INTROUCTION TO MACHINE LEARNING - 236756

MAJOR HW4

Itay Lasch & Yonatan Battat

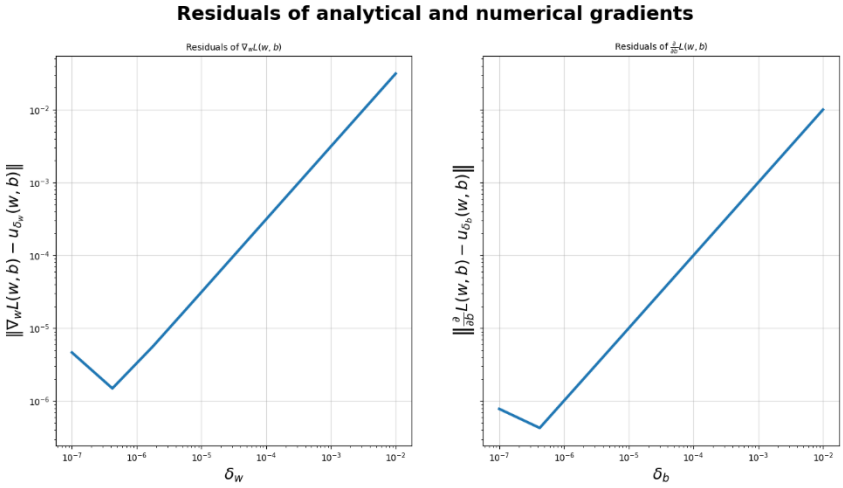
Section 1: Linear regression implementation

A1:
The MSE loss function for our regressor:

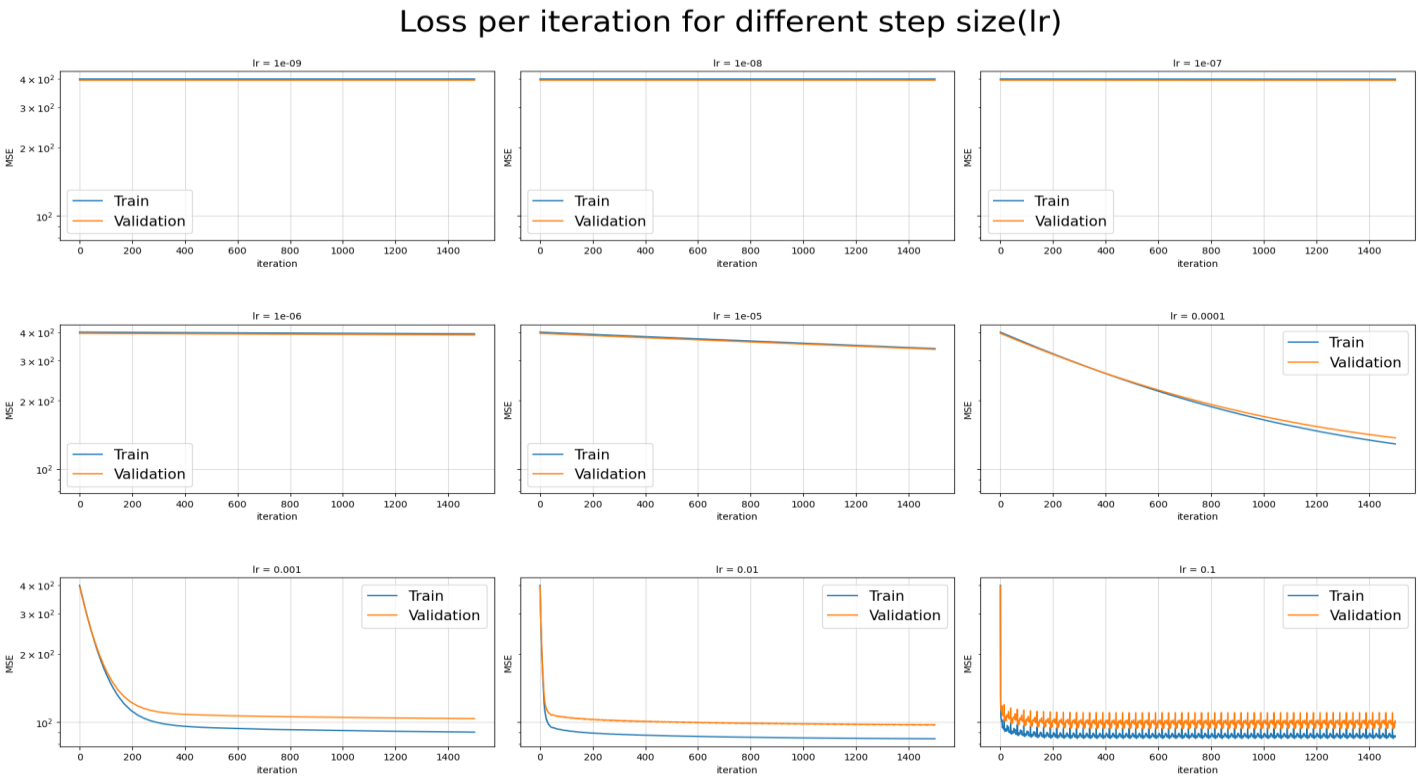
$$L(\underline{w}, b) = \frac{1}{m} \left\| X\underline{w} + \underline{1}_m \cdot b - \underline{y} \right\|_2^2$$

$$\frac{\partial}{\partial b} L(\underline{w}, b) = \frac{2}{m} \cdot (\underline{1}_m)^T \cdot \left(X\underline{w} + \underline{1}_m \cdot b - \underline{y} \right) = \frac{2}{m} \cdot (\underline{1}_m)^T \cdot \left(X\underline{w} - \underline{y} \right) + \frac{2}{m} \cdot mb$$
$$= \frac{2}{m} \cdot (\underline{1}_m)^T \cdot \left(X\underline{w} - \underline{y} \right) + 2b$$

A2:



A3:



As we can see from the graphs above, for $lr \in \{1e-9, \dots, 1e-6\}$ the loss is almost constant and there is a very small improvement per iteration. The reason is that the step size is too small and therefore w, b barely change between iterations, since our number of iterations is limited $(= 1500)$ we do not converge to ideal results for these lr values. Moreover for $lr = \{1e-9, 1e-8, 1e-7\}$ we tested the loss per iteration with 10000 iterations and still saw that there is no change in the loss and no improvement per iteration which made us conclude that for these small lr values there are numerical errors that convert the steps to zeros.
For lr values higher than $1e-5$ including we can see that MSE is starting to decrease which means there is an improvement in loss per iteration. The higher lr, in particular $lr = 1e-3, 1e-2$, the decrease in loss is bigger and it converges faster to its minimum value.
We can see for $lr = 0.1$ there is not convergence to a minimum value, this is happening because the step size is too big which makes MSE jump between different ideal w's and not reach the actual minimum.

