EE 5841 Exam submission 1

In [ ]:

```
submitted by Sarala Ravindra
```

Question 1

Curse of dimentionality refers to the need of exponential data points for linear increase in the dimension of the data space.As the number of features incrtease without increase in data samples the data ponits tend to reside at the surface of the dimensional sphere.In KNN algorithm where the nearest neighbors are averaged (regression) or voted (classification) the dimentionality increase with out sufficient increase in the data points leads to bad performance in predicting or classifying a input sample.This is because there is very little difference in the distance of a nearest neighbors and the farthest neighbors so their distinction is not properly.

The simplest approach to choose the best set of features among large number of features is described below

Let N be the total number of features. Let the cross validation test accuracy be the score .

Agorithm 1

1. Initialize m = 0 , f_score=0 ,f_score_new=0 ,and f_label , f_label_new be two zero vectors.
2. Select all possible features for feature subset size m+1 and produce the test score for each of the feature subset.
3. Select the best scoring (the highest) subset for m.
4. Let f_label_new hold the feature labels (or feature numbers) and f_score_new hold the corresponding score of the subet selected in step 3
5. Compare f_score_new and f_score and store the highest of the two scores in f_score and its corresponding feature labels in f_label.
6. Increment m by 1 .
7. Repeat steps 2 to 6 if m is not equal to N else continue to 8.
8. f_label holds the best feature subset with its corresponding score in f_score.

The approach described above is too time consuming and is not a very good method thinking in terms of processing time and other computation requirement.A better approach is described in the following algorithm

Algorithm to determine the best subset of features to use with kNN

1. Center the train features by subracting each feature component by it's respective feature mean.
2. Scale the train features by dividing each feature component by it's respective feature standard deviation.
3. Repeat data centering and scaling as in step 1 and 2 with the train output.
4. Calculate the Pearson's correlation coefficient between each feature and the ouput

4. Calculate the Pearson's correlation coefficient between each feature and the ouput.
5. Select a threshold between 0 and 1(choose values in higher range as we are looking for features high correlated with the output).
6. Select those features whose correlation coefficient magnitude is above the specified threshold.
7. Determine the test accuracy by cross-validation method and store the test accuracy corresponding to the threshold.
8. Repeat steps 5 to 7 for various values of thresholds.
9. Pick the threshold giving the best cross-validation test accuracy.
10. Select features with correlation coefficient above the threshold chosen in step 6.
11. Stop

The features selected form a best subset of features for kNN this is because the datapoints now have less features but those are the feautres which actually have more effect in classification.Now the datapoints have less features means the variability of the datapoints increase decreasing the bias which would have existed in high dimensional datapoints.That is when calculating the distances of the test sample to determine the nearest samples ,the nearest and farthest samples in high dimensional space have very less difference making higher chances of test samples to assume wrong data points to be nearest neighbours,leading to wrong classification.Thus best subset feature selection reduces the number of features in the data points making a good dimensional space to model the kNN prediction.

Question 2

Let the probability of disease occurence be **P(D)** and the probabilities of having disease given the first test result and two test results (written consecutively first test result first) be **P(D/T)** and **P(D/TT)** ,where **T** represents the test result either positive **P** or negative **N**.

The certainity of a patient having disease increases with the second test result also being positive as this probability has two features instead of one.To make it more clear P(D/T) only considers one test that is the input test case is just one while P(D/TT) considers two test cases.In simple terms the first probability model just has one feature and the second probability model has two features.We know that the two tests are independent so the two features are non-reduntant and they do not have relationship with one other.In fact as the number of tests are increased in the probability model the certainity of the information increases due to the tests being independent.More test results provides more information than fewer test results.So the certainity of a patient having disease is more with the second test also being positive (that is two test results known to be positive) when compared to first test being positive (where only one test result is considered).

To illustrate this let us now calculte and compare these two probabilitie.

To know the certainity of a disease present in a patient diagnosed to be positive on test 1 and positives on both test 1 and test 2 , our goal is to determine **P(D/P)** and **P(D/PP)** for which it is important to know the disease occurence probability and also the probabilities of right tests.The right test describes two situations where the result was positive when the patient diagnosed had the disease and the result was negative when the patient diagnosed did not have the disease.Let probability of positive result when the diseased patient is diagnosed be **P(P/D)** and the other right case described be **P($\bar{P}/\bar{D}$)** .

From Bayes' theorem of probability the probability of disease present in a patient diagnized as positive in a single test is positive in a single test is

$$P(D/P) = \frac{P(P/D) * P(D)}{P(P)}$$

where P(P) is given the law of total probability as follows

$$P(P) = P(P/D) * P(D) + P(P/\bar{D}) * P(\bar{D})$$

From the above two equations we get

$$P(D/P) = \frac{P(P/D) * P(D)}{P(P/D) * P(D) + P(P/\bar{D}) * P(\bar{D})}$$

Dividing the numerator and denominator by numerator we get

$$P(D/P) = \frac{1}{1 + \frac{P(P/\bar{D}) * P(\bar{D})}{P(P/D) * P(D)}}$$

Let **P(D) = 0.01** that is 1 in every 100 of the population has this disease.Let **P(P/D) = 0.8** and **P($P/\bar{D}$) = 0.096** that is there is 80 percent chance of a diseased patient being given positive result and 9.6 percent chance of a not diseased patient being given positive result which makes sense for a decent real-life situation.

