SL_HW1_part1

2023-05-12

- Giuseppe Di Poce : 2072371
- Enrico Grimaldi : 1884443
- Davide Vigneri : 2058036
- Nicola Grieco : 2081607

Question 1

Why this idea is yet another manifestation of our “be linear in transformed feature space” mantra. Technical difference with the “orthogonal series expansion” point of view.

Linear models are widely used because they are rather “simple” and at the same time **highly interpretable**. In fact, the linearity of such models refers to the relationship between *parameters* and *target* variable: the target variable (y) can be defined as a linear combination of other variables called *predictors*. However, the simple *linear regression* case can be relaxed by extending the linearity of the model by introducing new *predictors* defined as a transformation (e.g., power elevation) of the old *predictors*.
What we therefore pursue is the selection of a model that can more accurately approximate the true function to be estimated: the complexity of the model increases **without improperly affecting the interpretability of the parameters** and results obtained.
By summarizing with **regression splines** we map the input features in a different space that allows us to better capture the trend of the function underlying the observed data, using a model that is still linear.
In the case of the most basic *polynomial regression* we intend to approximate the target function to its **truncated power expansion**: the result will be a linear combination of basis functions that are polynomial functions of the predictor variable.
This set of monomial functions constitutes a particular design matrix called **Vandermonde matrix**: its columns represent a basis for the relative degree polynomial function (they are linearly independent).
We can now easily show that such expansion **does not constitute an orthogonal basis** since the inner product of any two of these basis functions is not necessarily zero. Consider the monomial basis functions:

$$\begin{cases} \phi_1(x) = 1 \\ \phi_2(x) = x \\ \phi_3(x) = x^2 \\ \dots \\ \phi_d(x) = x^{(d-1)} \end{cases}$$

Then we have that in general:

$$\langle \phi_i, \phi_j \rangle = \int_a^b x^{(i-1)} \cdot x^{(j-1)}\, dx = \left[\frac{x^{i+j-1}}{(i+j-1)} \right]_a^b \neq 0, \quad \forall i \neq j$$

The (natural) **spline regression** model can be thought of as a smoother version of the polynomial regression model. To the previous monomial basis of order d we add a truncated power basis **for each knot** ξ_i defined as follows:

$$h_i(x, \xi_i) = \begin{cases} (x-\xi_i)^d\ , & x > \xi_i \\ 0 & , \ otherwise \end{cases}$$

We have previously shown that power expansion (for polynomial regression) unfortunately is not an orthogonal series expansion and consequently this new derived basis also does not possess the property of orthogonality.

To the actual absence of orthogonality in the **Vandermonde matrix** above we must moreover impute a difference between the monomial expansion coefficients and the new coefficients (the optimal parameters) in the spline basis representation: by including the new h_i functions we find a better (smoother) fit that satisfy certain differentiability conditions, but this will involve in general different optimal values for the coefficients we already evaluated in the polynomial regression setup and the first $d+1$ coefficients in the spline regression setup (while in an orthogonal context like **Fourier expansion** we would only have to compute the coefficients for the new additional functions we decided to include in the approximation).

In summary, the studied expansion does not typically result in an orthonormal basis, but it is possible to construct an orthonormal basis using polynomial functions, such as the **Legendre polynomials**.

Question 2

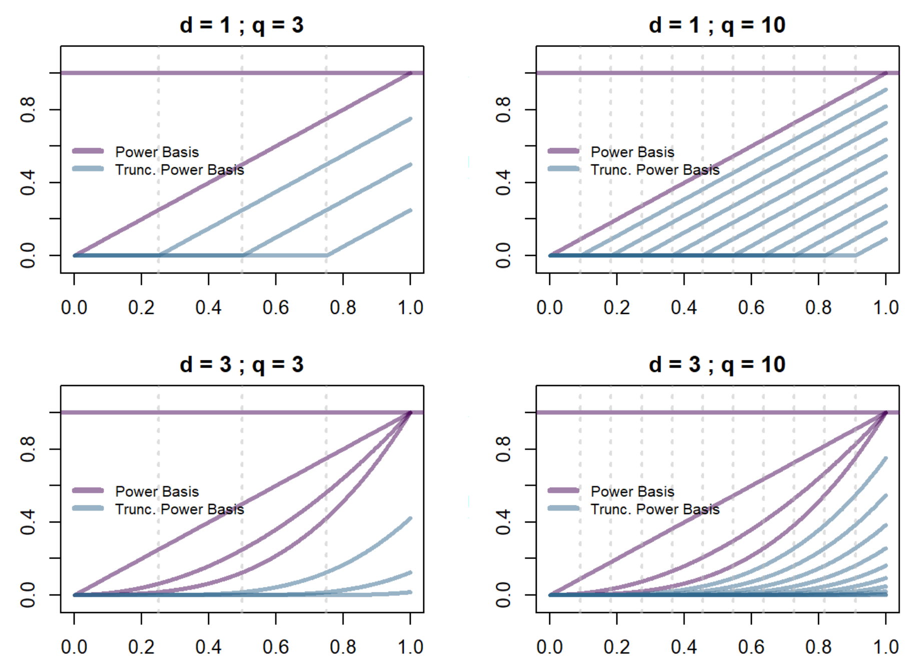
Developing our own implementation of the truncated power basis

The basis through which we want to represent the **Natural spline** is implemented as the combination of two parts:

> - the power expansion $\beta_0 + \sum_{i=1}^{d} \beta_i x^i$
> - the truncated power expansion $\sum_{i=1}^{q} max(0, x-k_i)^d$, where k_i is the $i-th$ knot

The coefficients of the basis $\{\beta\}_{i=0}^{d+q+1}$ will be tuned fitting our spline model to the training set.

Plot the example truncated power basis for each combo of d and q



Use ChatGPT (or any other chat-bot capable of coding) to achieve the same task. Report the sequence of prompts you tried and the bot's replies. How would you evaluate the quality of this tiny bot-based experience?