The best lr size in our opinion is $lr = 0.01$. As we explained it converges to minimum value very quick and although we saw that for $lr = 0.1$ we get even a smaller train and validation loss, we prefer $lr = 0.01$ since unlike $lr = 0$, it converges to a single minimum value and the difference in losses between the two $lr's$ is quite small.

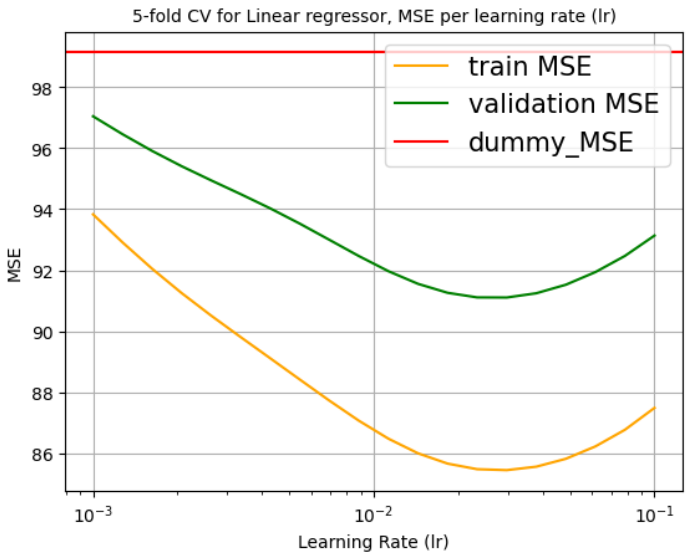
We don't think it makes sense to increase the number of iterations, we can see that as we approach the minimum value the slope of the validation MSE is getting smaller and smaller. That happens because as we get closer to the minimum value the gradient gets smaller as well, so each step is smaller than the one before him. Thus, adding more iterations won't make a significant change in the minimum validation loss (We verified this with 10000 iterations).

Section 2: Evaluation and Baseline

A4:

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	99.13589	99.3029

A5:



The optimal lr is $lr \cong 0.0297$,which results with validation MSE of $\cong 91.106$

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	99.13589	99.3029
Linear	2	85.45293	91.10669

A6:

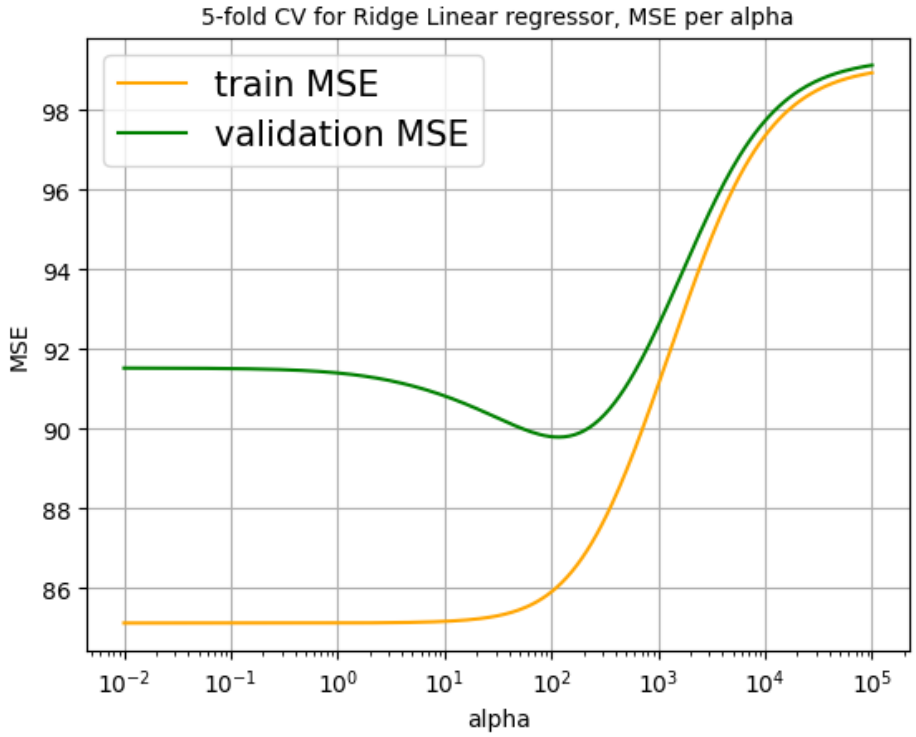
Had we chosen not to normalize the features, it wouldn't impact the Dummy model at all since it is always returns the mean value of the target variable 'contamination_level' as a prediction to all samples. Since the target variable is not affected by normalization the MSE will stay the same.

As for the linear model, we wouldn't see a difference as well. As we saw in the lectures LS problem is convex and has a global minimum, and normalizing preserves convexity since it performs convexity preserving operations such as scaling and summing. Since our problem is convex with or without normalizing the features, with the right learning step and enough iterations we are guaranteed to reach that global minimum. So, both models' training performances will not change.

* Each model will probably need different number of iterations and different learning rate which will yield different w.

