Hi everyone good afternoon!

EE2211 Introduction to Machine Learning

T14 & T22, Chua Dingjuan elechuad@nus.edu.sg

Materials @ tiny.cc/ee2211tut

Tutorial 7

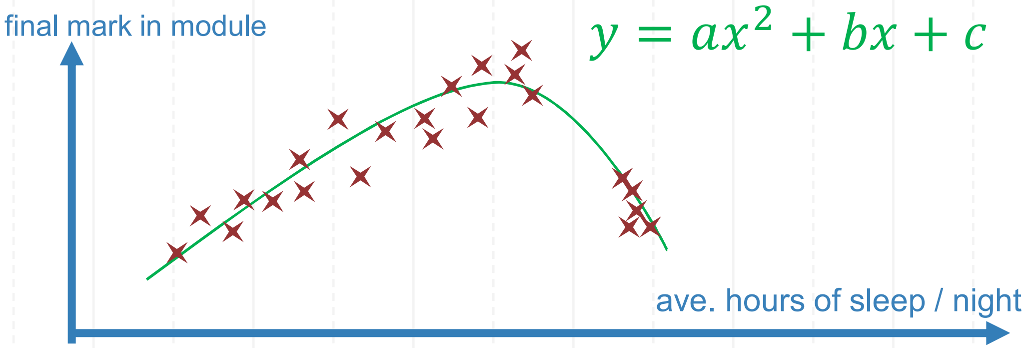
Over/Underfitting, Regularization (Bias, Variance)

Recall... Polynomial Regression

	1	… *d*	*y*	
	ave. hrs of sleep / night	final mark in module		
x_1	5	23		y_1
x_2	4	45		y_2
x_3	3	89		y_3
x_4	8	46		y_4
x_5	9	90		y_5
…	8.5	80		…
x_m	13	30		y_m

$$\boldsymbol{y} = \boldsymbol{P}\boldsymbol{w}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} c \\ b \\ a \end{bmatrix} \Rightarrow \begin{bmatrix} 23 \\ 45 \\ \dots \end{bmatrix} = \begin{bmatrix} 1 & 5 & 25 \\ 1 & 4 & 16 \\ 1 & \dots & \dots \end{bmatrix} \begin{bmatrix} c \\ b \\ a \end{bmatrix}$$



In polynomial regression, the system becomes "underdetermined" easily.

Potential for over/under fitting!

Ridge Regression / Regularization

Primal over, λI	Dual Form under, λI
$\hat{w} = \left(X^T X + \lambda I\right)^{-1} X^T y$	$\hat{w} = X^T\left(X X^T + \lambda I\right)^{-1} y$

- In ridge regression, an additional term is added during minimization:

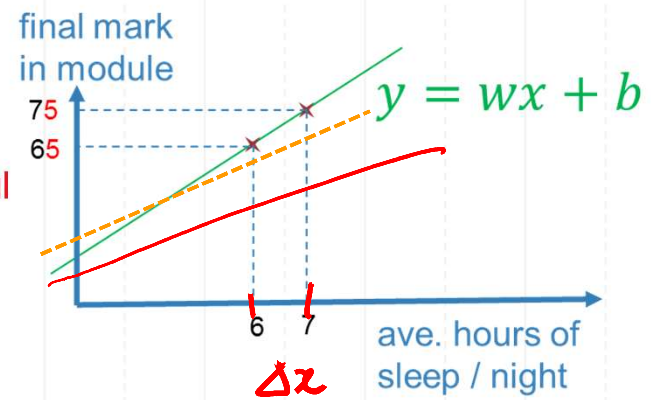
$$\min_{\mathbf{w}} \sum_{i=1}^{m} (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

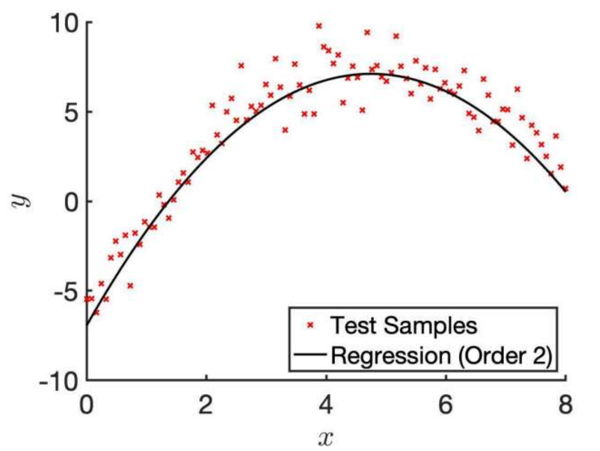
- In applications where number of samples m, much smaller than d (polynomial regression is one example), ridge regression can be useful

