Regression

Linear Regression

- Linear regression quantifies the relationship between one or more predictor variables and one outcome variable.
- Linear regression is used for predictive analysis and modeling.
- Linear Regression Model Prediction

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the ith feature value.
- θ_j is the jth model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

Linear Regression

- Linear Regression model prediction (vectorized form)

$$\hat{y} = h_{\theta}(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

- θ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- θ^T is the transpose of θ (a row vector instead of a column vector).
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\theta^T \cdot \mathbf{x}$ is the dot product of θ^T and $\mathbf{x}$.
- h_θ is the hypothesis function, using the model parameters θ.

Linear Regression

- Most common performance measure of a regression model is the Root Mean Square Error (RMSE).

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

- To train a Linear Regression model, you need to find the value of θ that minimizes the RMSE.
- In practice, it is simpler to minimize the Mean Square Error (MSE) than the RMSE, and it leads to the same result.

Linear Regression

- MSE cost function for a Linear Regression model.

$$\mathrm{MSE}\left(\mathbf{X}, h_\theta\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Linear Regression

- **Advantages:**

1. Linear Regression performs well when the dataset is linearly separable.
2. Linear Regression is easier to implement, interpret and very efficient to train.
3. Linear Regression is prone to over-fitting but it can be easily avoided using some dimensionality reduction techniques, regularization (L1 and L2) techniques and cross-validation.

Linear Regression

- **Disadvantages:**

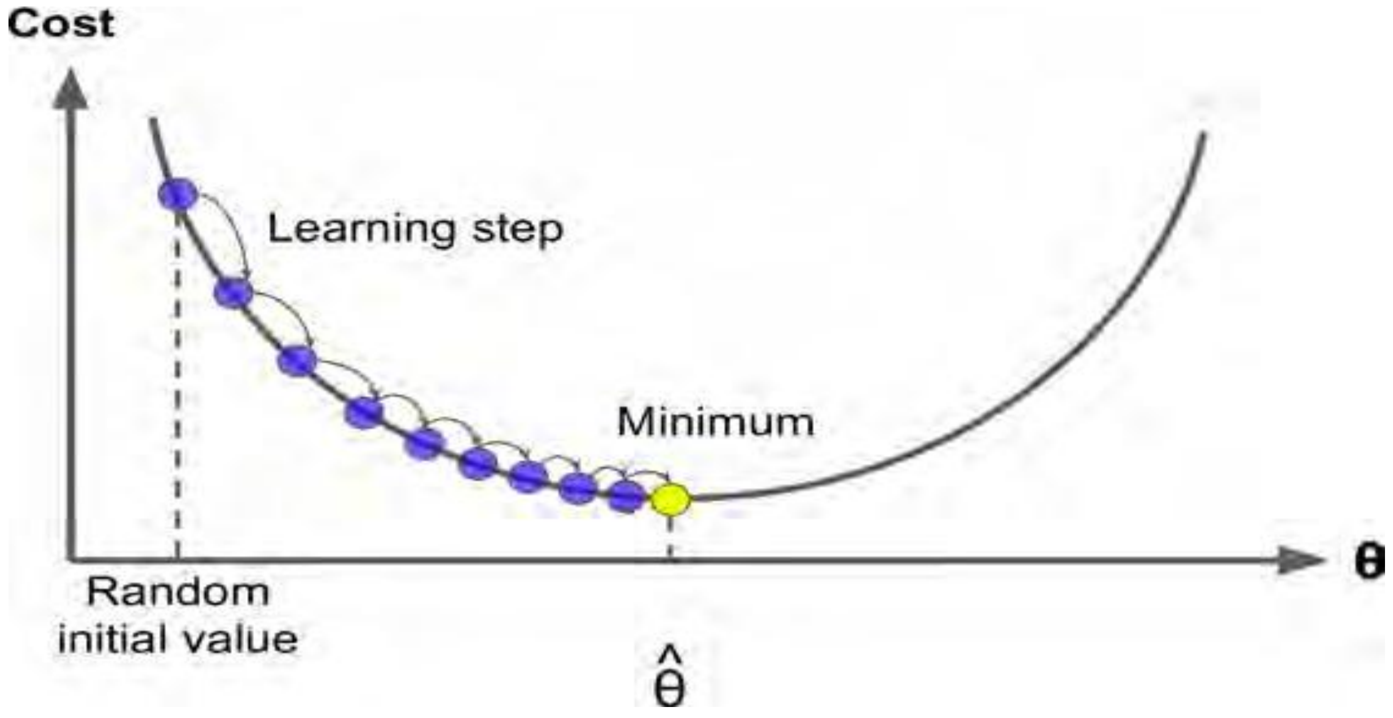
1. Main limitation of Linear Regression is the assumption of linearity between the dependent variable and the independent variables.
2. Prone to noise and overfitting
3. Prone to outliers
4. Prone to multicollinearity

Gradient Descent

- Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.
- Gradient Descent measures the local gradient of the error function with regards to the parameter vector θ, and it goes in the direction of descending gradient.
- Once the gradient is zero, you have reached a minimum.

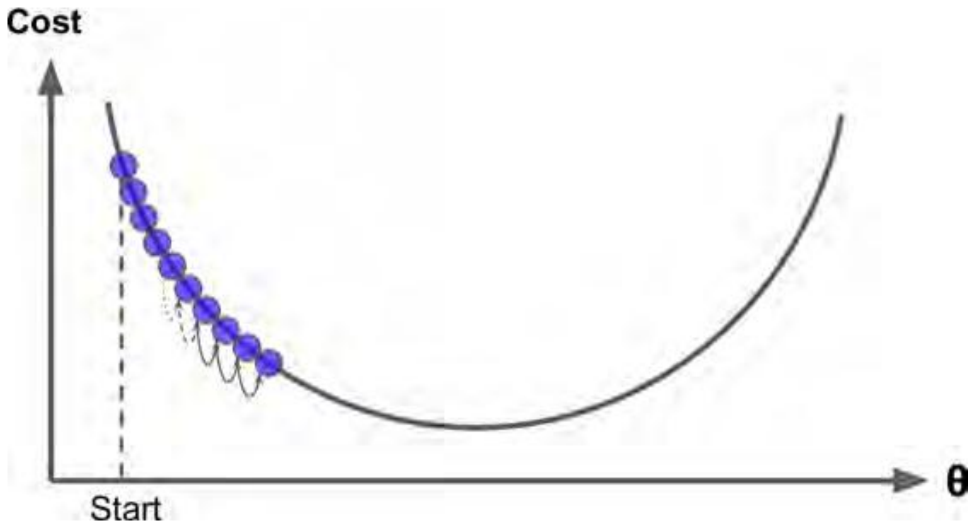
Gradient Descent

- Start by filling θ with random values, and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function until the algorithm converges to a minimum.



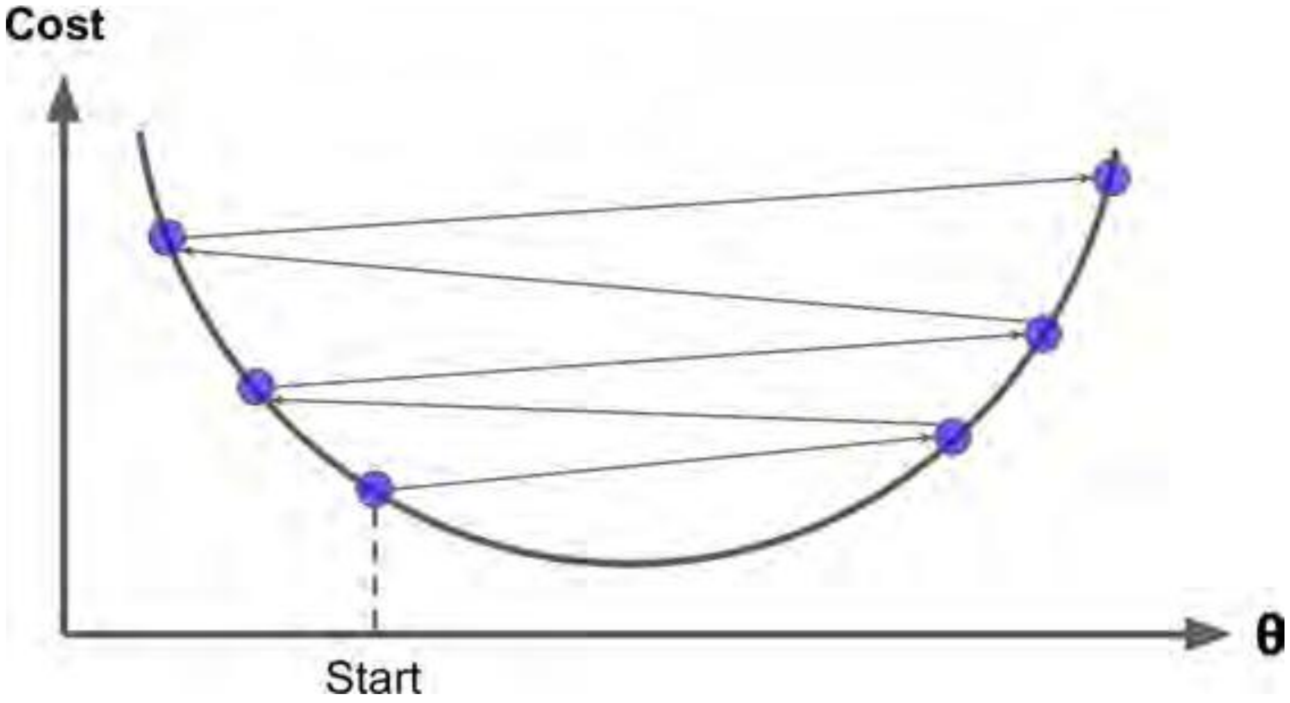
Gradient Descent

- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter.
- If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time.



Gradient Descent

- On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before.

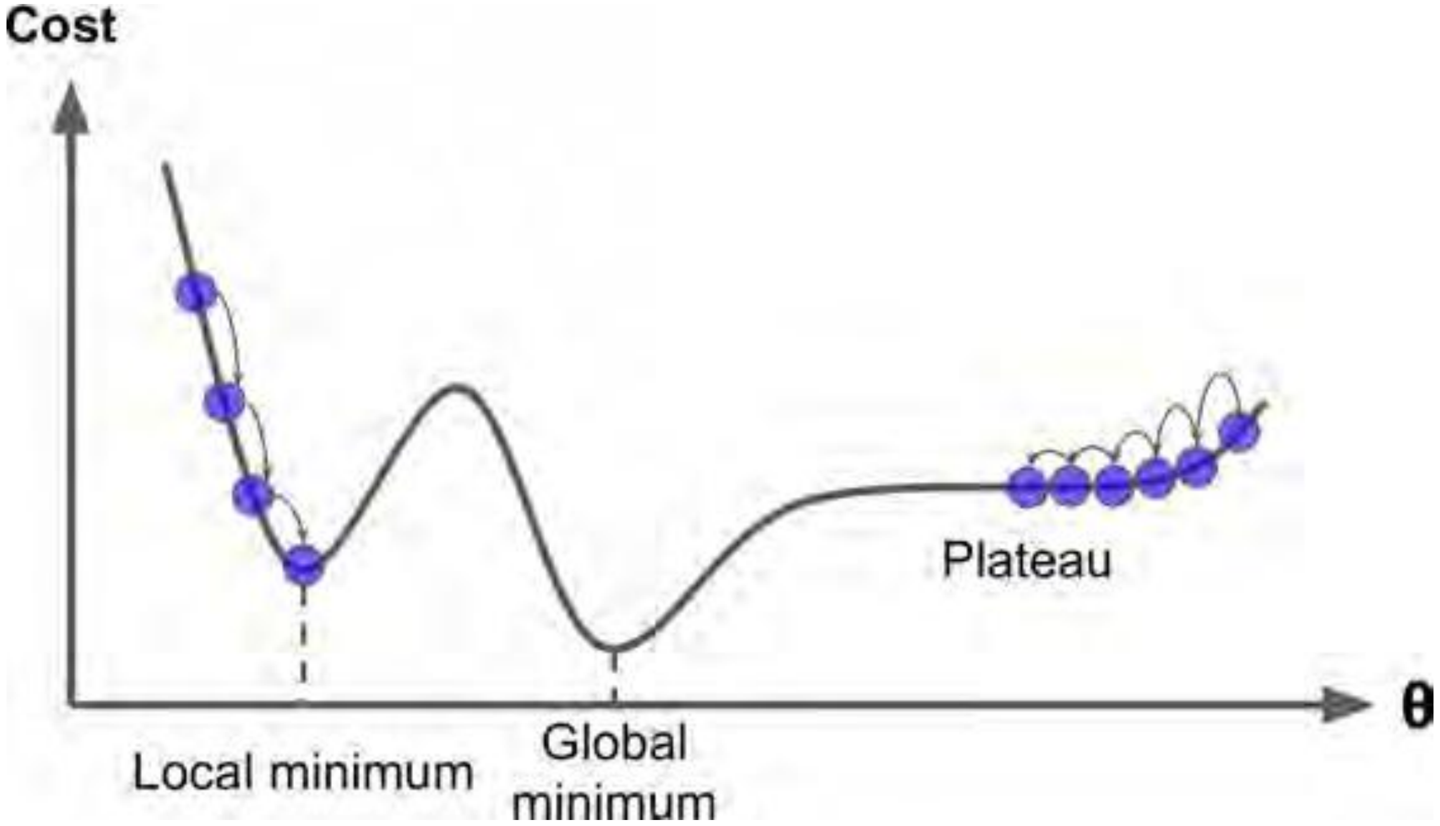


Gradient Descent

- Two main challenges with Gradient Descent:
- If the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum.
- If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.

Gradient Descent

- Two main challenges with Gradient Descent:

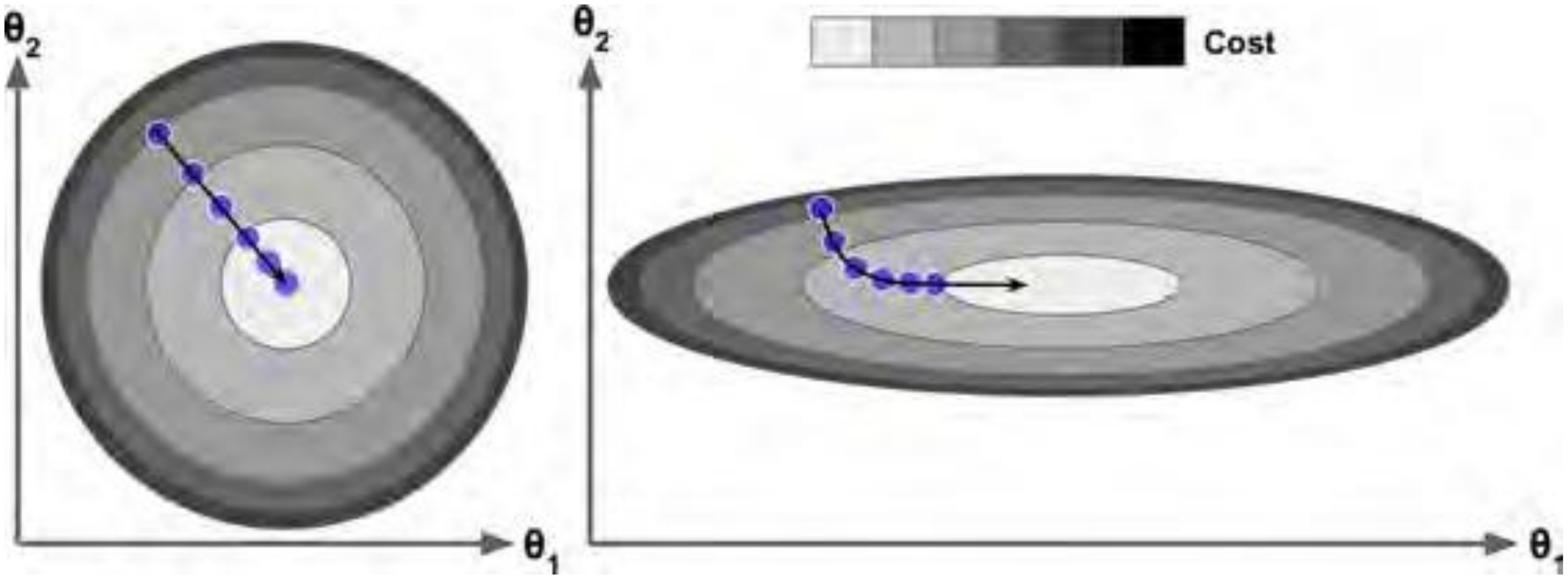


Gradient Descent

- The MSE cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve.
- This implies that there are no local minima, just one global minimum.
- It is also a continuous function with a slope that never changes abruptly.
- These two facts have a great consequence: Gradient Descent is guaranteed to approach arbitrarily close the global minimum.

Gradient Descent

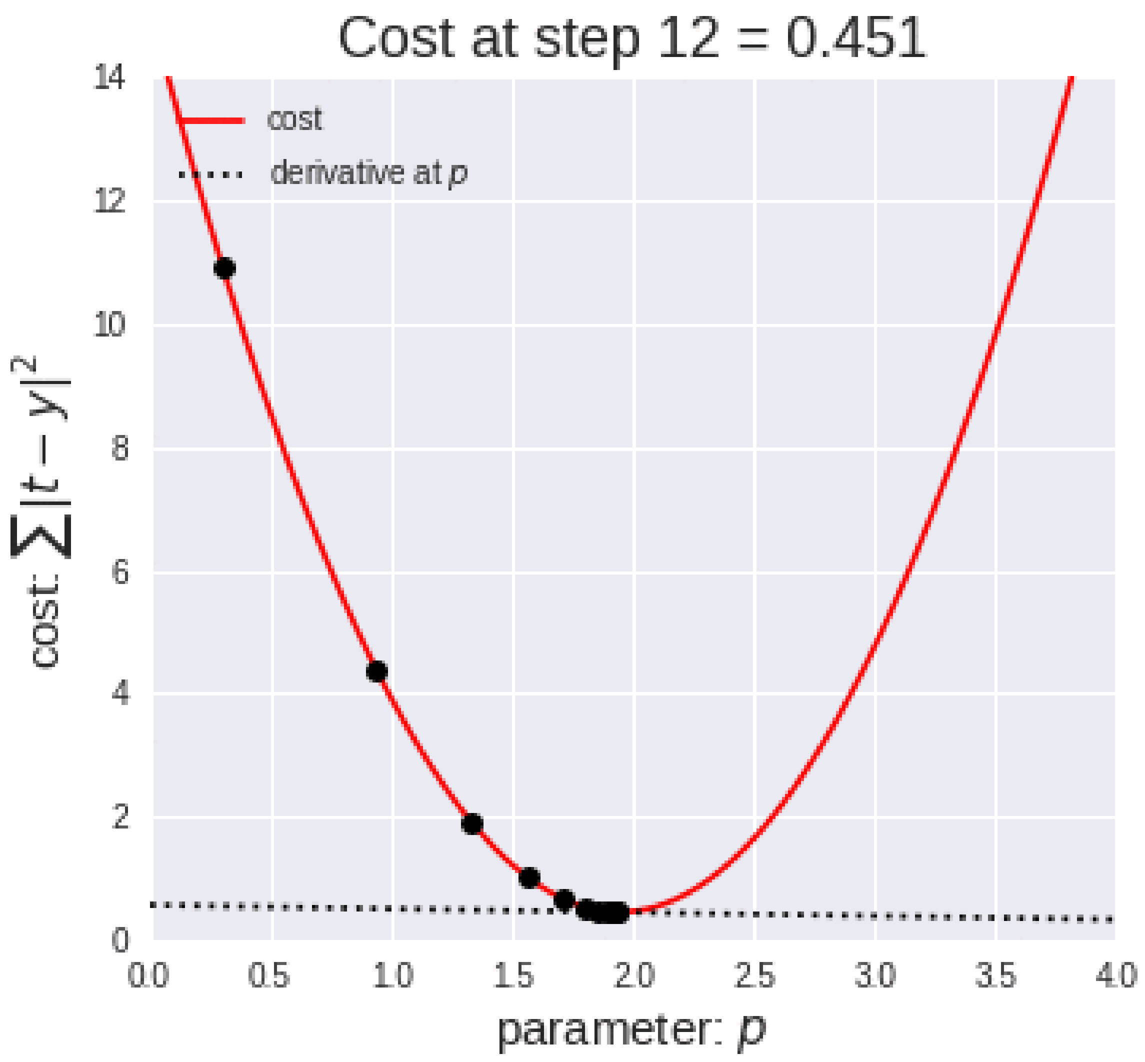
- Gradient Descent with and without feature scaling.



