236766 WET HW3- REPORT-208081588, 209462415

Q1:

In your report, derive the analytical partial derivative $\frac{\partial}{\partial b}L(\underline{\boldsymbol{w}}, b) \in \mathbb{R}$.

A1:

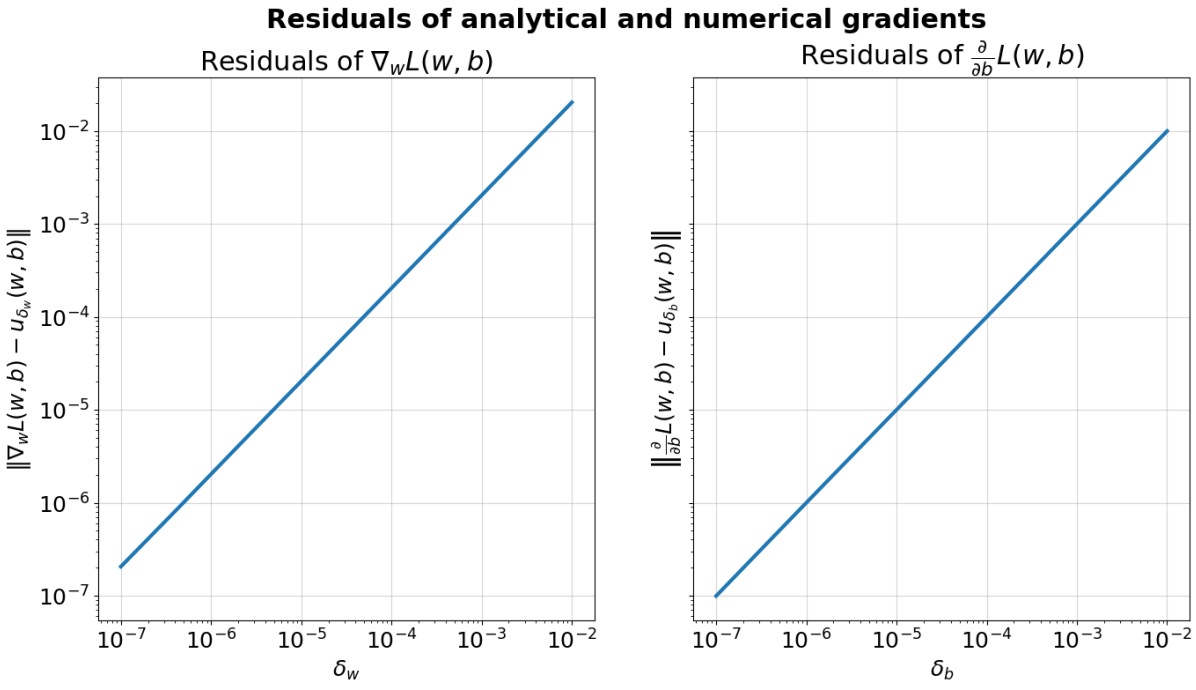
$$L(w,b) = \frac{1}{m}\sum_{i=1}^{m}(w^T x_i + b - y_i)^2 \Rightarrow \frac{\partial}{\partial b}L(w,b) = \frac{1}{m}\sum_{i=1}^{m} 2(w^T x_i + b - y_i)$$
$$= \frac{2}{m}\left\|Xw + \underline{1}_m \cdot b - y\right\|_1$$

When $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix}$.

Q2:

Using your preprocessed (and normalized) dataset and contamination_level as our target, generate a plot that compares the numerical gradients to the analytical ones.
Attach the plots to your report. No need to discuss these plots, but make sure that they make sense.

