Homework

Course: Introduction to Machine Learning

Assignment: Major HW3

Submitters: Ofer Nissim – 312367576

Lahav Fridlander – 209403781

Date: 22/01/23

Section 1: Linear regression implementation

(Q1) In your report, derive (ιnnɔ) the analytical partial derivative $\frac{\partial}{\partial b} L(\underline{\boldsymbol{w}}, b) \in \mathbb{R}$.

The analytical partial derivative is:

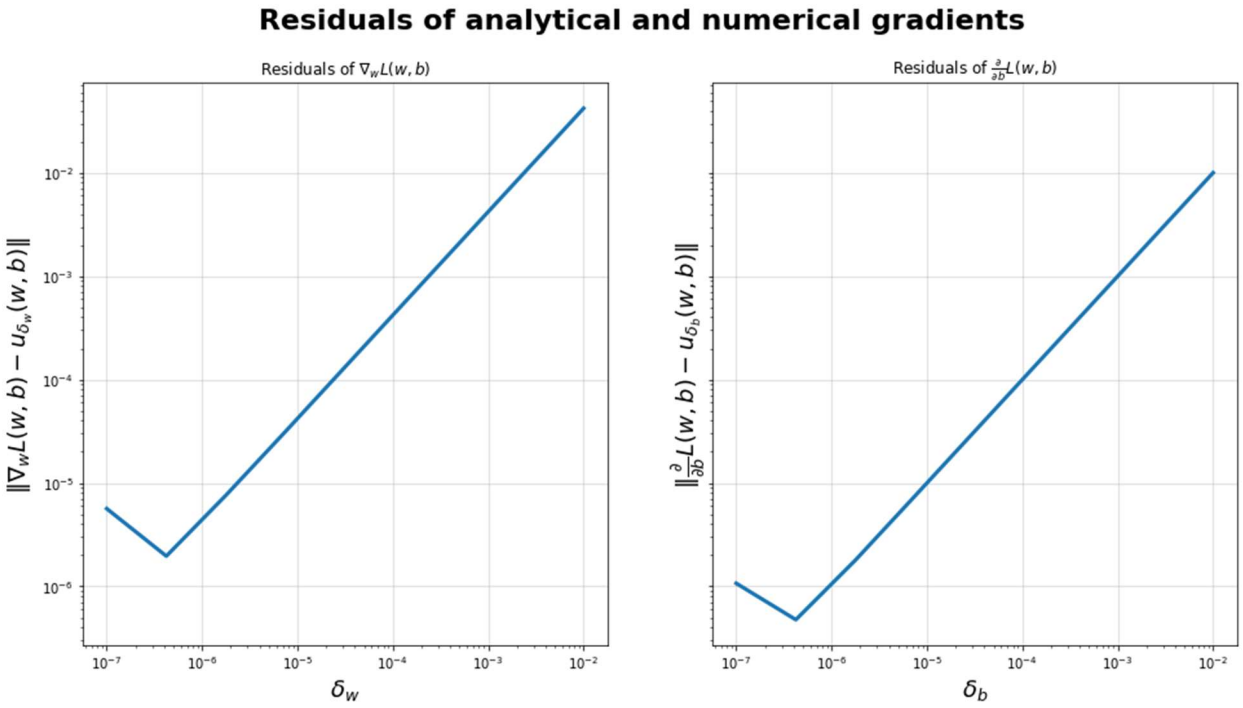
$$L(w,b) = \frac{1}{m}\sum_{i=1}^{m}(w^T x + b - y_i)^2 \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial b} L(w,b) = \frac{1}{m}\sum_{i=1}^{m} 2 \cdot (w^T x + b - y_i) =$$