PDF File

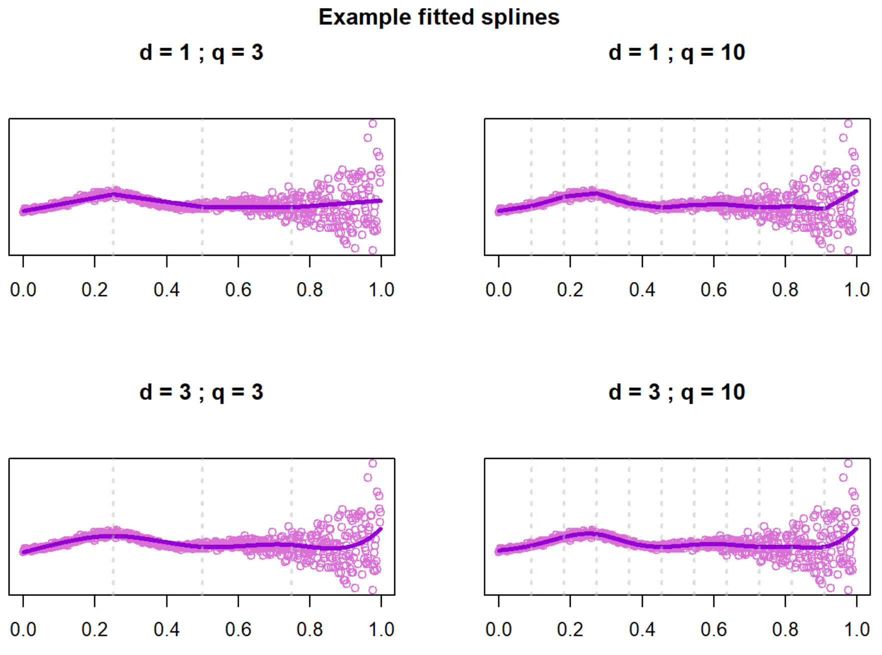
Although the chat-box accomplished the task, we cannot neglect the fact that we have been very precise in describing the steps to perform. Nevertheless, in the first attempt, ChatGPT misinterpreted the construction of the design matrix and provided us with an incorrect solution. As an AI language model, Chat has showed us that it does not have the ability to perform tasks on its own and it needs a carefull supervision. Anyhow, starting from its code, we arrived at the same results as ours. Of course, these claims have to been made considering the quality of the query that we wrote that was spot on and, in a certain sense, easy to interpret by a machine. The quality of the response provided by Chat would have been very different (and we imagine 'suboptimal') if we had limited ourselves to report the full content of the text of the homework.

Question 3

Use truncated power basis defined above to implement regression splines from scratch on the "wmap" data

Plot a few elements with $d \in \{1, 3\}$ and $q \in \{3, 10\}$ equispaced knots in the open interval (0, 1).

Previously we plotted TPB for certain example values of d and q. We then decide to report the plots of the splines corresponding to the bases above: these spline functions are in fact linear combinations of the bases themselves and the coefficients are appropriately computed by optimization algorithms implemented in the associated models (in this case 'lm').



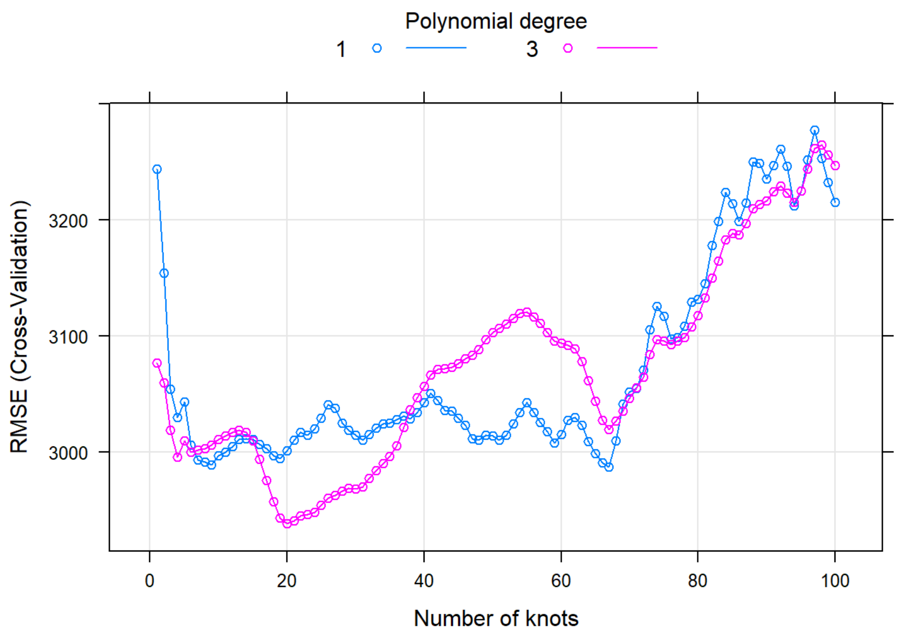
Defining the structure of our model commprehensive of fit and predict functions in order to fastly implement tuning techniques and Grid search CV

To define the structure we decide to use a list with named elements such as:

> - type of the model;
> - labeled grid of possible parameters;
> - custom “fit” and “predict” functions;
> - dynamic expansion of the design matrix given a new feature vector.

Setup the cross-validation we want to use and its hyperparameters with a specific method

First method - with **lm model** - for parameters tuning, using a simple **K-Fold cross-validation**

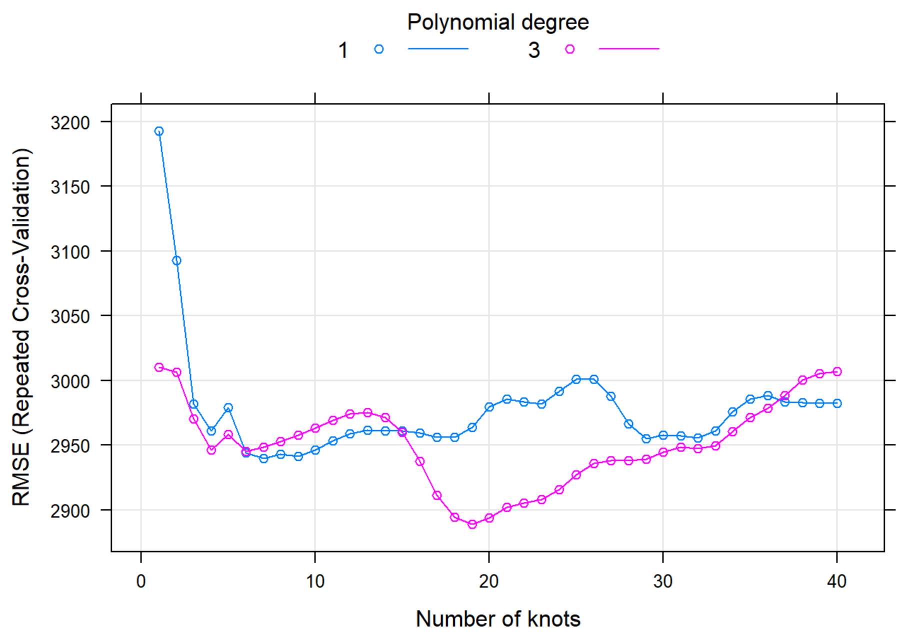


Looking at the graph, the orange line, representing a 3-degree polynomial, reaches the global bottom in correspondence of q=20. This bottom point is never touched by the blue line, representing a 1-degree polynomial, that remains way above this point of interest. Therefore, for now, the optimal ‘d’ and ‘q’ are respectively 3 and 20.

Report the best selected model - **min. RMSE**

```
##     d  q     RMSE Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 120 3 20 2938.178 0.208711 1487.336 529.3799  0.1338805 177.8535
```

Now, we restrict the grid of number of knots and contextually we change the type of cross validation, to wit we introduce the *Repeated cross validation* to get a more accurate result. We will perform 10-folds cross validation with 3 repetitions.

