Security Analytics

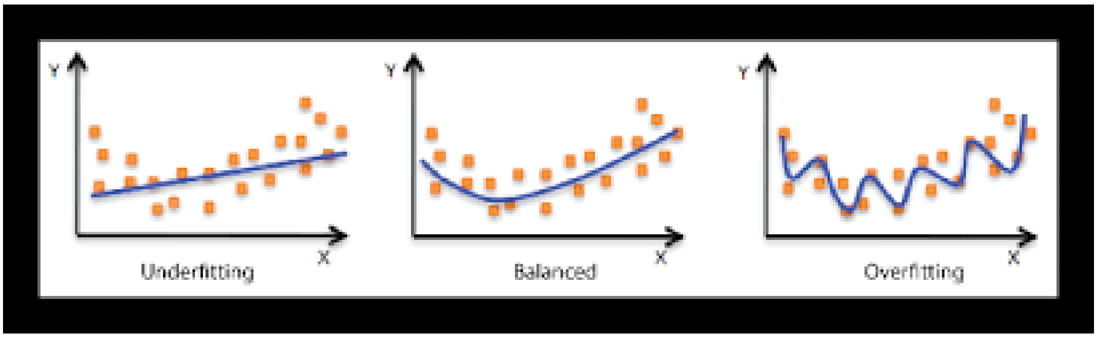
Assignment 1

Student Name: Chandrika Mukherjee
Student ID: 32808289

Problem 1

Background : part 1

- **Generalization:** How well is a trained model to classify or forecast unseen data. a generalized model should work for all subsets of unseen data. Diversity of Input is important factor in order to keep the model generalized, therefore, error rate doesn't vary when testing against unseen data.
- **Overfitting:** When a very complex model is learnt, the training output fits the training data very well, but while testing the model against unseen data, the model performs poorly. Overfitting model learns the variability in the training data which includes noise too, therefore, it has high variance.
- **Underfitting :** When a simple model is learnt that performs poorly on training data and testing data both. Underfitting model has high bias(predictions are very far from target values).



- **Regularization:** When a model learns the outliers in the training data, the model becomes very complex which overfits the data. The coefficients take larger value when the model learns the outliers. Therefore, to penalize the coefficients, regularization is applied on the model. Regularization parameter controls the strength of Regularization.
- **No free lunch theorem:** According to this theorem, there is no single best optimization algorithm. All optimization algorithm performs equally well when their performances are averaged over all possible object functions. Suppose, a model A performs better than model B against a dataset, there will be another dataset for which B will perform better than A.
- **Occam's razor:** According to this principle, We should prefer simple models with fewer coefficients over complex models. Complex models overfits the training data.

- **Independent and identically distributed data points:** When all the data points come from same probability distribution and their occurrences are independent of each other (there are no overall trend in the data points). Example- unbiased coin toss, unbiased dice roll etc.
- **Cross-validation:** To understand how well a model will generalize an independent/unseen data, cross validation is performed.
  - Holdout Method: Splitting a dataset into training and testing set. The model is trained against the training data and later the model is tested against the testing/validation data.
  - K-Fold Cross Validation: We iterate K times on the whole dataset. a data point within the dataset is given the opportunity to be used in one test case and rest of the (K-1) times, it will be used as training data.
- **Degrees of freedom:** The number of independent parameters for a system is described as the degree of freedom. When degree of freedom is more, the model is expected to overfit the training data.

part 2

Given, the observations of two different coin tosses as follows -
$Coin_1 = H, H, H, H, T, T, H, H, H, H, T, T, H, H, H, H, T$
$Coin_2 = H, H, T, T, T, T, H, H, T, T, T, T, H, H, T, T, T$

Coin tosses are Independent and Identically Distributed(i.i.d.) random variables. This implies that the coin toss events does not follow any trend and their occurrences are independent of each other. Also, for coin toss, only possible outcome is H and T. So, even if the observation suggests that for both the cases, if another toss is performed, will get T in both. But as these coin toss events are i.i.d., the probability of getting H and T are still same (0.5). So, there is no certainty of any specific outcome.

part 3

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{pmatrix}_{N,M+1}$$

Converting given X matrix to Z matrix using $z_{i,j} = x_i^j$ rule

$$Z = \begin{pmatrix} 1 & z_{1,1} & z_{1,2} & \cdots & z_{1,M} \\ 1 & z_{2,1} & z_{2,2} & \cdots & z_{2,M} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & z_{N,1} & z_{N,2} & \cdots & z_{N,M} \end{pmatrix}_{N,M+1}$$

Z matrix corresponding to given X matrix

. $$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_N \end{bmatrix}_{N,1}$$ t is the output matrix

. $$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_M \end{bmatrix}_{M+1,1}$$ w is the weight matrix

- total number of samples is N
- number of features is M

$$\begin{aligned} E(w) &= \frac{1}{2}\sum_{i=1}^{N}(w^T Z_i - t_i)^2 \\ &= \frac{1}{2}||Zw - t||_2^2 \\ &= \frac{1}{2}(Zw - t)^T(Zw - t) \\ &= \frac{1}{2}(w^T Z^T - t^T)(Zw - t) \\ &= \frac{1}{2}(w^T Z^T Zw - w^T Z^T t - t^T Zw + t^T t) \\ &= \frac{1}{2}w^T Z^T Zw - w^T Z^T t + \frac{1}{2}t^T t \end{aligned} \tag{1}$$

Taking derivative with respect to w and Using $\frac{\partial W^T AW}{\partial W} = 2\text{AW}$ where A is Symmetric Matrix

$$\begin{aligned} \frac{\partial E}{\partial w} &= \frac{1}{2} * 2Z^T Zw - Z^T t + 0 \\ &= Z^T Zw - Z^T t \end{aligned} \tag{2}$$

Setting derivative to zero gives,

$$w^* = (Z^T Z)^{-1} Z^T t \tag{3}$$

Z and Z^T are equivalent to X and X^T respectively, therefore, we can write as follows,

$$w^* = (X^T X)^{-1} X^T t \tag{4}$$

Resulting Polynomial in terms of w^* -

$$
\begin{aligned}
y(x, w^*) &= \sum_{j=0}^{M} (w^*)^T x^j \\
&= \sum_{j=0}^{M} ((X^T X)^{-1} X^T t)^T x^j \\
&= \sum_{j=0}^{M} t^T X((X^T X)^{-1})^T x^j \\
&= \sum_{j=0}^{M} t^T X((X^T X)^T)^{-1} x^j \qquad\qquad using\ \ (A^T)^{-1} = (A^{-1})^T \\
&= \sum_{j=0}^{M} t^T X(X^T X)^{-1} x^j
\end{aligned} \tag{5}
$$

$$
y(x, w^*) = \sum_{j=0}^{M} t^T X(X^T X)^{-1} x^j
$$

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
#Reading Data from adult.data.csv

adult_data = pd.read_csv('adult.data.csv',skipinitialspace=True)
# skipinitialspace is True to remove extra white space from input data
```

In [3]:
```
# checking the head of the data
adult_data.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	se)
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female