$$= \frac{2}{m}\sum_{i=1}^{m}(w^T x + b - y_i)$$

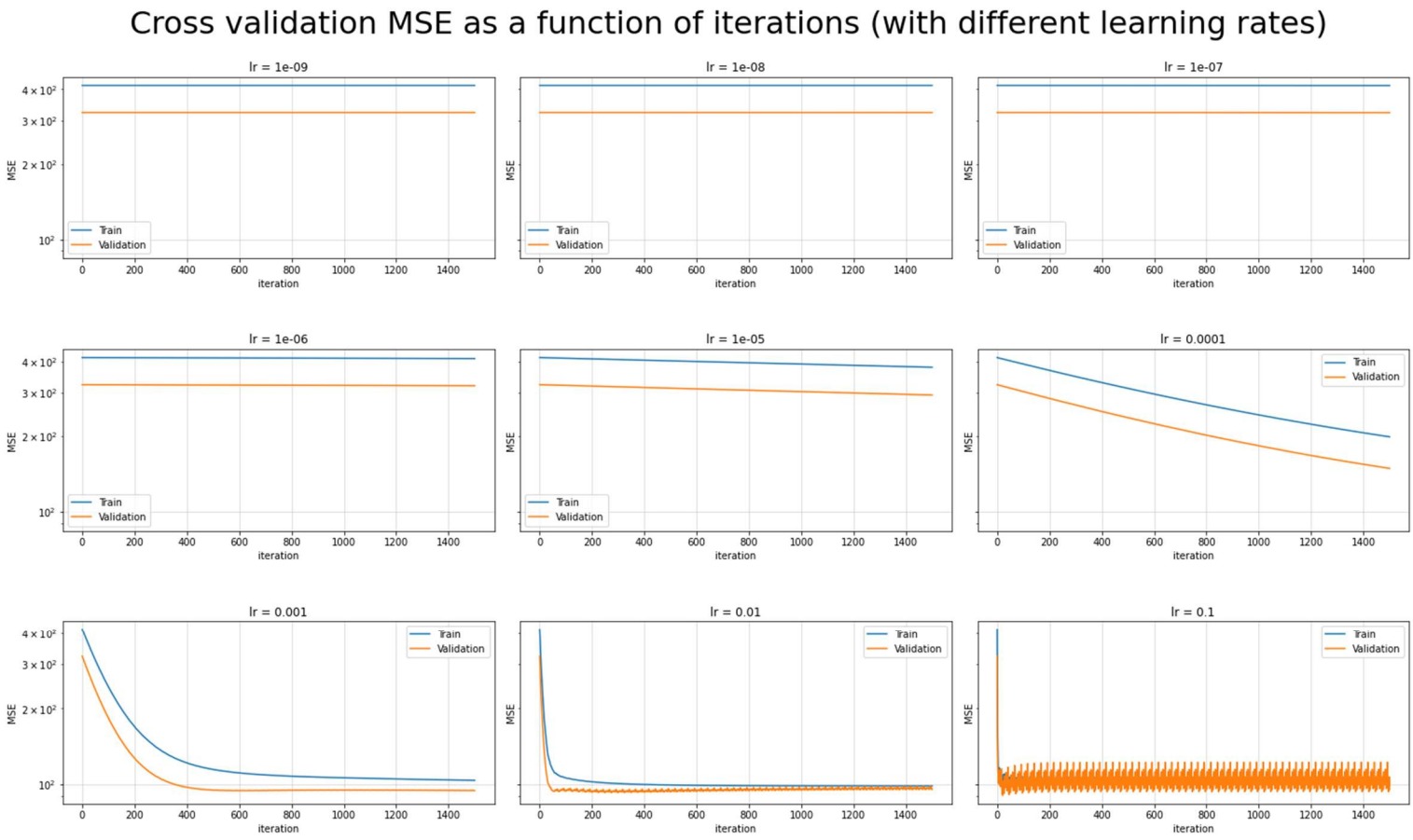
(Q2) Using your preprocessed (and normalized) dataset and `contamination_level` as our target, generate a plot that compares the numerical gradients to the analytical ones.

The comparison plot:



(Q3) 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.

The obtained plots:



The optimal learning rate: 0.01.

We see that for the first five learning rates (smaller or equal to 10^{-5}), there is no change in the MSE along the iterations, meaning that the SGD algorithm doesn't converge. Observing higher learning rates, we see the MSE becomes smaller with every iteration and goes toward convergence, where the rate of convergence grows with the learning rate. However, for the highest learning rate (0.1) we see a sharp decline to minimum area and then the MSE value oscillates around it, implying divergence. Hence, we choose 0.01 as optimal learning rate, which demonstrates stable and fast convergence to minimum in comparison to all other rates.