- Effect of λ reduces **w** ➔ same Δx → smaller Δy
➔ predictions are less sensitive to training data ➔ training error ↑

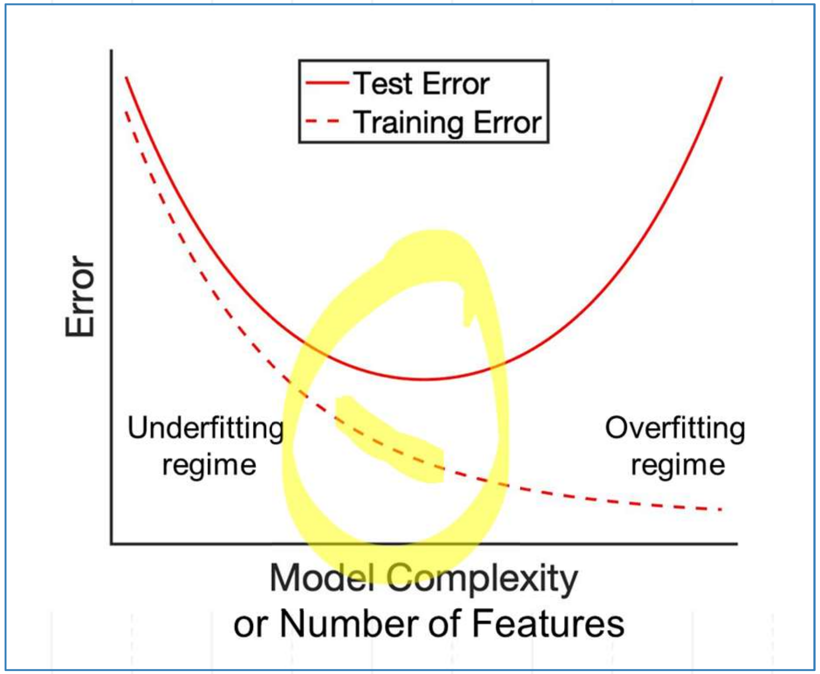
↓

testing error ↓





	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good



1) Underfitting is the inability of trained model to predict the targets in the training set

2) Overfitting occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly -- > (Feature Selection)

Bias-Variance Decomposition Theorem

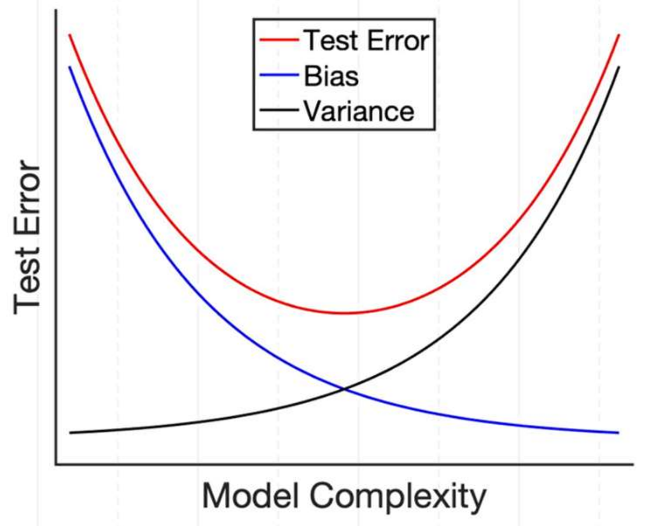
- Test error = Bias Squared + Variance + Irreducible Noise

- Variance :

Variability of prediction models across different training sets / trained models

- Bias :

How well an average prediction model will perform



Discussion of Solutions

7

Q1,2

Question1

This question explores the use of Pearson's correlation as a feature selection metric. What are the top two features we should select if we use Pearson's correlation as a feature selection metric?

	Datapoint 1	Datapoint 2	Datapoint 3	Datapoint 4	Datapoint 5
Feature 1	0.3510	2.1812	0.2415	-0.1096	0.1544
Feature 2	1.1796	2.1068	1.7753	1.2747	2.0851
Feature 3	-0.9852	1.3766	-1.3244	-0.6316	-0.8320
Target y	0.2758	1.4392	-0.4611	0.6154	1.0006

$$\text{r} = \frac{\overbrace{\frac{1}{N}\sum_{n=1}^{N}(a_i-\bar{a})(b_i-\bar{b})}^{\text{covariance}}}{\underbrace{\sqrt{\frac{1}{N}\sum_{n=1}^{N}(a_i-\bar{a})^2}}_{\text{std dev of a}}\underbrace{\sqrt{\frac{1}{N}\sum_{n=1}^{N}(b_i-\bar{b})^2}}_{\sigma_3}},$$