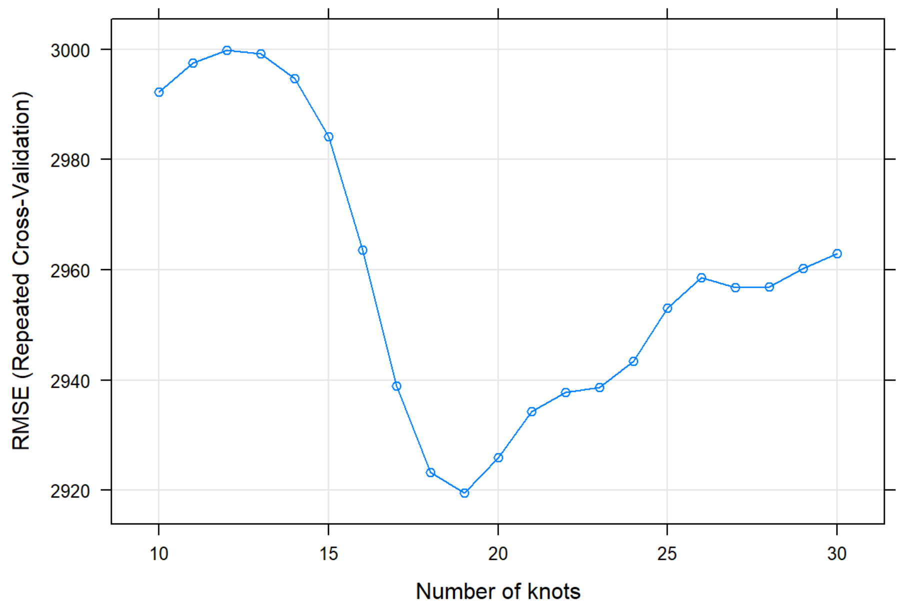


This graph allows us to have a more granular view of the optimal number of knots. Furthermore, the repeated cross validation with 10 folds spits out that the number of knots that minimizes the RMSE is 19. The optimal polynomial degree is once again 3.

Report the best selected model (min. RMSE)

```
##    d  q     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 59 3 19 2888.731 0.2226444 1485.054 791.2423   0.159512 350.1433
```

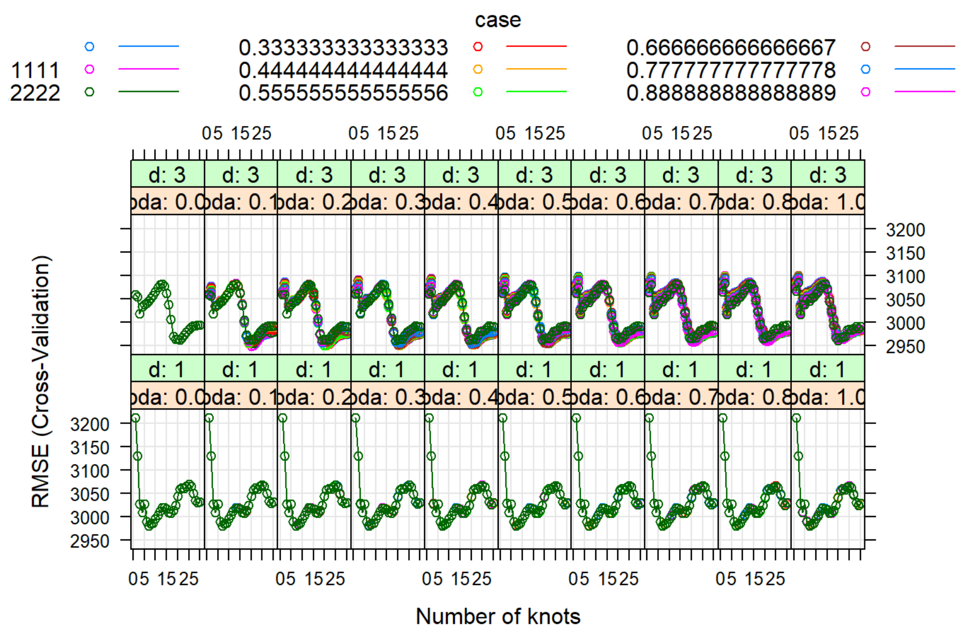
After these results we have gained enough confidence to claim that the best model is obtained with degree equals to 3. For this reason, from now on we restrict the grid-search only on the degree 3. Now, to gain more confidence on ‘q’, we increase the number of repetitions of the k-fold (10 folds 10 repetitions) cross validation and also we restrict the grid of q (from 10 to 30 knots). From the last analysis q is changed, the best model is again obtained with the degree equals to 3, so we restrict the Grid Search only on the degree 3. We also increase the number of repetitions of the k-fold cross validation and also we restrict the grid of q.



From the last plot, nothing seems to be changed and so we get that the optimal number of knots is 19.

```
##    d  q     RMSE  Rsquared     MAE   RMSESD RsquaredSD    MAESD
## 10 3 19 2919.471 0.2124042 1491.51 677.5285  0.1372001 288.1561
```

Until this moment we use only the linear model without penalization. We try to put in the game the **penalization** terms using **glmnet** (introduce the **elastic net**). As a first run, we use a 10 fold cross validation as we did for the ‘lm’ model. In addition to ‘d’ and ‘q’ we grid-search also ‘alpha’, the parameter that defines the type of regularization , and ‘lambda’, the parameter that controls the amount of regularization applied.



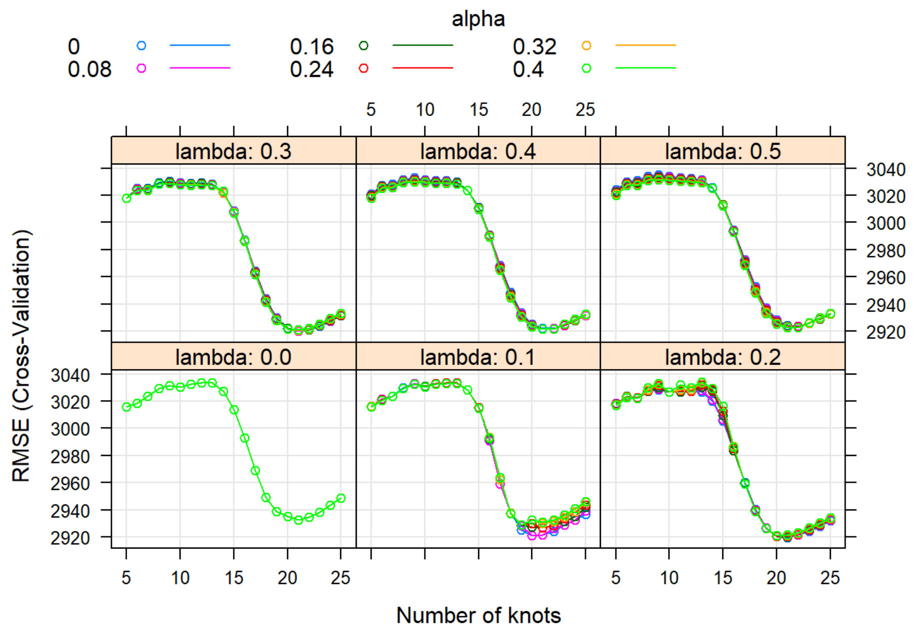
This image can be intimidating but is the very first starting point toward our research of the optimal hyperparameters. For d=3, we have 10 different panels for 10 different values of lambda. Inside each panel, we may find how the alpha parameters affect the RMSE of the model. For d=1, we basically have 10 identical plots because the penalization does not have any effect when the polynomial is of first degree.

