HW1B_Regression

March 14, 2020

1 Linear and Polynomial Regression

1.1 IMPORTANT: make sure to rerun all the code from the beginning to obtain the results for the final version of your notebook, since this is the way we will do it before evaluting your notebook!!! Do not change data files or their names

Student name: Alessandro Valente ID Number: 1234429

1.2 1) Linear Regression on the Boston House Price dataset

1.3 Dataset description

The Boston House Price Dataset involves the prediction of a house price in thousands of dollars given details about the house and its neighborhood.

The dataset contains a total of 500 observations, which relate 13 input features to an output variable (house price). The variable names are as follows: 1. CRIM: per capita crime rate by town. 2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft. 3. INDUS: proportion of nonretail business acres per town. 4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise). 5. NOX: nitric oxides concentration (parts per 10 million). 6. RM: average number of rooms per dwelling. 7. AGE: proportion of owner-occupied units built prior to 1940. 8. DIS: weighted distances to five Boston employment centers. 9. RAD: index of accessibility to radial highways. 10. TAX: full-value property-tax rate per $$$10,000. 11. PTRATIO: pupil-teacher ratio by town. 12. B: 1000*(Bk – 0.63)$ˆ2$ where Bk is the proportion of blacks by town. 13. LSTAT: % lower status of the population. 14. MEDV: Median value of owner-occupied homes in $$$1000s.

```
In [1]: #needed if you get the IPython/javascript error on the in-line plots
        %matplotlib inline

        import matplotlib.pyplot as plt
        import numpy as np
        import scipy as sp
        from scipy import stats
```

1.4 Import Data

Load the data from a .csv file

TO DO: insert a seed number (e.g., your ID number (matricola))

```
In [2]: # Load the data
        IDnumber = 1234428#PLACE A RANDOM SEED (your ID, but try also to change and see what h
        np.random.seed(IDnumber)

        filename = "data/house.csv"

        Data = np.genfromtxt(filename, delimiter=';',skip_header=1)
```

2 A quick overview of data

To inspect the data you can use the method describe()

```
In [3]: dataDescription = stats.describe(Data)
        print(dataDescription)

        print ("Shape of data array: " + str(Data.shape))

        #for more interesting visualization: use Panda!
```

```
DescribeResult(nobs=500, minmax=(array([6.3200e-03, 0.0000e+00, 4.6000e-01, 0.0000e+00, 3.8500
       3.5610e+00, 2.9000e+00, 1.1296e+00, 1.0000e+00, 1.8700e+02,
       1.2600e+01, 3.2000e-01, 1.7300e+00, 5.0000e+01]), array([ 88.9762, 100.    ,  27.74   ,
       100.    ,  12.1265,  24.    , 711.    ,  22.    , 396.9    ,
        37.97   , 500.    ])), mean=array([3.65578576e+00, 1.15000000e+01, 1.11317400e+01, 7.00
       5.54451400e-01, 6.28297000e+00, 6.84246000e+01, 3.81193180e+00,
       9.64200000e+00, 4.09624000e+02, 1.84286000e+01, 3.56208040e+02,
       1.26987400e+01, 2.25680000e+02]), variance=array([7.47252195e+01, 5.48905812e+02, 4.761
       1.35838329e-02, 4.97618330e-01, 7.99314163e+02, 4.46304043e+00,
       7.59617595e+01, 2.85605197e+04, 4.67663531e+00, 8.41657137e+03,
       5.13361942e+01, 8.52982926e+03]), skewness=array([ 5.17851447,  2.19924065,  0.29469001
        0.40691344, -0.58266003,  0.99061328,  0.9886802 ,  0.65098333,
       -0.79127942, -2.85972639,  0.89051925,  1.09625625]), kurtosis=array([36.3384518 ,  3.8
        1.84311394, -0.99523636,  0.43299159, -0.90519243, -1.16696682,
       -0.30903922,  7.00864873,  0.44527324,  1.4266838 ]))
Shape of data array: (500, 14)
```

3 Split data in training, validation and test sets

Given m total data, denote with m_{tv} the part used for training and validation. Keep m_t data as training data, $m_{val} := m_{tv} - m_t$ as validation data and $m_{test} := m - m_{val} - m_t = m - m_{tv}$. For instance one can take $m_t = m/2$ of the data as training, $m_{val} = m/4$ validation and $m_{test} = m/4$ as testing. Let us define:

- S_t the training data set
- S_{val} the validation data set
- S_{test} the testing data set

The reason for this splitting is as follows:

TRAINING DATA: The training data are used to compute the empirical loss

$$L_S(h) = \frac{1}{m_t} \sum_{z_i \in S_t} \ell(h, z_i)$$

which is used to estimate h in a given model class $\mathcal{H}$. i.e.

$$\hat{h} = \text{arg min}_{h \in \mathcal{H}} L_S(h)$$

VALIDATION DATA: When different model classes are present (e.g. of different complexity such as linear regression which uses a different number d_j of regressors $x_1, \ldots x_{d_j}$), one has to choose which one is the "best" complexity. Let $\mathcal{H}_{d_j}$ be the space of models as a function of the complexity d_j and let

$$\hat{h}_{d_j} = \text{arg min}_{h \in \mathcal{H}_{d_j}} L_S(h)$$

One can estimate the generalization error for model $\hat{h}_{d_j}$ as follows:

$$L_{\mathcal{D}}(\hat{h}_{d_j}) \simeq \frac{1}{m_{val}} \sum_{z_i \in S_{val}} \ell(\hat{h}_{d_j}, z_i)$$

and then choose the complexity which achieves the best estimate of the generalization error

$$\hat{d}_j := \text{arg min}_{d_j} \frac{1}{m_{val}} \sum_{z_i \in S_{val}} \ell(\hat{h}_{d_j}, z_i)$$

TESTING DATA: Last, the test data set can be used to estimate the performance of the final estimated model $\hat{h}_{\hat{d}_j}$ using:

$$L_{\mathcal{D}}(\hat{h}_{\hat{d}_j}) \simeq \frac{1}{m_{test}} \sum_{z_i \in S_{test}} \ell(\hat{h}_{\hat{d}_j}, z_i)$$

TO DO: split the data in training, validation and test sets (50%-25%-25%)

```
In [4]: #get number of total samples
        num_total_samples = Data.shape[0]

        print ("Total number of samples: ", num_total_samples)

        #size of each chunk of data (1/4 each): 2 of them for training, 1 for validation, 1 fo
        size_chunk = num_total_samples/4

        print ("Size of each chunk of data: ", size_chunk)

        #shuffle the data
        np.random.shuffle(Data)


        #training data

        X_training = np.split(Data, 2)[0][:,0:13]
        Y_training = np.split(Data, 2)[0][:,13:14]

        print(Data[1:2, :])
        print(X_training[1:2, :])
        print(Y_training[1:2, :])
        print ("Training input data size: ", X_training.shape)
        print ("Training output data size: ", Y_training.shape)

        #validation data, to be used to choose among different models
        X_validation = np.split(Data, 4)[2][:,0:13]
        Y_validation = np.split(Data, 4)[2][:,13:14]
        print ("Validation input data size: ", X_validation.shape)
        print ("Validation output data size: ", Y_validation.shape)

        #test data, to be used to estimate the true loss of the final model(s)
        X_test = np.split(Data, 4)[3][:,0:13]
        Y_test = np.split(Data, 4)[3][:,13:14]
        print ("Test input data size: ", X_test.shape)
        print ("Test output data size: ", Y_test.shape)
```

```
Total number of samples:  500
Size of each chunk of data:  125.0
[[1.1027e-01 2.5000e+01 5.1300e+00 0.0000e+00 4.5300e-01 6.4560e+00
  6.7800e+01 7.2255e+00 8.0000e+00 2.8400e+02 1.9700e+01 3.9690e+02
  6.7300e+00 2.2200e+02]]
[[1.1027e-01 2.5000e+01 5.1300e+00 0.0000e+00 4.5300e-01 6.4560e+00
  6.7800e+01 7.2255e+00 8.0000e+00 2.8400e+02 1.9700e+01 3.9690e+02
  6.7300e+00]]
[[222.]]
Training input data size:  (250, 13)
Training output data size:  (250, 1)
Validation input data size:  (125, 13)
Validation output data size:  (125, 1)
Test input data size:  (125, 13)
Test output data size:  (125, 1)
```

4 Data Normalization

It is common practice in Statistics and Machine Learning to scale the data (= each variable) so that it is centered (zero mean) and has standard deviation equal to 1. This helps in terms of numerical conditioning of the (inverse) problems of estimating the model (the coefficients of the linear regression in this case), as well as to give the same scale to all the coefficients.

```
In [5]: #scale the data

        # standardize the input matrix
        from sklearn import preprocessing
```

```
    # the transformation is computed on training data and then used on all the 3 sets
    scaler = preprocessing.StandardScaler().fit(X_training)

    X_training = scaler.transform(X_training)
    print ("Mean of the training input data:", X_training.mean(axis=0))
    print ("Std of the training input data:",X_training.std(axis=0))

    X_validation = scaler.transform(X_validation) # use the same transformation on validat
    print ("Mean of the validation input data:", X_validation.mean(axis=0))
    print ("Std of the validation input data:", X_validation.std(axis=0))

    X_test = scaler.transform(X_test) # use the same transformation on test data
    print ("Mean of the test input data:", X_test.mean(axis=0))
    print ("Std of the test input data:", X_test.std(axis=0))
```

```
Mean of the training input data: [-3.66817687e-16  1.08357767e-16  3.73034936e-16  4.26325641e
  4.97424324e-15  7.77999887e-15 -5.79092330e-16  6.56363852e-16
 -5.99520433e-17  1.15740750e-17  1.67528214e-14  6.24167384e-15
 -1.30651046e-15]
Std of the training input data: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean of the validation input data: [-0.06110281  0.07375861  0.02012471  0.19817407  0.0933710
  0.00834329 -0.05710738  0.03622412  0.04016298 -0.02504699  0.02674243
 -0.06300146]
Std of the validation input data: [0.81237733 1.1790915  1.01853886 1.32682606 0.98325564 1.10
 1.04641355 1.06587691 1.00559886 1.02242134 1.01883128 0.9619215
 0.95078514]
Mean of the test input data: [ 0.03066405  0.14742491  0.06713101  0.12611077  0.07801372 -0.0
  0.02162265 -0.00698105  0.0910247   0.12258647  0.03364521 -0.17150553
 -0.01404014]
Std of the test input data: [0.98070556 1.12388415 1.00234247 1.22189269 1.06924267 1.00655721
 1.06436936 1.18451436 1.03482106 0.99653644 1.01828891 1.18957881
 0.93213382]
```

5 Model Training

The model is trained (= estimated) minimizing the empirical error

$$L_S(h) := \frac{1}{m_t} \sum_{z_i \in S_t} \ell(h, z_i)$$

When the loss function is the quadratic loss

$$\ell(h, z) := (y - h(x))^2$$

we define the Residual Sum of Squares (RSS) as

$$RSS(h) := \sum_{z_i \in S_t} \ell(h, z_i) = \sum_{z_i \in S_t} (y_i - h(x_i))^2$$

so that the training error becomes

$$L_S(h) = \frac{RSS(h)}{m_t}$$

We recal that, for linear models we have $h(x) =< w, x >$ and the Empirical error $L_S(h)$ can be written in terms of the vector of parameters w in the form

$$L_S(w) = \frac{1}{m_t}\|Y - Xw\|^2$$

where Y and X are the matrices whose $i-$th row are, respectively, the output data y_i and the input vectors $x_i^\top$.

TO DO: compute the linear regression coefficients implementing the least square algorithm as done in the introductory lab **and** using np.linalg.lstsq from scikitlear

```
In [6]: #compute linear regression coefficients for training data

        #add a 1 at the beginning of each sample for training, validation, and testing (use ho

        # ADD YOUR CODE

        print(X_training[1:2, :])
        X_training=np.c_[np.ones(int(size_chunk*2)),X_training]
        print(X_training[1:2, :])

        X_validation=np.c_[np.ones(int(size_chunk)),X_validation]
        X_test=np.c_[np.ones(int(size_chunk)),X_test]
```

```
[[-0.39851843  0.67841305 -0.85368119 -0.23420572 -0.84119019  0.25111886
  -0.01524506  1.70741415 -0.15882881 -0.70789335  0.59626793  0.42763175
  -0.82941066]]
[[ 1.         -0.39851843  0.67841305 -0.85368119 -0.23420572 -0.84119019
   0.25111886 -0.01524506  1.70741415 -0.15882881 -0.70789335  0.59626793
   0.42763175 -0.82941066]]
```

```
In [7]: # Compute the least-squares solution using the same approach of LAB0

        n = len(X_training[:,1]);
        #print(n)


        A = np.zeros((14,14))
        b = np.zeros((14,1))
        for i in range(0,n):
          e = X_training[i,:].reshape(14,1)  #reshape to "matrix"
          A = A + np.matmul(e,np.transpose(e))
          b = b + Y_training[i]*e

        #print(A)
        w_hand = np.matmul(np.linalg.inv(A),b); # solve least squares


        # Compute the least-squares coefficients using linalg.lstsq
        w_np, RSStr_np, rank_Xtr, sv_Xtr = np.linalg.lstsq (X_training[:,0:14],Y_training[:],r

        print("LS coefficients by hand:", w_hand)
        print("LS coefficients with numpy lstsq:", w_np)
```

```
LS coefficients by hand: [[224.616     ]
 [ -8.41293851]
 [  3.42558884]
 [ -0.75500659]
 [  8.32122553]
 [-19.12389133]
 [ 40.6988505 ]
 [ -8.17088455]
 [-30.74966399]
 [ 16.50231282]
 [-18.43895046]
 [-17.98650666]
 [  6.02897429]
 [-22.85337374]]
LS coefficients with numpy lstsq: [[224.616     ]
 [ -8.41293851]
 [  3.42558884]
 [ -0.75500659]
 [  8.32122553]
 [-19.12389133]
 [ 40.6988505 ]
 [ -8.17088455]
 [-30.74966399]
 [ 16.50231282]
 [-18.43895046]
 [-17.98650666]
 [  6.02897429]
 [-22.85337374]]
```

```
In [8]: # compute Residual sums of squares by hand

        RSStr_hand = 0
        m_training=n

        for i in range(0,n):
            RSStr_hand += (Y_training[i]-np.matmul(X_training[i,:],w_hand))**2

        print("RSS by hand:",  RSStr_hand)
        print("RSS with numpy lstsq: ", RSStr_np)

        print("Empirical risk by hand:", RSStr_hand/m_training)
        print("Empirical risk with numpy lstsq:", RSStr_np/m_training)
```

```
RSS by hand: [371967.47204898]
RSS with numpy lstsq:  [371967.47204898]
Empirical risk by hand: [1487.8698882]
Empirical risk with numpy lstsq: [1487.8698882]
```

5.1 Data prediction

Compute the output predictions on both training and validation set and compute the Residual Sum of Sqaures (RSS) defined above, the Emprical Loss and the quantity $1 - R^2$ where

$$R^2 = \frac{\sum_{z_i \in S_t} (\hat{y}_i - \bar{y})^2}{\sum_{z_i \in S_t} (y_i - \bar{y})^2} \qquad \bar{y} = \frac{1}{m_t} \sum_{z_i \in S_t} y_i$$

is the so-called "Coefficient of determination" (COD)

TO DO: Compute these quantities on training, validation and test sets.

```
In [9]: def R2 (Y, Yp):
            m=len(Y)
            Ybar=np.sum(Y)/m
            #print(Y, Yp)
            #print (np.sum(Yp),np.sum(Y))
            Rsq1=Rsq2=0.0
            for i in range(0,m):
                Rsq1+=((Yp[i]-Ybar)**2)
                Rsq2+=((Y[i]-Ybar)**2)
            return Rsq1/Rsq2

        #Y=np.random.rand(10,1)
        #Yp=np.ones(10)
        #Yp/=2
        #print(R2(Y, Yp))
```

```
In [10]: #compute predictions on training and validation

         #prediction_training
         prediction_training = np.dot(X_training, w_np) #uso il w stimato con np (sono comunqu
         prediction_validation = np.dot(X_validation, w_np)
         prediction_test = np.dot(X_test, w_np)# COMPLETE

         #what about the loss for points in the validation and test data?
         m_validation = m_test = int(size_chunk)
         RSS_validation=0.0
         RSS_test=0.0
```

```
        for i in range(0,m_validation):
            RSS_validation += (Y_validation[i]-np.matmul(X_validation[i,:],w_hand))**2
            RSS_test       += (Y_test[i]      -np.matmul(X_test[i,:]      ,w_hand))**2

        print("RSS on validation data:",  RSS_validation)
        print("Loss estimated from validation data:", RSS_validation/m_validation)

        print("RSS on test data:",  RSS_test)
        print("Loss estimated from test data:", RSS_test/m_test)

        #another measure of how good our linear fit is given by the following (that is 1 - R^


        measure_training = 1-R2(Y_training, prediction_training)
        measure_validation = 1-R2(Y_validation, prediction_validation)
        measure_test = 1-R2(Y_test, prediction_test)

        print("Measure on Training Data (1-R^2):", measure_training)
        print("Measure on Validation Data(1-R^2):", measure_validation)
        print("Measure on Test Data(1-R^2):", measure_test)
```

```
RSS on validation data: [472550.49286003]
Loss estimated from validation data: [3780.40394288]
RSS on test data: [340647.03035574]
Loss estimated from test data: [2725.17624285]
Measure on Training Data (1-R^2): [0.18478713]
Measure on Validation Data(1-R^2): [0.35701766]
Measure on Test Data(1-R^2): [0.11366029]
```

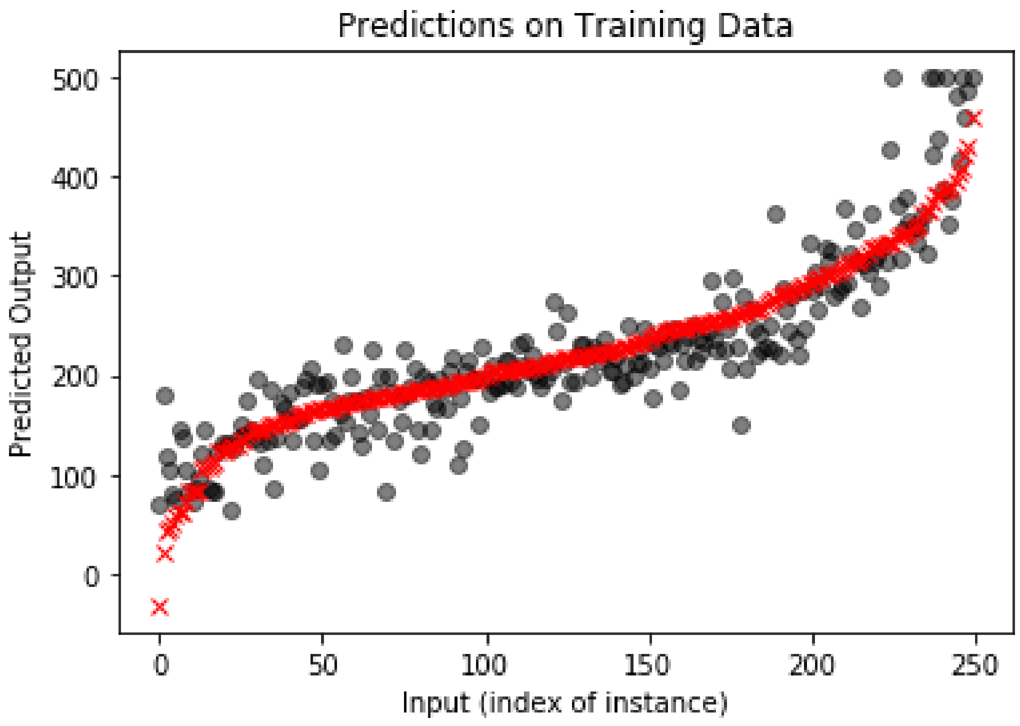
QUESTION 1: Comment on the results you get and on the difference between the train, validation and test errors.

The COD of the Test and Training set present similar values while the validation set values is smaller. Since a value of R^2 equal to 1 represent a perfect fit of the model to the data the value of (1-R^2) obtained can be considered good in particular observing that the number of data fitted is not very big while we have many parameters in the model. The error on the validation set shows that the model paramenters estimation can probably be improved especially in case of bigger data set but the model is still valid.

5.2 ... and plot:

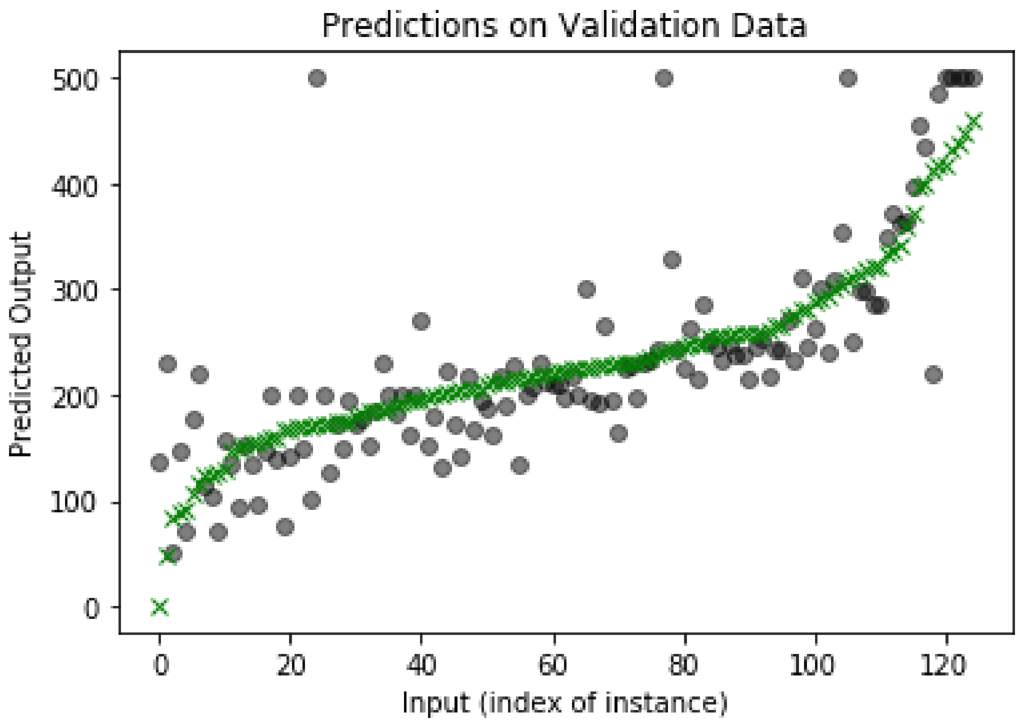
5.2.1 (1) output predictions on training data

```
In [11]: # Plot predictions on Training data
         plt.figure()

         #the following is just for nice plotting, not required: it sorts the predictions by v
         # a line and it's easier to spot the differences
         # NOTICE: This code is suitable for the "standard" solution, check that the variable
         # and structure are compatible with your implementation

         sorting_permutation = sorted(range(len(prediction_training[0:m_training])), key=lambda
         plt.plot(Y_training[sorting_permutation], 'ko', alpha=0.5)
         plt.plot(prediction_training[sorting_permutation], 'rx')

         plt.xlabel('Input (index of instance)')
         plt.ylabel('Predicted Output')
         plt.title('Predictions on Training Data')
         plt.show()
```



5.2.2 (2) output predictions on validation data

```
In [12]: # Plot predictions on validation data
         plt.figure()

         #the following is just for nice plotting, not required: it sorts the predictions by v
         # a line and it's easier to spot the differences

         # NOTICE: This code is suitable for the "standard" solution, check that the variable
         # and structure are compatible with your implementation

         sorting_permutation = sorted(range(len(prediction_validation[0:m_validation])), key=la
         plt.plot(Y_validation[sorting_permutation], 'ko', alpha=0.5)
         plt.plot(prediction_validation[sorting_permutation], 'gx')


         plt.xlabel('Input (index of instance)')
         plt.ylabel('Predicted Output')
         plt.title('Predictions on Validation Data')
         plt.show()
```



5.3 Ordinary Least-Squares using scikit-learn

A fast way to compute the LS estimate is through sklearn.linear_model

```
In [13]: from sklearn import linear_model
         LinReg = linear_model.LinearRegression()  # build the object LinearRegression
         LinReg.fit(X_training, Y_training)  # estimate the LS coefficients
         print("Intercept:", LinReg.intercept_)
         print("Least-Squares Coefficients:", LinReg.coef_)
         prediction_training = LinReg.predict(X_training)  # predict output values on training
         prediction_validation=LinReg.predict(X_validation)
         prediction_test = LinReg.predict(X_test)  # predict output values on test set
         print("Measure on training data:", 1-LinReg.score(X_training, Y_training))
         print("Measure on validation data:", 1-LinReg.score(X_validation, Y_validation))
         print("Measure on test data:", 1-LinReg.score(X_test, Y_test))
```

```
Intercept: [224.616]
Least-Squares Coefficients: [[  0.          -8.41293851   3.42558884  -0.75500659   8.32122553
  -19.12389133  40.6988505   -8.17088455 -30.74966399  16.50231282
  -18.43895046 -17.98650666   6.02897429 -22.85337374]]
Measure on training data: 0.18478713437035132
Measure on validation data: 0.3646970207432704
Measure on test data: 0.36517011408670386
```

6 2) Polynomial Regression

In this exercise you need to use polynomial regression to estimate the height reached by a ball thrown into air. The motion of the ball is controlled by the motion equation with uniform acceleration (in our case given by the gravity) that is a quadratic model. You need to estimate the initial height of the ball (h), the initial speed at which it was launched (v) and the gravity acceleration (g). The equation of the motion is : $y = h + vt + \frac{1}{2}gt^2$. In the motion.csv file you can find the measured height values (subject to noise) and the corresponding time instants.

```
In [14]: #import the required packages
         import matplotlib.pyplot as plt
         import csv
         from scipy import stats
         import numpy as np
         import sklearn as sl
         from sklearn import linear_model
```

```
In [15]: # load the data (time and height values) from the motion.csv file
         with open('data/motion.csv', 'r') as f:
             motion = csv.reader(f, delimiter=';')

             header = next(motion) #skip first line
             print(header)

             # get all the rows as a list
             data = list(motion)
             # transform data into numpy array
             data = np.array(data).astype(float)

         x = data[:,0].reshape(-1,1)
         y = data[:,1].reshape(-1,1)

         print(x.shape)
         print(y.shape)
```

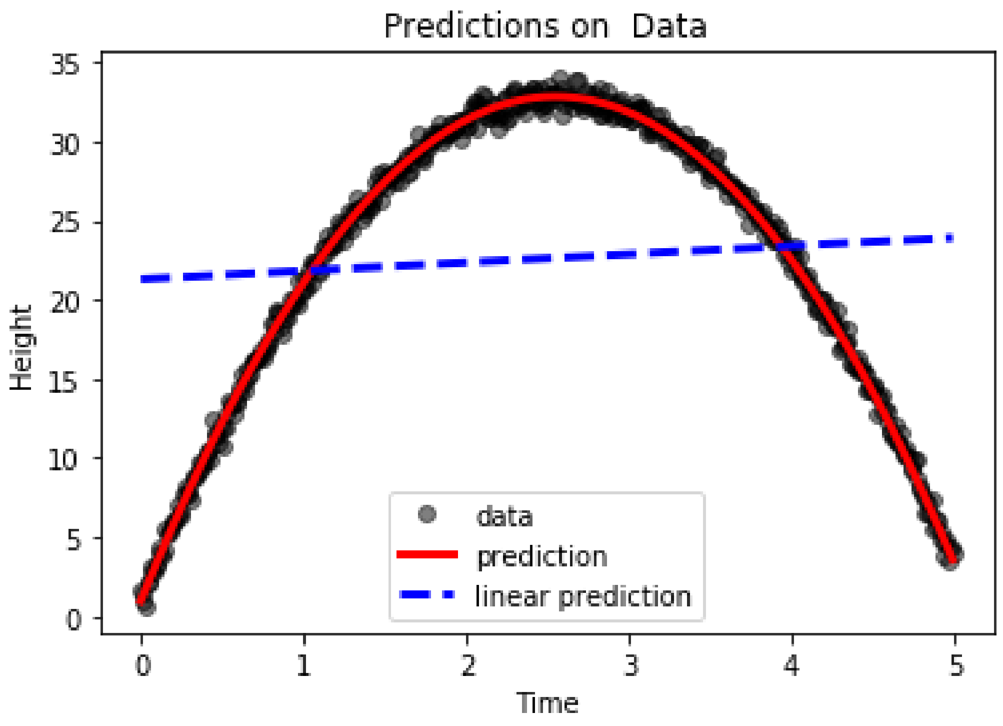
```
['time', 'height']
(500, 1)
(500, 1)
```

```
In [16]: # try to perform a linear regression (it does not work properly, the model is quadrat

         slope, intercept, r_value, p_value, std_err = stats.linregress(x.reshape(500),y.resha

         print('slope (linregress): ', slope,'  intercept (lnregress):', intercept);
         print('correlation coefficient:', r_value)

slope (linregress):  0.5227301908080864   intercept (lnregress): 21.304553347021823
correlation coefficient: 0.08206857514729192


In [17]: # use polynomial regression (the feature vectors have three components:
         # they contain all 1s (for bias), the input data $x$ and their squared values $x^2$

         dataX = np.zeros([500,3])
         dataX[:,0] = np.ones([500])  # dataX[:,0]: bias
         dataX[:,1] = x[:,0]    # dataX[:,1]: 1st order terms
         dataX[:,2] =x[:,0]*x[:,0]    # dataX[:,2]: 2nd order terms

         reg =  linear_model.LinearRegression()  # build the object LinearRegression
         reg.fit(dataX, y)  # estimate the LS coefficients
         print("Intercept:", reg.intercept_)
         print("Least-Squares Coefficients:", reg.coef_)
         prediction = reg.predict(dataX)  # predict output values on training set

         print("Measure on training data:", 1-reg.score(dataX, y))

         h = reg.intercept_ [0]
         v = reg.coef_ [0,1]
         g = reg.coef_ [0,2] *2
         sc = reg.score(dataX, y) #COMPUTE # reg.score contains the square of the correlation


         print('initial position: ', h,'  initial speed:', v, ' gravity acceleration:', g )
         print('correlation coefficient:', np.sqrt(sc))

Intercept: [0.96901643]
Least-Squares Coefficients: [[ 0.          25.02337708 -4.90994928]]
Measure on training data: 0.0029016942637539733
initial position:  0.9690164318338041   initial speed: 25.023377076576764  gravity acceleratio
correlation coefficient: 0.9985480988596623
```

Question 2 Explain what do you conclude looking at the linear and polynomial fitting ?

The difference between the linear and polynomial fit is very clear, in the first case (linear fit) the value of the correlation coefficient is close to 0 meaning that the model does not describe the data set while on the opposite the value next to 1 of the correlation coefficient in the polynomial fit is a good indication of the goodness of the model used to describe the data set. therefore we can say that the second degree polynomial is the best model to describe the data set, as expected.

```
In [18]: # plot the input data and the estimated models

         # Plot predictions on validation data
         plt.figure()

         #the following is just for nice plotting, not required: it sorts the predictions by v
         # a line and it's easier to spot the differences

         # NOTICE: This code is suitable for the "standard" solution, check that the variable
         # and structure are compatible with your implementation

         #sorting_permutation = sorted(range(len(prediction[0:len(y)])), key=lambda k: predict
         plt.plot(x,y, 'ko', alpha=0.5, label="data")
         plt.plot(x, h+x*v+g*x*x/2, 'r', linewidth=3, label="prediction")
         plt.plot(x, intercept+slope*x, '--b', linewidth=3, label="linear prediction")


         plt.xlabel('Time')
         plt.ylabel('Height')
         plt.title('Predictions on  Data')
         plt.legend()
         plt.show()
```



```
In [ ]:
```

```
In [ ]:
```