Gradient Descent

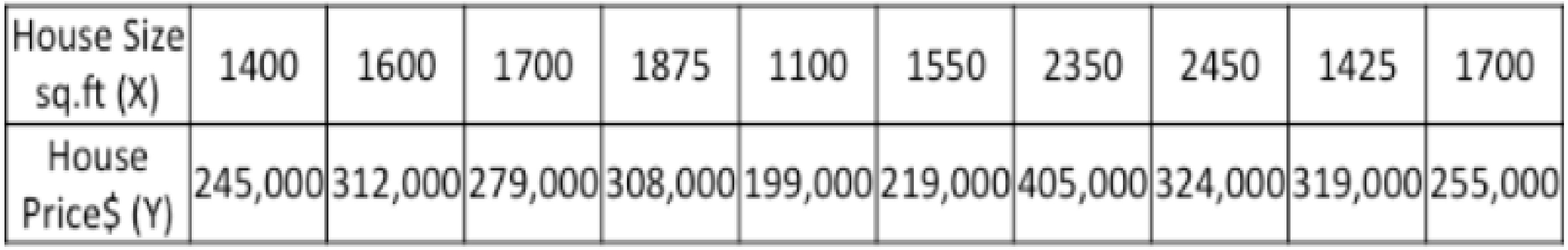
- Gradient Descent on a training set where features 1 and 2 have the same scale, and on a training set where feature 1 has much smaller values than feature 2.
- The left the Gradient Descent algorithm goes straight toward the minimum, thereby reaching it quickly.
- Whereas on the right it first goes in a direction almost orthogonal to the direction of the global minimum, and it ends with a long march down an almost flat valley.

Gradient Descent



Gradient Descent

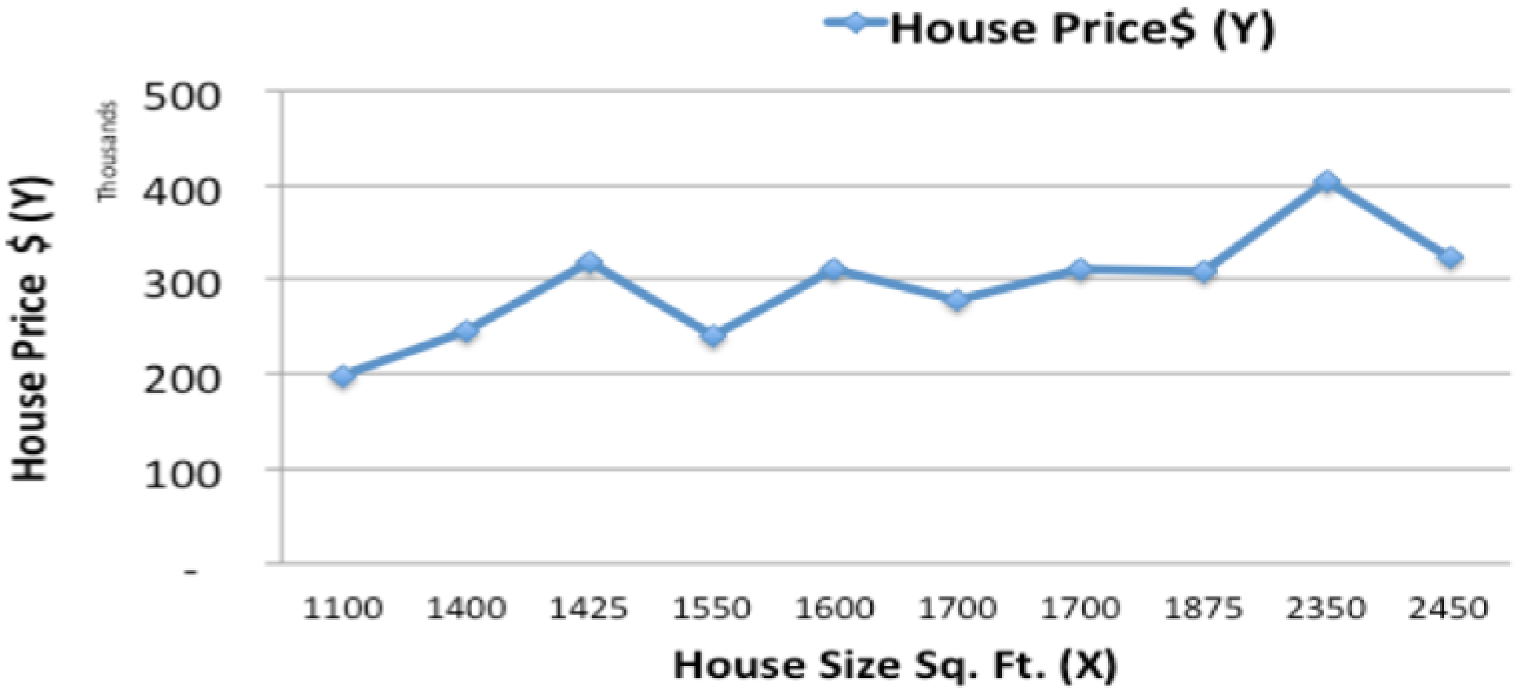
- Given historical housing data, the task is to create a model that predicts the price of a new house given the house size.



House Size sq.ft (X)	1400	1600	1700	1875	1100	1550	2350	2450	1425	1700
House Price$ (Y)	245,000	312,000	279,000	308,000	199,000	219,000	405,000	324,000	319,000	255,000

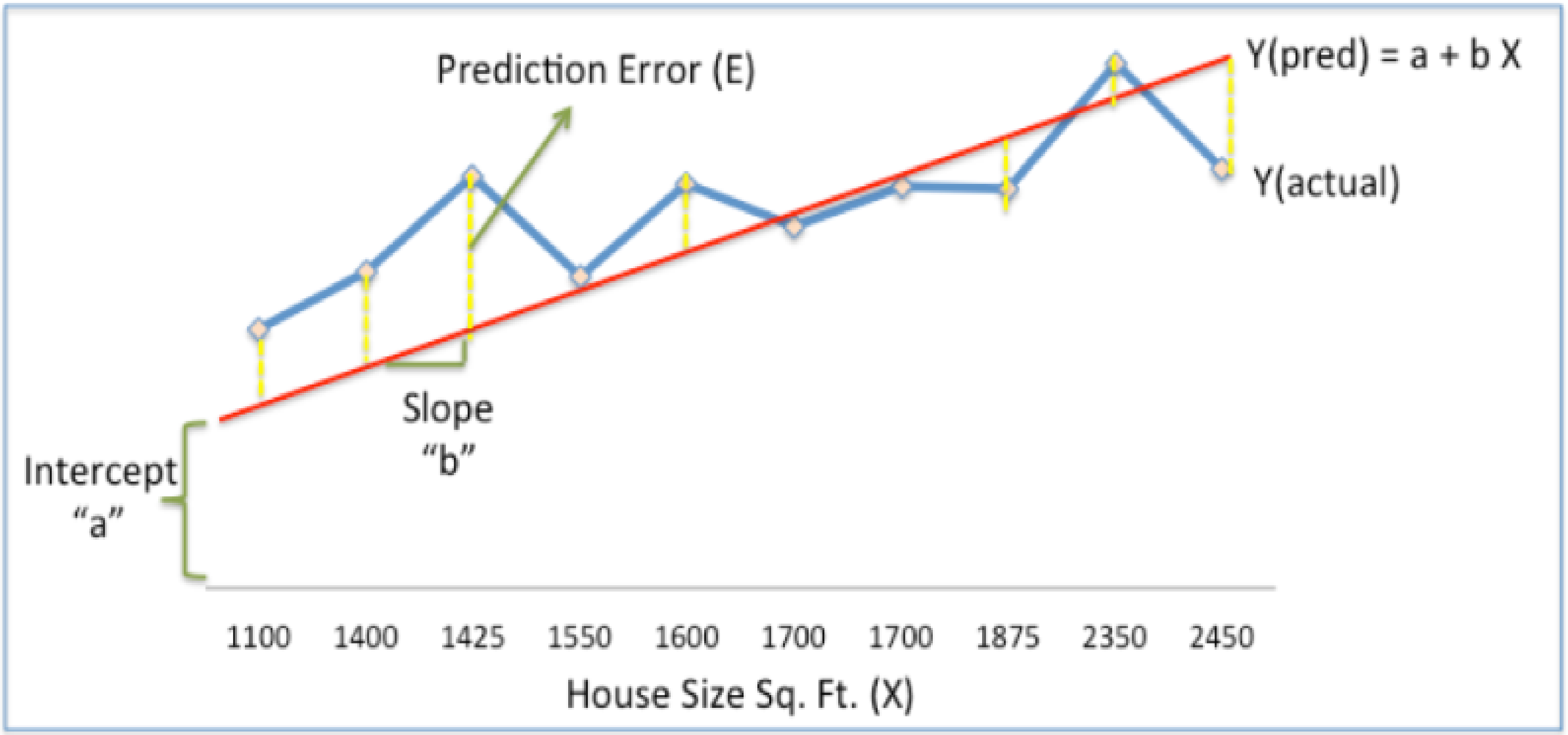
Gradient Descent

- The task – for a new house, given its size (X), what will its price (Y) be?
- Plotting the historical housing data



Gradient Descent

- Use a simple linear model, where we fit a line on the historical data, to predict the price of a new house (Ypred) given its size (X)



Gradient Descent

- Ypred = a+bX
- The blue line gives the actual house prices from historical data (Yactual)
- The difference between Yactual and Ypred (given by the yellow dashed lines) is the prediction error (E)
- So, we need to find a line with optimal values of a,b (called weights) that best fits the historical data by reducing the prediction error and improving prediction accuracy.

Gradient Descent

- **Sum of Squared Errors (SSE) = ½ Sum (Actual House Price – Predicted House Price)^2**
- = **½ Sum(Y – Ypred)^2**
- This is where Gradient Descent comes into the picture.
- Gradient descent is an optimization algorithm that finds the optimal weights (a,b) that reduces prediction error.

Gradient Descent

- **Gradient Descent Algorithm**
- Step 1: Initialize the weights(a & b) with random values and calculate Error (SSE)

- Step 2: Calculate the gradient i.e. change in SSE when the weights (a & b) are changed by a very small value from their original randomly initialized value.
- This helps us move the values of a & b in the direction in which SSE is minimized.

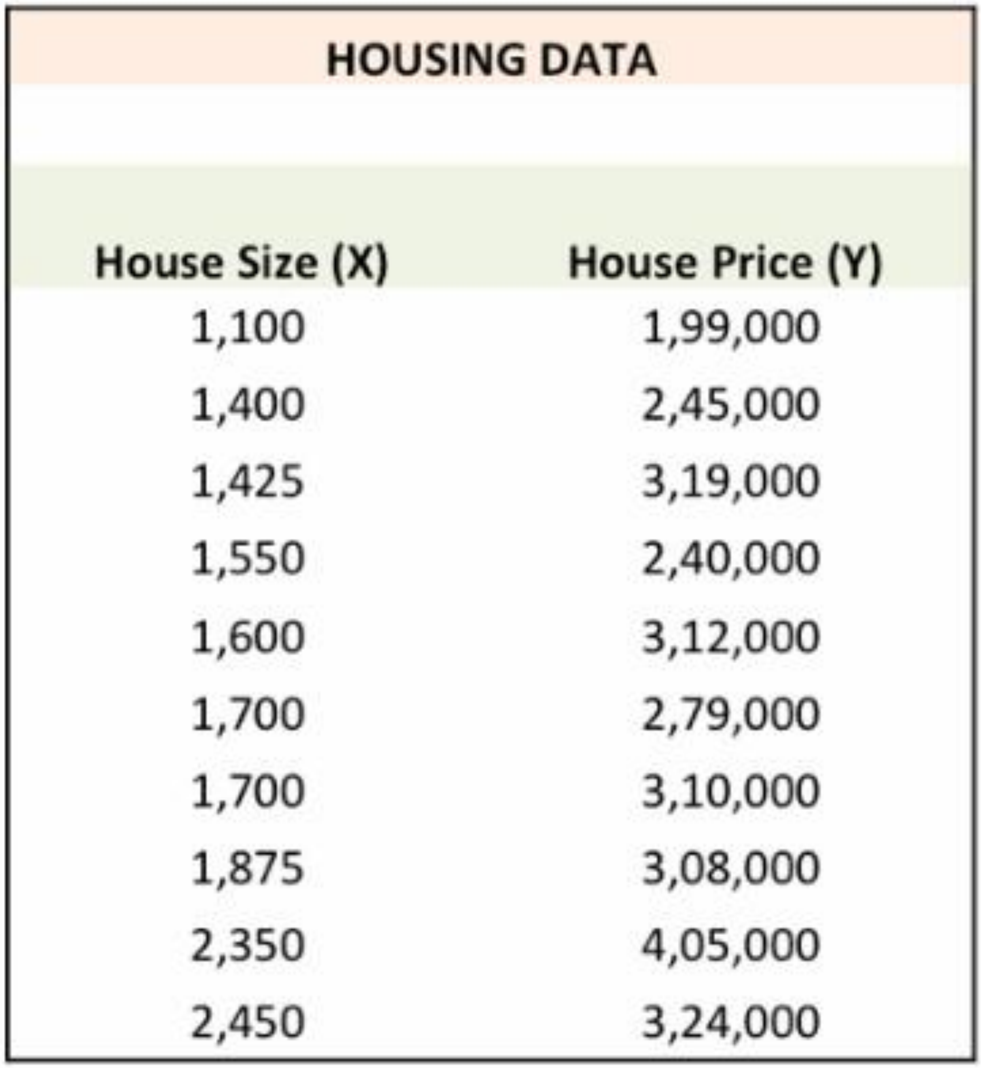
Gradient Descent

- Step 3: Adjust the weights with the gradients to reach the optimal values where SSE is minimized.

- Step 4: Use the new weights for prediction and to calculate the new SSE.

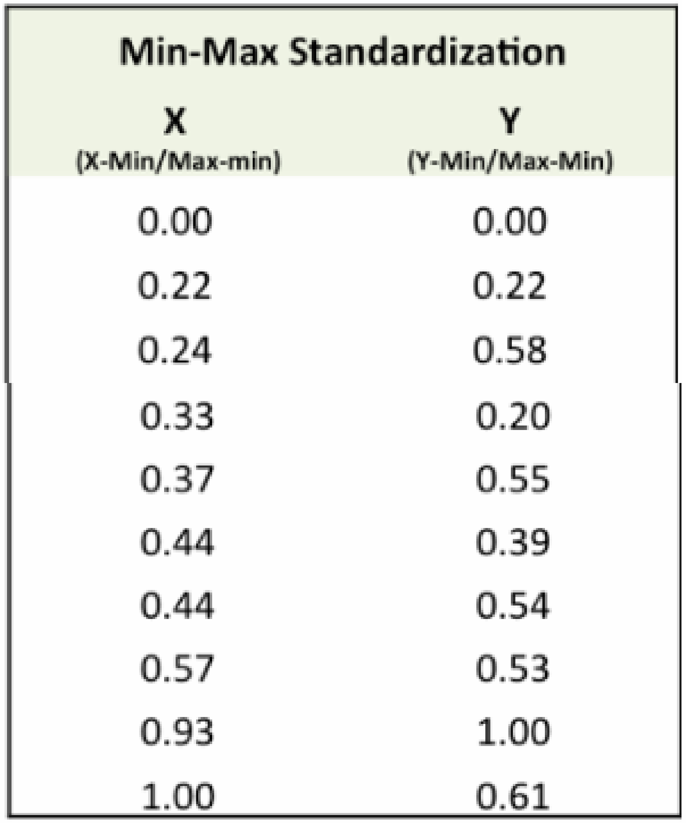
- Step 5: Repeat steps 2 and 3 till further adjustments to weights doesn’t significantly reduce the Error.

Gradient Descent



HOUSING DATA

House Size (X)	House Price (Y)
1,100	1,99,000
1,400	2,45,000
1,425	3,19,000
1,550	2,40,000
1,600	3,12,000
1,700	2,79,000
1,700	3,10,000
1,875	3,08,000
2,350	4,05,000
2,450	3,24,000

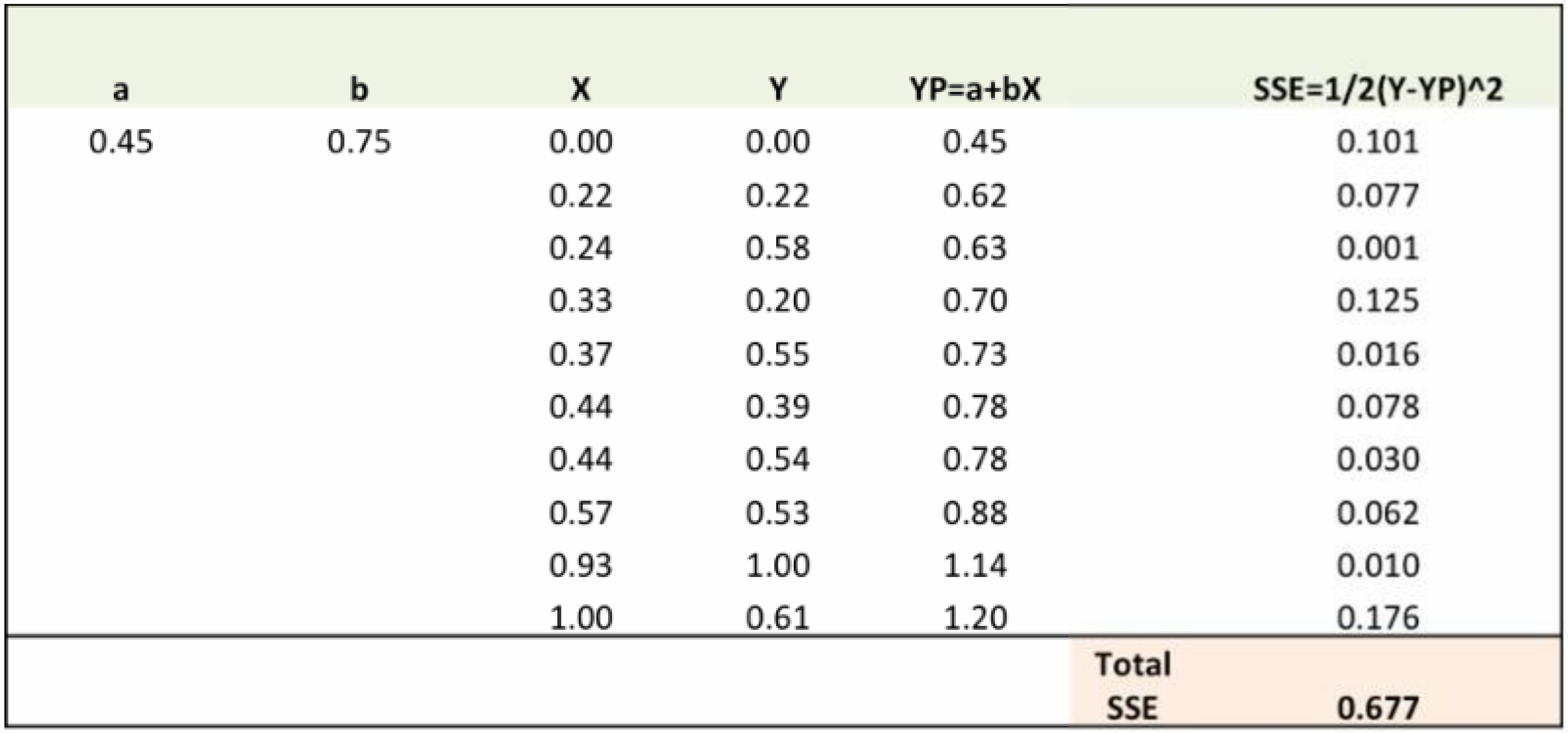


Min-Max Standardization

X (X-Min/Max-min)	Y (Y-Min/Max-Min)
0.00	0.00
0.22	0.22
0.24	0.58
0.33	0.20
0.37	0.55
0.44	0.39
0.44	0.54
0.57	0.53
0.93	1.00
1.00	0.61

Gradient Descent

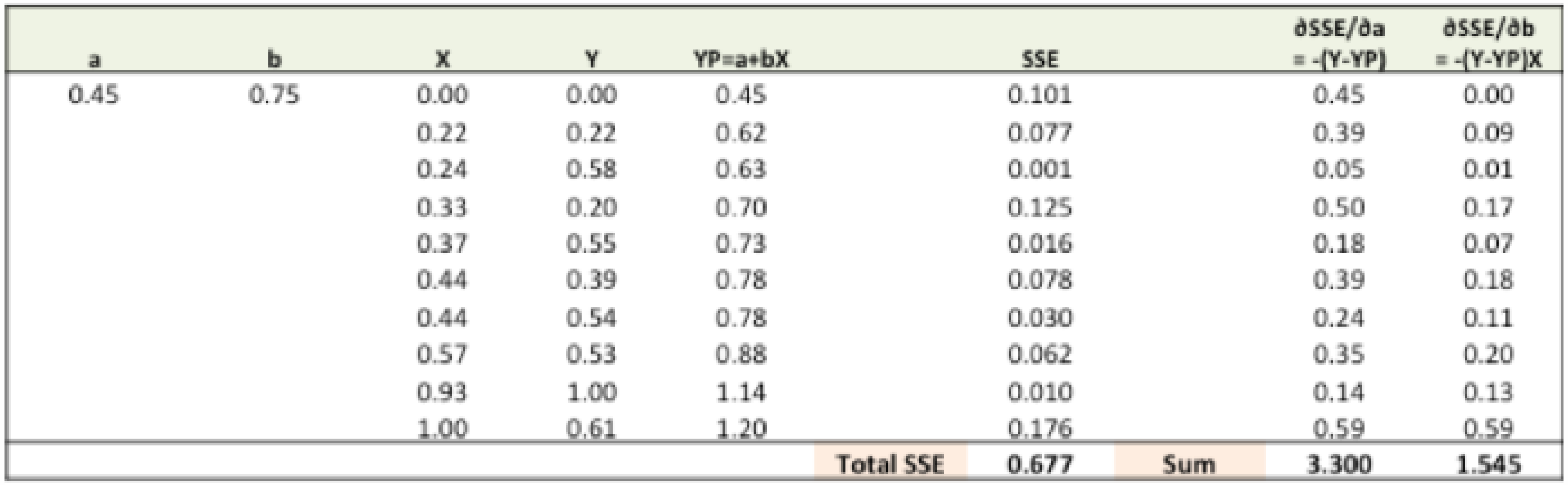
- Step 1: To fit a line Ypred = a + b X, start off with random values of a and b and calculate prediction error (SSE)
- SSE = ½(Y-YP)^2



a	b	X	Y	YP=a+bX		SSE=1/2(Y-YP)^2
0.45	0.75	0.00	0.00	0.45		0.101
		0.22	0.22	0.62		0.077
		0.24	0.58	0.63		0.001
		0.33	0.20	0.70		0.125
		0.37	0.55	0.73		0.016
		0.44	0.39	0.78		0.078
		0.44	0.54	0.78		0.030
		0.57	0.53	0.88		0.062
		0.93	1.00	1.14		0.010
		1.00	0.61	1.20		0.176
					Total SSE	**0.677**

