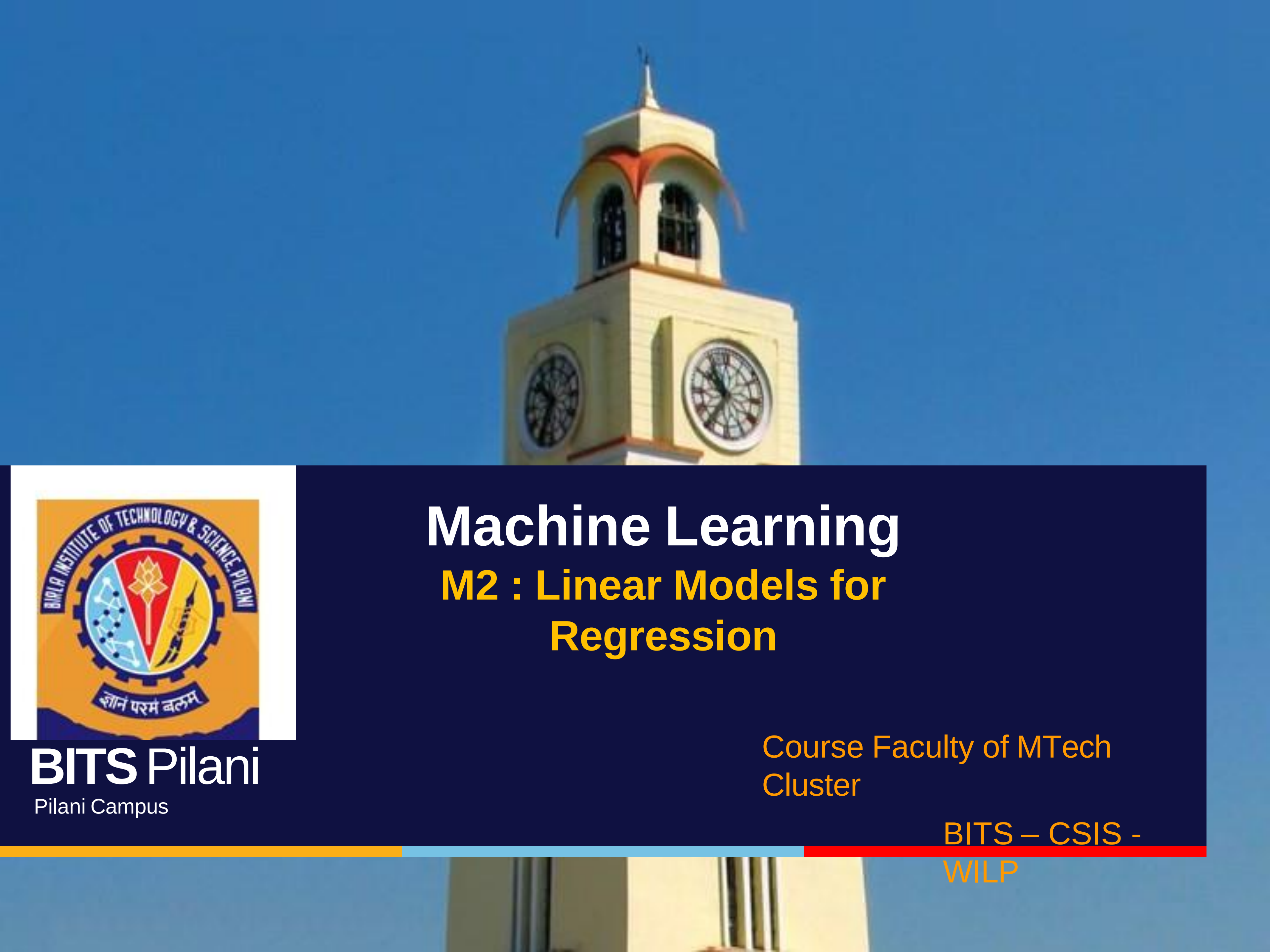


BITS Pilani
Pilani Campus

Machine Learning

M2 : Linear Models for Regression

Course Faculty of MTech Cluster

BITS – CSIS - WILP

Disclaimer and Acknowledgement

- The content for these slides has been obtained from books and various other source on the Internet
- We here by acknowledge all the contributors for their material and inputs.
- We have provided source information wherever necessary
- Students are requested to refer to the textbook w.r.t detailed content of the presentation deck shared over canvas
- We have reduced the slides from canvas and modified the content flow to suit the requirements of the course and for ease of class presentation

Source: "Probabilistic Machine Learning, An Introduction", Kevin P. Murphy, Slides of Prof.Sugata, Prof. Chetana from BITS Pilani, Prof. Raja vadhana from BITS Pilani , CS109 and CS229 stanford lecture notes and many others who made their course materials freely available online.

Course Plan

M1	Introduction
M2	Linear Models for Regression
M3	Linear Models for Classification
M4	Decision Tree
M5	Instance Based Learning
M6	Support Vector Machine
M7	Bayesian Learning
M8	Ensemble Learning
M9	Unsupervised Learning
M10	Machine Learning Model Evaluation/Comparison

Agenda

- Linear Model for Regression
- Direct solution vs Iterative Method
- Gradient Descent
- Linear Basis Function
- Notion of Bias vs Variance

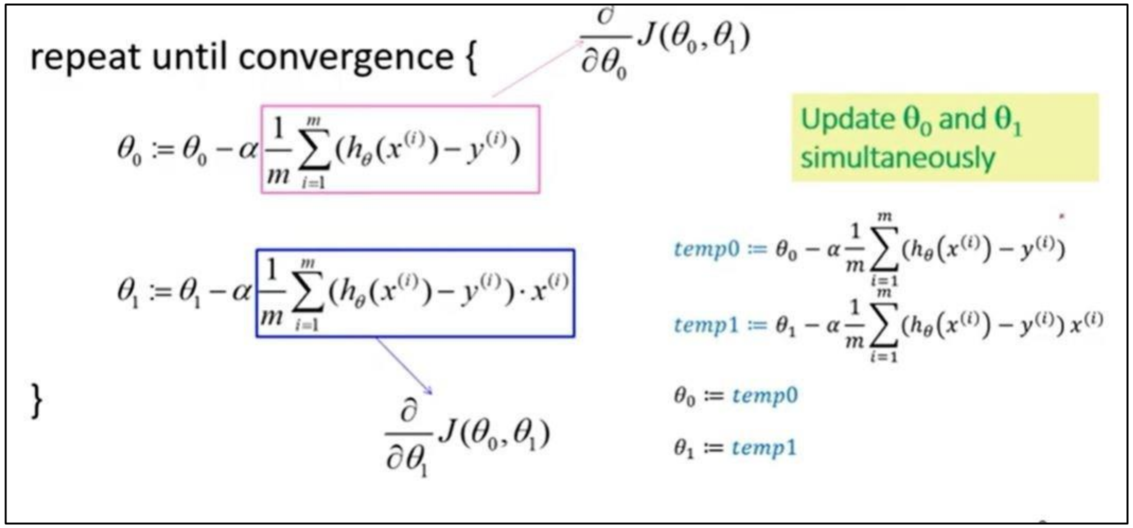
Types of Gradient Descent Algorithms

Gradient Descent: Variants

- **Batch** gradient descent refers to calculating the derivative from all training data before calculating an update.

- **Minibatch** refers to calculating derivative of mini groups of training data before calculating an update.