$$\text{Mean of Feature 1} = \mu_1 = \frac{0.3510+2.1812+0.2415-0.1096+0.1544}{5} = 0.5637$$

$$\text{Mean of Feature 2} = \mu_2 = \frac{1.1796+2.1068+1.7753+1.2747+2.0851}{5} = 1.6843$$

$$\text{Mean of Feature 3} = \mu_3 = \frac{-0.9852+1.3766-1.3244-0.6316-0.8320}{5} = -0.4793$$

$$\text{Mean of Target y} = \mu_y = \frac{0.2758+1.4392-0.4611+0.6154+1.0006}{5} = 0.5740$$

$$\text{Feature 1 std} = \sigma_1 = \sqrt{\frac{(0.3510-\mu_1)^2+(2.1812-\mu_1)^2+(0.2415-\mu_1)^2+(-0.1096-\mu_1)^2+(0.1544-\mu_1)^2}{5}} = 0.8229$$

$$\text{Feature 2 std} = \sigma_2 = \sqrt{\frac{(1.1796-\mu_2)^2+(2.1068-\mu_2)^2+(1.7753-\mu_2)^2+(1.2747-\mu_2)^2+(2.0851-\mu_2)^2}{5}} = 0.3924$$

$$\text{Feature 3 std} = \sigma_3 = \sqrt{\frac{(-0.9852-\mu_3)^2+(1.3766-\mu_3)^2+(-1.3244-\mu_3)^2+(-0.6316-\mu_3)^2+(-0.8320-\mu_3)^2}{5}} = 0.9552$$

$$\text{Target y std} = \sigma_y = \sqrt{\frac{(0.2758-\mu_y)^2+(1.4392-\mu_y)^2+(-0.4611-\mu_y)^2+(0.6154-\mu_y)^2+(1.0006-\mu_y)^2}{5}} = 0.6469$$

Question1

This question explores the use of Pearson's correlation as a feature selection metric. What are the top two features we should select if we use Pearson's correlation as a feature selection metric?

	Datapoint 1	Datapoint 2	Datapoint 3	Datapoint 4	Datapoint 5
Feature 1	0.3510	2.1812	0.2415	-0.1096	0.1544
Feature 2	1.1796	2.1068	1.7753	1.2747	2.0851
Feature 3	-0.9852	1.3766	-1.3244	-0.6316	-0.8320
Target y	0.2758	1.4392	-0.4611	0.6154	1.0006

covariance

$$r = \frac{\frac{1}{N}\sum_{n=1}^{N}(a_i-\bar{a})(b_i-\bar{b})}{\underbrace{\sqrt{\frac{1}{N}\sum_{n=1}^{N}(a_i-\bar{a})^2}}_{\text{std dev of } a}\underbrace{\sqrt{\frac{1}{N}\sum_{n=1}^{N}(b_i-\bar{b})^2}}_{\sigma_3}},$$

Cov(Feature 1, y) = $\frac{1}{5}[(0.3510-\mu_1)(0.2758-\mu_y)+(2.1812-\mu_1)(1.4392-\mu_y)+(0.2415-\mu_1)(-0.4611-\mu_y)+(-0.1096-\mu_1)(0.6154-\mu_y)+(0.1544-\mu_1)(1.0006-\mu_y)] = 0.3188$

Cov(Feature 2,y) = $\frac{1}{5}[(1.1796-\mu_2)(0.2758-\mu_y)+(2.1068-\mu_2)(1.4392-\mu_y)+(1.7753-\mu_2)(-0.4611-\mu_y)+(1.2747-\mu_2)(0.6154-\mu_y)+(2.0851-\mu_2)(1.0006-\mu_y)] = 0.1152$

Cov(Feature 3,y) = $\frac{1}{5}[(-0.9852-\mu_3)(0.2758-\mu_y)+(1.3766-\mu_3)(1.4392-\mu_y)+(-1.3244-\mu_3)(-0.4611-\mu_y)+(-0.6316-\mu_3)(0.6154-\mu_y)+(-0.8320-\mu_3)(1.0006-\mu_y)] = 0.4949$

Correlation of Feature 1 & y = $\frac{Cov(Feature\ 1,y)}{\sigma_1\sigma_y} = \frac{0.3188}{0.8229\times0.6469} = 0.5988$

Correlation of Feature 2 & y = $\frac{Cov(Feature\ 2,y)}{\sigma_2\sigma_y} = \frac{0.1152}{0.3924\times0.6469} = 0.4537$

Correlation of Feature 3 & y = $\frac{Cov(Feature\ 3,y)}{\sigma_3\sigma_y} = \frac{0.4949}{0.9552\times0.6469} = 0.8009$

Therefore, the top 2 features are Feature 1 and Feature 3.