In [4]:
```
adult_data.columns
```

Out[4]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-countr
y',
       'salary'],
      dtype='object')
```

In [5]:
```
#1. Male and Female Count :

adult_data['sex'].value_counts()
```

Out[5]:
```
Male      21790
Female    10771
Name: sex, dtype: int64
```

In [6]:
```
#2. Average Age of Women :
np.mean(adult_data[adult_data['sex']=='Female']['age'])
```

Out[6]: 36.85823043357163

In [7]:
```
#3. Percentage of German Citizen :

number_of_germans = adult_data[adult_data['native-country']=='Germany'].sha
total_citizen = adult_data.shape[0]
print ("Percentage of German : ",number_of_germans*100/total_citizen)
```

```
Percentage of German :  0.42074874850281013
```

In [8]:
```
#4. Mean and Std of age who earns more than 50k

salary_more_than_50k_age = adult_data[adult_data['salary']=='>50K']['age']
print (">50K mean : ",np.mean(salary_more_than_50k_age))
print (">50K std : ",np.std(salary_more_than_50k_age))

# Mean and Std of age who earns less than or equal to 50k

salary_less_than_50k_age = adult_data[adult_data['salary']=='<=50K']['age']
print ("<=50K mean : ",np.mean(salary_less_than_50k_age))
print ("<=50K std : ",np.std(salary_less_than_50k_age))
```

```
>50K mean :  44.24984058155847
>50K std :  10.518356927661575
<=50K mean :  36.78373786407767
<=50K std :  14.019804910115214
```

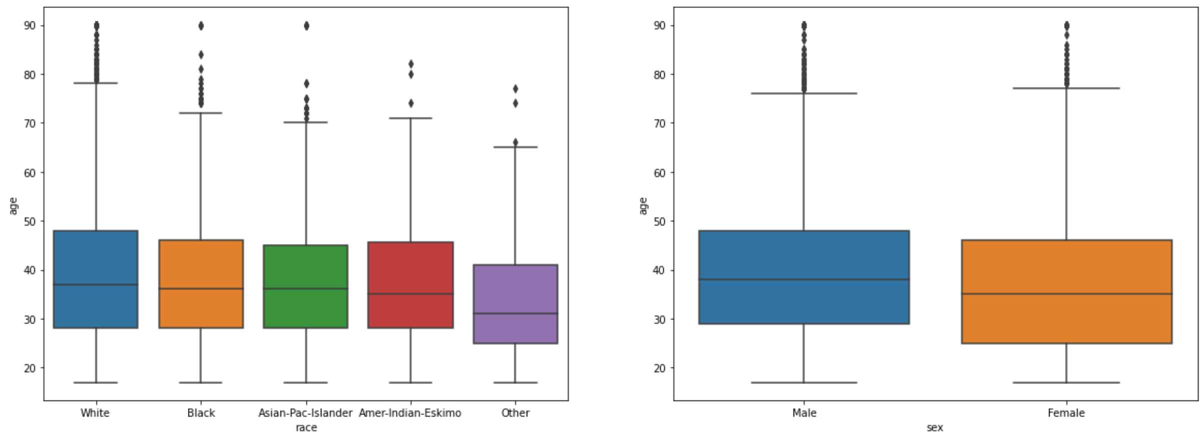
In [9]:
```
#5. Is it true that people who earn more than 50K have at least high school

salary_more_than_50k = adult_data[adult_data['salary']=='>50K']
salary_more_than_50k['education'].value_counts()

# Ans - No, there are people with lower qualification who earn more than 50
```

Out[9]:
```
Bachelors       2221
HS-grad         1675
Some-college    1387
Masters          959
Prof-school      423
Assoc-voc        361
Doctorate        306
Assoc-acdm       265
10th              62
11th              60
7th-8th           40
12th              33
9th               27
5th-6th           16
1st-4th            6
Name: education, dtype: int64
```

In [10]:

```python
#6.  Display age statistics for each race  (BoxPlot & Describe)
# Display age statistics for each gender (BoxPlot & Describe)

plt.figure(figsize=(20,7))
plt.subplot(1,2,1)
sns.boxplot(y='age',x='race',data=adult_data)
plt.subplot(1,2,2)
sns.boxplot(x='sex', y='age', data=adult_data)

print(adult_data.groupby(by='race')['age'].describe())
print(adult_data.groupby(by='sex')['age'].describe())
```

```
                      count       mean        std   min   25%   50%   75%  \
race                                                                       
Amer-Indian-Eskimo    311.0  37.173633  12.447130  17.0  28.0  35.0  45.5   
Asian-Pac-Islander   1039.0  37.746872  12.825133  17.0  28.0  36.0  45.0   
Black                3124.0  37.767926  12.759290  17.0  28.0  36.0  46.0   
Other                 271.0  33.457565  11.538865  17.0  25.0  31.0  41.0   
White               27816.0  38.769881  13.782306  17.0  28.0  37.0  48.0   

                     max  
race                      
Amer-Indian-Eskimo  82.0  
Asian-Pac-Islander  90.0  
Black               90.0  
Other               77.0  
White               90.0  
          count       mean        std   min   25%   50%   75%   max
sex                                                                
Female  10771.0  36.858230  14.013697  17.0  25.0  35.0  46.0  90.0
Male    21790.0  39.433547  13.370630  17.0  29.0  38.0  48.0  90.0
```



In [11]:

```
# 7. maximum number of hours a person works per week

max_hours_per_week = adult_data['hours-per-week'].max()
print("max work hours per week : ",max_hours_per_week)

# dataframe that stores the rows where each person works for max hours per
working_max_hours = adult_data[adult_data['hours-per-week']==max_hours_per_
# number of people working for max hours per week
no_of_adults_working_for_max_hours = working_max_hours.shape[0]
print("number of people working for max hours per week ",no_of_adults_worki

# dataframe that stores those rows where each person earns more than 50K an
no_of_adults_earning_more = working_max_hours[working_max_hours['salary']==
print("percentage of people working for max hours and also earning more tha
      (no_of_adults_earning_more*100)/no_of_adults_working_for_max_hours)
```

```
max work hours per week :  99
number of people working for max hours per week  85
percentage of people working for max hours and also earning more than 50K
is  29.41176470588235
```

In [12]:

```
#8. avg salary calculation for those who earns little and also lot.