A2:

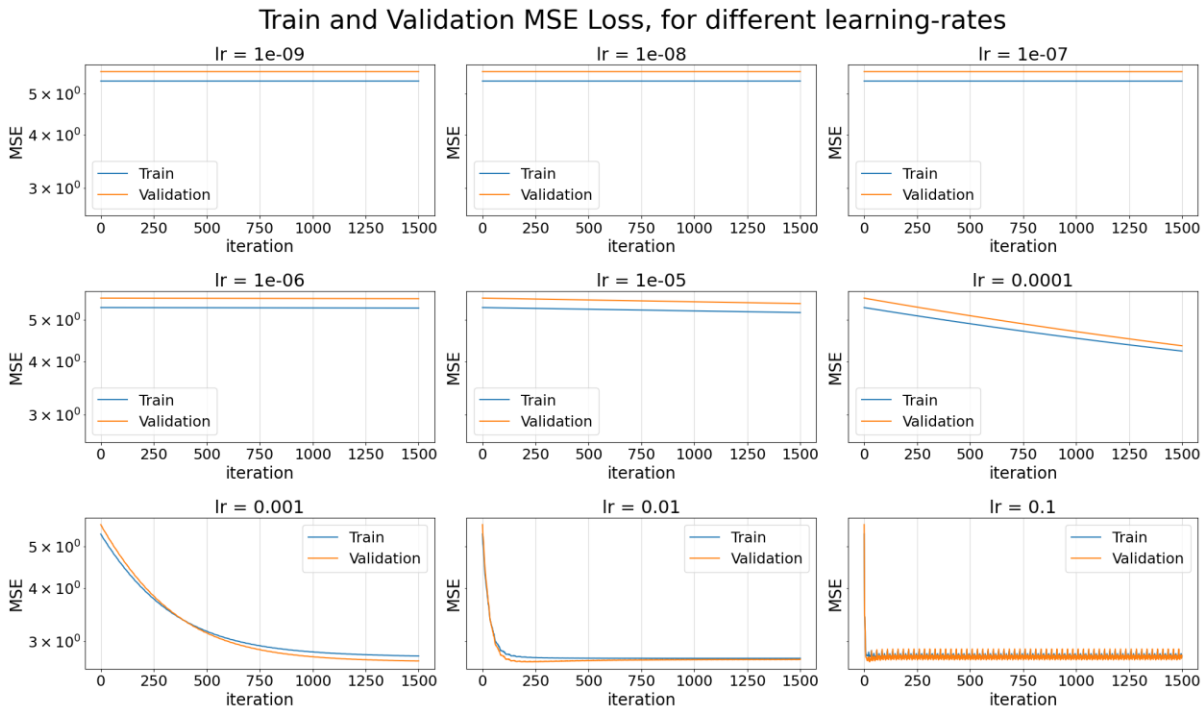


236766 WET HW3- REPORT-208081588, 209462415

Q3:

We now want to evaluate the effects of different learning rates on the learning procedure. Run the following command that plots a graph of the training and validation losses as a function of the iteration number for different learning rates.
Attach the plots to your report, briefly discuss the results and justify the demonstrated behaviors. Which learning rate is "optimal" (briefly explain)? For the best learning rate, does it make sense to increase the number of gradient steps (instead of the default 1500 steps)? Explain.

A3:



We can see that when the learning rate is 0.1, both train and validation losses fluctuate even after many GD steps, meaning we may get sub-optimal parameters (w and b).
When the learning rate is 0.01 we can see the validation MSE increasing slowly – this is likely due to overfitting (the train MSE still decreases).
When the learning rate is ≤ 0.0001, the loss doesn't decay after 1500 steps because the changes to w and b are insignificant.
The "optimal" learning rate then is 0.001, because the validation MSE converges to a minimum and does so without fluctuating or increasing.
It does not make sense to increase the number of steps, because we can see in the plot that 1500 steps are enough to converge, and more steps could lead to overfitting as it keeps optimizing for the training data.

236766 WET HW3- REPORT-208081588, 209462415

Q4:

Create a DummyRegressor. Evaluate its performance using cross-validation. In your report, fill in the cross-validated errors of the regressor.