Gradient Descent

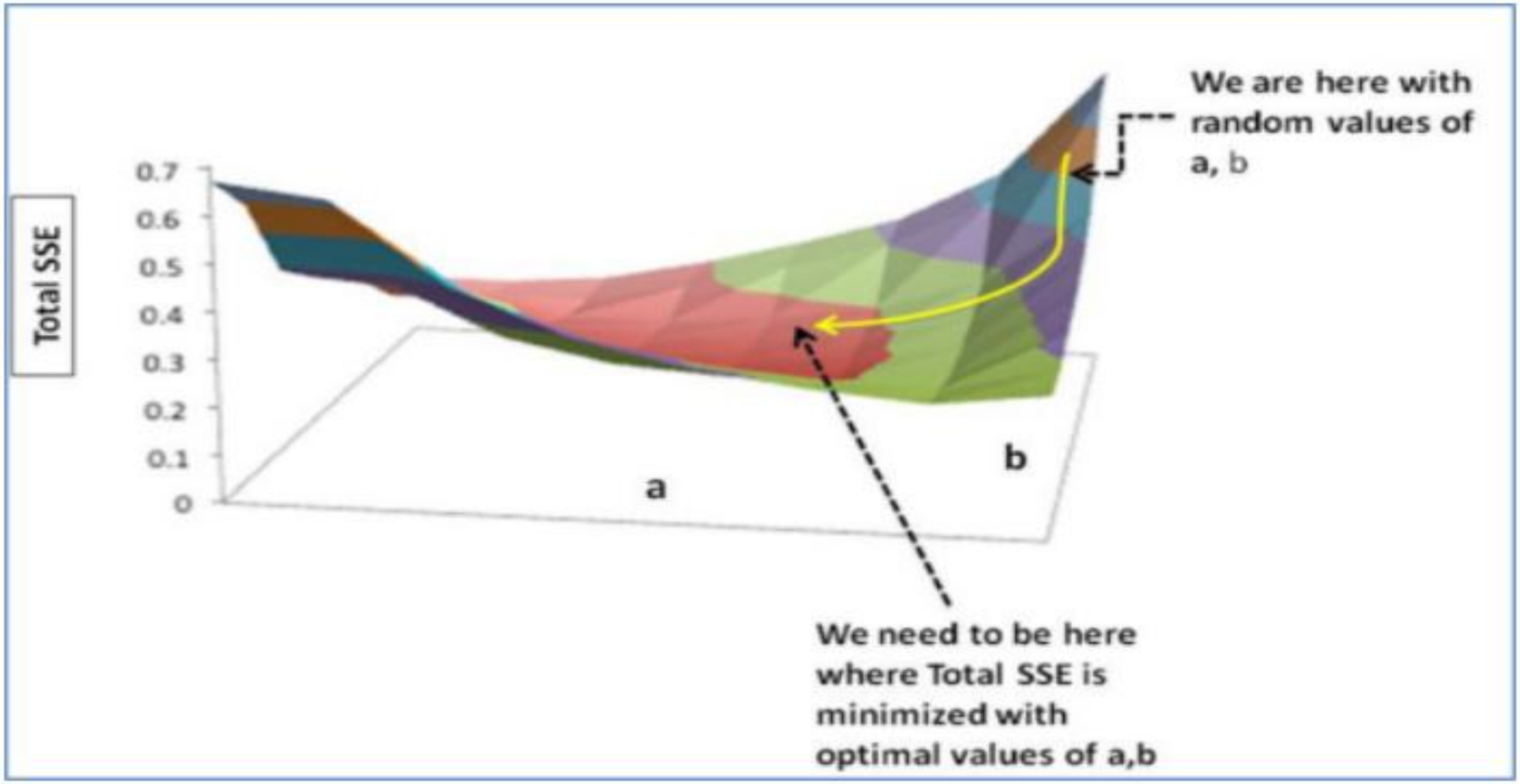
- Step 2: Calculate the error gradient w.r.t the weights
- $\partial SSE/\partial a = -(Y\text{-}YP)$
- $\partial SSE/\partial b = -(Y\text{-}YP)X$
- Here, $SSE=\frac{1}{2}(Y\text{-}YP)^2 = \frac{1}{2}(Y-(a+bX))^2$



a	b	X	Y	YP=a+bX		SSE		∂SSE/∂a = -(Y-YP)	∂SSE/∂b = -(Y-YP)X
0.45	0.75	0.00	0.00	0.45		0.101		0.45	0.00
		0.22	0.22	0.62		0.077		0.39	0.09
		0.24	0.58	0.63		0.001		0.05	0.01
		0.33	0.20	0.70		0.125		0.50	0.17
		0.37	0.55	0.73		0.016		0.18	0.07
		0.44	0.39	0.78		0.078		0.39	0.18
		0.44	0.54	0.78		0.030		0.24	0.11
		0.57	0.53	0.88		0.062		0.35	0.20
		0.93	1.00	1.14		0.010		0.14	0.13
		1.00	0.61	1.20		0.176		0.59	0.59
					Total SSE	0.677	Sum	3.300	1.545

Gradient Descent

- Step 3:Adjust the weights with the gradients to reach the optimal values where SSE is minimized.

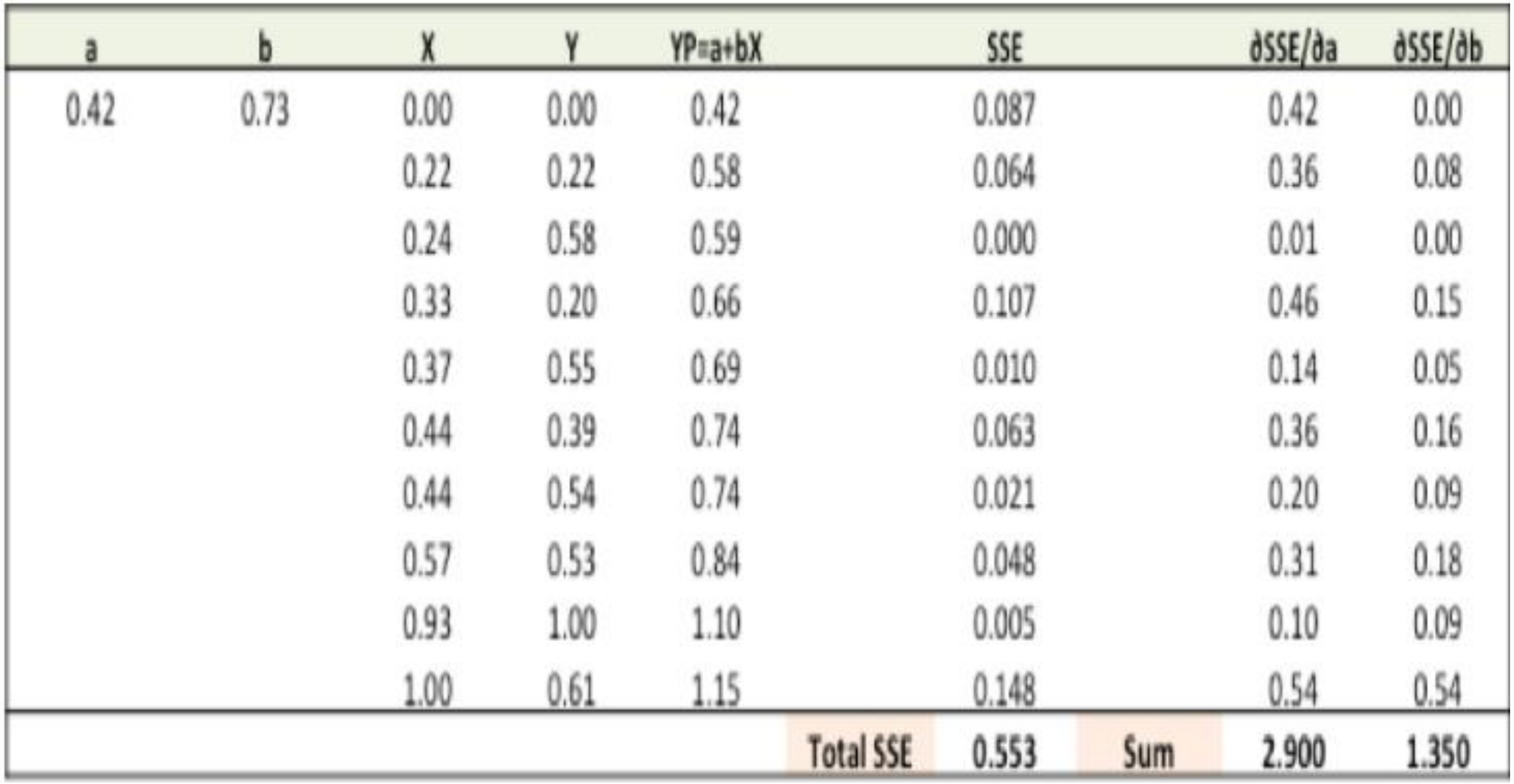


Gradient Descent

- We need to update the random values of a, b so that we move in the direction of optimal a, b.
- Update rules:
- a – ∂SSE/∂a
- b – ∂SSE/∂b
- So, update rules:
- New a = a – r * ∂SSE/∂a = 0.45-0.01*3.300 = 0.42
- New b = b – r * ∂SSE/∂b= 0.75-0.01*1.545 = 0.73
- here, r is the learning rate = 0.01, which is the pace of adjustment to the weights.

Gradient Descent

- Step 4: Use new a and b for prediction and to calculate new Total SSE.



a	b	X	Y	YP=a+bX		SSE		∂SSE/∂a	∂SSE/∂b
0.42	0.73	0.00	0.00	0.42		0.087		0.42	0.00
		0.22	0.22	0.58		0.064		0.36	0.08
		0.24	0.58	0.59		0.000		0.01	0.00
		0.33	0.20	0.66		0.107		0.46	0.15
		0.37	0.55	0.69		0.010		0.14	0.05
		0.44	0.39	0.74		0.063		0.36	0.16
		0.44	0.54	0.74		0.021		0.20	0.09
		0.57	0.53	0.84		0.048		0.31	0.18
		0.93	1.00	1.10		0.005		0.10	0.09
		1.00	0.61	1.15		0.148		0.54	0.54
					Total SSE	0.553	Sum	2.900	1.350

Gradient Descent

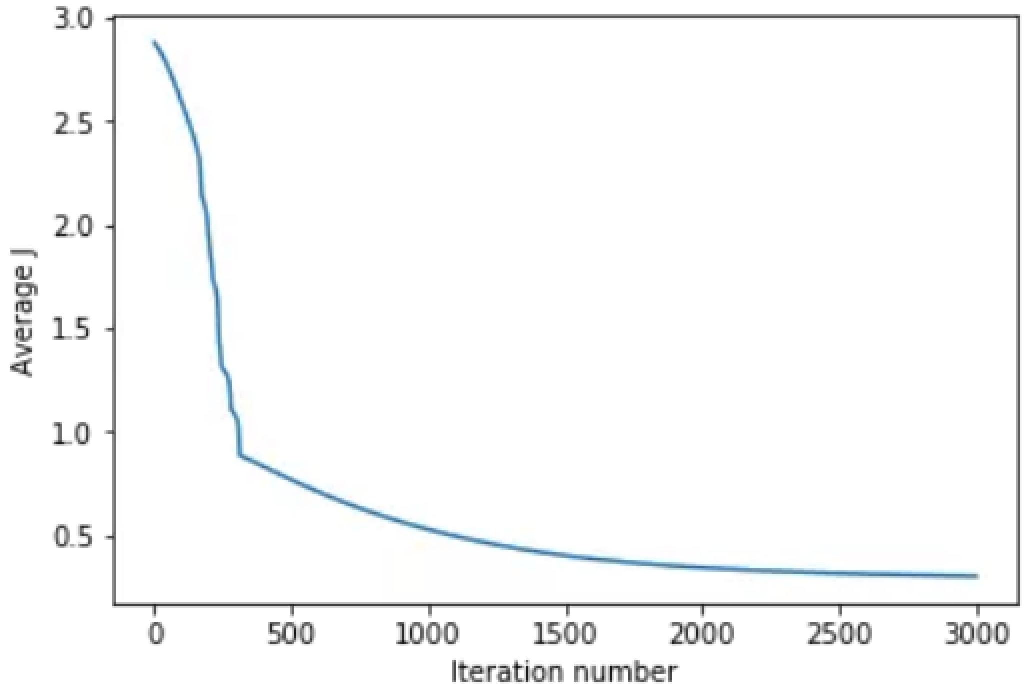
- Step 5: Repeat step 3 and 4 till the time further adjustments to a, b doesn’t significantly reduces the error.
- At that time, we have arrived at the optimal a, b with the highest prediction accuracy.
- This is the Gradient Descent Algorithm.
- This optimization algorithm and its variants form the core of many machine learning algorithms like Neural Networks and even Deep Learning.
- https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html

Batch Gradient Descent

- In Batch Gradient Descent, all the training data is taken into consideration to take a single step.
- We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters.
- So that’s just one step of gradient descent in one epoch.
- Batch Gradient Descent is great for convex or relatively smooth error manifolds.
- In this case, we move somewhat directly towards an optimum solution.

Batch Gradient Descent

- The graph of cost vs epochs is also quite smooth because we are averaging over all the gradients of training data for a single step.
- Average value of the cost function vs number of iterations / epochs.



Batch Gradient Descent

- **Advantages**

1. Less oscillations and noisy steps taken towards the global minima of the loss function due to updating the parameters by computing the average of all the training samples rather than the value of a single sample.
2. It can benefit from the vectorization which increases the speed of processing all training samples together.
3. It produces a more stable gradient descent convergence and stable error gradient than stochastic gradient descent.
4. It is computationally efficient as all computer resources are not being used to process a single sample rather are being used for all training samples.

Batch Gradient Descent

- **Disadvantages**

1. It requires the loading of the whole dataset into memory, which can be problematic for big data sets.
2. Batch gradient descent can’t be efficiently parallelized - this is because each update in the weight parameters requires a mean calculation of the cost function over all the training samples.
3. The smooth nature of the reducing cost function tends to ensure that the neural network training will get stuck in local minimums, which makes it less likely that a global minimum of the cost function will be found.

Stochastic Gradient Descent

- In Batch Gradient Descent we were considering all the examples for every step of Gradient Descent.
- But what if our dataset is very huge.
- Deep learning models crave for data.
- The more the data the more chances of a model to be good.
- Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples.
- This does not seem an efficient way.
- To tackle this problem we have Stochastic Gradient Descent.

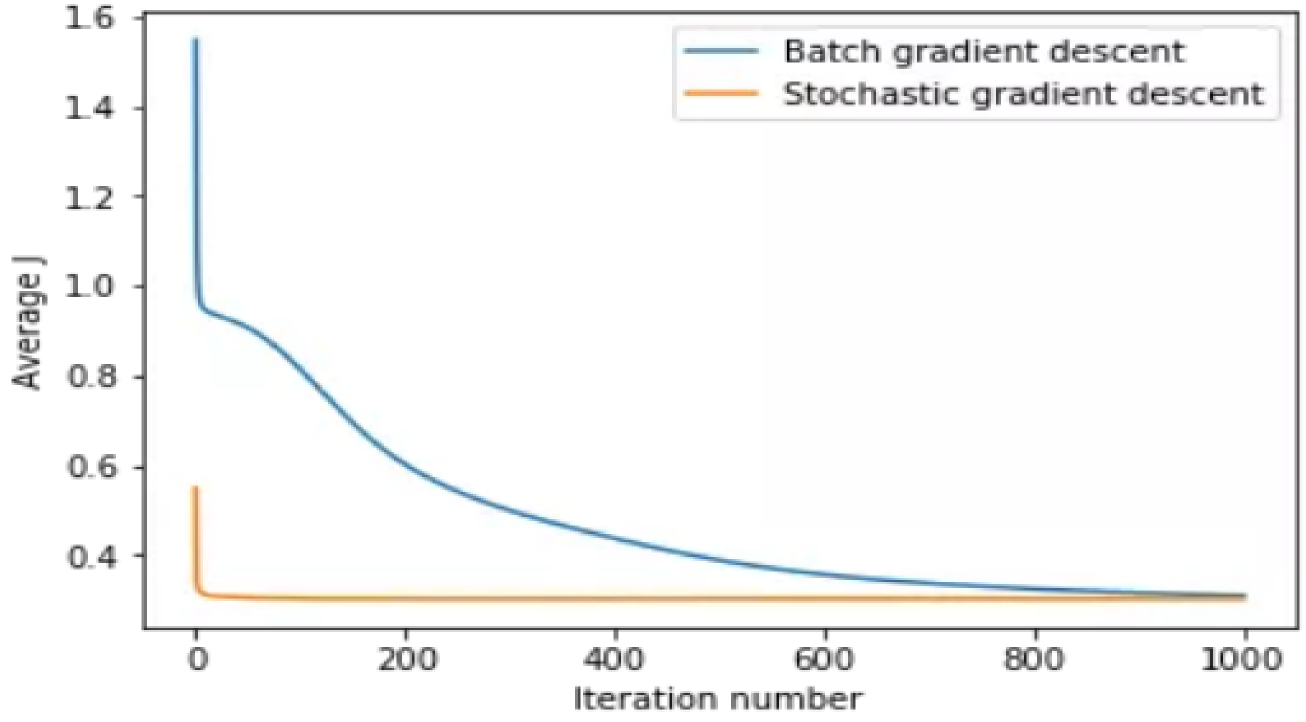
Stochastic Gradient Descent

- In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step.
- We do the following steps in one epoch for SGD:

1. Take an example
2. Feed it to Neural Network
3. Calculate it’s gradient
4. Use the gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the examples in training dataset.

Stochastic Gradient Descent

- Since we are considering just one example at a time the cost will fluctuate over the training examples and it will not necessarily decrease. But in the long run, you will see the cost decreasing with fluctuations.



Stochastic Gradient Descent

- **Advantages:**

1. It is easier to fit into memory due to a single training sample being processed by the network.
2. It is computationally fast as only one sample is processed at a time
3. For larger datasets it can converge faster as it causes updates to the parameters more frequently.
4. Due to frequent updates the steps taken towards the minima of the loss function have oscillations which can help getting out of local minimums of the loss function (in case the computed position turns out to be the local minimum).

Stochastic Gradient Descent

- **Disadvantages:**

1. Due to frequent updates the steps taken towards the minima are very noisy. This can often lead the gradient descent into other directions.
2. Also, due to noisy steps it may take longer to achieve convergence to the minima of the loss function.
3. Frequent updates are computationally expensive due to using all resources for processing one training sample at a time.
4. It loses the advantage of vectorized operations as it deals with only a single example at a time.

Mini Batch Gradient Descent

- Batch Gradient Descent can be used for smoother curves.
- SGD can be used when the dataset is large.
- Batch Gradient Descent converges directly to minima.
- SGD converges faster for larger datasets.
- But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it.
- This can slow down the computations.
- To tackle this problem, a mixture of Batch Gradient Descent and SGD is used.
- Neither we use all the dataset all at once nor we use the single example at a time.

