IA 1 report

Name: Yijie Ren

Part 1:

(a)

X axis is iteration number, Y axis is the MSE value. The first 2 figures cannot converge, so an early stop was applied to both models.

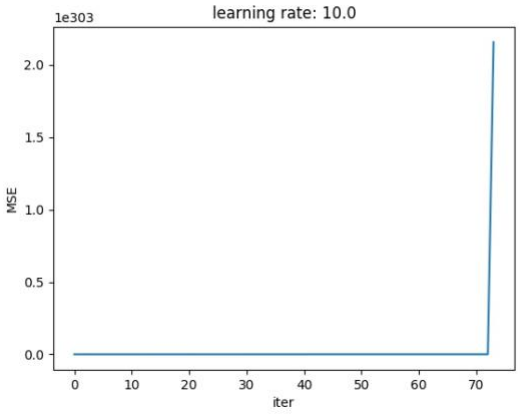

Figure 1 learning rate of 10

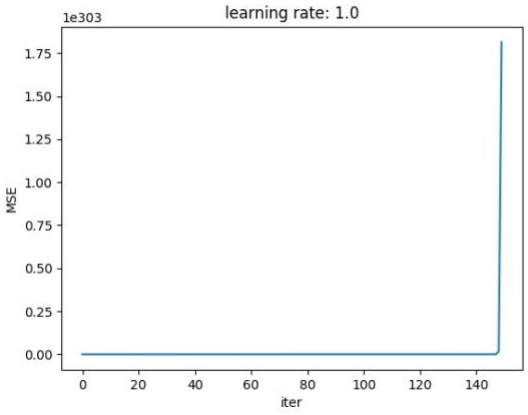

Figure 2 learning rate of 10^0

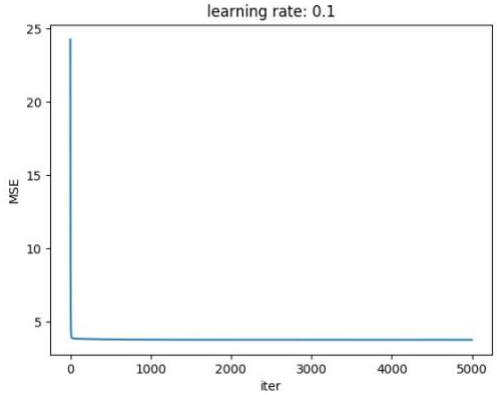

Figure 3 learning rate of 10^-1

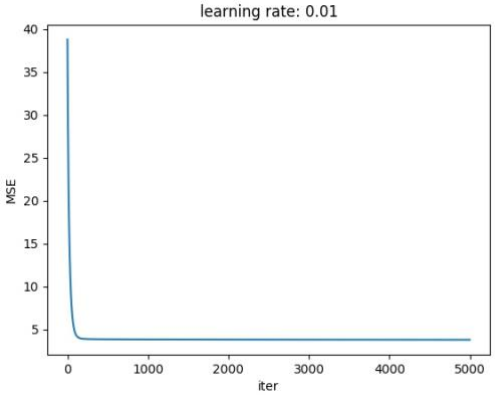

Figure 4 learning rate of 10^-2

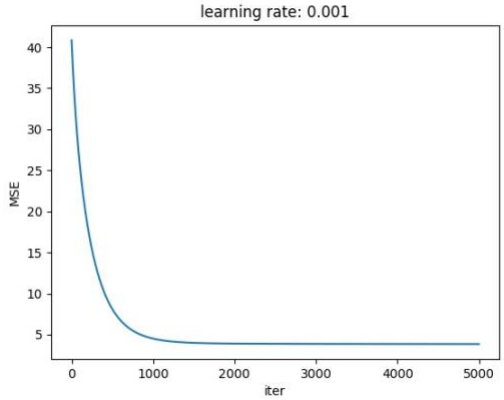

Figure 5 learning rate of 10^-3

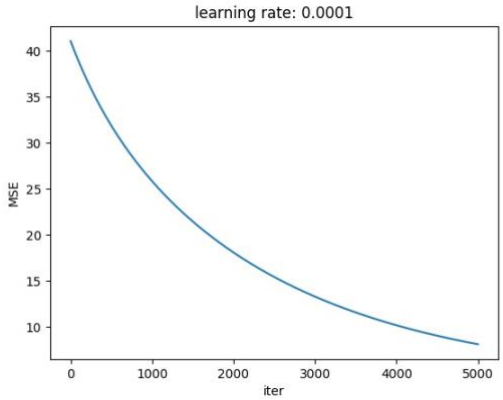

Figure 6 learning rate of 10^-4

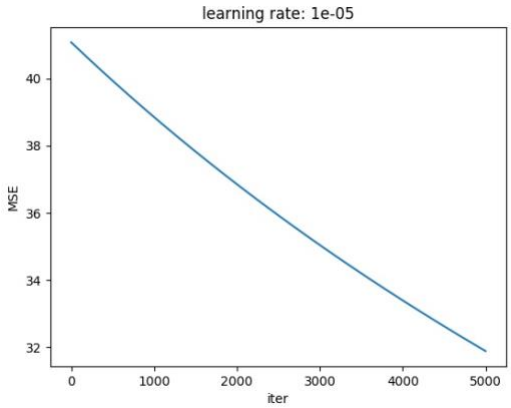

Figure 7 learning rate of 10^-5

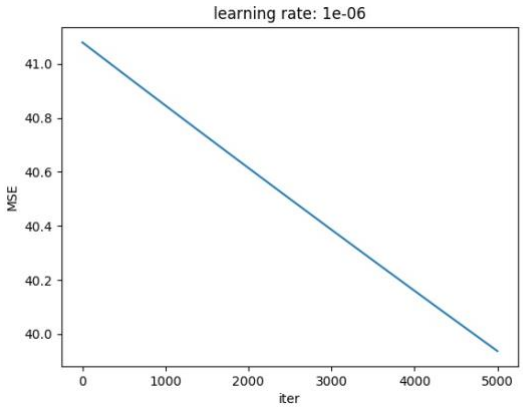

Figure 8 learning rate of 10^-6

As far as I observed, learning rates of 10^{-1}, 10^{-2}, 10^{-3} are all good for this dataset, since all these 3 learning rates can help the model converge and reach the final state. The learning rates of 10 and 10^{0} make gradient descent diverge.

(b)
Since the learning rates of 10^{-4}, 10^{-5}, 10^{-6} do not reach the final convergence, so I only tried 3 good learning rates to get the best MSE for validation set, below is the result:
Learning rate: 10^{-1} -> MSE: 4.50409967
Learning rate: 10^{-2} -> MSE: 4.63408348
Learning rate: 10^{-3} -> MSE: 4.76424005
From the results, it seems learning rate of 10^{-1} leads to the best validation MSE.