Question1

SOLUTIONS – Possible Python Solutions as suggested by students!

```
#----------------------------------------------------------------------
#EE2211 Tutorial 7 Question 1
#by Chua Dingjuan
#----------------------------------------------------------------------

import numpy as np

x1 = np.array([0.3510, 2.1812, 0.2415, -0.1096, 0.1544])
x2 = np.array([1.1796, 2.1068, 1.7753, 1.2747, 2.0851])
x3 = np.array([-0.9852, 1.3766, -1.3244, -0.6316, -0.8320])
y=np.array([0.2758, 1.4392, -0.4611, 0.6154, 1.0006])


#corrcoef() returns the correlation matrix, which is a two-dimensional array
#with the correlation coefficients.
#here, corrcoef(x1,y) returns correlation of (x1,x1) , then (x1,y), then (y,x1)
#we only need either (x1,y) / (y,x1) --> r1[0][1]
r1 = np.corrcoef(x1,y)
r2 = np.corrcoef(x2,y)
r3 = np.corrcoef(x3,y)

print('Pearson correlation values are : ', r1[0][1], r2[0][1], r3[0][1])
```

Question2

This question further explores linear regression and ridge regression.
(a) Use the polynomial model from orders 1 to 6 to train and test the data without regularization.
Plot the Mean Squared Errors (MSE) over orders from 1 to 6 for both the training and the test sets.
Which model order provides the best MSE in the training and test sets? Why?
(b) Use Regularization λ=1

Training Data	Testing Data
$\{x=-10\} \rightarrow \{y=4.18\}$	$\{x=-9\} \rightarrow \{y=3\}$
$\{x=-8\} \rightarrow \{y=2.42\}$	$\{x=-7\} \rightarrow \{y=1.81\}$
$\{x=-3\} \rightarrow \{y=0.22\}$	$\{x=-5\} \rightarrow \{y=0.80\}$
$\{x=-1\} \rightarrow \{y=0.12\}$	$\{x=-4\} \rightarrow \{y=0.25\}$
$\{x=2\} \rightarrow \{y=0.25\}$	$\{x=-2\} \rightarrow \{y=-0.19\}$
$\{x=7\} \rightarrow \{y=3.09\}$	$\{x=1\} \rightarrow \{y=0.4\}$
	$\{x=4\} \rightarrow \{y=1.24\}$
	$\{x=5\} \rightarrow \{y=1.68\}$
	$\{x=6\} \rightarrow \{y=2.32\}$
	$\{x=9\} \rightarrow \{y=5.05\}$

SOLUTIONS – Check your answers.

NO REGULARIZATION

Order	1	2	3	4	5	6
Training MSE	2.3071	8.4408e-03	8.3026e-03	1.7348e-03	3.8606e-25	2.3656e-17
Test MSE	3.0006	0.0296	0.0301	0.0854	1.0548	10.7674

REGULARIZATION

Order	1	2	3	4	5	6
Training MSE	2.3586	8.4565e-03	8.3560e-03	1.8080e-03	7.2650e-04	1.9348e-04
Test MSE	3.2756	0.0302	0.0314	0.0939	0.4369	6.0202

Question1

This question further explores linear regression and ridge regression.
(a) Use the polynomial model from orders 1 to 6 to train and test the data without regularization.
Plot the Mean Squared Errors (MSE) over orders from 1 to 6 for both the training and the test sets.
Which model order provides the best MSE in the training and test sets? Why?

Training Data

$\{x = -10\} \rightarrow \{y = 4.18\}$
$\{x = -8\} \rightarrow \{y = 2.42\}$
$\{x = -3\} \rightarrow \{y = 0.22\}$
$\{x = -1\} \rightarrow \{y = 0.12\}$
$\{x = 2\} \rightarrow \{y = 0.25\}$
$\{x = 7\} \rightarrow \{y = 3.09\}$

Testing Data

$\{x = -9\} \rightarrow \{y = 3\}$
$\{x = -7\} \rightarrow \{y = 1.81\}$
$\{x = -5\} \rightarrow \{y = 0.80\}$
$\{x = -4\} \rightarrow \{y = 0.25\}$
$\{x = -2\} \rightarrow \{y = -0.19\}$
$\{x = 1\} \rightarrow \{y = 0.4\}$
$\{x = 4\} \rightarrow \{y = 1.24\}$
$\{x = 5\} \rightarrow \{y = 1.68\}$
$\{x = 6\} \rightarrow \{y = 2.32\}$
$\{x = 9\} \rightarrow \{y = 5.05\}$

(a) SOLUTION (Similar to Tutorial 6 Q2(a) just with more orders!)