A4:

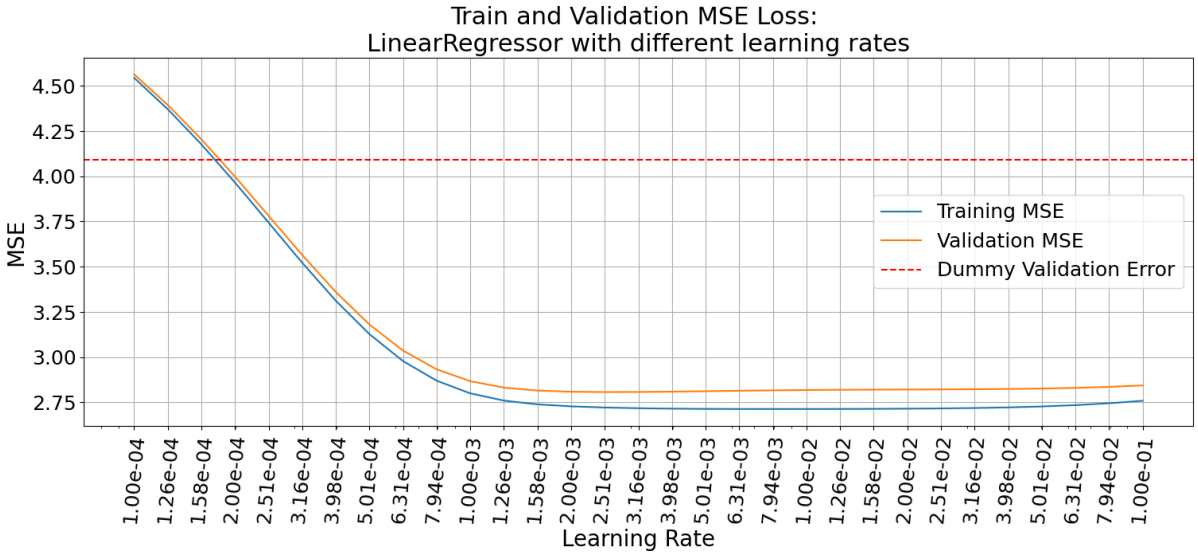
Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	4.08	4.09

Q5:

Create an instance of your custom LinearRegressor and evaluate its performance using cross-validation, remember to tune the LR!

Report the appropriate plots and values detailed above in “the repeated tuning process”. Fill in the cross-validated errors of the regressor yielded by the optimal hyperparameter.

A5:



From the hyperparameter tuning we found the optimal learning rate is 2×10^{-3}, with validation MSE score 2.81.

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	4.08	4.09
Linear	2	2.73	2.81

236766 WET HW3- REPORT-208081588, 209462415

Q6:

Had we chosen not to normalize features beforehand, would the training performance of these two models have changed (assume there are no numerical errors)? Explain.

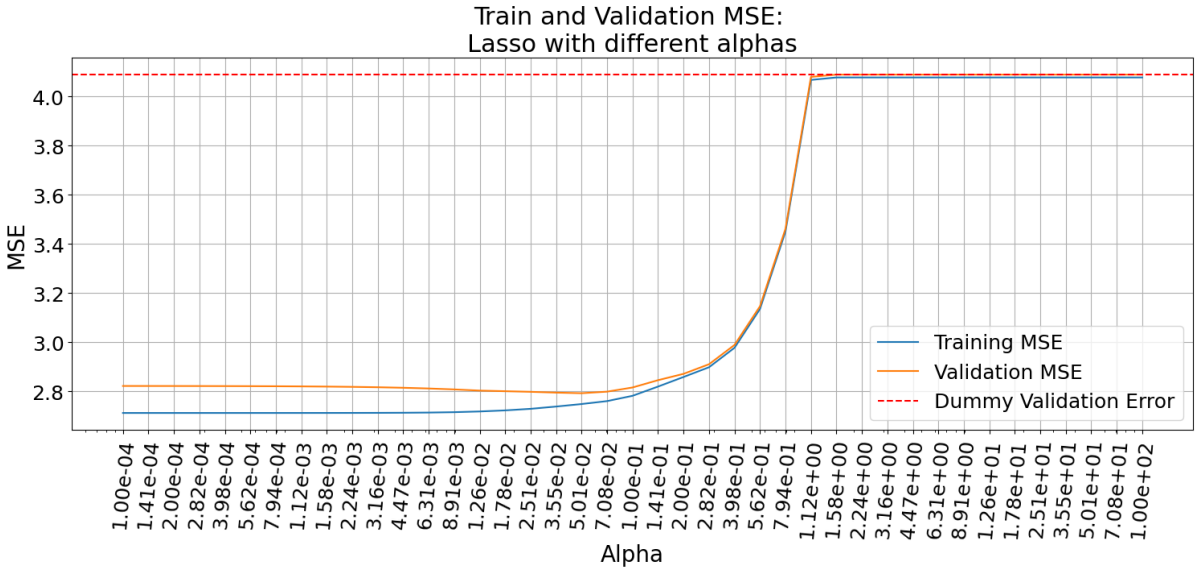
A6:

No. The dummy model's performance wouldn't change because its predictions are based only on the labels (specifically, their mean) and are independent of the features themselves.
Assuming no numerical errors, the Linear Regressor's performance wouldn't change either because both normalization methods we used are linear transformations.
Therefore, we can adjust w, b according to each feature's scaling to get a linear regressor with the same performance.

Q7:

Tune the regularization strength of the regressor. Follow the repeated tuning process described earlier. Remember to attach the required plot and specify the optimal strength with its validation error. Remember that plot should have suitable titles, axis labels, grid lines, etc.

A7:



From the hyperparameter tuning we found the regularization strength is 5×10^{-2}, with validation MSE score 2.79.

Q8:

Fill in the cross-validated errors of the regressor yielded by the optimal hyperparameter.

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	4.08	4.09
Linear	2	2.73	2.81
Lasso Linear	3	2.75	2.79

236766 WET HW3- REPORT-208081588, 209462415

Q9:

Specify the 5 features having the 5 largest coefficients (in absolute value) in the resulting regressor, from the largest to the smallest (among these 5).

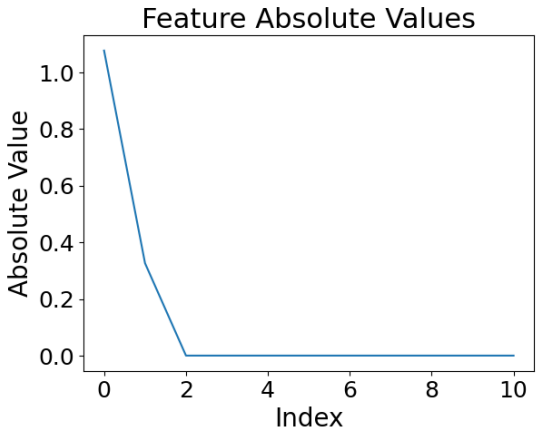
A9:

We have only 2 non-zero coefficients:
"Happiness_score" with absolute value: 1.07670152.
"PCR_03" with absolute value: 0.32620593.

Q10:

Sort and plot the absolute values of the learned coefficients. The x-axis should be the index of the parameter from largest to smallest.

A10:



Q11:

Why is the magnitude of the coefficients interesting? Explain briefly.

A11:

We can see that most coefficients are 0, and the only two non-zero coefficients have values ~0.3 and ~1 which are relatively far from each other and relatively large.

This can be explained by the LASSO algorithm which uses l_1 norm to regularize the coefficient vector – which prefers fewer non-zero coefficients rather than many small-valued coefficients.

236766 WET HW3- REPORT-208081588, 209462415

Q12:

Had we chosen not to normalize features beforehand, would the training performance of the Lasso model have changed (assume there are no numerical errors)? Explain.

A12:

Yes. Unlike the previous models, LASSO also penalizes coefficients with large absolute values. Had we not chosen to normalize the features, we could have ended up with features that have different scales and therefore coefficients with varying magnitudes. Since the regularization penalty applies equally to all coefficients, this can result in over-penalizing and under-penalizing of some features based only on their scale which may hurt the performance of the model. Normalization helps us avoid this problem.

Q13:

How would you expect the coefficients of the trained model to change, should we have used the Ridge regressor instead? Explain.

A13:

Had we used the Ridge regressor instead we would expect to have less non-zero coefficients but overall smaller absolute values of the coefficients. This is because Ridge penalizes based on the l_2 norm which favors more lower-absolute-valued coefficients, while l_1 which is used in LASSO prefers more zero-valued coefficients.