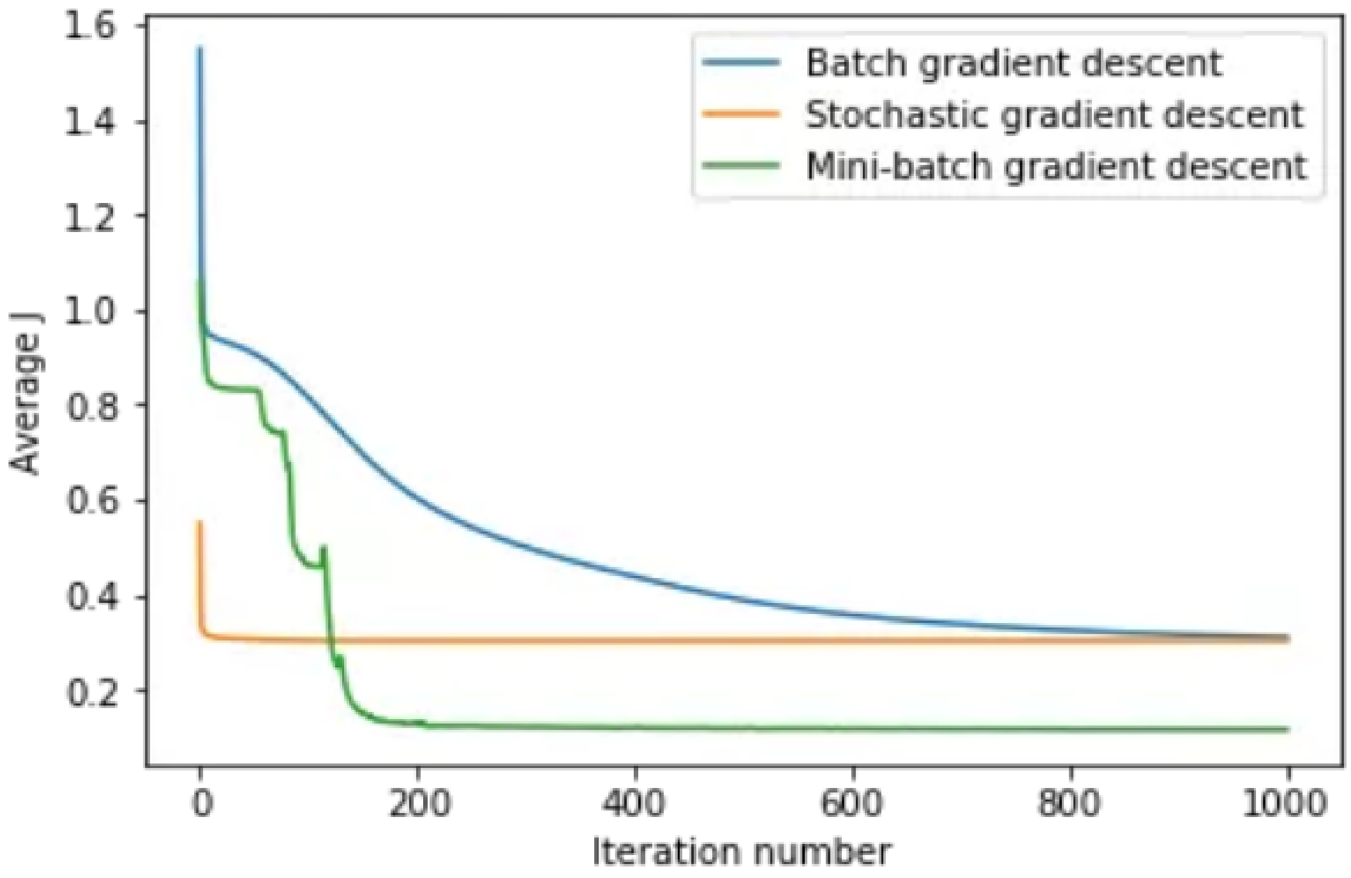
Mini Batch Gradient Descent

- We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch.
- Doing this helps us achieve the advantages of both the former variants we saw.
- So, after creating the mini-batches of fixed size, we do the following steps in one epoch:

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for the mini-batches we created

Mini Batch Gradient Descent

- Batch Gradient vs SGD vs Mini-batch gradient descent



Normal Equation

- Gradient descent is an algorithm which is used to reach an optimal solution iteratively using the gradient of the loss function or the cost function.
- In contrast, normal equation is a method that helps solve for the parameters analytically.
- Instead of reaching the solution iteratively, solution for the parameter θ is reached at directly by solving the normal equation.

Normal Equation

- Normal Equation:

$$\boxed{\theta = \left(X^T X\right)^{-1} . \left(X^T y\right)}$$

- In the above equation
  - θ : hypothesis parameters that define it the best.
  - X : Input feature value of each instance
  - Y : Output value of each instance.

- X is a m X (n+1) matrix that contains training instances.
- m is the number of training instances, and n is the number of features.
- Y is the vector of target values.

Normal Equation

- The computational complexity of inverting X^T . X , which is the (n+1) X (n+1) matrix, is pretty high when the number of features (i.e., n) is large.
- Also the training data need to fit in memory to compute the inversion.
- This means that it is hard to run on a parallel computing system.
- Iterative approaches would be better choice for a large number of features or many training instances.

Normal Equation

- Given the hypothesis function

$$h(\theta) = \theta_0 x_0 + \theta_1 x_1 + \dots \theta_n x_n$$

- where,
  n : the no. of features in the data set.
  x_0 : 1 (for vector multiplication)
- Notice that this is dot product between θ and x values. So for the convenience to solve we can write it as : $h(\theta) = \theta^T x$

Normal Equation

- The motive in Linear Regression is to minimize the cost function:

$$J(\Theta) = \frac{1}{2m} \sum_{i=1}^{m} \frac{1}{2}[h_\Theta(x^{(i)}) - y^{(i)}]^2$$

- where,
- xi : the input value of ith training example.
- m : no. of training instances
- n : no. of data-set features
- yi : the expected result of ith instance

Normal Equation

- Let us represent the cost function in a vector form

$$\begin{bmatrix} h_\theta\left(x^0\right) \\ h_\theta\left(x^1\right) \\ \cdots \\ h_\theta\left(x^m\right) \end{bmatrix} - \begin{bmatrix} y^0 \\ y^1 \\ \cdots \\ y^m \end{bmatrix}$$

Normal Equation

$$\begin{bmatrix} \theta^T\left(x^0\right) \\ \theta^T\left(x^1\right) \\ \dots \\ \theta^T\left(x^m\right) \end{bmatrix} - y$$

$$\begin{bmatrix} \theta_0 \begin{pmatrix} 0 \\ x_0 \end{pmatrix} + \theta_1 \begin{pmatrix} 0 \\ x_1 \end{pmatrix} + \dots.\ \theta_n \begin{pmatrix} 0 \\ x_n \end{pmatrix} \\ \theta_0 \begin{pmatrix} 1 \\ x_0 \end{pmatrix} + \theta_1 \begin{pmatrix} 1 \\ x_1 \end{pmatrix} + \dots.\ \theta_n \begin{pmatrix} 1 \\ x_n \end{pmatrix} \\ \dots \\ \theta_0 \begin{pmatrix} m \\ x_0 \end{pmatrix} + \theta_1 \begin{pmatrix} m \\ x_1 \end{pmatrix} + \dots.\ \theta_n \begin{pmatrix} m \\ x_n \end{pmatrix} \end{bmatrix} - y$$

Normal Equation

- This can further be reduced to $X\theta - y$
- But each residual value is squared.
- We cannot simply square the above expression.
- As the square of a vector/matrix is not equal to the square of each of its values.
- So to get the squared value, multiply the vector/matrix with its transpose.
- So, the final equation derived is $(X\theta - y)^T (X\theta - y)$

- Therefore, the cost function is $Cost = (X\theta - y)^T (X\theta - y)$

Normal Equation

- Final derived Normal Equation with θ giving the minimum cost value

$$\boxed{\theta = \left(X^T X\right)^{-1} \cdot \left(X^T y\right)}$$

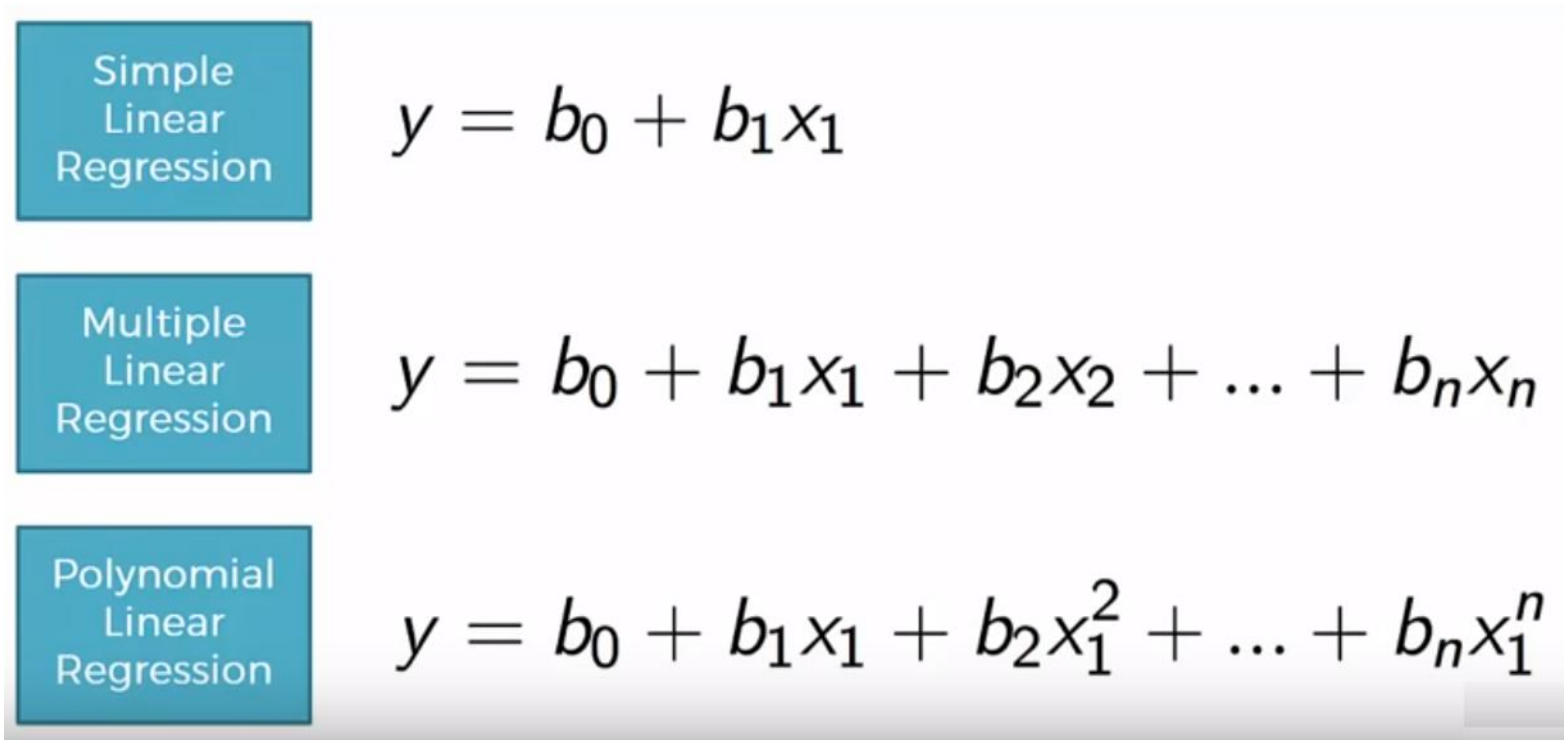
- https://adlersantos.com/articles/15/deriving-normal-equation

- https://www.geeksforgeeks.org/ml-normal-equation-in-linear-regression/

Polynomial Regression

- Used when the data is more complex than a simple straight line.
- Use a linear model to fit nonlinear data.
- A simple way to do this is to add powers of each feature as new features, then train a linear model on this extended set of features.
- This technique is called Polynomial Regression.

Polynomial Regression



Simple Linear Regression

$$y = b_0 + b_1 x_1$$

Multiple Linear Regression

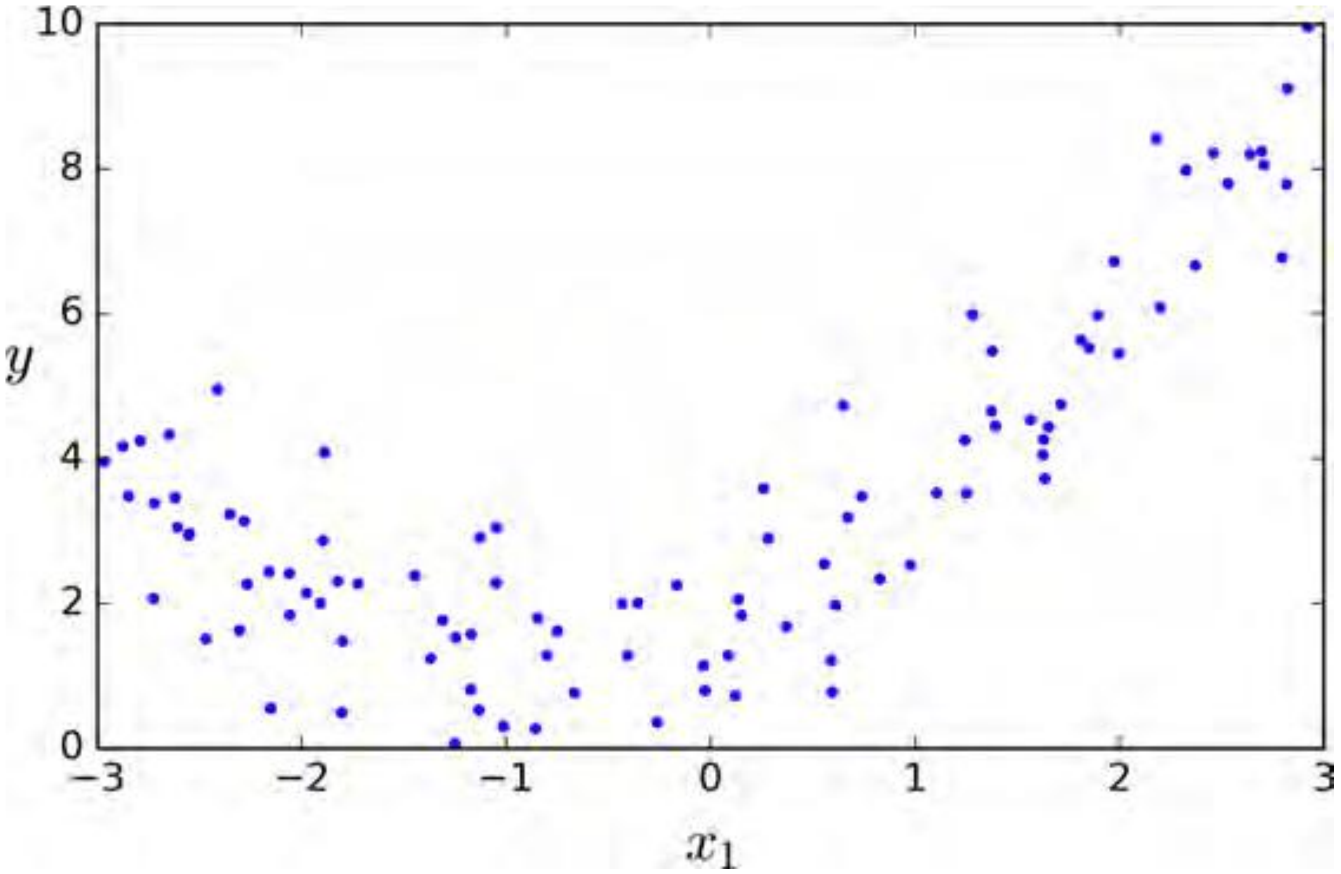
$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

Polynomial Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

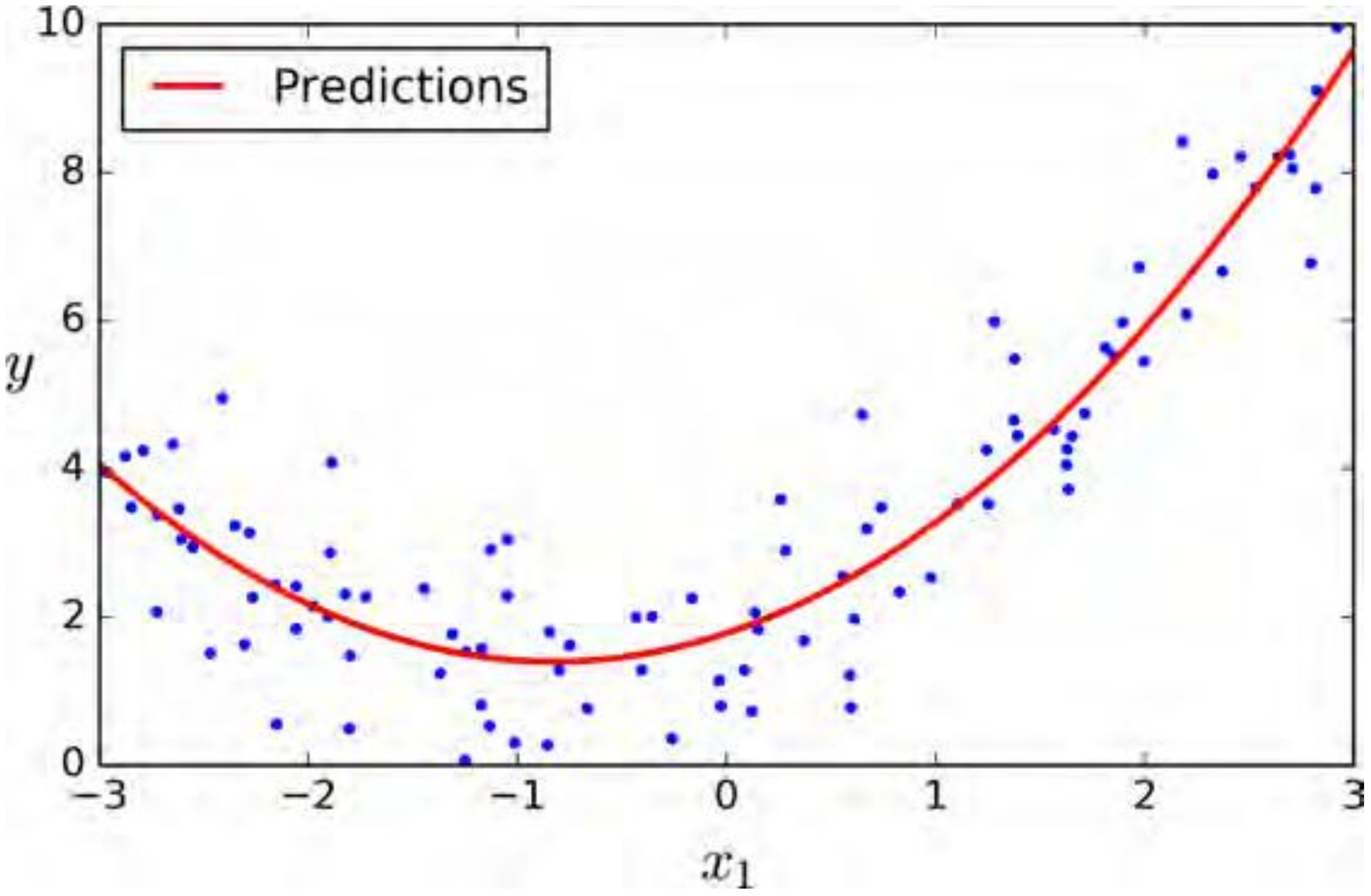
Polynomial Regression

- Generated nonlinear noisy dataset.



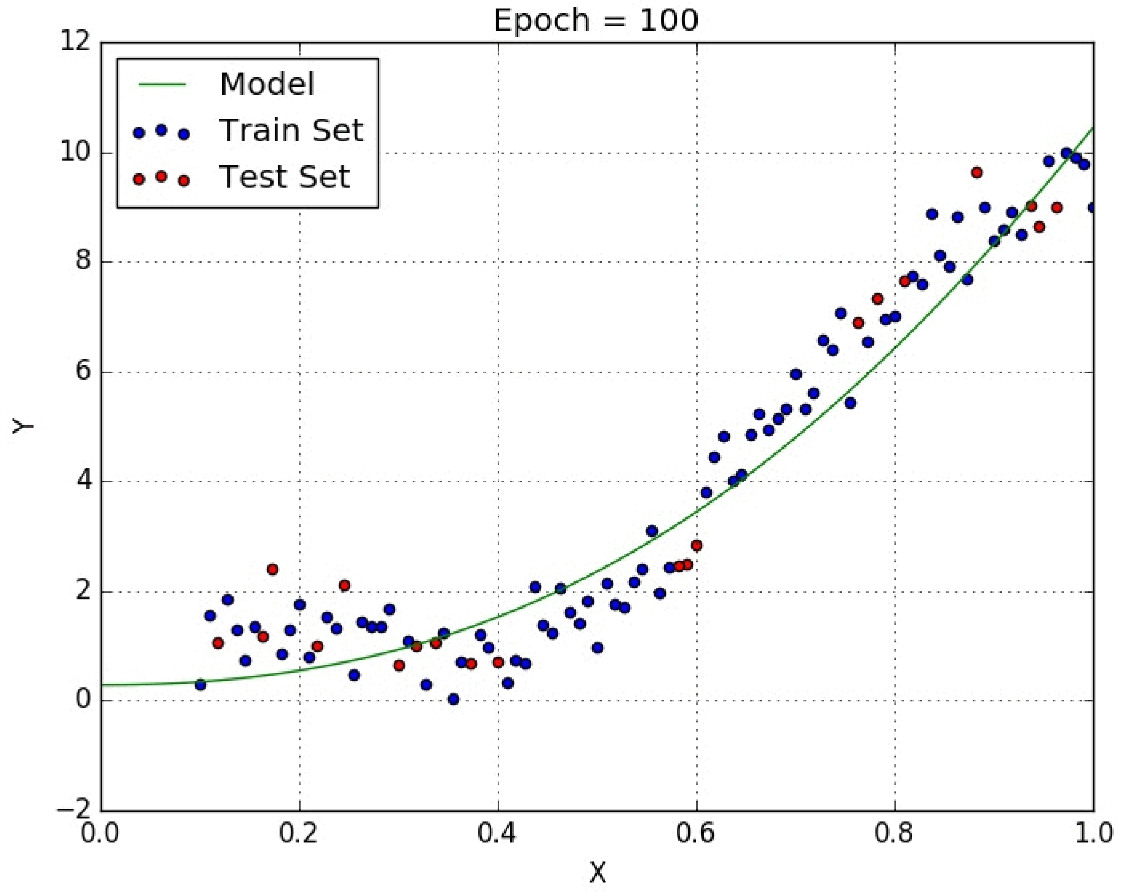
Polynomial Regression

- Polynomial Regression model prediction.



