Themes on machine learning (TOML) Third and Fourth project

Contents

1. Third Project

1.1. Introduction

The main goal of this project is to calibrate an air pollution sensor in an air pollution monitoring sensor network. To do this the data obtained by this network is modeled using different methods. The results will be compared using graphs and metrics and one of them will be selected as the best method in the conclusions part of the report.

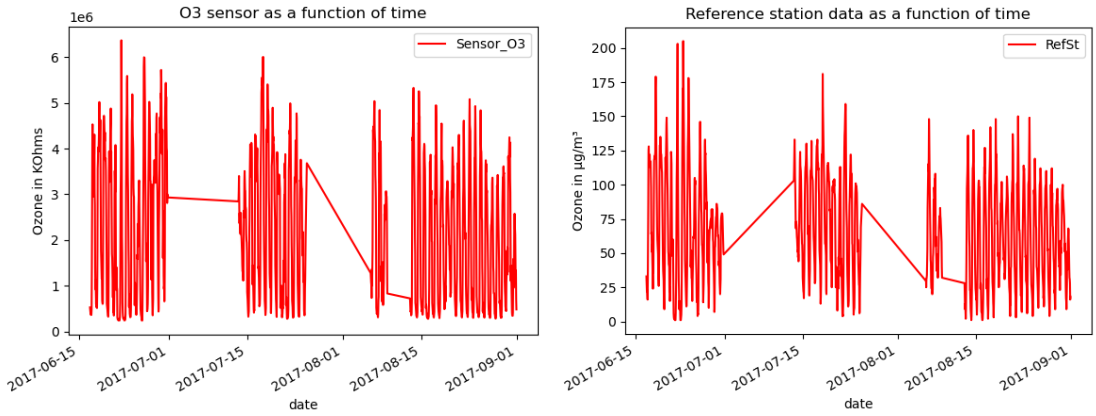
The distribution of the report is as follows: first the sensor data files and data is explained and analyzed using graphs, then several modeling techniques are applied to this data and, finally, the conclusion of which is the best modeling technique is made using graphs and metrics to compare in the last section.

1.2. Data observation and first graphs

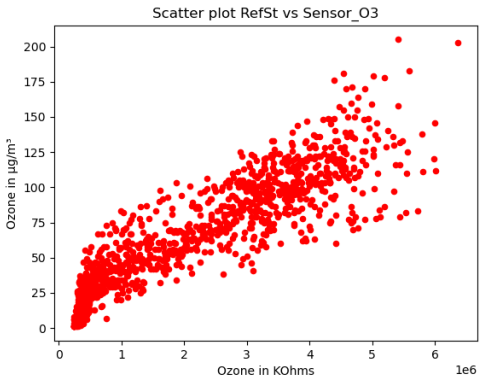
At the beginning various data files must be merged into a single data file to enable the modeling in the next steps. The final merged file has the following columns containing data from different sensors:

- **date:** This column has the timestamp of the data.
- **RefSt:** This column has the O3 value of the reference station. The value is in µg/m³.
- **Sensor_O3:** This column has the data of the O3 sensor in KOhms.
- **Temp:** This column has the temperature data in ºC.
- **RelHum:** This column has the relative humidity data in %.
- **Sensor_NO:** NO sensor data.
- **Sensor_NO2:** NO2 sensor data.
- **Sensor_SO2:** SO2 sensor data.

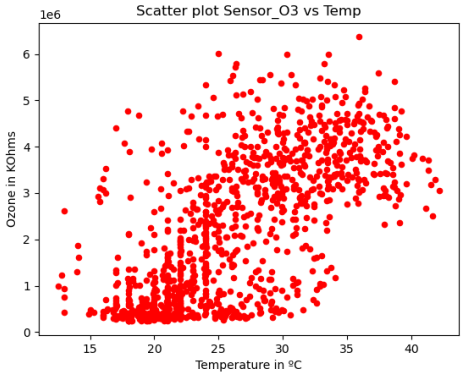
First, let us compare the plots of the reference station data and the O3 sensor data. The magnitude should be different, but the shape should be the same since the data is the same and is taken in the same periods/intervals of time.

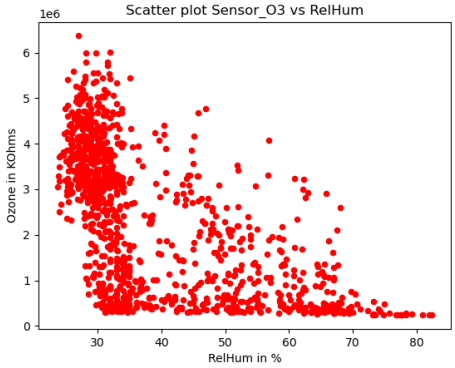


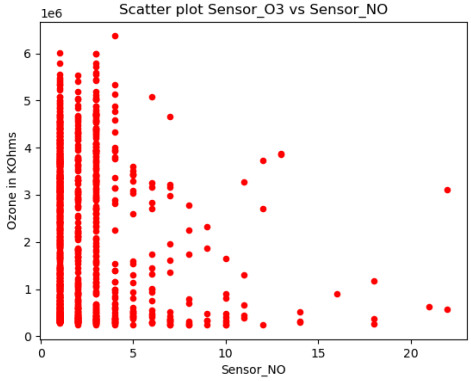
The shape of both graphs is the same. It has slight differences but overall, it has the same shape. The next step is to plot the relationship between both parameters. As mentioned before, both parameters are the same kind of data but taken by different sensors at the same time and with different units, thus, the relationship should give us a scatter plot with a 45 degrees shape approximately.

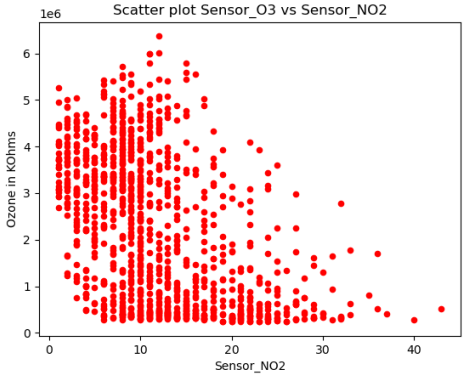


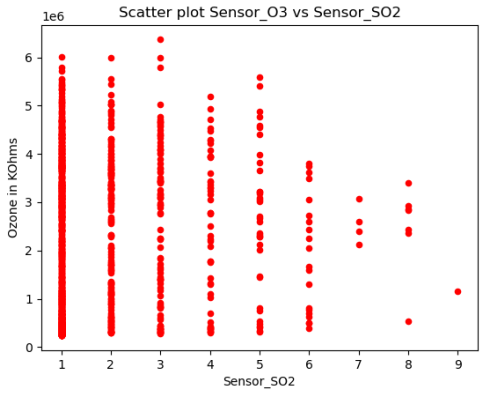
The graph confirms that the data of both sensors is related and has the same kind of variations. Next scatter plots show the relationship between the Sensor_O3 data end all the other features to be in the model.