STEPS :: For each order from 1 to 6

- Create P matrix (polynomial matrix) — 10
- Solve for w (coefficients of regression polynomial, taking note of P's size). 9

MSE

- For x (training) and x (test), find the corresponding y on your regression line.
- Calculate mean squared errors between that and the y data provided.

Plotting

- Plot training data and testing data as points
- Plot your regression line.

Xtrain (6 data points, 6 rows)				
Order	1	2 ..	5	6
P cols	2	3..	6	7
P rows	6	6 ..	6	6

(a) SOLUTION (part 1 - solving)

- 6 training data points
- Polynomial orders 1 to 5 : Primal solution → $\hat{w} = (P^T P)^{-1} P^T y$
- Order 6 : 7 unknowns, under-determined, so we use the Dual solution → $\hat{w} = P^T (P P^T)^{-1} y$

STEPS :: For each order from 1 to 6

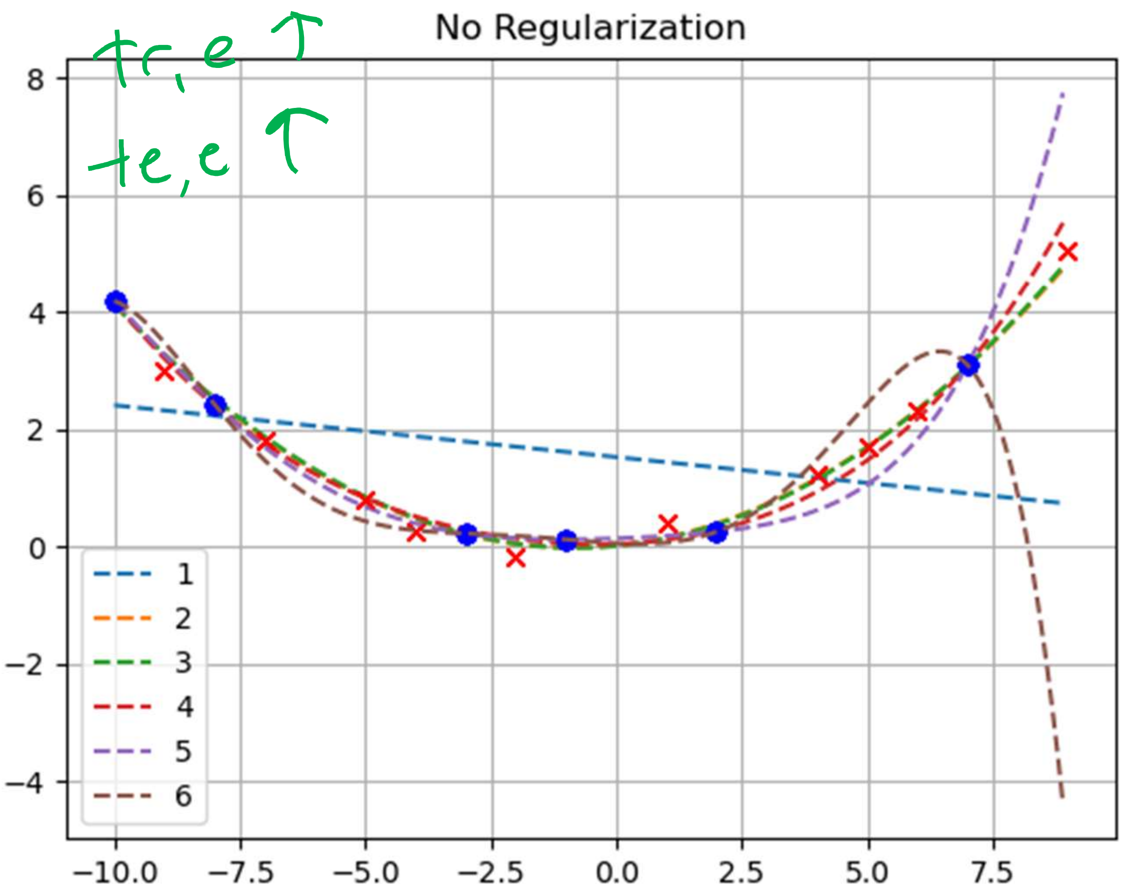
- Create P matrix (polynomial matrix)
- Solve for w (coefficients of regression polynomial, taking note of P's size).

```
# Create regressors
P_train_list = CreateRegressors(x, max_order)
print(P_train_list)
P_test_list  = CreateRegressors(xt, max_order)
P_plot_list  = CreateRegressors(x_plot, max_order)

#######################
# Q1 (a)
#######################

# Estimate coefficients WITHOUT REGULARIZATION
w_list = EstimateRegressionCoefficients(P_train_list, y)
```

```
def EstimateRegressionCoefficients(P_list, y, reg=None):

    # P_list is a list
    # P_list[i] are regressors for order i+1 and is of size N x (order+1), where
    # N is number of data points

    w_list = []
    if reg is None:

        for P in P_list:
            if(P.shape[1] > P.shape[0]): #use dual solution
                w = P.T @ inv(P @ P.T) @ y
            else: # use primal solution
                w = (inv(P.T @ P) @ P.T) @ y
            w_list.append(w)
    else:

        for P in P_list:
            w = (inv(P.T @ P + reg*np.eye(P.shape[1]) ) @ P.T) @ y
            w_list.append(w)

    return w_list
```