Plugging these values in the above equation we get,

$$P(D/P) = \frac{1}{1 + \frac{0.096 * (1 - 0.01)}{0.8 * 0.01}} = 0.0776$$

This means that the chances of a patient having disease when diagnosed positive in one single test is this case is **7.7 %**

Now let us look at the certainity of a patient having disease based on the two independent test (test1 and test2) results on the same patient .The possible outcomes of the two tests and their corresponding probabilities are shown below.

Result test 1	Result test 2	Representation	Probability
P	P	PP	$P(P)P(P)$
P	$\bar{P}$	$P\bar{P}$	$P(P)P(\bar{P})$
$\bar{P}$	P	$\bar{P}P$	$P(\bar{P})P(P)$
$\bar{P}$	$\bar{P}$	$\bar{P}\bar{P}$	$P(\bar{P})P(\bar{P})$

Again using the Bayes' theorem and the law of total probability we have

$$P(D/PP) = \frac{P(PP/D) * P(D)}{P(PP)}$$

$$P(PP) = P(PP/D) * P(D) + P(PP/\bar{D}) * P(\bar{D})$$

Similar to one test case the P(D/PP) is given by

$$P(D/PP) = \frac{P(PP/D) * P(D)}{P(PP/D) * P(D) + P(PP/\bar{D}) * P(\bar{D})}$$

Due to independency of tests the $P(PP/D) = P(P/D) * P(P/D)$ and $P(PP/\bar{D}) = P(P/\bar{D}) * P(P/\bar{D})$ the above equation now becomes

$$P(D/PP) = \frac{P(P/D) * P(P/D) * P(D)}{P(P/D) * P(P/D) * P(D) + P(P/\bar{D}) * P(P/\bar{D}) * P(\bar{D})} = \frac{1}{1 + \frac{P(P/\bar{D})*P(}{P(P/D)*P(}}$$

Again using the same probabilities used in the one test case P(D) = 0.01 $P(P/D) = 0.8$ and $P(P/\bar{D}) = 0.096$, we have

$$P(D/PP) = \frac{1}{1 + \frac{0.096*0.096*(1-0.01)}{0.8*0.8*0.01}} = 0.4122$$

This means that the chances of a patient having disease when diagnosed positive in both of the two independent single tests in our case is **41.22 %** which is higher than that obtained for the single test case.From this we can easily see that the ceratinity of disease present in a patient increases with the second test result also being positive compared to just the first test result being positive.

Now consider third test the possible outcomes of the third test after the two tests are either positive(P) or negative(N).

Result test 3	Representation
P	PPP
N	PPN

probability of disease after positive results in first two tests and negative in the third result is P(D/PPN) given by

$$P(D/PPN) = \frac{P(PPN/D) * P(D)}{P(PPN)} = \frac{P(PPN/D) * P(D)}{P(PPN/D) * P(D) + P(PPN/\bar{D}) * P(\bar{D})}$$

Using the independency of test results

$$P(D/PPN) = \frac{P(P/D) * P(P/D) * P(N/D) * P(D)}{P(P/D) * P(P/D) * P(N/D) * P(D) + P(P/\bar{D}) * P(P/\bar{D}) * P(N/\bar{D})}$$

plugging our values used in our example we have

$$P(D/PPN) = \frac{0.8 * 0.8 * (1 - 0.8) * 0.01}{0.8 * 0.8 * (1 - 0.8) * 0.01 + 0.096 * 0.096 * (1 - 0.096) * (1 - 0.01)} = 0$$

The probability of not having disease when the first two tests were positive and the third test was negative is given by

$$P(\bar{D}/PPN) = 1 - P(D/PPN)$$

(Totoal Sum of probabilities must equal 1)

$$P(\bar{D}/PPN) = 1 - 0.1343 = 0.8656$$

After the third test result is negative (with first test two tests being positive) the probability of that patient having disease is 13.4 % and the patient not having the disease is 86.5 %

Note though the certainity of having disease was around 41 percent with positives on the first two result a negative result following that gradually decreased the chances of the patient having the disease to about 13 perent.

Question 3

Generalization with respect to learning problems defines the adaptibility of the model trained on a certain dataset called the train model to the unkown data to produce proper classification or right prediction.With the intension to achieve higher performance the models are some times overtrained though it looks good with the train data it mostly ends up being a low performer when newer data is posed.Thus generality is an important requirement for any model to be able to be used in real-world applications as the real-world poses data unkown to the model many times.A model is considered to be a generalised model when it is able to perform well even when data from unkwon dataset is encountered.This nature of a model can be achieved if care is taken not to memorize the data from the train set but to have learning on the datapoints behaviour during the learning phase.

Suppose such care is not during the learning phase ofthe model usually the model end up being called **overfitted**.That is the model instead of learning from the data it adjusts itself to satisfy the train data requiremts.This kind of a model has least bias with the train data but surely tends to have high variability.Such model is shown below and the overfitting can be observed.

