Implementation Assign1

Seifeddine Mejri, Rameez Wajid, Aashish Adhikari

S. Mejri (PART1) Rameez.Wajid (PART3+prediction file) Aashish Adhikari(Part 2)

In this work we use linear regression with L2-regularization to predict housing prices using a number of features.

The goal is to implement the Linear regression algorithm and Batch Gradient descent to optimize the linear regression with L2-regularization.

SSE objective: $\sum_{i=1}^{N} \quad (y_i - w^T x_i)^2 \ + \lambda |w|^2$

We use the training dataset to train our model.

we need to see which column is important for us and which is not. Our main aim today is to make a model which can give us a good prediction on the price of the house based on the features.

We built three separate file for each part to test on server for each part. The results that we got are reported in the this file followed with our explanation.

Part 0:

(a)

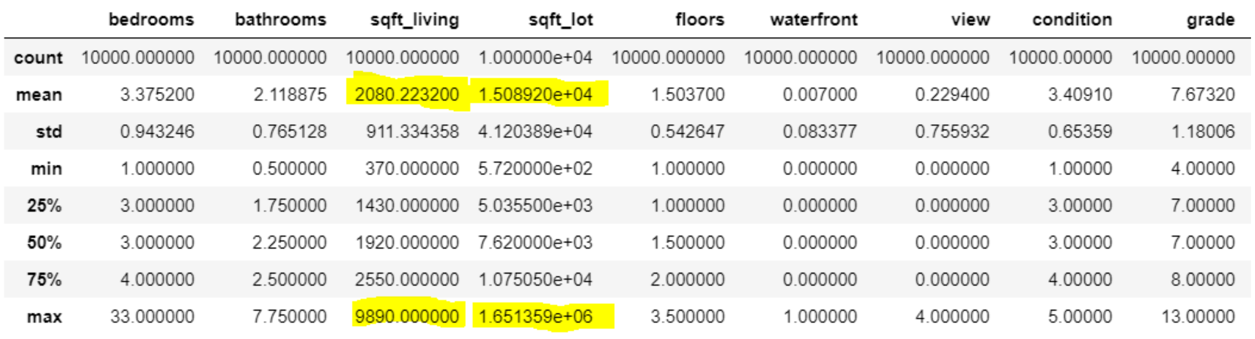
The id feature has been removed as seen in the code.

It is a bad idea to use this feature because there does not exist any correlation between a number that merely identifies a housing record and the actual price of the house. So, we cannot learn anything relevant from this id feature and thus we remove this feature.

(b)

The date feature has been split into three different features as shown in the code and following is an image of the table after the split. A better way of using the date feature would be by converting the months and the years into days and summing all three into a single feature. This should result in a feature that has a comparatively large numerical value but we can normalize to reduce it to a smaller range as (largest-mean)/(largest -smallest).The date of sale could be relevant because of certain factors. For example, the houses might have been sold in lower rates after a certain date when the economy got into recession.

(c) The tables depicting the statistics for each feature as shown below:



	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
count	10000.000000	10000.000000	10000.000000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.00000	10000.00000
mean	3.375200	2.118875	2080.223200	1.508920e+04	1.503700	0.007000	0.229400	3.40910	7.67320
std	0.943246	0.765128	911.334358	4.120389e+04	0.542647	0.083377	0.755932	0.65359	1.18006
min	1.000000	0.500000	370.000000	5.720000e+02	1.000000	0.000000	0.000000	1.00000	4.00000
25%	3.000000	1.750000	1430.000000	5.035500e+03	1.000000	0.000000	0.000000	3.00000	7.00000
50%	3.000000	2.250000	1920.000000	7.620000e+03	1.500000	0.000000	0.000000	3.00000	7.00000
75%	4.000000	2.500000	2550.000000	1.075050e+04	2.000000	0.000000	0.000000	4.00000	8.00000
max	33.000000	7.750000	9890.000000	1.651359e+06	3.500000	1.000000	4.000000	5.00000	13.00000

Table 0-1: the statistics for each feature

We notice that from the set of statistics that some features have higher values than other features. Usually feature like sqft_lot and sqrt_living take big values compared to other features like bedrooms and bathrooms. That's why we need to normalize.

(d) Based on the meaning of the features as well as the statistics, which set of features do you expect to be useful for this task? Why?