Polynomial Regression

- Polynomial Regression model prediction.



Polynomial Regression

- Polynomial Regression is capable of finding relationships between multiple features.
- Plain Linear Regression model cannot find the relationships between multiple features.
- This is made possible by the fact that Polynomial Features also adds all combinations of features up to the given degree.
- Polynomial regression is a special case of linear regression.
- It’s based on the idea of how to select your features.

Polynomial Regression

- Looking at the multivariate regression with 2 variables: x1 and x2.
- Linear regression will look like this:
- **y = a1 * x1 + a2 * x2.**
- Now you want to have a polynomial regression (let’s make 2-degree polynomial).
- We will create a few additional features: x1*x2, x1^2 and x2^2. So we will get your ‘linear regression’:
- **y = a1 * x1 + a2 * x2 + a3 * x1*x2 + a4 * x1^2 + a5 * x2^2**

Polynomial Regression

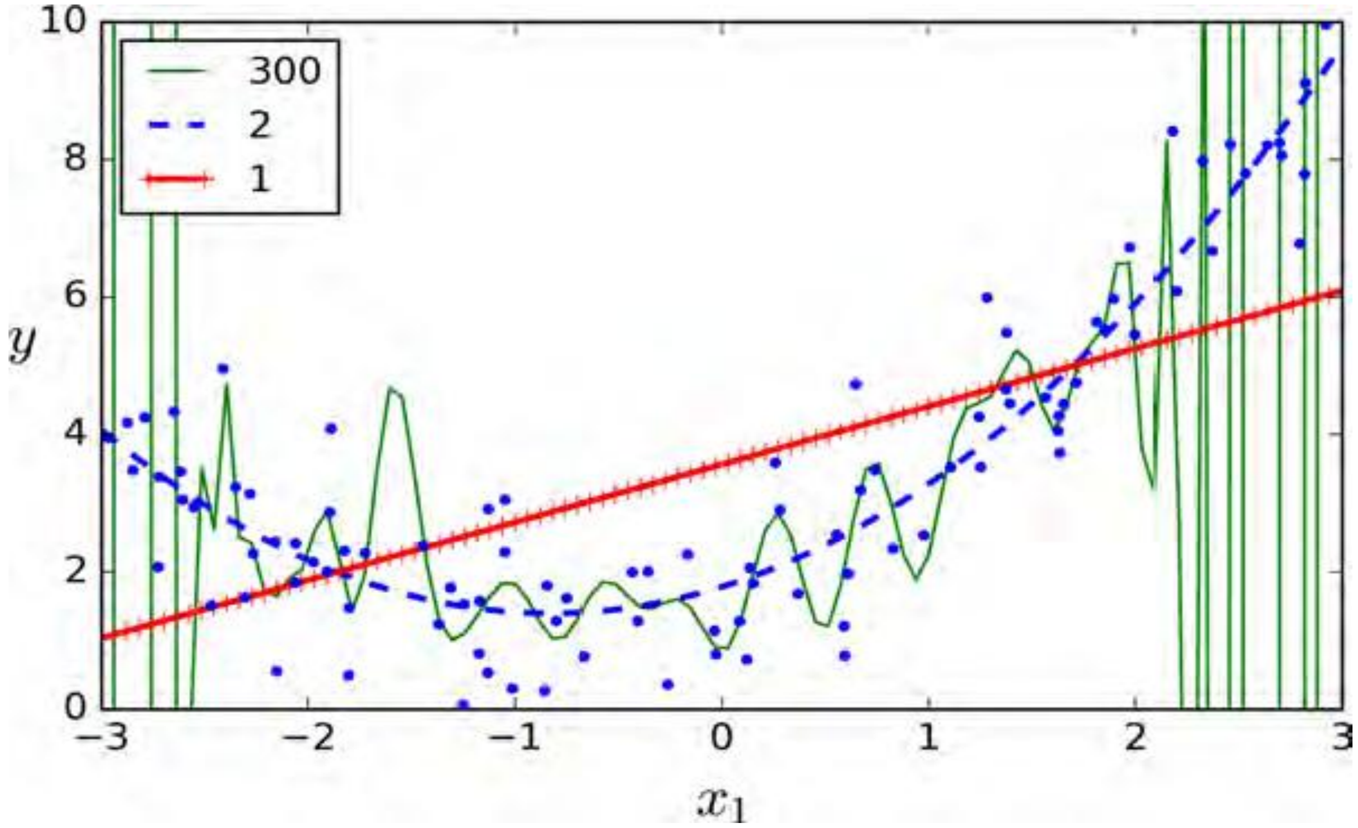
- **Advantages:**
- Polynomial provides the best approximation of the relationship between the dependent and independent variable.
- A Broad range of function can be fit under it.
- Polynomial basically fits a wide range of curvature.

Polynomial Regression

- **Disadvantages:**
- The presence of one or two outliers in the data can seriously affect the results of the nonlinear analysis.
- These are too sensitive to the outliers.
- In addition, there are unfortunately fewer model validation tools for the detection of outliers in nonlinear regression than there are for linear regression.

Polynomial Regression

- **Learning Curves**
- High-degree Polynomial Regression



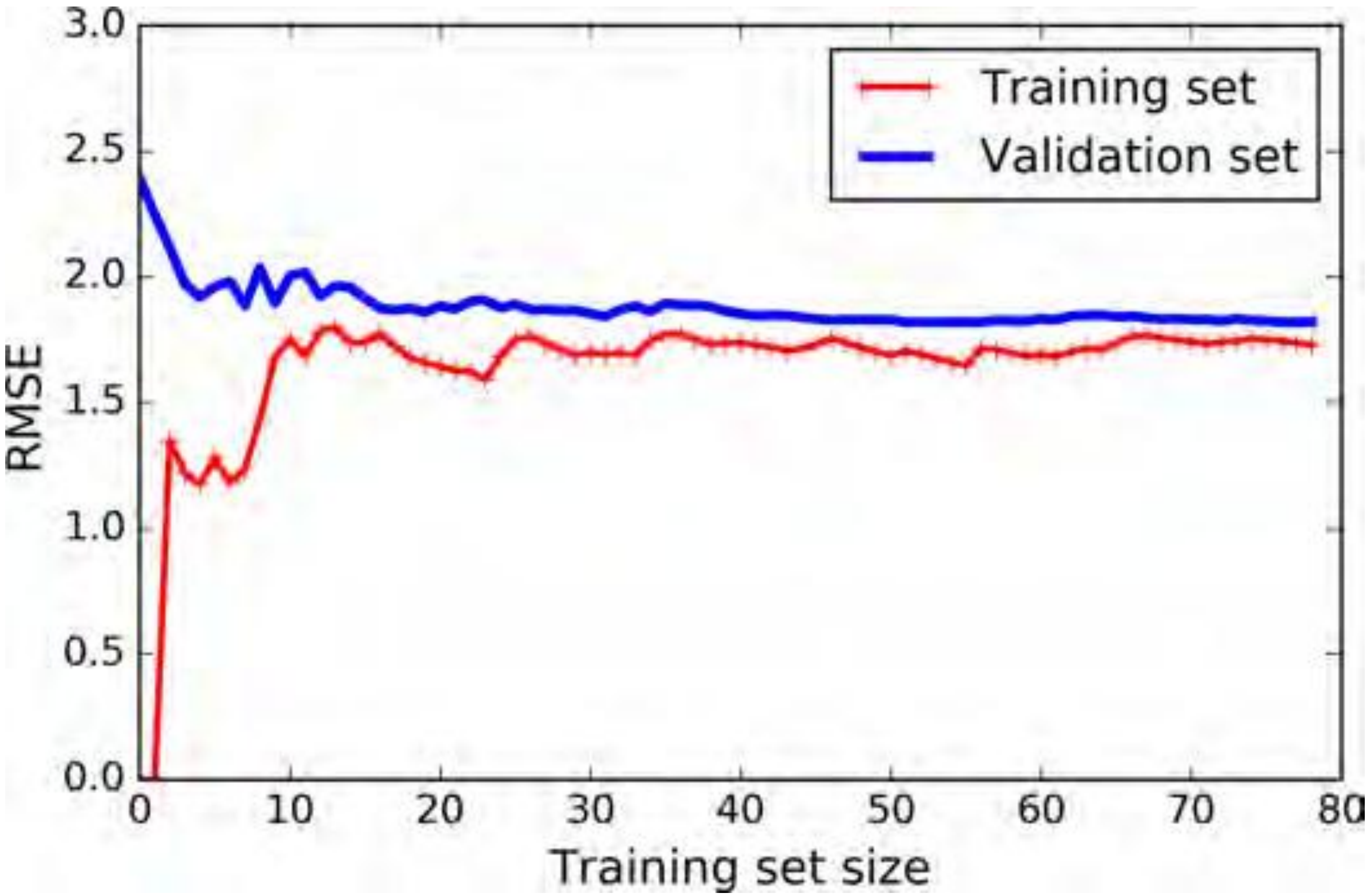
Polynomial Regression

- **Learning Curves**
- High-degree Polynomial Regression model is severely over fitting the training data,
- Linear model is under fitting the training data.
- The model that will generalize best in this case is the quadratic model.

Polynomial Regression

- Cross-validation to get an estimate of a model’s generalization performance.
- If a model performs well on the training data but generalizes poorly according to the cross-validation metrics, then your model is overfitting.
- If it performs poorly on both, then it is underfitting.
- This is one way to tell when a model is too simple or too complex.

Polynomial Regression



Polynomial Regression

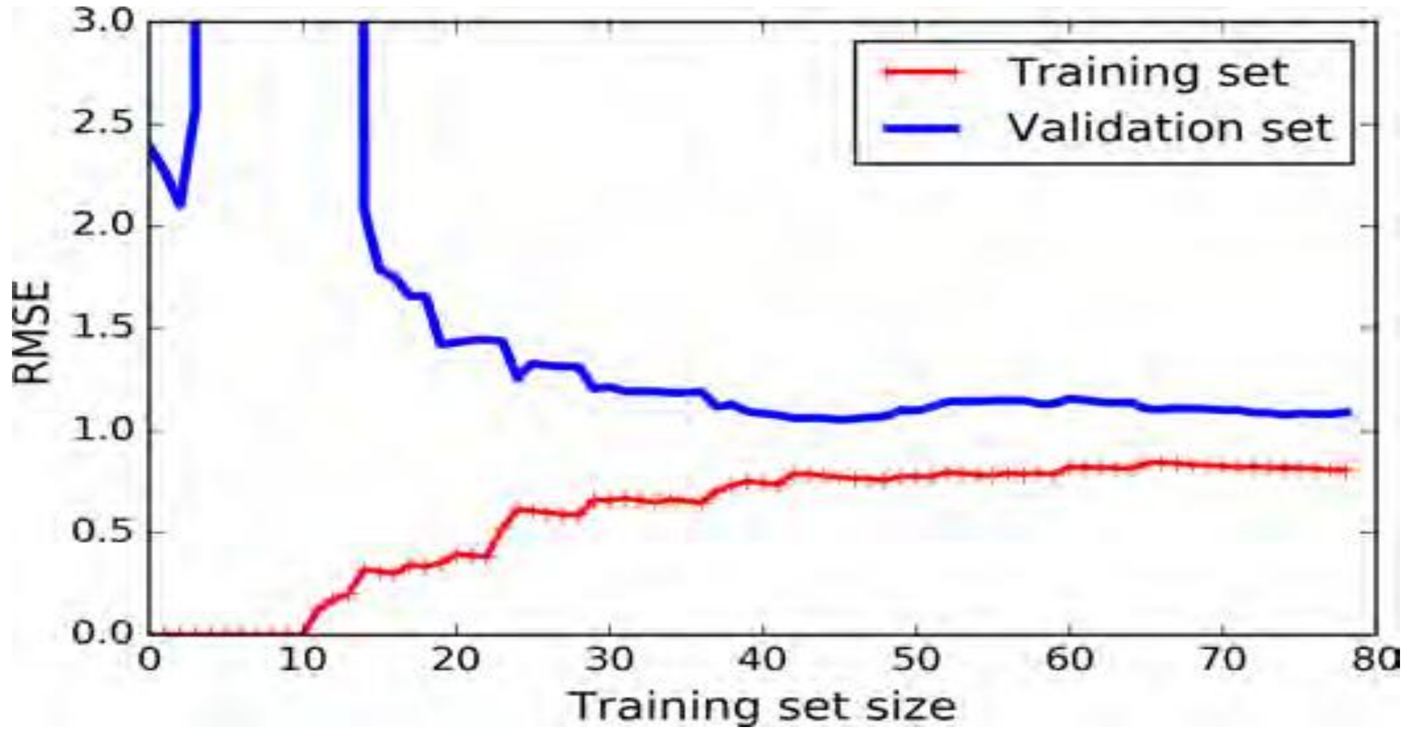
- **Performance on the training data:**
- When there are just one or two instances in the training set, the model can fit them perfectly, which is why the curve starts at zero.
- But as new instances are added to the training set, it becomes impossible for the model to fit the training data perfectly, both because the data is noisy and because it is not linear at all.
- So the error on the training data goes up until it reaches a plateau, at which point adding new instances to the training set doesn’t make the average error much better or worse.

Polynomial Regression

- **Performance of the model on the validation data:**
- When the model is trained on very few training instances, it is incapable of generalizing properly, which is why the validation error is initially quite big.
- Then as the model is shown more training examples, it learns and thus the validation error slowly goes down.
- However, once again a straight line cannot do a good job modeling the data, so the error ends up at a plateau, very close to the other curve.

Polynomial Regression

- Now let's look at the learning curves of a 10th-degree polynomial model on the same data.
- Learning Curves for the Polynomial Model



Polynomial Regression

- These learning curves look a bit like the previous ones, but there are two very important differences:

1. The error on the training data is much lower than with the Linear Regression model.
2. There is a gap between the curves. This means that the model performs significantly better on the training data than on the validation data.

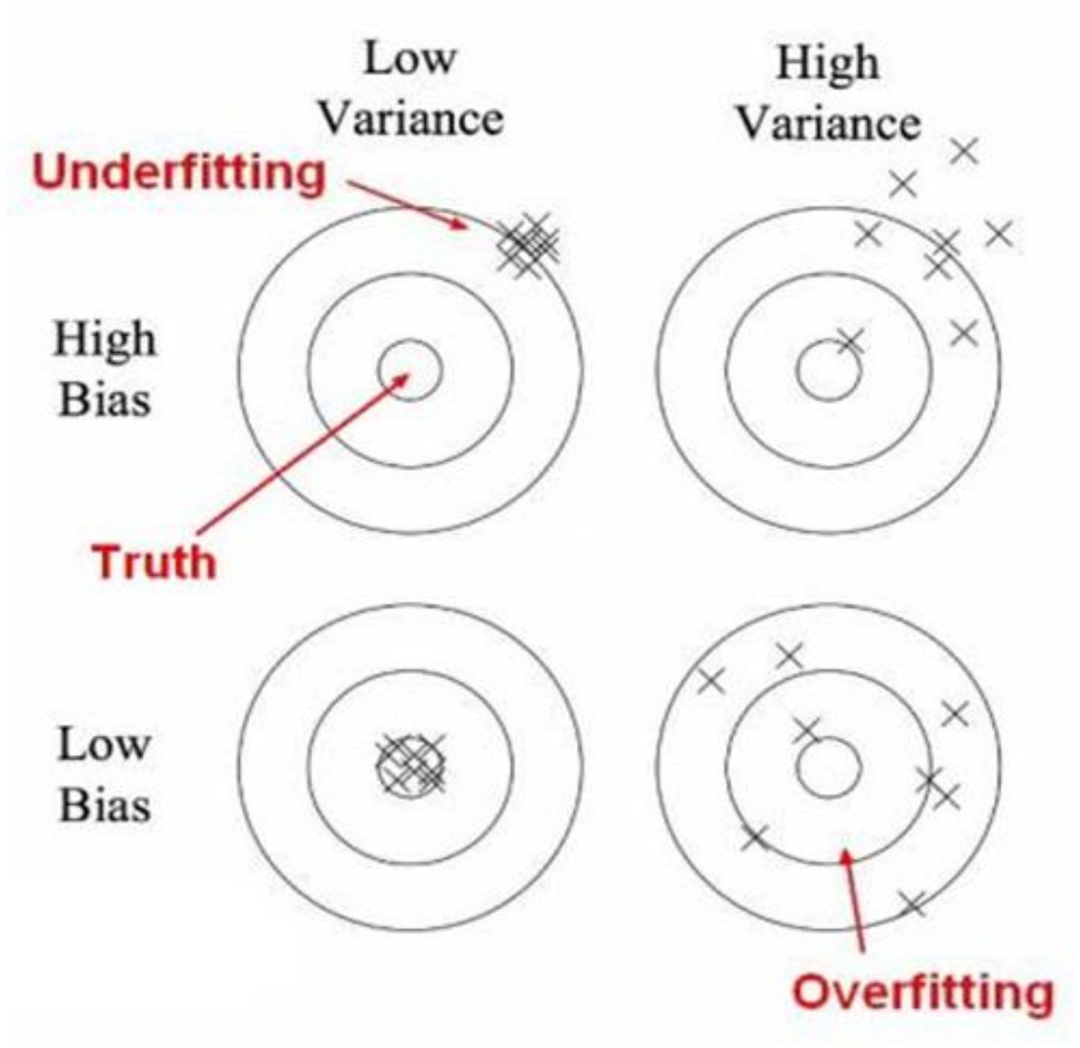
The Bias/Variance Tradeoff

- A model's generalization error can be expressed as the sum of three very different errors:
- **Bias**
  - Difference between the average prediction of our model and the correct value which we are trying to predict
  - This part of the generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic.
  - A high-bias model is most likely to under fit the training data.
  - Model with high bias always leads to high error on training and test data.

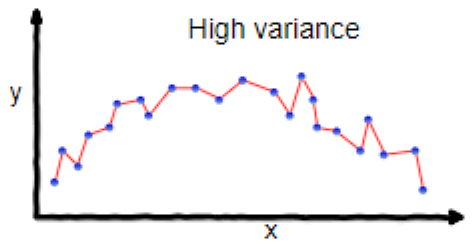
The Bias/Variance Tradeoff

- **Variance**
  - Variance is the amount that the estimate of the target function will change if different training data was used.
  - Excessive sensitivity to the model’s small variations in the training data.
  - A model with many degrees of freedom is likely to have high variance, and thus to over fit the training data.
- **Irreducible Error**
  - Due to the noisiness of the data itself.
  - The only way to reduce this part of the error is to clean up the data.

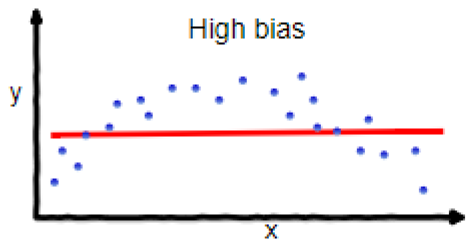
The Bias/Variance Tradeoff



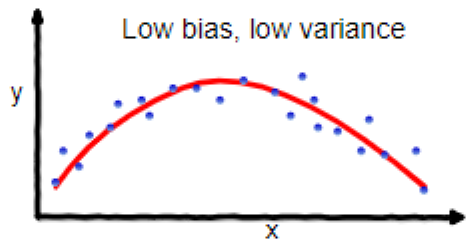
The Bias/Variance Tradeoff



overfitting



underfitting



Good balance

The Bias/Variance Tradeoff

- **If you have High Variance Problem:**
  1. You can get more training examples because a larger the dataset is more probable to get a higher predictions.
  2. Try smaller sets of features (because you are overfitting)
  3. Try increasing lambda, so you can not overfit the training set as much. The higher the lambda, the more the regularization applies, for Linear Regression with regularization.

The Bias/Variance Tradeoff

- **If you have High Bias Problem:**
  1. Try getting additional features, you are generalizing the datasets.
  2. Try adding polynomial features, make the model more complicated.
  3. Try decreasing lambda, so you can try to fit the data better. The lower the lambda, the less the regularization applies, for Linear Regression with regularization.