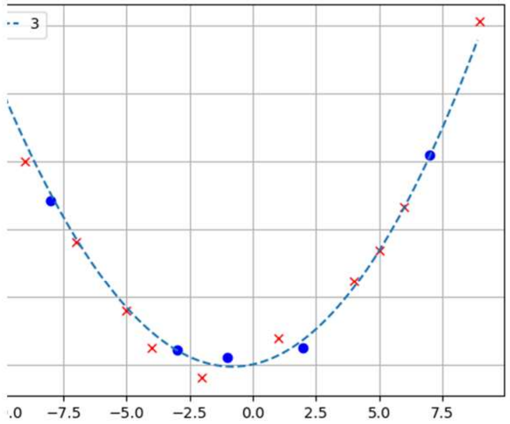
```
from sklearn.preprocessing import PolynomialFeatures as skpf

for order in range(1,7):

    #step 1, create a polynomial function object class with the right degree
    polyfn = skpf(order)

    #step 2, fit the polyfn to your actual data and test data.
    P = polyfn.fit_transform(x)

    #step 3, solving for coefficients of regression polynomial
    if (P.shape[0] >= P.shape[1]) :
        w = np.linalg.inv(P.T @ P) @ P.T @ y

    else : #P.shape[0] < P.shape[1], underdetermined systems
        w = P.T @ np.linalg.inv(P @ P.T) @ y
```

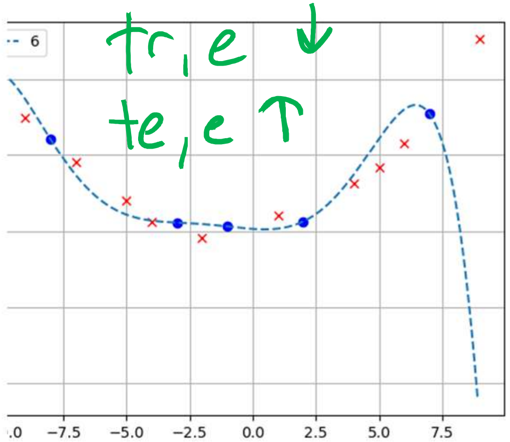
```
def CreateRegressors(x, max_order):

    # x is assumed to be array of length N
    # return P = list of regressors based on max_order
    # P[i] are regressors for order i+1 and is of size N x (order+1), where
    # N is number of data points

    P = [] #initialize empty list
    for order in range(1, max_order+1):
        current_regressors = np.zeros([len(x), order+1])
        current_regressors[:,0] = np.ones(len(x))
        for i in range(1, order+1):
            current_regressors[:,i] = np.power(x, i)
        P.append(current_regressors)

    return P
```



tr,e ↑

te,e ↑





tr,e ↓

te,e ↑

(a) SOLUTION (part 2 - mse)