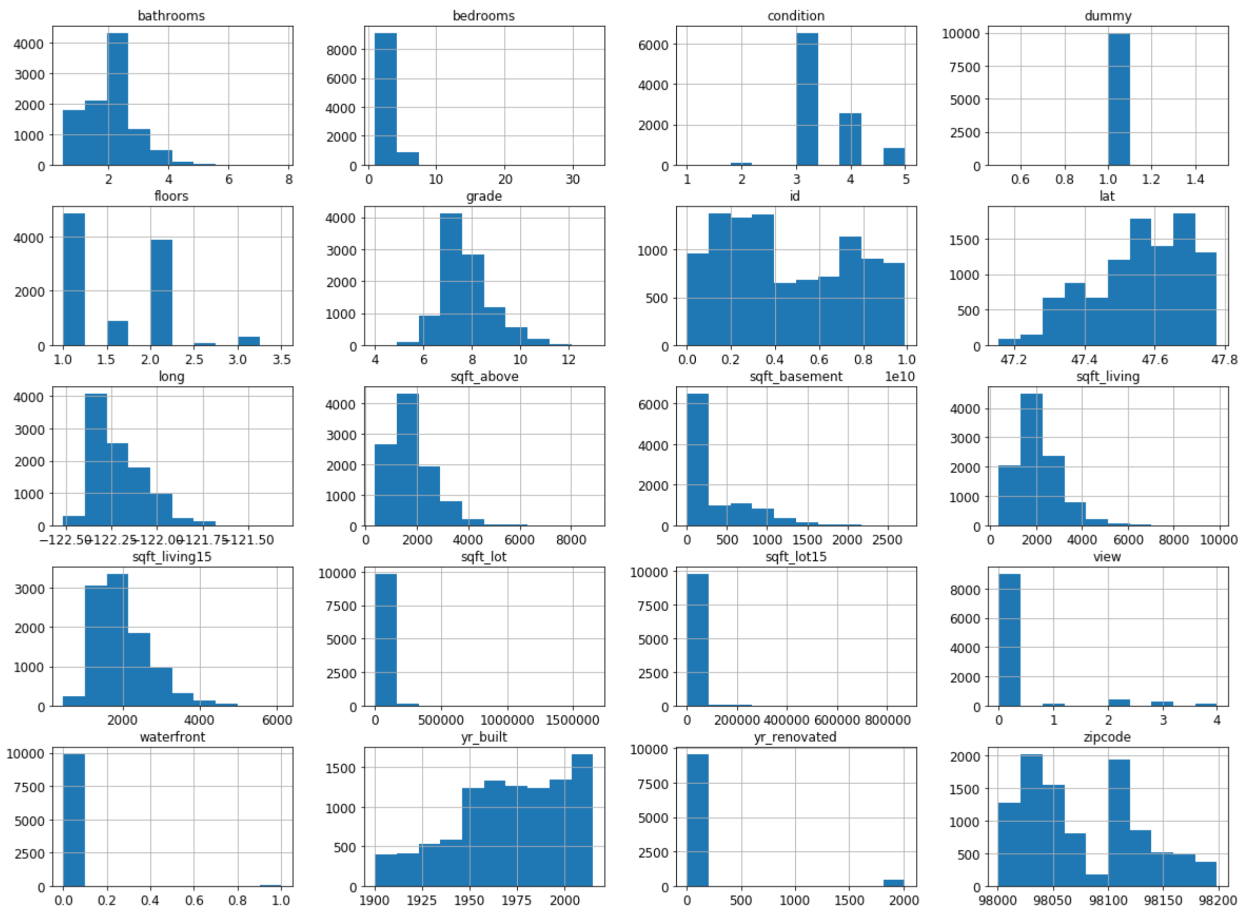


Figure 1 Plot of each feature histogram

As we can see from the histograms, the features number of bedrooms, area of the lot, area of the lot in 2015, view, waterfront, and year renovated are all concentrated at one single value. Thus, these information are all skewed and the contribution that they would make on the price of the house would be of little use. So these features could be discarded while building a model.

Also, the area of living area and the area of living area in 2015 are redundant information. As seen in the histogram above, these distributions look similar and one of them can be discarded.

Furthermore, the percentage of different values of the categorical data types can be seen below. Only waterfront and view have skewed data and can be discarded while the values of grade are more distributed.

Waterfront

0	99.3
1	0.7

View:

0	90.30
1	1.62
2	4.25
3	2.50
4	1.33

Grade:

4	0.11
5	1.05
6	9.33
7	41.30
8	28.38
9	11.82
10	5.42
11	2.10
12	0.39
13	0.05

Table: Statistics for categorical features.

Part 1:

Stop condition:

- put a limit on the step size of the gradient so that when the L2 norm of the gradient is more than 1e45, we stop our algorithm(that's a sign that our model is diverging: the gradient is exploiding)

- Epsilon< =0.5

a)

We vary the learning rate $\alpha = 10^0, 10^{-1} 10^{-2}\ 10^{-3}\ 10^{-4}\ 10^{-5}\ \ 10^{-6}\ 10^{-7}$ and plot the Sum square error(SSE) for each of the following learning rates.

For $\alpha = 10^0, 10^{-1} 10^{-2}\ 10^{-3}\ 10^{-4}\ 10^{-5}\ \ 10^{-6}\ 10^{-7}$, we plot the Sum square error function of the number of iterations.

For $\alpha = 1{,}10^{-1,}10^{-2}\ 10^{-3}\ 10^{-4}$ the gradient descent explode.

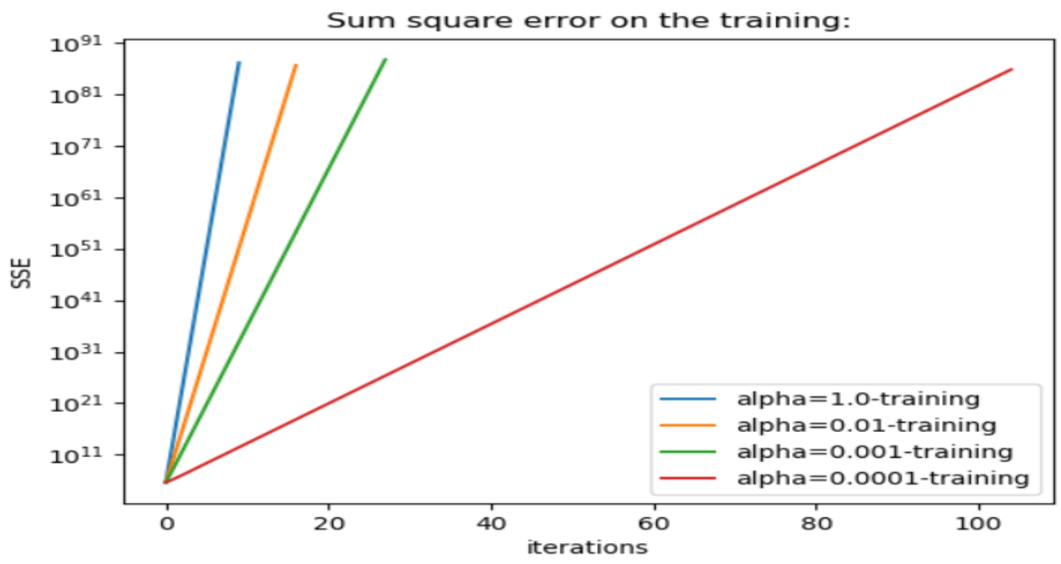


Figure 2: SSE for the training model using $1{,}10^{-1,}10^{-2}\ 10^{-3}\ 10^{-4}$ *learning rates.*

A possible explanation to this is that gradient is wiggling around the optimal back and forward which may have resulted in high sum error rate.