- **Stochastic** gradient descent refers to calculating the derivative from each training data instance and calculating the update immediately



repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \underbrace{\boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}}_{\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)}$$

$$\theta_1 := \theta_1 - \alpha \underbrace{\boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}}}_{\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$$

}

Update θ_0 and θ_1 simultaneously

$$temp0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$

$$temp1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$$

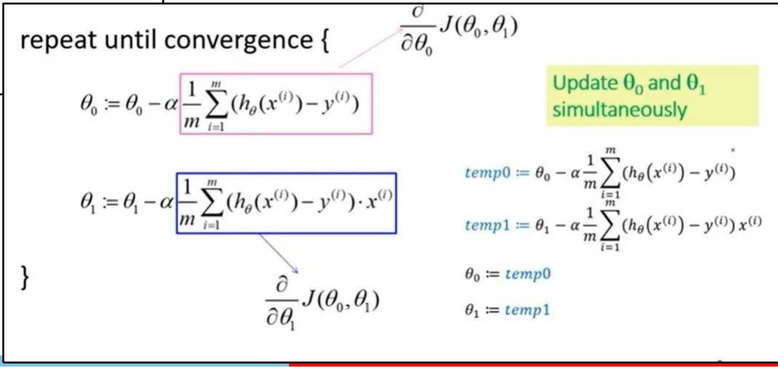
$$\theta_0 := temp0$$

$$\theta_1 := temp1$$

Gradient Descent: Variants

- **Batch** gradient descent refers to calculating the derivative from all training data before calculating an update.

```
Initialize the Parameters  ($\theta_0^1$, $\theta_1^1$ , …..)
K=1
Repeat until Convergence {
        $\theta_0^{k+1} = \theta_0^k - \frac{\alpha}{m} \sum_{i=1}^{m} ( h_\theta(x^{(i)}) - y^{(i)})$
        $\theta_1^{k+1} = \theta_1^k - \frac{\alpha}{m} \sum_{i=1}^{m} (( h_\theta(x^{(i)}) - y^{(i)}) * x_1^{(i)})$
        $\theta_2^{k+1} = \theta_2^k - \frac{\alpha}{m} \sum_{i=1}^{m} (( h_\theta(x^{(i)}) - y^{(i)}) * x_2^{(i)})$
        ……
K=k+1
}
return ($\theta_0^1$, $\theta_1^1$ , …..)
```



repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}$$

$$\frac{\partial}{\partial\theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}}$$

$$\frac{\partial}{\partial\theta_1} J(\theta_0, \theta_1)$$

}

Update θ_0 and θ_1 simultaneously

$$temp0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$

$$temp1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$$

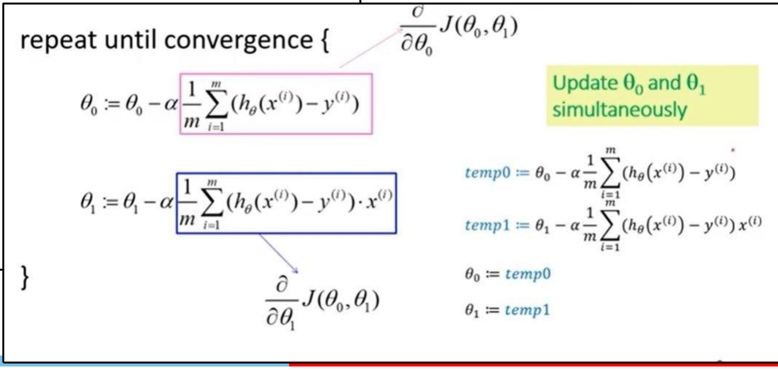
$$\theta_0 := temp0$$

$$\theta_1 := temp1$$

Gradient Descent: Variants

- **Minibatch** refers to calculating derivative of mini groups of training data before calculating an update.

```
Divide the training instances into "N" batches each of size "m"
Initialize the Parameters ($\theta_0^1$, $\theta_1^1$ , …..)
K=1
Repeat until Convergence {
    Repeat for every batch in 1 : N , each with 'm' instances {
        $\theta_0^{k+1} = \theta_0^k - \frac{\alpha}{m} \sum_{i=1}^{m} ( h_\theta(x^{(i)}) - y^{(i)})$
        $\theta_1^{k+1} = \theta_1^k - \frac{\alpha}{m} \sum_{i=1}^{m} (( h_\theta(x^{(i)}) - y^{(i)}) * x_1^{(i)})$
        $\theta_2^{k+1} = \theta_2^k - \frac{\alpha}{m} \sum_{i=1}^{m} (( h_\theta(x^{(i)}) - y^{(i)}) * x_2^{(i)})$
        ……
        K=k+1
    }
}
return ($\theta_0^1$, $\theta_1^1$ , …..)
```



repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})} \quad \frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)$$

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\cdot x^{(i)}} \quad \frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)$$

}

Update θ_0 and θ_1 simultaneously

$$temp0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$

$$temp1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\,x^{(i)}$$

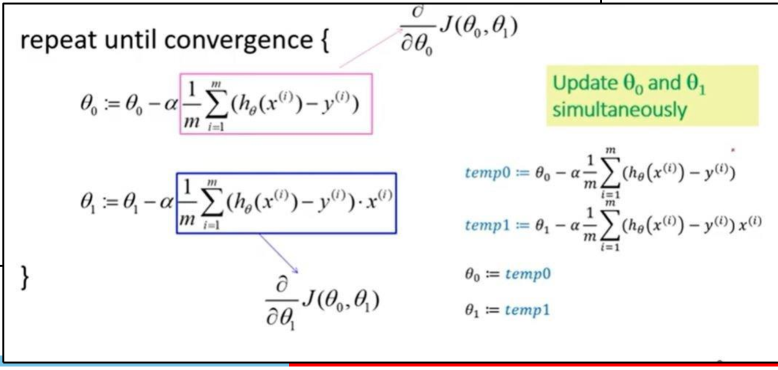
$$\theta_0 := temp0$$

$$\theta_1 := temp1$$

Gradient Descent: Variants

- **Stochastic gradient** descent refers to calculating the derivative from each training data instance and calculating the update immediately

```
Randomly shuffle training instances
Initialize the Parameters ($\theta_0^1$, $\theta_1^1$ , …..)
K=1
Repeat until Convergence {
    Sample with replacement, only one random training instance "i" at a time
        $\theta_0^{k+1} = \theta_0^k - \alpha( h_\theta(x^{(i)}) - y^{(i)})$
        $\theta_1^{k+1} = \theta_1^k - \alpha( h_\theta(x^{(i)}) - y^{(i)}) * x_1^{(i)}$
        $\theta_2^{k+1} = \theta_2^k - \alpha( h_\theta(x^{(i)}) - y^{(i)}) * x_2^{(i)}$
        ……
        K=k+1
        }
}
return ($\theta_0^1$, $\theta_1^1$ , …..)
```



repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}}$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

}

Update θ_0 and θ_1 simultaneously

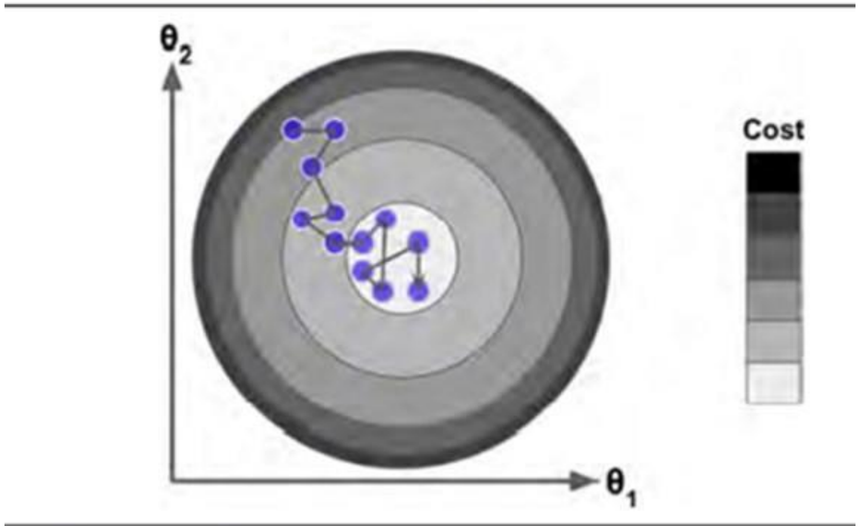
$temp0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$

