Gal Kaptsenel, Eitan Gronich 25/01/2022

Major HW3 – Final Report

(Q1)

Let us observe the MSE loss:

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

So,

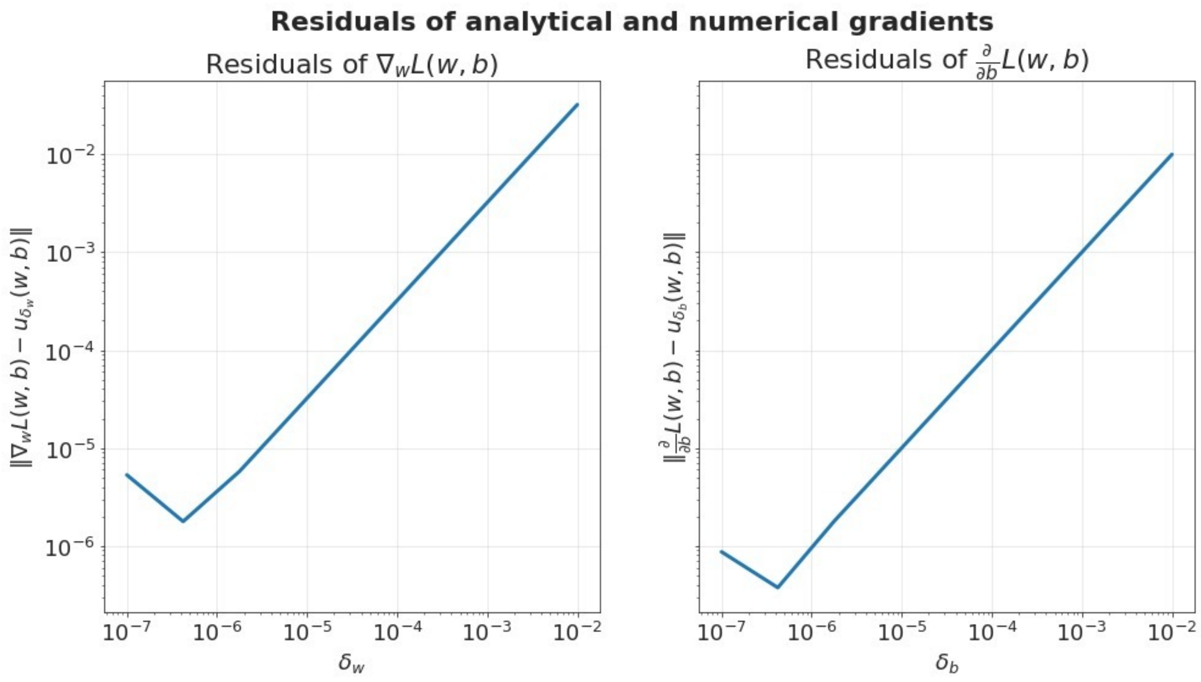
$$\frac{\partial L}{\partial b}(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b}(w^T x_i + b - y_i)^2 = \frac{1}{m}\sum_{i=1}^{m}2(w^T x_i + b - y_i)$$

$$= \frac{2}{m}\cdot \boldsymbol{1_m^T}\cdot(\boldsymbol{X}w + \boldsymbol{1_m}\cdot b - \boldsymbol{y})$$

The last transition is correct since the expression $\frac{1}{m}\sum_{i=1}^{m}2(w^T x_i + b - y_i)$ means summing the entries of the vector $\boldsymbol{X}w + \boldsymbol{1_m}\cdot b - \boldsymbol{y}$, and multiplying by the scalar $\frac{2}{m}$.

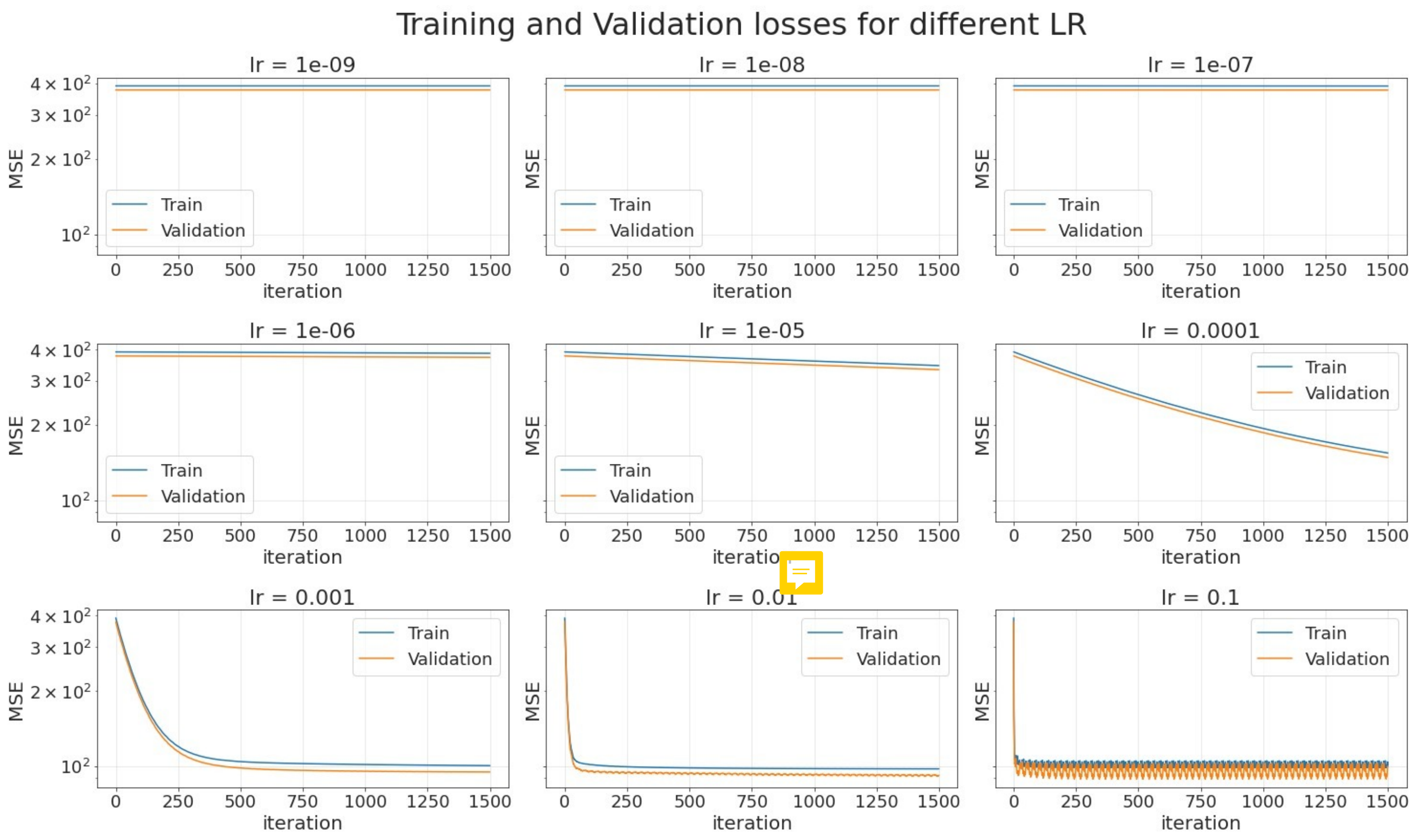
(Q2)