Let us consider a train data set generated using a sine function with some variance randomly added. We perform polynomial basis regression on the train data and learn weights w of the model.Any new data point is predicted or classified based on the w information.in a polynomial basis model it is the polynomial order which determines the learing of the model.This parameter is varied and the built model is closely observed to know how good the learning has taken place.

Polynomial basis regression example

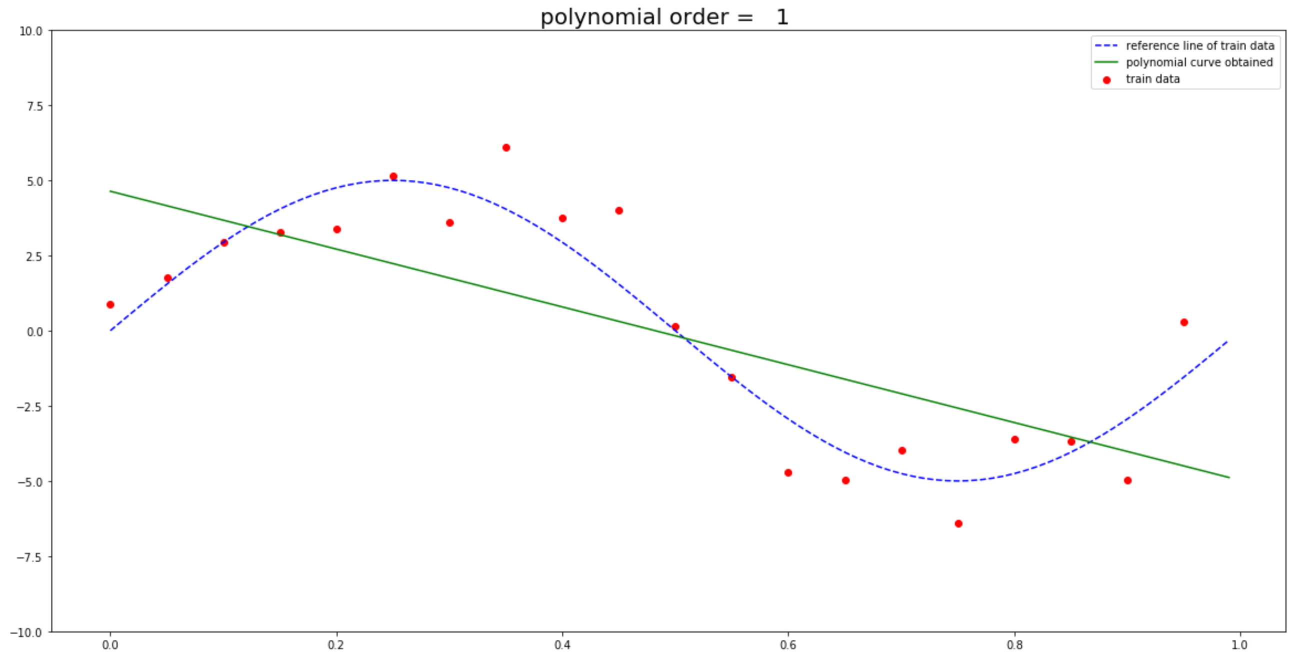
The below code loops for different values of polynomial order and graphs the output based on the model built

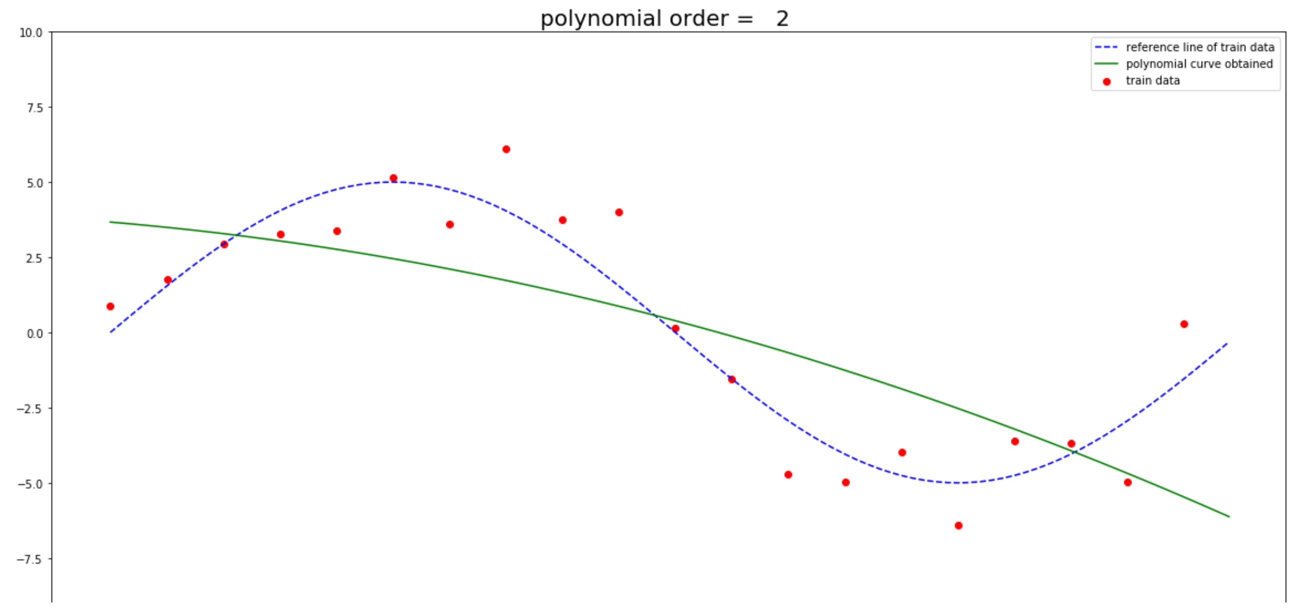
In [25]:

```
## generate train data
import numpy as np
x = np.arange(0,1,0.05)
y_noise = 5*np.sin(2*np.pi*x+0.001)+1.2* np.random.randn(20)
```