Best model for this fourth analysis

```
##      d  q alpha    lambda     RMSE  Rsquared      MAE   RMSESD RsquaredSD
## 4902 3 20     0 0.1111111 2948.944 0.1677909 1609.955 684.8652  0.0522892
##         MAESD
## 4902 285.1965
```

We see that, as expected, the optimal d is 3 and, as before, the optimal q is 20. We also find that the optimal alpha is 0, the Ridge penalty, and the optimal lambda is 0.11. We now proceed as before by restricting the Grid Search of our hyperparameters, focusing our attention on models with d=3.

We can restrict the Grid search and obtain also a better plot to interpret:

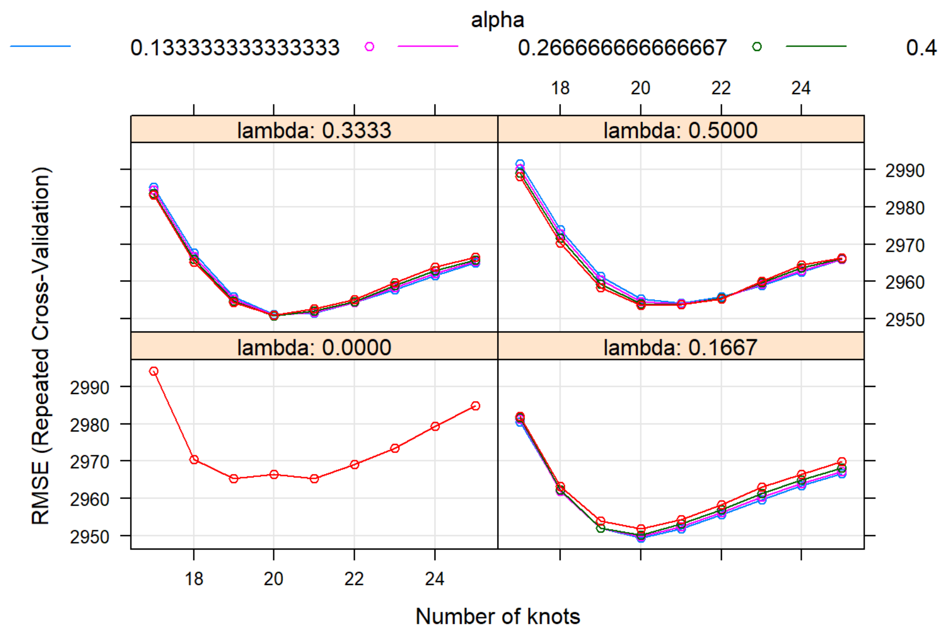


The graphs show that the penalization does not seem to be a big game changer at least when lambda is different from 0. So, it seems that we need to apply a penalization of modest intensity with alpha whatever.

```
##     d  q alpha lambda     RMSE  Rsquared     MAE   RMSESD RsquaredSD    MAESD
## 579 3 21     0    0.2 2919.621 0.2080964 1603.47 621.3471  0.1502036 244.7857
```

Going more deeply into the results, we find that the optimal q is 21, alpha is 0 and lambda 0.2.

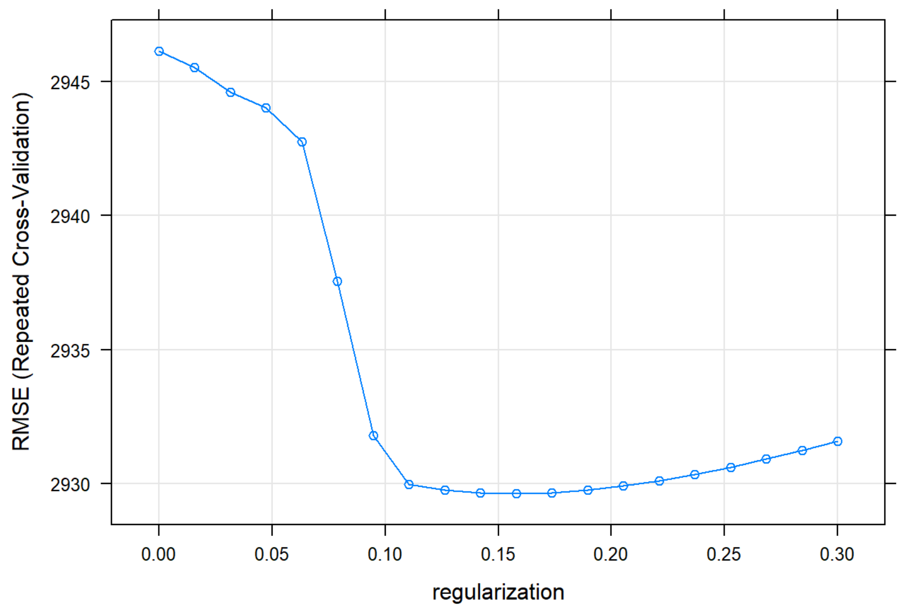
Finally, we keep on restricting the grid and we do the repeated k-fold validation with 10 repetitions to obtain a more reliable result.



The plot seems to confirm the suggestions of the previous one: lambda has to be small but greater than 0 and less than 0.5. In addition, now we have a more nitid information on alpha: it has to be less than 0.4.

```
##    d  q alpha    lambda     RMSE Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 50 3 20     0 0.1666667 2949.469  0.18832 1611.963 604.6558   0.121108 235.4835
```

The table reports that the optimal choices are d=3, q=20, alpha=0, and lambda=0.16. After these last results, we decided to only search on the best lambda fixing the other hyperparameters: d = 3, q=20, and alpha = 0.

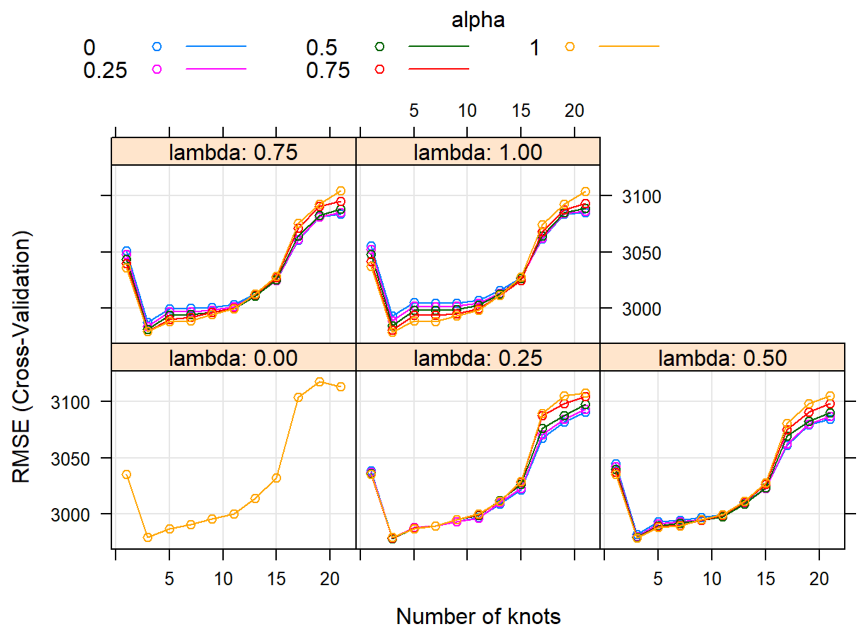


The graph suggest that the optimal lambda has to be inside the interval 0.11-0.2.

```
##    d  q alpha    lambda     RMSE  Rsquared     MAE   RMSESD RsquaredSD    MAESD
## 11 3 20     0 0.1578947 2929.623 0.2028608 1610.07 682.3814  0.1394808 263.7985
```

The table gives us the detail that we are looking for: lambda is close to 0.157.

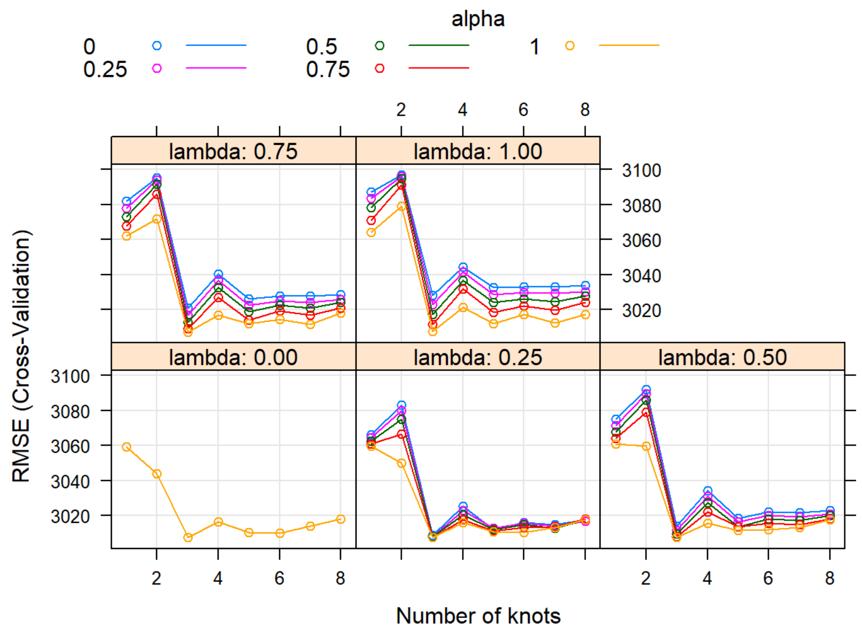
Until now we didn't use any type of pre-processing because as first look our predictor data are in the range of [0,1] and there is no need to standardize it. But, after the feature engineering through the method of the truncated basis we can notice that the variables have very different scales and we know that standardization can bring them to a similar scale. This is important for models that are sensitive to the scale of the variables, such as, for example, the linear regression. Standardization ensures that no single variable dominates the model simply because of its larger scale. So we have implementing a method that goes to standardize the matrix of X before estimating the model with the glmnet function, and we always see through cross validation what is the best model that we are then going to use for our predictions on the test set.



The graphs show that the type of penalization does not seem to be a big game changer. There are very little differences between in using a model closer to lasso or ridge. We can also see that rmse starts to rise a lot as knots increase and already from 13/15 knots it grows quite fast. Let's restrict the grid of knots on which we are searching the best solution in such a way to use a grid for q that increases one to one.

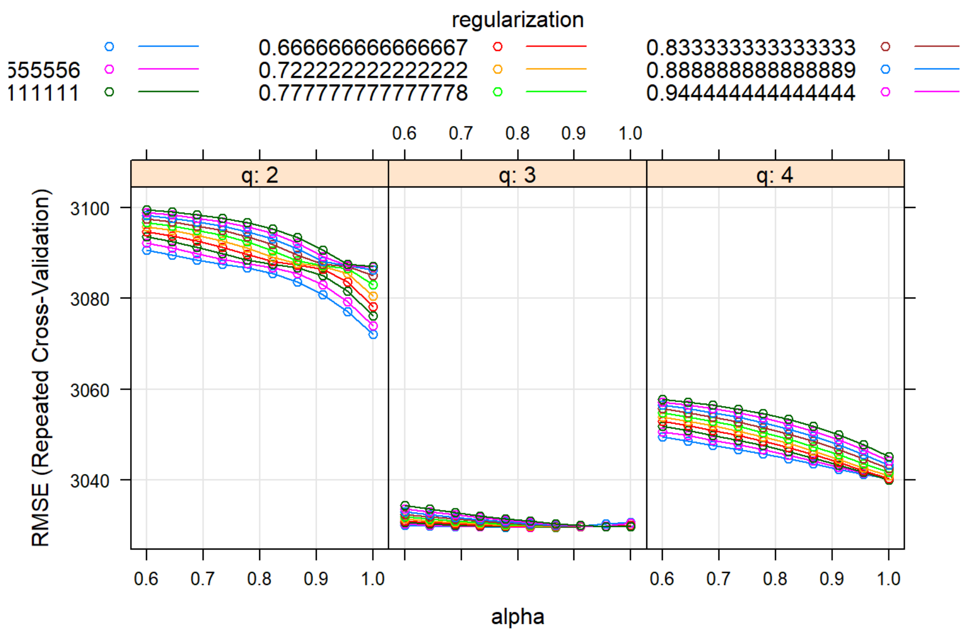
```
##    d q alpha lambda    RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 37 3 3   0.5   0.25 2978.31 0.1507758 1735.625 697.5704 0.08000598 347.1749
```

Now we restrict the grid on q and we search for the values of alpha e lambda



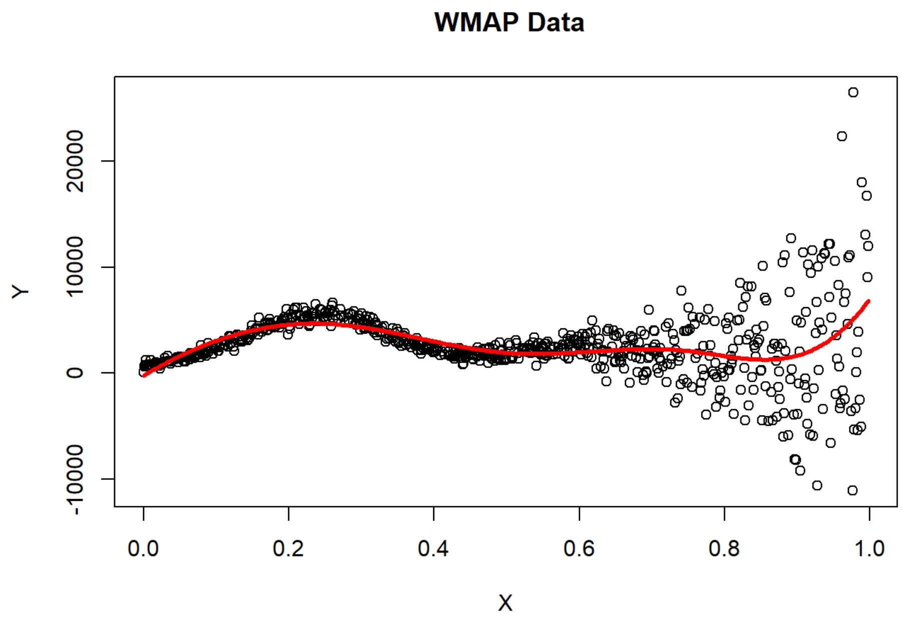
```
##    d q alpha lambda     RMSE  Rsquared     MAE   RMSESD RsquaredSD    MAESD
## 74 3 3     1   0.75 3007.212 0.1633703 1714.82 498.3382  0.1247712 282.1245
```

It is evident that the model with the lowest rmse is with 3 knots, so let us proceed by performing the repeated 10 fold cross validation to obtain a more stable result by repeating the whole procedure for 10 times.



```
##     d q     alpha lambda     RMSE  Rsquared      MAE  RMSESD RsquaredSD
## 151 3 3 0.8222222    0.5 3029.608 0.1448172 1718.319 549.598 0.09998392
##        MAESD
## 151 266.0903
```

Finally, we have confirmation that the model with the lowest rmse is the one generated using 3 degrees, 3 knots, alpha = 0.82222 and lambda = 0.5. The fact that the alpha is so high makes us realize that it is better to use an elastic net model that is closer to the L-1 solution and that we therefore prefer a model that brings the coefficients associated with the predictors completely to 0.



This is how our model fits on the data.

Bates-Hastie-Tibshirani Nested Cross-Validation implementation from from scratch

```r
## we show in the html the code of the implementation since it was explicitly requested

# create k folds given the training set
makeFolds<-function(training,K){
  folds<-createFolds(training$y,k = K, returnTrain = FALSE,list = T)
  folder <- lapply(folds, function(idx) training[idx, ])
  return(folder)
}

#Nested Cross-V (outer function)
nested.cv2<-function(training,R,K,d,q, alpha, lambda){
  es <- c()
  a.list<- c()
  b.list<- c()
  #outer cv
  for (r in 1:R){
    folds <- makeFolds(training,K)

    for (k in 1:K){
      # take all the folds except the k-th one
      training.folds <- folds[-k]     ## list of dataframe

      holdout.fold <- folds[[k]]

      # prediction error estimates with the inner-cv on the training set
      e.in<- inner.cv2(training.folds, d, q,alpha,lambda)

      # evaluate coefficients on the training folds
      training.folds <- bind_rows(folds[-k])      ## map list of dataframes in a unique dataframe

      model <- my_spline3(training.folds, d, q, alpha, lambda)
      coeffs <- model$beta

      # evaluate the loss (in terms of residuals) on the hold-out set
      predictions <- predict(model, as.matrix(build_dm(holdout.fold$x,d,q)), type="response")
      e.out <- (predictions - holdout.fold$y)^2  # error on the k-th hold-out set

      # populate our vectors for final aggregation of the results
      temp1 <- (mean(e.in) - mean(e.out))^2
      a.list <- c(a.list, temp1)

      temp2 <- var(e.out)/length(holdout.fold$x)
      b.list <- c(b.list, temp2)

      es <- c(es, e.in)
    }
  }
  rmse <- sqrt(mean(a.list) - mean(b.list))
  err.ncv <- mean(es)
  return(list("rmse"=rmse, "err.ncv"=err.ncv))#, "es"=es))
}

#inner loop
inner.cv2 <- function(folds, d, q, alpha, lambda){
  k <- length(folds) # number of folds from the previous partitioning
  # init the losses for each fold set
  e.in <- c()
  for(i in 1:k){
    # leave one set out for validation
    training.set <- bind_rows(folds[-i])
    holdout.set <- folds[[i]]
 # evaluate the coefficients on the new training set
    model <- my_spline3(training.set, d, q, alpha, lambda)
    coeffs <- model$beta

    # evaluate the loss (in terms of residuals) on the new hold-out set
    predictions <- predict(model, as.matrix(build_dm(holdout.set$x,d,q)), type="response")
    true.vals <- holdout.set$y
    # define the loss between the predicted value and the true value as the squared difference
    temp <- (predictions - true.vals)^2
    e.in <- c(e.in, temp) # add the error on the i-th hold-out set
  }
  return(e.in)
}
```

What are the issues that we try to solve with this implementation of the Nested CV ?

Basically, what the implementation of a nested cross-validation (Ncv) wants to do, as well as a vanilla cross-validation, is to evaluate the performance of a machine learning algorithm, and tuning $hyperparameters$ (H) is used to find the best set of H for that specific algorithm we are dealing with. Without this type of approach, the same data is used to tune the model parameters and evaluate its performance, which can lead to a biased evaluation of the model. One basic difference between the two types of CV is that in the vanilla approach the model is trained on the training set and evaluated on the hold out set. This process is repeated several times, with different splits of the data, to obtain the best possible estimate of the model's performance.

Let's define the following quantities:

> - Err_{XY} : the error of the model that was fit on our actual training set
> - Err: the average error of the fitting algorithm run on same-sized data sets drawn from the underlying distribution P … (average error over XY - training set)
> - $Err_X := E[Err_{XY}|X]$: average error over Y

Problems of classic CV:

Cross-validation estimate provides little information about Err_{XY} given a phenomenon called *weak correlation issue*. CV has lower MSE for estimating Err than it does for Err_{XY} in the special case of the linear model. The point is that CV should be viewed as an estimate of Err rather than of Err_{XY} and it is pretty useful to generalize to other unobserved data (in particular the K-Fold CV) but makes a poor estimate of the real prediction error.

The main conclusion is that a *linearly invariant estimate* of the prediction error that has precision $1/\sqrt{n}$, such as $\hat{Err}^{(CV)}$, it has (asymptotically) lower estimation error when estimating Err compared to Err_{XY}. Similarly, the correlation between a linearly invariant estimate and Err_{XY} goes to zero. Thus, cross-validation is estimating the average error Err more so than the specific error $ErrXY$. Similarly the Bates et al. (2022) (https://arxiv.org/pdf/2104.00673.pdf) claims that the correlation between a linearly invariant estimate and Err_{XY} goes to zero demonstrating that (assuming the homoskedastic Gaussian linear model holds and that we use squared-error loss) $\hat{Err}^{(CV)} \quad Err_{XY}|X$

What do we want to do with Nested CV?

The *nested cross-validation* essentially represents an expansion of Repeated cross-validation and well generalizes the prediction error. In fact if one looks solely at the inner loop of the algorithm this simply consists of a K-Fold CV performed on a smaller number of folds than the classical implementation, but repeated R times in the outer loop. The real novelty of the reported NCV implementation is to account for the variability of the prediction error and the poor level of correlation between mean error and point error on the training set.

Furthermore, the estimate of the variance of the classic CV point estimate assumes that the observed errors $e_1, \ldots, e_n$ are independent. This is not true: the observed errors have less information than an independent sample since each point is used for both training and testing, which induces dependence among these terms. This additional intuition also suggests that something beyond the usual cross-validation will be required to give good estimates of the standard error of $\hat{Err}^{(CV)}$.

With NCV we develop an estimator for the MSE of the cross-validation point estimate using the advantage offered by the *bootstrap simulation, then we use the estimated MSE to give **confidence intervals* for prediction error with approximately valid coverage.

This estimator of the MSE empirically estimates the variance of $\hat{Err}^{(CV)}$ across many subsamples. Avoiding the faulty independence approximation leads to intervals with superior coverage (remember that our problem with the linear models is that confidence intervals for Err_{XY} are larger than confidence intervals for Err)

Finally, how do we create confidence interval for the prediction error of our model?

We start by estimating the mean-squared error (MSE) of cross-validation: for a sample of size n split into K folds, the cross-validation MSE is

$$MSE_{K,n} := E\left[\left(\hat{Err}^{(CV)} - Err_{XY}\right)^2\right]$$

We can further decompose this estimate (using a little trick) and deriving bias and variance of the estimate from error quantities from our outer and inner loops of the algorithm.

We can view the MSE as a slightly conservative version of the variance of the cross-validation estimator from which we derive confidence intervals:

$$\left(\hat{Err}^{(NCV)} - \hat{bias} \pm z_{1-\alpha/2} \cdot \sqrt{\hat{MSE}}\right)$$

How can we use the NCV in our problem?

In our case we were able to fine tune the search for parameters and hyperparameters using Grid Searches based on Repeated/K_Fold CVs. Thus we were able to narrow down our search by at least excluding too high values of knots and spline degree less than third. Starting from this information we can now exploit our algorithm for the extremely computationally heavy Nested CV.

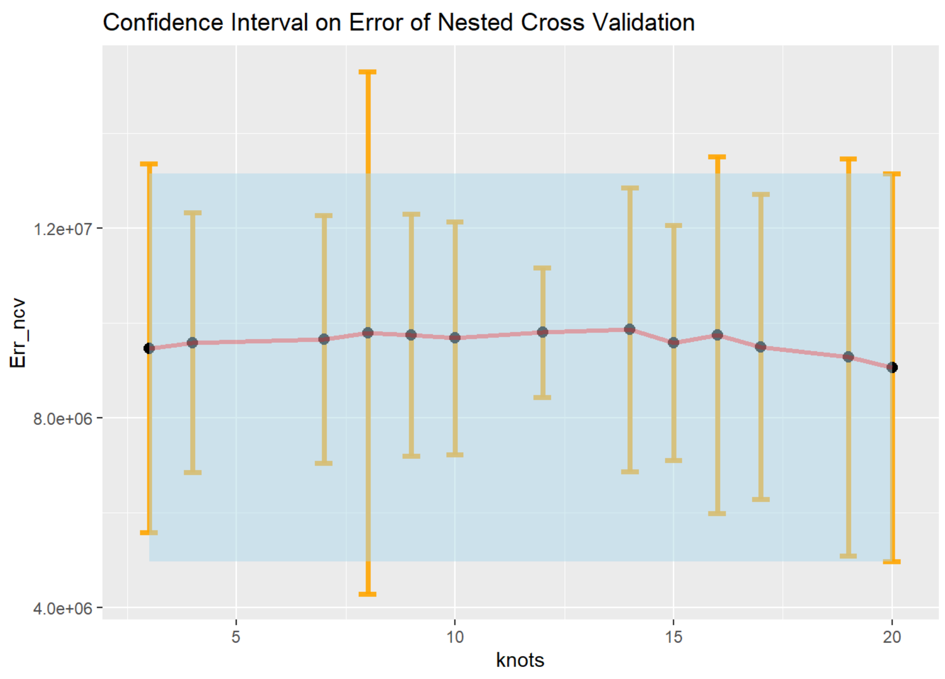
What we derive are rather similar point estimate values of the prediction error for given models, and the search results for the best models do not differ so much from previous cases hoping for a plausible generalization.

In this latter case, however, we have an additional piece of information: the confidence interval offered by the MSE estimates.

We then choose to proceed in the following way for the selection of the best model:

1. we start from the most efficient model in terms of predictive error detected by Grid Search with the NCV
2. we define the complexity of the model solely by the number of knots (since we assume that the degree of the best fit is 3)
3. considering the confidence interval defined on the optimal predictive error from step 1, we move to the simplest model (given the same number of knots and parameters related to the regularization effect)
4. select the latter model as the actual best model.

Note that in the procedure above we did not consider the calculation of the bias for the definition of the confidence interval.



As the Bates procedure suggests, we chose the best model by taking the $knot_i$ with the smallest Err_ncv as a reference point. Taking the error-derived confidence interval (RMSE) into account, we went on to choose the simplest model (in terms of number of knots) whose confidence interval falls entirely within the upper and lower bound of the reference error, i.e. the one that minimises Err_ncv.

Finally graphically and numerically compare the two fits on the training and also test data. Which one do you prefer? Explain.

```
##   Degree knots alpha lambda    RMSE Err_ncv    upper   lower
## 2      3     4     0    0.5 1400456 9589399 12334244 6844555
```

Numerically speaking, we can take into account the Err_{ncv} as the error of the model and compare our two best fit based on it.

