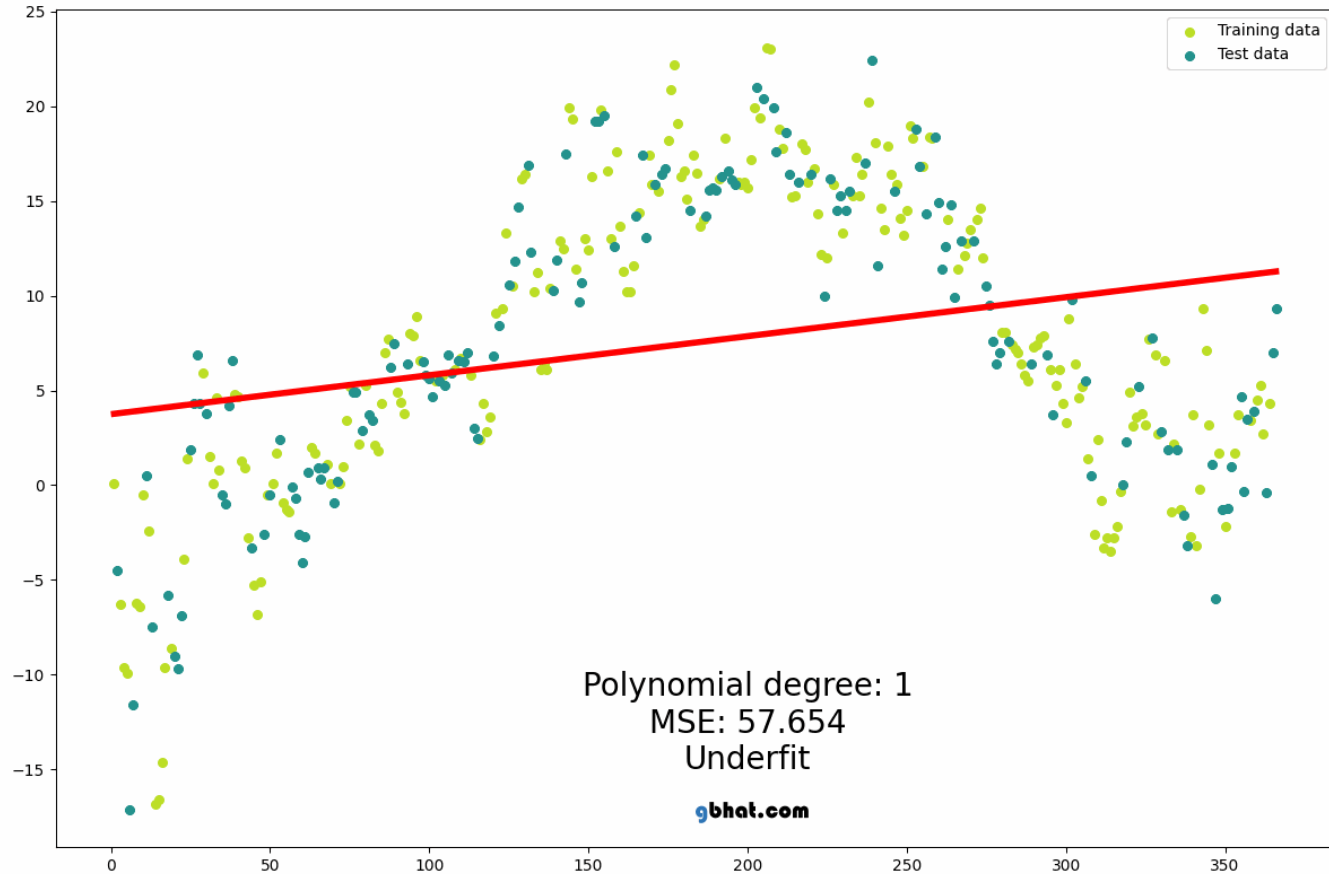


$$y = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots + w_9x^9$$





# Polynomial Regression

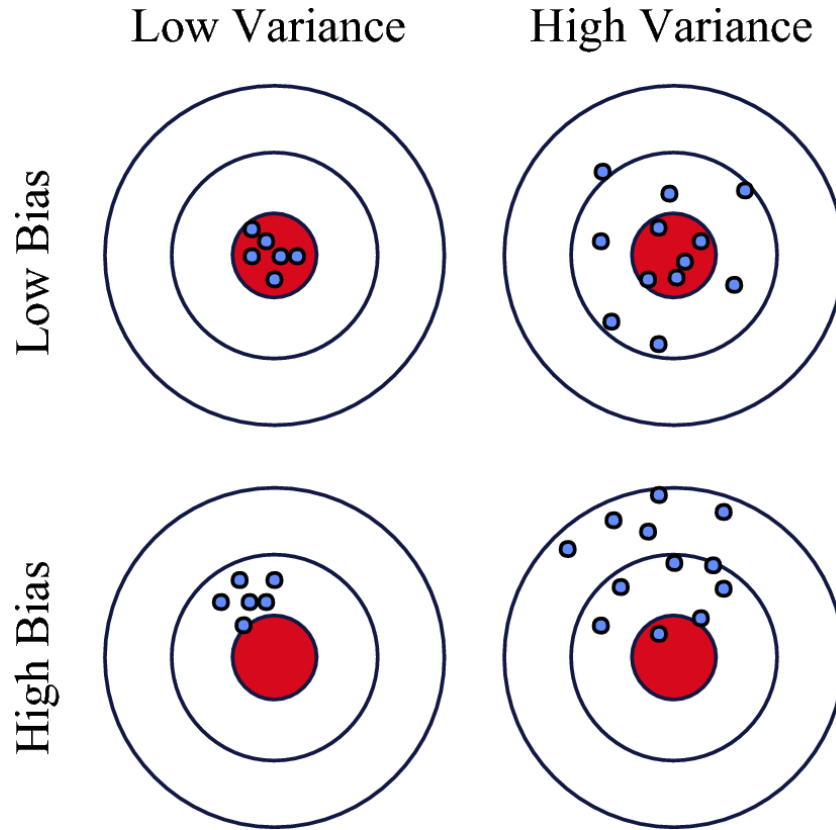


# **Regularization in Regression Models:**

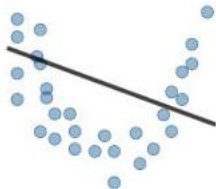


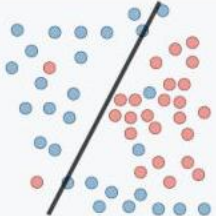
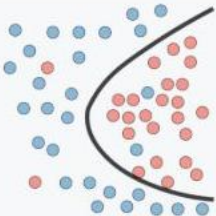
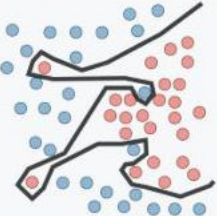

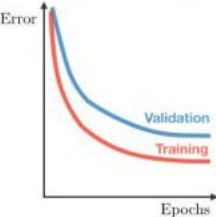