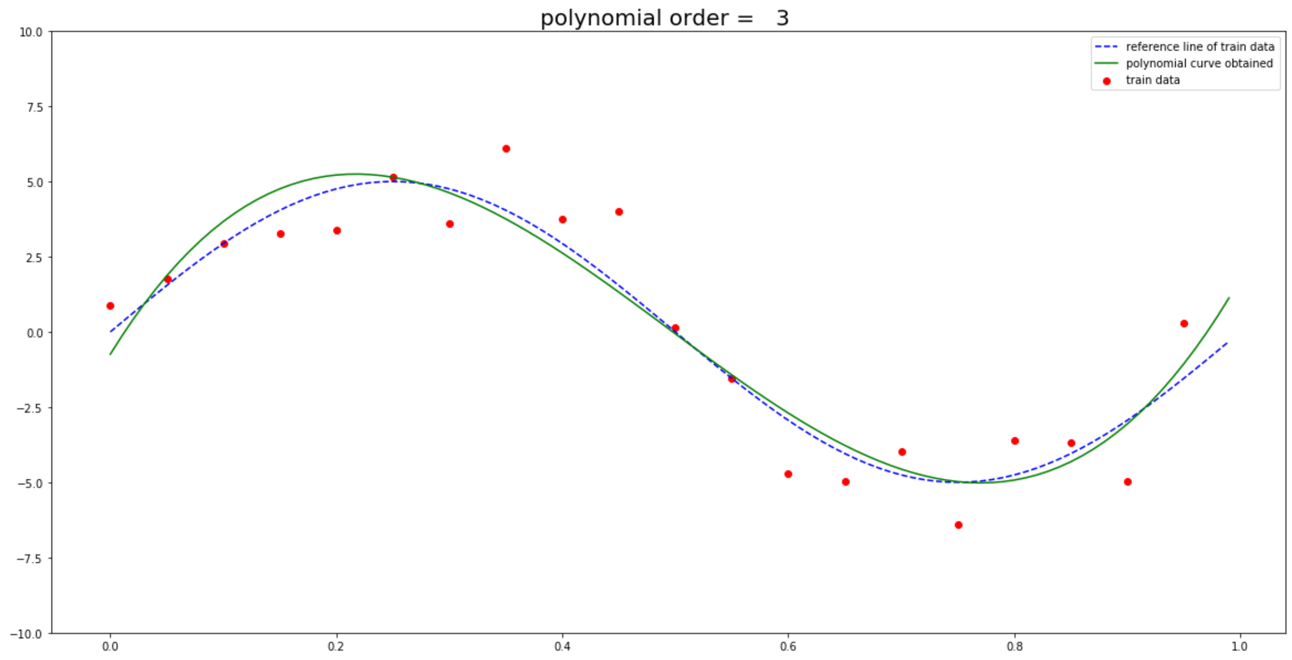
In [21]:

```
#### polynomial basis curve fitting

poly_order = [1,2,3,16,20,25]
for p in range(5):
    phi = np.zeros((len(x),poly_order[p]+1))
    for i in range(poly_order[p]+1):
        phi[:,i] = x**i
    phi_t = np.transpose(phi)
    I = np.identity(np.shape(phi)[1])
    lam = 0                         # regularization term
    w = np.matmul(np.matmul(np.linalg.pinv(np.matmul(phi_t,phi)+lam*I,rcond=
1e-500),phi_t),y_noise)

    import matplotlib.pyplot as plt
    import numpy as np

    x_n = np.arange(0,1,0.01)
    y = 5*np.sin(2*np.pi*x_n)
    phi_n = np.zeros((len(x_n),poly_order[p]+1))
    for i in range(poly_order[p]+1):
        phi_n[:,i] = np.power(x_n,i)

    out_n = np.matmul(np.transpose(w),np.transpose(phi_n))
    plt.figure(figsize=(20,10))
    plt.plot(x_n, y,'--b')
    plt.scatter(x, y_noise,color ='red')
    plt.plot(x_n,out_n,'g')
    plt.ylim(-10,10)
    plt.title('polynomial order =  % i '% poly_order[p],fontsize = 20)
    plt.legend(['reference line of train data', 'polynomial curve obtained'
,'train data'])
    plt.show()
```