For the optimal learning rate, it makes sense to increase the number of gradient steps in order to gain a bit lower MSE, which is closer to the exact convergence value. We'll mention that the MSE converges and stabilizes very early (around iteration 200), such that more iterations would change its value only by small factor – a consideration to keep in mind while facing limited resources (sometimes an estimated sub-optimal MSE would be enough for our needs, at the expense of further iterations).

Section 2: Evaluation and Baseline

(Q4) Create a DummyRegressor. Evaluate its performance using `cross-validation`. In your report, fill in the cross-validated errors of the regressor.

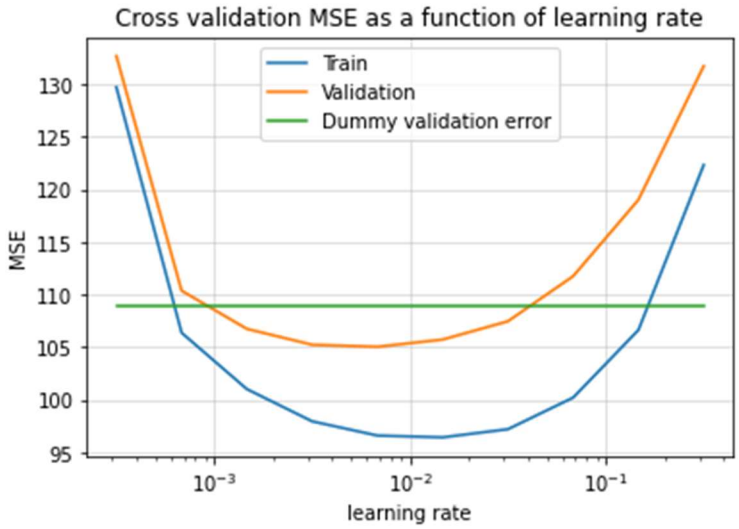
Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	108.841	108.889

(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.

We obtained the following plots:



Optimal learning rate: 0.0077

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	108.841	108.889
Linear	2	96.501	105.102

(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.

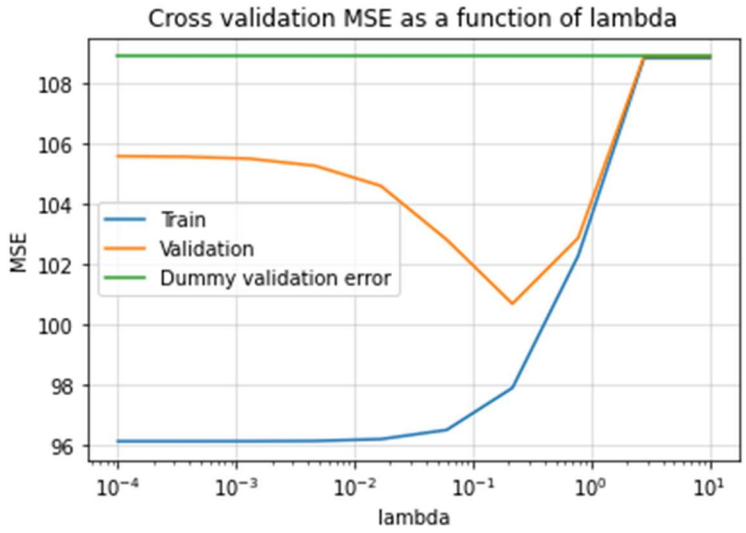
The dummy regressor's performance wouldn't be affected by changes in normalization of features, as it always predicts the average contamination level – which means it depends solely on the target labels.

The performance of the linear regressor might change if we don't normalize the features beforehand, since larger scale features would have a greater impact on the loss (the least squares). Since we only have one learning rate for all features, the scale of the learning rate wouldn't fit all the different features sizes. If this learning rate is not small enough, the size of the SGD's step toward a specific feature would be affected by the feature's scale. This could delay and even prevent the converge of the algorithm to appropriate w vector and b value, as needed.

Section 3: Linear regression with Lasso

(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.