Setting the **learning rate** α>0 too high is a dark art. Only if it is sufficiently small will gradient descent converge. If it is too large the algorithm can easily *diverge* out of control.

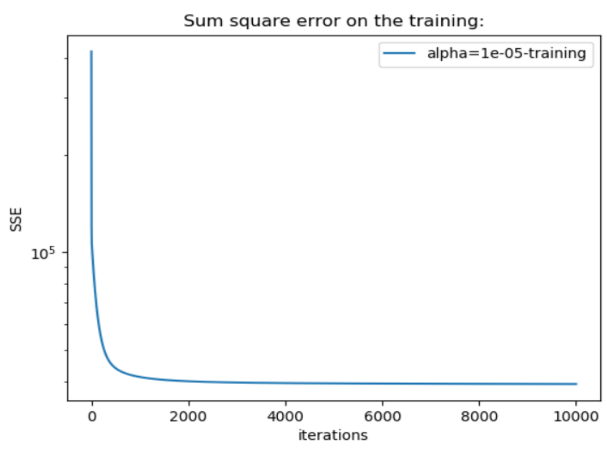


Figure 3: SSE with learning rate= 10^{-5}

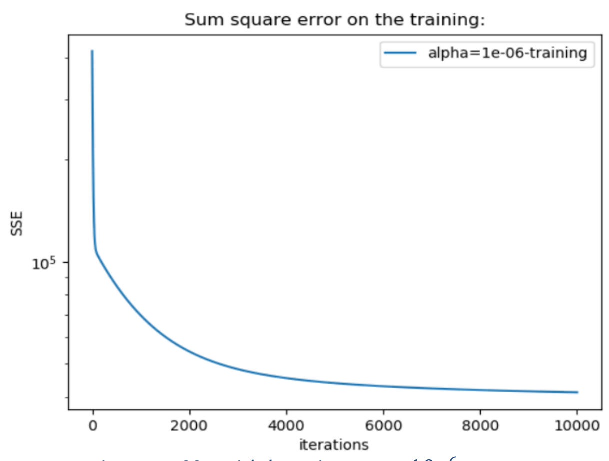


Figure 4: SSE with learning rate= 10^{-6}

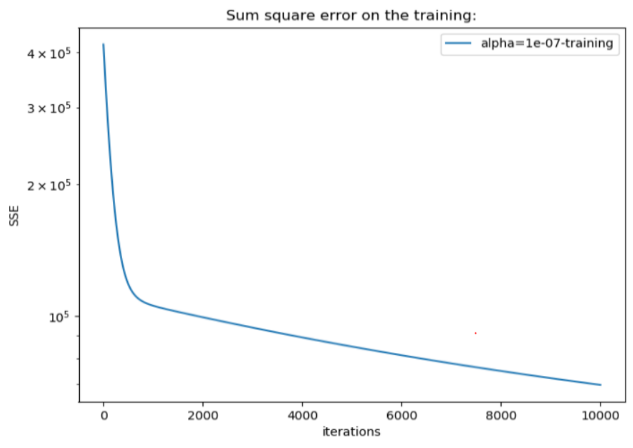


Figure 5: SSE with learning rate 10^{-7}

Figure2,3,4: SSE for training model only: SSE for the training model using $10^{-5}, 10^{-6}, 10^{-7}$ *learning rates.*

In gradient descent our goal is to find a vector weights that minimizes the cost function. In steepest descent we simply set

$$w \rightarrow w \ -\alpha \ g(w\ )$$

In the case where α is too large there is high chance that the gradiant will diverge. A safe choice is to choose α not too large and not too small.

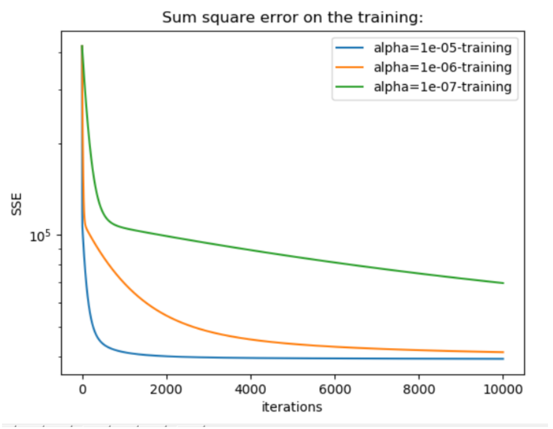


Figure 6: SSE for training model

For $\alpha = ,10^{-7}, 10^{-6}, 10^{-5}$, we have convergence of the gradient algorithm. (Epsilon=0.5)

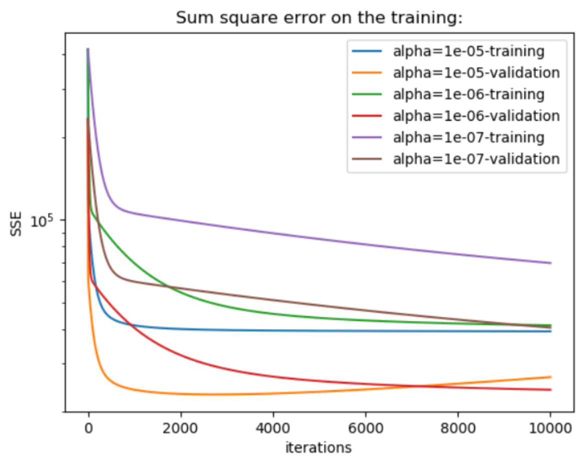
We notice that the Gradient is only converging starting from $\alpha = 10^{-5}..10^{-7}$.