$temp1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$

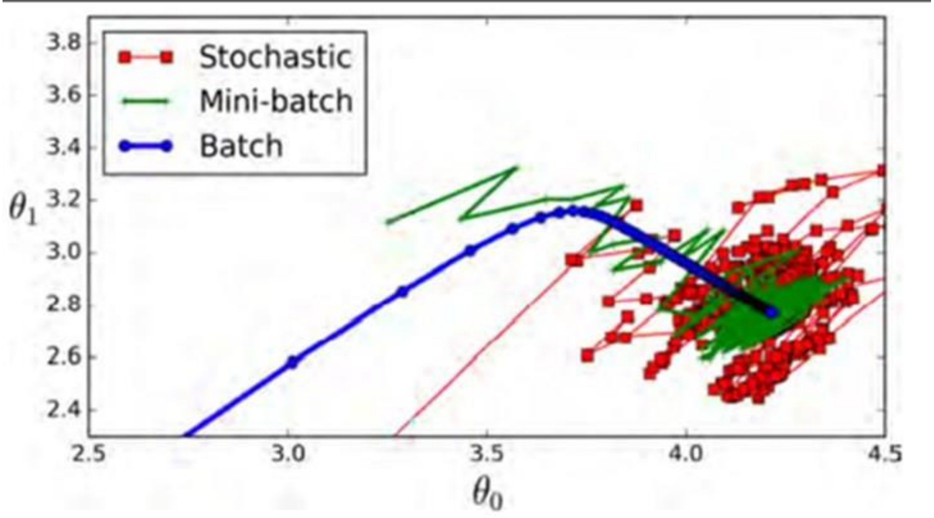
$\theta_0 := temp0$

$\theta_1 := temp1$

Gradient Descent: Variants



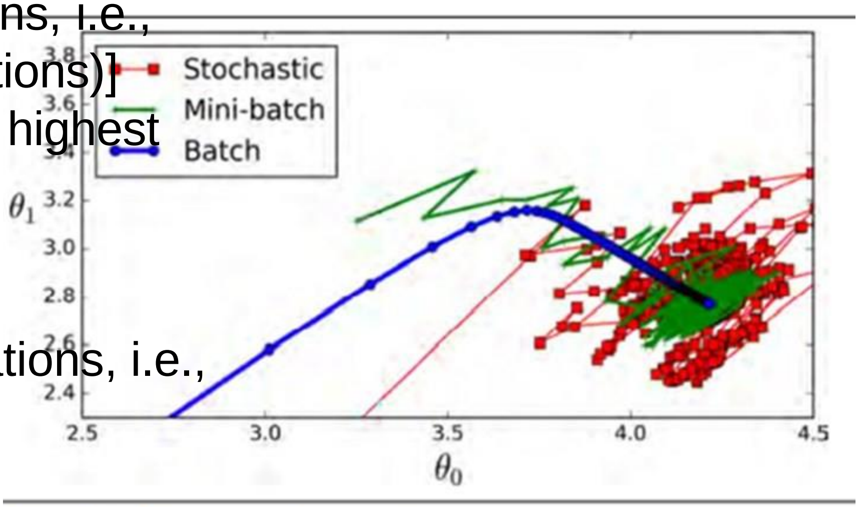
Stochastic Gradient Descent



1. Gradient Descent paths in parameter space

Gradient Descent: Variants

- Choice of batch size impacts the rate of convergence of gradient descent (GD)
- In Batch GD, entire training set is used to calculate the training error in each iteration/epoch and gradient is calculated and used for weight updates
  - Convergences in least number of iterations, i.e., rate of convergence is highest [*O*(1/iterations)]
  - Computation requirement per iteration is highest
  - Memory requirement is also highest
- In Stochastic gradient descent, a **randomly** selected training instance is used
  - Convergences in highest number of iterations, i.e., rate of convergence is slowest [~$O(1/\sqrt{iterations})$]
  - Computation requirement per iteration is lowest
  - Memory requirement is also lowest
- In mini batch GD, a subset of training data of size, say 64,128, 256 is used
  - very efficient implementation possible leveraging vector processing using GPUs



Gradient Descent paths in parameter space

Evaluation Metrics

Evaluation of Linear Regression Model

Mileage (in kmpl)	*Car Price (in cr)*
9.8	10.48
9.12	1.75
9.5	6.95
10	2.51
….	…..

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2}$$

Unseen Data

Mileage (in kmpl)	*Car Price (in cr)*
7.5	9.25
10	6.5
….	…..

$$\boxed{R^2 = 1 - \frac{SS_{residual}}{SS_{Total}}}$$

Model 1

CarPrice = 8.5 + 0.5 Mileage – 1.5 Mileage^2

Model 2

CarPrice = -5.5 + 1.5 Mileage

Evaluation of Linear Regression Model

R-squared

Mileage (in kmpl)	*Car Price (in cr)*
9.8	10.48
9.12	1.75
9.5	6.95
10	2.51
….	…..

variation in 'y' that is explained by a regression model

$$explained\ variation = \hat{y} - \bar{y}$$

variation in 'y' that is not captured/explained by a regression

$$unexplained\ variation = y - \hat{y}$$

$$total\ variation = (y - \hat{y}) + (\hat{y} - \bar{y}) = (y - \bar{y})$$

Car Price (in cr)
10.48
1.75
6.95
2.51
Mean Y

$$\boxed{R^2 = 1 - \frac{SS_{residual}}{SS_{Total}}}$$

$$SS_{explained} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$$

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

$$SS_{Total} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Model 1

CarPrice = 8.5 + 0.5 Mileage – 1.5 Mileage^2

$where:$

$SS_{explained} = explained\ variation\ sum\ of\ squares$

$SS_{residual} = unexplained\ variation\ sum\ of\ squares$

$SS_{Total} = total\ variation\ sum\ of\ squares$

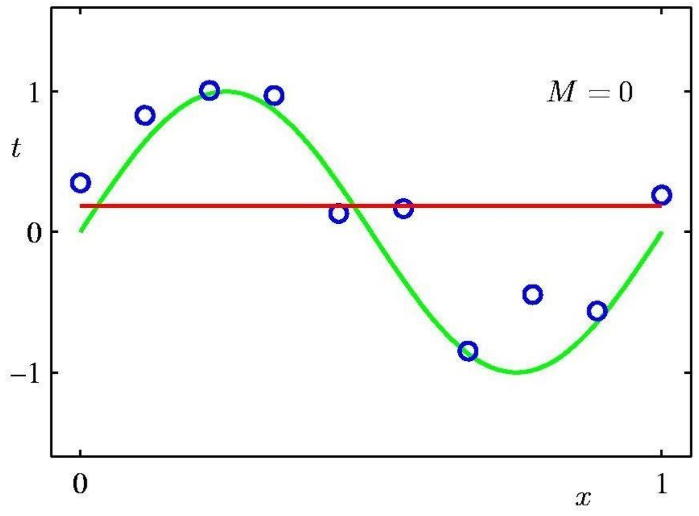
- **variation that is explained by a regression model**
- **measures the goodness of fit of a regression model**