Regularized Linear Models

- One of the major aspects of training your machine learning model is avoiding overfitting.
- The model will have a low accuracy if it is overfitting.
- This happens because your model is trying too hard to capture the noise in your training dataset.
- By noise we mean the data points that don’t really represent the true properties of your data.
- Learning such data points, makes your model more flexible, at the risk of overfitting.

Regularized Linear Models

- A good way to reduce overfitting is to regularize the model i.e., to constrain it.
- The fewer degrees of freedom it has, the harder it will be for it to overfit the data.
- This is a form of regression, that constrains / regularizes or shrinks the coefficient estimates towards zero.
- This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.

Regularized Linear Models

- For example, a simple way to regularize a polynomial model is to reduce the number of polynomial degrees.
- For a linear model, regularization is typically achieved by constraining the weights of the model.
  1. Ridge Regression
  2. Lasso Regression
  3. Elastic Net

Ridge Regression

- Ridge Regression is a regularized version of Linear Regression: a regularization term is added to the cost function.
- Regularization Tem: $\alpha\Sigma_{i=1}^{n}\theta_i^2$
- This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible.
- Note that the regularization term should only be added to the cost function during training.
- Once the model is trained, you want to evaluate the model’s performance using the un-regularized performance measure.

Ridge Regression

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$

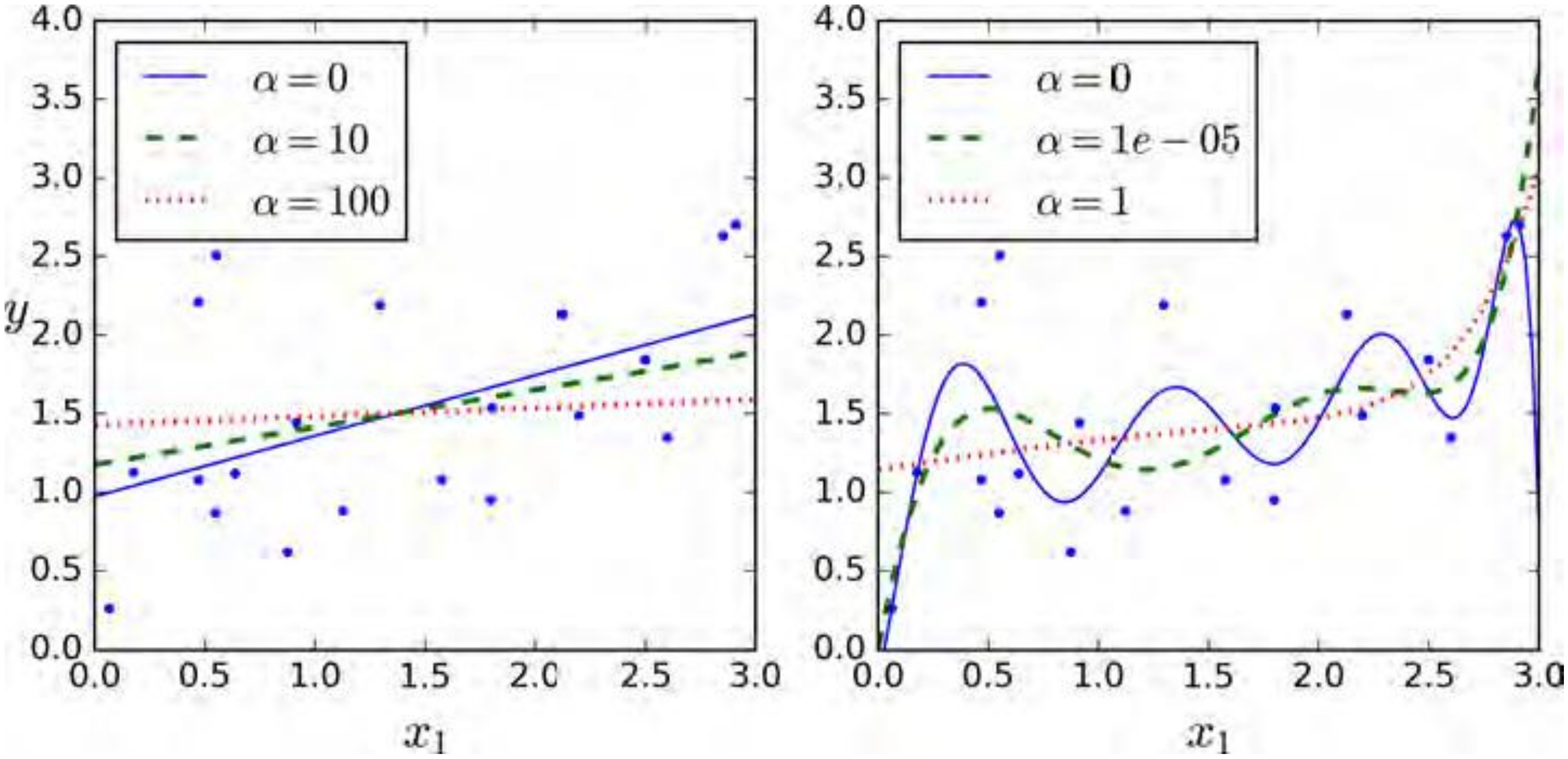
- RSS is modified by adding the shrinkage quantity.
- The coefficients are estimated by minimizing this function.
- Here, λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.
- The increase in flexibility of a model is represented by increase in its coefficients.

Ridge Regression

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$

- If we want to minimize the above function, then these coefficients need to be small.
- This is how the Ridge regression technique prevents coefficients from rising too high.
- When λ = 0, the penalty term has no effect.
- As λ → infinity, the impact of the shrinkage penalty grows, & the ridge regression coefficient estimates will approach zero.
- Selecting a good value of λ is critical.
- Cross validation comes in handy for this purpose.

Ridge Regression



Ridge Regression

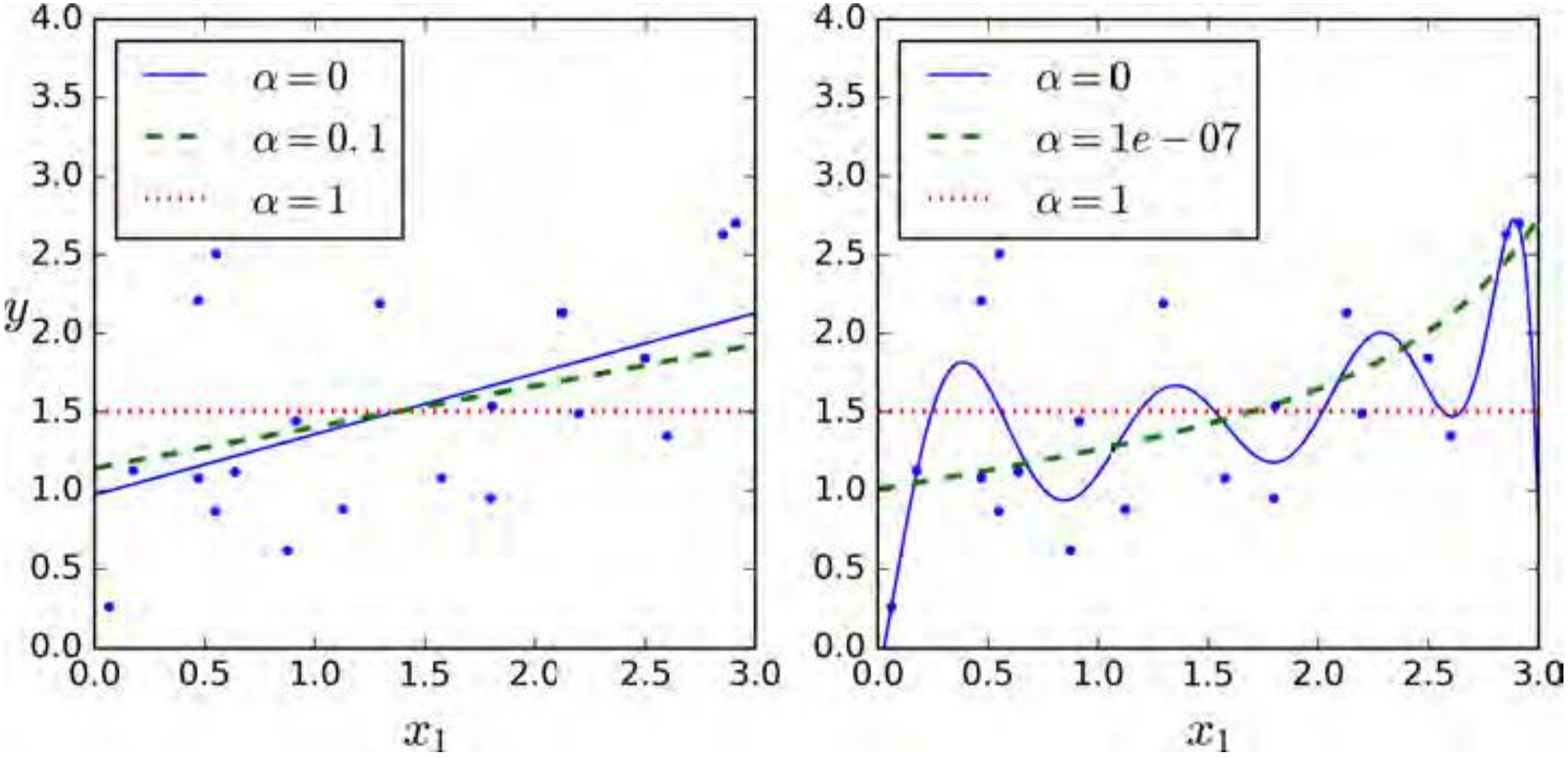
- **Advantages**
  1. Ridge regression can reduce the variance (with an increasing bias) → works best in situations where the OLS estimates have high variance.
  2. Can improve predictive performance
  3. Works in situations where $p < n$
  4. Mathematically simple computations
- **Disadvantages**
  1. Ridge regression is not able to shrink coefficients to exactly zero
  2. As a result, it cannot perform variable selection

Lasso Regression

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

- Lasso is another variation, in which the above function is minimized.
- It uses |βj|(modulus)instead of squares of β, as its penalty.
- Lasso can be thought of as an equation where summation of modulus of coefficients is less than or equal to s.
- Here, s is a constant that exists for each value of shrinkage factor λ.
- These equations are also referred to as constraint functions.

Lasso Regression



Lasso Regression

- An important characteristic of Lasso Regression is that it tends to completely eliminate the weights of the least important features (i.e., set them to zero).
- Lasso Regression automatically performs feature selection and outputs a sparse model (i.e., with few nonzero feature weights).

Elastic Net

- Elastic Net is a middle ground between Ridge Regression and Lasso Regression.
- The regularization term is a simple mix of both Ridge and Lasso's regularization terms, and you can control the mix ratio r.
- When r = 0, Elastic Net is equivalent to Ridge Regression.
- When r = 1, it is equivalent to Lasso Regression.
- Elastic Net Cost Function:

$$J(\theta) = \mathrm{MSE}(\theta) + r\alpha \sum_{i=1}^{n} \left|\theta_i\right| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \theta_i^2$$

Elastic Net

- Ridge is a good default, but if you suspect that only a few features are actually useful, you should prefer Lasso or Elastic Net.
- Lasso or Elastic Net tend to reduce the useless features’ weights down to zero.
- Elastic Net is preferred over Lasso since Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

Locally Weighted Regression

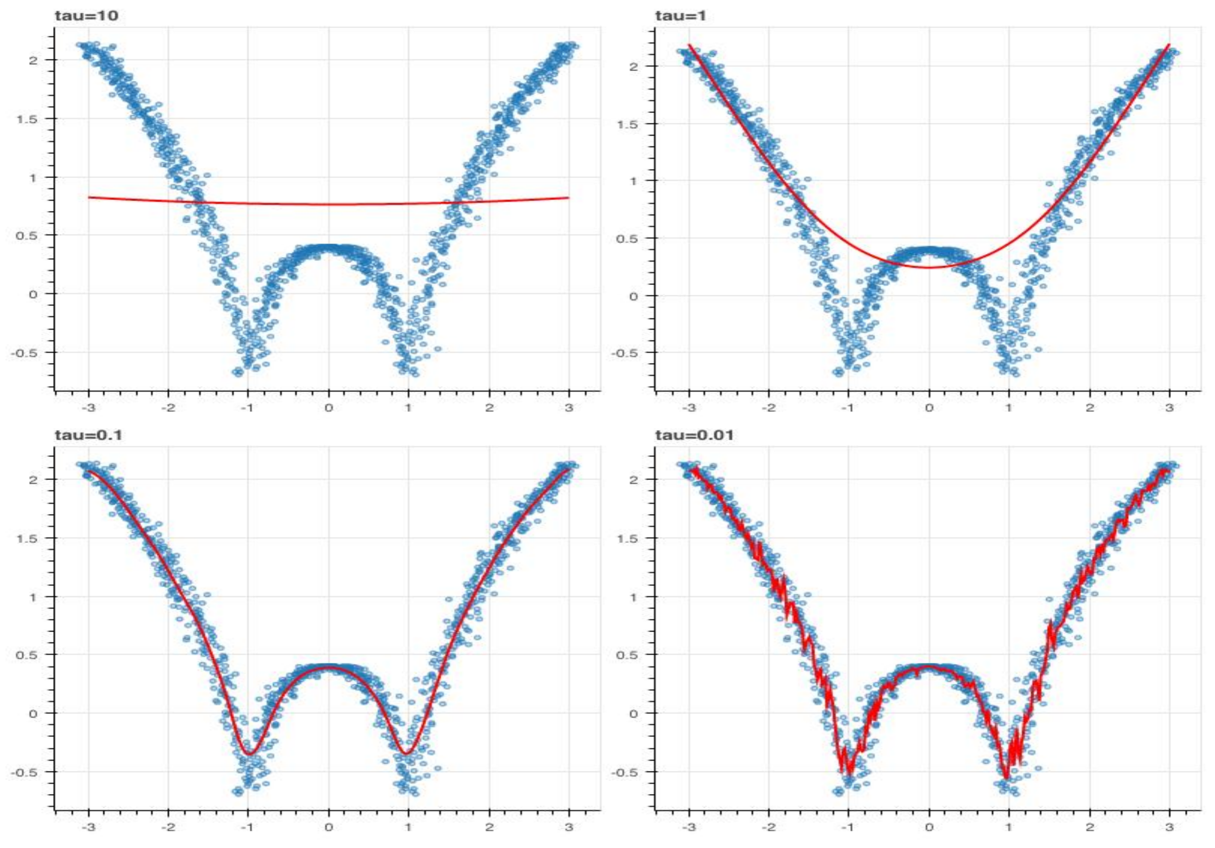
Locally Weighted Linear Regression

- Linear Regression tends to underfit the data.
- It gives us the lowest mean-squared error for unbiased estimators.
- With the model underfit, we aren’t getting the best predictions.
- There are a number of ways to reduce this mean-squared error by adding some bias into our estimator.
- One way to reduce the mean-squared error is a technique known as locally weighted linear regression (LWLR).

Locally Weighted Linear Regression

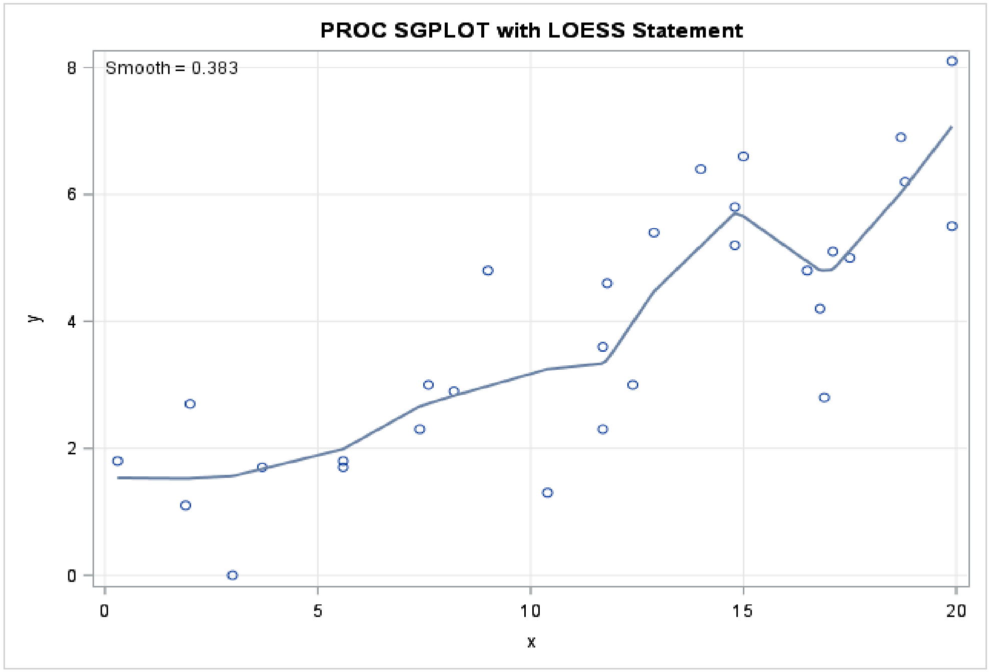
- In LWLR we give a weight to data points near our data point of interest; then we compute a least-squares regression.
- One option for non-linear regression is to fit a polynomial model of the form $Y = \beta0 + \beta1X + \beta2X2 + ...$
- These are superficially attractive, because higher order polynomials can be made to 'fit' almost any pattern.
- However, interpretation of such models is very difficult and their shape is sensitive to outliers.

Locally Weighted Regression Algorithm



Locally Weighted Regression Algorithm

- Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Locally Weighted Regression Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set X0 which is a subset of X
4. Determine the weight matrix using :

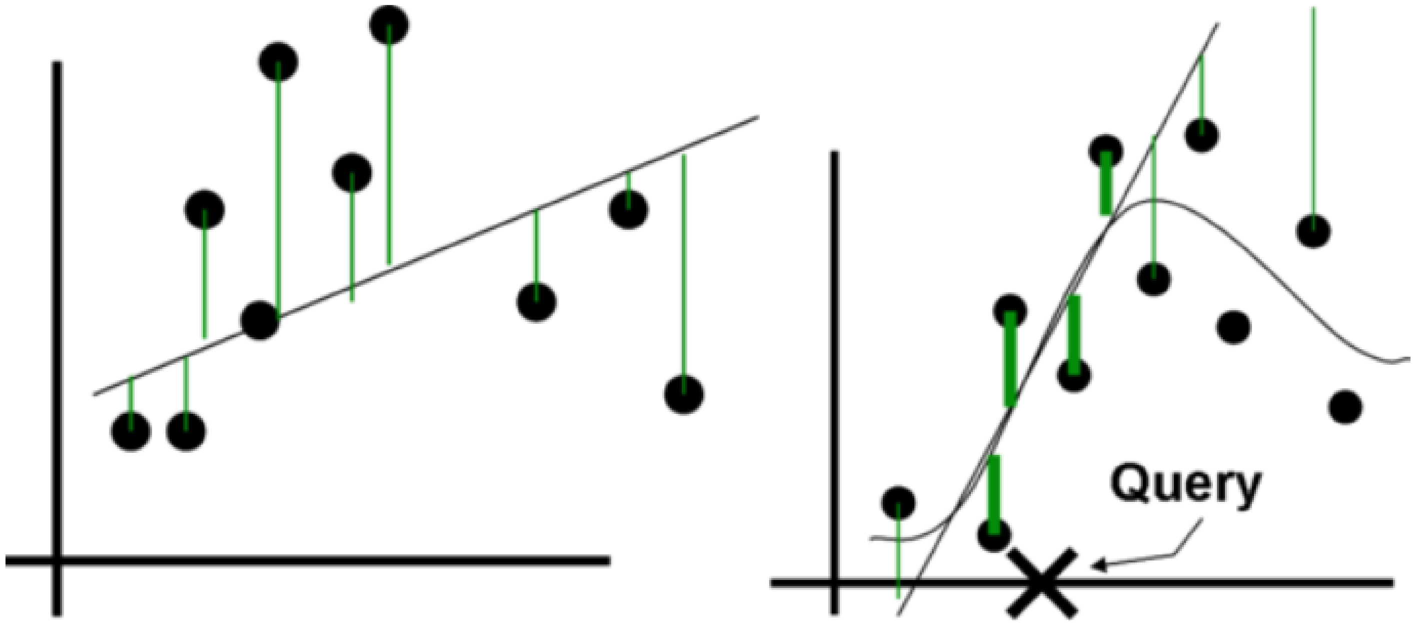
$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

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

6. Prediction = x0*β

Linear Regression vs Locally Weighted Regression



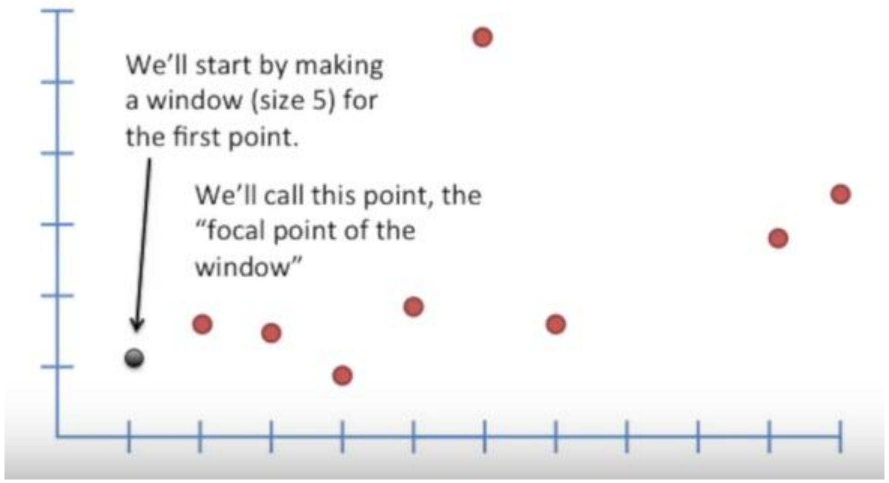
Locally Weighted Regression

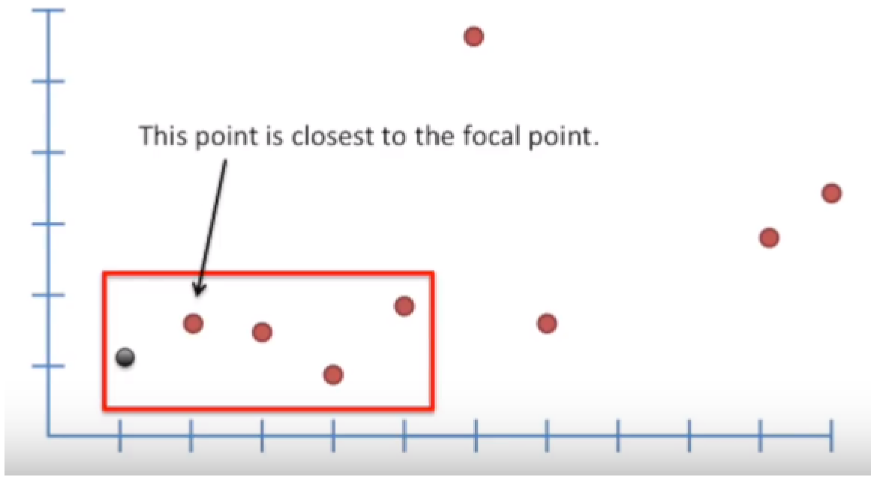
- **Advantages:**
  - Allows us to put less care into selecting the features in order to avoid overfitting
  - Does not require specification of a function to fit a model to all of the data in the sample
  - Only a Kernel function and smoothing / bandwidth parameters are required
  - Very flexible, can model complex processes for which no theoretical model exists
  - Considered one of the most attractive of the modern regression methods for applications that fit the general framework of least squares regression but which have a complex deterministic structure.