As we get to lower learning rates the gradient converges but takes longer time, and that's because our algorithm will try to learn the best values of the weights but the learning rate is too small.

$Our\ gradient\ converges\ but\ the\ step\ size\ gets\ slower.$

b)

For our converged learning rates we try to plot the loss for the validation model, we notice that



When $\alpha = 10^{-5}$ it is the best learning rate for the training set but we notice that for the validation set, the SSE ($\alpha = 10^{-5}$)is increasing (orange curve).

Using the validation set with $\alpha = 10^{-5}$, we can clearly see that the linear regression is overfitting through the validation.

However when $\alpha = 10^{-6}$ the SSE curve for the validation data converges the best. (red curve)

(c)

The best converged solution is for $\alpha = 10^{-6}$ using the validation data.

The learned weight using that $\alpha = 10^{-6}$ are reported after running until we reach the condition Gradient < Epsilon= 0.5.

['dummy', 'month', 'day', 'year', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15]:

These are the weights that we got:

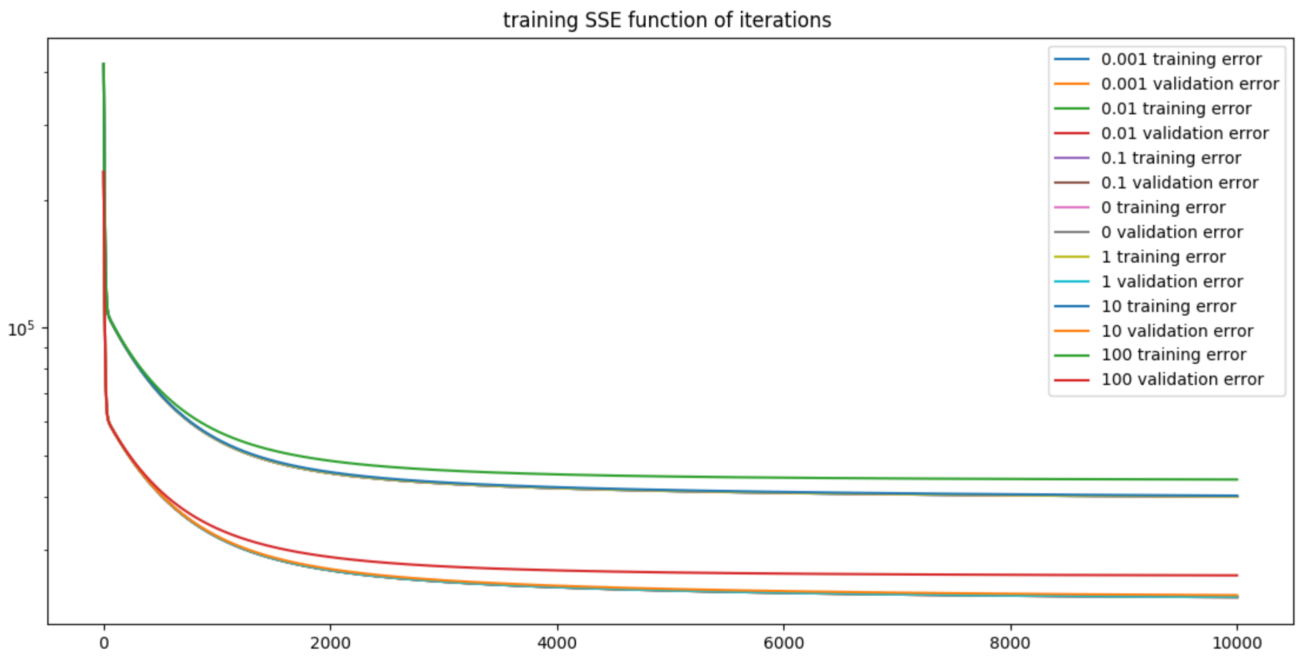
```
weights= [-2.02634728  0.18228678 -0.17278705  0.38104872 -9.18111258  3.33191845
  7.26277617  2.31366271  0.11770159  4.71107644  2.32284391  1.25636606
  8.87445487  7.72517037  1.30696057 -2.93634351  0.28851062 -1.02065174
  3.76772533 -2.67437418  1.33430007 -2.91737859]
```

'dummy', sqft_lot15 , long, zipcode, yr_renovated, are negatively correlated with the price. Month,year, floors year_built have very low weights.

The feature that are the most important are: Bedrooms, Bathrooms sqft living, sqft_lot, floors, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_renovated,lat,sqft_living.

Part 2:

First of all, let us see the curves for the SSE vs Number of Iterations for all values of lamda and the error threshold, Epsilon = 0.5. For this, we use only 1000 iterations for each lamda.

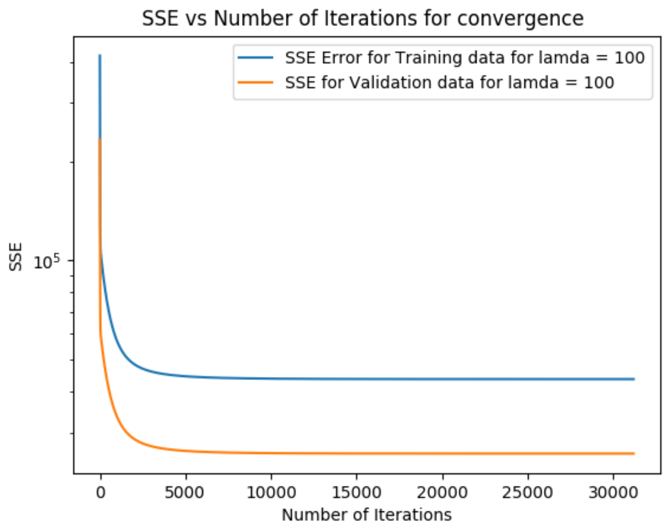


If we plot SSE vs the number of iterations for both training and validation data, the plot shows as if the SSE values converge early. However, running the algorithm until convergence gives us otherwise. We will see the plot for each individual lamda until it converges below. In the process, we will also increase the value of epsilon(the error threshold) to reach at convergence faster.