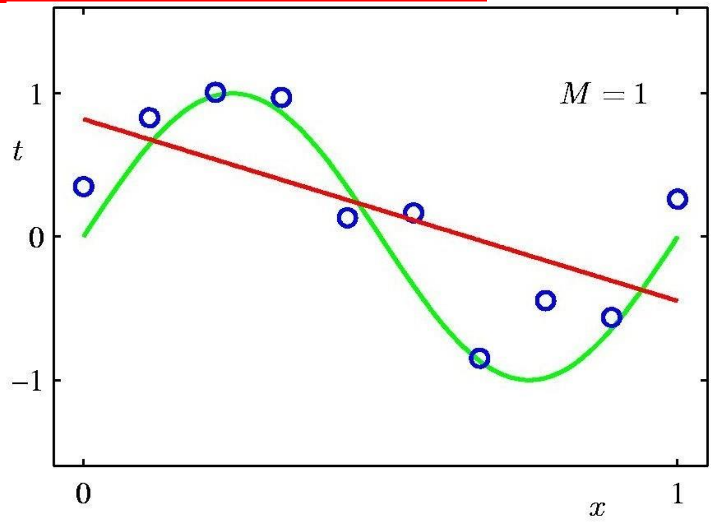
Linear Basis Models

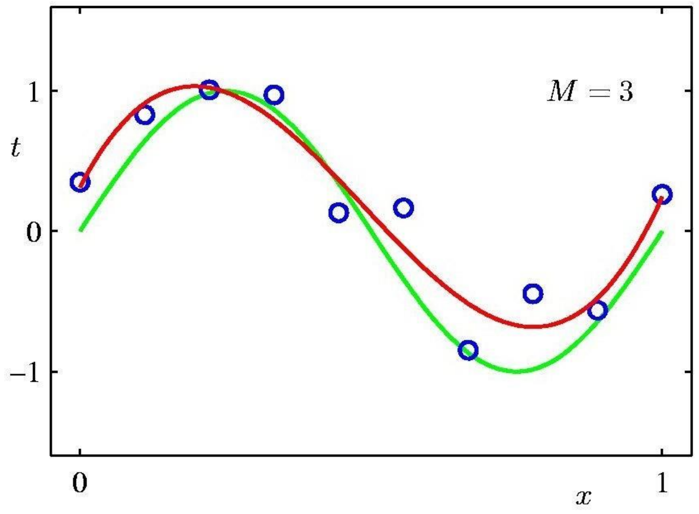
What if output is a non-linear function of input vector?

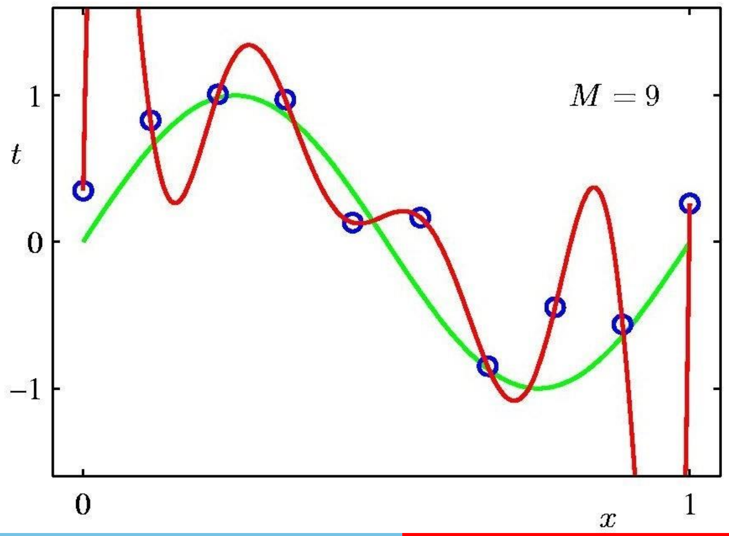
BITS Pilani

Polynomial Regression









Linear Basis Function Models

- The inputs **X** for linear regression can be:
  - Original quantitative inputs
  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.
  - Polynomial transformation
    - example: $y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot x^3$
  - Basis expansions
  - Dummy coding of categorical inputs
  - Interactions between variables
    - example: $x_3 = x_1 \cdot x_2$

This allows use of linear regression techniques to fit non-linear datasets.

Input X	Output Y
exp(2)	----
exp(4)	
exp(6.3)	
exp(9.2)	

X No.of.Years of Experience (in Years)	X^2	**Y Salary Of the Employee (in Lakhs)**
1	1	2
2	4	3
3	9	4
4	16	5
5	25	6

X1 = Graduate	**X2 = PostGraduate**	**X3 = Others**	**Y Salary Of the Employee**
0	0	1	2
1	0	0	3
0	0	1	4
0	1	0	5
1	0	0	6

Linear Basis Function Models

X No.of.Years of Experience (in Years)	X^2	Y Salary Of the Employee (in Lakhs)
1	1	2
2	4	3
3	9	4
4	16	5
5	25	6

[redacted]

Example: an M-th order polynomial function of one dimensional feature x:

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M} w_j x^j$$

where x^j = j-th power of x

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$\Phi_j(x)$ are known as *basis functions*. Typically, $\Phi_0(x) = 1$, so that w_0 acts as a bias.
In the simplest case, we use linear basis functions : $\Phi_d(x) = x_d$.

They are called linear models because this function is

linear in **w**.

Linear Basis Function Models

Simplest linear model for regression is one that involves a linear combination of the input variables

$$y(x,w)=w_0 + w_1 x_1 + \ldots + w_D x_D$$

Extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(x,w)=w_0 + \sum_{j=1}^{M-1} w_j \varphi_j(x)$$

where $\varphi_j(x)$ are known as basis functions.

By denoting the maximum value of the index j by M −1, the total number of parameters in this model will be M.

Convenient to define an additional dummy 'basis function' $\varphi_0(x)=1$. So,

$$y(x,w) = \sum_{j=1}^{M-1} w_j \varphi_j(x) = \mathbf{w}^T \boldsymbol{\varphi}_\mathbf{j}(\mathbf{x})$$

where $\mathbf{w} = (w_0, \ldots, w_{M-1})^T$ and $\boldsymbol{\varphi} = (\varphi_0, \varphi_1, \ldots \ldots \varphi_n)$

If the original variables comprise the vector x, then the features can be expressed in terms of the basis functions {φj(x)}

Linear Basis Function Models

- Generally,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \underbrace{\phi_j(\boldsymbol{x})}_{\text{basis function}}$$

- Typically, $\phi_0(\boldsymbol{x}) = 1$ so that θ_0 acts as a bias
- In the simplest case, we use linear basis functions :

$$\phi_j(\boldsymbol{x}) = x_j$$

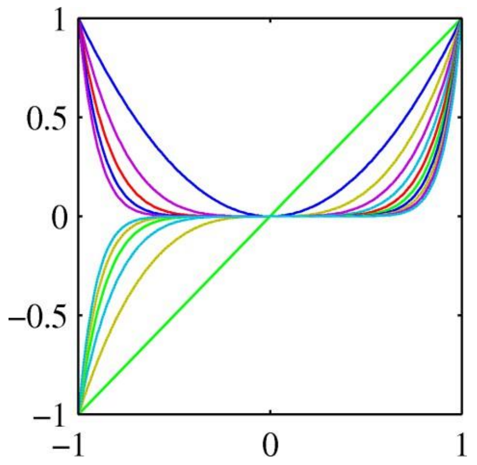
Based on slide by Christopher Bishop (PRML)