The obtained cross validation MSE plot:



Optimal lambda: 0.215

Training error of optimal lambda: 97.904

Validation error of optimal lambda: 100.693

(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	108.841	108.889
Linear	2	96.501	105.102
Lasso Linear	3	97.904	100.693

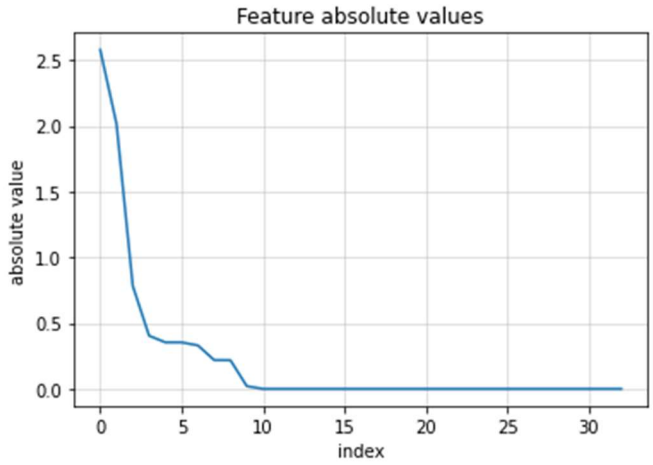
(Q9) Specify the 5 features having the 5 largest coefficients (in absolute value) in the resulting regressor, from the largest to the smallest (among these 5).

The 5 features with the largest coefficients are: (from the largest to the smallest)

Place	Feature name	Coefficient absolute value
1	PCR_01	2.579
2	sugar_levels	2.015
3	pcr_month	0.782
4	low_appetite	0.404
5	num_of_siblings	0.353

(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.

The obtained features absolute values plot:



(Q11) Why is the magnitude of the coefficients interesting? Explain briefly.

The magnitude of the coefficient tells us how important the feature is for data prediction. This can be seen in the loss equation – the weight of each feature is multiplied by the feature samples. Given the feature regularization (explained more thoroughly in the next question), larger weights mean that the feature has greater impact in target predication.

The regularization hyperparameter prevents the coefficient's magnitude from growing too much, i.e., it reduces the model's complexity and helps it avoid overfitting. As seen in the plot above, there are numerous coefficients with higher magnitudes than the rest – that indicates their higher importance for the prediction (their magnitudes are not "too big" as in overfitting situation, since they were regularized before).

(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.

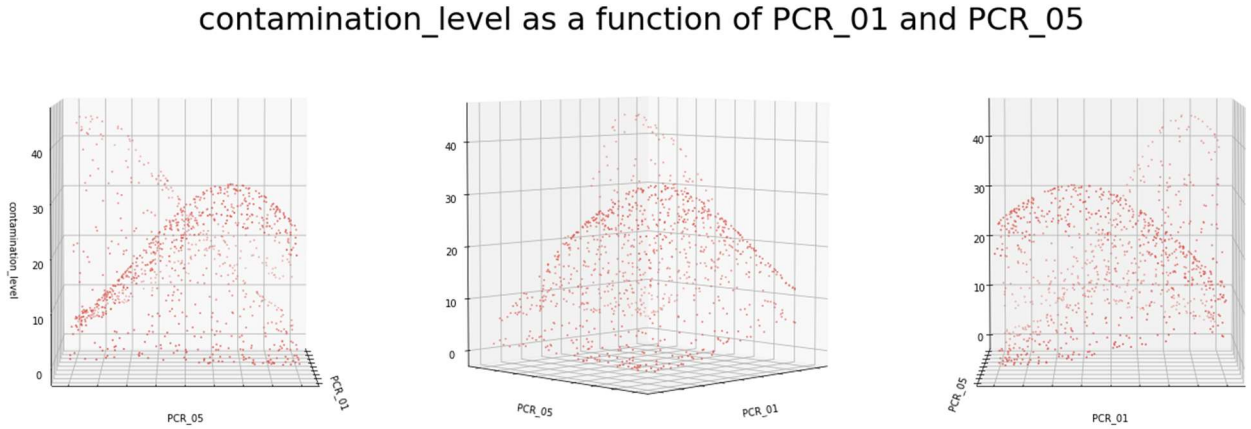
As seen in the loss equation, the weights we calculate are computed in a 1-norm. Meaning that the loss is directly affected by the size of each weight. Therefore, larger feature scales can be used by smaller weights, while also maintaining a small norm. Meaning that the lasso model will prefer features with greater scale. Note that low enough regularization factors would minimize the significance of the regularization term in the loss equation. Therefore, for small enough regularization factors, we will only care about bias, and not complexity. Hence, a larger feature scale would not impact the performance.