#  for little salary
print("for little salary people ")
sal_little = adult_data[adult_data['salary']=='<=50K']
print(sal_little.groupby(by='native-country')['hours-per-week'].mean())


# for lot salary
print("\n\nfor more salary people ")
sal_more = adult_data[adult_data['salary']=='>50K']
print(sal_more.groupby(by='native-country')['hours-per-week'].mean())

# people of japan who earns more than 50K
japan_more = sal_more.groupby(by='native-country')['hours-per-week'].mean()
print("\n\navg time of work in japan of people with more salary",japan_more

#people of japan who earns less than or equal to 50K.
japan_less = sal_little.groupby(by='native-country')['hours-per-week'].mean
print("\n\navg time of work in japan of people with less salary",japan_less
```

```
for little salary people
native-country
?                             40.164760
Cambodia                      41.416667
Canada                        37.914634
China                         37.381818
Columbia                      38.684211
Cuba                          37.985714
Dominican-Republic            42.338235
Ecuador                       38.041667
El-Salvador                   36.030928
England                       40.483333
France                        41.058824
Germany                       39.139785
Greece                        41.809524
Guatemala                     39.360656
Haiti                         36.325000
Holand-Netherlands            40.000000
Honduras                      34.333333
Hong                          39.142857
Hungary                       31.300000
India                         38.233333
Iran                          41.440000
Ireland                       40.947368
Italy                         39.625000
Jamaica                       38.239437
Japan                         41.000000
Laos                          40.375000
Mexico                        40.003279
Nicaragua                     36.093750
Outlying-US(Guam-USVI-etc)    41.857143
Peru                          35.068966
Philippines                   38.065693
Poland                        38.166667
Portugal                      41.939394
```

```
Puerto-Rico                     38.470588
Scotland                        39.444444
South                           40.156250
Taiwan                          33.774194
Thailand                        42.866667
Trinadad&Tobago                 37.058824
United-States                   38.799127
Vietnam                         37.193548
Yugoslavia                      41.600000
Name: hours-per-week, dtype: float64


for more salary people
native-country
?                       45.547945
Cambodia                40.000000
Canada                  45.641026
China                   38.900000
Columbia                50.000000
Cuba                    42.440000
Dominican-Republic      47.000000
Ecuador                 48.750000
El-Salvador             45.000000
England                 44.533333
France                  50.750000
Germany                 44.977273
Greece                  50.625000
Guatemala               36.666667
Haiti                   42.750000
Honduras                60.000000
Hong                    45.000000
Hungary                 50.000000
India                   46.475000
Iran                    47.500000
Ireland                 48.000000
Italy                   45.400000
Jamaica                 41.100000
Japan                   47.958333
Laos                    40.000000
Mexico                  46.575758
Nicaragua               37.500000
Peru                    40.000000
Philippines             43.032787
Poland                  39.000000
Portugal                41.500000
Puerto-Rico             39.416667
Scotland                46.666667
South                   51.437500
Taiwan                  46.800000
Thailand                58.333333
Trinadad&Tobago         40.000000
United-States           45.505369
Vietnam                 39.200000
Yugoslavia              49.500000
Name: hours-per-week, dtype: float64
```

```
avg time of work in japan of people with more salary 47.958333333333336

avg time of work in japan of people with less salary 41.0
```

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import random
random.seed(0)
np.random.seed(0)
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
```

In [2]:
```
# forming dataframe from the given data
data = pd.read_csv('data.txt',header=None,names=['x','y'],delim_whitespace=
```

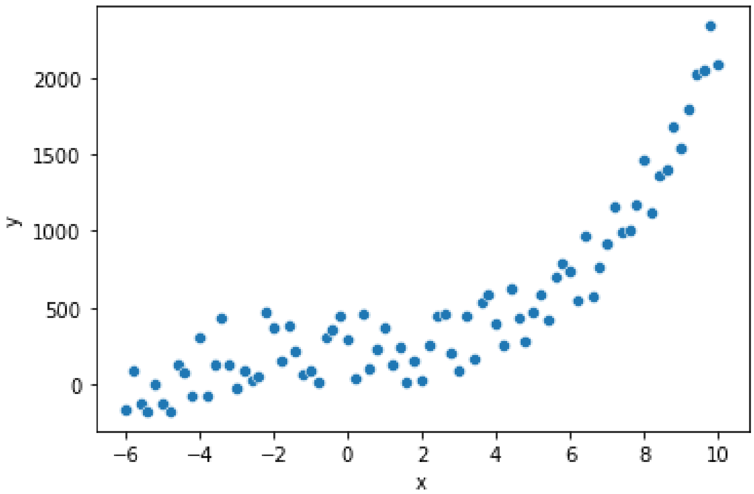
In [3]:
```
#checking the head of the data
data.head()
```

Out[3]:

	x	y
0	-6.0	-164.160590
1	-5.8	90.739607
2	-5.6	-131.842090
3	-5.4	-178.428200
4	-5.2	-4.838565

In [4]:
```
# scatterplot of the given data
sns.scatterplot(x='x',y='y',data=data)
```

Out[4]: <AxesSubplot:xlabel='x', ylabel='y'>



In [5]:
```
X=data['x'].to_numpy() #features
Y=data['y'].to_numpy() #output
kfold = KFold(n_splits=10) #10 splits
trainX=[]
trainY=[]
testX=[]
testY=[]
```

In [6]:
```
#1. Partitioning the data in 10 folds and producing 10 different set of tra
for train_index, test_index in kfold.split(X):
    trainX.append(X[train_index])
    testX.append(X[test_index])
    trainY.append(Y[train_index])
    testY.append(Y[test_index])
```

In [7]:
```
#2. Normalize training input and output sample using mean and standard devi
stdTrainX=[]
stdTrainY=[]
for i in range(10):
    stdTrainX.append((trainX[i]-np.mean(trainX[i],axis=0))/np.std(trainX[i]
    stdTrainY.append((trainY[i]-np.mean(trainY[i],axis=0))/np.std(trainY[i]
```

In [8]:
```
# compute mean squared error
def computeCost(X,y,weights):
    n = len(y);
    predictions = X.dot(weights)
    sq_err = (predictions-y)**2;
    return np.mean(sq_err)
```

In [9]:
```
# compute standard deviation of error
def computeStdCost(X,y,weights):
    n = len(y);
    predictions = X.dot(weights)
    sq_err = (predictions-y)**2;
    return np.std(sq_err)
```

In [10]:
```
# compute training weights
def computeModels(degree):
    model_weights=[]
    for ind in range(10):
        polyFeat = PolynomialFeatures(degree=degree)
        Xtrain = polyFeat.fit_transform(stdTrainX[ind].reshape(-1,1))
        cmodel = LinearRegression(fit_intercept=False)
        cmodel.fit(Xtrain,stdTrainY[ind])
        weight=[]
        for item in (cmodel.coef_):
            weight.append(item)
        model_weights.append(weight)
    return model_weights
```