MSE

- For x (training) and x (test), find the corresponding y on your regression line.
- Calculate mean squared errors between that and the y data provided.

```
#step 4, using regression coefficients to calculate predicted values of y
Ptest = polyfn.fit_transform(xtest)

y_reg = P @ w
ytest_reg= Ptest @ w

#step 5, calculating mse error for this order
#from sklearn.metrics import mean_squared_error as skmse

trainingmse.append(skmse(y,y_reg))
testmse.append(skmse(ytest,ytest_reg))
```

```
# Apply prediction: predictions are N x max_order
y_train_pred = PerformPrediction(P_train_list, w_list)
y_test_pred  = PerformPrediction(P_test_list, w_list)
y_plot_pred  = PerformPrediction(P_plot_list, w_list)
```

```
def PerformPrediction(P_list, w_list):

    # P_list is list of regressors
    # w_list is list of coefficients
    # Output is y_predict_mat which N x max_order, where N is the number of samples

    N = P_list[0].shape[0]
    max_order = len(P_list)
    y_predict_mat = np.zeros([N, max_order])
    for order in range(len(w_list)):
        y_predict = np.matmul(P_list[order], w_list[order])
        y_predict_mat[:,order] = y_predict

    return y_predict_mat
```

```
# Compute MSE
train_error = y_train_pred - np.matlib.repmat(y, max_order, 1).T
train_MSE   = np.power(train_error, 2)
train_MSE   = np.mean(train_MSE, 0)

test_error = y_test_pred - np.matlib.repmat(yt, max_order, 1).T
test_MSE   = np.power(test_error, 2)
test_MSE   = np.mean(test_MSE, 0)
```

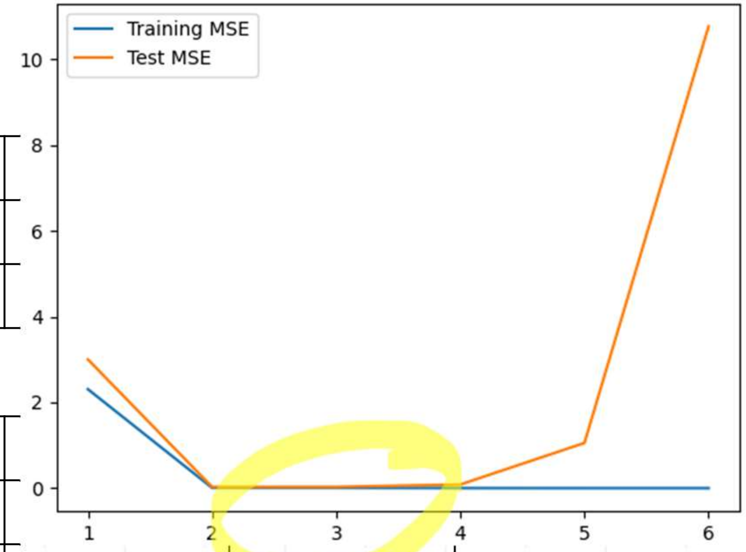
(a) SOLUTION (part 2 - mse)

Official Solutions

Order	1	2	3	4
Training MSE	2.3071	8.4408e-03	8.3026e-03	1.7348e-03
Test MSE	3.0006	0.0296	0.0301	0.0854

IDLE Compiler Produces the following for same code.

Order	1	2	3	4		
Training MSE	2.3071	8.4408e-03	8.3026e-03	1.7348e-03		
Test MSE	3.0006	0.0296	0.0301	0.0854	1.0548	10.7674



- Very low training MSE for orders 5 & 6 but high test MSE ➔ Overfitting
- High training and test MSE for order 1 ➔ Underfitting
- Orders 2 to 4 produce relatively low MSEs (even though model itself is order 2)

(a) SOLUTION (part 3 - plotting)

Training data

Test data

Regression Line

P Matrix

```
#--- Plotting---#
plt.figure(1)
plt.plot(x,y,'bo')
plt.plot(xtest,ytest,'rx')

xline= np.arange(min(np.concatenate((x,xtest))),max(np.concatenate((x,xtest),
xline_reshaped = xline.reshape((len(xline),1))
Pline = polyfn.fit_transform(xline_reshaped)
yline=Pline @ w
plt.plot(xline,yline,'--',  label=order)
plt.legend()
plt.grid()
```

Dotted Line

Training Errors

```
plt.figure(2)
plt.plot(np.arange(1,7),trainingmse,label="Training MSE")
plt.plot(np.arange(1,7),testmse,label="Test MSE")
plt.legend()

plt.show()
```

Question2 (b)

(b) Use Regularization (ridge regression) λ=1 for all orders and repeat the same analyses. Compare the plots of (a) and (b). What do you see?

