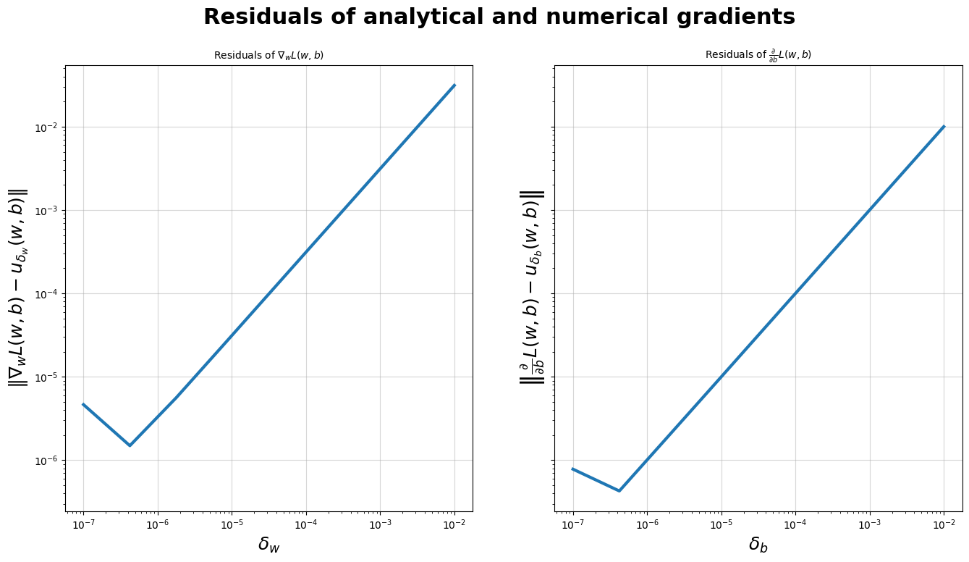
Itay Lasch & Yonatan Battat

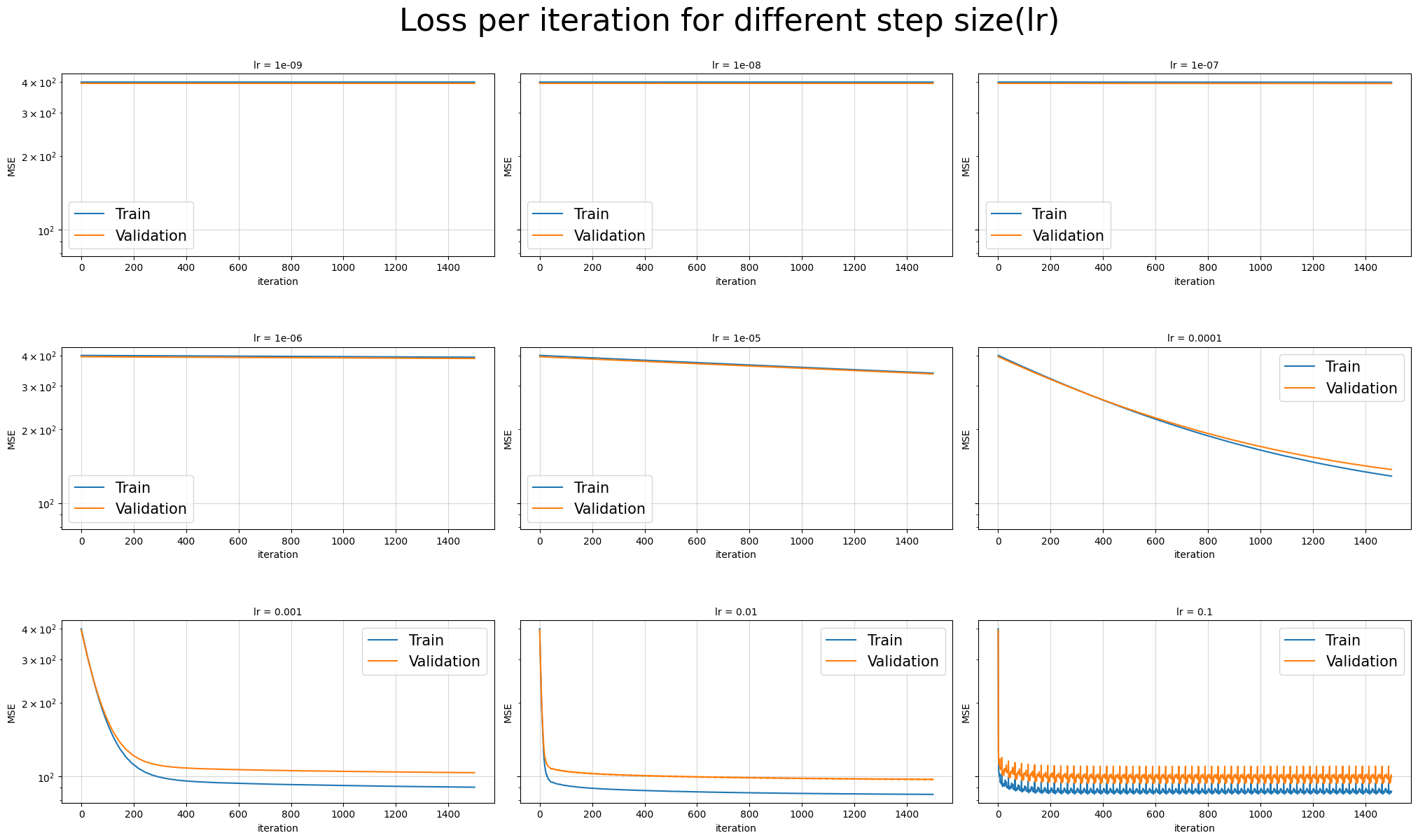
INTROUCTION TO MACHINE LEARNING - 236756

MAJOR HW4

Section 1: Linear regression implementation

A1:  
The MSE loss function for our regressor:

 A2:

A3:

As we can see from the graphs above, for 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 barely change between iterations, since our number of iterations is limited we do not converge to ideal results for these values. Moreover for we tested the loss per iteration with 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 values there are numerical errors that convert the steps to zeros.  
For values higher than including we can see that MSE is starting to decrease which means there is an improvement in loss per iteration. The higher in particular , the decrease in loss is bigger and it converges faster to its minimum value.  
We can see for 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 size in our opinion is . As we explained it converges to minimum value very quick and although we saw that for we get even a smaller train and validation loss, we prefer since unlike it converges to a single minimum value and the difference in losses between the two is quite small.

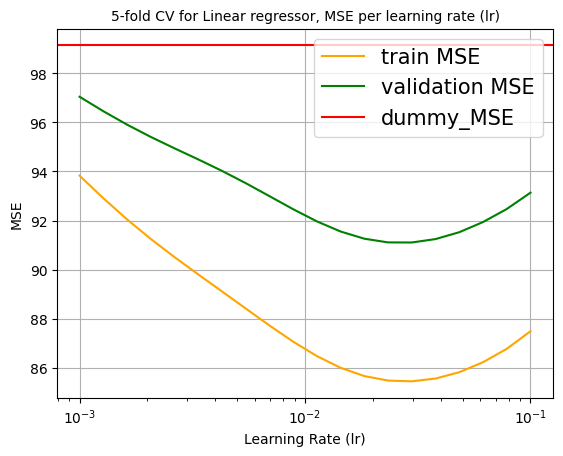
We don’t think it makes sense to increase the number of iterations, we can see that as we approach the minimum value the slope of the validation MSE is getting smaller and smaller. That happens because as we get closer to the minimum value the gradient gets smaller as well, so each step is smaller than 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:



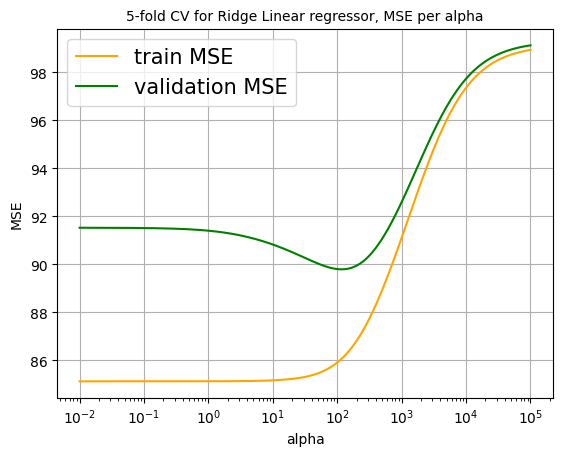
The optimal is ,which results with validation MSE of

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

A6:

Section 3: Ridge linear regression

A7:



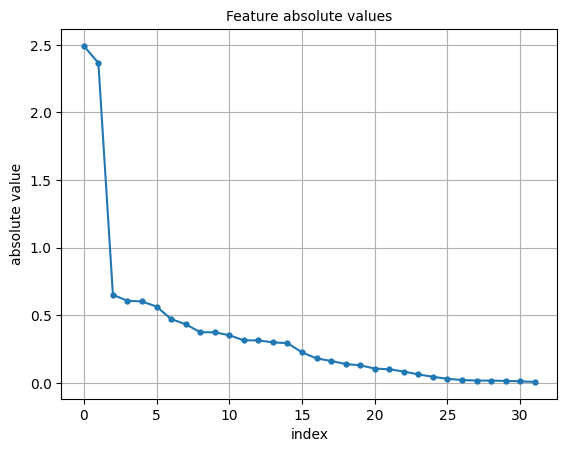
The optimal is , which results with validation MSE of

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 label, 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 be poor and might miss the optimal solution. There are a few problems that led us to this answer.  
First, as we learned in the tutorial:   
.  
In words, it means that we put constraints on the norm of which can be seen as constraints on the coefficients of the features of sample   
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 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 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 label. 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.

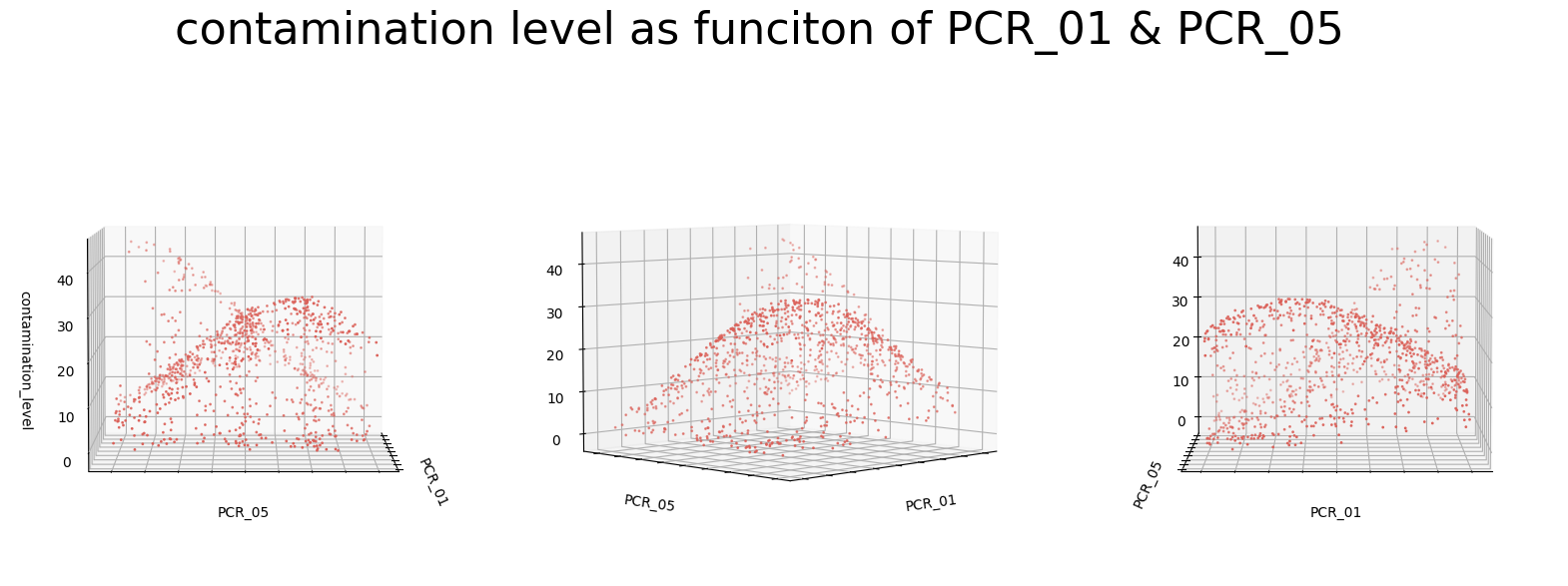
A13:  
If we had used Lasso regressor instead of Ridge I would expect most of the coefficients to zero out. As we saw in tutorial 9, Lasso regressor uses norm, and so our possible plain space is different than the one of Ridge regressor. To understand more visually we can look at the plot that shows the ERM loss as function of on a dataset with 2 features.

As we can see there are a few options to choose a proper that will yield us the minimum ERM loss, but since we use Ridge/Lasso regression we also emphasize on the norm of . Therefore, in Lasso regression, which uses the norm, SGD will choose the in our possible option space that has the smallest norm of them all. In order to achieve that the SGD will yield a 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:

A picture containing text, line, diagram, plot

Description automatically generated

Section 4: Polynomial fitting (visualization)

A14:

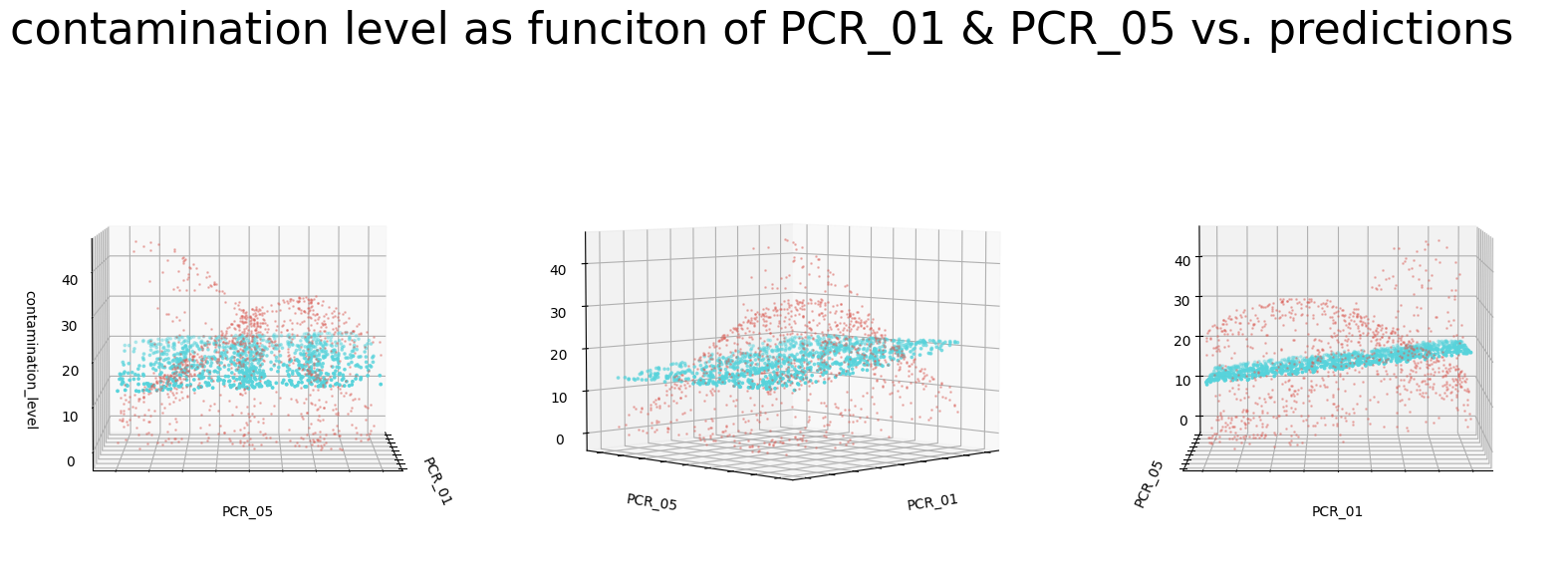
As we can see from the graphs above, it seems that the connection between ‘contamination\_level' and the features ‘PCR\_01’, ‘PCR\_05’ cannot be expressed using a straight line and therefore linear regression won’t work on the data as it currently is. It seems that the dots are scattered in some paraboloid shape so linear regression with polonomial feature mapping might work for us.