The following plot shows the residuals of the numerical and analytical gradient:



(Q3)

The following plot shows the training and validation losses as a function of iteration number, for different learning rates:



We can justify the behaviors for different learning rates:

- For very small learning rates (1e-09 – 1e-06), the gradient descent algorithm barely makes progress, because the step size is very small (so it would take a very long time to converge).
- For learning rates 1e-05, 1e-04, the rate of convergence is slightly better but still very slow.
- Learning rates 0.001, 0.01 seem best: we have fast convergence and high stability around the minimum. **The optimal learning rate is probably 0.01** since they reach virtually the same minimum point, but for 0.01 we reach it in much less iterations (however, it should be taken into account that for lr=0.01 we have small oscillations of the validation error around the minimum). **Increasing the number of steps is unnecessary** because clearly the algorithm has reached its minimum point and will not improve any further.
- For learning rate 0.1, the convergence is very fast, but the oscillations of the training and validation error are significant; this occurs since the step size is slightly too large and so the algorithm “overshoots” the minimum in every iteration.

Also, for all the learning rates we see that the train and validation performance is similar throughout the training process. This is a good sign, since it shows our model does not overfit. However, as the minimum MSE is still fairly high compared to the labels (~100 squared error compared to labels in the range 0-40), we deduce the models are underfit.

(Q4)

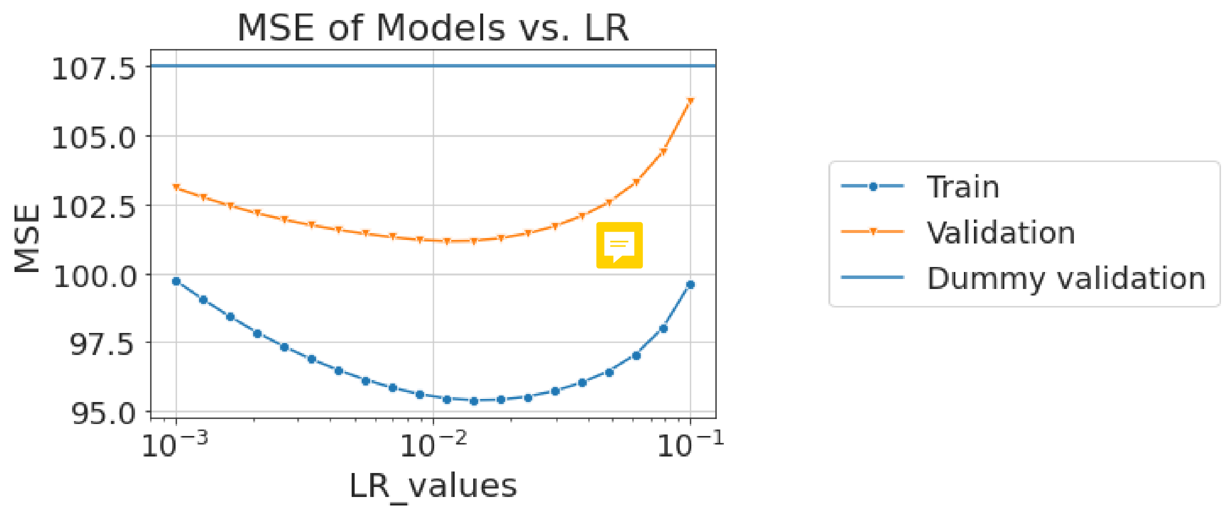
We trained a dummy model and obtained the following errors:

Model	Section	Train MSE (cross validated): Mean (std)	Valid MSE (cross validated): Mean (std)
Dummy	2	107.13 (1.7)	107.49 (6.7)

(Q5)

The range of learning rates we chose is 20 points in logspace between 10^{-3} and 10^{-1}. This choice was based on the previous question: this is roughly the range in which the algorithm converged in satisfactory time to the minimum point.

The following plot shows the mean train and validation MSE obtained from 5-fold cross validation:



As we can see, the plot behaves as expected: train and validation errors both fall and then rise since a too small learning rate slows down convergence and a too large learning rate creates instability around the minimum point. Note, there is no major difference in the trend between training and validation errors since the LR does not directly affect overfitting, only the quality of convergence of optimization. Both training and validation are better than the dummy regressor, as expected. The best validation error occurs for LR=0.0113, and the error itself is 101.15.

Model	Section	Train MSE (cross validated): Mean (std)	Valid MSE (cross validated): Mean (std)
Dummy	2	107.13 (1.7)	107.49 (6.7)
Linear	2	95.46 (0.95)	101.15 (4.29)

(Q6)

For the dummy model, not normalizing the features has no effect whatsoever, because the prediction of the dummy regressor is the mean of the labels, regardless of the features. The labels are not normalized in any case, and so the prediction of the dummy regressor does not change.

For the linear model, assuming there are no numerical errors, the optimal training error of the model should not change, but the appropriate learning rate and number of iterations for convergence could change. The reason is, normalization of a feature is a linear transformation applied to that feature (for all samples). This is true for both standardization and min-max scaling: for standardization we have $x' = \frac{x}{\sigma} - \frac{\mu}{\sigma}$ and for min-max $x' = \frac{x}{x_{max}-x_{min}} - \frac{x_{min}}{x_{max}-x_{min}}$.

Now let us observe the original data set X with d features, and parameters $w \in \mathbb{R}^d, b \in \mathbb{R}$. Suppose the normalized data set X' is obtained from X by applying a linear transformation $x_i' = a_i x_i + c_i$ to the i'th feature (for $1 \le i \le d$), with $a_i \neq 0$. We prove there are w', b' such that ${w'}^T x' + b' = w^T x + b$ for every $x \in X$ and $x' \in X'$ that comes from x by normalization.

Define a vector w' by $w_i' = \frac{1}{a_i} w_i,\ b' = b - \sum_{i=1}^{d} \frac{w_i c_i}{a_i}$.

Define $b' = b - \sum_{i=1}^{d} \frac{w_i c_i}{a_i}$. Then for all x and normalized x' we have

$$\begin{aligned} {w'}^T x' + b' &= \sum_{i=1}^{d} \frac{1}{a_i} w_i \cdot (a_i x_i + c_i) + b - \sum_{i=1}^{d} \frac{w_i c_i}{a_i} = \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \frac{w_i c_i}{a_i} + b - \sum_{i=1}^{d} \frac{w_i c_i}{a_i} \\ &= w^T x + b \end{aligned}$$

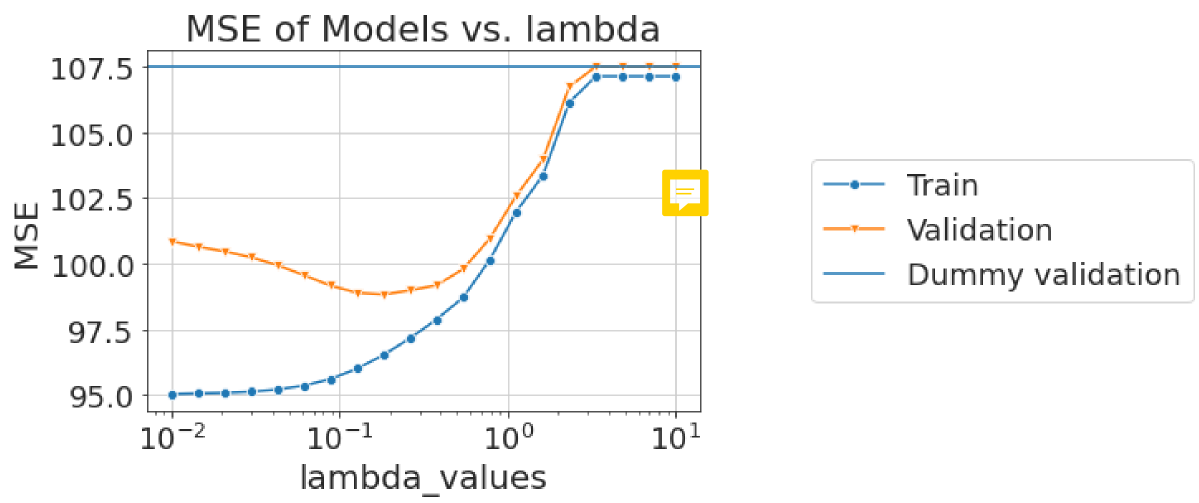
So, there is a one-to-one correspondence between parameters w, b for X and parameters w', b' for X' such that $w^T x + b = {w'}^T x' + b'$ for every matching x, x'.

Therefore, the set of possible linear models is the same before and after normalization. This means the possible values of the training loss, and in particular the optimal value, stay the same.

Since the optimization objective (squared loss) is convex, we should reach the same minimum point, given that we chose a suitable learning rate. However, the parameter space has changed, and therefore the gradients on the graph of the function have changed, which means we might need a different learning rate (if the gradients increased, we need a smaller learning rate, and vice versa), and a different number of iterations. Specifically, as we saw in lecture 8, for a non-normalized dataset we may need to choose a learning rate that is small enough for the feature with the biggest scale, which could make the convergence take longer because the chosen learning rate is too small for the scale of most of the features.

(Q7)

To tune the regularization parameter lambda, we chose (after first trying a wider range) a range of 20 points in logspace between 10^{-2} and 10^{1}.