In [11]:

```
# store mean, std of training and test error over 10 folds for errorbar plo
mean_training_err = []
mean_test_err = []
std_training_err =[]
std_test_err =[]
```

In [12]:

```
# prints mean and standard deviation of training
def computeErrorTerms(degree,nthModel,weight_list):
    polyFeat = PolynomialFeatures(degree=degree)
    train_mean_err_list=[]
    test_mean_err_list=[]
    train_std_err_list=[]
    test_std_err_list=[]

    for i in range(10):
        weights = np.array(weight_list[i]).reshape((degree+1,1))
        Xtrain = polyFeat.fit_transform(stdTrainX[i].reshape(-1,1))
        Xtest = polyFeat.transform(testX[i].reshape(-1,1))

        train_err = computeCost(Xtrain,stdTrainY[i].reshape(len(stdTrainY[i
        test_err = computeCost(Xtest,testY[i].reshape(len(testY[i]),1),weig
        train_std_err = computeStdCost(Xtrain,stdTrainY[i].reshape(len(stdT
        test_std_err = computeCost(Xtest,testY[i].reshape(len(testY[i]),1),

        train_mean_err_list.append(train_err)
        test_mean_err_list.append(test_err)
        train_std_err_list.append(train_std_err)
        test_std_err_list.append(test_std_err)

        print("for ",str(i+1)," fold - ")
        print("mean train error : ",train_err)
        print("std train error : ",train_std_err)
        print("\n")

    mean_training_err.append(np.mean(train_mean_err_list))
    std_training_err.append(np.std(train_std_err_list)/np.sqrt(10))
    mean_test_err.append(np.mean(test_mean_err_list))
    std_test_err.append(np.std(test_std_err_list)/np.sqrt(10))
    min_val = min(train_mean_err_list)
    min_ind = train_mean_err_list.index(min_val)

    if(degree==1):
        global min_id_deg1 # min index for degree 1
        min_id_deg1 = min_ind
        global min_val_deg1 #min value for degree 1
        min_val_deg1 = min_val
    elif(degree==3):
        global min_id_deg3 #min index for degree 3
        min_id_deg3 = min_ind
        global min_val_deg3 #min value for degree 3
        min_val_deg3=min_val
    elif(degree==5):
        global min_id_deg5 #min index for degree 5
        min_id_deg5 = min_ind
        global min_val_deg5 #min value for degree 5
        min_val_deg5 = min_val
    else:
        global min_id_deg50 #min index for degree 50
        min_id_deg50 = min_ind
        global min_val_deg50 #min value for degree 50
        min_val_deg50 = min_val
```

In [13]:

```
# to plot minimum training error output
def plotTrainingOutput(degree,min_ind,min_val,modelNo,weight_list):
    polyFeat = PolynomialFeatures(degree=degree)
    print("min training error ",min_val," at index",min_ind+1,"for Hypothes
    plt.scatter(stdTrainX[min_ind],stdTrainY[min_ind])
    X_Poly = polyFeat.fit_transform(stdTrainX[min_ind].reshape(-1,1))
    pred = X_Poly.dot(np.array(weight_list[min_ind]).reshape(degree+1,1))
    plt.plot(stdTrainX[min_ind],pred,'r')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title("Hypothesis "+str(modelNo+1))
```

In [14]:

```
#3a. weights for degree 1
weight_list1 = computeModels(1)
print(weight_list1)
```

```
[[1.0467283057891832e-16, 0.826237884186147], [-1.2994177644568626e-16,
0.837988904180901], [0.0, 0.8374049255899256], [2.0790684231309803e-16,
0.8350692770852614], [2.5988355289137254e-17, 0.8475765867011976], [1.559
3013173482354e-16, 0.852937261008124], [-7.796506586741176e-17, 0.8585603
20903085], [-7.796506586741176e-17, 0.8503812151283634], [-5.197671057827
45e-17, 0.8113684542714112], [-7.796506586741176e-17, 0.824784326676302
9]]
```

In [15]:

```
#3b. weights for degree 3
weight_list3 = computeModels(3)
print(weight_list3)
```

```
[[-0.5108279551946254, 0.3657825428543696, 0.5108279551946261, 0.25587434
69628608], [-0.48049478981316146, 0.3354573187060557, 0.5417493471413588,
0.3040068151638949], [-0.47337889896686985, 0.32959893539250107, 0.542115
9646376559, 0.3307715752293232], [-0.48667047797476937, 0.298336513990445
5, 0.5375576540039286, 0.35638623440427003], [-0.46744469962216706, 0.240
94044833097156, 0.485181112940571, 0.3848868508010609], [-0.4918193463888
138, 0.20424354116107768, 0.4694705977856366, 0.3785880947591973], [-0.46
60789799640613, 0.16400122860286387, 0.410385739448531, 0.363451178415226
56], [-0.41913368370646215, 0.16006092116611867, 0.3502617273233245, 0.32
480084499020545], [-0.3679985427463086, 0.12828654533730244, 0.3085654800
1585654, 0.3090243059276885], [-0.33531663240337045, 0.2143813049524664,
0.3353166324033703, 0.3391976741494163]]
```

In [16]:
```
#3c. weights for degree 5
weight_list5 = computeModels(5)
print(weight_list5)
```

```
[[-0.4977314091349073, 0.44744529585160664, 0.4671334346960806, 0.1286632
612647645, 0.017003248795586984, 0.03820755594977226], [-0.45488620975355
204, 0.32331945891828606, 0.45425828057623685, 0.3159608736021651, 0.0322
8357330841147, 0.0006549379020433441], [-0.44252737410616944, 0.340401433
3288223, 0.4208920080364828, 0.3112758265782487, 0.05010078599107182, 0.0
09693097192176842], [-0.4654329792564865, 0.3319243424204555, 0.453344531
0037116, 0.29632666100221555, 0.03706687709504779, 0.02135755726102747],
[-0.3996057600594 5025, 0.20427911931100398, 0.2800329007537325, 0.4348126
5747527176, 0.08435537979925178, -0.013770394656372184], [-0.466542770494
85054, 0.20461811757355858, 0.39053701863532414, 0.3837159323406133, 0.03
2827398507162764, -0.00298652740332344], [-0.4424415959489049, 0.15432370
409431248, 0.3226362062020005, 0.3863543736996728, 0.037609619740702, -0.
009863295405918137], [-0.3979321077406826, 0.16177688406005908, 0.2685083
457120445, 0.3235051711824230 6, 0.03304172810245238, -0.00221534056802434
55], [-0.3397408187134994, 0.14704980713352409, 0.2147844233258075, 0.289
81683641751577, 0.03392587255990767, 0.0007334179348860853], [-0.33045937
063006475, 0.10223603567180302, 0.31911155730587937, 0.5138865005942226,
0.006305918793414557, -0.0524657309638065]]
```