Under the situation of nearly identical MSE with different convergent learning rates, I'll tend to choose smallest learning rate. From the pictures in Part 1 (a), the line in Figure 5 with 10^-3 as learning rate is smoother, and small learning rate tend to have smaller step when updating parameters, which can avoid missing the minimum point, to some degrees.

(c)
The best converged solution that minimizes the validation MSE is to use learning rate of 10^-1, according to the MSE value in Part 1 (b). The learned weights for each feature are:

feature name	**learned weight**	**feature name**	**learned weight**
dummy	**5.33471603**	sqft_basement	0.15539435
bedrooms	-0.28139213	yr_built	-0.88339552
bathrooms	0.33905573	zipcode	-0.26342338
sqft_living	0.76346266	lat	0.83660396
sqft_lot	0.05816262	long	-0.30372827
floors	0.01813873	sqft_living15	0.14355167
waterfront	**3.99367919**	sqft_lot15	-0.09928372
view	0.4472015	month	0.05487913

condition	0.19985323	day	-0.05062428
grade	1.11534671	year	0.17376014
sqft_above	0.75631508	age_since_renovated	-0.10258699

If “dummy” can be considered as a feature, then dummy is the most important in deciding the house prices according to the learned weights. If “dummy” does not count as a feature, then “waterfront” is the most important.

Part2:

(a)

X axis is iteration number, Y axis is the MSE value. The first 2 figures cannot converge, so an early stop was applied to both models.