The plot behaves as expected for a tuning plot of a regularization parameter: for low values of lambda, the model is overfit and so the training error is low and the validation error is high. For a lambda value of 0.183, we have optimal validation error of 98.82, and for high values of lambda, both training and validation errors are low since the model is underfit (in fact, the errors reach the area of the dummy model validation error).

(Q8)

Model	Section	Train MSE (cross validated): Mean (std)	Valid MSE (cross validated): Mean (std)
Dummy	2	107.13 (1.7)	107.49 (6.7)
Linear	2	95.46 (0.95)	101.15 (4.29)
Lasso Linear	3	96.52 (0.8)	98.82 (3.95)

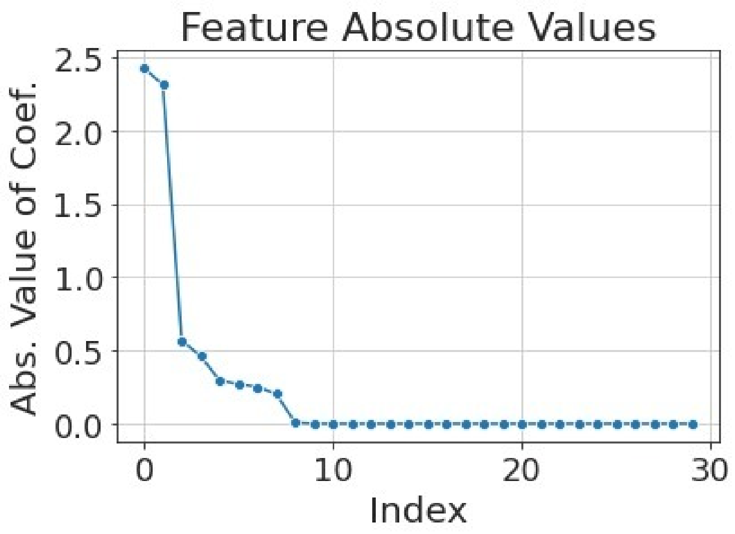
(Q9)

The five features with the largest coefficients:

Feature	Absolute value of coefficient
Sugar levels	2.42
PCR_01	2.31
PCR_10	0.57
blood_type_O	0.46
sport_activity	0.29

(Q10)

The following plot shows the absolute value of each coefficient of the regressor:



(Q11)

The magnitude of the coefficients is interesting for two reasons:

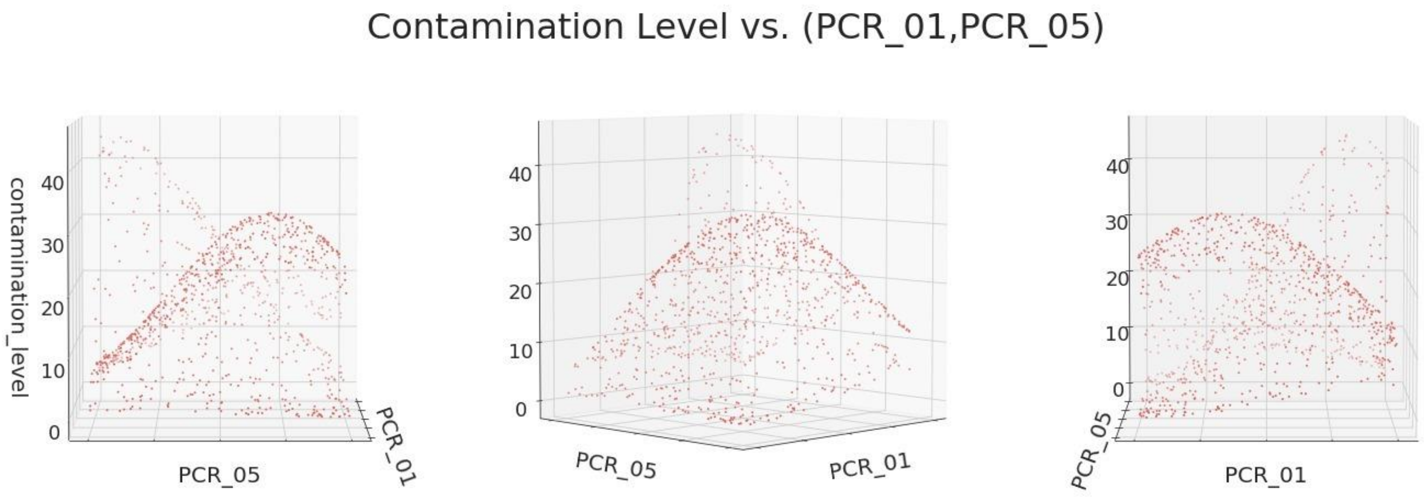
1. Features with high coefficients have a larger effect on the prediction $w^T x + b$, which means they are more important to the regression model. This gives us a clue which features are good for predicting the label (in a linear model, at least). Also, we know that the LASSO regressor prefers sparse solutions (in which many of the coefficients are zero) which further strengthens the fact that the features with high coefficients are important.
2. In general, large coefficients (in abs. value) indicate high model complexity, as we learned in the Lecture and Tutorial. In fact, this is why we use the norm of w (whichever norm we choose) for the purpose of regularization.