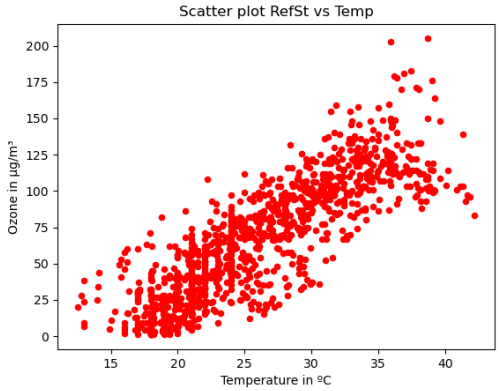


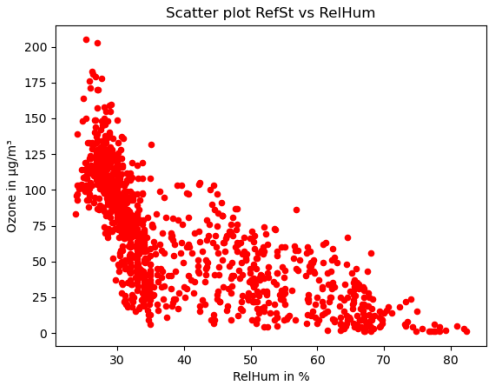


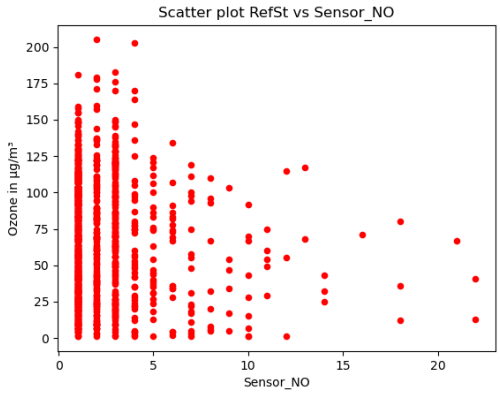


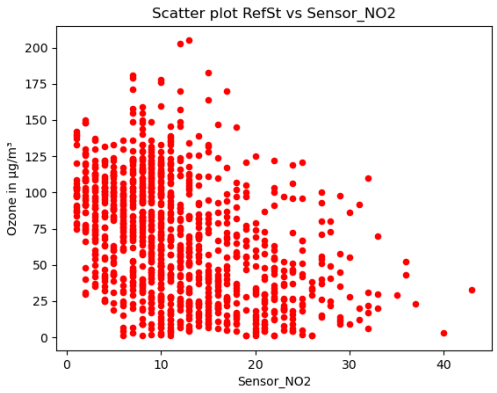


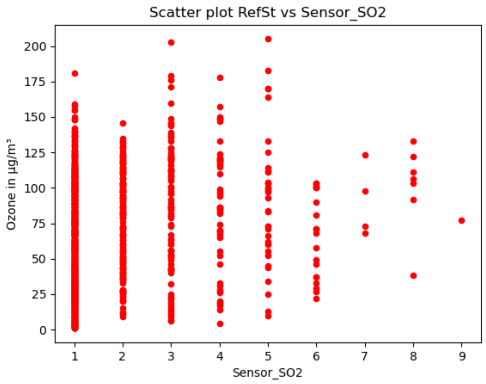
The same is done for the RefSt data. Next plots show the relationship of the reference station data with all the other parameters in the data file.











In both groups of plots (for the O3 sensor and for the reference station sensor), the temperature data is the one that has the most visible pattern. The next feature that shows a relationship pattern is the RelHum data. All the other variables do not have a distinctive pattern with the dependent variables of the model (Sensor_O3 and RefSt). This indicates that these parameters should be the most important in terms of the construction of a model. As we will see in the next sections, this is true.

For the plotting of the former graphs, the data is not normalized since the analysis done in this section is more focused in the shape rather than in the magnitude or the value of the variables. For the modeling part, all the data is normalized due that is a mandatory previous step to get correct models.

1.3. Modeling and calibration

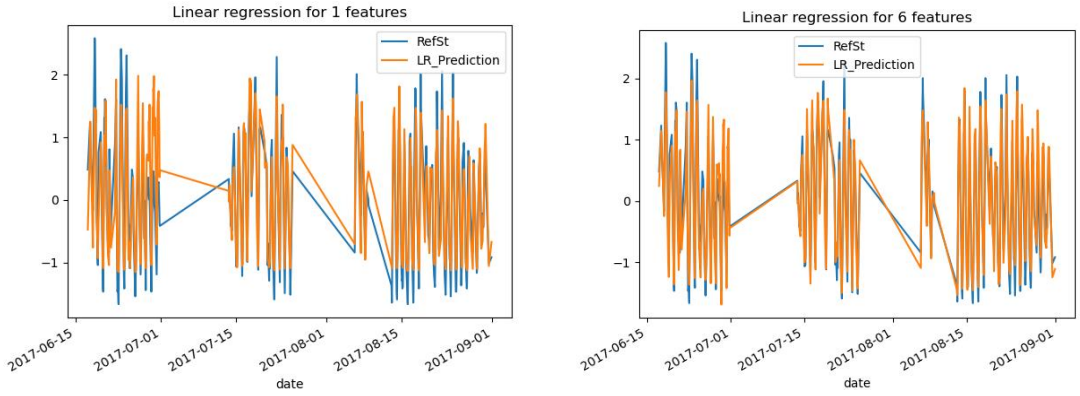
This section has the explanation of the different methods used to calibrate the data of the sensors and the different graphs obtained from the implementations in python. The first two sections are done using all the data features, then from KNN and on, the features used are the ones obtained as the best ones in the subset selection (the ones that are the most significative in the plots of the regularization methods).

1.3.1. Forward Subset Selection (MLR- FSS)

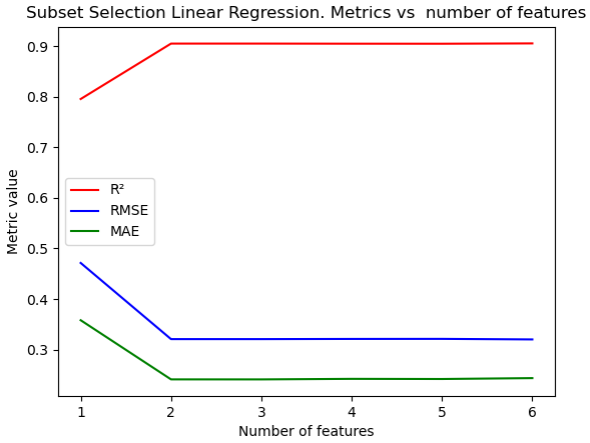
The first part of the project is picking which variables of the data in the datasets model the reference station data with the best metrics (the most significative amongst all the features). This selection is done by implementing Forward Subsect Selection in python. Forward subset selection has two phases:

1. First, it fits all the possible models that come from the combinations of k features (independent variables of the model) out of a total of P. For example, if the data has P = 3 features it will create a model with k = 1 (3 models with a single feature), k = 2 (3 combinations of two features) and k = 3 (one model involving all features). From each k group (k = 1, k = 2 and k = 3) the best model is selected using metrics as RSS, CV error or Adjusted R-squared. The k is given to the function and is the hyper parameter of the method. Then the criteria to choose the model is also an input of the python function.
2. The last step consists of picking the best model of all the models gotten in the first step. The selection in this phase is done using the same metric as in the first step.

The next graphs show the predictions done by a model with just k = 1 and a model done with k = 6 (all the features).



As expected, the predictions done by the simple model (k = 1) are bad and the predictions with all the features in the model are the best ones. The problem is that the last model is complex, and this can end up causing overfitting. The model can be made of significative features avoiding more features to be in. This can be seen in the next plot:



The changes in the metrics when 2 main features are included in the model are not so significative. Therefore, the model of the sections KNN and forward is going to have just 3 features ['Sensor_O3', 'Temp', 'RelHum'].

1.3.2. Multiple Linear Regression (MLR) with regularization

Multiple linear regression is a method to create models using two or more independent variables to predict the outcome of a dependent variable. The model is fitted using training data which adjusts the coefficients of the model to get to a point where the cost function between the model line and the training data is at its minimum. The cost function usually applied is least squares error (LS).