Following is a table that shows the lamdas and the corresponding epsilon values used:

lamdas	epsilon
100	0.5
10	0.5
1	1
0	0.5
0.1	10
0.01	10
0.001	10

For lamda = 100, Epsilon = 0.5

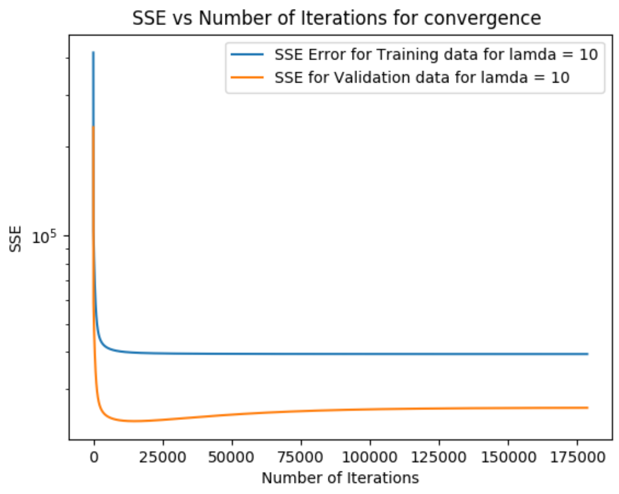


As we can see, for lamda = 100, we reach convergence with an error threshold epsilon = 0.5 in between 30000 and 40000 iterations.

The corresponding weights [w0,w1,w2...........wn] are:

```
[-1.72208259  0.08617223 -0.17944288  0.27264127  0.11227703  2.61432737
  4.01678042  0.14936035  1.16069814  2.03277081  2.85468839  1.12466049
  5.71765174  3.96079093  1.69582155 -1.8608979   0.56665252 -0.57831544
  3.37184153 -1.08263176  3.38520058  0.06923937]
```

For lamda = 10, Epsilon = 0.5



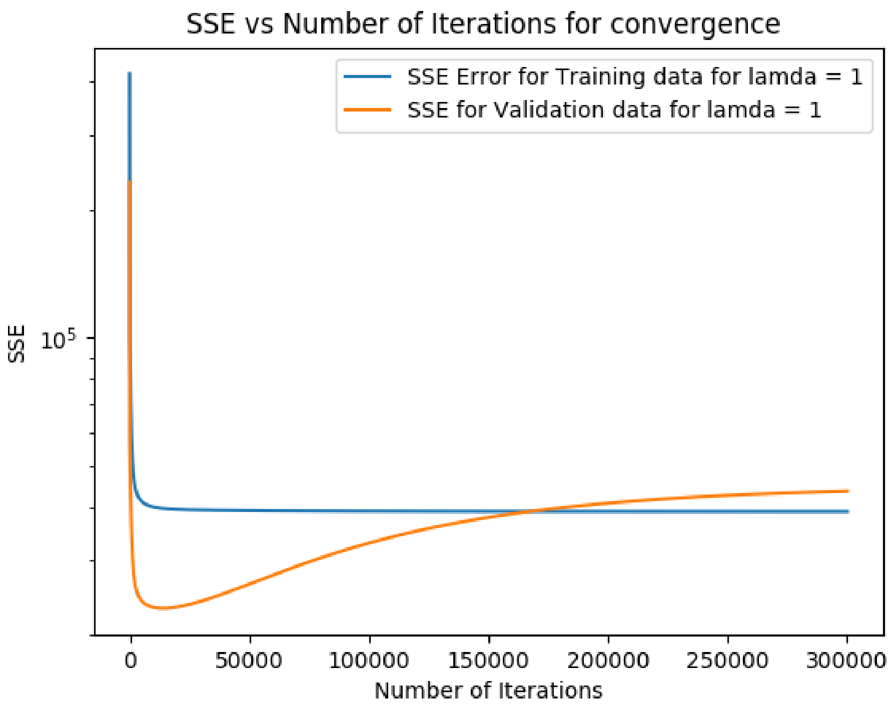
As it can be seen, we reach a convergence in between 175000 and 180000.

However, we can see the SSE values for the validation set increasing for these values of lamda and epsilon.

The weights for this convergence are:

```
 [-1.72208259  0.08617223 -0.17944288  0.27264127  0.11227703  2.61432737
  4.01678042  0.14936035  1.16069814  2.03277081  2.85468839  1.12466049
  5.71765174  3.96079093  1.69582155 -1.8608979   0.56665252 -0.57831544
  3.37184153 -1.08263176  3.38520058  0.06923937]
```