A15:

A picture containing text, line, plot, diagram

Description automatically generated

The optimal is which results with validation MSE of

A16:

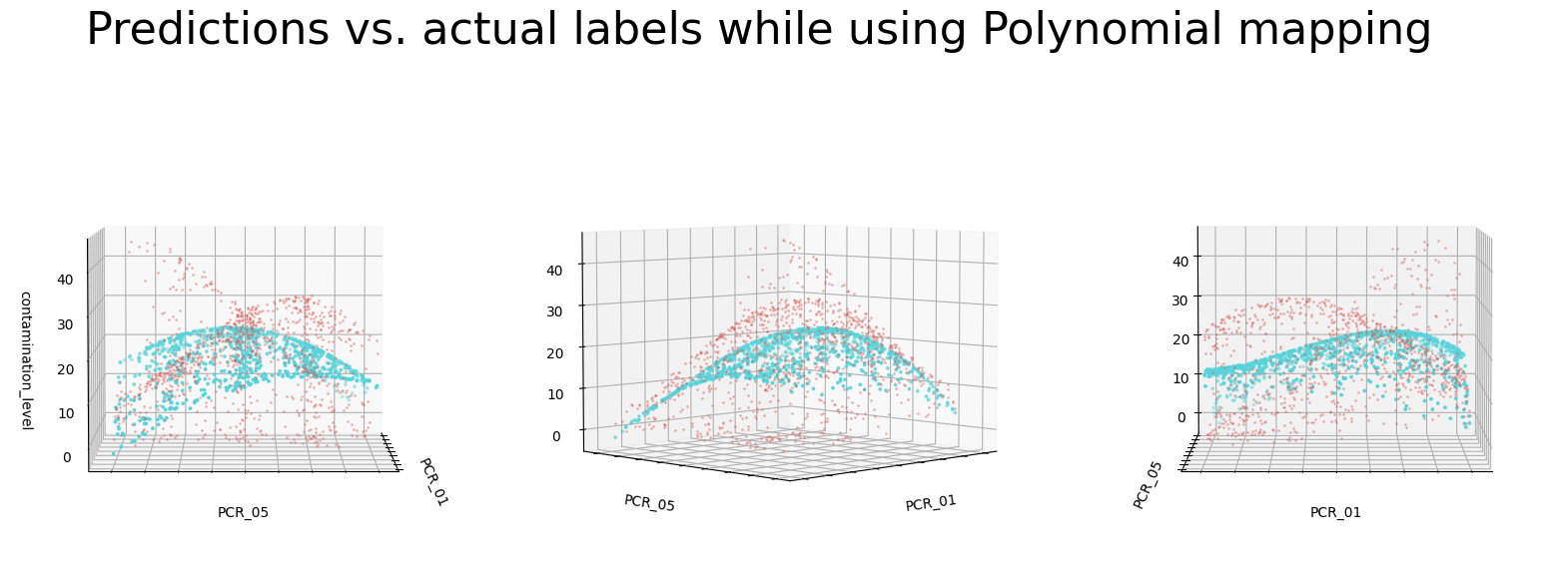
A17:  
It is important to normalize the features once more after the polynomial feature mapping for the same reason explained in (A12). We start with the 2 features ‘PCR\_01’,’PCR\_05’ let’s mark them as appropriately.  
Now after the degree polynomial feature mapping the features are: .   
Since we 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:

A picture containing text, line, diagram, plot

Description automatically generated

The optimal is which results with validation MSE of .

A19:

A20:  
There are a few different aspects to discuss when comparing the two different models:  
First, as we can see after using the degree polynomial feature mapping our validation MSE was lower than the model on the original data ( compared to ), 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 labels. 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 labels 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)  
  
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 labels. The squared error loss function penalizes large deviations from the target labels 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 labels. 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 we would want to use the squared error function to neglect those residuals and for residuals above this 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 –

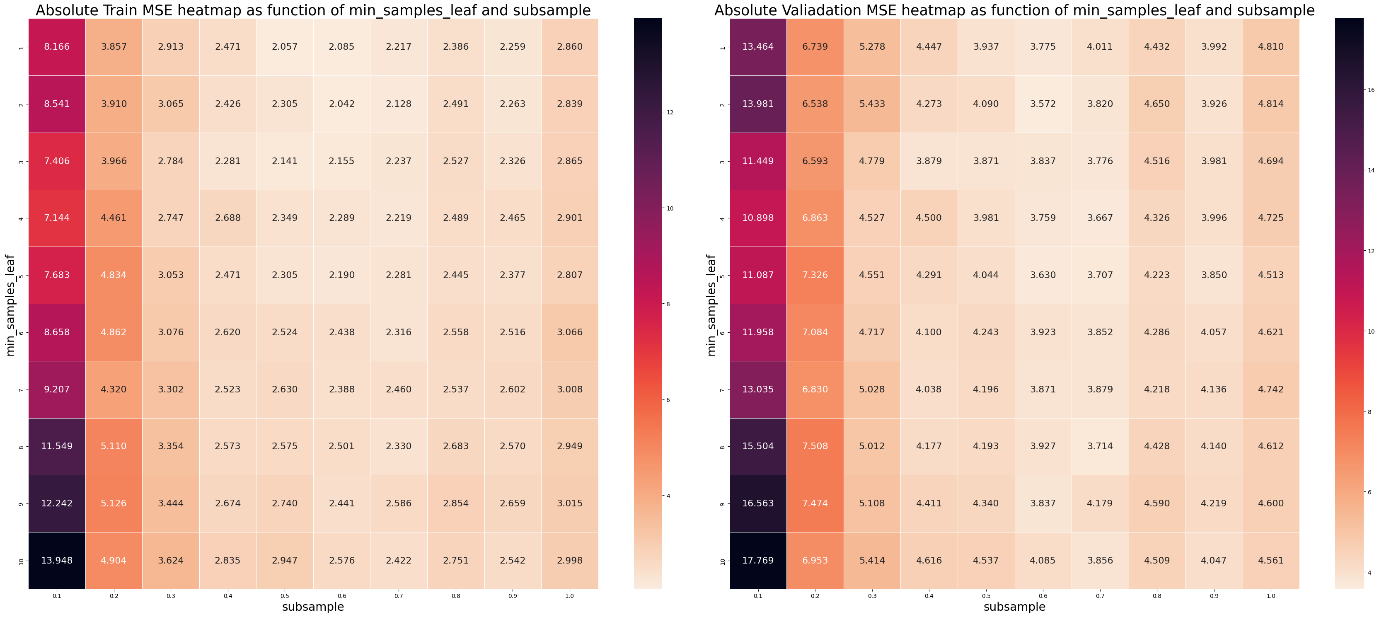
A22:  
Let’s see the similarities and differences between Random Forests and Gradient Boosted Trees:  
Similarities:

* Both algorithms use assembling to put together information from different small depth-bounded decision trees and make predictions with it.
* Both algorithms 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 label value based on the labels decided by each independent tree (majority in classification or mean in regression) whereas in Gradient Boosted Trees the labels predicted by each tree in weighted and the total sum of the weighted labels 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:



A picture containing text, screenshot, rectangle, parallel

Description automatically generated

A picture containing text, screenshot, number, font

Description automatically generated

A picture containing text, screenshot, rectangle

Description automatically generated

A picture containing text, screenshot

Description automatically generated

A picture containing screenshot, text, rectangle, design

Description automatically generated