One of the problems with the models is that they can get complex whenever they have a lot of features (as in our case). When a model gets too complex it can suffer from overfitting, i.e., it fits the training data perfectly, but it has a huge bias with test data (hence, it has huge variance between the train and the test data). A way of solving this problem is to use a regularizing method. These methods shrink the coefficient estimates of the model to zero, so the model gets less complex (some of its less significative features disappear as they have coefficients close to 0 or equal to 0) and it avoids the risk of overfitting. Another way of saying the same is that whenever the model is penalized it becomes less sensitive to the changes in the features it has, so the ones that already had low impact on the model become inexistent (or almost inexistent) for the model. This also reduces the variance of the model without substantially increasing its bias. Ridge regression and Lasso regression are two examples of MLR with regularization methods.

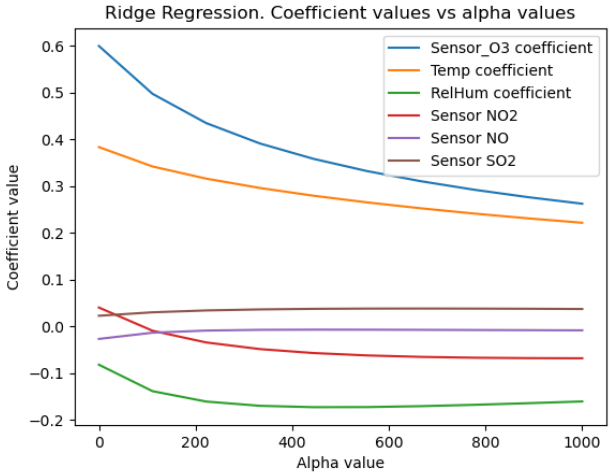
1.3.2.1. Ridge Regression (RR)

Ridge regression, also called L2 regularization, has the following formulation for N data points and P features:

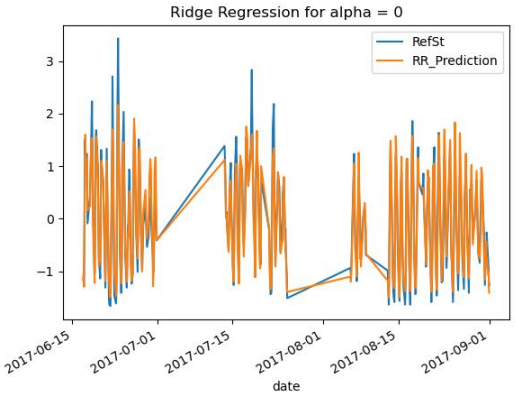
$$\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{P}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{P}\beta_j^2=RSS+\lambda\sum_{j=1}^{P}\beta_j^2$$

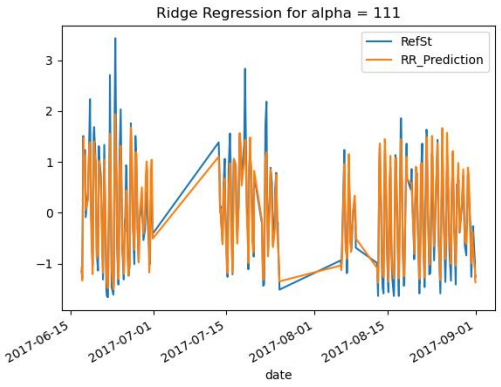
The goal is to find the lambda that gets the minimum of the former formulation where RSS stands for Residual Sum of Squares. A lambda equal to zero corresponds to a simple linear regression (the penalizing term goes to zero). With greater lambda is, the less flexible the model gets, and it reduces model complexity and multicollinearity. Since great values of coefficients mean more flexibility (complexity), increasing the lambda of our model should give coefficients

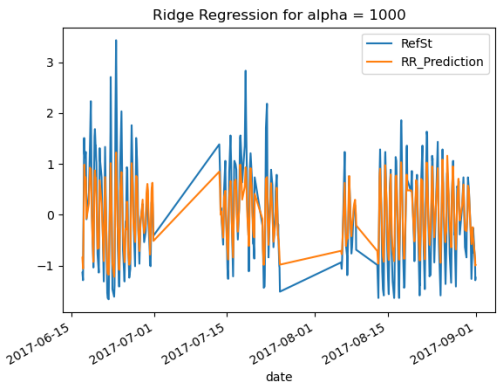
that get asymptotically near zero (but never equal to zero). This can be seen in the next graph where the coefficients of the model are plotted against the value of lambda (alpha in python).



Also, we should have better fits for the data with alpha values in the lower range of the 0 to 1000 interval of the plot, because higher alphas give a hipper simplified model (insensible to almost all changes in data). Next plots show the reference station data in blue and the prediction in orange for different values of alpha.

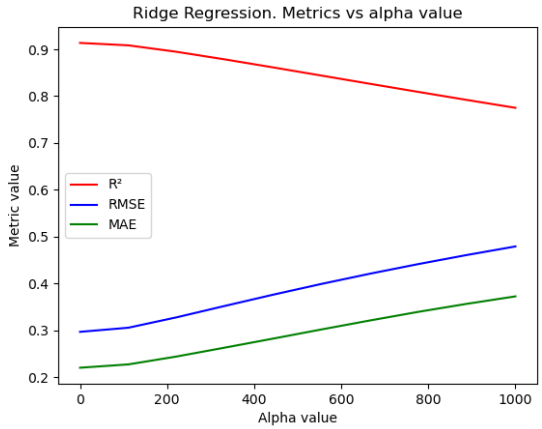






The first graph corresponds to a simple linear regression prediction. The next two graphs to a low penalization (left) and a high penalization (right). The low penalization model gets closer to a good prediction as the simple linear model. The high penalization model gets worse predictions, the amplitude of the data is smaller in comparison with the other two plots.

Next plot shows the impact of the value of lambda in the metrics of the model. What we should see is a loose of relevance of the model (R squared should go low) and an increase in the error metrics since the less flexible a model is (less features the model has), less accuracy it has when predicting data.

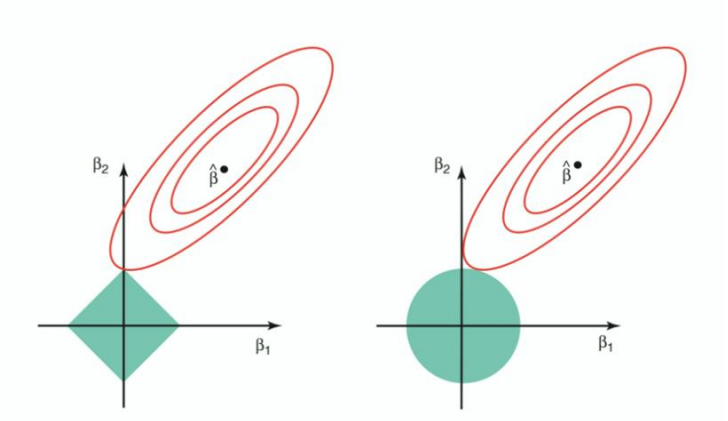


1.3.2.2. Lasso Regression (LR)

LASSO (Least Absolute Shrinkage and Selection Operator), also called L2 regularization, has the following formulation for N data points and P features:

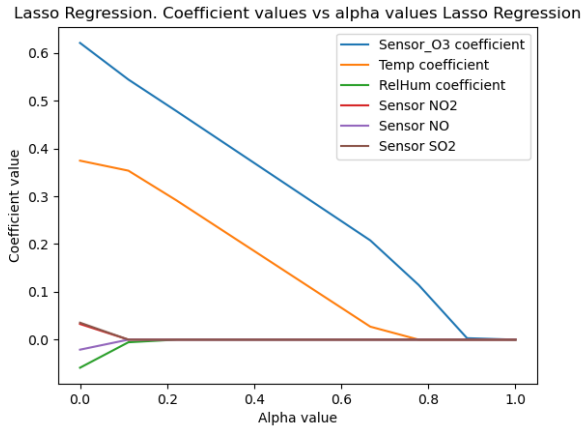
$$\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{P}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{P}|\beta|=RSS+\lambda\sum_{j=1}^{P}|\beta|$$