Linear Basis Function Models

- Basic Linear Model: $$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$
- Generalized Linear Model: $$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \phi_j(\boldsymbol{x})$$
- Once we have replaced the data by the outputs of the basis functions, fitting the generalized model is exactly the same problem as fitting the basic model
  - Unless we use the kernel trick – more on that when we cover support vector machines
  - Therefore, there is no point in cluttering the math with basis functions

Based on slide by Geoff Hinton

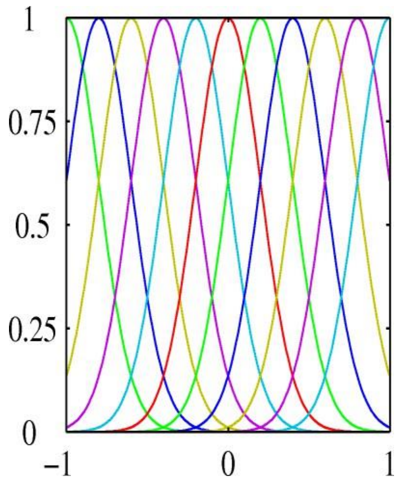
Linear Basis Function Models - Examples



Polynomial basis functions:

$$\phi_j(x) = x^j.$$

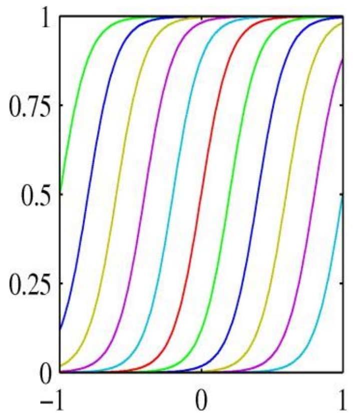
These are global; a small change in *x* affect all basis functions.



Gaussian basis functions:

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

These are local; a small change in *x* only affect nearby basis functions. μ_j and *s* control location and scale (width).



Sigmoidal basis functions:

where $\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$

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

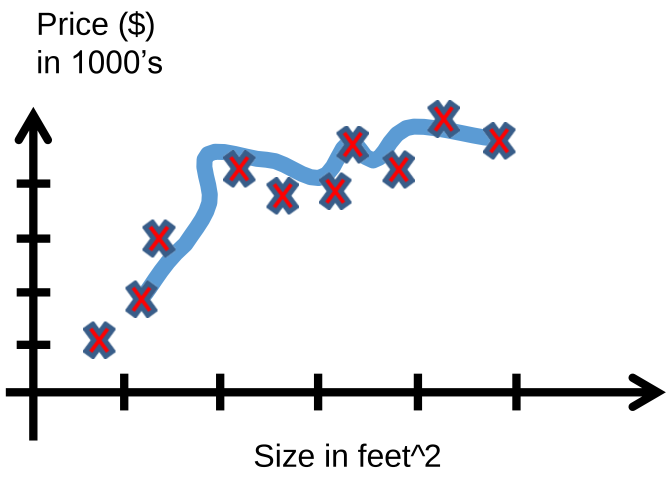
Also these are local; a small change in *x* only affect nearby basis functions. μ_j and *s* control location and scale (slope).

Notion of Bias - Variance

BITS Pilani

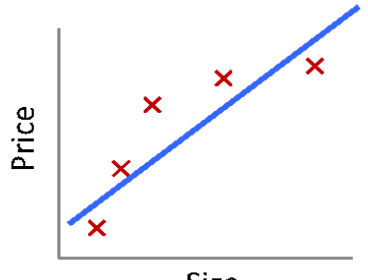
Addressing overfitting

- x_1 = size of house
- x_2 = no. of bedrooms
- x_3 = no. of floors
- x_4 = age of house
- x_5 = average income in neighborhood
- x_6 = kitchen size
- $\vdots$
- x_{100}



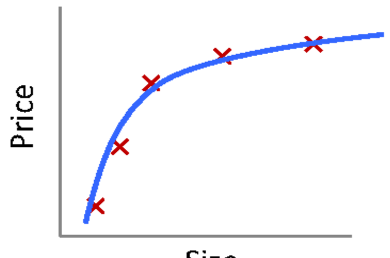
Slide credit: Andrew Ng

Quality of Fit



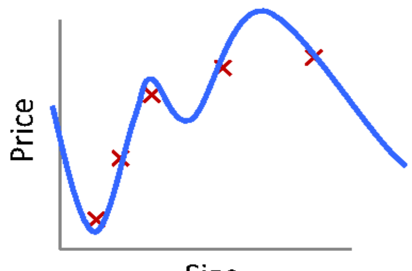
$\theta_0 + \theta_1 x$

Underfitting
(high bias)



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

Correct fit



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

Overfitting
(high variance)

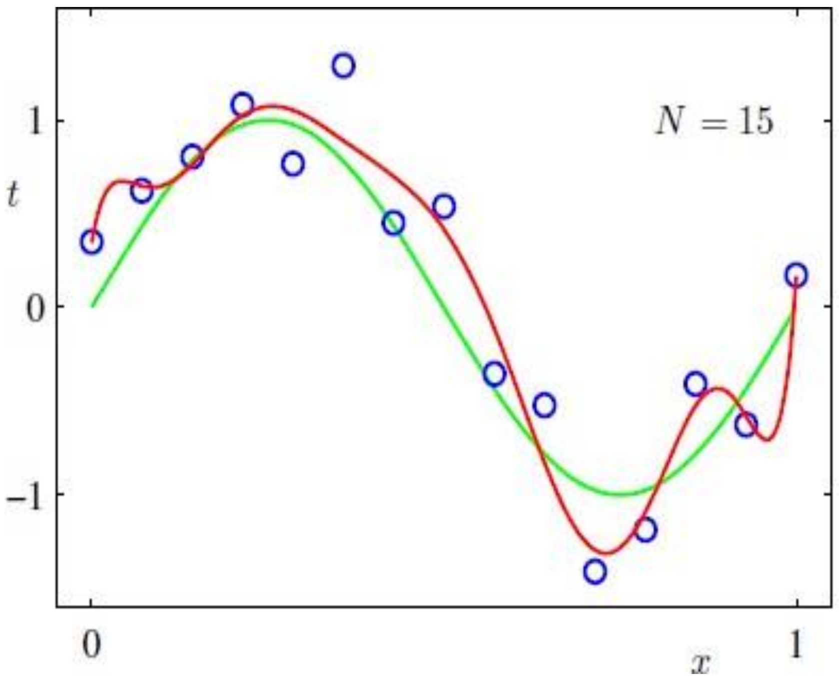
Overfitting:

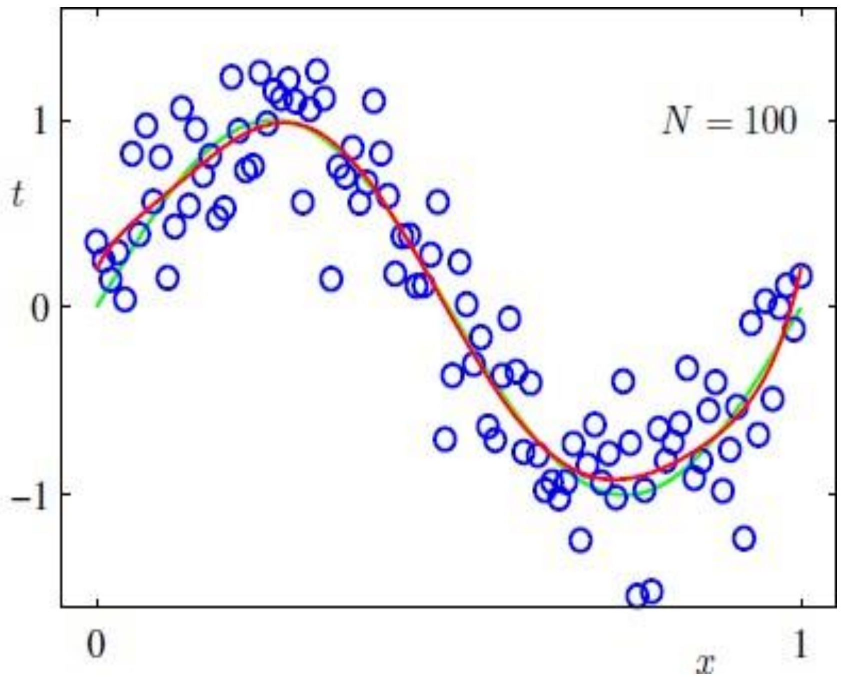
- The learned hypothesis may fit the training set very well ($J(\boldsymbol{\theta}) \approx 0$)
- ...but fails to generalize to new examples