For lamda = 1, Epsilon = 1

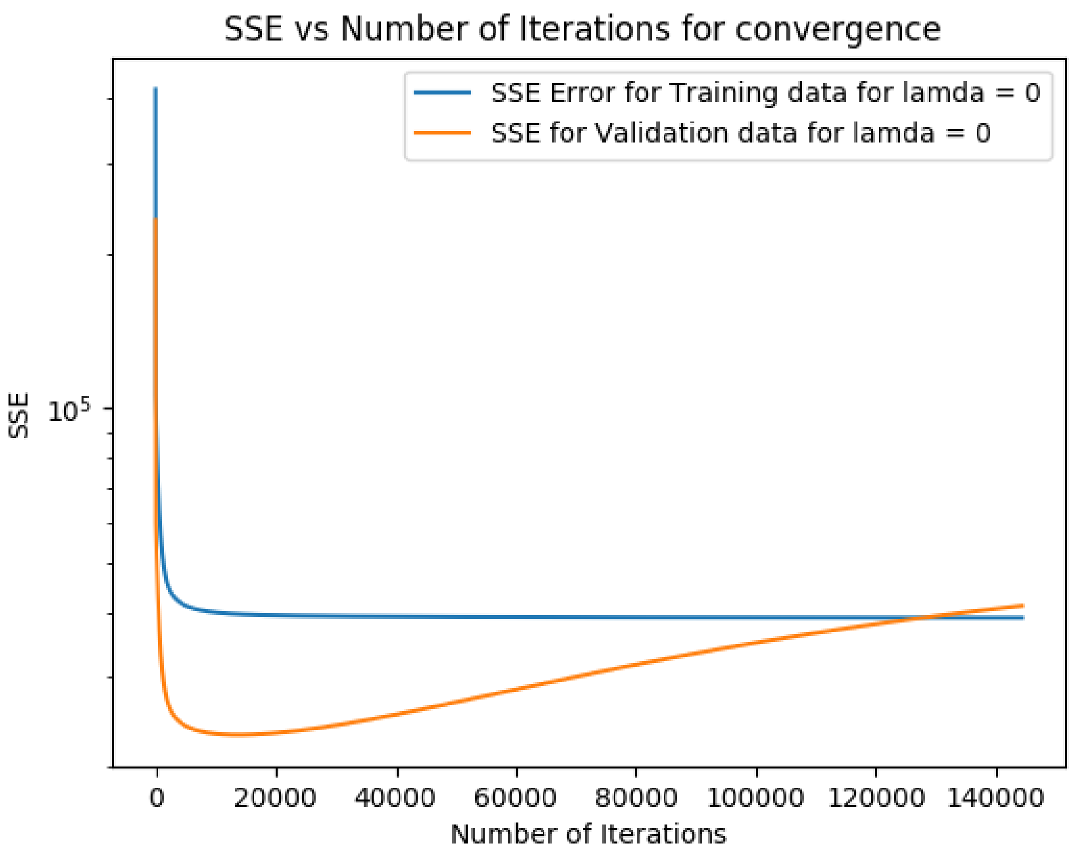


As seen above, this is the first instance where the validation SSE goes beyond training SSE while going towards convergence.
The weights are:

The weights for which we stopped or converged are [-2.11564773 0.1820145 -0.17302897
0.37948412 -7.36560784 3.24345473
  7.00220681 1.17448328 0.15858994 4.65746432 2.35760786 1.24515328
  8.89051677 7.44692667 1.26345639 -2.91472394 0.29746817 -1.00200202

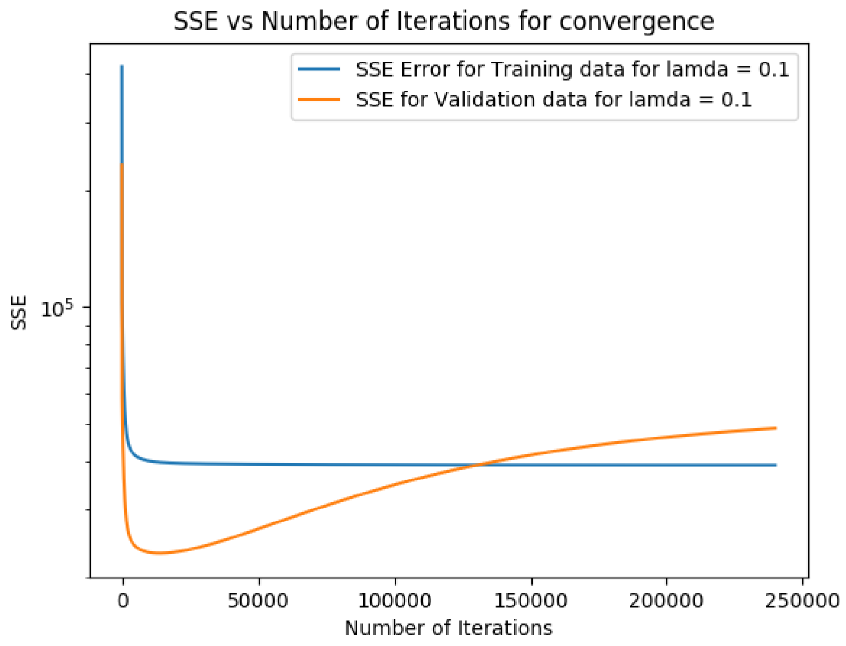
3.76476637 -2.64824489 1.45493025 -1.83522788]

For lamda = 0, Epsilon = 25



The weights for which we stopped or converged are [-2.14559955 0.1843593 -0.17251504
0.38130165 -6.96373327 3.1897843
7.03426115 0.82507188 0.13440296 4.73741577 2.34833265 1.24154944
9.00673755 7.49129083 1.2371717 -2.92501322 0.29263853 -1.0082259
3.77052188 -2.67836224 1.34840469 -1.54604492]

For lamda = 0.1, Epsilon = 10

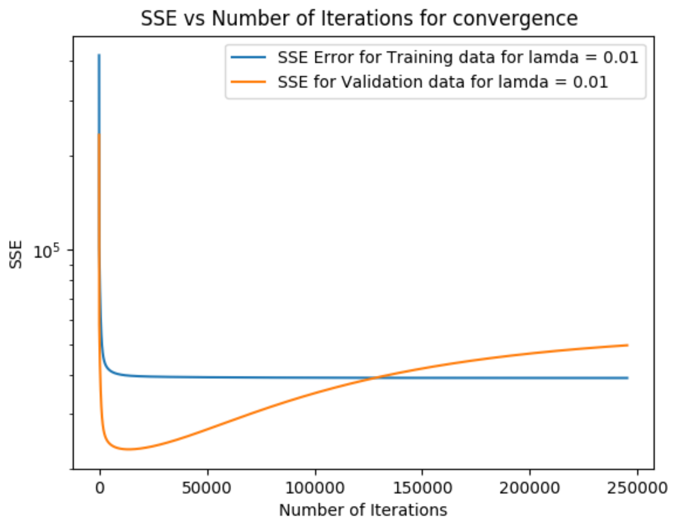


As we can see, we converged at less than 250000 iterations with an error threshold of 10. The threshold was increased because we failed to converge with smaller values of epsilon. We can see that while the SSE for training set has converged, the testing set has an increasing SEE after less than 10000 iterations.