Section 3: Ridge linear regression

A7:



The optimal α is $\alpha \cong 118.46$, which results with validation MSE of $\cong 89.79$

A8:

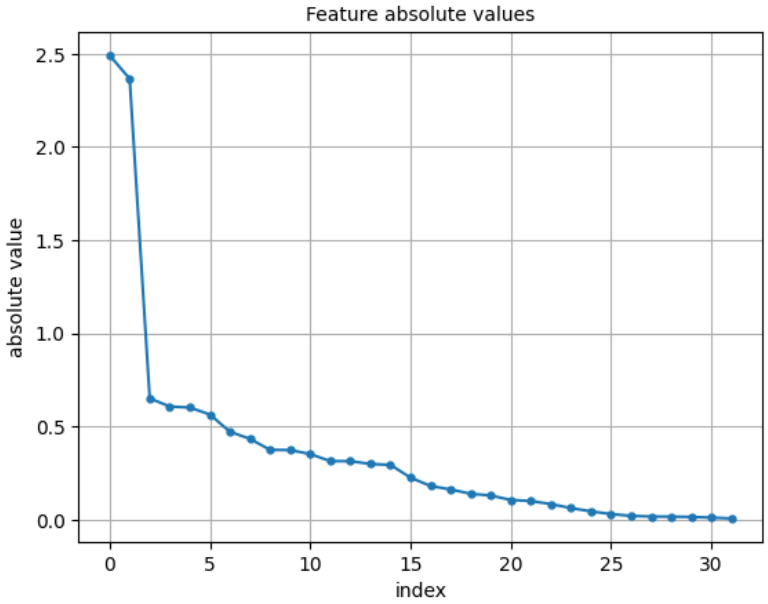
Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	99.13589	99.3029
Linear	2	85.45293	91.10669
Ridge Linear	3	86.07993	89.7909

A9:

The 5 features with the largest coefficients (in absolute value) are:

1. PCR_01 – 2.49206403
2. sugar_levels – 2.36611658
3. household_income – 0.6518341
4. sport_activity – 0.60772657
5. happiness_score – 0.60249684

A10:



A11:

The magnitude of each coefficient is interesting because as we learned each feature gets a coefficient based on how much correlated it is with our target variable, i.e. the more expressive and meaningful a feature is for our prediction the bigger its corresponding coefficient would be.
This can help us for example with feature selection when we want to decrease the number of dimensions in our problem, in order lower the memory and time complexity without losing a lot accuracy wise.

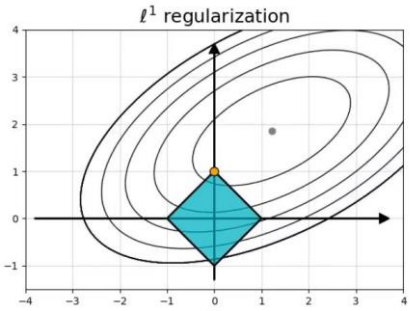
A12:

If we had not normalized the training performance of our linear ridge regressor would suffer and might miss the optimal solution. There are a few problems that led us to this answer.
First, as we learned in the tutorial: $w^{Ridge} = \underset{w}{\operatorname{argmin}} \frac{1}{m}(\sum_{i=1}^{m}(y_i - w^T x_i)^2 + \lambda\|w\|_2^2)$
$= \underset{w}{\operatorname{argmin}} \frac{1}{m}(\sum_{i=1}^{m}(y_i - w^T x_i)^2 \ , s.t \ \|w\|_2^2 \leq c.$
In words, it means that we put constraints on the norm of w which can be seen as constraints on the coefficients of the features of sample x.
Let's look at a dataset that has different features with different scales. Now we want to look at two different features, one on a large scale and the other on a small scale. There is a large difference in scales between these two features, in order to compensate for that and let both of them have an impact on the prediction we are likely to give the small scale feature a bigger coefficient, but due to the constraint on the coefficients this solution might won't be in our possible solutions space and therefore we won't be able to get that result.

Another problem is that Ridge regularization uses l_2 norm which is sensitive to different scales of the features. If we are deciding to not normalize our data, the features with bigger scales will have a larger impact on the regularization part of the ERM loss and therefore would get smaller coefficients.
As we explained in A11 the magnitude coefficient is directly proportional to how much the corresponding feature is correlated\uncorrelated with the target variable. Unregularized data could lead our regressor to yield a suboptimal classifier that his coefficients are decided based on the scales of the features instead of how much they actually impact our predictions.

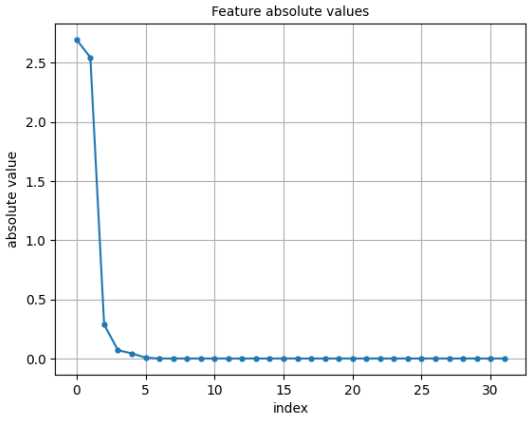
A13:

If we had used Lasso regressor instead of Ridge I would expect most of the coefficients to zero out. As we saw in tutorial 9, Lasso regressor uses l_1 norm, and so our possible plain space is different than the one of Ridge regressor. To understand more visually we can look at the plot that shows the ERM loss as function of w on a dataset with 2 features.