Based on example by Andrew Ng

Handling Overfitting – Way 1

Increase in Size of the data set reduces the over-fitting problem



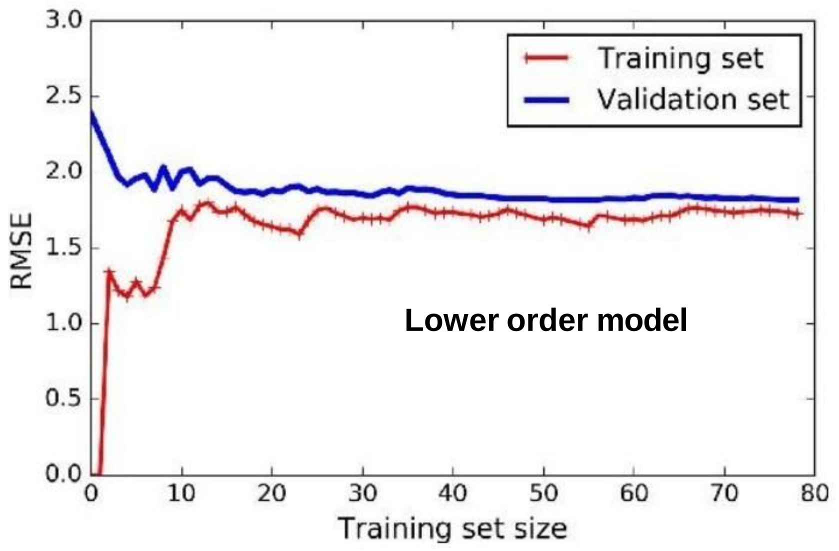


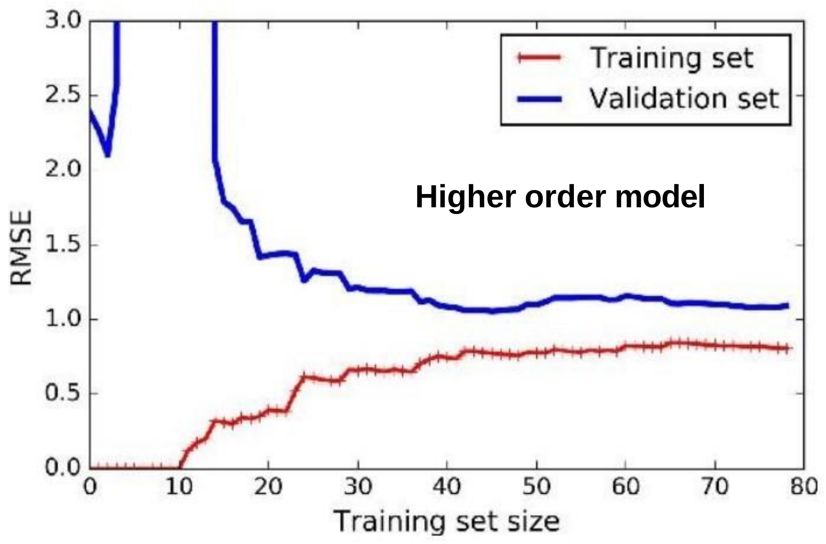
BITS Pilani, Pilani Campus

Effect of Training Size on Over fitting

Problem Type 4 : Interpretation of the Model Fit

- Size of training dataset needs to be large to prevent when higher order model is used.

