stint_6

February 5, 2021

[1]:
```
import pandas as pd
import pickle as pkl
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
import random
```

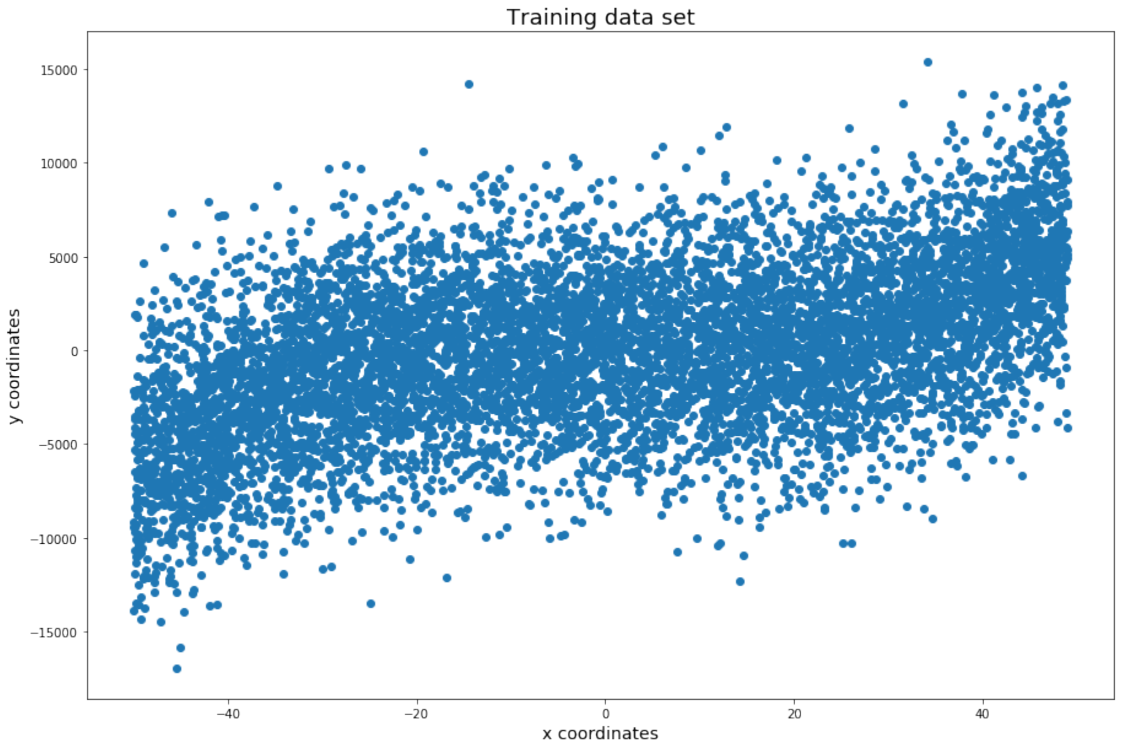
0.0.1 Importing the dataset

[2]:
```
train_data = pd.read_pickle('data/train.pkl')
test_data = pd.read_pickle('data/test.pkl')
print(len(train_data))
```

```
8000
```

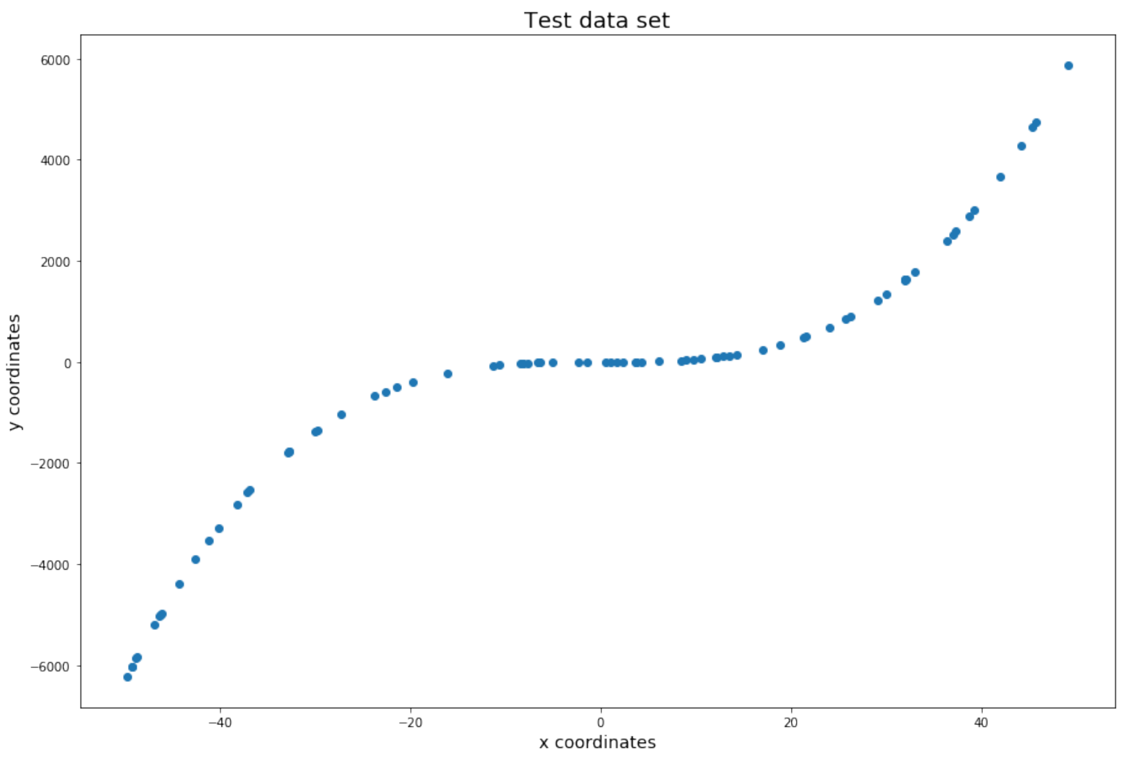
0.0.2 Plotting the training data set

[3]:
```
plt.figure(figsize=(15,10))
plt.scatter(train_data[:,0],train_data[:,1])
plt.xlabel('x coordinates',fontsize = 14)
plt.ylabel('y coordinates',fontsize = 14)
plt.title('Training data set', fontsize = 18)
plt.show()
```



0.0.3 Plotting the test dataset

[4]:
```
plt.figure(figsize=(15,10))
plt.scatter(test_data[:,0],test_data[:,1])
plt.xlabel('x coordinates',fontsize = 14)
plt.ylabel('y coordinates',fontsize = 14)
plt.title('Test data set', fontsize = 18)
plt.show()
```



[5]:
```python
def PolyCoefficients(x, coeffs):
    """ Returns a polynomial for ``x`` values for the ``coeffs`` provided.

    The coefficients must be in ascending order (``x**0`` to ``x**o``).
    """
    o = len(coeffs)
    #print(f'# This is a polynomial of order {ord}.')
    y = 0
    for i in range(o):
        y += coeffs[i]*x**i

#print("returning ",y)
    return y
```