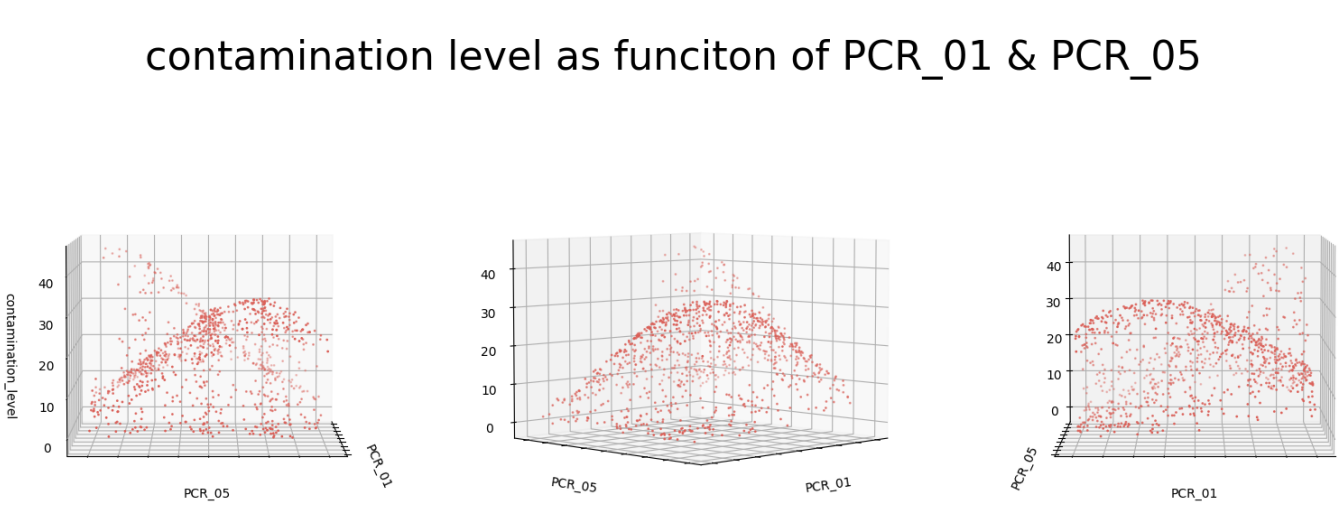
As we can see there are a few options to choose a proper $\underline{w}$ that will yield us the minimum ERM loss, but since we use Ridge/Lasso regression we also emphasize on the norm of $\underline{w}$. Therefore, in Lasso regression, which uses the l_1 norm, SGD will choose the $\underline{w}^*$ in our possible option space that has the smallest norm of them all. In order to achieve that the SGD will yield a $\underline{w}^*$ that most of its features are zero out i.e. their coefficients are zero.

Just to clarify here is a coefficient graph if we were actually using Lasso regression:



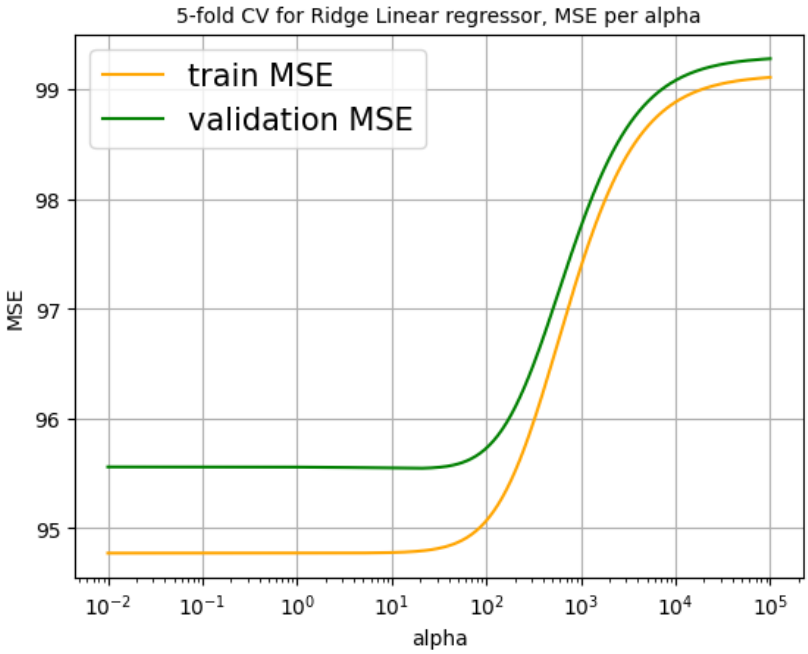
Section 4: Polynomial fitting (visualization)

A14:



As we can see from the graphs above, it seems that the connection between 'contamination_level' and the features 'PCR_01', 'PCR_05' cannot be expressed using a straight line and therefore linear regression won't work on the data as it currently is. It seems that the dots are scattered in some paraboloid shape so linear regression with polonomial feature mapping might work for us.

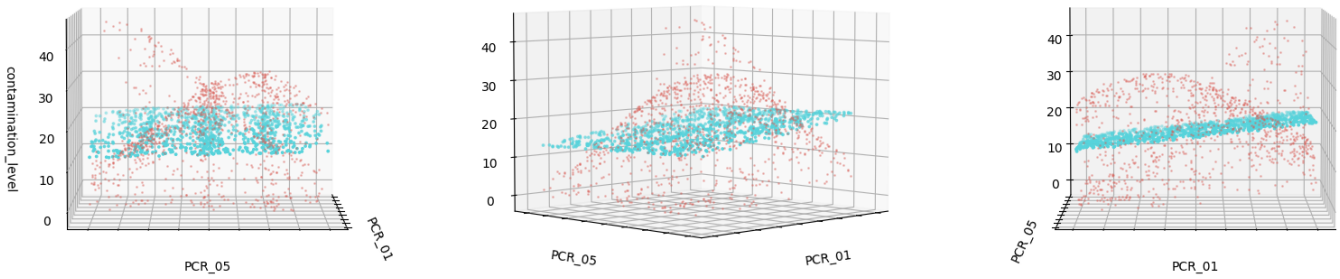
A15:



The optimal α is $\alpha \cong 16.119$ which results with validation MSE of 95.5436.

A16:

contamination level as funciton of PCR_01 & PCR_05 vs. predictions



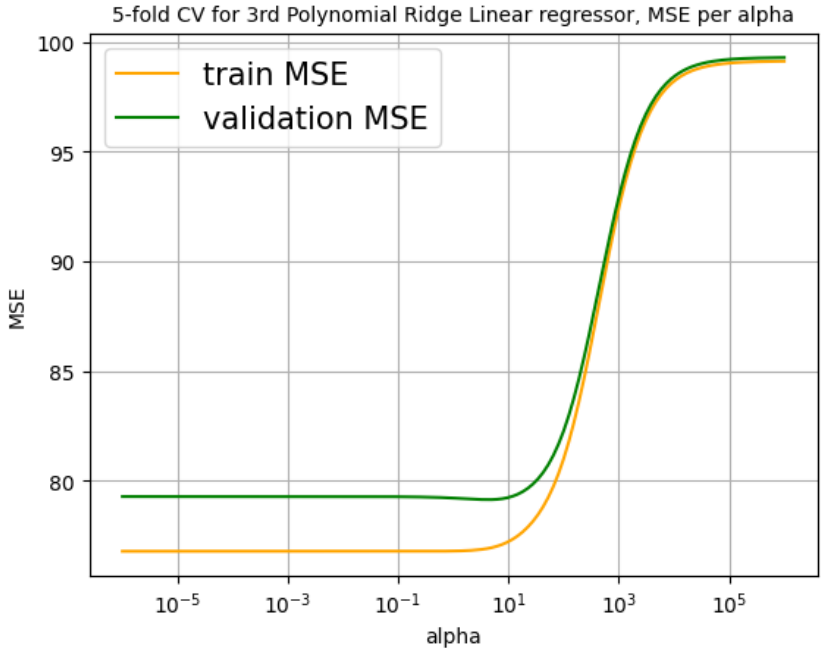
A17:

It is important to normalize the features once more after the polynomial feature mapping for the same reason explained in (A12). We start with the 2 features 'PCR_01','PCR_05' let's mark them as x_1, x_2 appropriately. Now after the 3^{rd} degree polynomial feature mapping the features are: $x_1,\ x_2,\ x_1^2,\ x_2^2,\ x_1 \cdot x_2,\ x_1^3,\ x_2^3,$ $x_1^2 \cdot x_2, x_1 \cdot x_2^2$.

Since we normalized the features with MinMax normalization to be in between (-1,1) prior to the feature mapping, all the new features will be much smaller than the original ones.

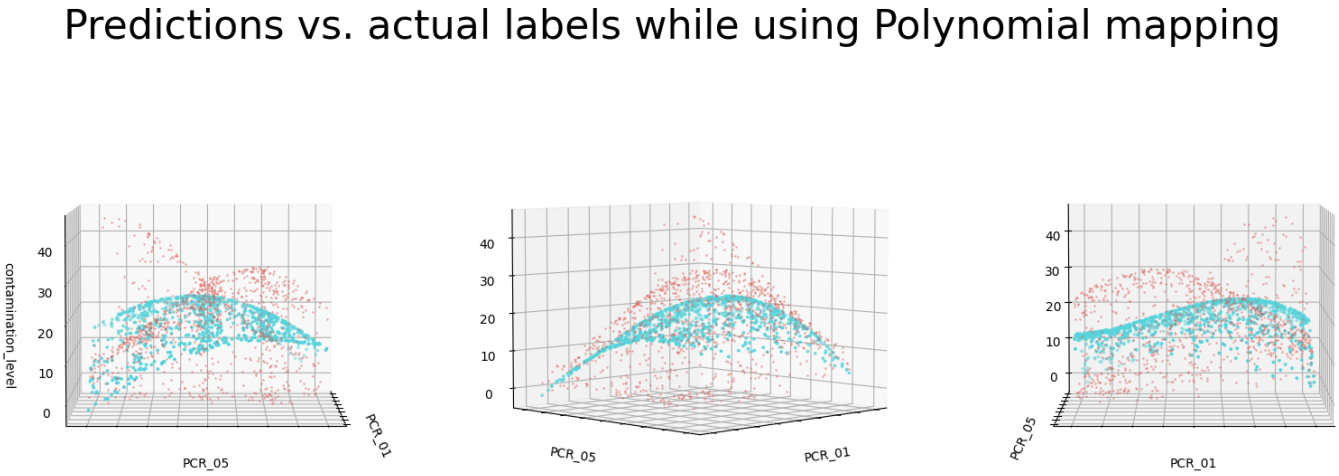
To prevent the problem that occur from different scales in the features, as explained in (A12), it is important to normalize the features to the same scale once more.

A18:



The optimal α is $\alpha \cong 4.023$ which results with validation MSE of 79.1425.

A19:



A20:

There are a few different aspects to discuss when comparing the two different models:
First, as we can see after using the 3^{rd} degree polynomial feature mapping our validation MSE was lower than the model on the original data (79.14 compared to 95.54), which means our predictions were closer to the actual label after the feature mapping.
Also we can see from the scattered plots in (A16) and (A19) that on the original data set our predictions created a straight plane that crossed the actual target variable values. On the other hand, in the polynomial model, we can see that even though our predictions were not perfect, most of the time they were pretty close to the actual values and depicted the same shapes as them.

The reason for those differences is that the feature mapping added complexity to the model and enabled it to fit the data more accurately.

Section 5: Fitting Gradient Boosted Machines (GBM)

Task before Q21:
The preprocessing we did is the one we did in the beginning of the exercise and is identical to the preprocessing we did in previous exercises and for the same reasons. We normalized the features using StandardScaler and MinMaxScaler in the following manner:
MinMaxScaler: PCR_01, PCR_02, PCR_03, PCR_09, sport_activity
StandardScaler: PCR_04, PCR_05, PCR_10, sugar_levels

A21:
Squared error – As we learned in the lecture this loss function tries to fit the residuals as much as possible at each iteration. This is relevant for us since we use bounded decision trees as weak leaners and therefore at each iteration, we won't be able to fit the data perfectly and we would have residuals between our predictions and the actual target variable. The squared error loss function penalizes large deviations from the target variable while neglecting small residuals.
By emphasizing the residuals at each iteration, we are minimizing the MSE and with enough iterations can get good results.

Absolute error – Using this loss function, we are trying to minimize the mean of absolute differences between our predictions and the actual target variable. As opposed to the squared error loss function the absolute error is more robust to outliers because it doesn't magnify the results of large residuals.
It's important to note that this loss function is not differentiable at zero, but we can overcome that using methods like SGD.
Overall it seems that this loss function will perform well on our data and can be relevant for our cause.

Hubert – The Hubert loss function combines the two above loss functions by defining a threshold of the residuals. This way we can enjoy both worlds, for small residuals in prediction, under some δ we would want to use the squared error function to neglect those residuals and for residuals above this δ we would use the absolute error function to target them as our main objective to minimize with robustness to outliers.
We decided above that both MSE and MAE loss function are relevant

Quantile – The quantile loss function behaves a bit differently than the rest of the loss functions above. It does not try to minimize the MSE or MAE, but instead it tries to minimize a loss function that captures the difference between the predicted quantiles and the actual quantiles. Therefore, for each sample, $h(x_i)$ is supposed to return a value in [0,1] which will represent the quantile of x_i, i.e. what is the percentage of samples that their contamination level is less than $x_i's$ contamination level. For this to work we need to hold a quantile for each x_i as well as it's contamination level variable. This loss function works with values between [0,1] and we want to work with values across all $\mathbb{R}$.
Since this loss function requires unnecessary calculations of quantiles and doesn't work with values in our preferred domain, we find it less relevant for our scenario and would probably not work with it.