In [17]:

```python
#3d. weights for degree 50
weight_list50 = computeModels(50)
print(weight_list50)
```

```
[[-0.4333067930782929, 1.061520694735234, 2.326571028999715, -1.682695182
9638408, -40.170274579566765, -28.216903239820017, 177.15026379227317, 15
5.23772008270606, -249.5067285863199, -245.40536537518872, -2.87434125440
7131, 12.491781122569693, 166.6780426701131, 202.86138308017476, 80.76639
67750362, 56.90072986592289, -82.78019071221355, -150.6228996607472, -12
6.56899332519713, -143.5865755198168, -34.61430790434059, 42.735808644523
78, 87.35087580496045, 162.9204181026649, 104.67227742001296, 65.43298740
309008, -2.0008671284123523, -117.96649425228843, -103.73839094778478, -1
23.34768125328763, -61.3425291839889, 70.33023780280489, 75.611599856015O
3, 138.61623434990102, 85.56998199201425, -75.1664477852877, -73.46772830
618303, -119.85201289604414, -73.15610700320852, 171.02670320127325, 129.
260157638134, -100.98956799965622, -82.5673608379541, 34.11996656363128,
29.362320370357814, -6.892475285892743, -6.168858561830987, 0.77983729118
39735, 0.7209177499675548, -0.03823801577216557, -0.03634469988528366],
[-0.4537544565650236, 0.9855273508348406, 1.3680526487115252, -4.31068593
8081051, -3.3939350377003663, 7.787010999754689, 1.7894671003846945, 0.61
82021221012315, 2.2050163821883957, -3.829143784307058, 0.375771879405670
1, -4.013332879546862, -1.0632179301549554, -1.2871671659569321, -1.28085
21862570712, 1.7415834000726256, -0.7090445744605068, 3.1350559840533516,
0.3929783341863672, 2.4014736811144854, 1.1820776115749263, 0.17183670490
191144, 1.0410001201049952, -1.9467588172080252, -0.035033136425027714, -
2.371379794264104, -1.2541753743462145, -0.7522426525479434, -1.38297547
072752, 1.4163645910694713, 0.1148018468374920 6, 1.8385444516839933, 1.82
51267225690433, -0.005341877342084189, 1.0834827389437596, -1.50928204000
74909, -1.884546552207835, -0.3210711594737422, -1.4309895371704568, 0.96
75502447498114, 3.0887741920333633, -0.06754232270965504, -1.919421699040
178, -0.3408985397305704, 0.576512260164352, 0.18357324515288553, -0.0801
473136688361, -0.03895219173499331, 0.0021788198860897, 0.003088912113933
695, 0.0003870991940333468], [-0.514254721996 5261, 1.508290925657618, -0.
14725841448109625, -13.600100100032499, 10.937237851850437, 44.9828376151
0584, -30.1152278711924 6, -42.1694213069977 9, 11.467468828200992, -13.073
668742769813, 24.044083550578694, 18.218168876512372, 4.246540207770324,
18.802183390810455, -17.592805486795644, 1.4677395582306842, -16.79905724
4431786, -13.295713515892542, -3.0363544412024877, -13.278670046198005, 1
3.218728421100412, -1.232525383129519, 15.636238808404022, 11.17785421977
2504, 2.9032057535953024, 12.046854130181963, -11.71020290693833 5, -0.396
74403388673257, -12.516227196023072, -13.31956544235024, 1.33038733914157
04, -9.538222422658432, 11.98659283372494, 9.486751978475764, 4.294109501
375976, 14.958817216709278, -8.818305725379204, -9.202074884429365, -3.37
4429000131128, -16.021969715781214, 6.78062569180795, 22.7283473028076 8,
-1.928219047601969, -12.734472832627176, -0.7392282828401235, 3.800094078
831493, 0.5748890587936941, -0.5991106269657465, -0.13177896597498062, 0.
03947872289248622, 0.010793214005403229], [-0.532344488186366, 0.85091340
28080646, 1.2529609760257152, -1.657498474139631 7, 19.15780049030333, -1
9.94610795552043, -212.29736033996522, 128.224402270196, 644.862278914241
9, -170.304340315311 9, -596.1623588857695, -100.96707411298274, -291.7646
0703763314, 169.55868896940783, 426.0546135762501, 173.29555301058713, 44
8.75334909532904, -26.206456426641616, -182.24542867894112, -200.07575702
637328, -499.8598485154023, -149.19978060720092, -159.71849119083677, 61.
157764373276756, 365.6130226730131, 211.86573469238098, 384.889539878063
7, 130.01629262838944, -130.69735396516126, -126.79217203733862, -438.301
90491249823, -250.51853451173025, -31.48191771526072, 6.23876850781987, 4
20.3066108703507, 322.88486386260564, 13.644545161663245, 7.8874547741453
```

```
4, -401.5508130215294, -444.74197107587656, 298.7801448364406, 442.647097
7448937, -80.31537399904423, -213.1243642471502, -4.900971102310024, 57.5
8542858202043, 8.203928826410305, -8.422476246138245, -1.876569664940760
2, 0.5224012121119443, 0.1469034778494258], [-0.37173374440956924, 0.0254
10045496670572, -11.245868544623992, 20.154285814624927, 189.842522951805
56, -220.1566434708881, -1087.6296186434163, 910.9694247668266, 2671.1413
56158479, -1513.1065514619363, -2406.078730042528, 445.7566012249277, -80
9.9695630709448, 1156.3010040487452, 1818.5635491527303, -280.82751597933
88, 997.3296891193761, -994.2309956765407, -1001.875734254654, -194.55323
628692602, -1339.87727220989, 702.1816338172783, 99.45886722555876, 527.5
433203456914, 1241.0089138714281, -301.3222851492527, 603.3802241968647,
-581.100192089446, -841.4429400477409, 5.5615499557742964, -931.338580337
359, 482.64190853283804, 519.5111652313717, 88.41499528035149, 995.474194
4240814, -392.36121638717486, -581.5660635792947, 7.498211853797443, -81
8.0326772700644, 299.643203832322, 1231.3400760427608, -248.022915051811
7, -751.036336702816, 99.94230500599042, 261.8390573313093, -22.787929314
232315, -54.64149341690796, 2.830813240815587, 6.395139483661524, -0.1498