As it can be seen the LASSO regression is like the Ridge Regression, in fact they only differ in the penalization and what they achieve with it. In the case of LASSO, the penalization is the module of squares of the coefficients. This makes the penalization more aggressive towards bigger coefficients. This also enables the model to get rid of some of the coefficients because they can be 0. The demonstration can be explained as follows. If we think Ridge and LASSO as a problem to solve. For example, for Ridge of 2 parameters would be $\beta_1^2+\beta_1^2\leq s$ where s is a constant that exists for each value of lambda. Graphically:

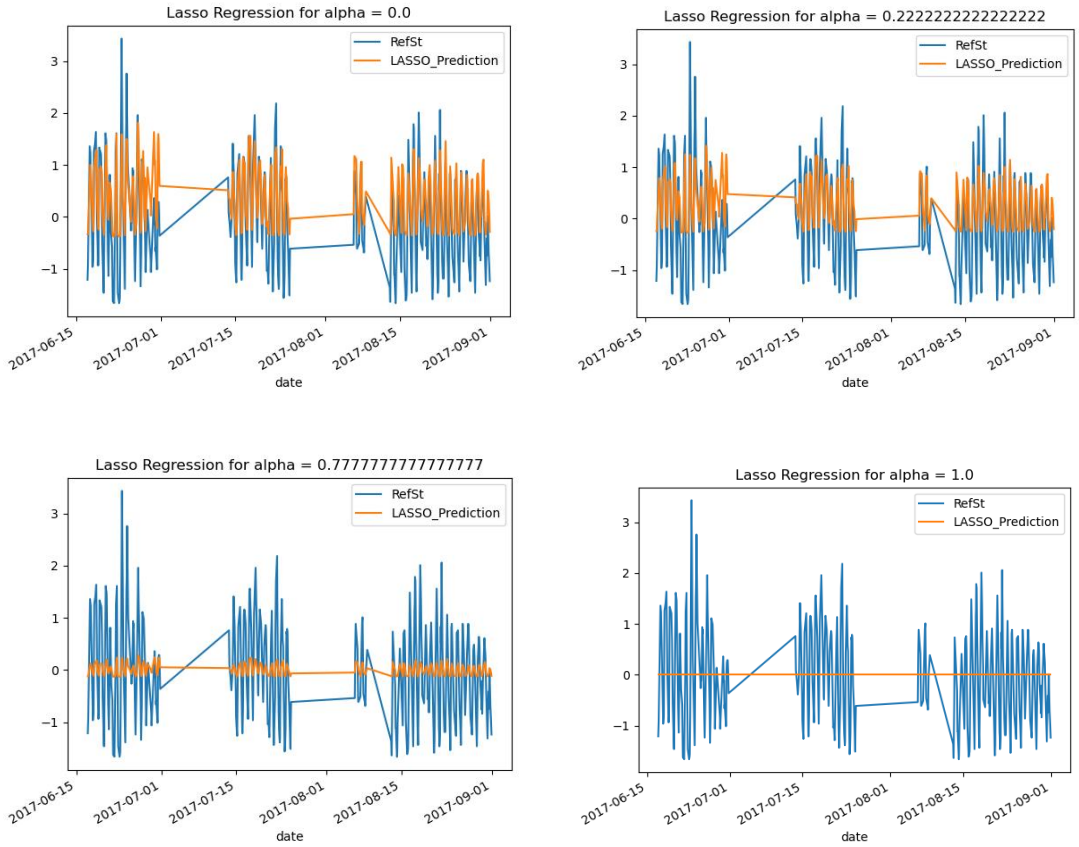


The representation of the left is the LASSO problem, on the right is the Ridge regression problem. The red ellipses are the contours of RSS. Points of these ellipses are the RSS values. Regressions with point on the green zones have the lowest RSS value. When an ellipse has a huge s, its center lies in the green zone, this means that we are in a case of simple linear regression. For points of

the ellipse that just touch the green zone we have the coefficients of LASSO/Ridge regressions. In the case of Ridge, since it is a circle, no point will touch a null value in any of the axes. In LASSO this can happen as it can be seen in the image. This makes 0 coefficients possible. In out model the coefficients of lasso are zero for maximum values of lambda:

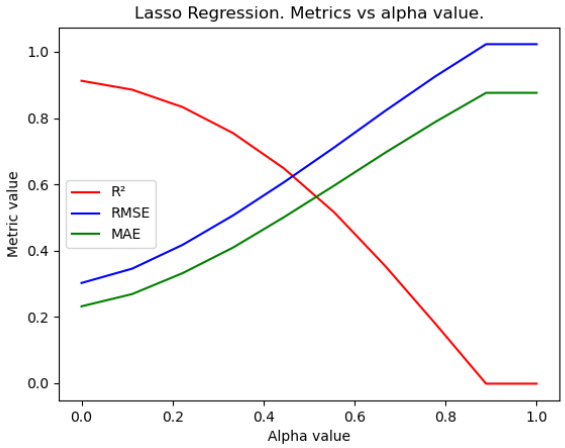


The most significative variables are the last ones turning 0 and the less significative ones are the ones that get to 0 earlier. In LASSO this also implies that the model will have more error and worse estimations. Next plots show four predictions two predictions with low alpha and two with high alpha (lambda).



The greater the penalization, the worse the model gets, and the estimation is worse.

Next plot shows the evolution of the metrics of the model with respect to the value of the lambda.



As we can see, with the penalization, the R square, which evaluates the fit and the sensibility of the model, decreases. Also, the errors increase because the model gets worse in the predictions when its less sensible (more rigid/inflexible).

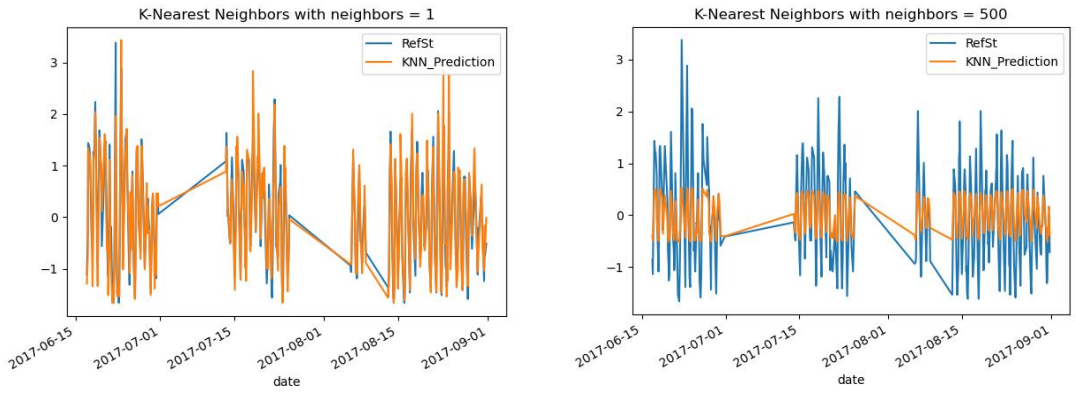
1.3.3. K-nearest neighbor (KNN)

KNN is a non-parametric method that approximates the association of independent variables and the outcomes by averaging the observations in the same neighborhood. It uses feature similarity to predict the values of the new data points, i.e., new points are assigned values depending on how close they resemble the training data. The size of the neighborhood is a hyperparameter. This method becomes impractical with huge dimensionality (great number of independent variables).