(Q12)

Yes, the training performance would change, due to the addition of the regularization component to the objective function. As we explained in Q6, normalizing a feature by a factor α (neglecting the intercept for simplicity) means roughly that the corresponding coefficient for that feature is multiplied by $\frac{1}{\alpha}$ to preserve the same output of the linear model. However, when taking into account the regularization, bigger coefficients have a higher penalty in the objective function, and therefore the optimization will prefer vectors w with small coefficients, even if the coefficients deviate significantly from the coefficient resulting by multiplying by $\frac{1}{\alpha}$. Therefore, the algorithm will converge to parameters w, b that define a different predictor, which results in different training performance (in contrast to Q6). Namely, in a dataset in which the scale of features is not balanced, small-scale features (which need larger weights) will be harmed (since the objective function prefers small weights) and could affect the model less than their true importance.

(Q13)

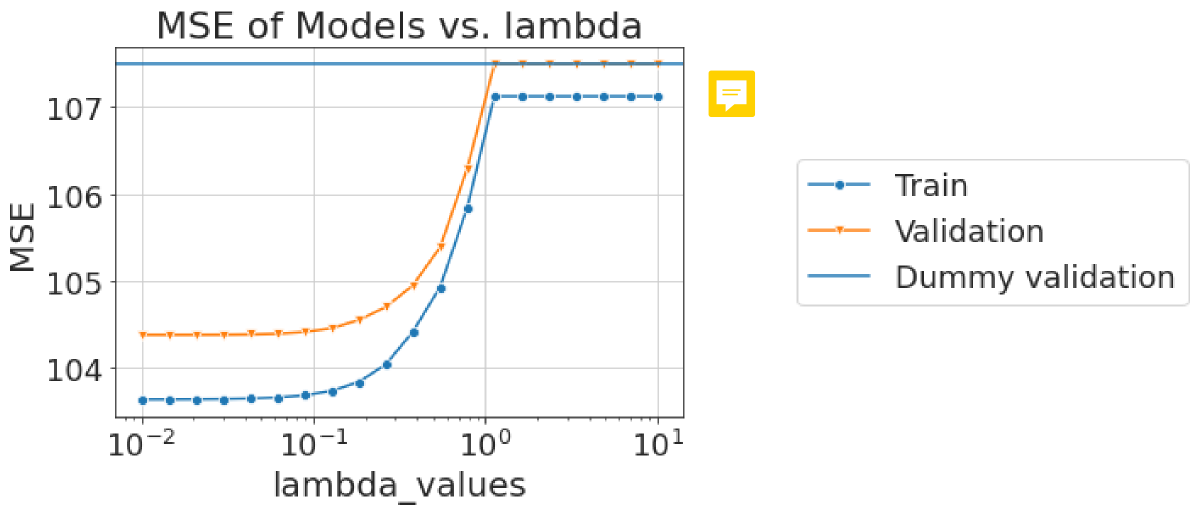
The following is a visualization of Contamination Level as a function of PCR_01, PCR_05:



We can infer from this visualization that the dependency of *contamination_level* on *(PCR_01, PCR_05)* is clearly non-linear: we may be able to estimate it with a polynomial surface, perhaps of 2nd or 3rd degree.

(Q14)

We train as a baseline a linear model for predicting *contamination_level* with *(PCR_01, PCR_05).* The following plot shows the MSE for train and validation, for models with different regularization parameters (lambda):

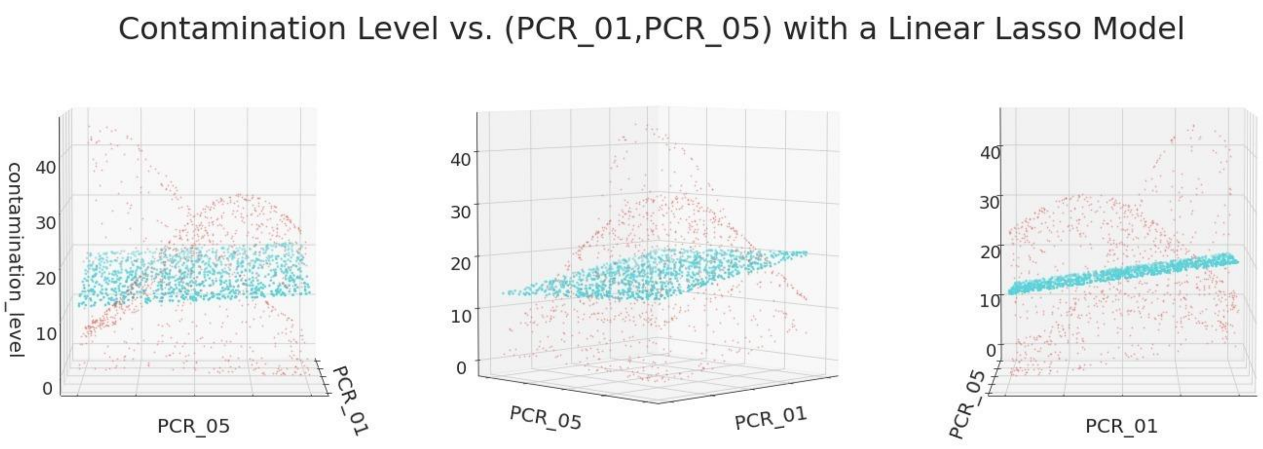


This plot is slightly irregular because there seems to be no improvement in validation error in the beginning of the range (we also checked lower value of lambda, the same trend continues), i.e. the validation error for very small lambda values is not higher than the validation error for slightly larger values. One explanation for this is that the linear model is so unsuitable to the data, that there is no overfitting even for small lambda values, so the best (linear LASSO) model is obtained for any small enough lambda that does not cause underfitting.

The optimal lambda (one of them) is 1.44e-2 and the validation score is 104.38.