In the mentioned case, the training performance of the Lasso model might change, because without normalization there could be some features that have a lower value scale than others. That will cause the Lasso loss to give greater significance to the features with higher scales, in consideration with their corresponding weights. This changes the importance of features determined by the Lasso model and yields a different optimization problem with a different solution. The new solution might have lower accuracy, due to arbitrary feature scales, which causes arbitrary significance.

Section 4: Polynomial fitting (visualization)

(Q13) Attach the 3-d plot to your report. What can we understand from this visualization? How should this affect our choice of model to regress `contamination_level`?

The obtained 3-D plot:

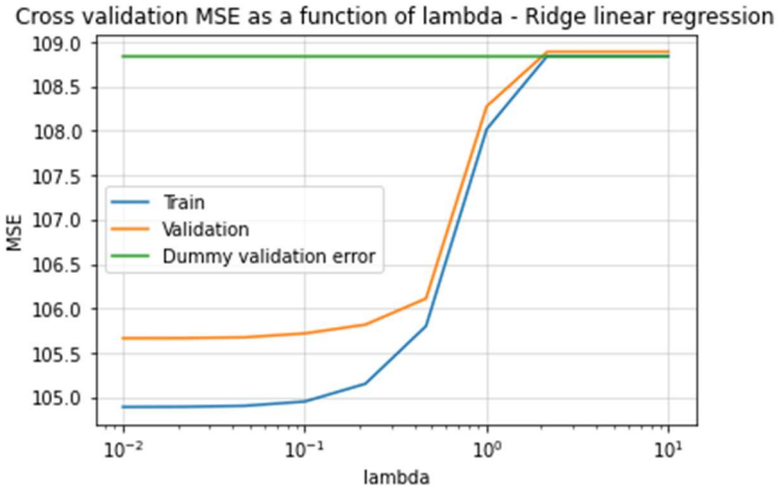


In this visualization we see a graph with a paraboloid shape (quadratic surface), which can be represented as a 3-dimensional quadratic equation. Due to this non-linear shape, using a linear regressor (which is visualized as a 3-dimensional plane) to regress contamination level would be inappropriate for our data and lead to a higher MSE. If we wish to choose a good model to regress contamination level, we'll need to perform a polynomial feature mapping.

(Q14) We will once again create a baseline, this time our baseline will consist solely of linear Lasso regression (using the two features only), remember again to make sure your models are non-homogeneous (`fit_intercept=True`).

Tune for regularization strength as before. Remember to attach required plots, optimal strengths, and optimal validation errors (on this new target).

The obtained cross validation MSE plot:



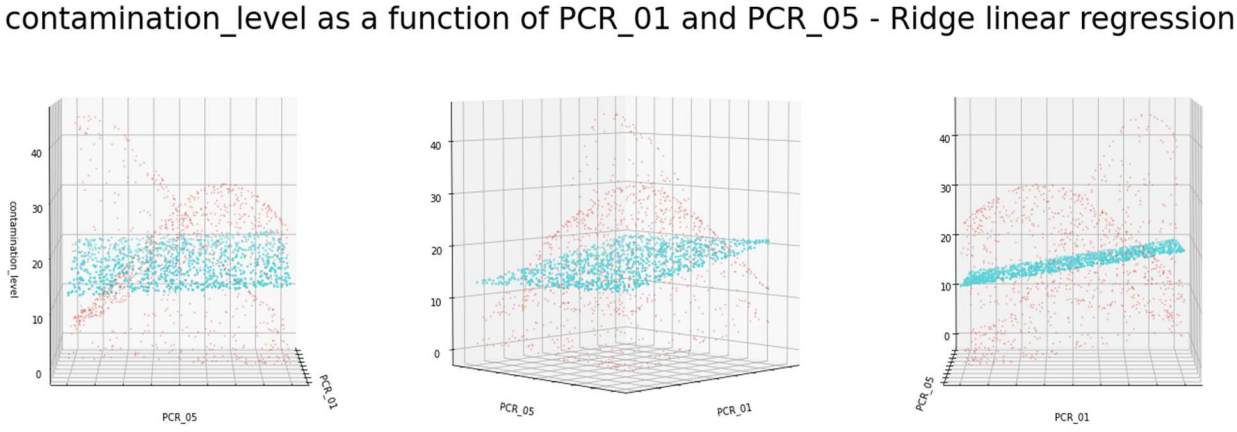
Optimal lambda: 0.01

Training error of optimal lambda: 104.889

Validation error of optimal lambda: 105.664

(Q15) We will now visualize the model you just trained. Use `plot3d` to plot all the training samples and their true labels (as before). Also pass your model's predictions in a list to the `predictions` keyword to compare the true labels to the predicted values. Attach the plot to your report.