Q14:

Why is re-normalization important after applying a polynomial mapping?

A14:

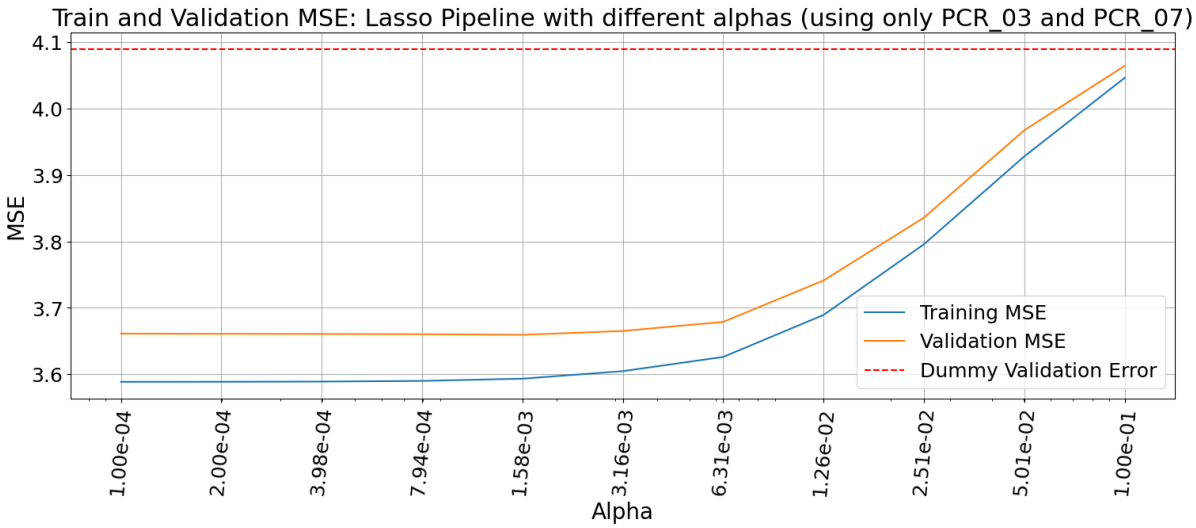
It is important to re normalize the features we got after polynomial mapping, because although the original features are normalized, the resulting features may have different scaling and ranges. For example, a feature that is a square of an original feature is now non-negative and therefore its range may now be different from a feature that is a cube of an original feature.

236766 WET HW3- REPORT-208081588, 209462415

Q15:

Tune the regularization strength of a Lasso regressor with the polynomial mapping using cross validation (k = 5 as always).

A15:



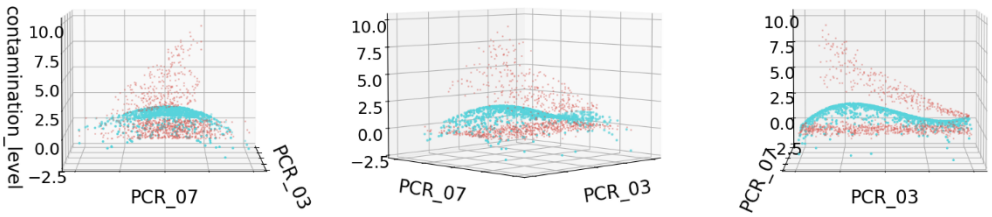
As we can see from the plot, we get the best validation MSE for alphas between 10^{-3} and 10^{-4}. For simplicity, we will choose $\alpha = 10^{-4}$. We have a validation MSE of 3.66.

Q16:

We will now visualize the polynomial model you just trained. Use the plot3d function to plot all the training samples and their true labels (as before). Pass your model predictions in a list to the predictions keyword to compare the true labels to the predicted values. Attach the plot to your report.

A16:

3dplot of contamination_level as a function of PCR_03 and PCR_07, w/predictions



236766 WET HW3- REPORT-208081588, 209462415

Q17:

Fill in the train and validation errors of the regressor yielded by the best performing hyperparameter. Remember to compute the errors using cross-validation.