The weights for this convergence are:

```
[-2.07453389  0.18273088 -0.17259769  0.38092724 -8.22553078  3.27257917
  7.15960328  1.24046688  0.12840576  4.71432669  2.33722012  1.24908545
  8.92324305  7.61876262  1.27798846 -2.93152771  0.29035288 -1.0150404
  3.76744147 -2.67406016  1.34520077 -2.02111993]
```

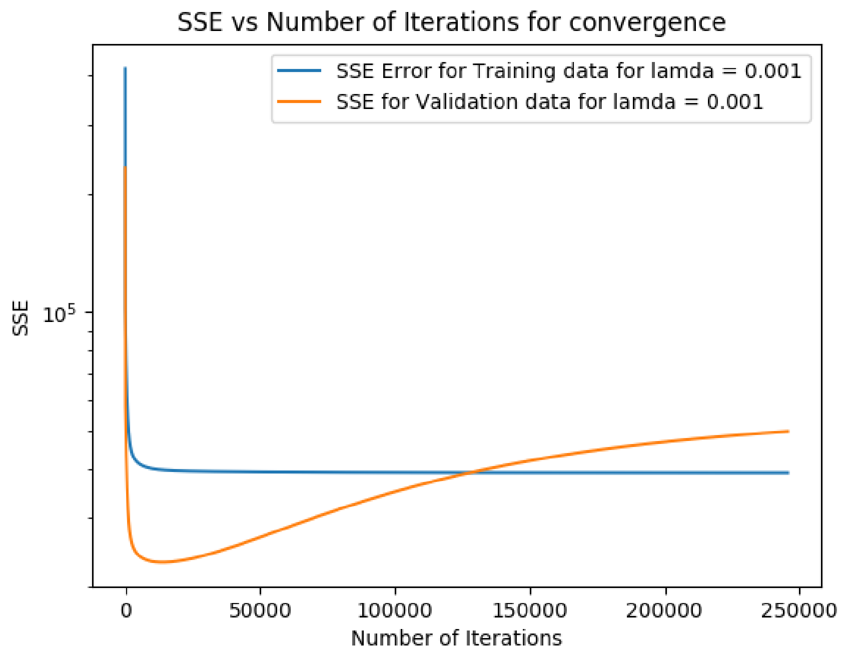
For lamda = 0.01, Epsilon = 10



As seen above, the convergence was reached at less than 250000 iterations. However, the SSE for the validation set is increasing.
The weights for this convergence are:

```
 [-2.06611591  0.18273116 -0.17255641  0.38106715 -8.39075208  3.28044831
  7.18373633  1.28544666  0.12477421  4.71917562  2.3343136   1.24998423
  8.92199761  7.64456417  1.28191914 -2.93364952  0.28947308 -1.01681308
  3.76758701 -2.67654659  1.33321703 -2.0780504 ]
```

For lamda = 1/1000, Epsilon = 10



We reached a convergence in between 240000 and 250000. However, the validation SSE is diverging as shown.
The weights for the convergence obtained are:

```
 [-2.06525518  0.18273104 -0.17255224  0.38108122 -8.40763449  3.2812567
  7.1861889   1.29013537  0.12440797  4.71965738  2.33401882  1.25007668
  8.92185343  7.64718473  1.28232352 -2.93386345  0.28938439 -1.01699241
  3.76760116 -2.6767946   1.33201238 -2.08392991]
```

Part 2.a. What trend do you observe from the training SSE as we change λ value?
Solution
As we saw in the first diagram, we see the trend that the value of SSE increases for the training data as we increase the value of lamda. It is because we introduce lamda to stop overfitting, thus as the value of lamda increases, SSE increases and we go towards underfitting.

2.b. **What tread do you observe from the validation SSE?**
Solution
For the validation data, as we can see, the SEE value is lower than the SEE for the training set for the initial case. However, as we decrease the value of lamda from 100, the SSE values tend to increase for the validation data despite the fact that we have used gradient descent to optimize the algorithm as seen in the plots above. The two SSE's intersect when lamda equals 1. So, basically, the model underfits for the validation set.

2.c. **Provide an explanation for the observed behaviors.**
Solution

It is observed that the SSE for training decreases and eventually converges while SSE increases with the variation of lamda as shown above. It is because when we introduce lamda, it is because we want to limit the importance of the features that have been given comparatively larger weights. So, our aim is to decrease the overfitting. While doing so, our model works fine for the training set because it has been modelled upon the training set. However, the model goes towards underfitting as we decrease the value of lamba for the validation set.

2.d. **What features get turned off for λ= 10, 10−2 and 0 ?**
Solution
As we can see from the weights reported above,
for lamda = 10, the features "month","day","year" and "square feet of lot in 15" get turned off because the weights for them reach near zero.
for lamda = 1/100, the features "month","day","year", and "year of renovation" turn off because of the same reason, their weights are near to zero in comparision to the other weights.
for lamda = 0, the features "month","day","year","square feet of living area","square feet of lot" , and "year of renovation" get reduced to zero.