(Q15)

The following is a visualization of the true labels compared to the baseline linear model:



(Q16)

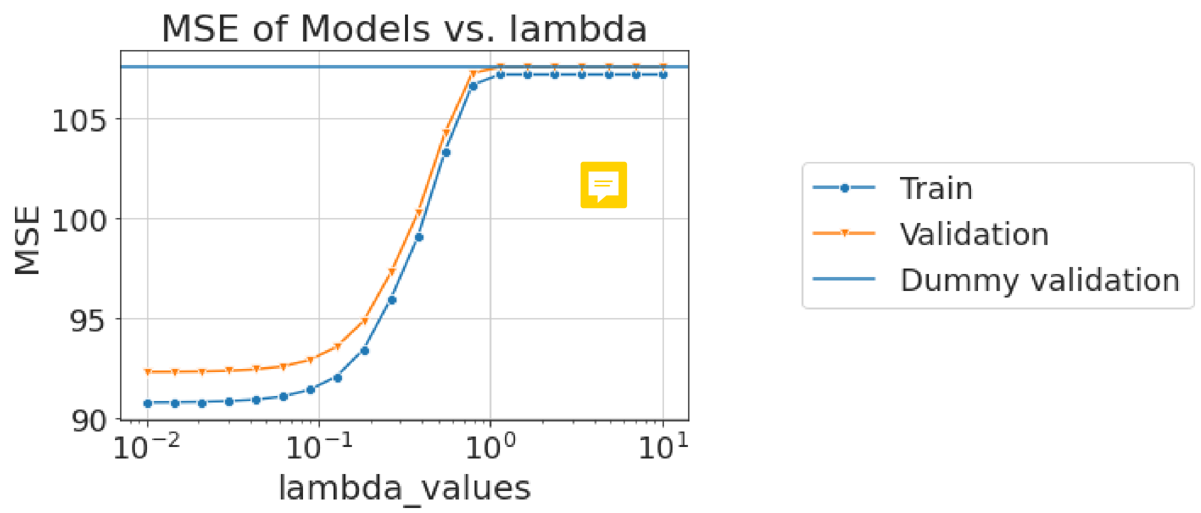
Normalization after applying a polynomial mapping is important because the mapping itself ruins the existing normalization. This is because different monomials in the polynomial have different degrees: for example, in 2d polynomials we have monomials of degree 2 (x_1^2, x_2^2, x_1x_2), monomials of degree 1 (x_1,x_2) (and even a monomial of degree 0). Assuming our original features are normalized and hence roughly in the range $[-1,1]$, the monomials of higher degree will have a smaller scale than monomials of degree 1. This means our new sample set (after the feature mapping), will no longer be normalized.

We know normalization is important for optimizing a regularized linear model with SGD, because, as we saw in Lecture 8, SGD uses the same learning rate η for all features. But, for features with larger scales we need smaller learning rates, because their gradients are larger, and the product LR*gradient must not be too large as to diverge. This means the optimization for a non-normalized dataset may not function optimally – the large features force us to take a small learning rate, which makes the convergence slower. Also, as we explained in question 12, the regularization term prefers small weights, and so small-scale features (which need larger weights) will be harmed and could affect the model less than their true importance.

Therefore, a re-normalization step allows us to improve our optimization process.

(Q17)

We train a LASSO regressor over a mapping of the data into a 2nd degree polynomial subspace to predict *contamination_level* with *(PCR_01, PCR_05).* The following plot shows the MSE for train and validation, for models with different regularization parameters (lambda).



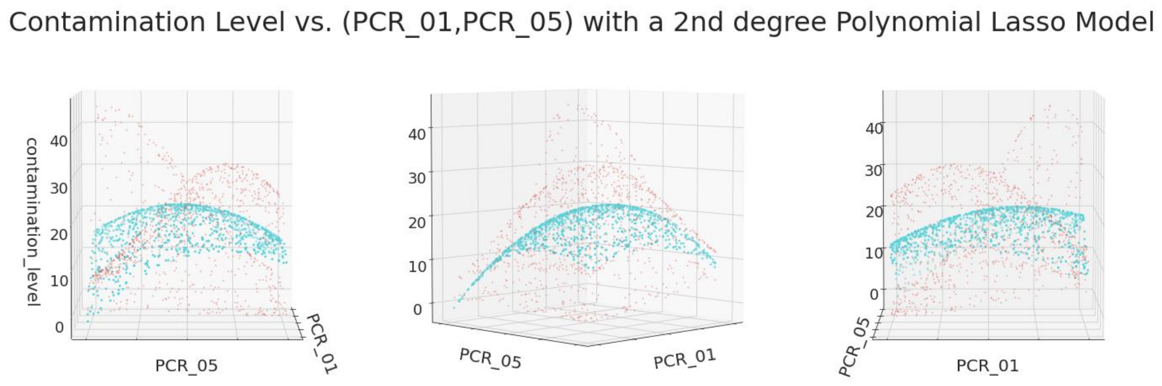
As can be seen from the plot above, for small lambda values, we have an improvement over the baseline model from Q14 of $\sim 11.5\%$ (for large lambda values ($\lambda \geq 1$) we see no improvement in the MSE, which is reasonable because of underfitting). The reason we may have an improvement is that a 2nd degree polynomial may approximate the data better than a linear model.

Again, there is no improvement in validation error when we move from small ($\leq 10^{-2}$) to medium range ($10^{-2} - 10^{-1}$) lambda values (we checked also a bigger range, and the same trend continues): this could be because no significant overfitting occurs.

The optimal lambda is 0.01 and the validation score is 92.27.