# Regularization in Regression Models:

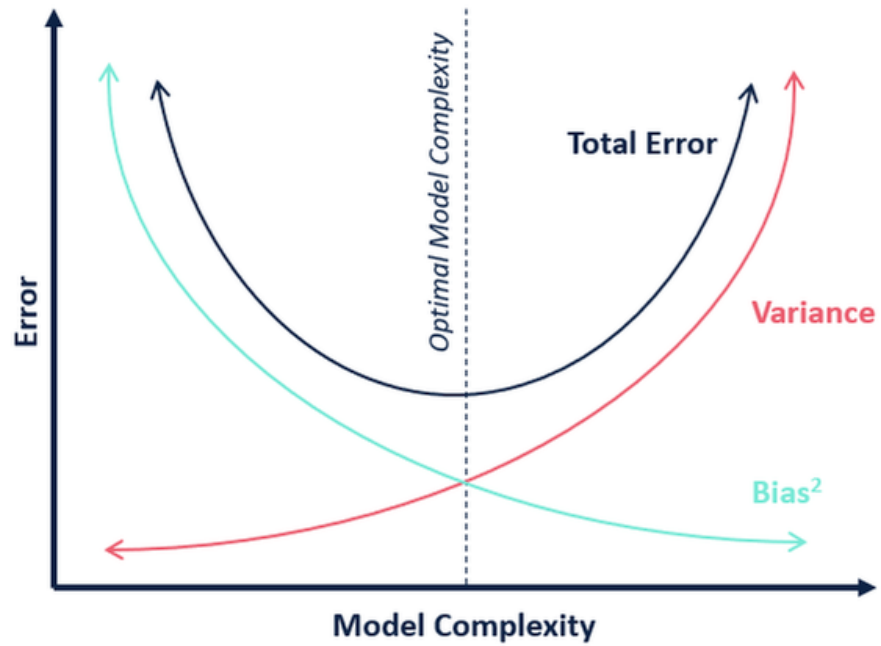
- Bias and Variance Tradeoff in Multiple Regression:
- The “**bias**” is the difference between the true population parameter ( $\beta$ ) and the expected estimator ( $\hat{\beta}$ ). It measures the inaccuracy between the estimates.
- “**Variance**” measures the spread or uncertainty, in these estimates.
- Both, the bias and the variance are desired to be low.