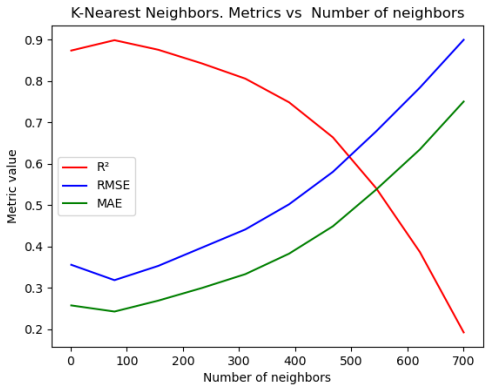
The steps of the algorithm are the following:

1. First, the distances (Euclidean distances) of the test data points and the training data points are calculated.
2. The closest k (number of neighbors hyperparameter) neighbors of each test data point are selected based on the distance calculated in 1.
3. The average of these k neighbors values is calculated for each test data point. This average is the prediction of the algorithm for a datapoint.

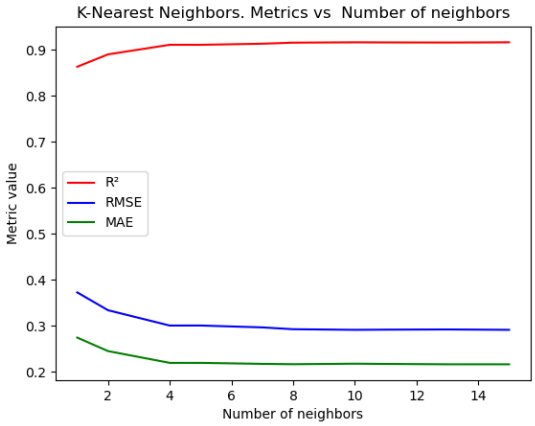
Knowing how the algorithm works we should expect bad predictions for small neighborhoods because the model becomes too simple and bad predictions for big values of k. For huge values of k, the algorithm selects neighbors with big distances to the test point that is being evaluated, and thus, the average value is perverted with wrong data (huge error). The next graph shows a model with a low number of neighbors and one with too many neighbors.



For a small number of neighbors, the result is an estimation with more amplitude that the real data in some places (it has error). If the number of neighbors is too big (close to the full size of the training set) the model is even worse. This can be seen in the next metrics graph.



It gets to a point where the model is not fitting anything. The correct value of neighbors can be found in the k = 1 to k = 15.



Looking at the metrics tables, we can see that the best k value is 10.

1.3.4. Kernel Ridge Regression with RBF kernel (KRR-RBF)

Kernel Ridge regression consists of the kernelization of the Ridge Regression method. The method can be mathematically explained in the following (simplified) way:

1. Express $h: X \to Y$ using real-valued function $f: Z \to \mathbb{R}$, this is:

$$y = \mathbb{R}: \quad h(x) = f(x) \text{ with } f: X \to \mathbb{R} \quad (X = Z)$$

2. Define an empiric risk function $R_n(f)$ to assess how "good" a candidate function f (from the training set S_n) is. Typically, it is the average of a loss function.

$$R_n(f) := \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i, y_i)$$ where n is the size of the training set S_n.

The penalization in this method is applied on the functions of the feature space. This regularization also prevent overfitting.

3. Define a positive definite kernel on Z and solve $\min_{f \in H} R_n(f) + \lambda \|f\|_H^2$ where the function can be expressed as:

$$\hat{f}(x) = \arg\min_{f \in H} \frac{1}{n}\sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \|f\|_H^2$$

Also, we have: $\hat{f}(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ Where k is the Gram matrix.

4. Then the KRR problem is equivalent to:

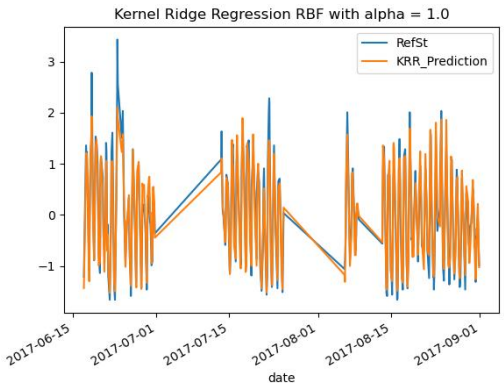
$$\arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{n}(K\alpha - y)^T (K\alpha - y) + \lambda \alpha^T K \alpha$$

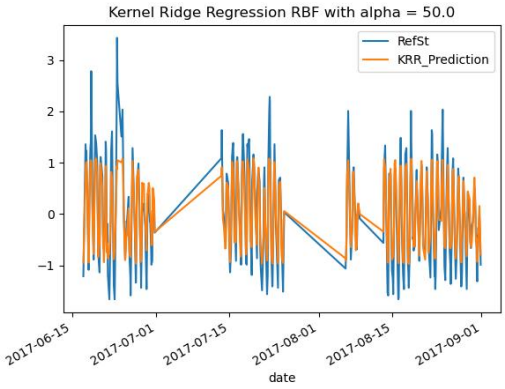
This is a convex quadratic minimization problem.

The kernel measures the difference between the training set and the test set, in our case the kernel is RBF (Radial Basis Function) which has the following formulation:

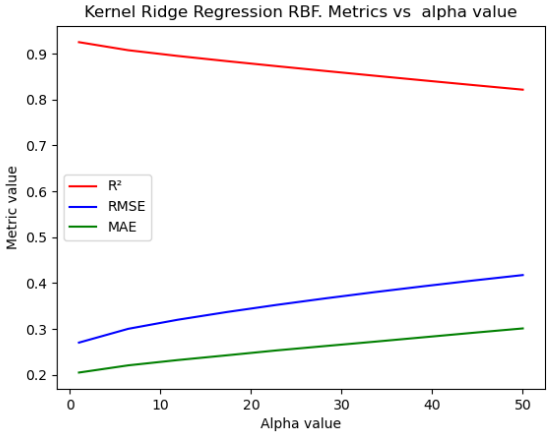
$$K_{RBF}(a, b) = e^{-\gamma \|a-b\|^2} \text{ where } \gamma = \frac{1}{2\sigma^2}$$

The results of increasing lambda should be the same as in the ridge regression method. The next plots demonstrate that this is true (and that a much smaller lambda is needed to penalize at the same rate).





The predictions get worse with higher lambda get worse results. This also is seen in the errors graph with respect to the lambda value:



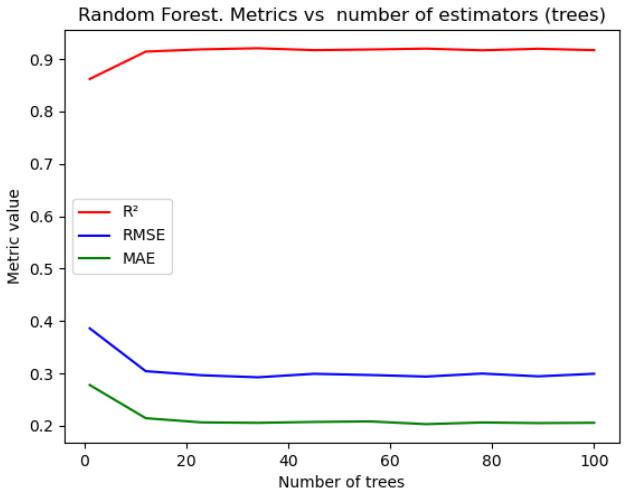
Another parameter that is important for the RBF kernel is the $\gamma = \frac{1}{2\sigma^2}$ which plays with the width of the gaussians of each dot of the reining set. If the gaussian is wider, it can cause overfitting since more points will enter in the range of significative weights (more points will be identified as similar, as it happens in k-nearest neighbors). This parameter has been left as default.