Locally Weighted Regression

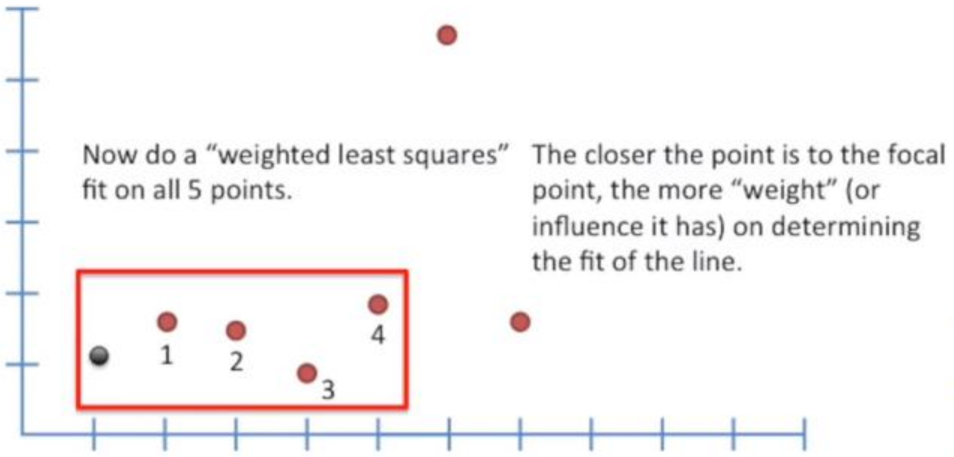
- **Disadvantages:**
  - Requires to keep the entire training set in order to make future predictions.
  - The number of parameters grows linearly with the size of the training set.
  - Computationally intensive.
  - Requires fairly large, densely sampled data sets in order to produce good models.
  - Does not translate into a model that can be described by a mathematical formula.
  - Can perform poorly if p is much larger than about 3 or 4 because there will generally be very few training observations close to x0.

Locally Weighted Regression - Working



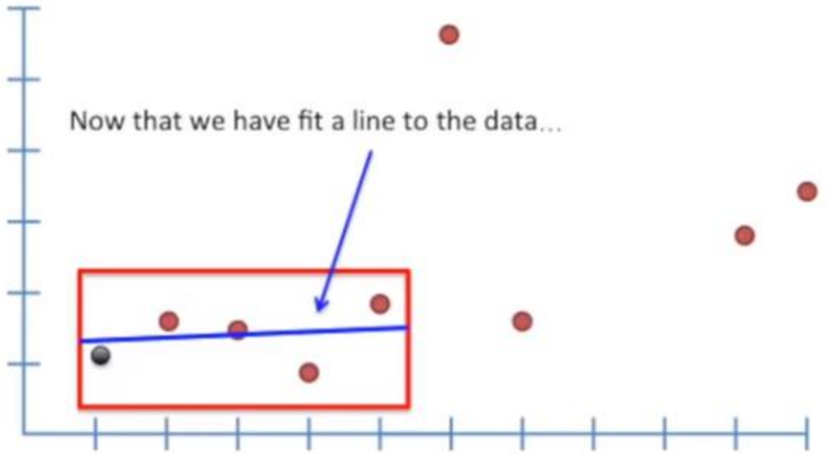


Locally Weighted Regression - Working



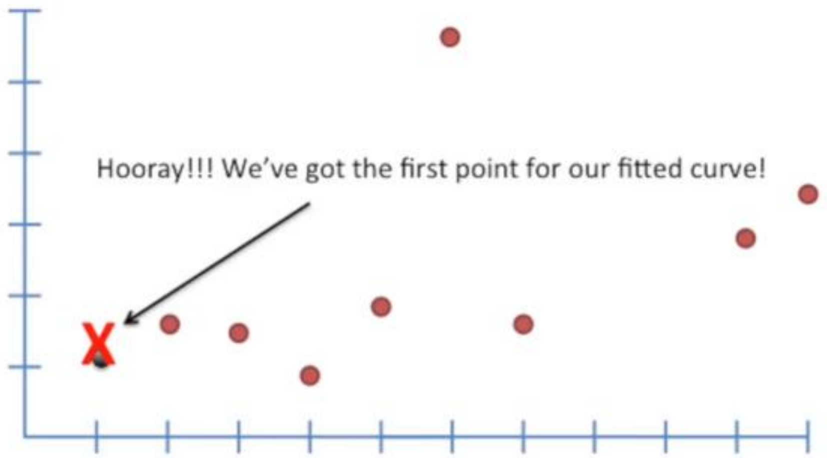
Now do a "weighted least squares" fit on all 5 points.

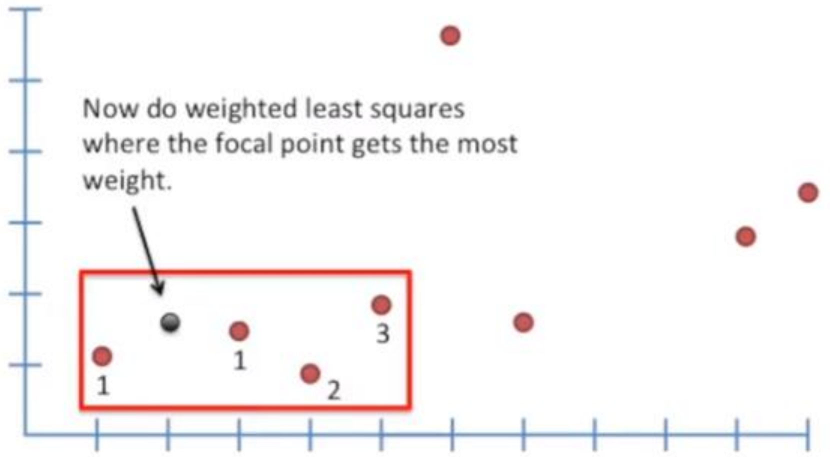
The closer the point is to the focal point, the more "weight" (or influence it has) on determining the fit of the line.



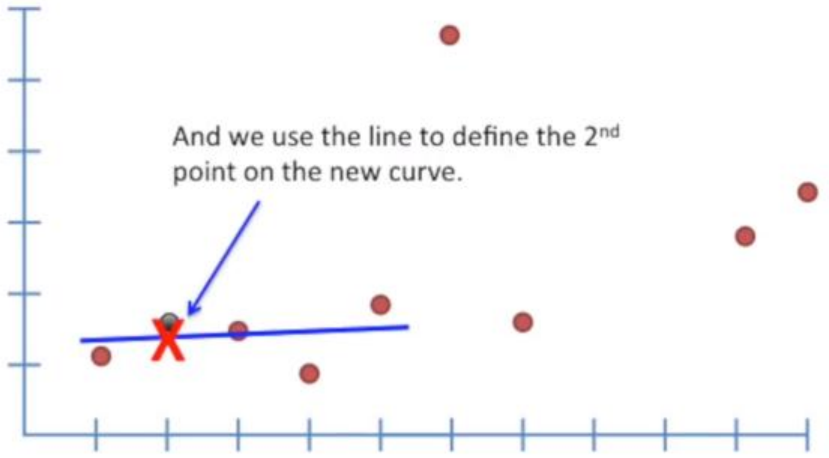
Now that we have fit a line to the data...

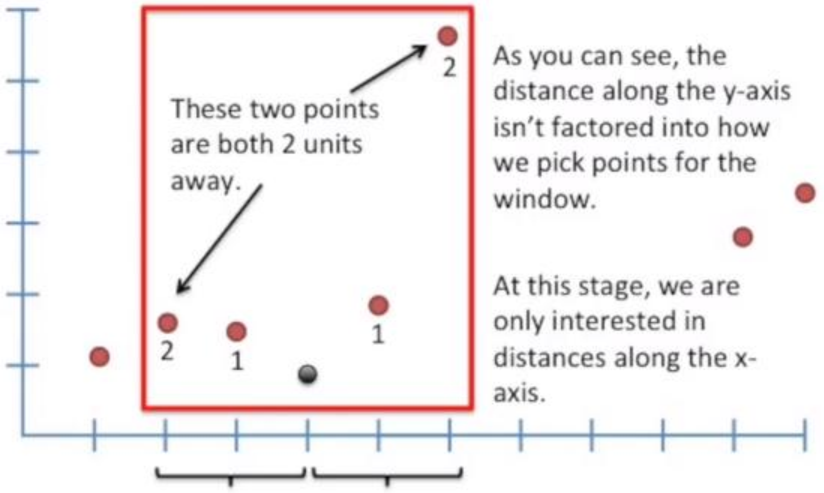
Locally Weighted Regression - Working



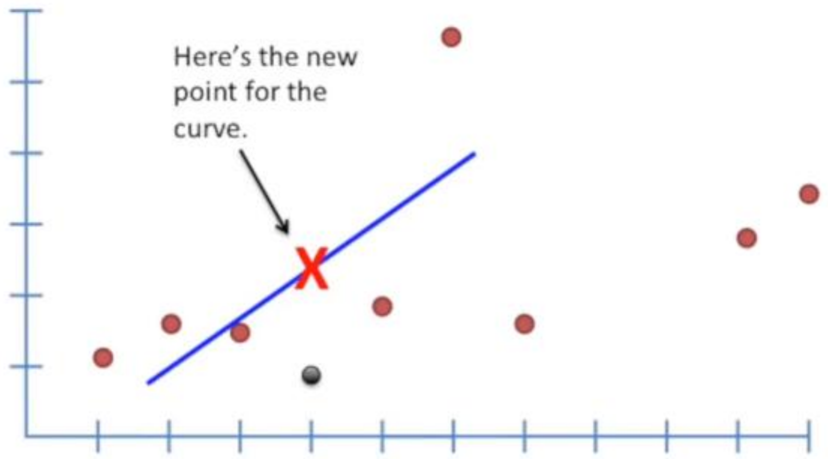


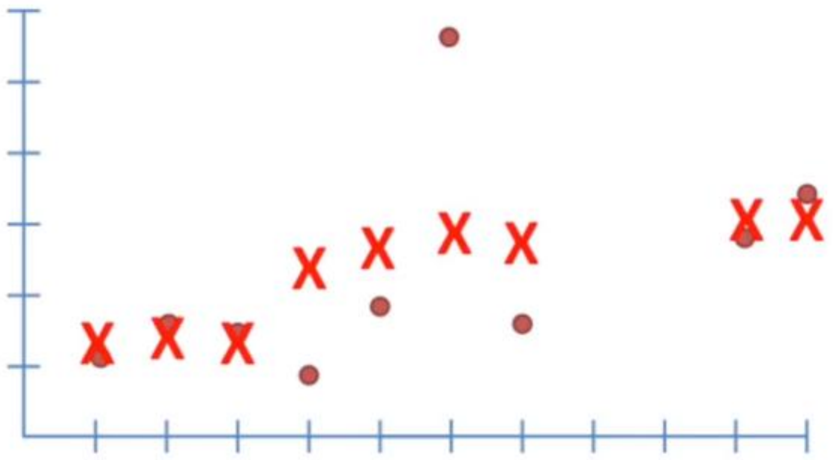
Locally Weighted Regression - Working


And we use the line to define the 2nd point on the new curve.
X

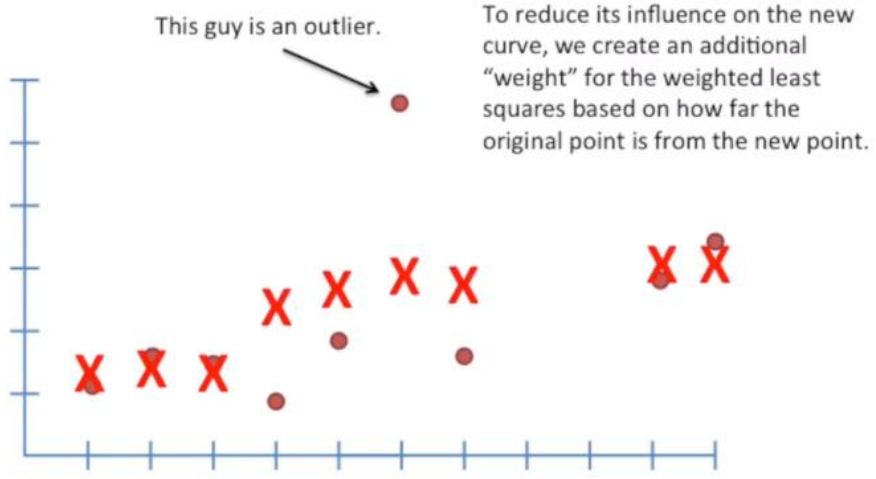

These two points are both 2 units away.
2
1
1
2
As you can see, the distance along the y-axis isn't factored into how we pick points for the window.
At this stage, we are only interested in distances along the x-axis.

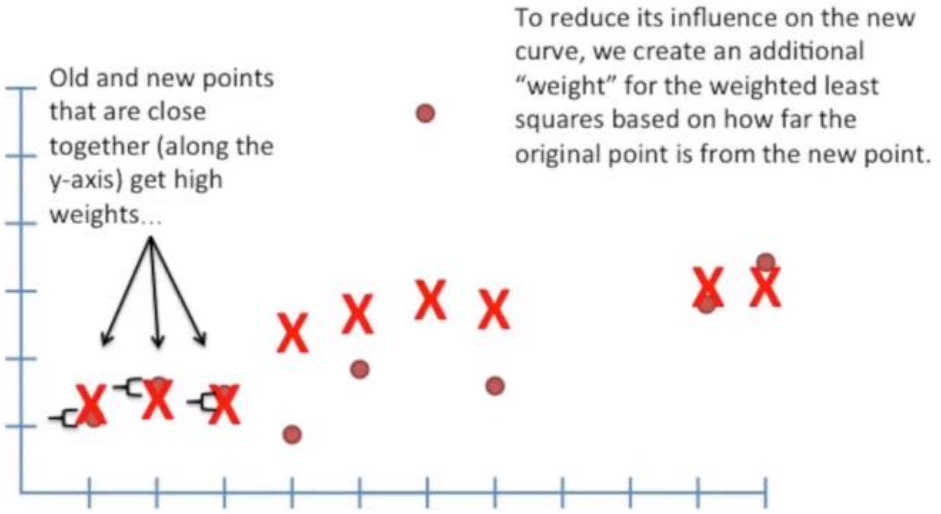
Locally Weighted Regression - Working



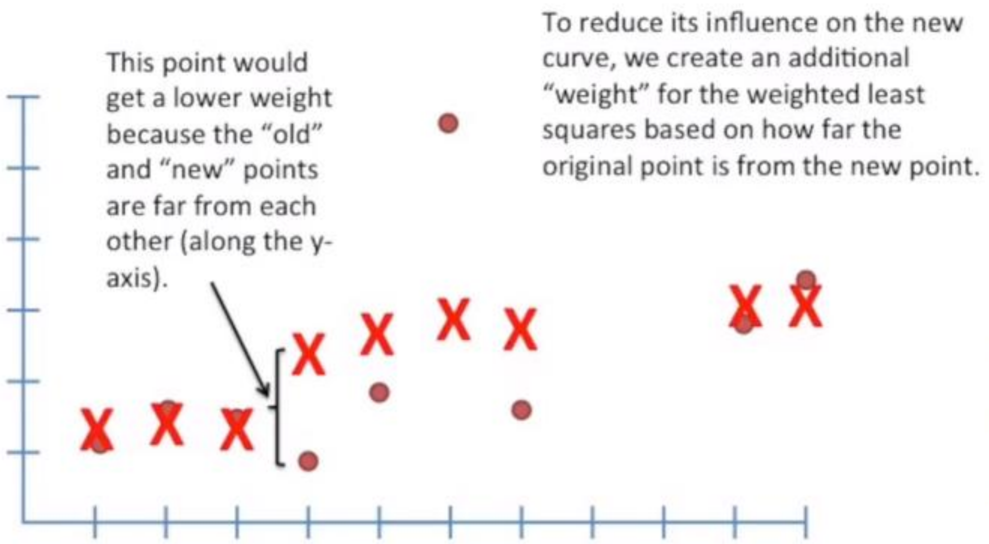


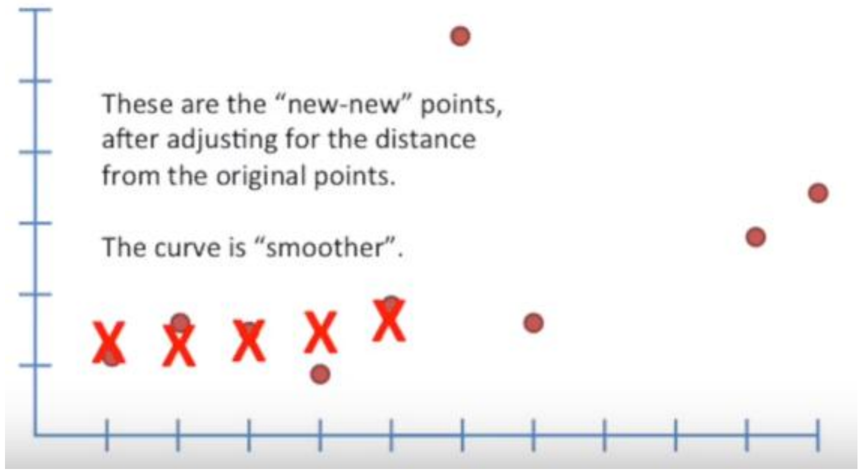
Locally Weighted Regression - Working


This guy is an outlier.
To reduce its influence on the new curve, we create an additional "weight" for the weighted least squares based on how far the original point is from the new point.

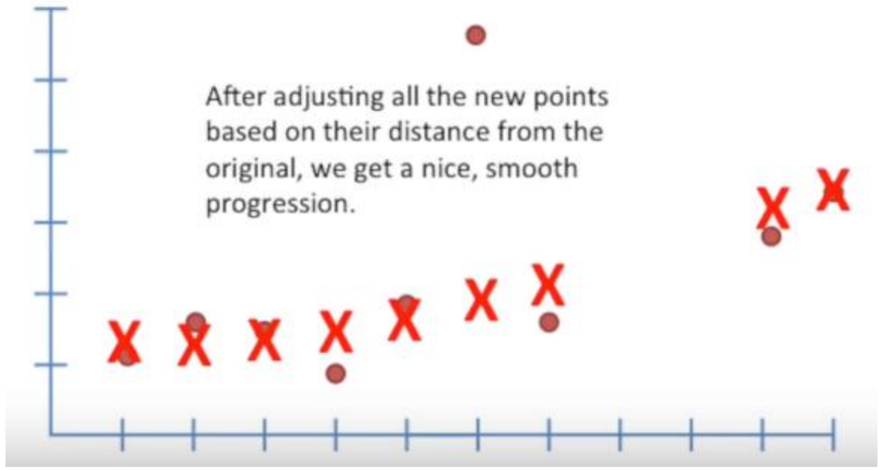

Old and new points that are close together (along the y-axis) get high weights...
To reduce its influence on the new curve, we create an additional "weight" for the weighted least squares based on how far the original point is from the new point.