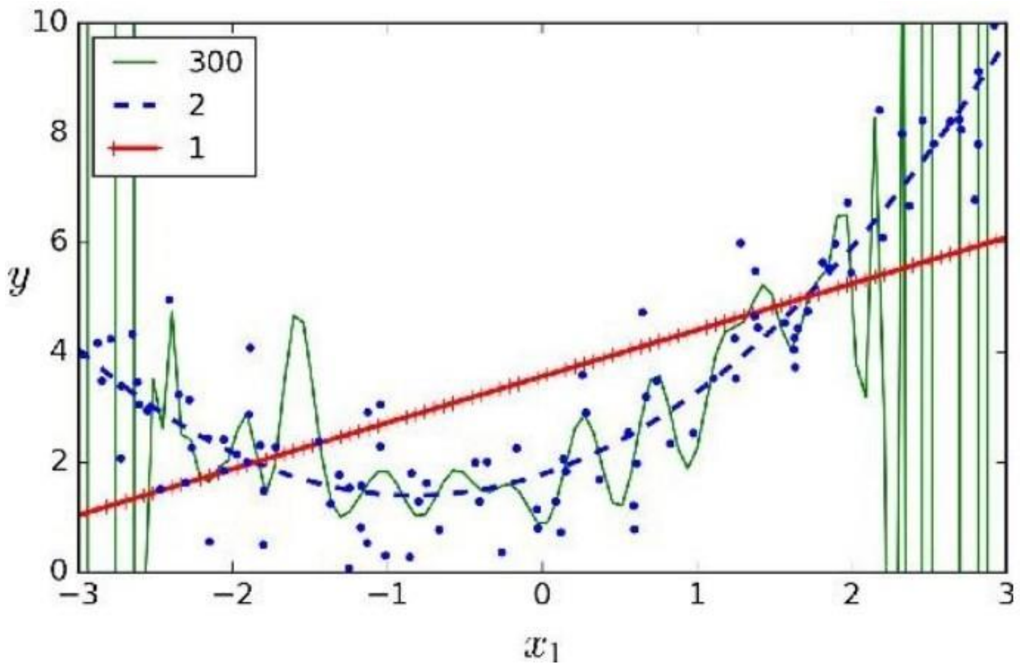




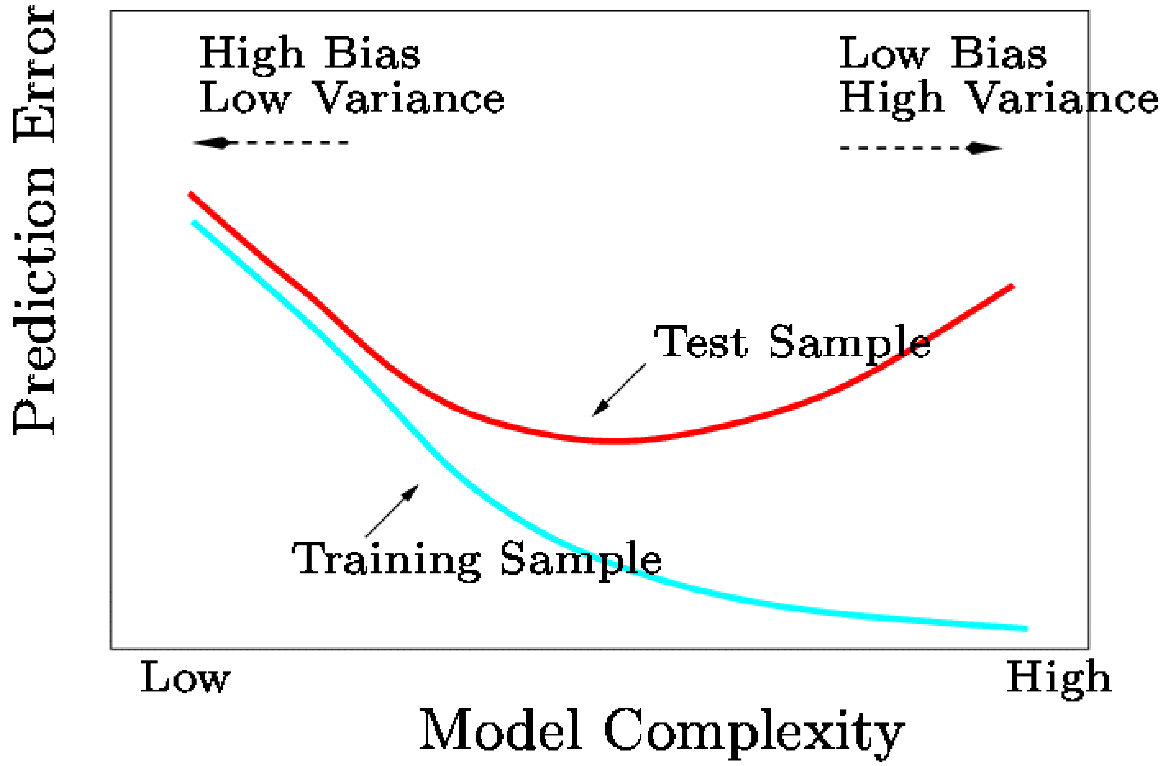
Polynomial Fitting can lead to Over fitting

Handling Overfitting – Way 2 – Reduce the complexity of the model

- Underlying target function is quadratic
- Linear model results in under fitting with large bias
- Polynomial of order 300 results in a large variance

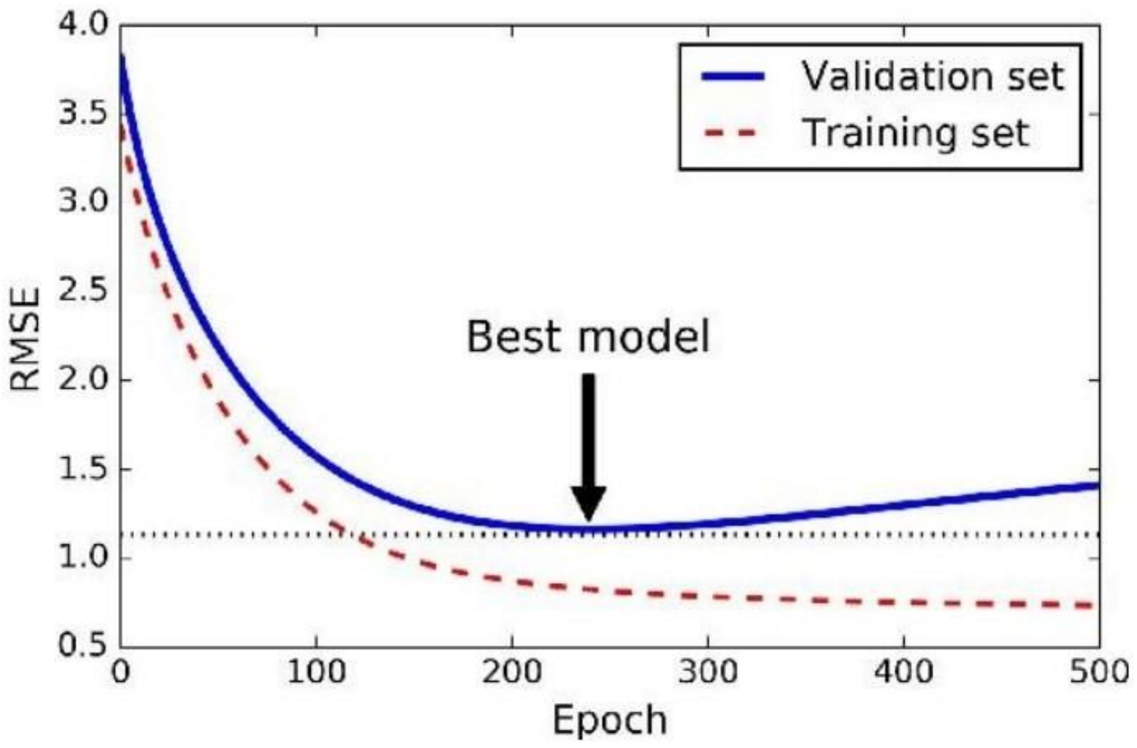


How to experiment on Model Complexity ?



Handling Overfitting – Way 3

Early Stop the Training



Stop training once error on the validation set starts showing an upward trend, even if the error on the training set keeps decreasing

Regularization

(*This notion is common for both Linear Regression – Module 3 and Logistic Regression – Module 4)

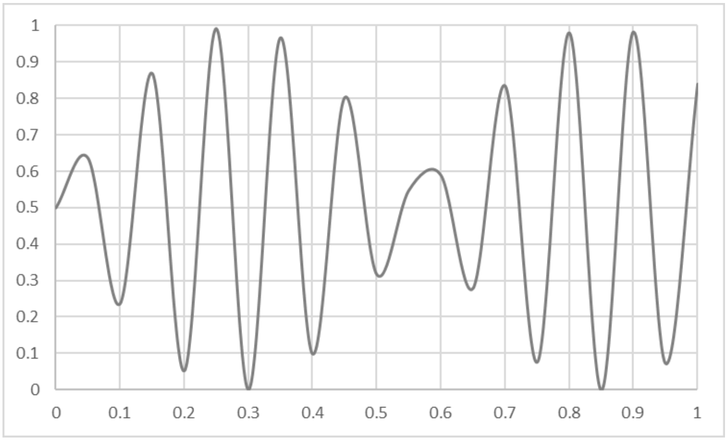
Overfitting vs Underfitting

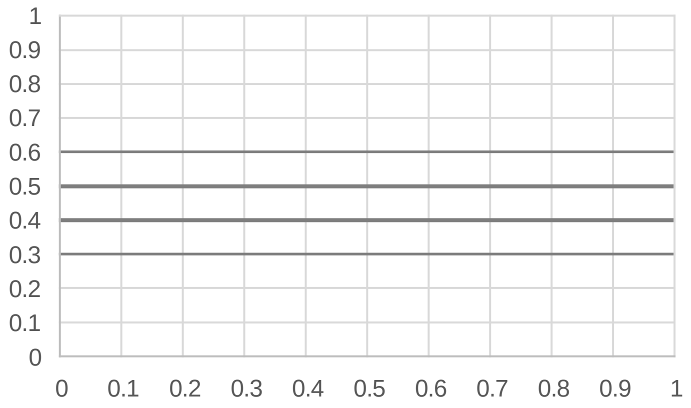
Overfitting

- Fitting the data too well
  - Features are noisy / uncorrelated to concept

Underfitting

- Learning too little of the true concept
  - Features don't capture concept
  - Too much bias in model





Regularization

- A method for automatically controlling the complexity of the learned hypothesis
- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label
- Can also address overfitting by eliminating features (either manually or via model selection)

Regularization

- Linear regression objective function

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

  – λ is the regularization parameter ($\lambda \geq 0$)
  – No regularization on θ_0!

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Note that $\sum_{j=1}^{d}\theta_j^2 = \|\boldsymbol{\theta}_{1:d}\|_2^2$
  - This is the magnitude of the feature coefficient vector!