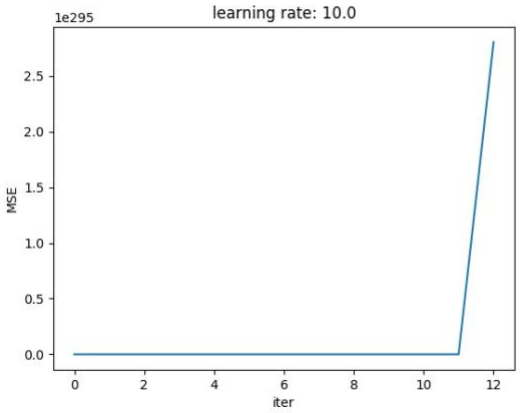

Figure 9 learning rate of 10, no normalization

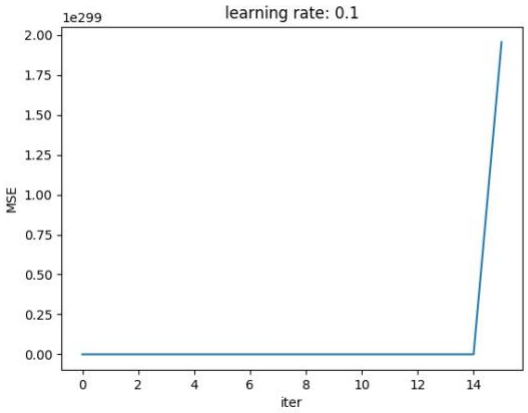

Figure 10 learning rate of 10^-1, no normalization

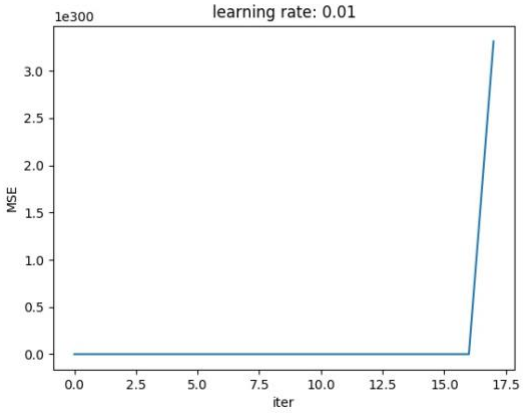

Figure 11 learning rate of 10^-2, no normalization

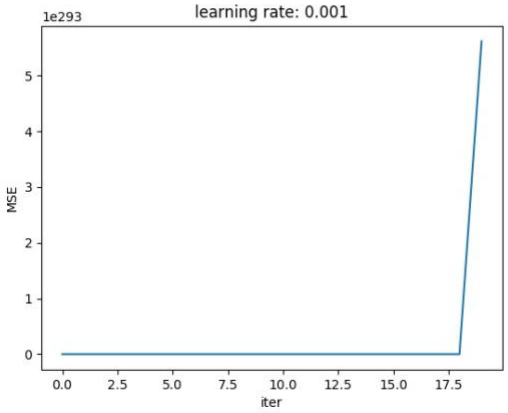

Figure 12 learning rate of 10^-3, no normalization

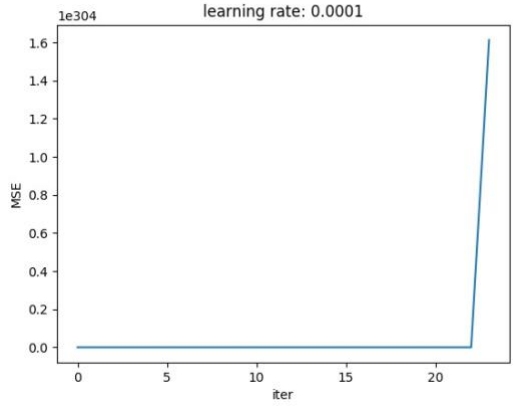

Figure 13 learning rate of 10^-4, no normalization

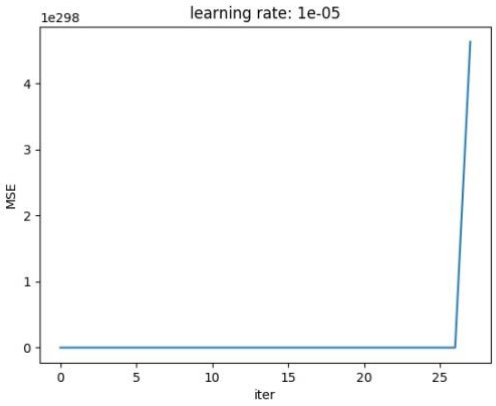

Figure 14 learning rate of 10^-5, no normalization

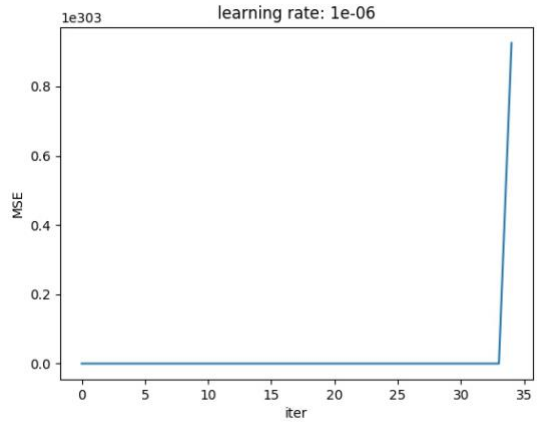

Figure 15 learning rate of 10^-6, no normalization

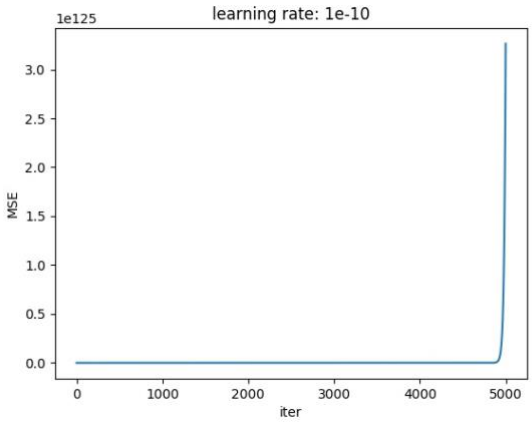

Figure 16 learning rate of 10^-10, no normalization

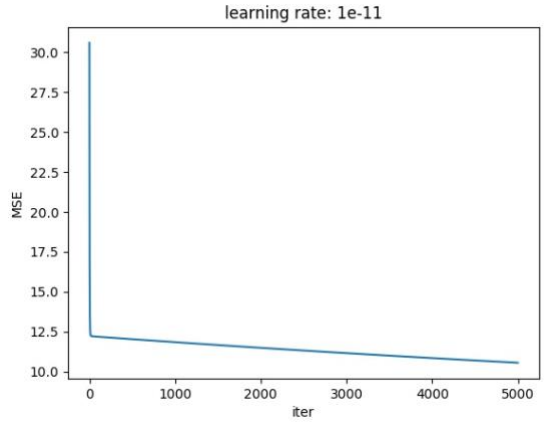

Figure 17 learning rate of 10^-11, no normalization

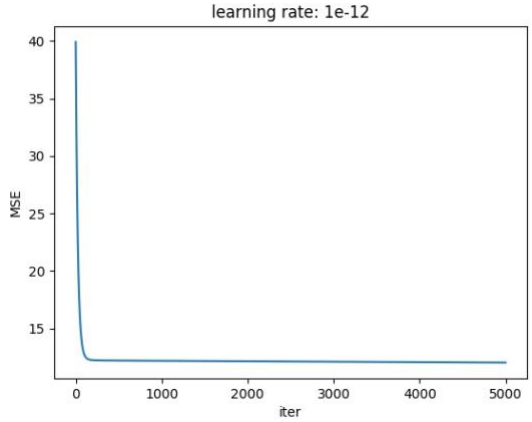