317453347795, -0.32474595506948845], [-0.7236038490723833, -1.38885295647
6741, 0.19439514112329434, 27.267823156895773, 77.54605111126838, -165.17
74588044327, -572.8595565950305, 482.6365138938914, 1457.7107658933264, -
628.3137644848977, -1123.2583749357807, 92.09056819266115, -766.190855423
318, 433.7848520135285, 843.0425595721206, 44.55196262889478, 828.4922654
761635, -373.32770490690274, -215.4156188260299, -243.66032180478024, -84
8.6229134460427, 202.03234011505518, -332.7118961199744, 374.865861693423
37, 527.6617076588609, 32.76553059083819, 599.6290450922392, -370.8981397
169536, -145.27518287795482, -222.06567886304288, -592.9903258465037, 31
3.46206668250886, -59.3813101015798, 314.1851453895893, 515.102915116576,
-346.44881830586746, 14.053272197172646, -260.3281804807418, -452.1106983
1387283, 582.6689337059947, 343.50187728140946, -421.0061141415576, -110.
40945464233889, 167.36155393222492, 10.646240558497425, -39.2367494758473
2, 3.12677545461014, 5.120169306131743, -0.9746283188221128, -0.288630039
25118846, 0.08043598655261519], [-0.5136997976770907, 0.8533346031316393,
4.910310975133129, -16.31824915117861, -26.42987745030353, 126.8238146523
644, 20.234006305511826, -355.89871015069485, 92.74017192555225, 305.8896
2802925624, -106.98645169526867, 184.27455059395007, -94.09652239546254,
-200.50774116990803, 54.81734893121672, -241.60030272091979, 108.91568625
614825, 3.862949098895292, 53.575789368863504, 222.95083178006823, -57.81
753496025985, 170.45330890632007, -107.54315178639526, -69.6393235578562
9, -44.89968000224064, -212.97534562445355, 71.75860185738676, -89.633189
1977579, 111.51462204191711, 144.07296199268953, 5.1579255793417165, 165.
38442253887948, -125.33622276346948, -70.74836550201957, -61.978278262502
71, -176.1964388477988, 138.71895791961663, 69.55981528022505, 59.8250860
0245418, 147.29195127929054, -209.26985356988263, -173.21325843629995, 16
8.748029569845, 85.90560660155356, -71.54878633177275, -23.1639509755225
8, 17.553127621543126, 3.3407569713031933, -2.371181707249889, -0.2032228
3955609066, 0.13740977054706605], [-0.46715530214830553, -0.3807173881782
6467, 5.529613950127801, 7.634070268850543, -34.725574254143766, -20.3412
83784505283, 73.07776079515406, 7.663571195244705, -24.76015646667966, 1
9.228479823534215, -51.93582195733352, 4.056837884274073, -11.12914240872
1592, -15.177342912817572, 32.24692023896771, -16.363532161125583, 39.695
624649073736, -3.997396401544296, 11.76098507774779, 11.008860888071471,
-23.224248641770554, 15.167025860875135, -35.798198295602496, 6.253849737
316474, -15.257152322941847, -6.767004624496758, 19.847989287062823, -12.
087555928104619, 33.78598775386108, -5.817123675817596, 7.79558649666781
6, 4.876221490540361, -29.52306751970639, 9.619282426677165, -23.07118969
298545, 3.878940923727424, 25.05235731726129, -8.859891798392113, 22.7234
4011341892, -9.81494860280561, -38.201576306365304, 19.508128216633104, 2
```

```
0.588299237282754, -12.448243975611902, -5.242899643450492, 4.01450271773
8022, 0.5058288208419981, -0.6663362905733186, 0.03091150446659441, 0.045
492412474512633, -0.007247010402174681], [-0.3143308634725738, 0.20368441
052299116, 0.8126702324517163, 0.23102508870750219, -0.7700416854000234,
-0.1553466164159513, -0.6282601449429773, 0.07661118106346668, -0.0951594
2777093767, 0.29065185810889416, 0.37615696241692914, 0.3241219560891948,
0.6242371209534735, 0.1600291275501901, 0.5680454181016762, -0.0894324109
218191, 0.280055093074414, -0.31688676582092207, -0.12042334318263952, -
0.38959766403065893, -0.44871144351886577, -0.25956186385978075, -0.54524
02935676329, 0.029764512350704743, -0.3414263198500933, 0.321756588198648
7, 0.06359067667664806, 0.40048735956254594, 0.4048691023748791, 0.140792
2389670186, 0.4008732864755704, -0.30004156840205554, 0.02606111642071789
3, -0.4603571521192952, -0.3339063862697681, -0.00350556387665317, -0.217
15155403900094, 0.5573535932407059, 0.18671561463231146, 0.10620119093608
124, 0.13339524557366614, -0.7226676736432286, -0.10694072803205196, 0.56
08393744895819, -0.009338222275934414, -0.20147503588043852, 0.0243558265
53386528, 0.03609234013145062, -0.00713158951552284, -0.00261459666271568
47, 0.0006721552846276624], [-0.509925589667073, -0.17927641988382684, 1
2.052185501948587, -14.7550776336001, -112.72536139768972, 188.0210857516
43, 371.5172771033147, -652.431527469316, -420.1720362365319, 760.6384746
322193, -36.020409051306025, 124.35248029596684, 154.1626799500761, -586.
8022957666421, 247.68381995254867, -216.6021545070603, -54.2158781134567
7, 305.99155834224484, -239.75037093556745, 397.0254607352845, -116.25942
95823927, -29.141010030088598, 122.46863415827809, -379.2697673918309, 18
6.4713536401579, -202.35230197699238, 14.627013223283626, 249.21102773662
47, -155.2208377690756, 311.96047206231, -90.35227728500209, -141.2959760
1960128, 103.17225647896464, -339.7524517998077, 103.39746259145682, 165.
71083383118932, -85.27767037942908, 299.00513940198726, -84.5621224142836
9, -416.2659807830965, 142.405453943545, 246.26613079835025, -87.64218613
805349, -83.9744992201517, 30.118311871565766, 17.185299923389664, -6.126
15833003872, -1.9743234384435056, 0.6940963541769847, 0.0984392645769389
7, -0.03396366659777739]]
```