A22:
Let's see the similarities and differences between Random Forests and Gradient Boosted Trees:
Similarities:

- Both algorithms use assembling to put together information from different small depth-bounded decision trees and make predictions with it.
- Both algorithms are nonlinear and can capture non-linear relationships between the features and the target variable due to the fact they are both using decision trees.

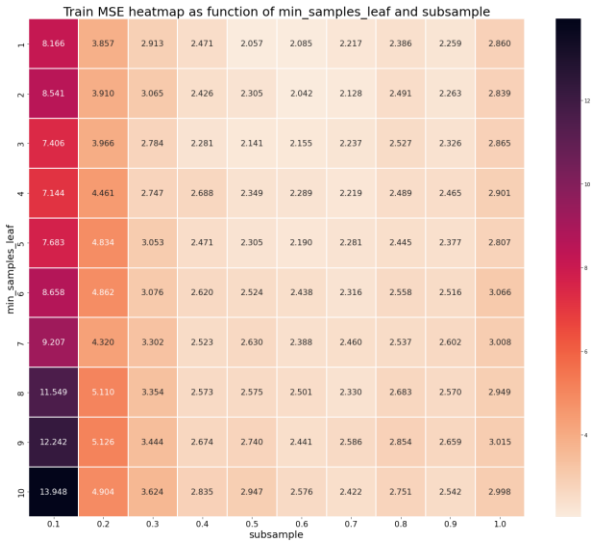
Differences:

- Training set: Random Forest creates different subsets of out training dataset using bootstrapping while in Gradient Boosted Trees we are using the entire dataset.
- Trees creation: Random Forest creates individual independent small decision trees based on each subset. Gradient Boosted Trees creates each tree serially where each tree is dependent on the trees that were created before him.
- Final prediction: Random Forests decides the final prediction value based on the prediction values decided by each independent tree (majority in classification or mean in regression) whereas in Gradient Boosted Trees the values predicted by each tree in weighted and the total sum of the weighted values is the prediction of the algorithm.
- Usage of features: Random Forests randomly choose at each iteration d' features and creating the decision tree based on them, whereas Gradient Boosted Trees takes into consideration all the features in each iteration.
- Variance and bias: Random Forests increases variance between the trees by random selection of features which in turn results in less variance in the predictions as we learned in the lecture, whereas Gradient Boosted Trees reduces bias by lowering the error at each iteration.

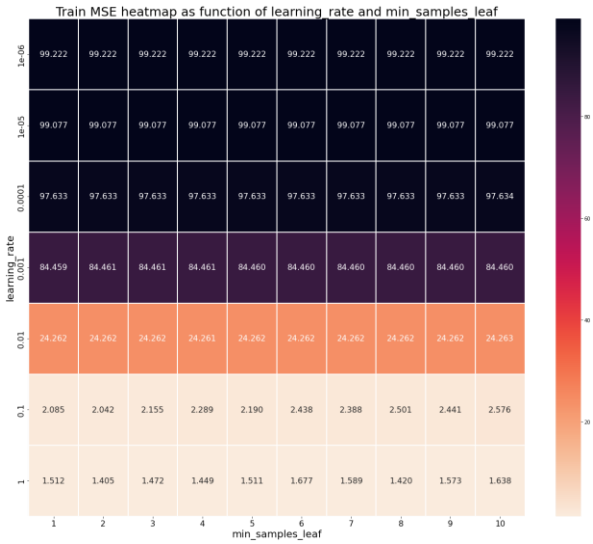
A23:

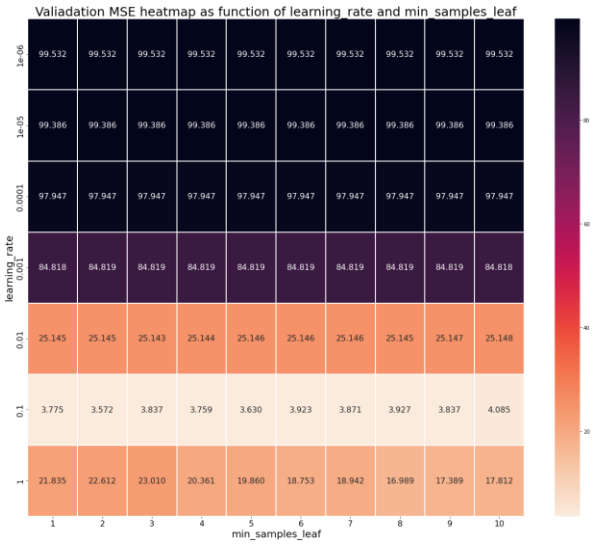
First we ran GridSearch CV on the 4 hyper parameters 'loss', 'learning_rate', 'subsample' , 'min_samples_leaf' and we got that the optimal values for them are:
$loss: hubert$
$learning\_rate: 0.1$
$subsample: 0.6$
min_sample_leaf: 2

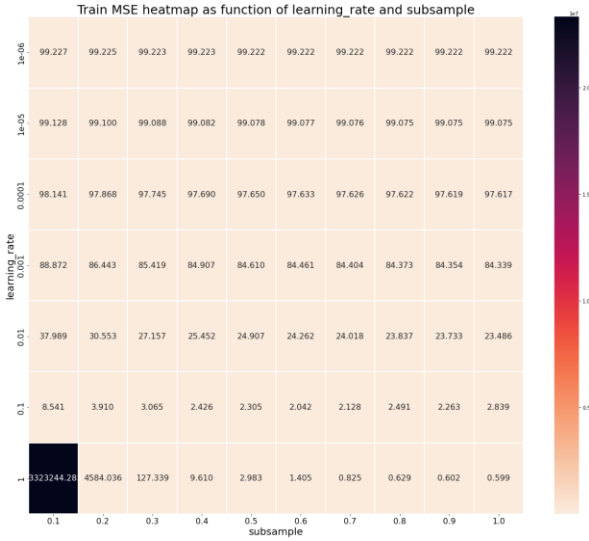
With these optimal values the **validation MSE score is** $\cong \mathbf{3.57}$ and the **train MSE score is** $\cong \mathbf{2.04.}$
Now in order to get heatmap for each pair of parameters out of there four we set each time 2 parameters to their optimal value and get the results for the other two parameters.