Figure 18 learning rate of 10^-12, no normalization

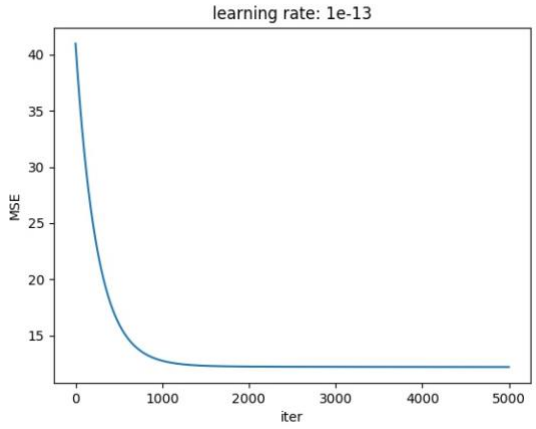

Figure 19 learning rate of 10^-13, no normalization

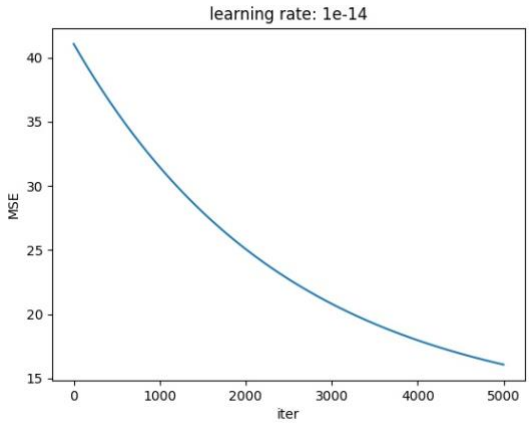

Figure 20 learning rate of 10^-14, no normalization

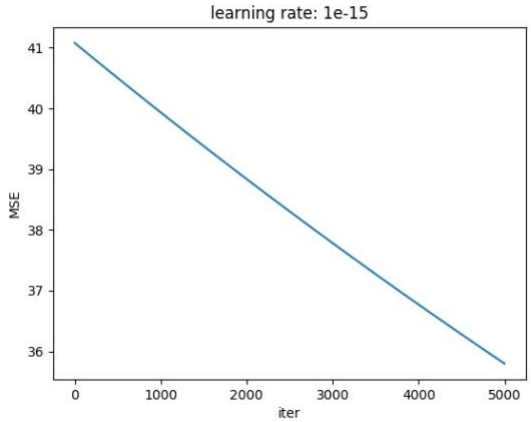

Figure 21 learning rate of 10^-15, no normalization

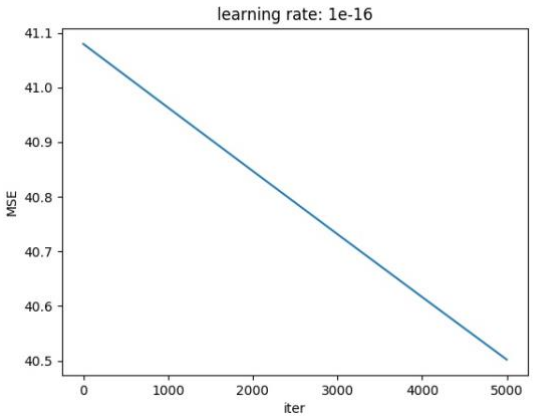

Figure 22 learning rate of 10^-16, no normalization

From the figures above, only very small learning rates of 10^{-12}, 10^{-13} work for the non-normalized data, leading to convergence.

The data with normalization is easier to train. Normalization can make all the features of same weights at the beginning of the training, which avoid the features with large values originally contributing too much to the weight update during training.

(b)
I tried the converged 2 learning rates to get the best MSE for validation set, below is the result:
Learning rate: 10^{-12} -> MSE: 14.13813261
Learning rate: 10^{-13} -> MSE: 14.31251084

From the results, it seems learning rate of 10^{-12} leads to the best validation MSE, so the weights learned under learning rate of 10^{-12} are listed below:

feature name	**learned weight**	**feature name**	**learned weight**

dummy	5.33293968e-10	sqft_basement	4.44015100e-06
bedrooms	1.17640522e-08	yr_built	1.09154532e-06
bathrooms	1.47708573e-08	zipcode	5.22475666e-05
sqft_living	2.26354936e-05	lat	2.69106739e-08
sqft_lot	4.49491746e-06	long	-6.52036300e-08
floors	5.86298011e-09	sqft_living15	1.46725303e-05
waterfront	5.61433625e-10	sqft_lot15	4.40756145e-06
view	1.01507412e-08	month	3.65202854e-09
condition	3.00982406e-09	day	-1.85064324e-09
grade	3.08826999e-08	year	1.07410942e-06
sqft_above	1.81953426e-05	age_since_renovated	-6.11490188e-08