The obtained updated 3-D plot:

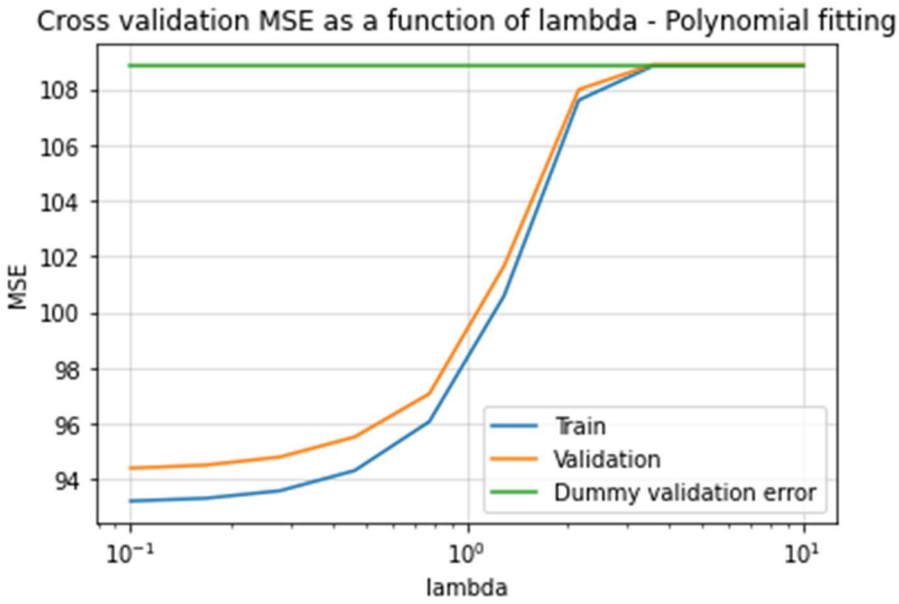


(Q16) Why is re-normalization important after applying a polynomial mapping?

Re-normalization is important after applying a polynomial mapping since the ridge linear regression is sensitive to scale changes (as explained in Q12). Thus, if we don't re-normalize the features, their different scales might affect their corresponding weights, i.e., affect the obtained solution, which might not have optimal accuracy in this situation. For example – a feature with a magnitude in the range of 0 and 1 will lose its importance after applying a polynomial mapping, due to the shrinkage effect of the power operator over the small (fraction) magnitude. Such small magnitude could lead to numerical instability as well.

(Q17) Tune the regularization strength of a Lasso regressor with the polynomial mapping using cross validation ($k = 5$ as always).

The obtained cross validation MSE plot:



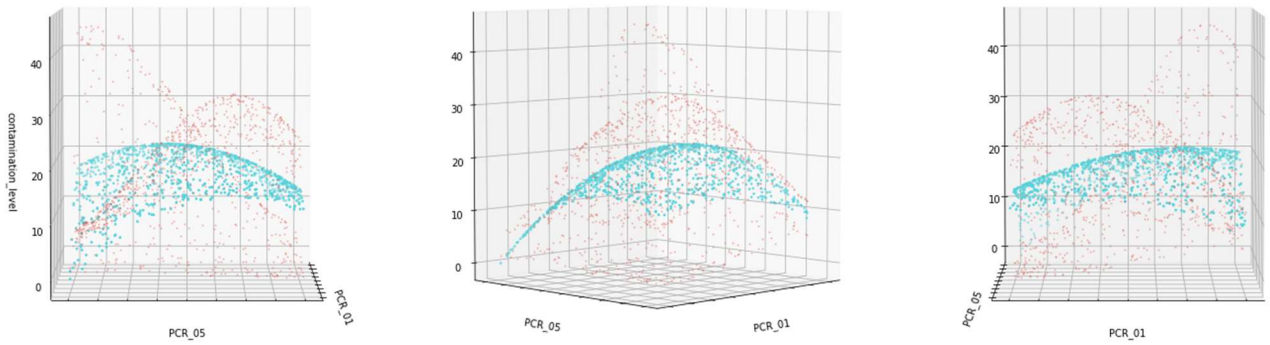
Optimal lambda: 0.1

Training error of optimal lambda: 93.181

Validation error of optimal lambda: 94.346

(Q18) We will now visualize the polynomial model you just trained. Use the new `plot3d` function to plot all the training samples and their true labels (as before).