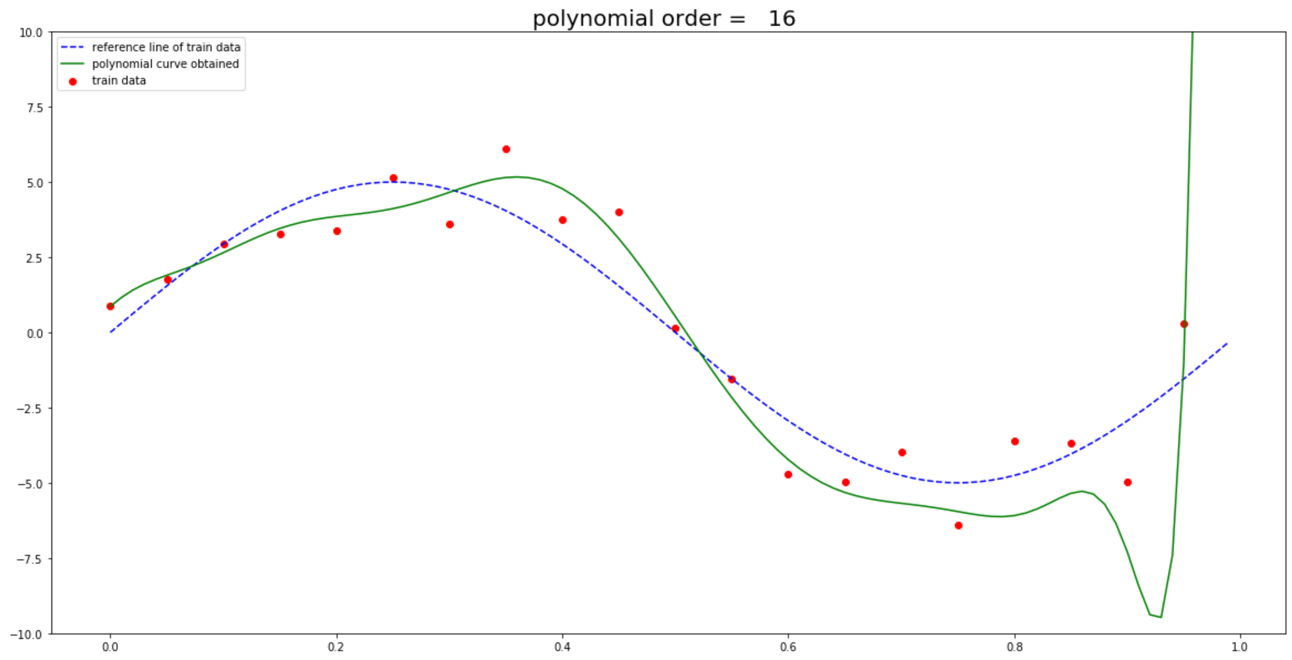




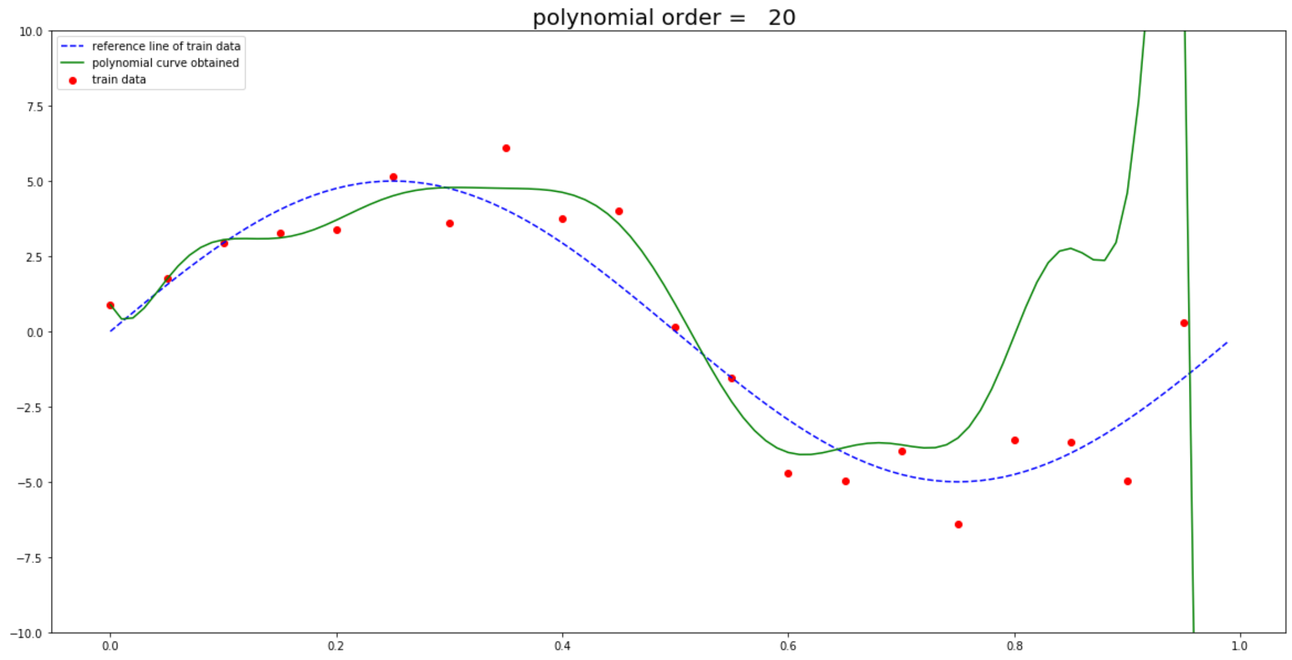
−10.0
0.0
0.2
0.4
0.6
0.8
1.0



polynomial order = 3
reference line of train data
polynomial curve obtained
train data
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0



polynomial order = 16
reference line of train data
polynomial curve obtained
train data
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0