In [18]:

```
#4a. print mean and std of training errors for degree 1
computeErrorTerms(1,0,weight_list1)
```

```
for  1  fold -
mean train error :  0.3173309587355997
std train error :  0.46279615963827253


for  2  fold -
mean train error :  0.2977745964696934
std train error :  0.45262588797813363


for  3  fold -
mean train error :  0.29875299059773075
std train error :  0.45988078253877074


for  4  fold -
mean train error :  0.3026593024682998
std train error :  0.4637003859665985


for  5  fold -
mean train error :  0.2816139296759475
std train error :  0.45323572831283193


for  6  fold -
mean train error :  0.27249802878395984
std train error :  0.4233104035894403


for  7  fold -
mean train error :  0.2628741753707918
std train error :  0.3994387160717923


for  8  fold -
mean train error :  0.2768517889568073
std train error :  0.40371714795795277


for  9  fold -
mean train error :  0.3416812314132205
std train error :  0.5442314520114779


for  10  fold -
mean train error :  0.31973081446911766
std train error :  0.4496777113336276
```

In [19]:

```
#4a. print mean and std of training errors for degree 3
computeErrorTerms(3,1,weight_list3)
```

```
for  1  fold -
mean train error :  0.06837596678758616
std train error :  0.060689990622467375


for  2  fold -
mean train error :  0.05982967845204849
std train error :  0.05288446436513692


for  3  fold -
mean train error :  0.05947991117497331
std train error :  0.055340599589408256


for  4  fold -
mean train error :  0.058248811165242984
std train error :  0.05616310043660449


for  5  fold -
mean train error :  0.06152794299750575
std train error :  0.054829092769489926


for  6  fold -
mean train error :  0.05826000207333155
std train error :  0.055728898734900285


for  7  fold -
mean train error :  0.059147714857882214
std train error :  0.052689648976030054


for  8  fold -
mean train error :  0.05835745582128888
std train error :  0.054609876445316625


for  9  fold -
mean train error :  0.06952391228854007
std train error :  0.062072633982918554


for  10  fold -
mean train error :  0.1589727921423581
std train error :  0.14739702561802223
```

In [20]:
```
#4a. print mean and std of training errors for degree 5
computeErrorTerms(5,2,weight_list5)
```

```
for  1  fold -
mean train error :  0.06772569846903997
std train error :  0.05943384537027068


for  2  fold -
mean train error :  0.059284720722800166
std train error :  0.052942804837405065


for  3  fold -
mean train error :  0.058559424037951543
std train error :  0.056181572440077536


for  4  fold -
mean train error :  0.057754665107394826
std train error :  0.05673141673812384


for  5  fold -
mean train error :  0.05932228692031022
std train error :  0.05759651770925058


for  6  fold -
mean train error :  0.05792796726686325
std train error :  0.056419446339446455


for  7  fold -
mean train error :  0.058651560839176546
std train error :  0.05386749013937117


for  8  fold -
mean train error :  0.05791265845413088
std train error :  0.055873924098201855


for  9  fold -
mean train error :  0.06885117332138874
std train error :  0.06318097242194338


for  10  fold -
mean train error :  0.15798282261197497
std train error :  0.14907199778121244
```

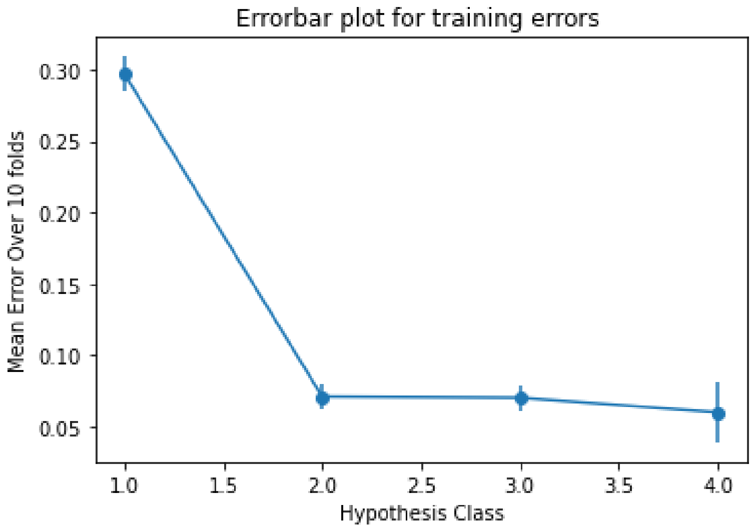
In [21]:
```
#4a. print mean and std of training errors for degree 50
computeErrorTerms(50,3,weight_list50)
```

```
for  1  fold -
mean train error :  0.04417908056972276
std train error :  0.05597253629859364


for  2  fold -
mean train error :  0.05166114971267614
std train error :  0.05766816591057447


for  3  fold -
mean train error :  0.047819507926513675
std train error :  0.05830312081669739


for  4  fold -
mean train error :  0.038369099921729606
std train error :  0.048541571059325654


for  5  fold -
mean train error :  0.04456805242557462
std train error :  0.051333483940945994


for  6  fold -
mean train error :  0.0349377834355395
std train error :  0.046461916488650466


for  7  fold -
mean train error :  0.046712722553534154
std train error :  0.05700056326380344


for  8  fold -
mean train error :  0.06553111001779907
std train error :  0.08175740346667046


for  9  fold -
mean train error :  0.10937087708476805
std train error :  0.26060802203513617


for  10  fold -
mean train error :  0.11750756662242982
std train error :  0.16083894943506818
```

In [22]:

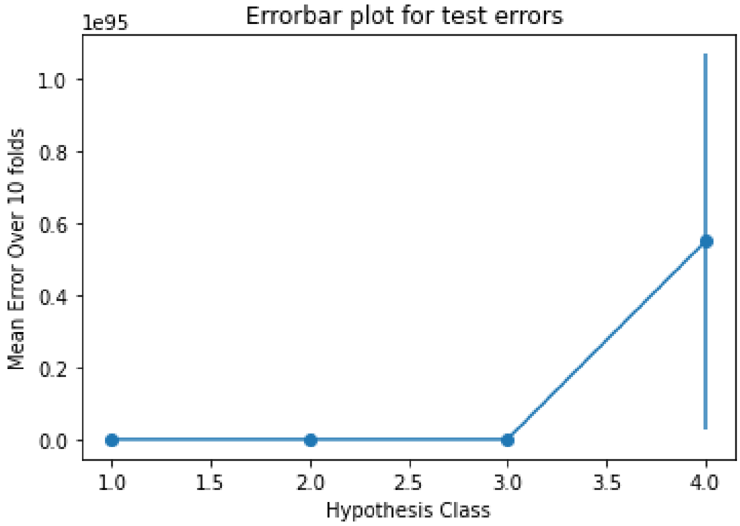
```
# 4a. Error bar plot for training
ind=[1,2,3,4]
plt.errorbar(x=ind,y=mean_training_err,fmt='o',yerr=std_training_err, ls='-
plt.xlabel('Hypothesis Class')
plt.ylabel('Mean Error Over 10 folds')
plt.title("Errorbar plot for training errors ")
print(mean_training_err)
print(std_training_err)
```

```
[0.2971767816941169, 0.07117241877607575, 0.0703972977751031, 0.060065695
02702874]
[0.01214128325992253, 0.008707995186968787, 0.008784525231512694, 0.02086
94140392961]
```