(b) SOLUTION

- With regularization, we can simply use the primal solution even for order 6
- Only one small difference in code:

```
#step 3, solving for coefficients of regression polynomial
w = np.linalg.inv(P.T @ P + np.identity(len(P.T))) @ P.T @ y
```

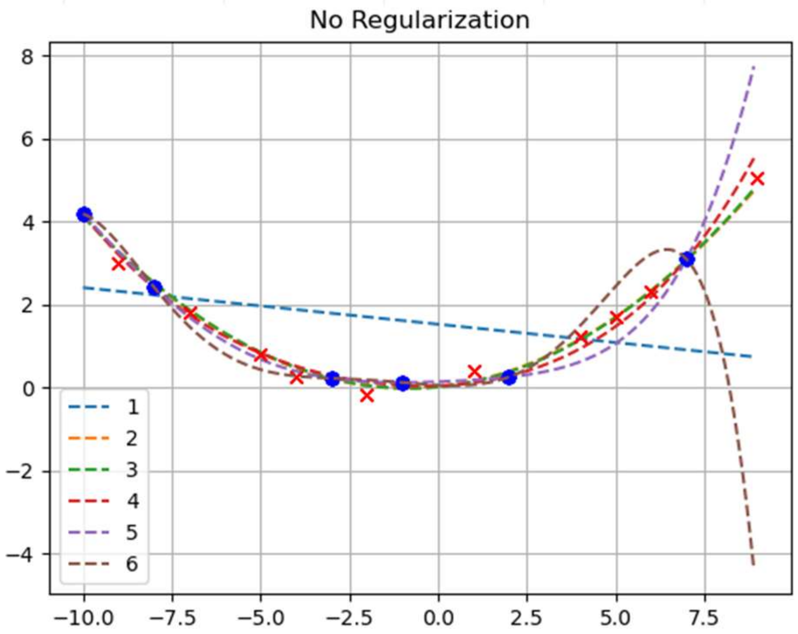
NO REGULARIZATION

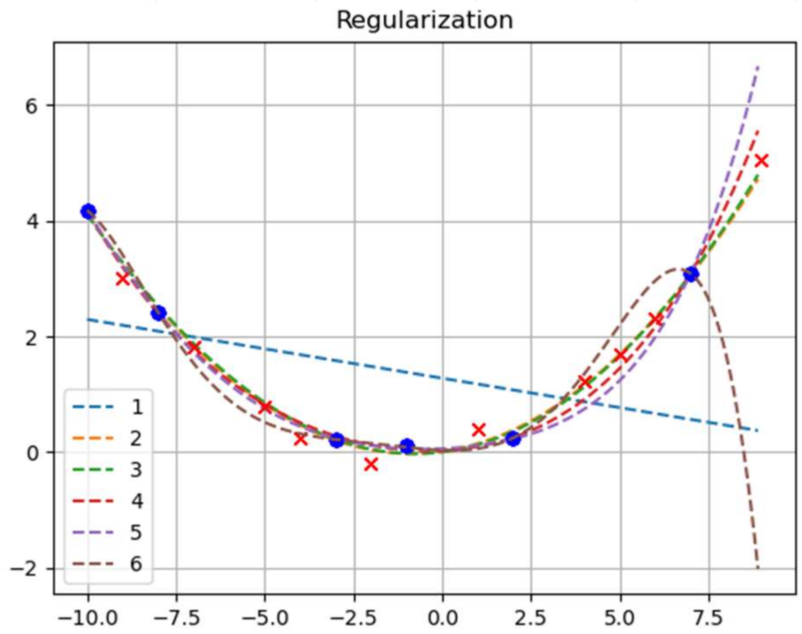
Order	1	2	3	4	5	6
Training MSE	2.3071	8.4408e-03	8.3026e-03	1.7348e-03	3.8606e-25	2.3656e-17
Test MSE	3.0006	0.0296	0.0301	0.0854	1.0548	10.7674

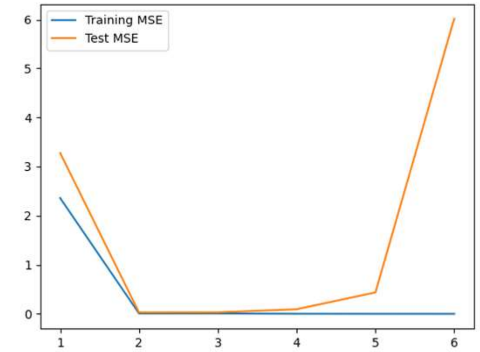
REGULARIZATION

Order	1	2	3	4	5	6
Training MSE	2.3586	8.4565e-03	8.3560e-03	1.8080e-03	7.2650e-04	1.9348e-04
Test MSE	3.2756	0.0302	0.0314	0.0939	0.4369	6.0202

- None of the curves pass through training data exactly ➔ training error increased slightly
- Test MSE for orders 5 & 6 reduced ➔ Overfitting reduced.
- But Test MSE for orders 1 to 4 increased!! ➔ Overly strong regularization.
- Regularization did not help in identifying best polynomial order ➔ cross-validation to be learnt in future!


No Regularization
1
2
3
4
5
6


Regularization
1
2
3
4
5
6


Training MSE
Test MSE