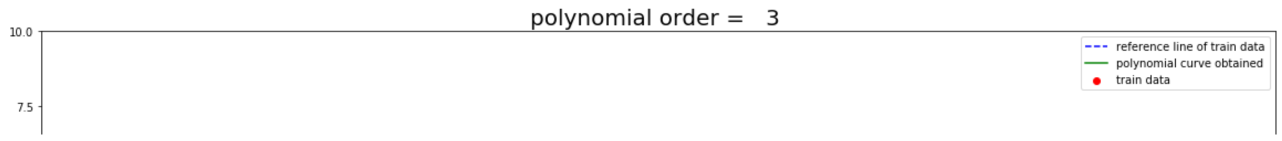
polynomial order = 20
reference line of train data
polynomial curve obtained
train data
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0

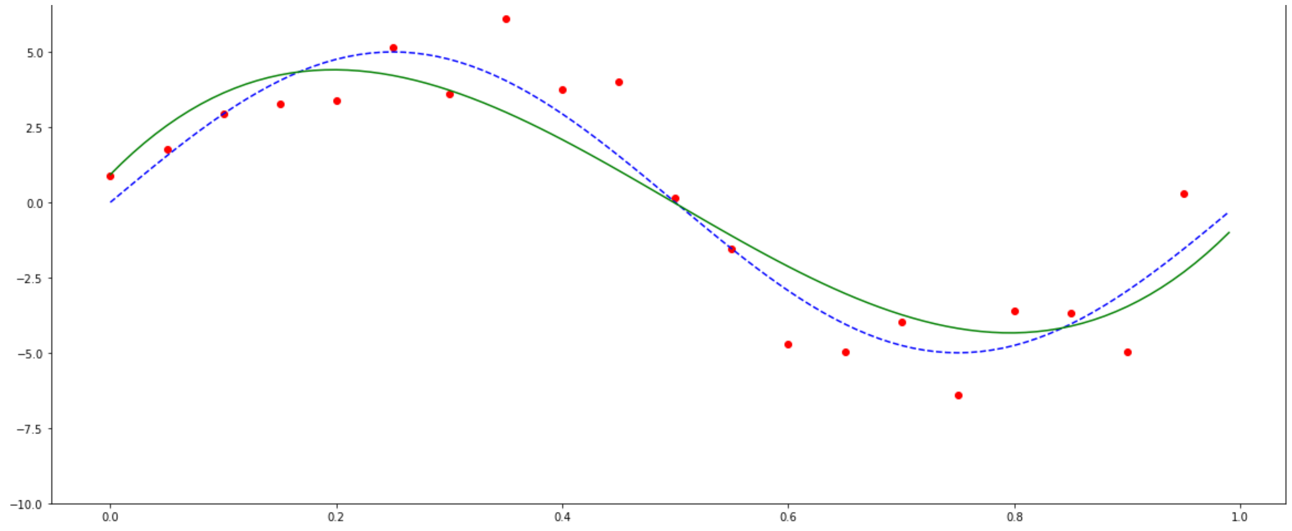
For polynomial order = 1, the model seems to have not learnt properly as the number of weights it produces is only one along with the bis term which is not sufficient to characterize the model.With order 2 the polynomial curve is not sufficient again but it is better than the curve for order 1.But order 3 seems to have got the nature of the model captured .But further increase in the polynomila order almost makes the model vary but only fit thr train data points.This fitting increases with increase in polynomial order.Look at the curve generated for polynomial order 25 , it almost fits the data exactly.During training this can be mistaken to be a good model as it is for sure performing well for the given data points.Suppose any new data point is input there is very less chance of the model doing well as the polynomial curve varies too much over the dataset but fits the train specific datapoints.They are tuned to do this with increse in polynomila order.

This overfitting problem is overcome by using a regularization parameter in building the model.This parameter limits the amount of training data influence on the model.Suppose a high regularisation is achieved the model may not understand the train data behaviour which is again a problem resulting in model underfit.Thus the regularisation parameter must be chosen appropriately. The code below does the polynomial basis curve fitting with regularisation for higher order polynomials.