(Q18)

The following is a visualization of the true labels compared to the 2nd degree polynomial mapping with a lasso regressor:



(Q19)

As can be seen from the MSE graphs for both models, we can see that there is a noticeable improvement of $\sim 11.5\%$ for the lasso regressor model with the 2nd degree mapping over the linear model. This is because the given data is spread in a non-linear manner, which is better approximated with the model class of the 2nd degree polynomials. This could be visually seen at the 3D plots for both optimal models from each model class, as the linear model is situated around the middle of the dataset, because it is the best fit it could obtain, and therefore it performs poorly over the whole training set, in contrast to the 2nd degree lasso regressor which conveniently placed between the training samples and therefore has bigger success rate (and lower MSE).

However, it is important to note that the 2nd degree polynomial is still not highly suitable to the data; we see this both from the visualization (the surface still seems rather far from the label for many training points) and the MSE, which stays at around 90 (which is significant – roughly speaking, if the mean squared error is 90 then the differences between label and predictions are roughly $\sqrt{90} \approx 9$, which is significant related to the *contamination_level* label which ranges around 0-40).

(20)

After trial and error, we decided to use a 2nd degree polynomial mapping for *(PCR_01, PCR_05)*, as well as an RBF mapping for *sugar_levels*

Explanation: to decide which feature mappings to use, we tried various options of polynomial and RBF mappings, and for each option we created a pipeline that consists of (standard) normalization, feature mapping and a Random Forest regressor with default parameters (n_estimators=100, min_sample_leaf=1). We then did 5-fold cross-validation of the regressor on our training set.

We tried multiple approaches to finding appropriate features and feature mappings to improve our validation error:

1. We looked at features that had high coefficients (hence high importance) in the LASSO model we trained previously (*sugar_level*, *PCR_01, PCR_10, blood_type_O, sport_activity)*, and also at *PCR_05* which we know from the previous section has potential for a polynomial feature mapping. For these features we tried polynomial feature mappings of various degrees (for each feature by itself and also multivariable polynomials with combinations of the features), including specifically the pair *(PCR_01, PCR_05)* we used in the last section. The general trend was that the higher the degree of the polynomial (especially above 3), the higher the MSE – most likely due to adding many irrelevant features. However, staying with a 2nd degree polynomial mapping of *(PCR_01, PCR_05)* seemed to yield good results. Also, *sugar_levels*, which had a very high coefficient in the LASSO model, yielded good results when applied with an RBF mapping. After some trial and error we choose gamma=0.01 for this mapping.
2. We looked at features with high statistical correlation to the label (a similar set of features to point 1, but not exactly the same), and tried the same approach by searching for polynomial mappings and RBF mappings with different combinations of the features, but this did not yield any exciting results compared to point 1.
3. At last, to make sure we didn't miss any additional features, we checked each and every feature in the dataset to see if adding an RBF mapping with that feature improves the performance of the model – but it did not. A possible continuation of this brute-force approach would be to check every combination of 2 or 3 features, but this quickly becomes computationally unfeasible.

(Q21)

We know that the RBF feature mapping is infinite-dimensional, and has a very high expressive power (we learned that a linear combination of the components of an RBF mapping can express **any** function of the original features).

In every iteration, decision tree algorithms choose a feature, based on the training set, with which to define decision rules; since the components of the RBF mapping are highly expressive, there is a good chance that they are helpful in reducing impurity of the nodes, and therefore will be chosen. Also, an RBF mapping typically has a large number of components (the default in sklearn is 100); this means that when Random Forest randomizes a subset of the features from which to choose the optimal feature for a decision rule, it is very probable that some of the RBF mapping features will be present in the subset. Therefore, we can expect the trees in the forest to perform better on the training set compared to the raw data, i.e. have a lower training error.

Regarding validation error, we know that with high expressivity comes the danger of overfitting to the training set, which means higher validation errors. But, Random Forest is a bagging algorithm, which means it is precisely meant to aggregate many overfit (low-bias high-variance) models into a low-bias low-variance model. Therefore we expect that the algorithm could handle this without overfitting. If overfitting of the final model still occurs, tuning the regularization parameter gamma should help mitigate this effect.

(Q22)

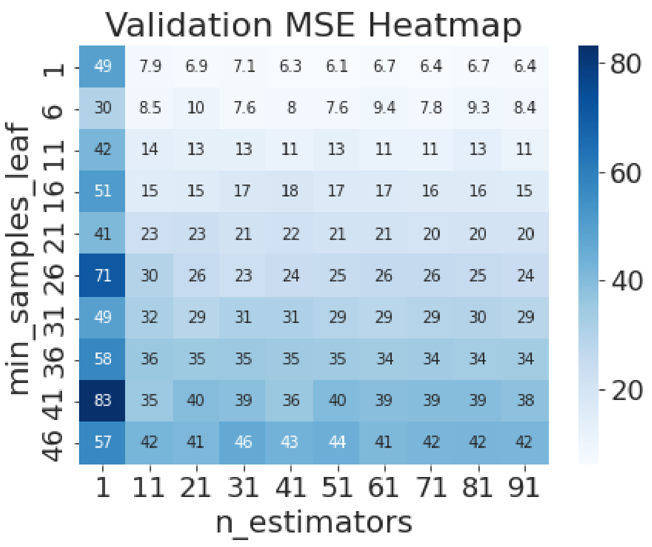
The major difference between Random Forests and polynomial regressors is that Random Forests are ensemble predictors, and in particular they use the “bagging” method: each tree in the Random Forest is a high-depth decision tree, which as we know is a complex model that is prone to low bias and high variance (aka overfitting). The ensemble aggregates many of these models, over “bootstrapped” sample sets, in order to drastically reduce the variance, resulting in a low-bias low-variance model. Random Forest in particular utilizes randomness when choosing subsets of features, in order to further decrease the correlation between the different trees in the forest, and hence improve the overall accuracy of the ensemble.