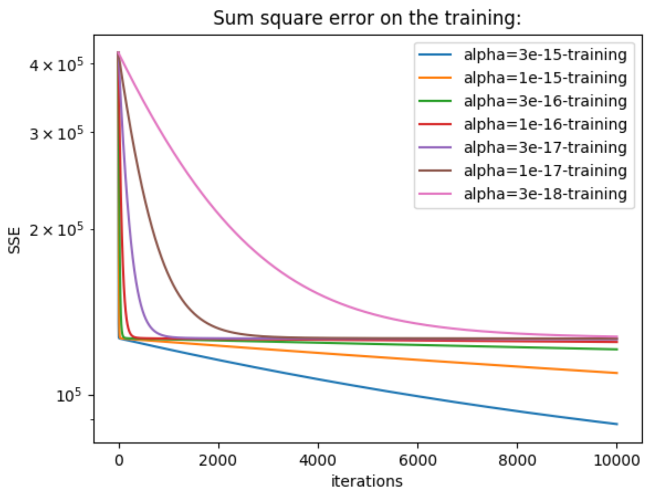
As we see, more the number of features get turned off, the more the model goes towards underfitting.
Also, it is important to note that since the two curves cross at lamda = 1, it is the best value for this hyperparameter.

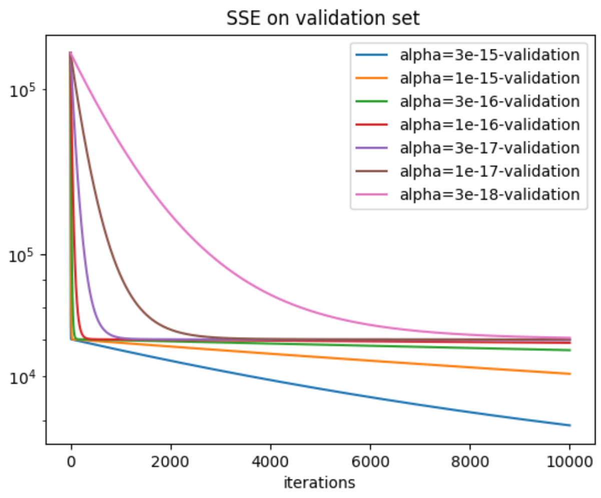
Part 3:

The complete feature set was used as prepared in part 0 without the normalization.

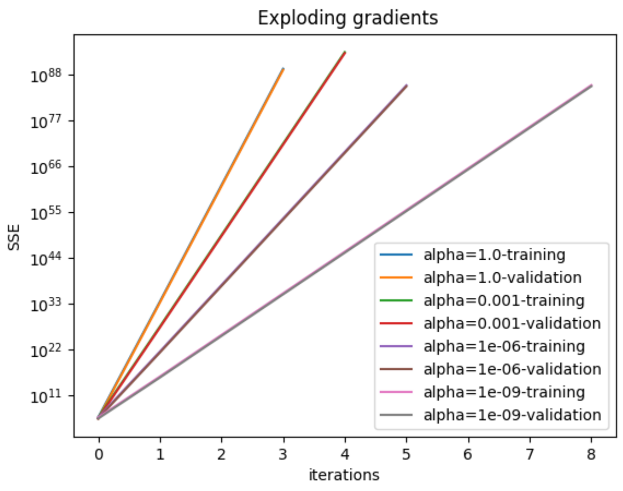
The iterations were fixed at 10000 per learning rate.

The results suggested that on most learning rates the gradient descent exploded. However, learning rates lower than 10^{-14} prevented the explosion. The best results came from learning rate





of $3*10^{-15}$ as shown below:



The **normalized** version was definitely easier to train. This is because normalization ensures all features are on the same scale and hence training weights are not influenced by feature scales.

Final Prediction for Test file:

The parameters used were:

Learning rate: 10^{-5} , Lambda: 10^{-3}

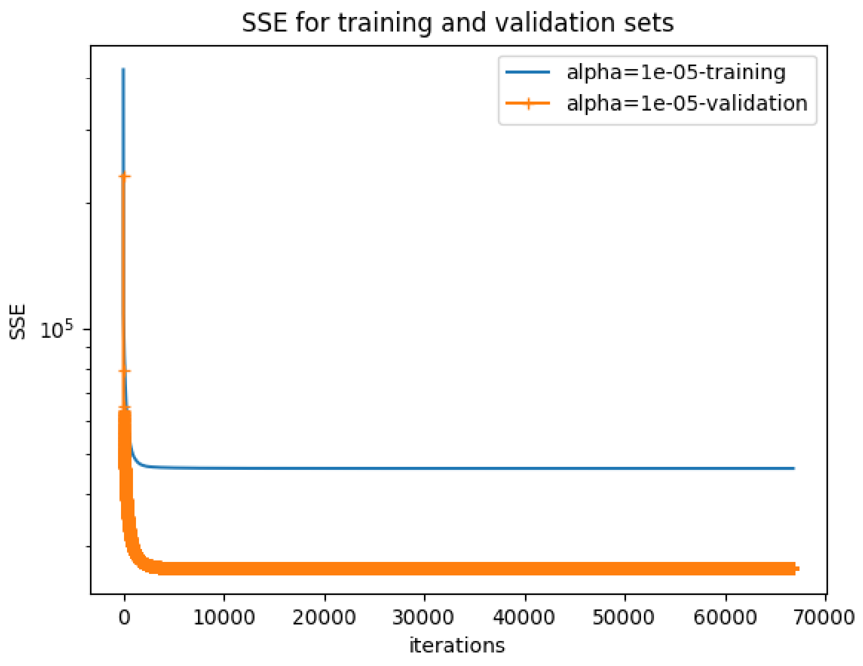
For this part only the reduced feature set was used and the following best features were selected based on previous experiments:

['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'grade', 'yr_built', 'zipcode']

With weight vector:

[-0.20096335 3.59983054 13.54391502 -3.22790081 0.39873157 4.76332901 1.97737031 12.34270594 -4.07021785 0.22470227]

And SSE value of *26522*.


SSE for training and validation sets
alpha=1e-05-training
alpha=1e-05-validation
SSE
10^5
iterations