# Bull's eye diagram - Bias vs Variance



# Underfitting vs Overfitting

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"><li>• High training error</li><li>• Training error close to test error</li><li>• High bias</li></ul>	<ul style="list-style-type: none"><li>• Training error slightly lower than test error</li></ul>	<ul style="list-style-type: none"><li>• Very low training error</li><li>• Training error much lower than test error</li><li>• High variance</li></ul>
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"><li>• Complexify model</li><li>• Add more features</li><li>• Train longer</li></ul>		<ul style="list-style-type: none"><li>• Perform regularization</li><li>• Get more data</li></ul>



# Regularization in Regression Models:

- OLS estimator can have huge variance that results due to:
  - too many predictors
  - Predictors are highly correlated with each other.

## Solution:

Reduce variance at the cost of introducing some bias. This approach is called  
**“Regularization”**

# Regularization in Regression Models:

**Ridge Regression**

**Lasso Regression**

**Elastic Net**



# Ridge Regression

- Instead of forcing the predictors to be zero (i.e. removing them from the model), ridge regression penalize them if they are too far from zero.
- Thus, forcing the predictors to be small in a continuous way.
- This way, it decreases model complexity while keeping all variables in the model.
- Also, called as L2-penalty.

$$L_{ridge}(\beta^{\wedge}) = \sum_{i=1}^n (y_i - x_i' \beta^{\wedge})^2 + \lambda \sum_{j=1}^m \beta^{\wedge}_j{}^2$$

- if  $\lambda = 0$ , then  $\beta^{\wedge}$  – ridge is same as  $\beta^{\wedge}$  - OLS
- the larger the value of  $\lambda$  is, the more are the coefficients shrunk towards zero.

# Lasso Regression (Least Absolute Shrinkage and Selection Operator)

- Similar to ridge
- Unlike ridge regression which penalizes sum of squared coefficients, it penalizes the sum of their absolute values.
- L1 penalty

$$L_{Lasso}(\beta^{\wedge}) = \sum_{i=1}^n (y_i - x_i' \beta^{\wedge})^2 + \lambda \sum_{j=1}^m \beta_j$$

# Elastic Net

Elastic Net combines the penalties of Ridge and LASSO.

Addresses several shortcomings of LASSO:

- for  $n < p$  (more covariates/features than samples) LASSO can select only  $n$  covariates (even if more are truly associated with the response)
- it tends to select only one covariate from any set of highly correlated covariates
- for  $n > p$ , if the covariates are strongly correlated, Ridge tends to perform better

Elastic Net:

- highly correlated covariates will tend to have similar regression coefficients (desirable *grouping effect*)

$$L_{Elastic}(\hat{\beta}) = \sum_{i=1}^n (y_i - x_i' \hat{\beta})^2 + \lambda_1 \sum_{j=1}^m \beta_j + \lambda_2 \sum_{j=1}^m \beta_j^2$$

## Vector Notation

- We can organize all the training examples into a **design matrix**  $\mathbf{X}$  with one row per training example, and all the targets into the **target vector**  $\mathbf{t}$ .

one feature across  
all training examples

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{pmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{pmatrix}$$

one training  
example (vector)

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^T \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^T \mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

## Vectorization

- Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N} \|\mathbf{y} - \mathbf{t}\|^2$$

- Note that sometimes we may use  $\mathcal{J} = \frac{1}{2} \|\mathbf{y} - \mathbf{t}\|^2$ , without normalizer. That would correspond to the sum of losses, and not the average loss. The minimizer does not depend on  $N$ .
- We can also add a column of 1s to the design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ 1 & [\mathbf{x}^{(2)}]^\top \\ \vdots & \vdots \end{bmatrix} \in \mathbb{R}^{N \times D+1} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our predictions reduce to  $\mathbf{y} = \mathbf{X}\mathbf{w}$ .