The obtained updated 3-D plot:



(Q19) Discuss the results from this section. Compare the two models' capacities to fit the data at hand (use the visualizations, MSE, etc.). This should take 3-6 sentences

In this section we used two different models – linear lasso regressor and lasso regressor with polynomial mapping. As can be seen visually and by the MSE difference, the polynomial fitting capacity is better than linear fitting capacity and its MSE was noticeably lower – 94.4 in comparison to 105.6 of the linear. This could be explained visually, observing the shape of our data, which is paraboloid like. The polynomial mapping regressor's estimations were closer to the original points than the estimations of the linear one, due to the round shape of its surface, which is more similar to the paraboloid (rather than linear plane surface). The polynomial fitting is generally more diverse than the linear fitting, as higher polynomials can fit more complex data (seen in class), hence can suit the data better.

Section 5: RandomForest fitting of the CovidScore

(Q20) List the features you chose to transform and the transformations that you used, and why.
Tip: we advise transforming only a small subset of the features while keeping the remaining ones (i.e., instead of dropping them).

- The features we chose to transform using polynomial mapping are: 'weight' and 'PCR_02'.
- The feature we chose to transform using RBF mapping is 'num_of_siblings'.

In order to find these features, at first, we calculated the cross-validation error for each feature transformation (polynomial/RBF) separately, meaning that we transformed a single feature (ignoring the binary features, which do not typically require any mapping) using one type of mapping on each iteration. Next, we took the top 5 features with the lowest errors for each mapping method, assuming that together they might minimize the error, and calculated the cross-validation error for different subsets of these groups (using also different polynomial degrees up to 5, which is reasonable for our data). At last, we found out that the lowest error (-2.77 according to negative squared error scale) was obtained when transforming the top 3 of the polynomial group we assembled earlier and the top 2 of the RBF group. We'll mention that we also observed correlations between features throughout the process to avoid using features that don't provide any new information for the prediction. The final result was satisfying, as we also tried combinations of transformations of other features – focusing on noticeable features from our answers in the previous sections.

(Q21) Explain how (and why) we can expect the training and validation errors (each) of such random forests to change when using the RBF mapping (compared to just using the raw data).

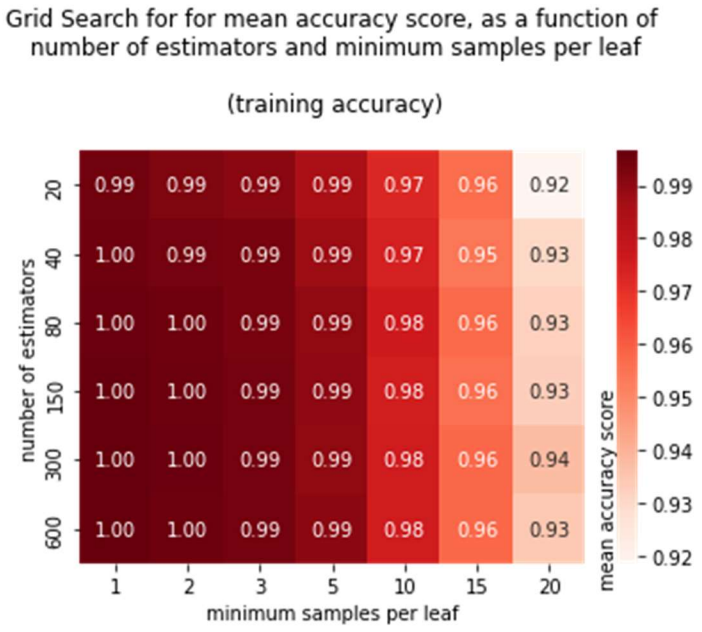
RBF mapping is used in a random forest to increase the diversity of training samples. When using this non-linear transformation over our samples, we create new samples, which contain "virtual" features (that depend on some original features), such that they represent the distribution of the data better – by spreading along the mapping space. With the RBF mapping we can better generalize and reduce overfitting, due to the diversity of the obtained new samples. Thus, we expect both training and validation errors to become smaller, compared to using the raw data only. We'll mention that the RBF sampling must be performed correctly (using right hyper-parameters, over the right features), otherwise its effect might not improve the performance or make it worse.