1.3.5. Random Forest (RF)

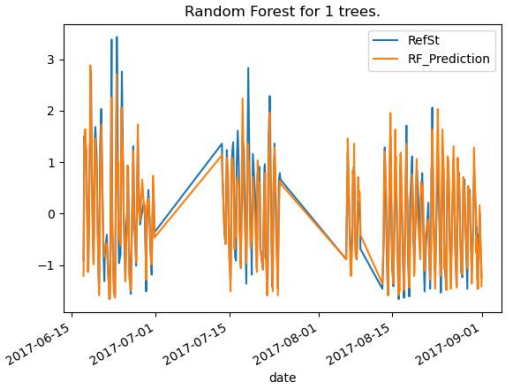
Random forest algorithm uses decision trees as base. The algorithm works in the following way:

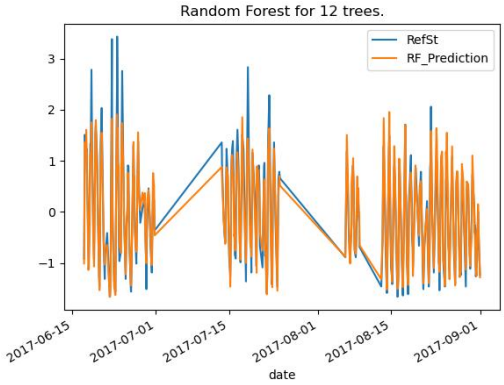
1. N number of random records are taken from the training set. N is a hyper parameter and corresponds to the number of trees in the forest.
2. A decision tree is created from each sample of data.
3. Each decision tree generates an output.
4. The final output of the forest will be the mean of all the values of the leaves of the trees.

This means that more trees will have better results since the mean will consider more information from all the trees. Less trees will have the opposite effect. As in the other methods, the trick is to get the best number of trees which is the one that minimizes the error in the predictions of the model. Also, there is a point where including more trees makes no difference in the result of the forest since the mean is not changed. This can be seen in the next metric plot:



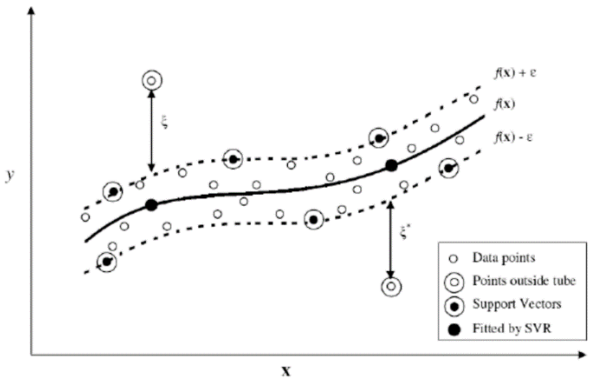
In this method, the difference from estimations of 1 tree and estimations of, for example, 12 trees are huge. This is shown in the next two plots.





1.3.6. Support Vector Regression (SVR)

SVR is a generalization of the classification in the Support Vector Machines method. In this case the output of the method is continuous instead of labeled. Next image shows how does the method work:



SVR, as SVM, is based on the construction of a tube (called epsilon insensitive tube). All the points are called vectors. This method only considers the vectors that are outside of the tube. The vectors outside of the tube are called support vectors since the regression line of the model is based on the value of the slack variables of these support vectors. The line in the middle of the tube is the regression line. We aim to create a tube that fits the best regression line with the data that we have. For this, the epsilon can be modified to set the tube width. In our case this parameter has been left to default (0.1). The wider the tube, the worse SVR is going to fit since the errors taken ion consideration will be fewer (all point will be inside the insensitive tube). The formulation for the Support Vector Regression is the following:

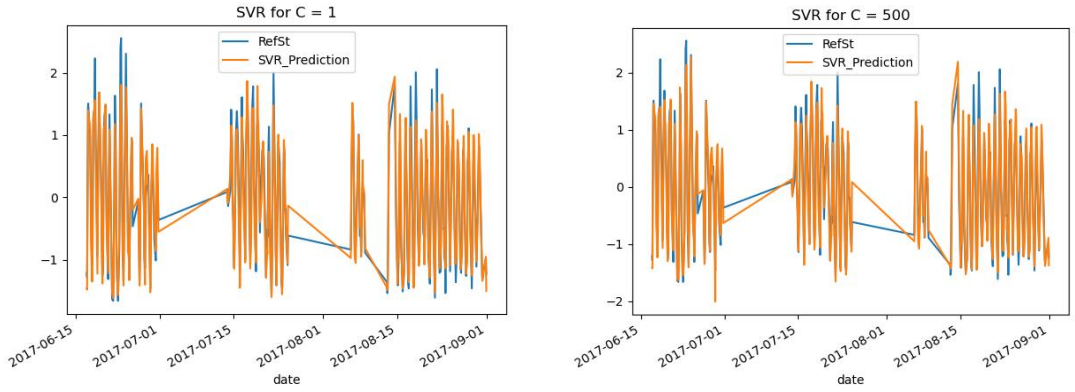
$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

We aim to find a tube that minimizes the former expression. Notice that the last formula has a penalization term that comes from the summation of the distances of the support vectors and the tube (summation of slack variables) multiplied by a constant C. In the python code, this constant C can be tuned, and it is the hyperparameter used to create the graphs of this section. C can be seen as a regularization parameter.

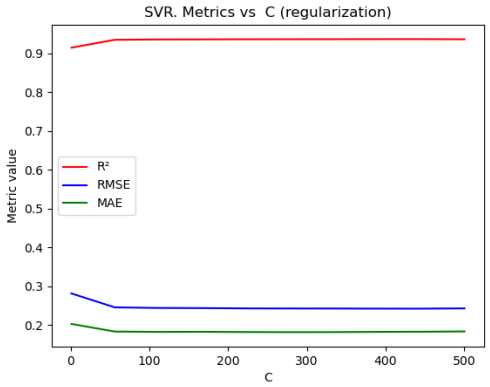
The kernel used by the SVR method is the same as in KRR, the RBF kernel. The formulation of this kernel has been seen before and is:

$$K_{RBF}(a,b) = e^{-\gamma\|a-b\|^2} \text{ where } \gamma = \frac{1}{2\sigma^2}$$

In our case, Gamma is set to auto. The results of the method for lower C and higher C are the following (in python, C works inversely, more C means less penalization):



As we can see both models look good but the one that fits better is the one with less penalization (C = 500). Next plot shows the evolution of the metric with respect to the C value and confirms that less penalization gets better results (remember that C it is inverted in python).

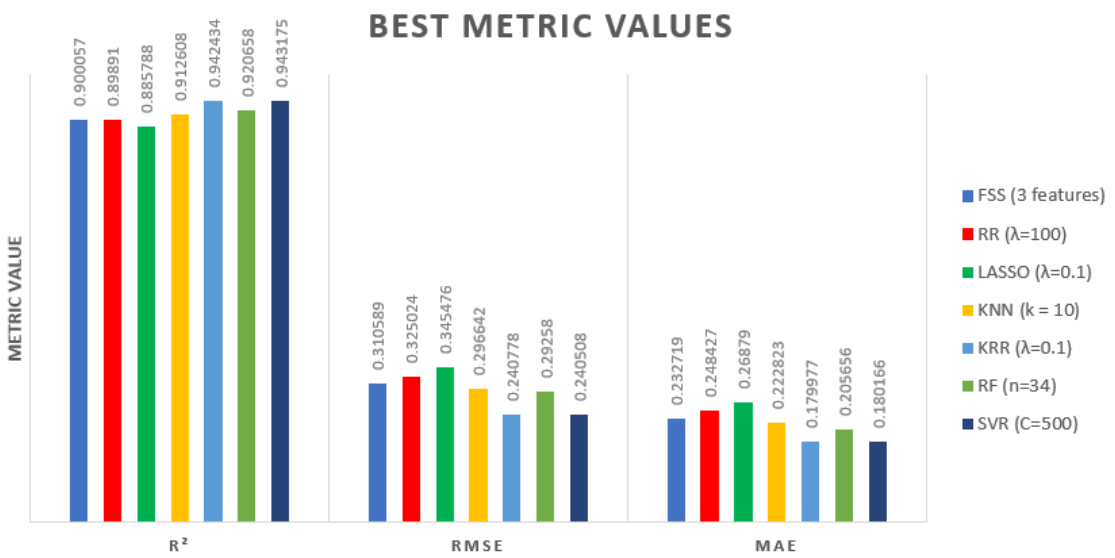


1.4. Comparison between the method results

After selecting the hyperparameters that give the best results for each one of the regression methods, the resulting table for all the metrics is the following:

Method	R^2	RMSE	MAE
FSS (3 features)	0.900057	0.310589	0.232719
RR (λ=100)	0.89891	0.325024	0.248427
LASSO (λ=0.1)	0.885788	0.345476	0.26879
KNN (k = 10)	0.912608	0.296642	0.222823
KRR (λ=0.1)	0.942434	0.240778	0.179977
RF (n=34)	0.920658	0.29258	0.205656
SVR (C=500)	0.943175	0.240508	0.180166

The plot that corresponds to the results of this table is the following:



The conclusions of these results are the following:

- **The best method is SVR** it has the highest R squared, and the lowest RMSE and MAE. Its complexity makes it a good estimation method which can achieve better fits than the other methods explained during the report.
- **KRR has similar results as SVR** due that the basis of both algorithms is the same. Both use RBF kernel to get the results.
- The simple linear regression and the regularized regression methods have the overall worse results. This happens because those methods are simple compared to the other ones in the project.
- **The best method amongst the MLR and MLR with regularization is simple MLR with the subsect selection.** This is because all RR and LASSO have a penalization factor that makes them worse in the final model but better in the variance between the estimations. Also, LASSO has a more aggressive penalization and that is why it is the one with the worst results.
- Also, **RF and KNN have similar results and the best of them is RF** because in a more randomized way so can learn more significant differences between the data points.

2. Fourth Project

2.1. Introduction

In this project a 3 layered FNN (Feedforward Neural Network) that performs binary classification is tasted to understand how it works. The input of this neural network is a three-component vector. One of the components of this vector is fixed to 1 to add bias in the first stage. The result of the computation of the neural network is a probability of being in one of the two groups (0 or 1). The changes in the neural network weights applying gradient descent method with back-propagation with a momentum of value α. The learning rate parameter remains constant during all the executions.

The neural network used in this project performs two steps:

1. Forward propagation: The first step of the neural network is to calculate the outputs for the inputs, the weights of the neurons and the active functions of each one of them. It is called forward propagation because this is done from the input layer to the output layer.
2. Backward Propagation: The second step begins from the output layer and moves along the neural network up to the input layer (goes backwards) performing the gradients of each layer. This step is costly in terms of computational cost due that each parameter must go through the gradient calculation and each layer needs the gradient values of the former layer. This is called a sequential computational model.

The activation functions used in the neural network for the predictions of the output are nonlinear sigmoid functions and for the hidden layer they are Rectified Linear activation Units.

2.1.1. Parameters

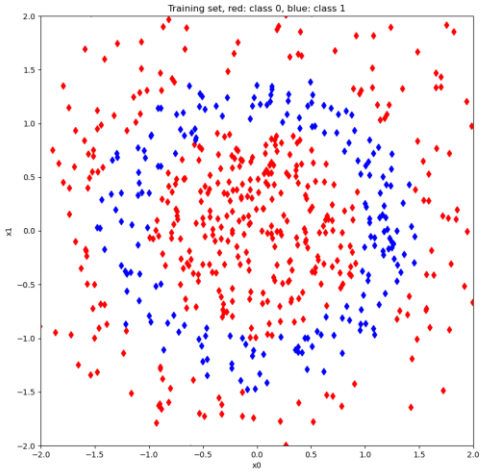
The neural network parameters used during the following sections are:

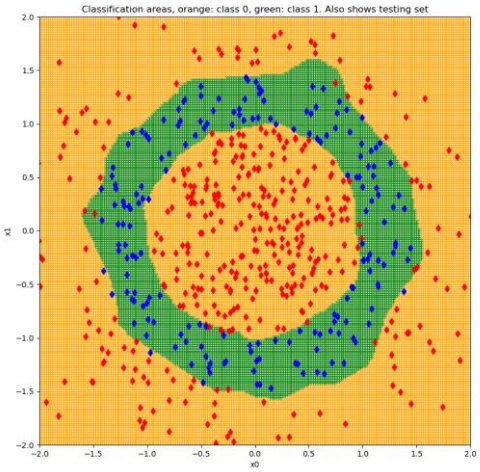
- T: Batch size of the training set and the test set. Both are generated using a normal distribution.
- H: Stands for the number of hidden dimensions in the Neural Network.
- Learning Rate η: Sets the speed of the gradient descent method movement towards the gradient.
- Momentum α: Amount of inertia speed conserved from the previous gradient descent step.

2.1.2. Initial execution

If the initial values of the given code are not changed, the results of the execution are the ones reviewed in this section. This serves as a starting point for the analysis conducted in this second part of the report.

The initial values of the code are: $T = 640,\ H = 20,\ \eta = 1e^{-4},\ \alpha = 0.7$





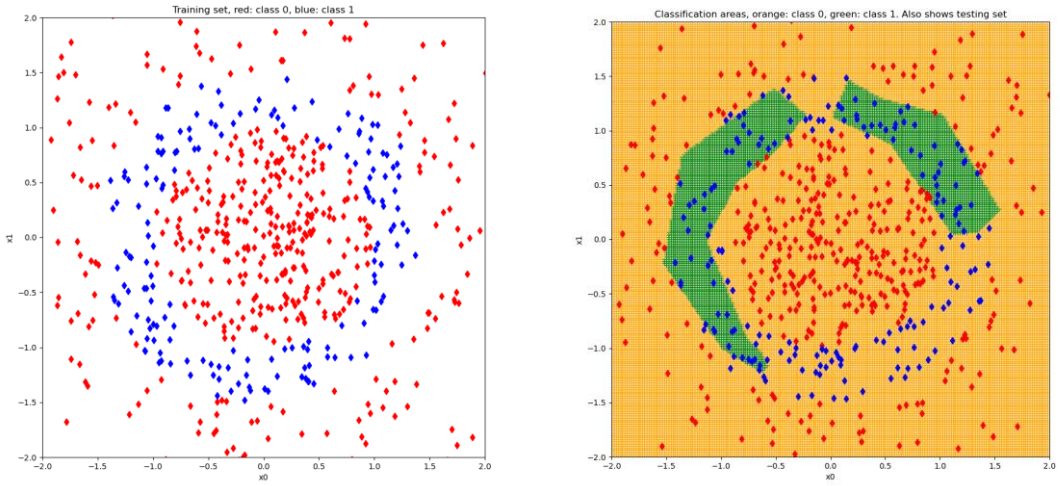
Training set points (left) and resulting model at iteration 2000. The training data model already classifies the points in a good way, we can say that we have low training loss. Let's look to the test and training loss functions.

The model has a good fit due that the values of the loss functions are small. Also, the values of bot loss functions are similar so the model has low variance.

2.2. Underfitting

An under fitted model is one that is far from the true relationship. When the model is created with the training data, it is not sensible enough to get all the significative complexities of the data and, thus, it ends doing oversimplified predictions that are far from good. Also underfitted models usually have high bias but low variance since all the results from the model are consistently (low variance) bad (bias between the result and the truth).