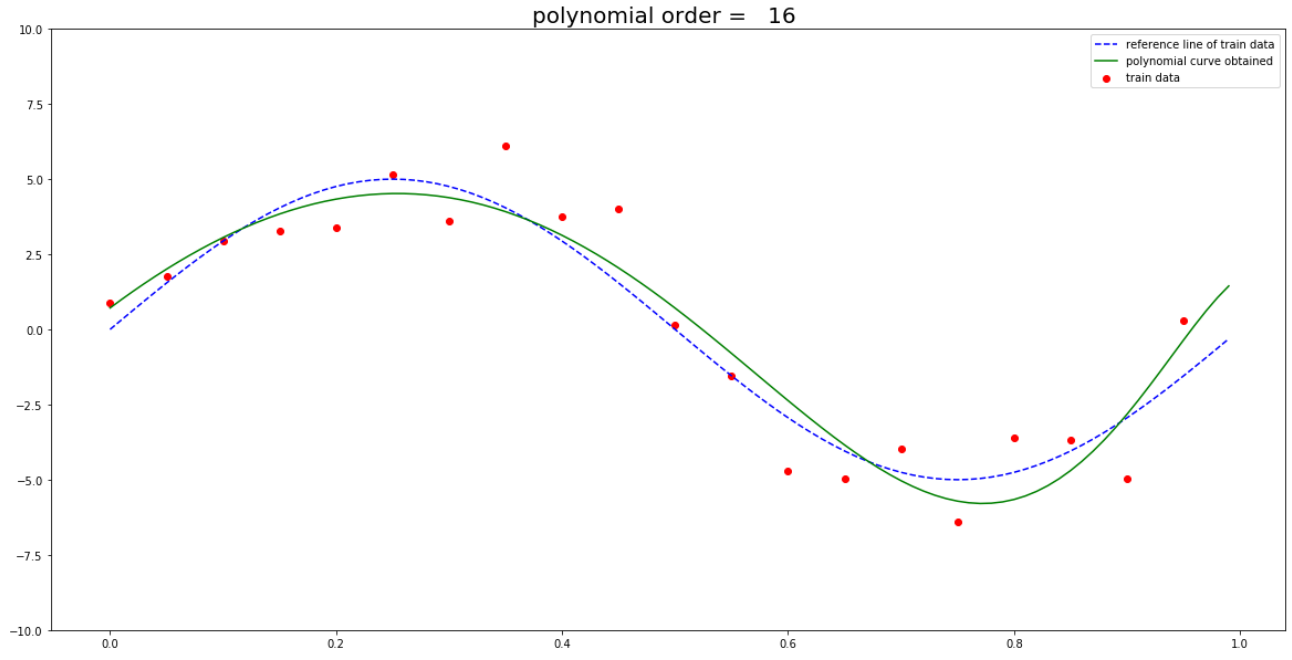
In [24]:

```
#### polynomial basis curve fitting with regularisation

poly_order = [3,16,20,25]
for p in range(4):
    phi = np.zeros((len(x),poly_order[p]+1))
    for i in range(poly_order[p]+1):
        phi[:,i] = x**i
    phi_t = np.transpose(phi)
    I = np.identity(np.shape(phi)[1])
    lam = 0.001                     # regularization term
    w = np.matmul(np.matmul(np.linalg.pinv(np.matmul(phi_t,phi)+lam*I,rcond=
1e-500),phi_t),y_noise)

    import matplotlib.pyplot as plt
    import numpy as np

    x_n = np.arange(0,1,0.01)
    y = 5*np.sin(2*np.pi*x_n)
    phi_n = np.zeros((len(x_n),poly_order[p]+1))
    for i in range(poly_order[p]+1):
        phi_n[:,i] = np.power(x_n,i)

    out_n = np.matmul(np.transpose(w),np.transpose(phi_n))
    plt.figure(figsize=(20,10))
    plt.plot(x_n, y,'--b')
    plt.scatter(x, y_noise,color ='red')
    plt.plot(x_n,out_n,'g')
    plt.ylim(-10,10)
    plt.title('polynomial order =  % i '% poly_order[p],fontsize = 20)
    plt.legend(['reference line of train data', 'polynomial curve obtained'
,'train data'])
    plt.show()
```