0.0.4 Setting some constants and shuffling the training data

[6]:
```python
# Setting some constants
poly_deg = 20
divs = 10
part_size = int(len(train_data)/divs)
random.shuffle(train_data)
```

## A brief explanation of what function does the method, LinearRegression().fit() performs. Let us assume that our current training dataset D consists of ‘n’ instances. Let the degree of polynomial be ‘*deg*’. Now, there will be (deg+1) coefficients (after including the constant term as well). The given function takes 2 inputs: * Matrix containing the input features for each instance (n X $(deg + 1)$) matrix * Corresponding actual y (the ‘y’ value which was observed in the real world) which is a (n X 1 array)

As far as the theoretical part of linear regression is concerned, we are aware of the **gradient descent method** which is used to obtain the coefficients which reduce the average value of the squared error to be the minimum. Also, in gradient descent we usually make small steps. Thus, for the jth coefficient, we do the following:

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Here, alpha is predetermined and can be manipulated. If the alpha is very small, chances are that the number of iterations required for convergence will be very large. If alpha is very large, chances are that we escape the minima(jump over the local minima).

A thing we noticed We noticed that the LinearRegression.fit() does not obtain optimal coefficients always. We tailormade a test case for function $f(x) = 1 + 2x + 3x^2 + 3x^3 + 2x^4 + x^5$ and kept training dataset size as 2 for the 1st case and 9 for the second case. In the 1st case, the coefficients produced did were not optimal and overall MSE was not zero which proves that the **internal implementation does not go all the way but stops after some fixed precision/error threshold.**(refer to manipulation of alpha). But in second case, since number of training examples is more, the error with non-optimal coefficients would have been large and hence, the LinearRegression.fit() would have given some more iterations and reach d optimal value.

In this case, our train set is just 2 instances. As a result, the original coefficients are not restored.

```
[7]: coeffs=[1,2,3,3,2,1]
     ########################################
     x_arr=np.arange(1,3,1)
     #x_arr = x_arr[:,np.newaxis]

     #print("arange is ", x_arr)
     poly = PolynomialFeatures(degree=len(coeffs)-1)
     x_check = poly.fit_transform(x_arr[:,np.newaxis]) # now a matrix of size (80 X
      ↪(n+1))
     #print("x check is ",x_check)
     regr = linear_model.LinearRegression()
     y_check=[]
     for i in range(0,len(x_arr)):
```

```
    # print(i)
     #print(x_arr[i])
    y_check.append(PolyCoefficients(x_arr[i],coeffs))
#x_arr[:,np.newaxis]
#print("y check is ",y_check)
regr.fit(x_check,y_check)
regr.coef_[0]=regr.intercept_


print("intercept,coefficient: ", regr.coef_)
```

```
intercept,coefficient:  [7.74216867 0.0746988  0.22409639 0.52289157 1.12048193
2.31566265]
```

In this case, our train set is just 9 instances. As a result, the original coefficients are restored as sufficient data points are present.

[8]:
```
coeffs=[1,2,3,3,2,1]
#######################################
x_arr=np.arange(1,10,1)
#x_arr = x_arr[:,np.newaxis]

#print("arange is ", x_arr)
poly = PolynomialFeatures(degree=len(coeffs)-1)
x_check = poly.fit_transform(x_arr[:,np.newaxis]) # now a matrix of size (80 X␣
 ↪(n+1))
#print("x check is ",x_check)
regr = linear_model.LinearRegression()
y_check=[]
for i in range(0,len(x_arr)):
    # print(i)
     #print(x_arr[i])
    y_check.append(PolyCoefficients(x_arr[i],coeffs))
#x_arr[:,np.newaxis]
#print("y check is ",y_check)
regr.fit(x_check,y_check)
regr.coef_[0]=regr.intercept_


print("intercept,coefficient: ", regr.coef_)
```

```
intercept,coefficient:  [1. 2. 3. 3. 2. 1.]
```

Assignment specific use of LinearRegression.fit():

Each training data set size is 8000/10=800 and so, the first argument is a matrix of size (800 X (deg+1)) and the second argument is an array of (800 X 1).

0.1 Details regarding training of the models

For the given data sets, we are trying to fit a linear regression model on the training dataset for polynomials ranging from degree 1 to 20. Let's say that our regression function is a polynomial of degree $'n'$ and hence, the polynomial would be **determined by the** $(n+1)$ **coefficients** (after including the constant term as well). Also, let the size of the entire training set be 'sz'. We are ***partitioning the training dataset into 10 equal parts*** (after randomly shuffling the entries in the training dataset) each of size '$sz/10$' and then training a polynomial of degree 'n' on each of these partitions. In other words, we are finding the $(n+1)$ coefficients by linear regression on each of the partitions as a training dataset. So, for a fixed polynomial degree, we will have 10 classifiers. It is worthwhile to note that **each of these classifiers can be represented by a** $(n+1)$ **length vector which corresponds to the (n+1) coefficients of the 'n' degree polynomial.**

To summarize, Number of models we are using: 20

Number of different versions(realizations) of a model using a particular degree: 10

In the outer loop, we are iterating over integers from 1 to 20 (the degrees of the polynomials). In the inner loop, we are training a polynomial of degree 'deg' 10 times (each time on a different partition of the dataset) and using the coefficients obtained to calculate the bias and variance quantities for a model using polynomials of a particular degree 'deg'. **Our net model for a given degree of polynomial will be the average of the coefficients of these classifiers.**

[9]:
```python
plot_curves=[]

# keeping track of degrees (length = poly_deg)
degs = [0] * poly_deg
bias_sqr = [0] * poly_deg # initializing with zeroes
bias_mean = [0] * poly_deg # initializing with zeroes
variance = [0] * poly_deg # initializing with zeroes
noise = [0] * poly_deg # initializing with zeroes
tot_error = [0] * poly_deg # initializing with zeroes

for deg in range(1, poly_deg+1):

    degs[deg-1] = deg
    print("degree being investigated is : ",deg)

    poly = PolynomialFeatures(deg)

    # here net model keeps track of the E[f_bar(x)] in the notation used in the␣
 ↪pdf (h_bar in our notation)
    net_model = [0] * (deg+1)

    # keeps track of all the H's (f_bar(x) as per assignment pdf) for a current␣
 ↪degree (length after final operation must be 'divs')
    models = []

    for part in range(0,divs):
        # extracting the data to be used for training in current iteration
        x_c = train_data[part*part_size:(part+1)*part_size ,0][:,np.newaxis]
        y_c = train_data[part*part_size:(part+1)*part_size ,1]

        # getting the other powers
        x_mat = poly.fit_transform(x_c)

        # fitting the model
        reg = LinearRegression().fit(x_mat,y_c)
        reg.coef_[0] = reg.intercept_
        net_model+= reg.coef_

        # appending the coefficients derived in this current realization
        models.append(reg.coef_)
        # print('------------------------------------------')

    net_model/=divs # average over the coeffecents of the 10 realizations
    print('net model: ',net_model)

    # extracting x and y coordinates of test set
    x_test = test_data[:,0]
    y_test = test_data[:,1]

    # getting higher powers of the x coordinates of the test set
    x_mat = poly.fit_transform(x_test[:,np.newaxis])

    # size of net model is (deg+1 X 1) -> h_bar
    y_predic = np.dot(x_mat, net_model)


    ######### VARIANCE ########################
    for part in range(0,divs):
        res = np.dot(x_mat,(net_model - models[part]))
        variance[deg-1] += np.dot(res,res)/(len(res))
    variance[deg-1]/=divs;
    print("Calculated Variance: ",variance[deg-1])


    ######### BIAS ######################################
    plot_curves.append({"predicted":y_predic,"expected":y_test, "x_c":x_test})

    bias = y_predic - y_test
    bias_mean[deg-1]=np.sum(np.abs(bias))/len(bias)
    print("bias absolute mean: ", bias_mean[deg-1])
    bias_sqr[deg-1] = np.dot(bias, bias)/len(bias)
    print("Bias squared: ",bias_sqr[deg-1])
```

```
############## TOT ERROR IS #############################

tot_err_mat=np.dot(x_mat, np.array(models).T) - np.vstack(y_test)
squared_err=np.square(tot_err_mat)
tot_error[deg-1]=np.sum(squared_err)/(divs*len(test_data))
print("TOTAL error is ", tot_error[deg-1])


############## NOISE ############################################
noise[deg-1]=tot_error[deg-1] - bias_sqr[deg-1] - variance[deg-1]
print("Noise found is ", noise[deg-1])

print("############################################################")
```

```
degree being investigated is :  1
net model:  [-7.13831561 74.15703863]
Calculated Variance:  42722.03415775878
bias absolute mean:  822.7527715127908
Bias squared:  1010592.9944223702
TOTAL error is  1053315.028580129
Noise found is  -8.731149137020111e-11
############################################################
degree being investigated is :  2
net model:  [78.68997047 74.06904193 -0.10649636]
Calculated Variance:  64676.486006278545
bias absolute mean:  814.2053808051334
Bias squared:  965363.7301849304
TOTAL error is  1030040.2161912089
Noise found is  0.0
############################################################
degree being investigated is :  3
net model:  [ 7.52703500e+01 -6.64426279e+00 -8.68419717e-02  5.49303976e-02]
Calculated Variance:  78492.73221894923
bias absolute mean:  94.94053768408783
Bias squared:  16843.399605654027
TOTAL error is  95336.13182460329
Noise found is  4.3655745685100555e-11
############################################################
degree being investigated is :  4
net model:  [ 1.78596687e+02 -6.72700648e+00 -4.99123084e-01  5.49821438e-02
  1.95901326e-04]
Calculated Variance:  90145.11299575878
bias absolute mean:  124.2987036279887
Bias squared:  19482.223840861785
TOTAL error is  109627.33683662058
Noise found is  1.4551915228366852e-11
```

```
############################################################
degree being investigated is :  5
net model:  [ 1.76848861e+02 -5.86904563e+00 -4.85869088e-01  5.31310777e-02
  1.88933236e-04  7.66411606e-07]
Calculated Variance:  177243.88665202298
bias absolute mean:  128.4619309713906
Bias squared:  20741.631310626602
TOTAL error is  197985.51796264955
Noise found is  -2.9103830456733704e-11
############################################################
degree being investigated is :  6
net model:  [ 2.41158122e+02 -6.59836054e+00 -1.00959723e+00  5.46591959e-02
  8.23212521e-04  1.83820912e-07 -1.88128251e-07]
Calculated Variance:  199155.48159041427
bias absolute mean:  144.1247432030748
Bias squared:  28025.517618670896
TOTAL error is  227180.99920908525
Noise found is  8.731149137020111e-11
############################################################
degree being investigated is :  7
net model:  [ 2.48698536e+02  6.50228746e+00 -1.07160519e+00  7.32104023e-03
  9.14016642e-04  4.24691108e-05 -2.20440344e-07 -1.06750414e-08]
Calculated Variance:  200853.41495723967
bias absolute mean:  153.43763299267752
Bias squared:  30731.26792021261
TOTAL error is  231584.68287745214
Noise found is  -1.4551915228366852e-10
############################################################
degree being investigated is :  8
net model:  [ 1.46188225e+02  8.03866105e+00  4.06596161e-01  1.09311449e-03
 -2.41530629e-03  4.88851877e-05  2.15012519e-06 -1.25045399e-08
 -5.22046926e-10]
Calculated Variance:  243769.69590717676
bias absolute mean:  180.78969345369063
Bias squared:  44306.985886879644
TOTAL error is  288076.6817940561
Noise found is  -3.2014213502407074e-10
############################################################
degree being investigated is :  9
net model:  [ 1.49225546e+02  9.13804878e+00  3.62569778e-01 -6.70066024e-03
 -2.31216946e-03  6.17148231e-05  2.06777032e-06 -2.01794750e-08
 -5.00917803e-10  1.51754721e-12]
Calculated Variance:  257742.06072442108
bias absolute mean:  177.39671033602858
Bias squared:  42550.25621591332
TOTAL error is  300292.3169403347
Noise found is  2.6193447411060333e-10
############################################################
```

```
degree being investigated is :  10
net model:  [ 1.43515052e+02 -5.94009758e-03  6.08521682e-01  3.31190377e-02
 -3.21580890e-03  8.12027227e-06  3.16015824e-06  8.00898802e-09
 -1.02519813e-09 -3.52973490e-12  8.66291040e-14]
Calculated Variance:  272529.50130170875
bias absolute mean:  181.403572148694
Bias squared:  43889.721182696674
TOTAL error is  316419.22248440713
Noise found is  1.6880221664905548e-09
############################################################
degree being investigated is :  11
net model:  [ 1.67695524e+02 -8.71534135e-04  3.05770759e-05 -2.71125619e-02
 -1.08569038e-03  2.21175278e-04  5.36426904e-07 -2.40496409e-07
  2.96156505e-10  1.14122511e-10 -1.45465219e-13 -1.95140618e-14]
Calculated Variance:  316093.3438522543
bias absolute mean:  178.7310820180693
Bias squared:  47016.37485701068
TOTAL error is  363109.71870926366
Noise found is  -1.280568540096283e-09
############################################################
degree being investigated is :  12
net model:  [ 1.89587459e+02 -1.36804385e-04 -7.92157176e-06  2.38123883e-05
 -2.40942386e-03  1.41615839e-04  4.28359296e-06 -1.61133622e-07
 -3.47810445e-09  8.13813286e-11  1.46222029e-12 -1.47291658e-14
 -2.46230600e-16]
Calculated Variance:  304650.2166660943
bias absolute mean:  180.54986663894738
Bias squared:  49399.550781385755
TOTAL error is  354049.7674474868
Noise found is  6.752088665962219e-09
############################################################
degree being investigated is :  13
net model:  [ 1.33167442e+02  3.56009232e-06 -1.30016037e-08  3.53813116e-07
  7.87233807e-08  8.20047008e-05 -1.70166999e-06  5.33780785e-09
  1.92832537e-09 -8.18175719e-11 -6.34423514e-13  5.27942386e-14
  4.93729449e-17 -1.00446037e-17]
Calculated Variance:  361071.5182273021
bias absolute mean:  175.8415447006149
Bias squared:  50032.37408128042
TOTAL error is  411103.8923085884
Noise found is  5.820766091346741e-09
############################################################
degree being investigated is :  14
net model:  [ 9.02499140e+01  8.03734980e-12  1.21111006e-12  1.07876298e-12
  1.16272630e-12  6.08400862e-10  3.64493025e-10  2.29256720e-07
 -9.23249369e-10 -2.99284098e-10  9.05500115e-13  1.42737900e-13
 -2.07960728e-16 -2.35141744e-17 -5.54413464e-21]
Calculated Variance:  351990.1371832296
```

```
bias absolute mean:  169.9700546202577
Bias squared:  63286.50629885596
TOTAL error is  415276.643482084
Noise found is  -1.57160684466362e-09
############################################################
degree being investigated is :  15
net model:  [ 8.73581359e+01 -5.48852425e-13 -2.70766734e-14  8.13680812e-18
 -2.26899291e-15  1.49311110e-15 -1.48878395e-12  7.00646510e-13
 -6.22728845e-10  2.38568459e-10  3.66726032e-13 -3.15847774e-13
  1.03063479e-16  1.45790662e-16 -6.36798927e-20 -2.29277236e-20]
Calculated Variance:  379661.2404595643
bias absolute mean:  213.92632087553457
Bias squared:  94285.68852582194
TOTAL error is  473946.92898537696
Noise found is  -9.313225746154785e-09
############################################################
degree being investigated is :  16
net model:  [ 5.06822450e+01 -2.74016402e-15  5.74413922e-15 -3.86271601e-18
  1.60128940e-18  9.55848124e-16  9.93333951e-16  6.07190936e-13
  3.35899396e-13  2.51458055e-10  1.89246118e-13 -3.38162281e-13
 -6.55219230e-16  1.58286633e-16  5.04162504e-19 -2.52040506e-20
 -1.11055767e-22]
Calculated Variance:  453496.6534385484
bias absolute mean:  188.8245792876869
Bias squared:  112377.50597107605
TOTAL error is  565874.1594096224
Noise found is  -2.0372681319713593e-09
############################################################
degree being investigated is :  17
net model:  [ 4.93863410e+01  2.81789642e-16 -6.91420979e-17  3.81177641e-20
  7.35603986e-21  9.65080597e-21  1.05187616e-18  5.20464834e-18
  7.13307027e-16  1.92058691e-15  3.10018001e-13  2.08089037e-13
 -8.63249853e-16 -2.77311381e-16  6.21208507e-19  1.26390415e-19
 -1.32568363e-22 -1.94748560e-23]
Calculated Variance:  459459.8492848302
bias absolute mean:  259.3797374198608
Bias squared:  163582.774851507
TOTAL error is  623042.6241362921
Noise found is  -4.511093720793724e-08
############################################################
degree being investigated is :  18
net model:  [ 2.30080466e+01 -7.26487129e-17  1.39344275e-17 -6.67472613e-21
  4.07191316e-22  8.80769839e-22  1.31296789e-21  7.27100712e-19
  7.82593837e-19  4.97836810e-16  2.85355555e-16  2.19337499e-13
  6.93682068e-16 -2.95972043e-16 -1.25747854e-18  1.36478982e-19
  7.25419217e-22 -2.12586562e-23 -1.34703409e-25]
Calculated Variance:  554331.5532223546
bias absolute mean:  251.42724407876327
```

```
Bias squared:  200304.38202989954
TOTAL error is  754635.9352521463
Noise found is  -1.0780058801174164e-07
############################################################
degree being investigated is :  19
net model:  [ 2.06710957e+01  1.25286068e-18  3.64779482e-20  7.58024465e-24
 -2.56919973e-24  8.80657632e-27  2.66463254e-24  8.96699973e-24
  2.36066171e-21  5.35999776e-21  1.70014181e-18  2.09248456e-18
  7.79364821e-16  1.57799931e-16 -1.38993258e-18 -2.10152921e-19
  7.93768014e-22  9.48414028e-23 -1.46422578e-25 -1.44092173e-26]
Calculated Variance:  549674.2374760299
bias absolute mean:  326.5156340039574
Bias squared:  275813.2359326688
TOTAL error is  825487.473408765
Noise found is  6.635673344135284e-08
############################################################
degree being investigated is :  20
net model:  [ 4.15195158e+00 -2.85557750e-20 -5.96580816e-21  1.91113845e-24
 -1.97410009e-25 -1.64133603e-29  1.30631742e-27  5.66876575e-25
  1.01602476e-24  4.98685663e-22  6.34070112e-22  3.59877866e-19
  2.41027959e-19  1.65879585e-16  7.67846545e-19 -2.23065298e-19
 -1.24849708e-21  1.01607006e-22  6.63701367e-25 -1.55738250e-26
 -1.15630578e-28]
Calculated Variance:  658122.5667224834
bias absolute mean:  319.222807048203
Bias squared:  329177.6931332887
TOTAL error is  987300.2598555444
Noise found is  -2.277083694934845e-07
############################################################
```

0.1.1 Tabulating the bias, variance, noise, bias squared and other quantities

```
[10]: data = {'deg': degs, 'variance': variance, 'bias square':bias_sqr, 'mean bias':␣
       ↪bias_mean, 'noise': noise,"total error":tot_error}
      df = pd.DataFrame(data=data)
      print(df)
```

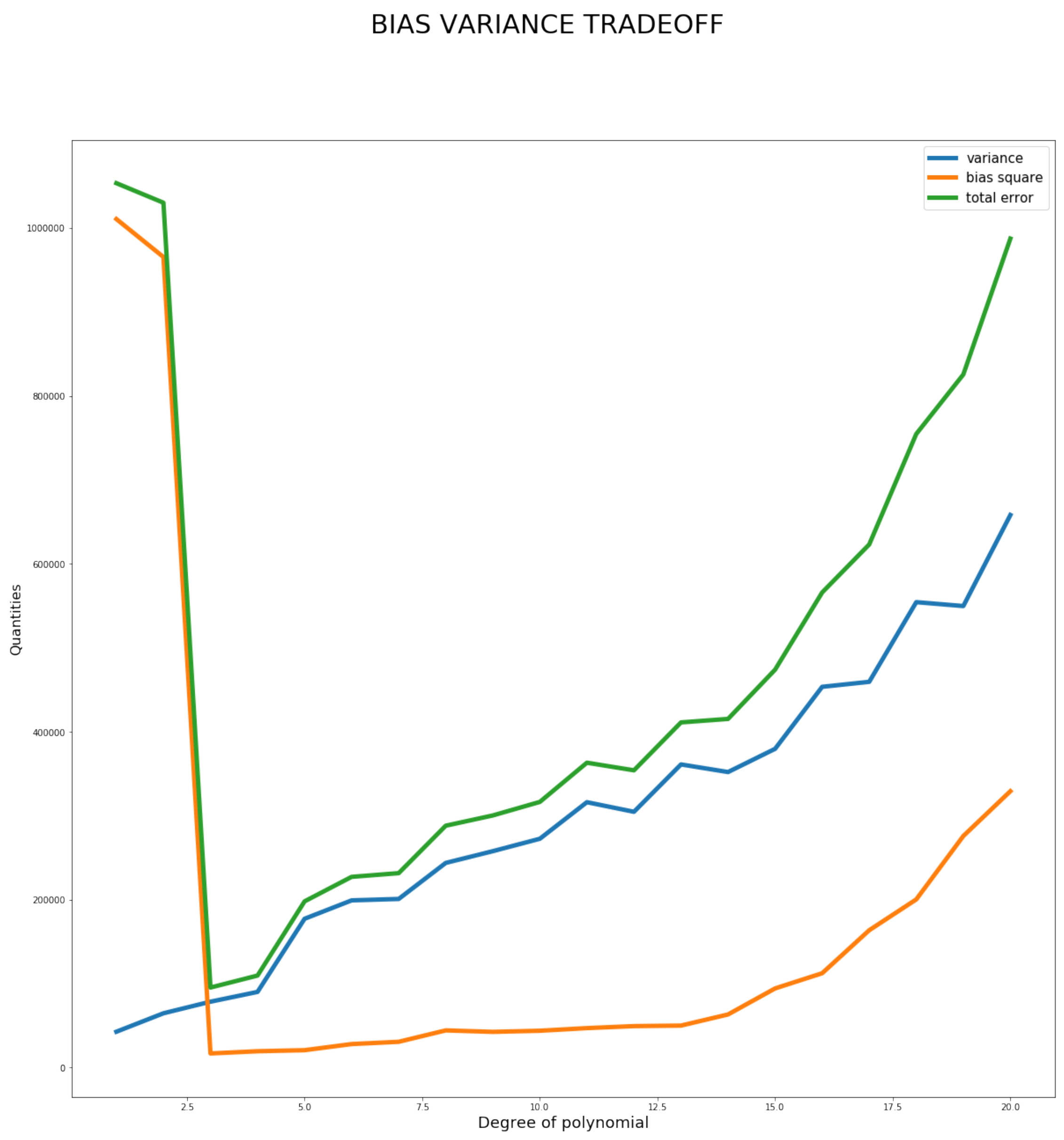
```
    deg       variance   bias square   mean bias         noise   total error
0     1   42722.034158  1.010593e+06  822.752772 -8.731149e-11  1.053315e+06
1     2   64676.486006  9.653637e+05  814.205381  0.000000e+00  1.030040e+06
2     3   78492.732219  1.684340e+04   94.940538  4.365575e-11  9.533613e+04
3     4   90145.112996  1.948222e+04  124.298704  1.455192e-11  1.096273e+05
4     5  177243.886652  2.074163e+04  128.461931 -2.910383e-11  1.979855e+05
5     6  199155.481590  2.802552e+04  144.124743  8.731149e-11  2.271810e+05
6     7  200853.414957  3.073127e+04  153.437633 -1.455192e-10  2.315847e+05
7     8  243769.695907  4.430699e+04  180.789693 -3.201421e-10  2.880767e+05
8     9  257742.060724  4.255026e+04  177.396710  2.619345e-10  3.002923e+05
```

```
9     10  272529.501302  4.388972e+04  181.403572  1.688022e-09  3.164192e+05
10    11  316093.343852  4.701637e+04  178.731082 -1.280569e-09  3.631097e+05
11    12  304650.216666  4.939955e+04  180.549867  6.752089e-09  3.540498e+05
12    13  361071.518227  5.003237e+04  175.841545  5.820766e-09  4.111039e+05
13    14  351990.137183  6.328651e+04  169.970055 -1.571607e-09  4.152766e+05
14    15  379661.240460  9.428569e+04  213.926321 -9.313226e-09  4.739469e+05
15    16  453496.653439  1.123775e+05  188.824579 -2.037268e-09  5.658742e+05
16    17  459459.849285  1.635828e+05  259.379737 -4.511094e-08  6.230426e+05
17    18  554331.553222  2.003044e+05  251.427244 -1.078006e-07  7.546359e+05
18    19  549674.237476  2.758132e+05  326.515634  6.635673e-08  8.254875e+05
19    20  658122.566722  3.291777e+05  319.222807 -2.277084e-07  9.873003e+05
```

1 Bias squared variance curve

[11]:
```
fig=plt.figure(figsize=(20,20))
plt.plot(degs,variance, label='variance',linewidth=5)
plt.plot(degs,bias_sqr, label='bias square',linewidth=5)
plt.plot(degs,tot_error, label='total error',linewidth=5)
fig.suptitle('BIAS VARIANCE TRADEOFF', fontsize=30)
plt.xlabel('Degree of polynomial', fontsize=18)
plt.ylabel('Quantities', fontsize=16)
# fig.savefig('test.jpg')
plt.legend(prop={'size': 15})
```

[11]:
```
<matplotlib.legend.Legend at 0x7f54c50b4b10>
```



1.1 Explaining the curves obtained

1.2 Our note regarding bias

Bias is the difference between the average prediction of our model and the actual value which we are trying to predict. A model with high bias does not generalize the data well and oversimplifies the model. It always leads to a high error on training and test data. High bias arises due to our classifier being "biased" to a particular kind of solution (e.g. linear classifier). **It means that the model being used is not robust enough to produce an accurate prediction. In other words, bias is inherent to our model .**

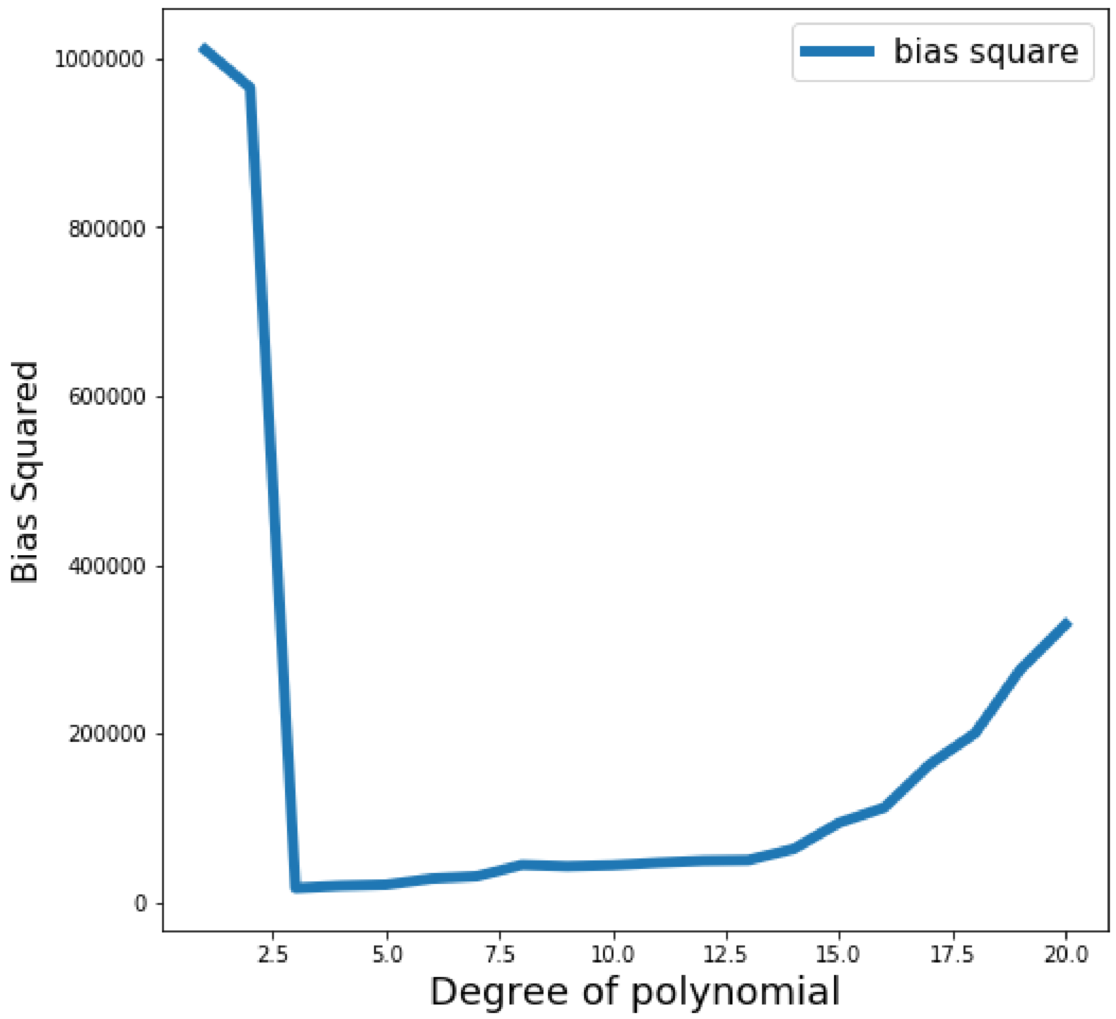
1.2.1 Model specific obervations related to bias

In order to calculate bias squared for a polynomial of specific degree, we are doing the following: Firstly, as has been mentioned above, we are training 10 versions of the same model. Now, let a particular point in the test set be <x,y>. Now, let's say that the 10 different realizations predict a y_value of $val_1, val_2, ...val_{10}$ for this particular 'x'. Then, the contribution of this test point to squared bias is $[(val_1 + val_2 + ... + val_{10})/10 - y]^2$. Obviously, we average this squared bias over all the test points. [Here, (val_1+val_2+...+val_10)/10 : represents the average prediction we might make if we end up using a model of this particular polynomial degree].

In the assignment, **the trained model for polynomial degrees 1 and 2 is over simplified and does not fit the data well causing a high bias.** At degree 3, the bias comes out to be minimum. This matches our expectations as we see that the plot of the dataset seems to be fitting a cubic function best. Till degree 12, the bias remains almost constant or increases slightly. From 12 onward, we notice that bias squared increases rapidly, even though theoretically it shouldn't because we are actually overfitting. **The reason for this, we believe, is that at high degrees of polynomial, LinearRegression().fit() is not able to produce optimal results because of insufficient number of data points. The algorithm (gradient descent) is unable to produce precise coefficients. As demonstrated above, we showed that when LinearRegressin.fit() had to fit on a training data set of just 2 points, the optimal coefficients were not produced, rather, some other coefficients were produced. We feel that this is because 8000 (800 X 10) points aren't good enough for manipulating more than 13 coefficients (for polynomials of degree greater than 11). However, the same size is sufficient for a model of polynomial degree 3 as the number of coefficients to manipulate is less.** Moreover, there is noise in the training data set which may produce inaccuracy.

[12]:
```
fig=plt.figure(figsize=(8,8))
plt.plot(degs,bias_sqr, label='bias square',linewidth=5)
plt.xlabel('Degree of polynomial', fontsize=18)
plt.ylabel('Bias Squared', fontsize=16)
plt.legend(prop={'size': 15})
```

[12]:
```
<matplotlib.legend.Legend at 0x7f54c506e290>
```



1.3 A note about variance

Variance is the variability of a model prediction for a given data point. Again, imagine we can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the models.

In other words, variance captures how much your classifier changes if we train on a different training set. **It represents how "over-specialized" the classifier is to a particular training set (overfitting).**

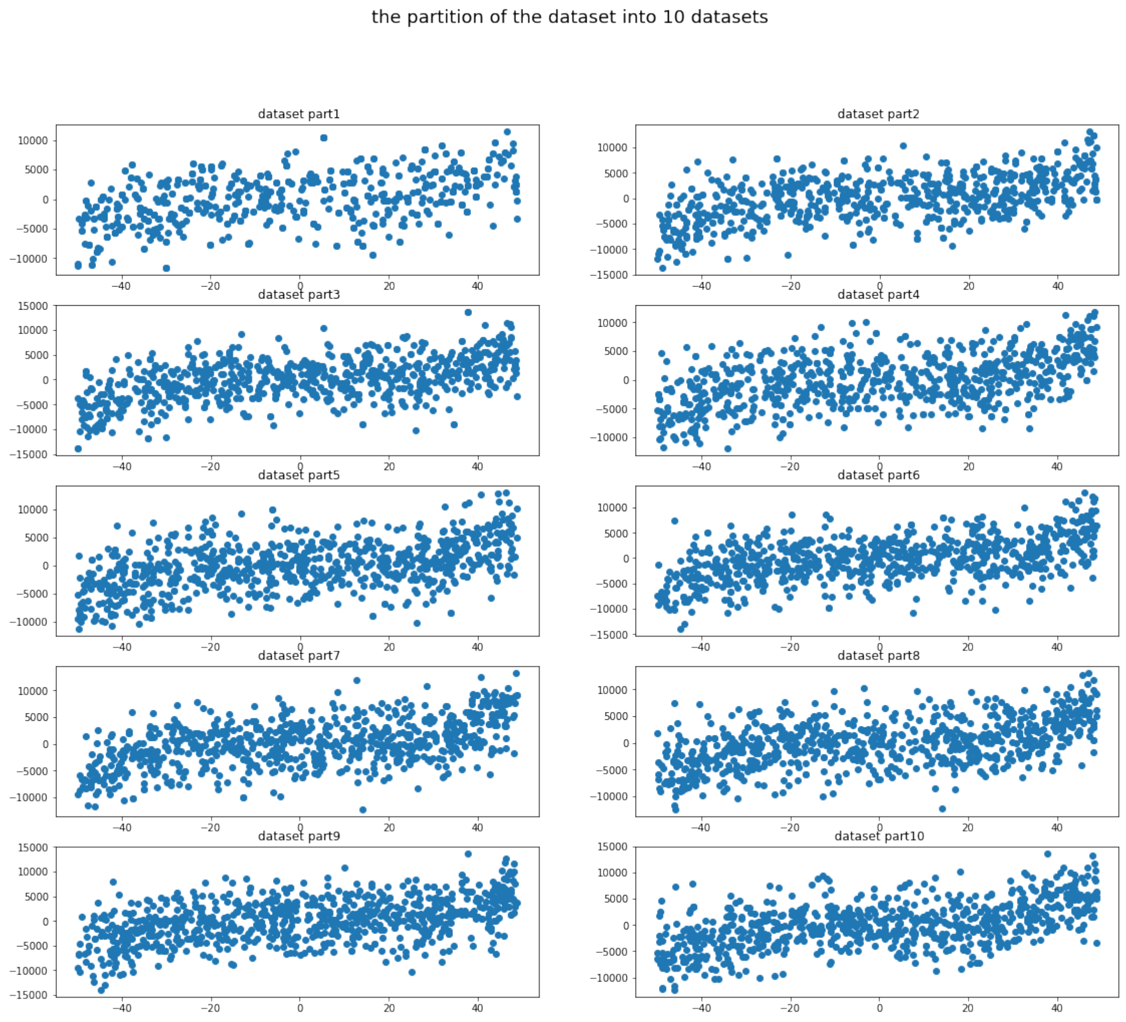
1.3.1 Assignment specific details about variance

In general, variance increases with increase in model complexity (here, the degree of the polynomial). This is because as we increase polynomial complexity, **it becomes more flexible for the model**

to fit even the minor deviations in the current training dataset partition. Each time the model is trained, the increased flexibility of the higher degree polynomials causes the coefficients to turn out considerably different due to differences in the training set (the classifier fails to generalize on the data). In other words, the model tries to fit the data points for each of the 10 training sets. As the degree increases, the model better fits each of these data chunks (each of size 800) . This results in higher variation among the different training sets. At higher degrees of polynomial, the variance begins to saturate at a high value as there is no better fitting of data left to do. The difference in the placement of coordinates in these training chunks can be seen below.

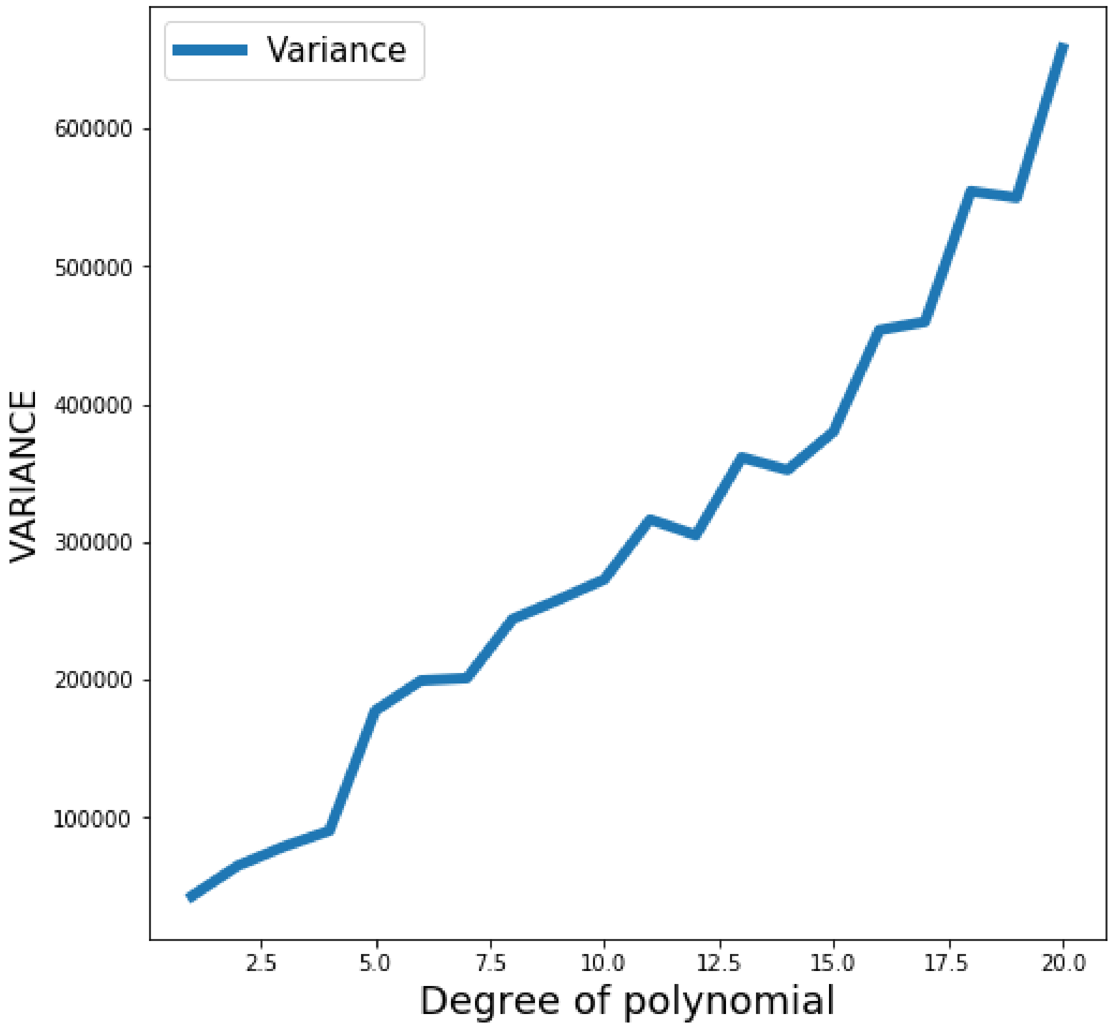
[13]:
```
nrows=5
ncols=2
fig, ax = plt.subplots(nrows, ncols, figsize=(19,16))
#print(type(ax[0]))
#print(ax)
part=0
for i in range(0,nrows):
    for j in range(0,ncols):
     #random_df.plot(kind='scatter', x=col, y='MEDV', ax=ax[i])
     #plt.plot(x=train_data[part*part_size:(part+1)*part_size ,0],␣
 ↪y=train_data[part*part_size:(part+1)*part_size ,1], ax=ax[i])
        x_c=train_data[part*part_size:(part+1)*part_size ,0]
        x_c= np.asarray(x_c).reshape(-1)
        y_c=train_data[part*part_size:(part+1)*part_size ,1]
        y_c= np.asarray(y_c).reshape(-1)
        #print(x_c)
        #print(type(x_c))
        ax[i][j].scatter(x_c,y_c)
        ax[i][j].set_title(" dataset part" + str(part+1))
        part+=1
        if part>=divs:
            break
    if part>=divs:
            break
fig.suptitle('the partition of the dataset into 10 datasets', fontsize=18)

#fig.subtitle('My Scatter Plots')
#fig.tight_layout()
#fig.subplots_adjust(top=0.95)

#plt.show()
#plt.clf()
#plt.close()
```

[13]:
```
Text(0.5, 0.98, 'the partition of the dataset into 10 datasets')
```



```
[14]: fig=plt.figure(figsize=(8,8))
      plt.plot(degs,variance, label='Variance',linewidth=5)
      plt.xlabel('Degree of polynomial', fontsize=18)
      plt.ylabel('VARIANCE', fontsize=16)
      plt.legend(prop={'size': 15})
```

```
[14]: <matplotlib.legend.Legend at 0x7f54c40e4790>
```

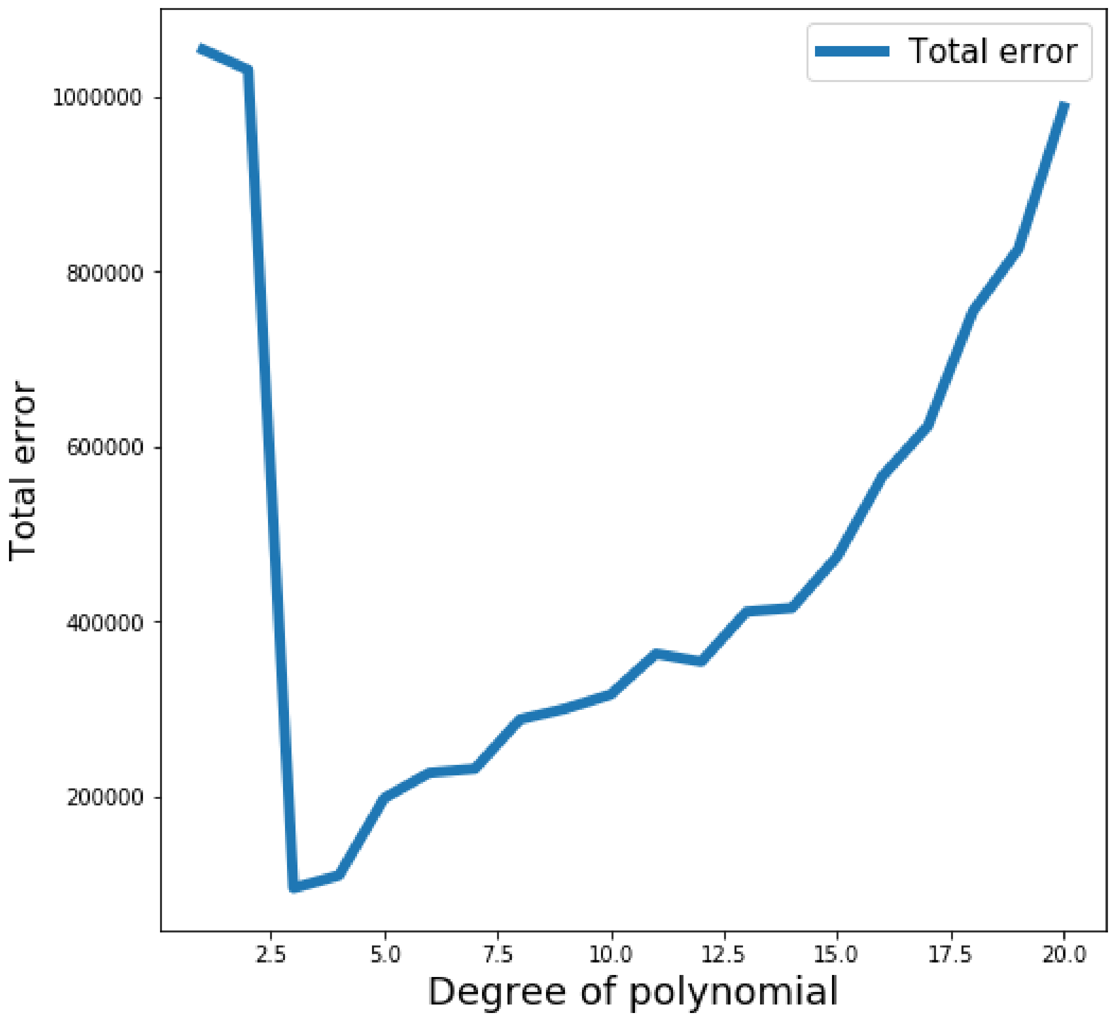


1.4 Observeration

We notice that the total error get minimzed at polynomial of degree 3. Details have been mentioned in the bias-variance tradeoff explaination below.

```
[15]: fig=plt.figure(figsize=(8,8))
      plt.plot(degs,tot_error, label='Total error',linewidth=5)
      plt.xlabel('Degree of polynomial', fontsize=18)
      plt.ylabel('Total error', fontsize=16)
      plt.legend(prop={'size': 15})
```

```
[15]: <matplotlib.legend.Legend at 0x7f54c4051dd0>
```



1.5 A detailed description as to why noise (irreducible error) does not change even when we change polynomial

Irreducible error is the **property of the dataset**. **The value of irreducible error must not change and is independent of the type of the algorithm being used**. It is the error which a model cannot escape, no matter how perfect it is. On a glance at the coordinates in the test data set, we noticed that the test data set has a one-one mapping and hence, we are able to predict, even before we trained our model, that the theoretical value of **irreducible error should be zero**. This prediction of ours was validated by the values we obtained while training using polynomials of different degrees. **The irreducible error is almost zero for all polynomial degrees (not exactly zero due to precision errors in python).** As expected, the value of irreducible error does not change as 'irreducible error' is the aspect of data.

1.6 A note on the bias-variance tradeoff

We explain bias variance tradeoff with help of some cases: For degrees 1 and 2: For polynomial degrees 1 and 2, the trained model is oversimplified (the polynomial degree is not sufficient enough to capture the insights from the dataset) and results in a high bias (underfitting). It performs equally badly for all 10 training sets, hence variance is low and hence, even on the test set, the predictions are almost equally bad for all the 10 realizations of the model.

For degree 3: At degree 3, total error (sum of bias and variance) is at its minimum, showing that the data is actually best fit on a cubic function.

From degrees 4 to 10: We see that bias keeps almost constant whereas variance increases. We already know that a cubic fit the test set best. The polynomials of degree greater than 3 are almost able to obtain the same level of bias as they manage to keep coefficients of higher 'x' powers (x^4 onwards) very small and hence, not much difference is able to arise. However, variance keeps on increasing and as even though the higher coeffs are low, the higher coeffs of the 10 realizations of a model are different to fit their corresponding train sets and hence, produce different values for the points in the test set.

1.7 Our comments on the dataset

Our perception about the datasets are as follows:

The training data set follows a cubic function trend which is why we obtain a low total error when training our model to be a polynomial of degree 3. The test data set and training data set seem to display different properties as far as noise is concerned. Evidence of this is the fact that the training dataset has a very high amount of noise (even for the same input 'x', there are several 'y' values in the training set). (A brief example regarding this would be the selling price of 5 identical houses being sold at quite different prices (2Cr, 3Cr, 2.5 Cr, 2.25 Cr, 3.5 Cr). However, the test dataset theoretically has zero noise as whenever the same input 'x' is present, the output is the same as well. In the training set, even for a fixed value of 'x', the corresponding 'y' values seem to be highly dispersed.

```
[ ]:
```

Created in Deepnote