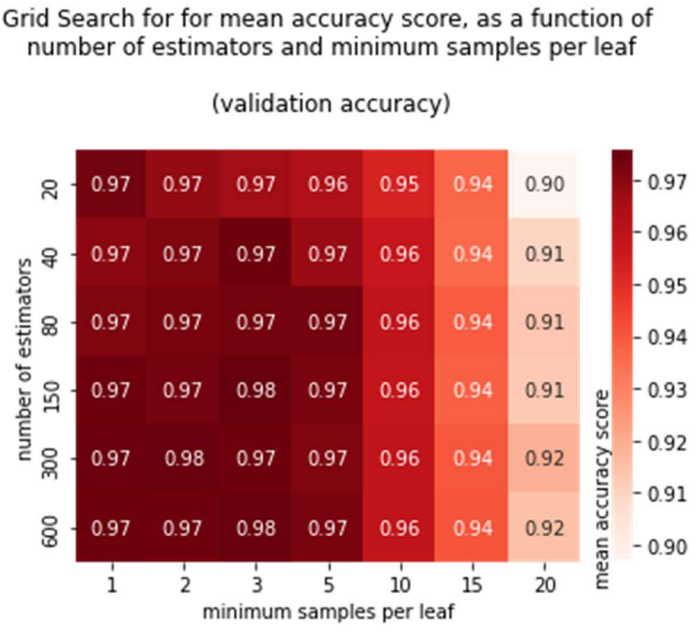
(Q22) Explain the major difference between a random forest, and the previous polynomial regressors we have used (for the same features).

The previous polynomial regressors used the samples without significance to any particular sample. The random forest, on the other hand, creates a lot of regressors, for different sample groups (meaning that some of the samples are repeated, and some aren't used for a single regressor). That way, a single regressor can have different focus on the samples. The random forest also considers only a subset of features for each regressor, creating more diversity in the feature consideration. The previous regressors may consider different features with unequal importance, but they won't give the "non-important" features the same consideration as the random forest could give. The random forest averages the regressors, to create a more balanced and diverse regressor. This process helps avoid overfitting, by combining the predictions of many decision trees. Also, we haven't used a decision tree before, however, the random forest does use decision trees.

(Q23) Tune the 'n_estimators' and 'min_samples_leaf' parameters of the RF model while using RBF mapping using cross-validation with a grid search (see Q8 in Major HW2). 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.

The obtained accuracy heatmaps: (training – on the left, validation – on the right)





Optimal stats attached:

Optimal train score: 0.996

Optimal test score: 0.976

Optimal hyper-parameters: **min_samples_leaf:** 3

n_estimators: 150

Train error: 0.637

Validation error: 2.715

(Q24) Fill in the train and validation errors of the regressor yielded by the best performing hyperparameter. Remember to compute the errors using cross-validation.

Model	Section	Train MSE	Valid MSE
		Cross validated	
Dummy	2	108.841	108.889
Linear	2	96.501	105.102
Lasso Linear	3	97.904	100.693
Polynomial	4	93.181	94.346
RF Regressor	5	0.637	2.715

(Q25) Complete the entire table

Model	Section	Train MSE	Valid MSE	Test MSE
		Cross validated		Retrained
Dummy	2	108.841	108.889	103.36
Linear	2	96.501	105.102	96.39
Lasso Linear	3	97.904	100.693	94.84
Random Forest	5	0.637	2.715	1.43

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).

As we could have expected, the dummy model performed the worst. The linear models did a bit better, but they're still limited, therefore their test MSE has the same scale as the dummy MSE. Comparing the lasso model to the formers, its result is a bit better than the linear model, as the lasso model combines regularization, thus it overfits less than the linear model.

The random forest model, which has greater complexity than the linear models, has MSE on a significantly lower scale. We notice that the random forest model didn't overfit, as the validation and test MSEs are on the same magnitude. This was expected, because the random forest model has averages numerous trees, in a way that helps the model to not overfit.