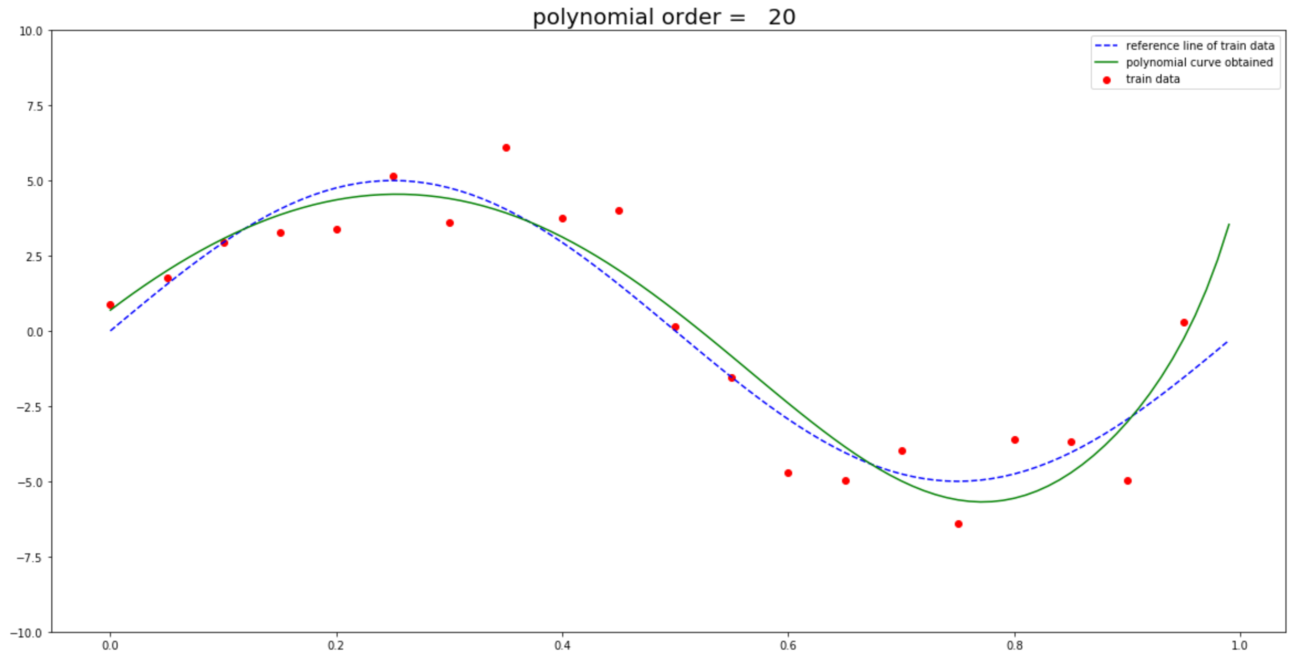




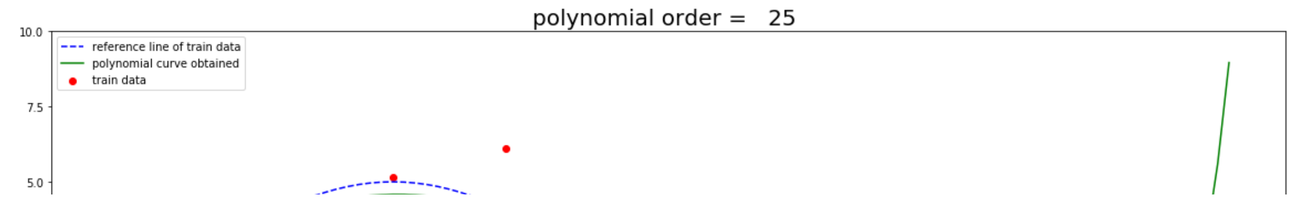
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0



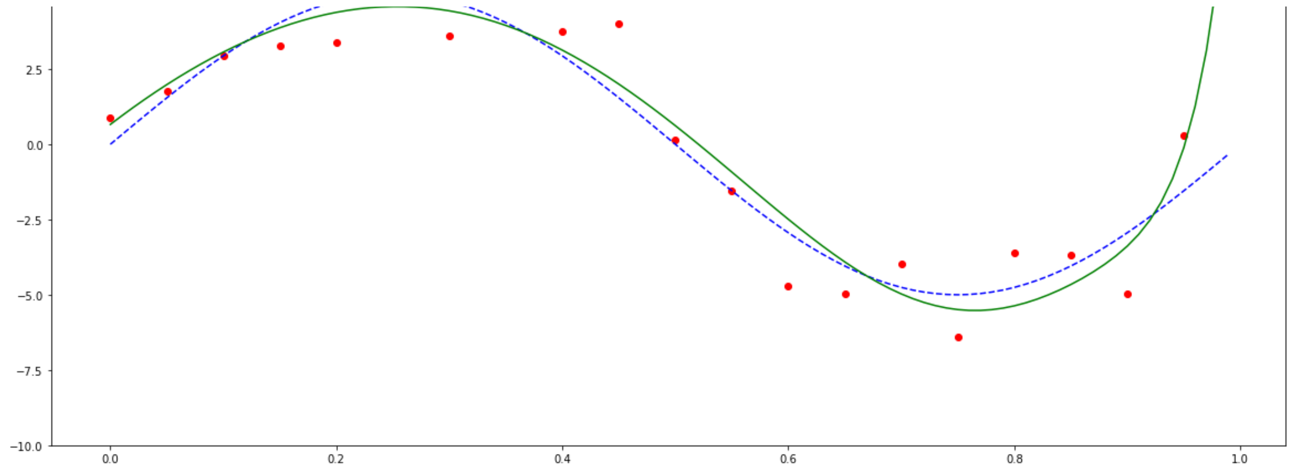
polynomial order = 16
reference line of train data
polynomial curve obtained
train data
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0



polynomial order = 20
reference line of train data
polynomial curve obtained
train data
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0.0
0.2
0.4
0.6
0.8
1.0



polynomial order = 25
reference line of train data
polynomial curve obtained
train data
10.0
7.5
5.0



The curve fit obtained with regularization is a better model as it behaves similar to the model but does not blidly learn the train data points.The green lines in the above graphs are the curves of the model which go along the reference sine line but with much less variability compared to the models without regularization.Suppose any new data point is a test sample on this model the model seems to perform better accordinf to the training information.Thus the problem of overfitting can be overcome