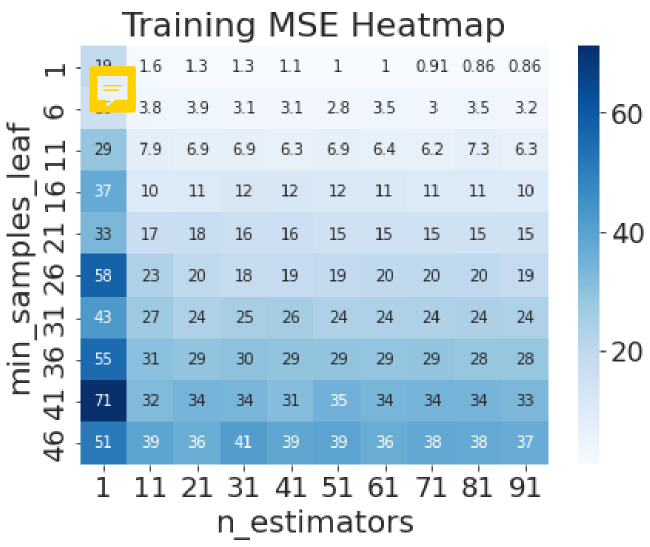
All these properties are very different from a polynomial regressor: first and foremost it is a single regressor, trained on the ordinary sample set (with no “bootstrapping”) and therefore does not benefit from the reduction of variance that we discussed. The bias-variance balance that the polynomial model achieves depends on such factors as 1) the degree of the polynomial, 2) how suited the data is to a polynomial fit, and 3) the strength of regularization used. Typically, is not as impressive as Random Forest achieves. In other words, a polynomial regressor is a much simpler model, that fits the distribution well only in certain cases. In order to achieve such low bias as Random Forest it must sacrifice low variance, i.e. lose out on generalization.

(Q23)

We tuned the parameters “n_estimators” and “min_samples_leaf” with a GridSearch on the range $n\_estimators \in range(1{,}100{,}10)$ and $min\_samples\_leaf \in range(1{,}50{,}5)$. (In python notation, i.e. n_estimators was sampled between 1 and 100 in steps of 10).

The following plots show the mean training and validation error (mean on the cross-validation folds) for each parameter combination:





We can see a general trend of improvement, for both training and validation scores, as n_estimators increases and min_samples_leaf decreases. These are both results that make sense:

1. Regarding min_samples_leaf, we know that the Random Forest model is built around the idea that each decision tree is a complex, high-variance model. As min_samples_leaf gets smaller, this is exactly what happens (the tree is more tuned to the specific bootstrapped training set and to the random choice of feature subsets), therefore we expect the performance of RF to improve. In fact, in the lecture we learned that RF usually uses an unbounded-depth tree, based on the same idea (increasing depth and decreasing min_samples_leaf have a similar effect).
2. Regarding n_estimators, we know that in bagging algorithms, "the more the merrier", i.e. more models in the ensemble mean a stabler and lower-variance final model (albeit with a cost of training and prediction time). Therefore we expect both training and validation scores to improve when increasing n_estimators. Indeed this is what we see in the plots. However, starting from roughly 50, adding additional estimators no longer has any significant effect.

The optimal combination of parameters (although there were many similar ones) is

$$min\_samples\_leaf = 1, n\_estimators = 51$$

With a training score (MSE) of 1.01 and a validation score of 6.1.

(Q24)

Model	Section	Train MSE (cross validated): Mean (std)	Valid MSE (cross validated): Mean (std)
Dummy	2	107.13 (1.7)	107.49 (6.7)
Linear	2	95.46 (0.95)	101.15 (4.29)
Lasso Linear	3	96.52 (0.8)	98.82 (3.95)
RF Regressor	5	1.01 (0.35)	6.10 (2.24)

(Q25)

At last, we check the performance of our different models on the **test** set:

Model	Section	Train MSE (cross validated): Mean (std)	Valid MSE (cross validated): Mean (std)	Test MSE (retrained)
Dummy	2	107.13 (1.7)	107.49 (6.7)	110.22
Linear	2	95.46 (0.95)	101.15 (4.29)	104.76
Lasso Linear	3	96.52 (0.8)	98.82 (3.95)	103.78
RF Regressor	5	1.01 (0.35)	6.10 (2.24)	3.33

First, in all cases we can see the test error is in range of approximately one-two standard deviations from the validation error. This is a good sign – it means we performed cross-validation correctly, and the validation sets were indeed a good replacement for the test set (which affirms the theory).

Also, we note that the RF regressor performed very well on the test set, and again it shows the practice behaves according to the theory of bagging: we know that each tree in the forest is prone to being a complex, overfit model, but thanks to the large ensemble, and the randomness in the RF algorithm, the aggregated model is not overfit, and the generalization to the test set is very good.

It is clear that the RF regressor performs much, much better than the linear and LASSO linear models (and the dummy model, of course). We have explained in (Q22) why this is reasonable: an RF regressor is an expressive model, and as we explained its properties make it not prone to overfitting. This comes in contrast to linear models (with or without regularization), which are very simple and in our case **underfit** the data, which is simply unsuitable to a linear model. (Needless to say, the dummy model, which simply predicts the average, also underfits the data)

Regarding the difference between the linear and LASSO linear models (which both perform badly), we can see there is a slight improvement in the generalization capability for LASSO, both in validation error and test error. Of course, this makes sense, since improving generalization is the very point of the regularization we introduced in the LASSO model.