Comparing to the weights to Part 1 (c), the learned weights here are very small. The explanation for this phenomenon could be the magnitude imbalance of values for different features. There are very large values like feature "sqft_lot", versus small values like feature "bathrooms". When updating weights during training, the large values will contribute much more to Δw than small values, so the weights of these large features must be very small to counteract the domination of large values, trying to balance contributions from small values.

The interpretation of weights is not considered as larger weights indicating more importance of the corresponding features. The larger weights here tend to mean the values of corresponding features are very small.

Part 3:
I tried learning rate of 10^{-1}, 10^{-2}, 10^{-3}, and 10^{-1} gives me smaller MSE on validation set, so I'll use 10^{-1} as my choice of learning rate, where the MSE is 4.51988334. The learned weights for each feature are listed below:

feature name	**learned weight**	**feature name**	**learned weight**
dummy	5.33483197	sqft_basement	0.160669622
bedrooms	-0.283035353	yr_built	-0.874144492
bathrooms	0.333383131	zipcode	-0.272837819
sqft_living	0.803505219	lat	0.841386418
sqft_lot	0.0519313895	long	-0.286708261
floors	0.00440807630	sqft_lot15	-0.0954975224
waterfront	3.97650194	month	0.0550034440
view	0.461591746	day	-0.0504091496
condition	0.195831784	year	0.172768064
grade	1.15783427	age_since_renovated	-0.0905210450
sqft_above	0.797492511		

The new model here performs almost the same as the one in Part 1 (c), since the MSE on validation set is almost the same. The weight of feature “sqft_living” changes from 0.7 to 0.8 from model in Part 1 (c) to the current model without feature “sqft_living15”, which increases just a little bit.

Thus, consider the situation in general when two features x1 and x2 are redundant, I expect the weights (w1 and w2) learned by both features will be either very similar (same) or slightly smaller than w1 learned with just x1. The reason is that the 2 features are redundant, so the information that one feature carries is basically very similar to the other feature. Then the weight either learned by both features or single feature should be very similar. I think the slightly higher weight learned by only one feature x1 is caused by the abandon of the other feature x2. Due to the abandon of x2, though x2 is redundant, it still carries slightly different information comparing to x1. So the weight of x1 will be slightly larger as a kind of compensation to weight x2.

Part 4:

I tried several feature engineering ways, listed as below:

1. First, I tried to convert feature “grade” into 3 categories (-1, 0, 1), since it is stated in the data description that the feature “grade” can be categorized as “short of building construction and design der”, “average level of construction” and “high quality level of construction”. However, this way does not work, it makes the MSE of test set much higher than before. I assume this is because I actually reduced the information of feature “grade” by categorizing them into only 3 numbers, instead of the original consecutive numbers from 1-13.
2. Second, I found both “sqft_above” and “sqft_basement” are indicating the interior housing space, so I add them together, forming a new feature. It does improve very little of MSE on test set, around 1%... So definitely, it is not good enough.
3. Third, I found that the feature “zipcode” has its own distinct value set, so I treat each unique “zipcode” as a new feature, then do one hot encoding for each unique feature value. It turns out adding more features with 0 and 1. This time, the MSE of test set decreases around 1/3, so this is a good way to improve the model performance. However, I this method can only be applied to this dataset, since the feature “zipcode” will be different when it generalizes to different district area (means different zip code). Since one hot encoding works well for this dataset, in order to get better results of this dataset, I also changed the features with Index values – “grade”, “view”, “condition”. The performance then increases a little bit after the above change.
4. I’m not sure if this can be counted as feature engineering. I found there are some predictions of price are negative, so I change the negative predictions to the min price value in the training dataset. It also improves the model performance on test set. Since in real cases, a house price cannot be negative, it is necessary to make it the min value of training set – we cannot see test set in real case.