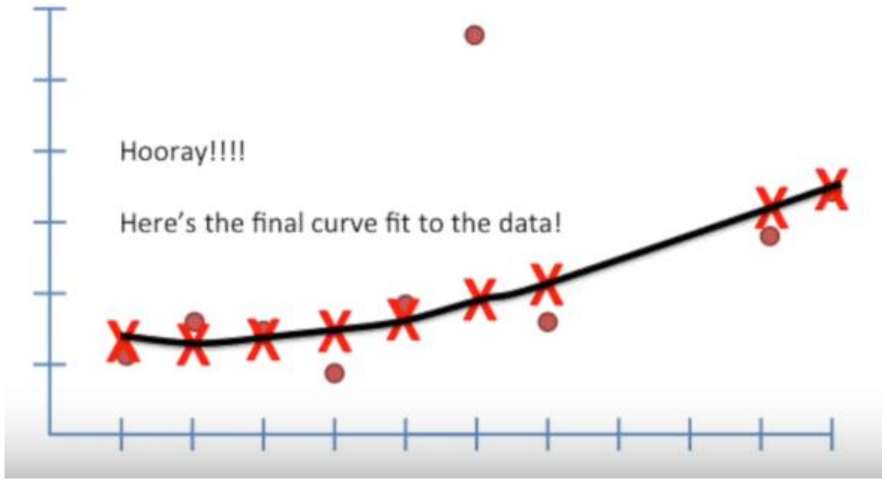
Locally Weighted Regression - Working





Locally Weighted Regression - Working





Remarks on Locally Weighted Regression

- The literature on locally weighted regression contains a broad range of alternative methods for distance weighting the training examples, and a range of methods for locally approximating the target function.

- In most cases, the target function is approximated by a constant, linear, or quadratic function.

Multivariate Linear Regression

- More generally, a linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias term* (also called the *intercept term*).

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

  - $\hat{y}$ is the predicted value.
  - *n* is the number of features.
  - *x*i is the ith feature value.
  - θ*j* is the jth model parameter (including the bias term *θ*0 and the feature weights θ1, θ2, , θ*n*).

Multivariate Linear Regression

- Training a model means setting its parameters so that the model best fits the training set.
- For this purpose, we first need a measure of how well (or poorly) the model fits the training data.
- The most common performance measure of a regression model is the Root Mean Square Error (RMSE). Therefore, to train a Linear Regression model, you need to find the value of θ that minimizes the RMSE.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

Multivariate Linear Regression

- In practice, it is simpler to minimize the Mean Square Error (MSE) than the RMSE, and it leads to the same result (because the value that minimizes a function also minimizes its square root).

$$\mathrm{MSE}\left(\mathbf{X}, h_{\theta}\right) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^{T}\cdot\mathbf{x}^{(i)} - y^{(i)}\right)^{2}$$

Logistic Regression

- Some regression algorithms can also be used for classification.
- Logistic Regression (Logit Regression) is commonly used to estimate the probability that an instance belongs to a particular class.
- If the estimated probability is greater than 50%, then the model predicts that the instance belongs to positive class.
- Else it predicts that it does not the negative class.
- This makes it a binary classifier.

Logistic Regression

- Logistic Regression model computes a weighted sum of the input features (plus a bias term).
- But instead of outputting the result directly like the Linear Regression model does, it outputs the logistic of this result.
- Logistic Regression model estimated probability.

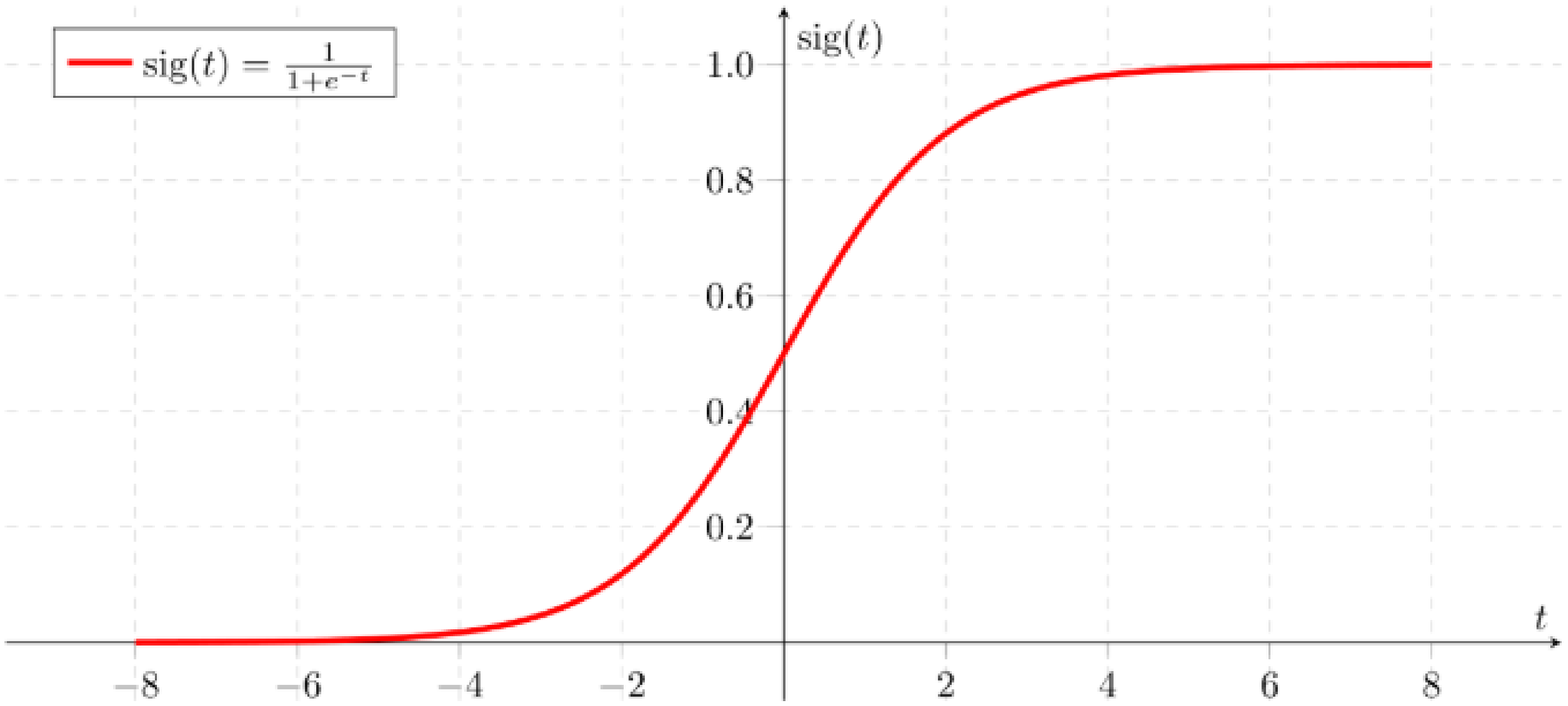
$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma\left(\theta^{T} \cdot \mathbf{x}\right)$$

- The logistic noted σ(·)—is a sigmoid function (i.e., S-shaped) that outputs a number between 0 and 1.

Logistic Regression

- Logistic function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Logistic Regression

- Once the Logistic Regression model has estimated the probability that an instance x belongs to the positive class, it can make its prediction ŷ easily.

- Logistic Regression model prediction

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5, \\ 1 & \text{if } \hat{p} \geq 0.5. \end{cases}$$

Logistic Regression

- Logistic Regression is used when the dependent variable is categorical.
- For example,
  - To predict whether an email is spam (1) or (0)
  - Whether the tumor is malignant (1) or not (0)

- Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable.

Logistic Regression

- Types of Logistic Regression

1. **Binary Logistic Regression**:
   - The categorical response has only two 2 possible outcomes.
   - Example: Spam or Not
2. **Multinomial Logistic Regression**:
   - Three or more categories without ordering.
   - Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)
3. **Ordinal Logistic Regression**:
   - Three or more categories with ordering.
   - Example: Movie rating from 1 to 5.

Logistic Regression

- **Advantages:**
  1. Logistic Regression performs well when the dataset is linearly separable.
  2. Logistic regression is less prone to over-fitting but it can overfit in high dimensional datasets. You should consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.
  3. Logistic Regression not only gives a measure of how relevant a predictor (coefficient size) is, but also its direction of association (positive or negative).
  4. Logistic regression is easier to implement, interpret and very efficient to train.

Logistic Regression

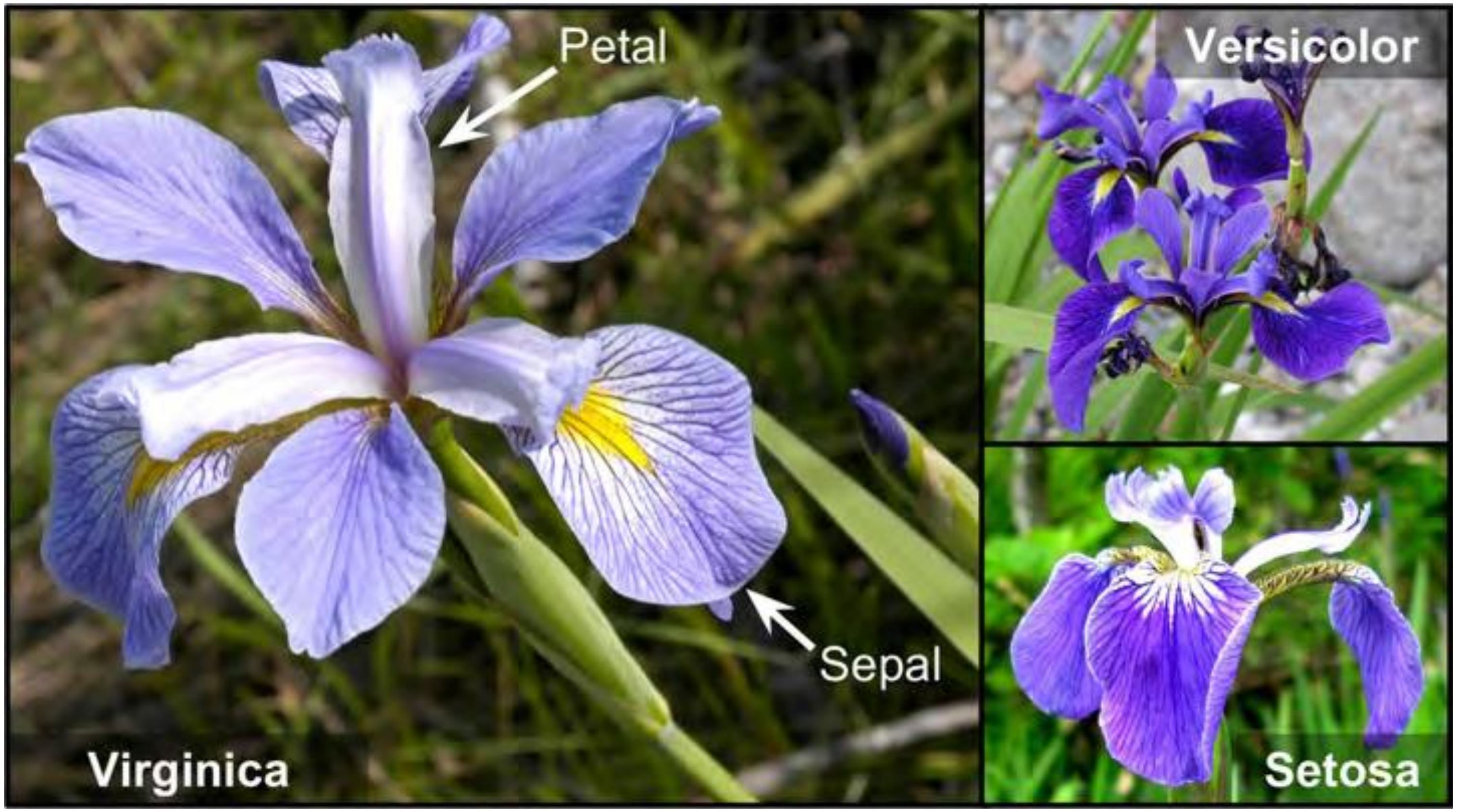
- **Disadvantages:**
  1. Main limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data is rarely linearly separable.
  2. If the number of observations are lesser than the number of features, Logistic Regression should not be used, otherwise it may lead to overfit.
  3. Logistic Regression can only be used to predict discrete functions. Therefore, the dependent variable of Logistic Regression is restricted to the discrete number set. This restriction itself is problematic, as it is prohibitive to the prediction of continuous data.

Logistic Regression

- **Decision Boundary**
- To predict which class a data belongs, a threshold can be set.
- Based upon this threshold, the obtained estimated probability is classified into classes.
- Say, if predicted_value $\geq$ 0.5, then classify email as spam else as not spam.
- Decision boundary can be linear or non-linear.
- Polynomial order can be increased to get complex decision boundary.

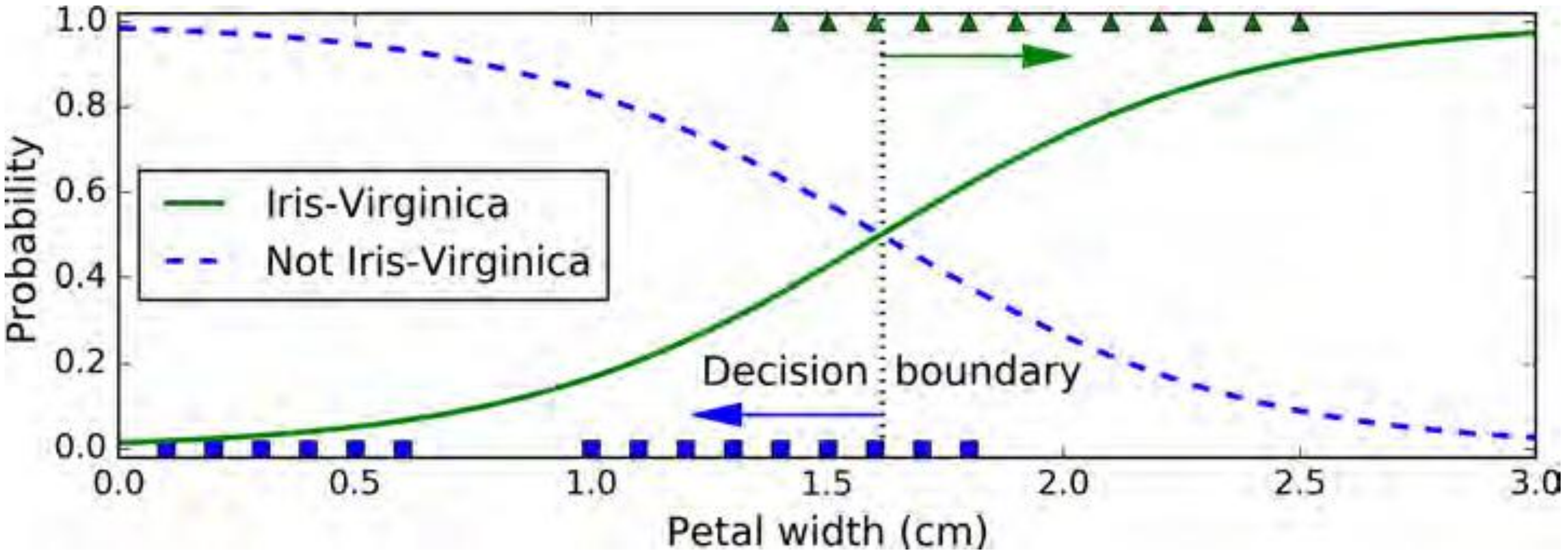
Logistic Regression

- **Decision Boundary**
- Flowers of three iris plant species



Logistic Regression

- **Decision Boundary**
- Estimated probabilities and decision boundary



Logistic Regression

- **Using optimization to find the best regression coefficients**
- The input to the sigmoid function described will be z, where z is given by the following.

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- In vector notation we can write this as $z = w^T x$.
- All that means is that we have two vectors of numbers and we'll multiply each element and add them up to get one number.
- The vector x is our input data, and we want to find the best coefficients w.

Logistic Regression

- **Gradient Ascent**
- Gradient ascent is based on the idea that if we want to find the maximum point on a function, then the best way to move is in the direction of the gradient.
- We write the gradient with the symbol and the ∇ gradient of a function f(x,y) is given by the equation.

$$\nabla f(x,y) = \begin{pmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

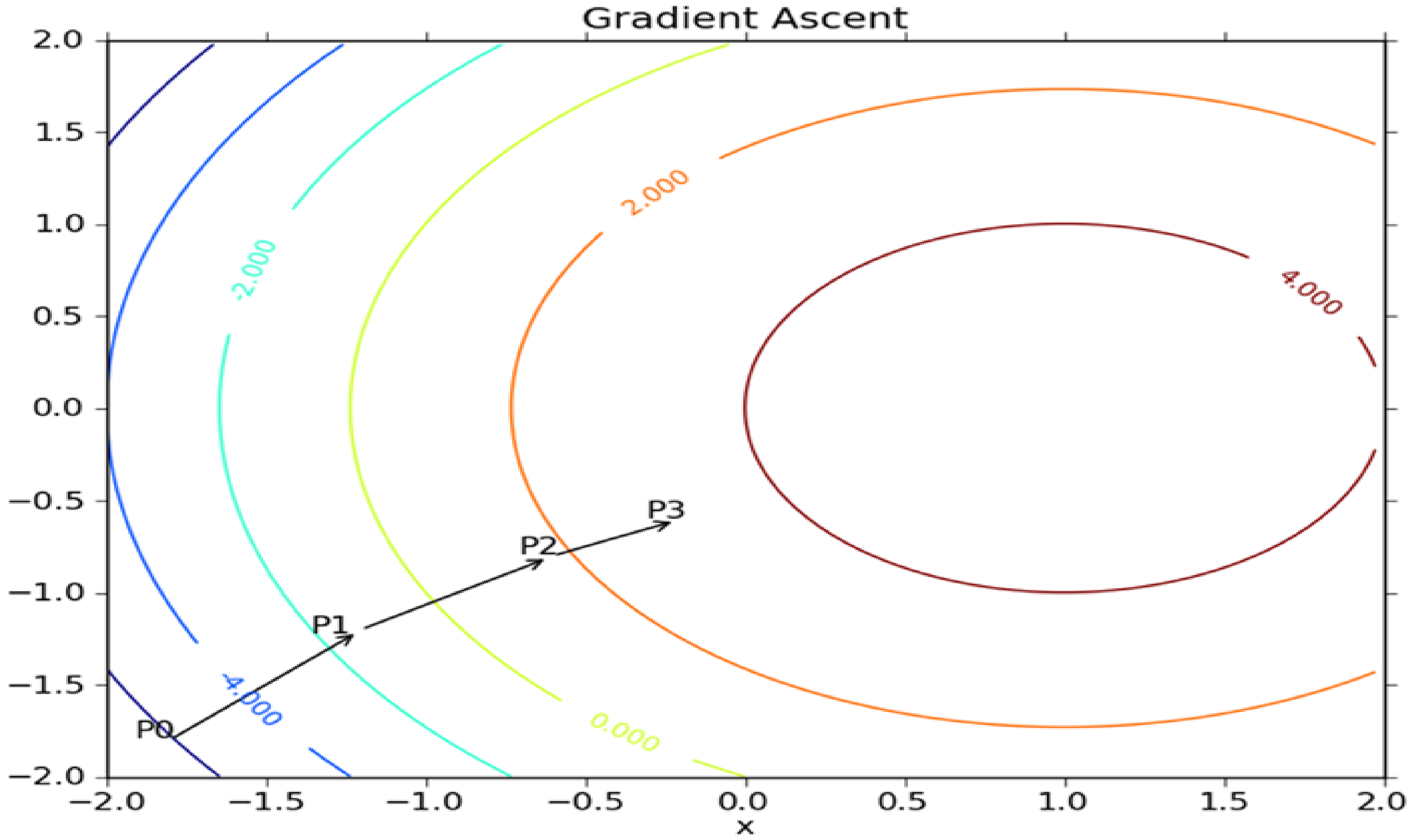
Logistic Regression

- **Gradient Ascent**
- So this gradient means that we'll move in the x direction by amount $\frac{\partial f(x,y)}{\partial x}$ and in the y direction by amount $\frac{\partial f(x,y)}{\partial y}$

- The function f(x,y) needs to be defined and differentiable around the points where it's being evaluated.
- In vector notation we can write the gradient ascent algorithm as $w := w + \alpha \nabla_{\mathbf{w}} f(w)$

Logistic Regression

- **Gradient Ascent**



Logistic Regression

- **Gradient Ascent**
- The gradient ascent algorithm moves in the direction of the gradient evaluated at each point.
- Starting with point P0, the gradient is evaluated and the function moves to the next point, P1.
- The gradient is then reevaluated at P1, and the function moves to P2.
- This cycle repeats until a stopping condition is met.
- The gradient operator always ensures that we’re moving in the best possible direction.

THANK YOU