## Direct Solution

- The minimum must occur at a point where the partial derivatives are zero, i.e.,

$$\frac{\partial \mathcal{J}}{\partial w_j} = 0 \quad (\forall j), \quad \frac{\partial \mathcal{J}}{\partial b} = 0.$$

- If  $\partial \mathcal{J} / \partial w_j \neq 0$ , you could reduce the cost by changing  $w_j$ .
- This turns out to give a system of linear equations, which we can solve efficiently. **Full derivation in the preliminaries.pdf.**
- Optimal weights:

$$\mathbf{w}^{\text{LS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

- Linear regression is one of only a handful of models in this course that permit direct solution.

# Vectorization

- Computing the prediction using a for loop:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

- For-loops in Python are slow, so we **vectorize** algorithms by expressing them in terms of vectors and matrices.

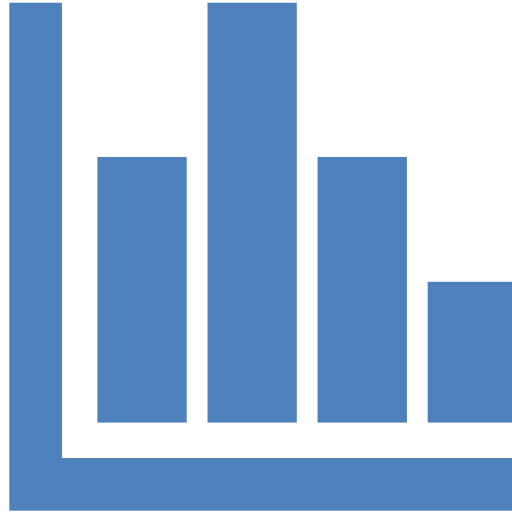
$$\mathbf{w} = (w_1, \dots, w_D)^T \quad \mathbf{x} = (x_1, \dots, x_D)$$

$$y = \mathbf{w}^T \mathbf{x} + b$$

- This is simpler and much faster:

```
y = np.dot(w, x) + b
```

# Different Statistical Performance Metric for Regression





# Evaluation metrics for Regression

## 1. Mean Absolute Error (MAE)

Mean Absolute Error is the average of the difference between the original value and the predicted value. It gives you the measure of how far the predictions are from the actual output and obviously, you would want to minimize it. However, it doesn't give you an idea of the direction of the error since you are taking only the absolute values. It is most Robust to outliers.

$$\text{MAE} = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

**Demerits:** The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

where,

$n$  is the number of observations

$y_j$  is the actual value for sample  $j$

$\hat{y}_j$  is the predicted value for sample  $j$

# Evaluation metrics for Regression

## 2. Mean squared error (MSE)

This is the mean of the squared difference of the actual value in the dataset and the value predicted by the model. Mean Squared Error (MSE) is similar to MAE, the only difference being that MSE takes the average of the square of the difference between the original values and the predicted values which eases the process of gradient calculation, penalizes the error terms more, and is unbiased towards the direction of error (since you are squaring). However, this makes it more sensitive to outliers. The graph of MSE is differentiable, so you can easily use it as a loss function.

$$\text{MSE} = \frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2$$

**Demerits:** The value you get after calculating MSE is a squared unit of output.

where,

$n$  is the number of observations

$y_j$  is the actual value for sample  $j$

$\hat{y}_j$  is the predicted value for sample  $j$

# Evaluation metrics for Regression

## 3. Root mean squared error (RMSE)

It is the mean of root squared subtraction between the actual value in the dataset and the value predicted by the model. It's the same as MSE, we are just taking the root of it. The smaller the value of root mean squared error, the more accurate the model is. The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

**Demerits:** It is not that robust to outliers as compared to MAE.

where,

$n$  is the number of observations

$y_j$  is the actual value for sample  $j$

$\hat{y}_j$  is the predicted value for sample  $j$

# Evaluation metrics for Regression

## **4. Sum of Squares Total (SST)**

The sum of squared differences between individual data points and the mean of the response variable.