- We can also think of this as:

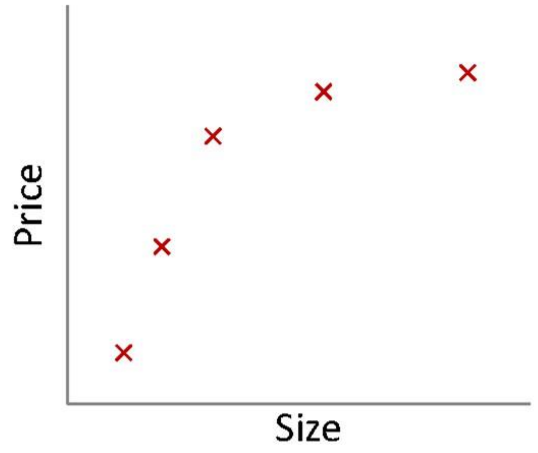
$$\sum_{j=1}^{d}(\theta_j - 0)^2 = \|\boldsymbol{\theta}_{1:d} - \vec{\mathbf{0}}\|_2^2$$

  - L_2 regularization pulls coefficients toward 0

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



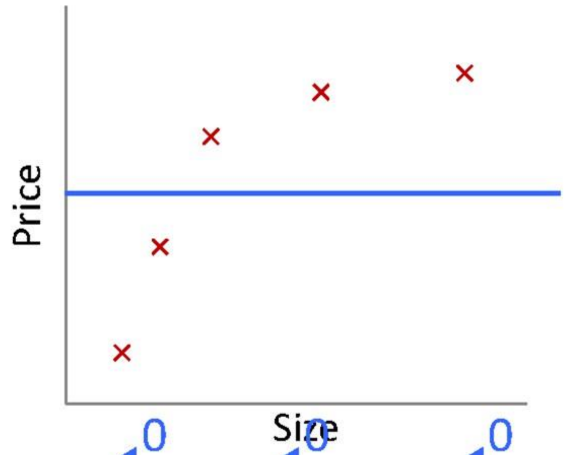
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Based on example by Andrew Ng

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \overset{0}{\cancel{\theta_1}} x + \overset{0}{\cancel{\theta_2}} x^2 + \overset{0}{\cancel{\theta_3}} x^3 + \overset{0}{\cancel{\theta_4}} x^4$$

Based on example by Andrew Ng

Regularization

Ridge Regression / Tikhonov regularization

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$$\frac{\partial}{\partial\theta_0}J(\theta) \qquad \theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\frac{\partial}{\partial\theta_j}J(\theta) \qquad \theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \underbrace{- \lambda\theta_j}_{\text{regularization}}$$

- We can rewrite the gradient step as:

$$\theta_j \leftarrow \theta_j\,(1-\alpha\lambda) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Regularization

Lasso Regression (Least Absolute Shrinkage and Selection Operator Regression)

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{d}|\theta_j|$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$\frac{\partial}{\partial\theta_0}J(\theta)$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$\frac{\partial}{\partial\theta_j}J(\theta)$

$$\theta_j = \theta_j - \frac{\alpha}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\boldsymbol{x}_j^{(i)} - \alpha\lambda\,\mathrm{sign}(\theta_j)$$

regularization

$$\text{where}\quad \mathrm{sign}(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

Regularization

Elastic Net

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + r\,\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2 + \frac{(1-r)}{2}\,\lambda\sum_{j=1}^{d}|\theta_j|$$

Control the regularization using **the Mix Ration “r”:**
When,
r = 0, Elastic Net is equivalent to Ridge Regression,
r = 1, it is equivalent to Lasso Regression

from sklearn.linear_model import ElasticNe
ElasticNet(alpha=0.1, l1_ratio=0.5)

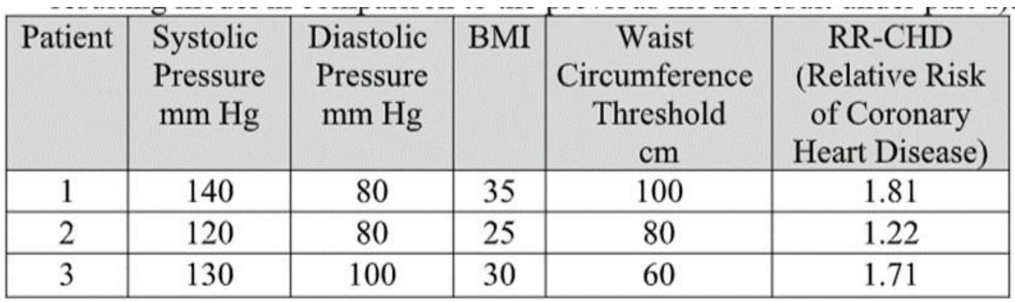
How to choose the right Regularization?

Common Usage & Observation

- L1 regularization has the ability to set some coefficients to 0 exactly leading to a *sparse* model
- L1 regularization helps in feature selection by eliminating the features that are not important
- **L1 cannot be used efficiently in gradient-based approaches since it is not-differentiable unlike L2**
- L2 will in general lead to small magnitudes of weights but not exactly 0.
- Elastic net – Prefer in Highly correlated features

Numerical Exercise – For Student Practice

Fit a linear regression. Show only the first iteration of Gradient descent algorithm using learning rate of **0.02** for the following data , if the Relative Risk of Coronary Heart Disease is believed to be only linearly dependent on BMI as well as Diastolic Pressure. Assume the intercept of the regression model as **5** and the slope of independent variables as **-0.03 (negative)**.



Patient	Systolic Pressure mm Hg	Diastolic Pressure mm Hg	BMI	Waist Circumference Threshold cm	RR-CHD (Relative Risk of Coronary Heart Disease)
1	140	80	35	100	1.81
2	120	80	25	80	1.22
3	130	100	30	60	1.71

Apply a regularization on the same problem for 2 iterations & interpret the results. Try both ridge regression as well as lasso regression. Below equation is changed only for ridge regression

Steps :

1. Identification of the equations y = w0+w1X1+W2X2
2. Cost function & derivative
   1. W0` = w0 – 1/3 * learning rate * (sum (w0+w1X1+W2X2 - y))
   2. W1` = w1* (1- learning rate * regularization constant) – 1/3 * learning rate * (sum (w0+w1X1+W2X2 - y) * x1)
   3. W2` = w2 * (1- learning rate * regularization constant) – 1/3 * learning rate * (sum (w0+w1X1+W2X2 - y)*x2)
1. Apply the equations

Python Lab Exercise – For Student Practice

Go to your virtual lab file under : Machine Learning → LabCapsule 2 Linear _ Polynomial Regression → 3 Polynomial Regression

Download ML_Lab 5 PolynomialRegression.ipynb

Try to change the degree to {2, 5, 7, 10} in the below function line : PolynomialFeatures(degree=3)

Observe and interpret on the performance of the training data

Note: Training vs Test Data split is not coded in this implementation. Below are the suggestions to experiment further:

- Add few hundreds of data (Use synthetic data generation python libraries available : Refer here)
- Split the data into 80% train vs 20% test set
- Built the polynomial model using above four different degree on training set
- For each of the model apply in the test set
- Find the MSE for both training data and separately for test data
- Plot the four experiment results in a plot and interpret the notion of overfit vs underfit
- Suggest which degree is a best fit .

References

- T1 - Chapter 1 – Machine Learning, Tom Mitchell
- Chapter 1, 2 – Introduction to Machine Learning, 2nd edition, Ethem Alpaydin
- R1 – Chapter #1, # 3,#4 (Christopher M. Bhisop, Pattern Recognition & Machine Learning) & Refresh your MFDS course basics

Thank you
!