Train MSE heatmap as function of min_samples_leaf and subsample
min_samples_leaf
subsample
1 8.166 3.857 2.913 2.471 2.057 2.085 2.217 2.386 2.259 2.860
2 8.541 3.910 3.065 2.426 2.305 2.042 2.128 2.491 2.263 2.839
3 7.406 3.966 2.784 2.281 2.141 2.155 2.237 2.527 2.326 2.865
4 7.144 4.461 2.747 2.688 2.349 2.289 2.219 2.489 2.465 2.901
5 7.683 4.834 3.053 2.471 2.305 2.190 2.281 2.445 2.377 2.807
6 8.658 4.862 3.076 2.620 2.524 2.438 2.316 2.558 2.516 3.066
7 9.207 4.320 3.302 2.523 2.630 2.388 2.460 2.537 2.602 3.008
8 11.549 5.110 3.354 2.573 2.575 2.501 2.330 2.683 2.570 2.949
9 12.242 5.126 3.444 2.674 2.740 2.441 2.586 2.854 2.659 3.015
10 13.948 4.904 3.624 2.835 2.947 2.576 2.422 2.751 2.542 2.998
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

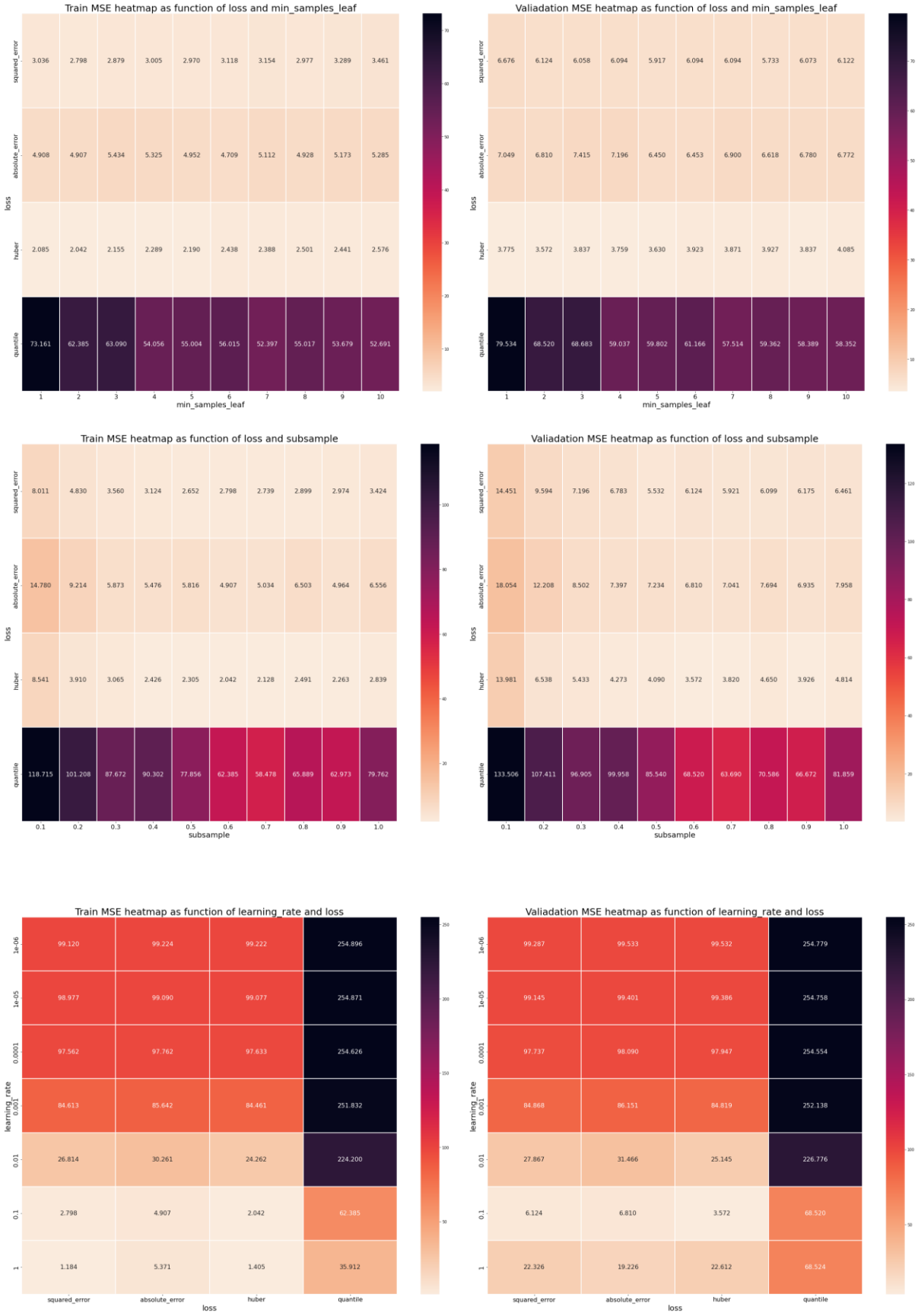

Valiadation MSE heatmap as function of min_samples_leaf and subsample
min_samples_leaf
subsample
1 13.464 6.739 5.278 4.447 3.937 3.775 4.011 4.432 3.992 4.810
2 13.981 6.538 5.433 4.273 4.090 3.572 3.820 4.650 3.926 4.814
3 11.449 6.593 4.779 3.879 3.871 3.837 3.776 4.516 3.981 4.694
4 10.898 6.863 4.527 4.500 3.981 3.759 3.667 4.326 3.996 4.725
5 11.087 7.326 4.551 4.291 4.044 3.630 3.707 4.223 3.850 4.513
6 11.958 7.084 4.717 4.100 4.243 3.923 3.852 4.286 4.057 4.621
7 13.035 6.830 5.028 4.038 4.196 3.871 3.879 4.218 4.136 4.742
8 15.504 7.508 5.012 4.177 4.193 3.927 3.714 4.428 4.140 4.612
9 16.563 7.474 5.108 4.411 4.340 3.837 4.179 4.590 4.219 4.600
10 17.769 6.953 5.414 4.616 4.537 4.085 3.856 4.509 4.047 4.561
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0