A17:

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	4.08	4.09
Linear	2	2.73	2.81
Lasso Linear	3	2.75	2.79
Polynomial Lasso	4	3.59	3.66

Q18:

Tune the 'learning_rate', and 'min_samples_leaf' parameters of the GradientBoostingRegressor model using cross-validation with a grid search (see Q8 in Major HW2). For the 'loss' parameter, use 'huber' (for further details regarding the Huber loss see here). Use a similar pipeline to the one created earlier in this section. Remember to attach required heatmaps, optimal hyper-parameters, and optimal train and validation errors (use the default values for all other parameters).

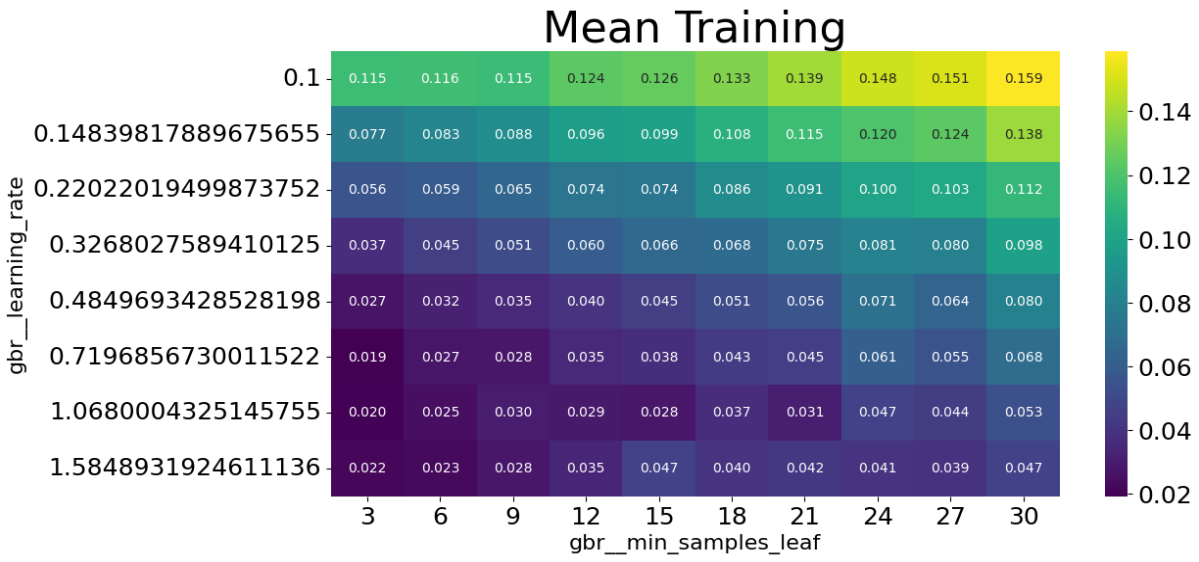
A18:

Heatmaps below.