The results when we force underfitting, i.e., $T = 640$, $H = 7$, η=1e^{-4}, α=0.7 are the following:



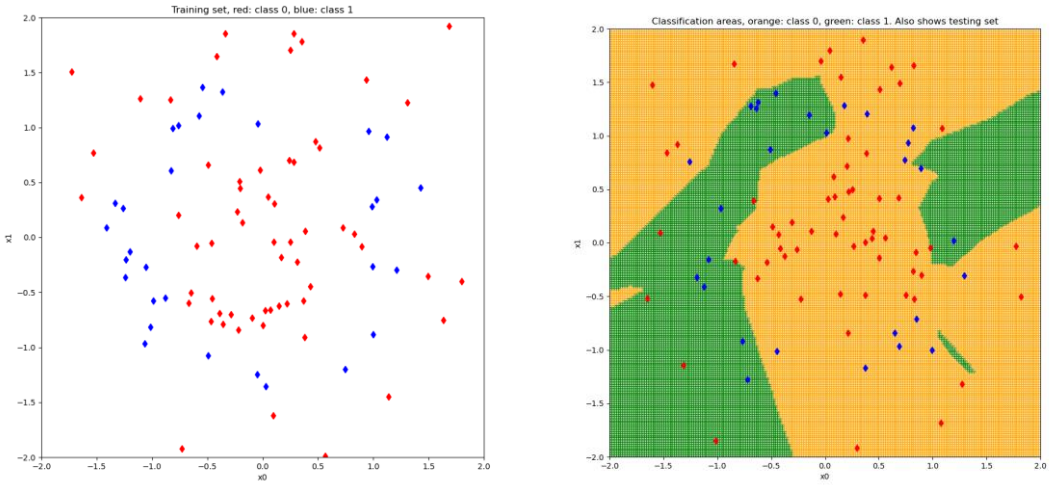
As we clearly can see, the model is underfitting. The training set is not fitted, but if we were to place the generated model over the training data it also would not fit. This also is reflected on the loss functions plot.

Both loss functions end up having worse values than in the first execution. Also, the difference between the loss functions is greater than in the previous execution.

2.3. Overfitting

An overfitted model is one that has a high complexity and perfectly fits the training set but is incapable of fitting the test set or new data that is not used to train. This happens when the model is to complex in comparison to the training data used.

A way of forcing overfitting is to input a small training set. If we set the batch to 100 and we return to the H=20 of the first execution, we have these results:



The model is not fitting properly because the data is not enough to get the complexity that the model requires to solve this problem. The training data is well classified in comparison to the test data which is erratically clasiffied. The loss functions plot demonstrates that this is a worse situation than the hidden layer number modification (underfitting).

The loss is greater and the distance between the set results is also greater. This underfitting case is worse. The training data loss rate (red) is much smaller that the one of the test data (blue). This happens because the model is good at predicting the training data but as mentioned before, is not able to predict new data that enters in the neural network and that has not been used for the training/learning phase.

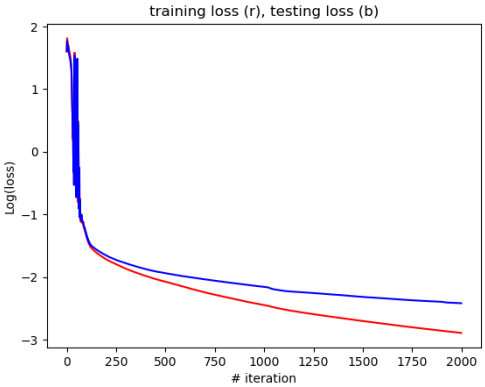
2.4. Finding the right values of H

Fixing the value of T to the 640 that we had at the beginning and modifying the valuer of H we can get to the optimal H value. The objective is to get a value of H higher than the underfitting scenario but lower that a overfitting scenario.

Changing the learning rate to a lower one makes weights of the neural network change. This makes that the loss functions loose correction and increases the computational time since the steps are smaller and the model is reached slower. Lowering too much this parameter can lead to a underfitting scenario. If the modification is to increase, it and it is done too much the scenario can be overfitting. Both things happen because that is the reaction of the gradient descent method to changes in the learning rate, so that would affect to the GD solution and thus, to the final neural network. The best value of this parameter is the one that is given at the beginning of the project (η=1e^{-4}).

The momentum value in GD helps to avoid the oscillations of a noisy gradient and the coast across flat spots of the search space. It changes the importance of the inertia speed of the gradient descent. For this section the value has been maintained to the one at the beginning (α=0.7).

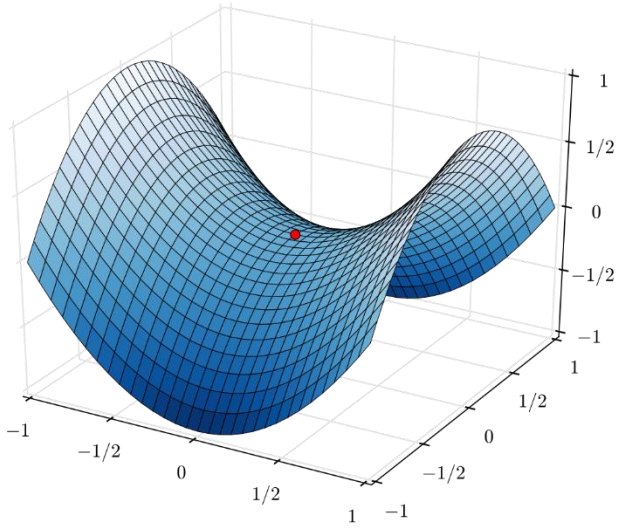
With an H of 45 the loss function of both test data and training data is lowered. Next plots are the loss functions of the first execution (right) and the best H model (right), which has less loss function error.



2.5. Problems with the gradient

If the value of H is increased to, for example, 90, the results of the model are the following in terms of the loss function:

The neural network starts trying to fit the model but suddenly it stops at a value of loss function and never changes again. This m3eans that the model is not improving (lot learning) from one iteration to the next one. This also means that the gradients calculated in the back propagation `has e are all equal to 0 and the weights are not updated. I guess this happens because the point found as minimum is a saddle point due to the complexity of the neural network. A saddle point is a trouble for gradients since the gradients fall into a minimum that at the same time is a maximum. Escaping from this point is difficult and the gradient keeps falling into it because e mathematically is the best point found at the time. The next plot represents a saddle point.



2.6. Conclusions

2.6.1. Conclusions on the initial execution

To get better models the initial parameters must be changed. In this kind of Machine Learning problems there is not a single neural network that works well. The best fit is obtained by changing the parameters executing, analyzing, and then iterating over the process. Also, the objective always is to avoid underfitting and overfitting.

2.6.2. Conclusions on underfitting

Decreasing the size of the hidden layer simplifies the model. This loss in flexibility of the model, translates into underfitting because the model is not able to get good predictions. In our case this happened with H = 7 and maintaining all the other parameters as default (as in the first execution).

2.6.3. Conclusions on underfitting

Also, making changes in the size of the training and test sets is dangerous. If we aggressively change the batch size the model can overfit as we saw. This happens because with less training data the model lacks the needed knowledge and learning pace that it has with greater datasets. In neural networks the greater the number of data, the better the results and the training of the model will be if the complexity of the model fits the amount of data needed.

An overfitting scenario can be solved lowering the H value for low data samples, using regularization (dropping out some neurons in a randomized way and then decreasing the complexity of the neural network) and stopping the learning step before the overfitting scenario happens.

2.6.4. Conclusions on the problems with the gradient

When increasing the complexity of the neural network, we have found a saddle point situation. In a saddle point the GD thinks that has found the minimum but at the same time it has found a maximum. This makes that the following gradients are 0 and that the weights are not updated, thus, the final loss function plot has no improvements since the moment GD finds the saddle point. A good method to escape from a simple saddle point is to use Stochastic Gradient Descent method. It has the capability of escaping the flat part of the saddle point and get into the path of the minimum.