Train MSE heatmap as function of learning_rate and min_samples_leaf
learning_rate
min_samples_leaf
1e-06 99.222 99.222 99.222 99.222 99.222 99.222 99.222 99.222 99.222 99.222
1e-05 99.077 99.077 99.077 99.077 99.077 99.077 99.077 99.077 99.077 99.077
0.0001 97.633 97.633 97.633 97.633 97.633 97.633 97.633 97.633 97.633 97.634
0.001 84.459 84.461 84.461 84.461 84.460 84.460 84.460 84.460 84.460 84.460
0.01 24.262 24.262 24.262 24.261 24.262 24.262 24.262 24.262 24.262 24.263
0.1 2.085 2.042 2.155 2.289 2.190 2.438 2.388 2.501 2.441 2.576
1 1.512 1.405 1.472 1.449 1.511 1.677 1.589 1.420 1.573 1.638
1 2 3 4 5 6 7 8 9 10


Valiadation MSE heatmap as function of learning_rate and min_samples_leaf
learning_rate
min_samples_leaf
1e-06 99.532 99.532 99.532 99.532 99.532 99.532 99.532 99.532 99.532 99.532
1e-05 99.386 99.386 99.386 99.386 99.386 99.386 99.386 99.386 99.386 99.386
0.0001 97.947 97.947 97.947 97.947 97.947 97.947 97.947 97.947 97.947 97.947
0.001 84.818 84.819 84.819 84.819 84.819 84.819 84.819 84.819 84.819 84.818
0.01 25.145 25.145 25.143 25.144 25.146 25.146 25.146 25.145 25.147 25.148
0.1 3.775 3.572 3.837 3.759 3.630 3.923 3.871 3.927 3.837 4.085
1 21.835 22.612 23.010 20.361 19.860 18.753 18.942 16.989 17.389 17.812
1 2 3 4 5 6 7 8 9 10


Train MSE heatmap as function of learning_rate and subsample
learning_rate
subsample
1e-06 99.227 99.225 99.223 99.223 99.222 99.222 99.222 99.222 99.222 99.222
1e-05 99.128 99.100 99.088 99.082 99.078 99.077 99.076 99.075 99.075 99.075
0.0001 98.141 97.868 97.745 97.690 97.650 97.633 97.626 97.622 97.619 97.617
0.001 88.872 86.443 85.419 84.907 84.610 84.461 84.404 84.373 84.354 84.339
0.01 37.989 30.553 27.157 25.452 24.907 24.262 24.018 23.837 23.733 23.486
0.1 8.541 3.910 3.065 2.426 2.305 2.042 2.128 2.491 2.263 2.839
1 3323244.28 4584.036 127.339 9.610 2.983 1.405 0.825 0.629 0.602 0.599
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0


Valiadation MSE heatmap as function of learning_rate and subsample
learning_rate
subsample
1e-06 99.537 99.534 99.533 99.532 99.532 99.532 99.532 99.532 99.531 99.531
1e-05 99.441 99.411 99.398 99.392 99.388 99.386 99.386 99.385 99.384 99.384
0.0001 98.485 98.193 98.065 98.008 97.964 97.947 97.940 97.935 97.929 97.920
0.001 89.502 86.890 85.823 85.300 84.967 84.819 84.769 84.740 84.698 84.630
0.01 39.819 31.554 28.072 26.335 25.805 25.145 24.948 24.857 24.796 24.537
0.1 13.981 6.538 5.433 4.273 4.090 3.572 3.820 4.650 3.926 4.814
1 7839595.14 7669.634 380.165 57.854 30.169 22.612 16.764 14.549 11.975 12.197
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0



As we can see from the 6 heat maps above, we get the optimal validation MSE score from the optimal parameters that we stated in the beginning of the question.

A24:

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	99.13589	99.3029
Linear	2	85.45293	91.10669
Ridge Linear	3	86.07993	89.7909
Polynomial Ridge Linear	4	76.9089	79.1425
GBM Regressor	5	2.0148	3.5724

Section 6: Testing your models

A25:

Model	Section	Train MSE	Valid MSE	Test MSE
		Cross validated		Retrained
Dummy	2	99.13589	99.3029	108.1524
Linear	2	85.45293	91.10669	93.3292
Ridge Linear	3	86.07993	89.7909	93.9205
Polynomial Ridge Linear	4	76.9089	79.1425	83.9243
GBM Regressor	5	2.0148	3.5724	5.1886

The Gradient Boosting model performed the best on the training set, which was quite predictable since its train and validation MSE's were much smaller than those of the other models.
The target variable 'contamination_level' does not behave linearly with all the features and therefore is not separable. All the other models (except the dummy and GBM) are trying to find a linear plain that will separate our data which is why they are not performing well. Whereas the gradient boosting model is nonlinear and more flexible due to the fact it is using decision trees, its train, validation, and test MSE are all much lower than the others thanks to those properties.
We can see that in all of the models, except gradient boosting, we have very high train, validation and test MSE, which can be seen as underfitting of the data caused by an unfit hypothesis class chosen to capture the connection between the features and the target variable as explained above.
All the models above have higher test than train MSE but we don't think the difference is high enough to conclude that they suffer from overfitting.

Additional insight is that in Q9 we were told to find the features with the top 5 largest coefficients and as we explained in A11 the magnitude of each coefficient can be seen as how correlated the corresponding feature with our target variable is. In section 5 we are using only a subset of the features and in that subset some of the features are the ones listed as the top 5 most correlated features with 'contamination_level'. This might also be able to explain why the gradient boosting model performed so much better than all the other models that tried to capture the relationships of all the features instead of using only a subset of the most expressive and informative features.