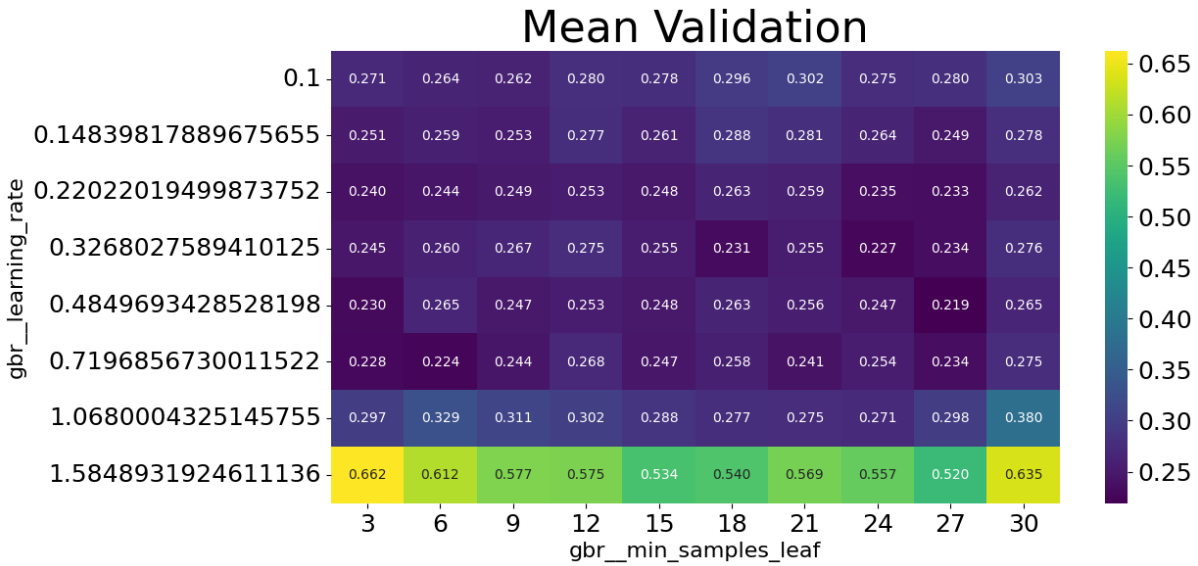
We used only happiness_score, PCR_03 and PCR_07 for this part. We chose them after viewing the coefficients of the polynomial Lasso for reference (their combinations had the largest coefficients in absolute value).

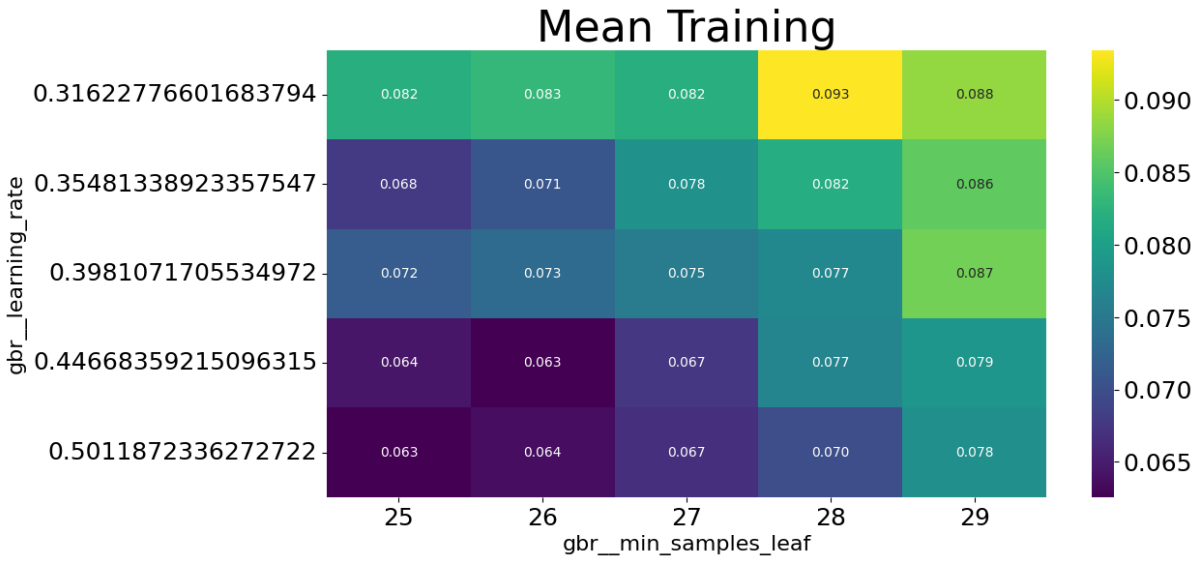
The optimal hyperparameters based on the grid search are:
Learning rate = 0.35. Min_Samples_Leaf = 26.

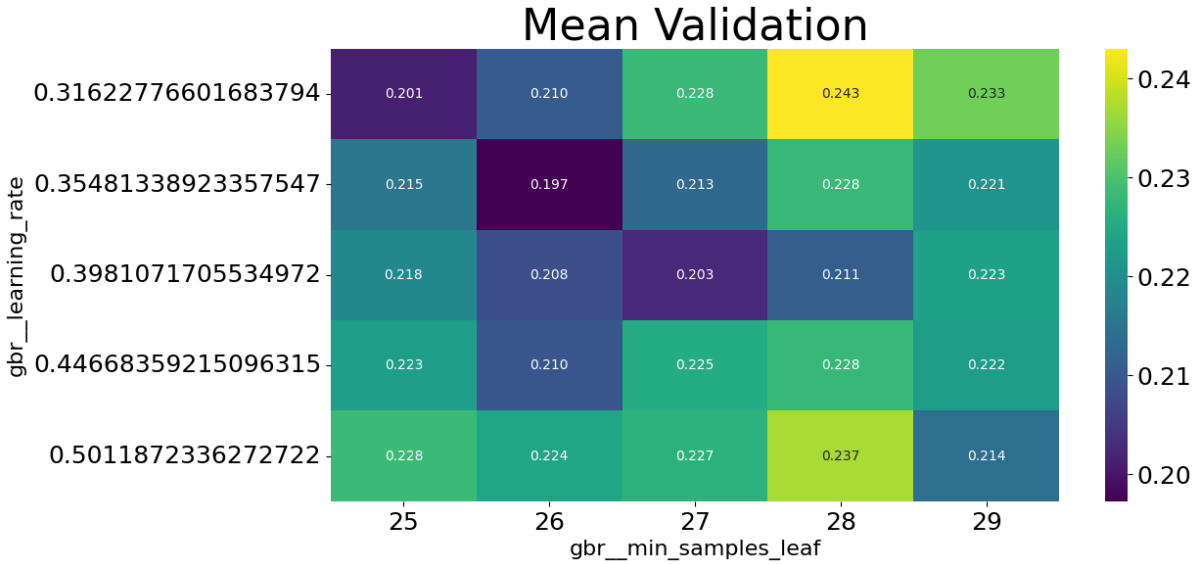
They resulted in mean training and validation MSE of 0.08 and 0.2 respectively.



236766 WET HW3- REPORT-208081588, 209462415







236766 WET HW3- REPORT-208081588, 209462415

Q19:

Fill in the train and validation errors of the regressor yielded by the best performing hyperparameter. Remember to compute the errors using cross-validation.

A19:

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	4.08	4.09
Linear	2	2.73	2.81
Lasso Linear	3	2.75	2.79
Polynomial Lasso	4	3.59	3.66
GBM Regressor	5	0.08	0.2

Q20:
Complete the entire table. Which model performed best on the test set?
Briefly discuss the results in the table (from an overfitting and underfitting perspective, or any other insightful perspective).

A20:

Model	Section	Train MSE	Valid MSE	Test MSE
		Cross validated		
Dummy	2	4.08	4.09	3.78
Linear	2	2.73	2.81	2.73
Lasso Linear	3	2.75	2.79	2.73
Polynomial Lasso	4	3.59	3.66	3.52
GBM Regressor	5	0.08	0.2	0.24

The GBM regressor performed best, achieving the lowest test MSE (0.24).
The dummy regressor was the worst, which is expected since it predicts using the mean of the target values – making the least educated guess. Overfit and underfit are irrelevant here.
The Linear and Lasso Linear models came out similar in performance, which can be explained by the fact that the Linear model achieved similar results in Train MSE – meaning it is not overfitted nor underfitted – and so the LASSO model cannot optimize much more when reducing overfit (although it did result in a sparse coefficient vector w).
The polynomial lasso came out second-to-worst, almost as bad as the dummy regressor. We can reason that the two features it was trained on (PCR_03 and PCR_07) are not enough to predict the contamination level by themselves (at least using polynomial lasso).
This can be reinforced by the GBM Regressor, in which we also included happiness_score in the training features which dramatically improved results.
Also, we can say that the Polynomial Lasso is quite underfitted since it performs badly even on the training set itself.
The GBM regressor performed best, probably because it uses gradient boosting which is a very powerful learning tool, which allows it to make better predictions.
The GBM regressor might be slightly overfitted – as suggested by the very low train MSE – but still achieves the best score regardless of that.