$$SST = \sum (y_i - \bar{y})^2$$

# Evaluation metrics for Regression

## **5. Sum of Squares Regression (SSR)**

The sum of squared differences between predicted data points ( $\hat{y}_i$ ) and the mean of the response variable( $\bar{y}$ ).

$$SSR = \sum (\hat{y}_i - \bar{y})^2$$

# Evaluation metrics for Regression

## 6. Sum of Squares Error (SSE)

The sum of squared differences between predicted data points ( $\hat{y}_i$ ) and observed data points ( $y_i$ ).

$$SSE = \sum (\hat{y}_i - y_i)^2$$

Relationship between SSR and SSE with SST is **SST = SSR + SSE**

# Evaluation metrics for Regression

## 7. R Square / Coefficient of Determination/ Goodness of fit

It estimates the ratio of the variance of the dependent element described by the target element. It's used for finding the accuracy of the model. It depicts the closeness of the data points to the trend line made by the model. This helps to make a link between the independent element and the target element. R square is from zero to one. The nearer R square is to one, the more accurate the model.

$$R - Square = 1 - \frac{\sum(Y_{actual} - Y_{predicted})^2}{\sum(Y_{actual} - Y_{mean})^2}$$

$$R^2 = 1 - \frac{SSE}{SST}$$

# Evaluation metrics for Regression

## 8. Adjusted R Squared

The disadvantage of the  $R^2$  score is while adding new features in data the  $R^2$  score starts increasing or remains constant but it never decreases because It assumes that while adding more data variance of data increases. But the problem is when we add an irrelevant feature in the dataset then at that time  $R^2$  sometimes starts increasing which is incorrect. Hence, To control this situation Adjusted R Squared came into existence.

$$R_a^2 = 1 - \left[ \left( \frac{n-1}{n-k-1} \right) \times (1 - R^2) \right]$$

where:

$n$  = number of observations

$k$  = number of independent variables

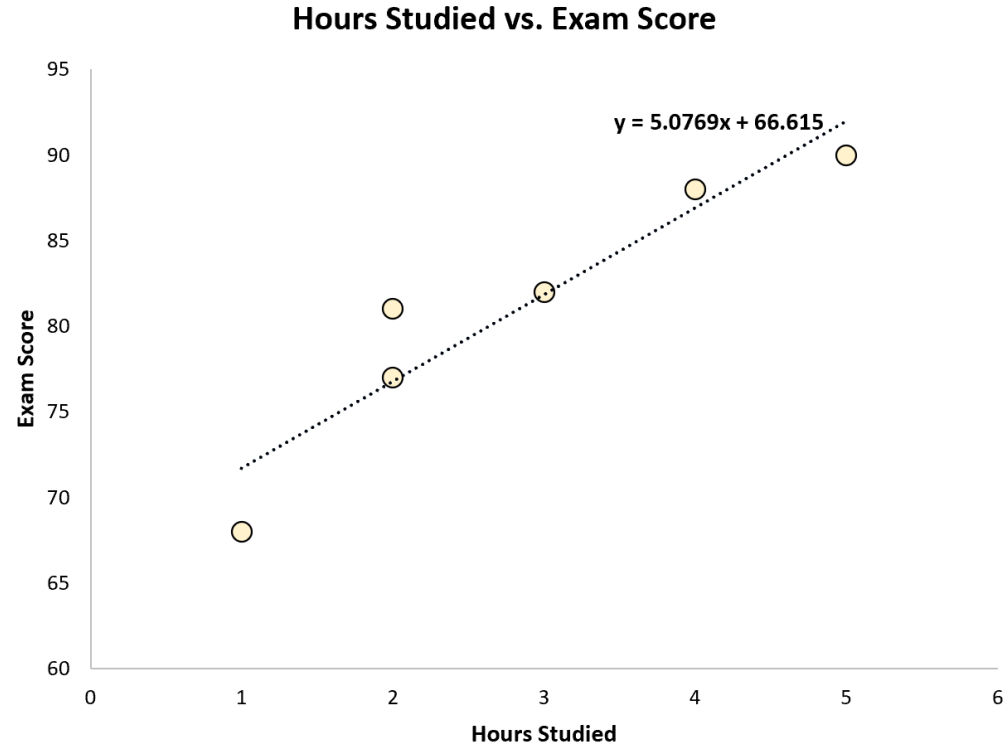
$R_a^2$  = adjusted  $R^2$



**Example 1:** Suppose we have the following dataset that shows the number of hours studied by six different students along with their final exam scores:

Hours Studied	Exam Score
1	68
2	77
2	81
3	82
4	88
5	90

- We can find that the line of best fit is:
- $\text{Score} = 66.615 + 5.0769 * (\text{Hours})$
- Once we know the line of best fit equation, we can use the following steps to calculate SST, SSR, and SSE:



**Step 1: Calculate the mean of the response variable.**

The mean of the response variable turns out to be **81**.

Hours Studied	Exam Score	$\bar{y}$
1	68	81
2	77	81
2	81	81
3	82	81
4	88	81
5	90	81

**Step 2: Calculate the predicted value for each observation.**

Next, we can use the line of best fit equation to calculate the predicted exam score ( $\hat{y}$ ) for each student.

For example, the predicted exam score for the student who studied one hour is:

$$\text{Score} = 66.615 + 5.0769(1) = \mathbf{71.69}.$$

We can use the same approach to find the predicted score for each student:

Hours Studied	Exam Score	$\bar{y}$	$\hat{y}$
1	68	81	71.69
2	77	81	76.77
2	81	81	76.77
3	82	81	81.85
4	88	81	86.92
5	90	81	92.00

**Step 3: Calculate the sum of squares total (SST).**

Next, we can calculate the sum of squares total.  
For example, the sum of squares total for the first student is:

$$(y_i - \bar{y})^2 = (68 - 81)^2 = \mathbf{169}.$$

We can use the same approach to find the sum of squares total for each student:

The sum of squares total turns out to be **316**.

**Step 4: Calculate the sum of squares regression (SSR).**

Next, we can calculate the sum of squares regression.  
For example, the sum of squares regression for the first student is:

$$(\hat{y}_i - \bar{y})^2 = (71.69 - 81)^2 = \mathbf{86.64}.$$

We can use the same approach to find the sum of squares regression for each student:

Hours Studied	Exam Score	$\bar{y}$	$\hat{y}$	$(y_i - \bar{y})^2$
1	68	81	71.69	169
2	77	81	76.77	16
2	81	81	76.77	0
3	82	81	81.85	1
4	88	81	86.92	49
5	90	81	92.00	81
				316

**SST**

Hours Studied	Exam Score	$\bar{y}$	$\hat{y}$	$(y_i - \bar{y})^2$	$(\hat{y}_i - \bar{y})^2$
1	68	81	71.69	169	86.64
2	77	81	76.77	16	17.90
2	81	81	76.77	0	17.90
3	82	81	81.85	1	0.72
4	88	81	86.92	49	35.08
5	90	81	92.00	81	120.99
				316	279.23

**SST**

**SSR**

The sum of squares regression turns out to be **279.23**.

**Step 5: Calculate the sum of squares error (SSE).**

Next, we can calculate the sum of squares error. For example, the sum of squares error for the first student is:

$$(\hat{y}_i - y_i)^2 = (71.69 - 68)^2 = \mathbf{13.63}.$$

We can use the same approach to find the sum of squares error for each student:

We can verify that  $SST = SSR + SSE$

- $SST = SSR + SSE$
- $316 = 279.23 + 36.77$

We can also calculate the R-squared of the regression model by using the following equation:

- $R\text{-squared} = 1 - SSR / SST$
- $R\text{-squared} = 279.23 / 316$
- $R\text{-squared} = 0.8836$

Hours Studied	Exam Score	$\bar{y}$	$\hat{y}$	$(y_i - \bar{y})^2$	$(\hat{y}_i - \bar{y})^2$	$(\hat{y}_i - y_i)^2$
1	68	81	71.69	169	86.64	13.63
2	77	81	76.77	16	17.90	0.05
2	81	81	76.77	0	17.90	17.90
3	82	81	81.85	1	0.72	0.02
4	88	81	86.92	49	35.08	1.16
5	90	81	92.00	81	120.99	4.00
				316	279.23	36.77

**SST      SSR      SSE**

This tells us that **88.36%** of the variation in exam scores can be explained by the number of hours studied.

Thank you