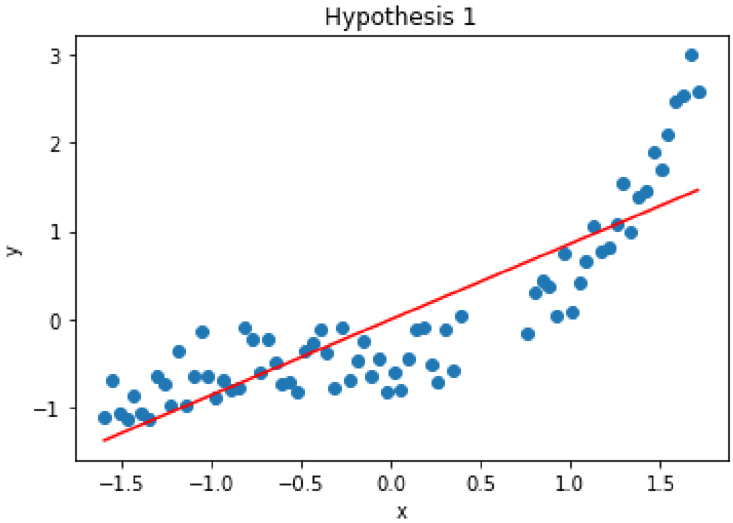
In [23]:
```python
# 4a. Error bar plot for test
ind=[1,2,3,4]
plt.errorbar(x=ind,y=mean_test_err,yerr=std_test_err ,fmt='o', ls='-')
plt.xlabel('Hypothesis Class')
plt.ylabel('Mean Error Over 10 folds')
plt.title("Errorbar plot for test errors ")
print(mean_test_err)
print(std_test_err)
```

```
[597293.7298207106, 444567.58355719165, 2906386.85508043, 5.4875152131823
59e+94]
[333223.8351833852, 234630.46318061807, 2580887.045489795, 5.205914030504
48e+94]
```



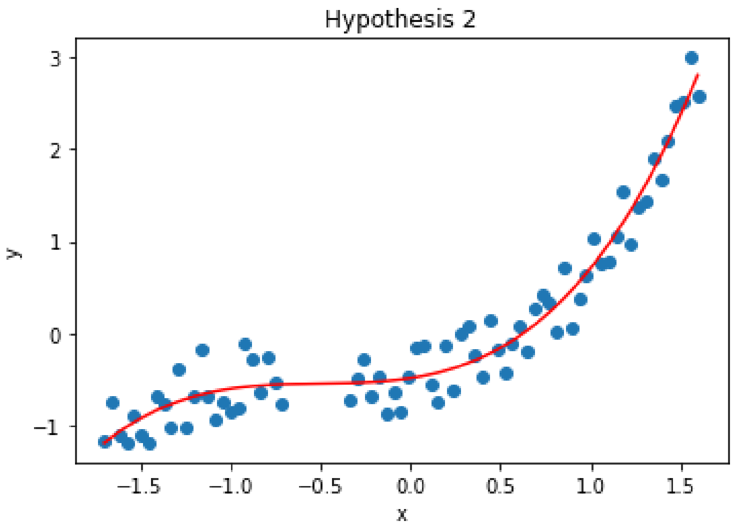
In [24]:
```python
# 4b. for degree 1 plot
plotTrainingOutput(1,min_id_deg1,min_val_deg1,0,weight_list1)
```

```
min training error  0.2628741753707918  at index 7 for Hypothesis  0
```



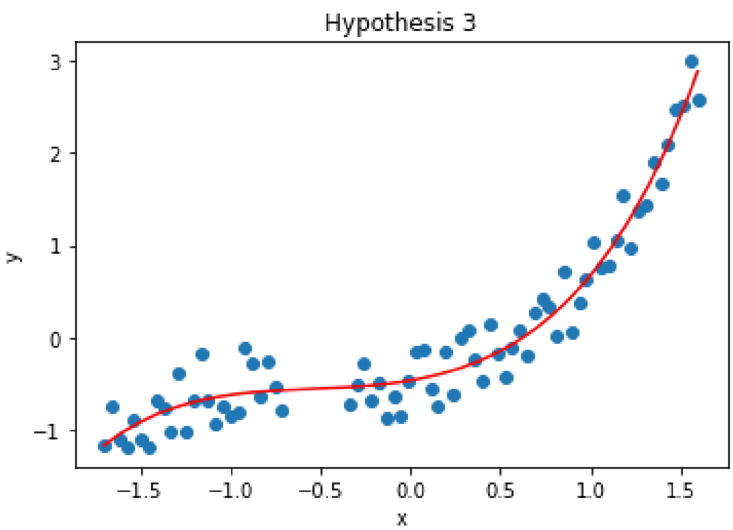
In [25]:
```
#4b. for degree 3 plot
plotTrainingOutput(3,min_id_deg3,min_val_deg3,1,weight_list3)
```

```
min training error  0.058248811165242984  at index 4 for Hypothesis  1
```



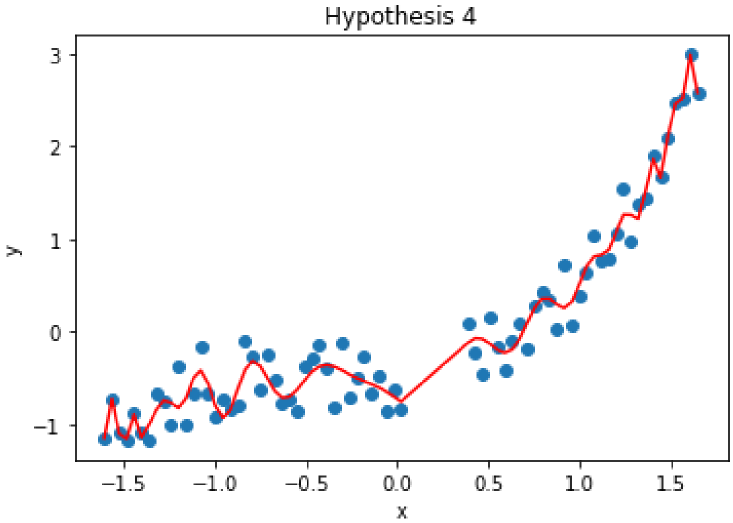
In [26]:
```
#4b. for degree 5 plot
plotTrainingOutput(5,min_id_deg5,min_val_deg5,2,weight_list5)
```

```
min training error  0.057754665107394826  at index 4 for Hypothesis  2
```



In [27]:
```
#4b for degree 50 plot
plotTrainingOutput(50,min_id_deg50,min_val_deg50,3,weight_list50)
```

```
min training error  0.0349377834355395  at index 6 for Hypothesis  3
```



5) Hypothesis2 with degree 3 is better than other hypotheses.

- hypothesis 1 with linear model underfits the training data (simple model)
- hypothesis 3 with degree 5 starts overfitting the training data (complex model)
- hypothesis 4 with degree 50 exactly overfits the training data (extremely complex model)
- Test Error Drops in Hypothesis2 with degree 3 from Hypothesis1 with degree 1, also, Test error increases largely in Hypotheses3 with degree 5 and Hypotheses4 with degree 50.
- Therefore, I will choose Hypothesis2

In [ ]: