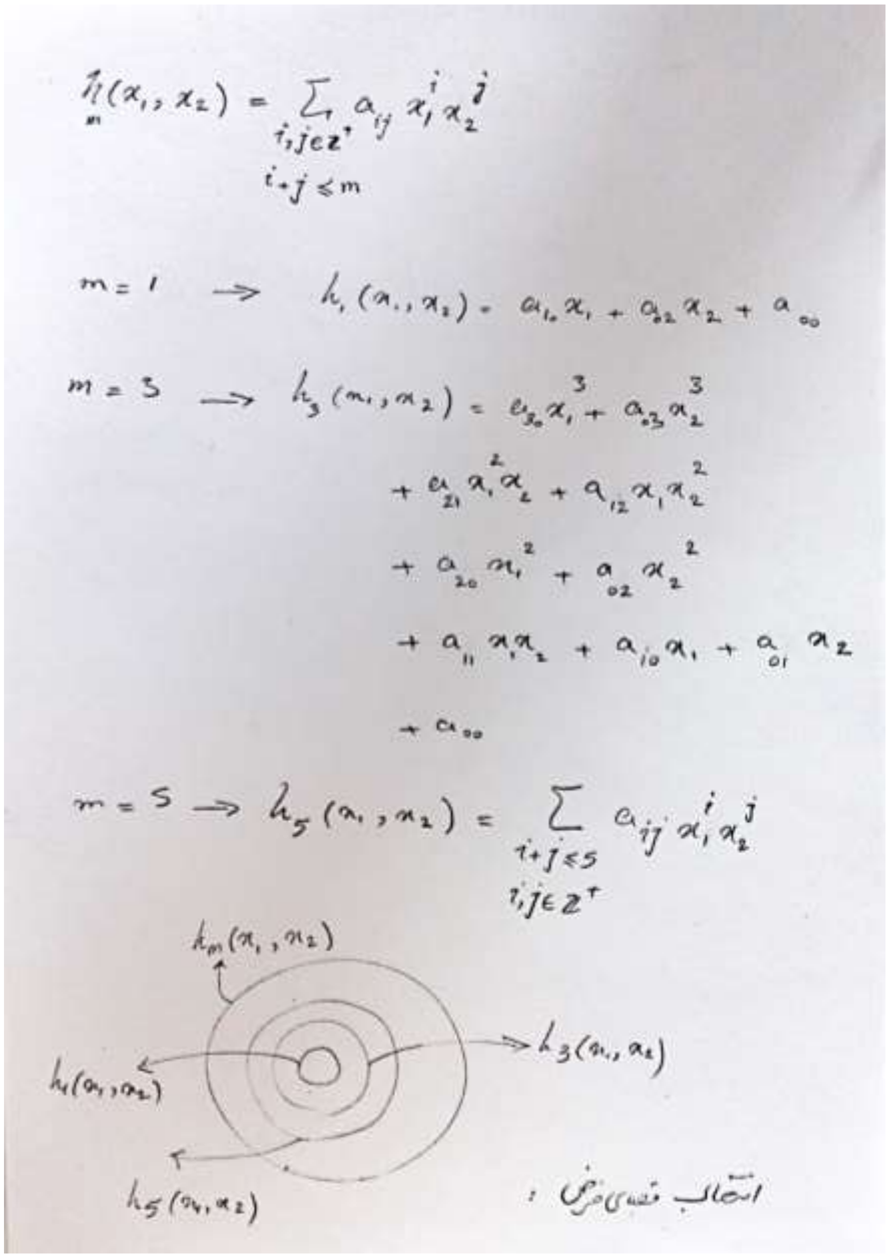
Machine learning

Assignment report: Regression

Attached file: HW1.ipynb

Before entering the fitting section, I need to explain about the poly_features function and how to generate the data matrix. We are going to generate a polynomial regression model. That is, the set of our hypothesis functions is the space of two-variable functions of h, which in each linear part, degrees 3 and 5 are limited as follows.



$$h_m(x_1, x_2) = \sum_{\substack{i,j \in \mathbb{Z}^+ \\ i+j \leq m}} a_{ij} x_1^i x_2^j$$

$$m = 1 \longrightarrow h_1(x_1, x_2) = a_{10} x_1 + a_{02} x_2 + a_{00}$$

$$m = 3 \longrightarrow h_3(x_1, x_2) = a_{30} x_1^3 + a_{03} x_2^3 + a_{21} x_1^2 x_2 + a_{12} x_1 x_2^2 + a_{20} x_1^2 + a_{02} x_2^2 + a_{11} x_1 x_2 + a_{10} x_1 + a_{01} x_2 + a_{00}$$

$$m = 5 \longrightarrow h_5(x_1, x_2) = \sum_{\substack{i+j \leq 5 \\ i,j \in \mathbb{Z}^+}} a_{ij} x_1^i x_2^j$$

$h_m(x_1, x_2)$
$h_3(x_1, x_2)$
$h_1(x_1, x_2)$
$h_5(x_1, x_2)$

انتخاب فضای فرض :

The final model is the values of the unknown coefficients. There are two columns x1 and x2 in the data set, but in addition to these two values, **<u>the model needs the products of the maximum degree of the hypothesis space of these two variables.</u>** In poly_features, for the degree of the space of hypothesis functions, this characteristic of multiplications on two input vectors x1 and x2 is calculated and provided as a matrix x ready to be used in the polynomial regression equation. In all sections, the same matrix is used both for fitting the model on the training data and for estimating the test and training data.

XW = y

Let a_{ij} s be the elements of vector W'.

$$h_1(x_1, x_2) \rightsquigarrow \overset{X\_mat}{\begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix}} \begin{bmatrix} a_{00} \\ a_{10} \\ a_{01} \end{bmatrix} = y$$

Note that x_1 and x_2 are vectors. Here for instance, in the training set we have a vector of size 8000x1.

$$h_3(x_1, x_2) \rightsquigarrow \overset{X\_mat}{\begin{bmatrix} 1 & x_2^3 & x_2^2 & x_2^1 & x_1 & x_1x_2 & x_1x_2^2 & x_1^2 & x_1^2x_2 & x_1^3 \end{bmatrix}}$$

This function is made by poly_features and a_{ij} s are picked from the optimal model during the training process. The same goes for h_5.

$$[X_1, X_2] \rightsquigarrow \boxed{\text{poly-features}} \rightsquigarrow X\_mat$$

To obtain the error measure that the SSE function is selected, it is necessary to predict the y values for the training and test data for each category separately with the obtained model. Finally, using these values and the target values given in the data, it can be calculated in the sum_square_error function and the same error is reported in all sections.

$$SSE = \sum_{i=1}^{n} (y - \hat{y})^2$$

a) Obtaining the regression equation from the closed formula

The regression equation of a linear device is Xw = y. In each linear case, the 3rd degree and 5th degree equations and unknowns are as follows.

$$Xw = Y$$

$$\boxed{i = 1 \dots n}$$

$$m = 1 \rightsquigarrow w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} = y^{(i)}$$

$$m = 3 \rightsquigarrow w_0 + w_1 x_2^{(i)^3} + w_2 x_2^{(i)^2} + \dots + w_9 x_1^{(i)^3} = y^{(i)}$$

$$m = 5 \rightsquigarrow w_0 + w_1 x_2^{(i)^5} + \dots + w_{21} x_1^{(i)^5} = y^{(i)}$$

W is obtained by the system of n equations and m variables.

To solve these system of equations, in the general case that the matrix of coefficients is not square, the unknown value can be calculated using the pseudo-inverse formula. In using the closed formula method, I use the same formula. The multivar_poly_regression function solves this.

$$J(w) = \| Y - Xw \|_2^2$$

$$\nabla J(w) = -2X^T (Y - Xw)$$

$$= 0 \Rightarrow X^T Y = X^T X w$$

$$\times (X^T X)^{-1}$$

$$\rightsquigarrow \underbrace{(X^T X)^{-1} X^T}_{X^+} Y = W$$

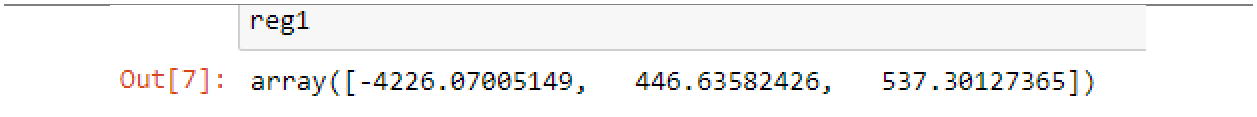
$$\boxed{w = X^+ w}$$

The following values are obtained for each of the models.

	SSEtrain	**SSEtest**
Linear regression	18317759690.368008	**12944759955.772043**
3rd degree polynomial regression	1.6877332437997384e-18	**7.636451083577081e-19**
5th degree polynomial regression	7.823644442756832e-17	**6.206512135546437e-17**

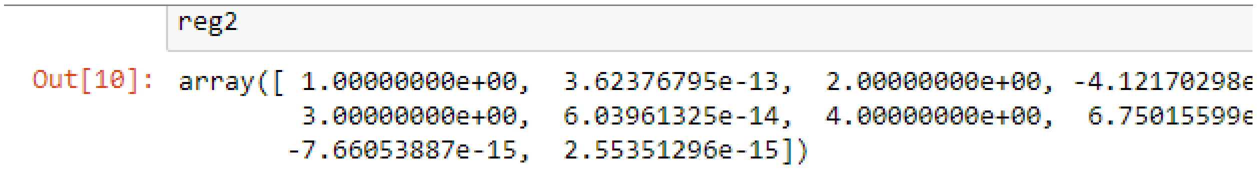
The linear model that cannot be used in any way and its high error value indicates that these data do not have a linear relationship and the complexity of the assumption space is very low. On polynomials of degree 3 and 5, the error values are both acceptable, of course, the value is better in degree 5. From the closeness of the test and training values of both of these, it can be understood that the model has a suitable complexity of at least level 5, and with this amount of data, we do not have more fitting problems.

Coefficients obtained in each method:



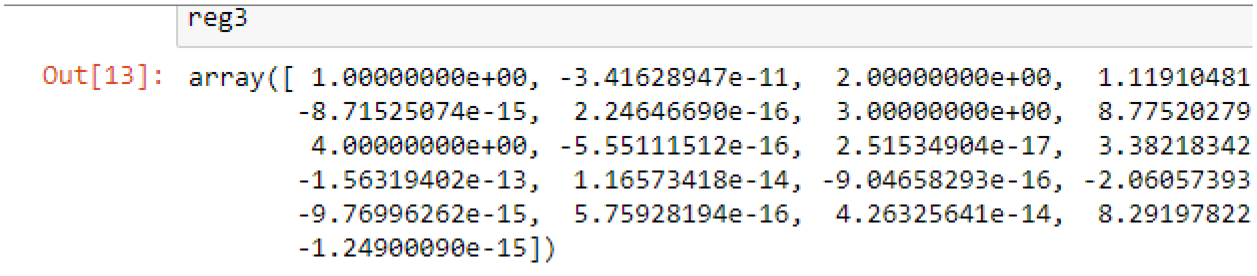
```
         reg1

Out[7]:  array([-4226.07005149,   446.63582426,   537.30127365])
```

Linear regression



```
          reg2

Out[10]:  array([ 1.00000000e+00,  3.62376795e-13,  2.00000000e+00, -4.12170298e
                  3.00000000e+00,  6.03961325e-14,  4.00000000e+00,  6.75015599e
                 -7.66053887e-15,  2.55351296e-15])
```

3rd degree polynomial regression



```
          reg3

Out[13]:  array([ 1.00000000e+00, -3.41628947e-11,  2.00000000e+00,  1.11910481
                 -8.71525074e-15,  2.24646690e-16,  3.00000000e+00,  8.77520279
                  4.00000000e+00, -5.55111512e-16,  2.51534904e-17,  3.38218342
                 -1.56319402e-13,  1.16573418e-14, -9.04658293e-16, -2.06057393
                 -9.76996262e-15,  5.75928194e-16,  4.26325641e-14,  8.29197822
                 -1.24900090e-15])
```

5th degree polynomial regression

b) Obtaining the regression equation from the decreasing gradient method

The decreasing gradient method is an iterative method, in these methods, instead of solving the regression equation, an iterative method is used to reproduce a vector of coefficients. This new model has less error on the training data each time than the previous value. It will be a good approximation of the original solution of the regression equation, with the difference that the amount of calculations required to reach this acceptable approximation is less than the amount of calculations to solve the equation. Here, I used gradient reduction to minimize the MSE error, which calculates the weights in gamma intervals in the opposite direction of the gradient value. Because I know the assumption space, I directly use the following relation for optimization.

$$w^{t+1} = w_t - \gamma \nabla J(w)$$

$$\nabla J(w) = \left(\frac{\partial J(w)}{\partial w_1}, \frac{\partial J(w)}{\partial w_2}, \dots \frac{\partial J(w)}{\partial w_j} \right)$$

J(w) is the SSE loss function. so we have:

$$J(w) = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

که $n =$ train.size , $\hat{y}^{(i)} = x^{(i)} w^t$ یعنی برآورد داده‌ی iام
به ازای پارامترهای مدل فعلی t هست.

And the n = train.size and

$$\hat{y}^{(i)} = x^{(i)} w^t$$

Which means the estimation of i-th data using current model parameters is t.

$$J(w) = \| Y - XW \|_2^2$$

$$\nabla J(w) = -2 X^T (Y - XW)$$

$$\rightarrow \quad w^{t+1} = w_t + 2 X^T (Y - XW)$$

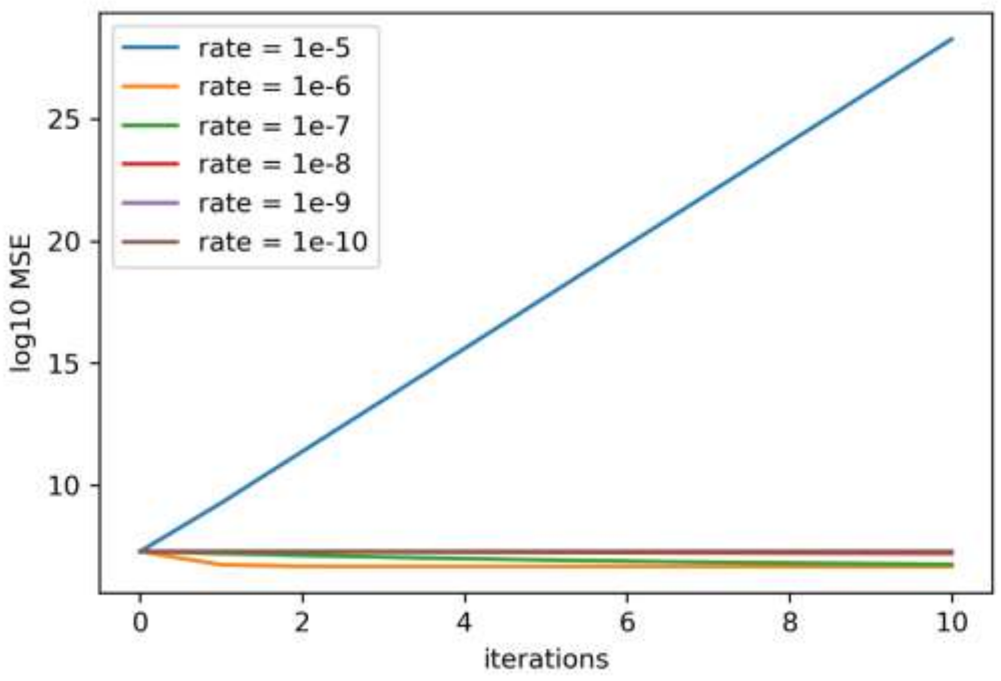
To make the error on each data more understandable, I used the MSE calculated from the following formula for the graphs. The calculation method is to use the entire training data every time, that is, batch mode calculation, so here MSE is always a fixed coefficient of the same SSE and there is no difference in the work.

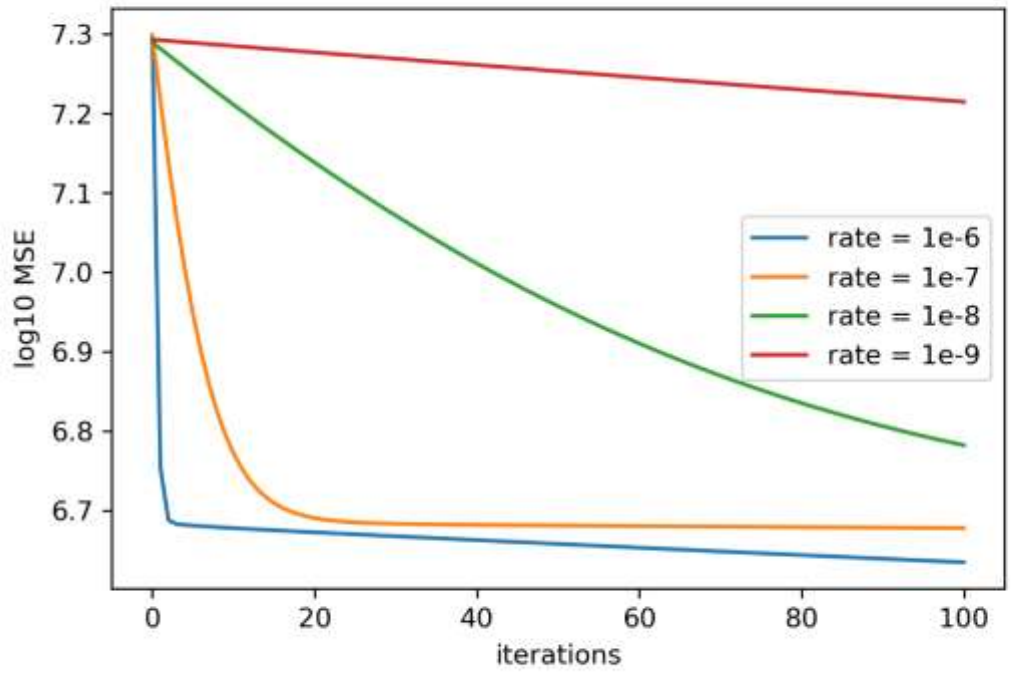
$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y-\hat{y})^2 = \frac{1}{n} SSE$$

The data matrix used in this method is the same matrix that was produced and used in the first method with the poly_features function.

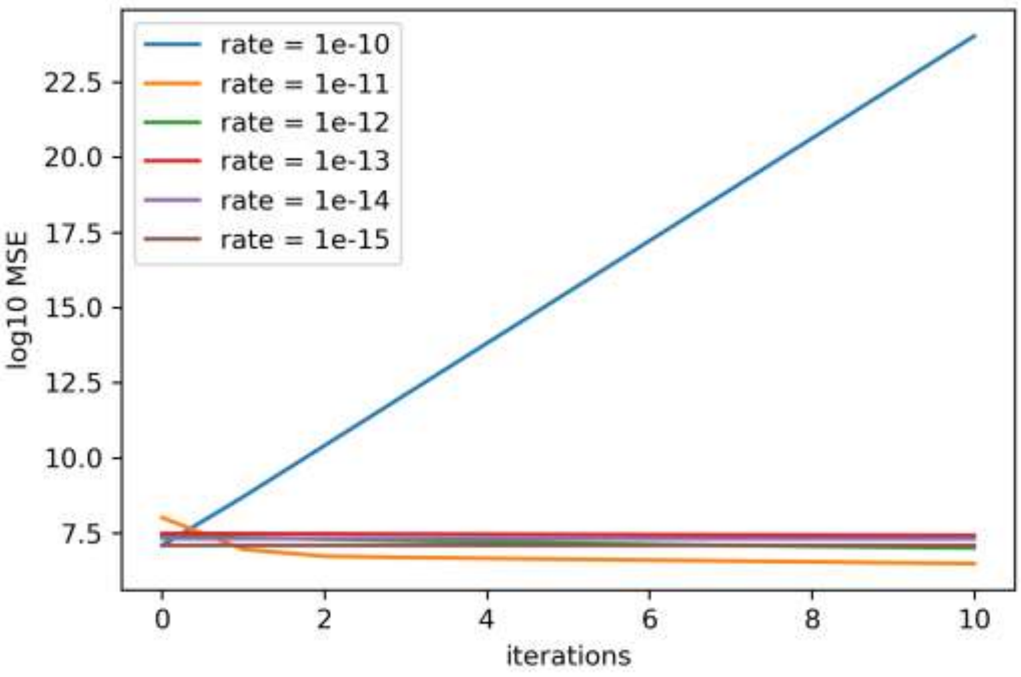
All described sections are implemented in the grad_descent function. This function returns 3 parameters, among which only weight is the model parameter. I used the other two only to measure the appropriate gamma and select it in the final model.

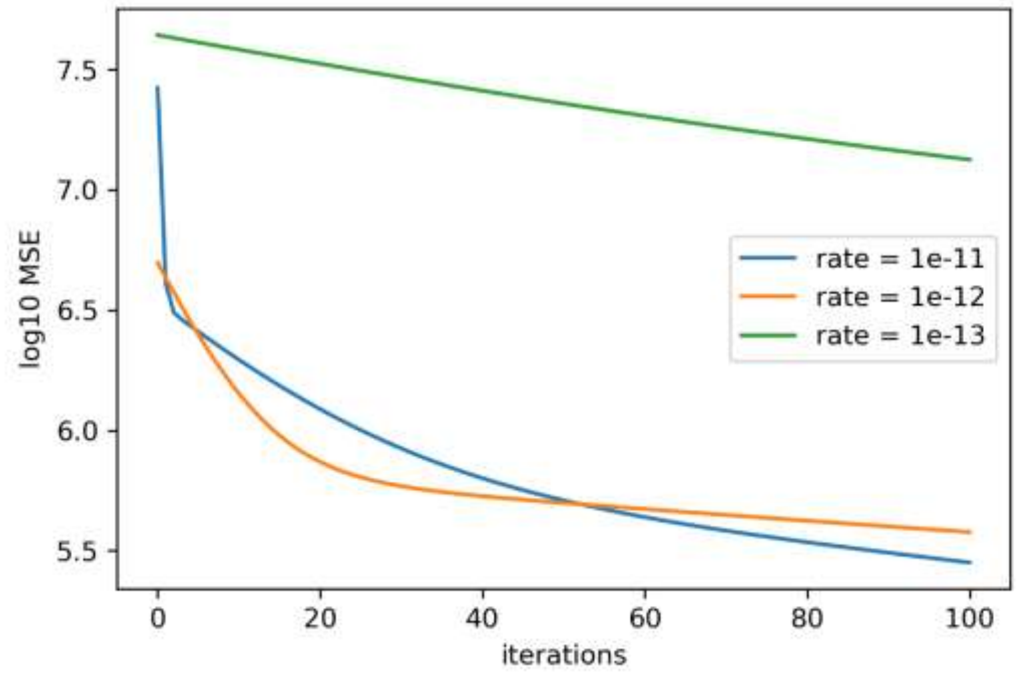
In order to choose a suitable value for gamma, before producing each of the three models, I first selected several models with different gamma sizes and according to the MSE reduction curve of each of the gammas (drawn in logarithm 10), I chose a value for gamma.



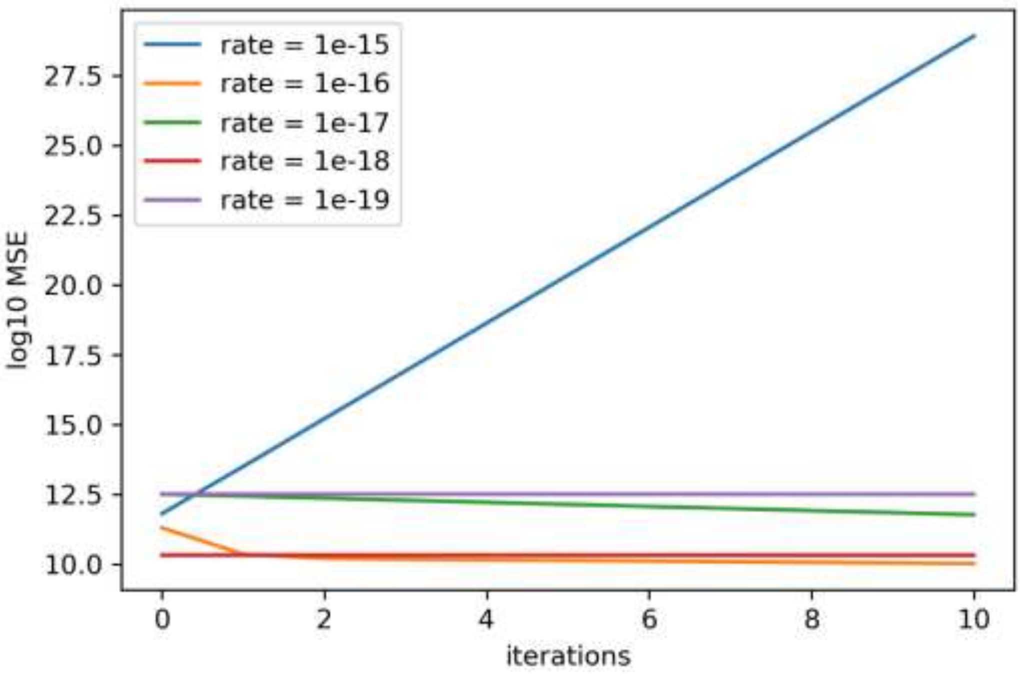


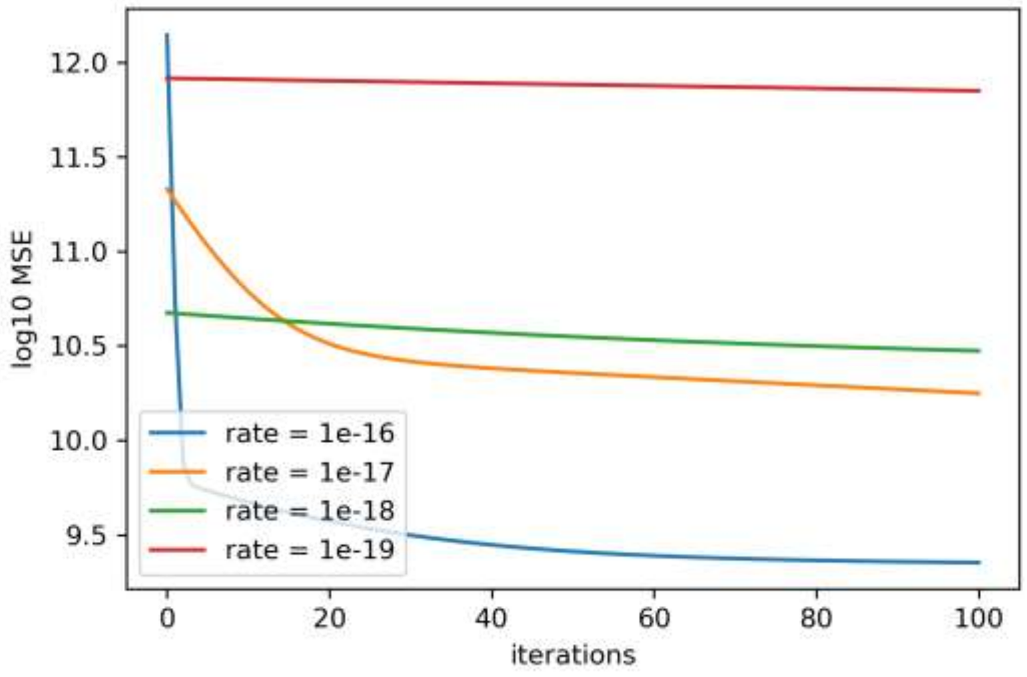
Linear regression





3rd degree polynomial regression





5th degree polynomial regression

Pay attention to the graphs that the model diverges for larger gamma values. Finally, I chose gamma values equal to 1e+7, 1e+11, and 1e+16 for the three models, respectively.

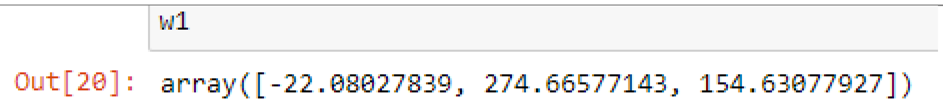
The condition for stopping training in all cases is the same and equal to 100 times of model reproduction cycle. This value can be considered different or the number of repetitions can be made dependent on the obtained error, but here, because I wanted to produce and compare all 3 models in the same way, and on the other hand, with different tests of the gamma value, the limits of the coefficients and the obtained error They were converging to the head, it seemed to me that this number of repetitions was enough for the stopping condition.

The SSE error values of training and testing data for all three models are given below.

	SSEtrainGD	**SSEtestGD**
Linear regression	38097270716.17465	**25775186283.36409**
3rd degree polynomial regression	2872831801.8607516	**1882719031.7481675**
5th degree polynomial regression	9628195119367.512	**12180370540265.457**

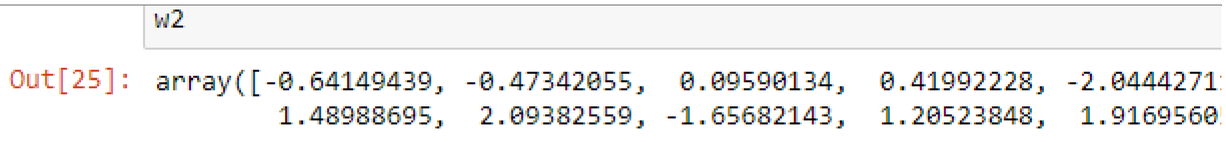
As expected after seeing the results of the first part, the linear model is unusable. Two grade 3 and 5 models work well and it doesn't matter if the method of solving the problem is computationally, analytical method or approximation and repetition, in the end, the model that is selected from the assumption space as the optimal solution, in terms of error within a certain limit Placed. (Assuming that the iterative method does not suffer from problems such as divergence or local extremes.) But here, although the 3rd and 5th degree regression converged in the iterations with the chosen gamma, the error amount is so high that the models can still be used practically. are not used The main cause of this problem is the difference in the order of the data and their relative size. Iterative methods and especially gradient descent are sensitive to these factors, and to draw conclusions in these methods, pre-processing, especially data scaling, must be used, which of course is not required here, and I will leave the results here. Maybe this problem can be compensated without pre-processing and by reducing the gamma every time in the cycle to a certain extent, which requires further investigations.

The final coefficients obtained for the models:



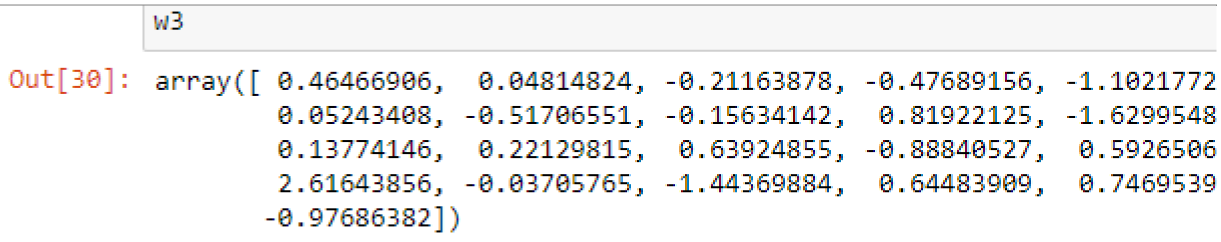
```
         w1

Out[20]: array([-22.08027839, 274.66577143, 154.63077927])
```

Linear regression



```
         w2

Out[25]: array([-0.64149439, -0.47342055,  0.09590134,  0.41992228, -2.0444271
                 1.48988695,  2.09382559, -1.65682143,  1.20523848,  1.9169560
```

3rd degree polynomial regression



```
         w3

Out[30]: array([ 0.46466906,  0.04814824, -0.21163878, -0.47689156, -1.1021772
                 0.05243408, -0.51706551, -0.15634142,  0.81922125, -1.6299548
                 0.13774146,  0.22129815,  0.63924855, -0.88840527,  0.5926506
                 2.61643856, -0.03705765, -1.44369884,  0.64483909,  0.7469539
                -0.97686382])
```

5th degree polynomial regression

c) Obtaining the regression equation using the regularizer selected by the k-fold cross-validation method.

Here I use two new functions. First, reg_multivar_poly_regression, which implements the regression formula together with a regularizing sentence *l* for the x data matrix and y target values, and finds and returns the regression coefficients from the closed formula calculation method. The data matrices used in this section are exactly the same as the previous sections, which were obtained from the poly_features function.

$$J(w) = \sum_{i=1}^{n} (y^{(i)} - x^{(i)}w) + l w^T w$$

$$J(w) = \|Y - Xw\|_2^2 + l\|w\|_2^2$$

$$\rightsquigarrow \nabla J(w) = 0$$

$$-\cancel{2} X^T(Y - Xw) + \cancel{2} l w = 0$$

$$X^TXw + lw = X^TY$$

$$(X^TX - lI)w = X^TY$$

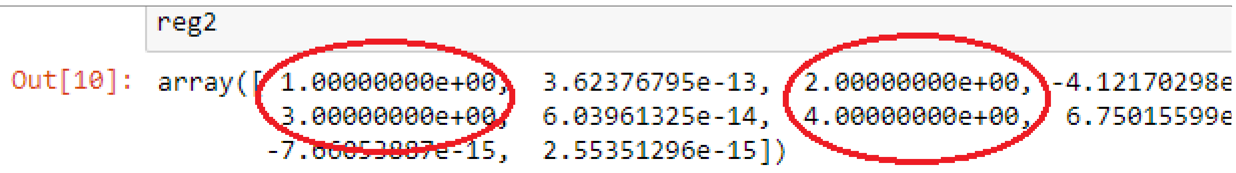
$$\boxed{w = (X^TX - lI)^{-1} X^TY}$$

To choose the appropriate *l* value, I calculate the desired model for different coefficients. In order to compare these models, as mentioned in the question, I use the k-fold cross-validation method with a value of k=5. This action is implemented in cross_val. In this method, it takes 8000 test data once. Then it divides into 5 categories. Each time, it obtains the model on all the training data except the i-th category, that is, on 6400 data with the reg_multivar_poly_regression function, and finally uses the data of the discarded category to measure the validation error. A total of 5 models are built and 5 validation errors and 5 training errors are calculated. The reported result is the average of these errors.

Finally, for the reported results of the k-fold cross-validation method, I will draw a graph for each *l* of the following set to see the impact of different *l* on the errors.

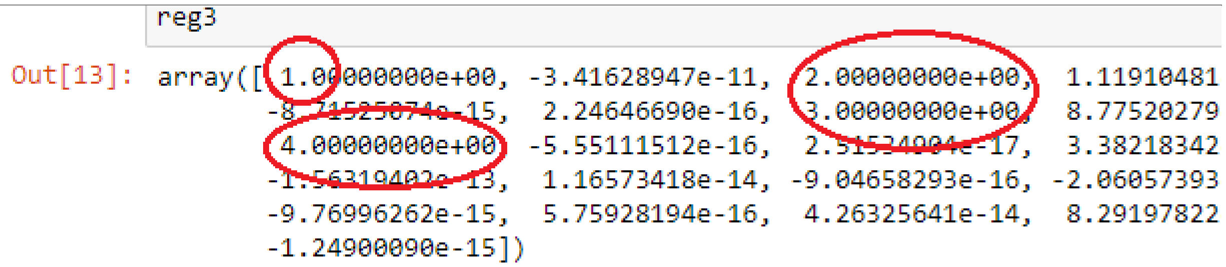
{1e-4 ,1e-3, 1e-2 ,1e-1, 1, 1e+1, 1e+2, 1e+3, 1e+4}

To select the first point from the graph, it is desired to reduce the error of the validation data and increase the training error in the most suitable mode. In fact, the main goal is to choose a coefficient that brings these two curves as close as possible. But here the regularizer is not useful. In the first model, I have the problem of the low complexity of the model, which the regularizer does not help to solve. In the two grade 3 and 5 models, first of all, the models obtained from part a had very good performance and the error rates of both testing and training were the same, so there was no overfitting problem that needed to be solved. Secondly, if we look once again at the obtained coefficients of the first part for grade 5, although this model has grade 5, most of the obtained coefficients for the model have a very small value close to zero and are the only coefficients that determine the model. For the 3rd grade model, the same sentences were obtained (I specified it on the table of coefficients). The 5th grade model might suffer from overfitting, but due to the large number of training data, this problem has been solved and the model works well. In a situation where the complexity of the model is more than needed, what the regularizer does is to reduce the coefficients of the model for sentences of higher degrees, which is not needed here due to the sufficient number of training data.



```
         reg2

Out[10]: array([ 1.00000000e+00,  3.62376795e-13,  2.00000000e+00, -4.12170298e
                 3.00000000e+00,  6.03961325e-14,  4.00000000e+00,  6.75015599e
                -7.66053887e-15,  2.55351296e-15])
```

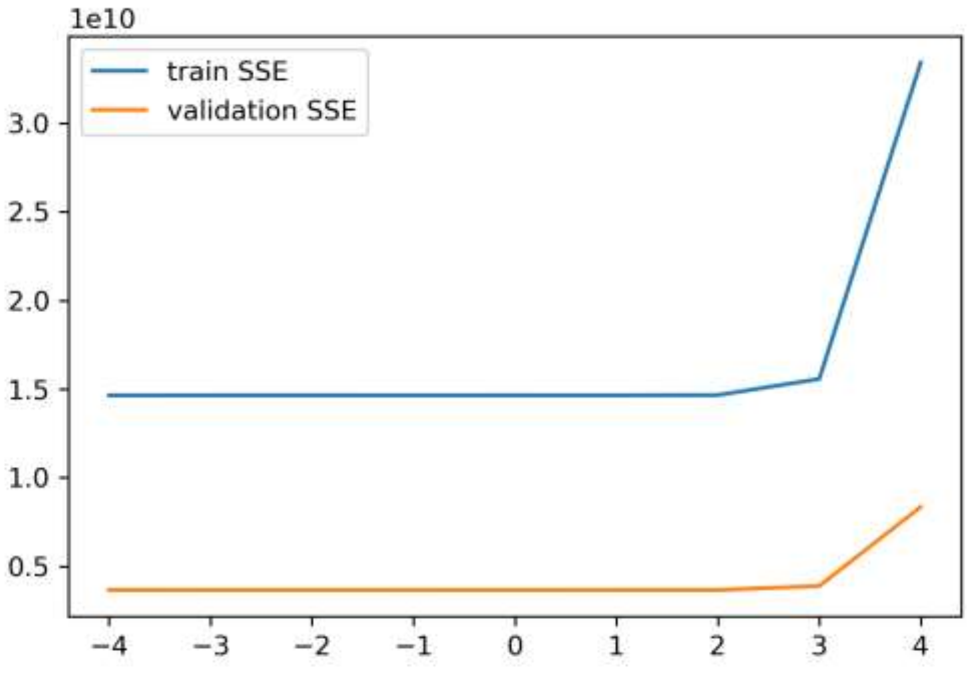
From section a) 3rd degree polynomial regression



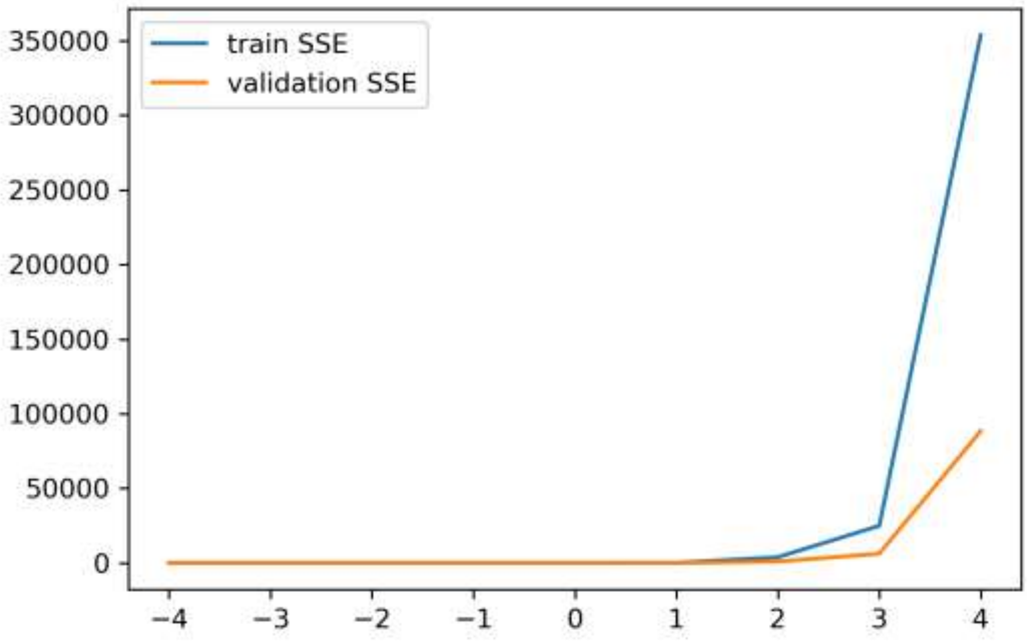
```
         reg3

Out[13]: array([ 1.00000000e+00, -3.41628947e-11,  2.00000000e+00,  1.11910481
                -8.71525074e-15,  2.24646690e-16,  3.00000000e+00,  8.77520279
                 4.00000000e+00, -5.55111512e-16,  2.51524004e-17,  3.38218342
                -1.56319402e-13,  1.16573418e-14, -9.04658293e-16, -2.06057393
                -9.76996262e-15,  5.75928194e-16,  4.26325641e-14,  8.29197822
                -1.24900090e-15])
```

From section a) 5th degree polynomial regression

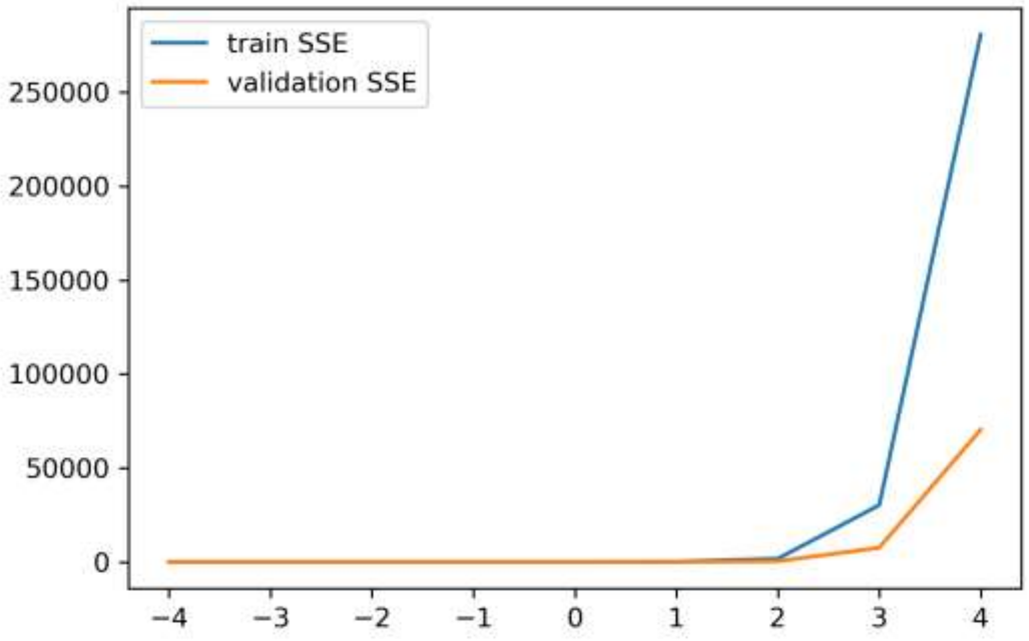
The graphs for different / are given below. In each case, I set / equal to 1e-4 to proceed.



Linear regression



3rd degree polynomial regression

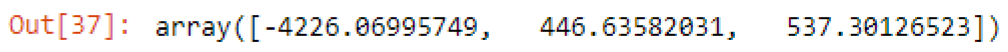


5th degree polynomial regression

In the last step, I build the final model with all the training data and the selected regularizer sentence and calculate the error of the test and training data, which for each of the three models of linear regression, polynomial regression of the 3rd and 5th degree as It is below. Also, in the table below, the values of SSE errors of the same model, which were obtained from the k-fold cross-validation method and were expected, are also reported.

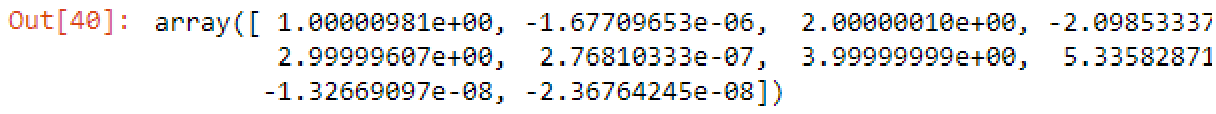
	SSEvalidation	**SSEtrainkKfold**	**SSEtrain**	**SSEtest**
Linear regression	3670043788.025161	14651324730.91531	18317759690.367973	**12944760110.83464**
3rd degree polynomial regression	3.1444955092089305e-09	1.2507476793264266e-08	9.982392722555469e-09	**9.515612484722473e-09**
5th degree polynomial regression	1.647542130942108e-06	6.495571521099052e-06	1.6958291308018977e-06	**9.57101909929323e-06**

Final models:



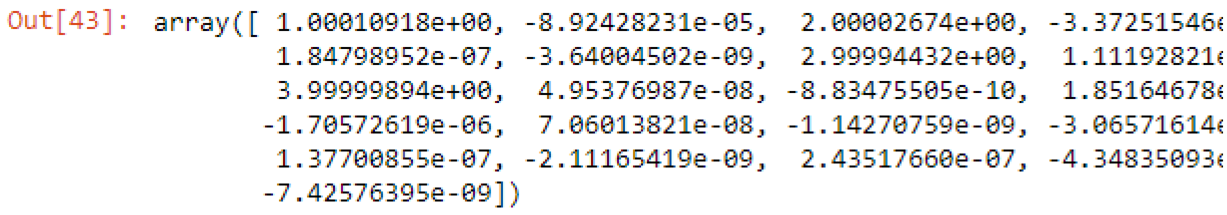
```
Out[37]: array([-4226.06995749,   446.63582031,   537.30126523])
```

Linear regression



```
Out[40]: array([ 1.00000981e+00, -1.67709653e-06,  2.00000010e+00, -2.09853337
                 2.99999607e+00,  2.76810333e-07,  3.99999999e+00,  5.33582871
                -1.32669097e-08, -2.36764245e-08])
```

3rd degree polynomial regression



```
Out[43]: array([ 1.00010918e+00, -8.92428231e-05,  2.00002674e+00, -3.37251546
                 1.84798952e-07, -3.64004502e-09,  2.99994432e+00,  1.11192821
                 3.99999894e+00,  4.95376987e-08, -8.83475505e-10,  1.85164678
                -1.70572619e-06,  7.06013821e-08, -1.14270759e-09, -3.06571614
                 1.37700855e-07, -2.11165419e-09,  2.43517660e-07, -4.34835093
                -7.42576395e-09])
```

5th degree polynomial regression