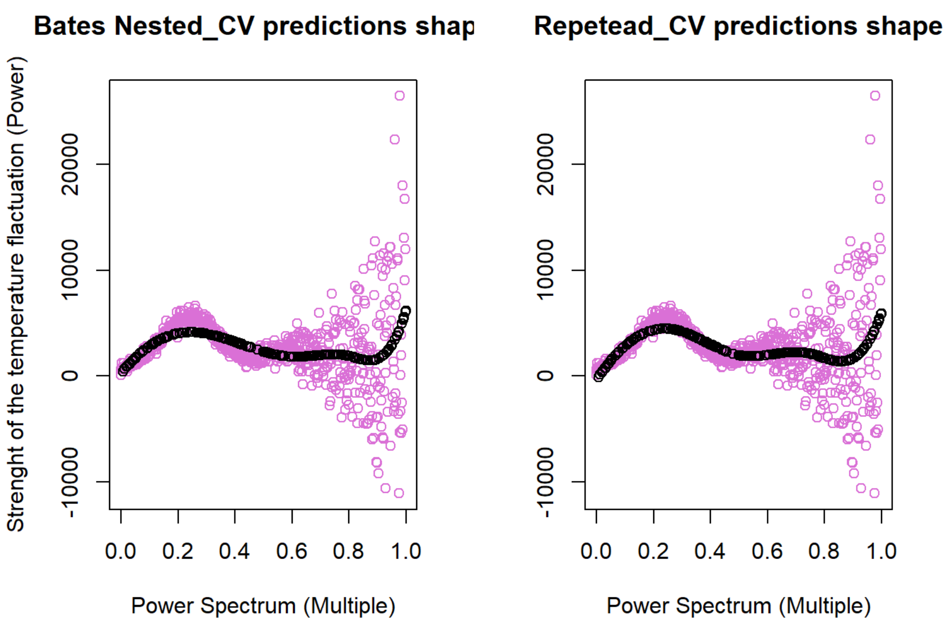
```
## [1] -102973.7
```

Based on the conclusions drawn during our analysis, we came up with our best models whose hyperparameters (H) result from two different procedures such as repeated-cv and nested-cv, which have been discussed at length above. Note that when varying R and K (number of folds) we will have slightly different results, but for the sake of simplicity our analysis has focused exclusively on the regularization parameters (and) and the number of knots q, setting the number of folds equal to 10 (that tend to select models less complex) and R=10 due to the fact that for smaller numbers of loops the Bates algorithm returns negative values. This is simply due to the fact that in

$$MSE \leftarrow mean(a_{list}) - mean(b_{list})$$

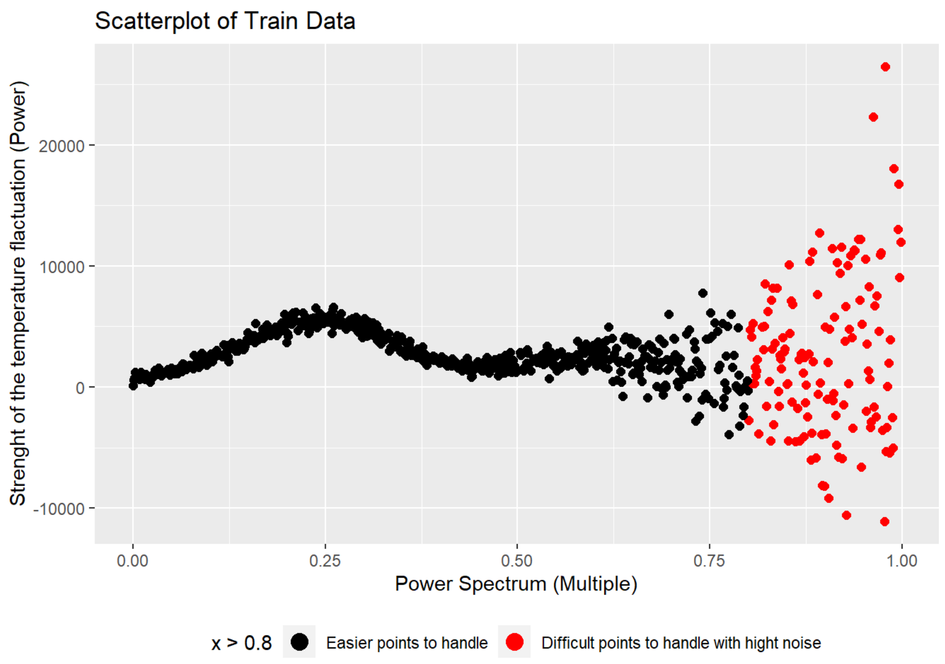
the right quantity $(var(e^{(out)})/|I_k|)$ may be greater than the left term.

In simplicity, our goal is to select by various validation methods the best values of the set of H such that our model is as generalisable as possible and, therefore, the least complex. In the specific case of the implementation of splines, the complexity depends on the number of knots and the combination of the parameters and : in our case we have decided to use a mixture method such as $elasticnet$ where corresponds to the value associated with the Lasso (L1 penalization, that takes the coefficient to zero, choosing simpler models) and to the Ridge (L2). The following scatterplot depicts the behavior of the predictions based on the two different splines models.



Thus taking into consideration the two terms (α and q) that determine complexity our choice of the best and most generalisable model will fall on the one generated by repeated cross validation with α equal to 0.899 and q=3. Specifically, this choice is due to the fact of trying to avoid $overfitting$ on the test set as much as possible.

Due to the $Heteroschedatics$ of our train data that we will discuss in a moment, our predictions may be affected by the big noise of the training data in the final part of the shape.The following plot explain at what data point we are referring.



To solve this issue we have tried to by removing outliers or by collapsing points with very large y values to their exponential mean.

By the way, until now, we do a pre-processing only on the predictors variables. But we can take a look to our data and we can notice something relevant:

The shape of the data resembles a funnel on the right side, it suggests a relationship between the predictor variable x and the response variable y known as heteroscedasticity. Specifically, it indicates that the variability of y tends to increase as the values of x increase. In other words the variability of y is not constant across the different levels or values of the predictor variable x. Heteroscedasticity can pose challenges in regression analysis because it violates the assumption of homoscedasticity, which assumes that the variability of the errors (residuals) is constant across all levels of the predictor variable. When heteroscedasticity is present, the standard errors of the regression coefficients may be biased, leading to incorrect inference and potentially misleading results. To cope with heteroscedasticity, we can apply the Box-Cox transformation to the response variable, y. This transformation can be useful from different perspectives:

- **Homoscedasticity**: By applying a suitable Box-Cox transformation, the transformation can stabilize the variance, making it more constant across the range of the predictor variable. This can result in improved estimates of standard errors, as the assumption of homoscedasticity is better satisfied.
- **Normality**: This transformation can also address non-normality in the data. Non-normality can affect the validity of statistical inference, including the estimation of standard errors. By transforming the data to achieve a more symmetric and normally distributed outcome, the assumptions of linear regression, such as normality of residuals, are better met. This can lead again to more accurate estimates of standard errors.

- **Outliers and influential observations**: This transformation can also help mitigate the impact of outliers and influential observations, which can distort the estimation of standard errors. By transforming the data, extreme observations may be pulled closer to the bulk of the data, reducing their influence on the estimates. This can result in more robust standard error estimates.

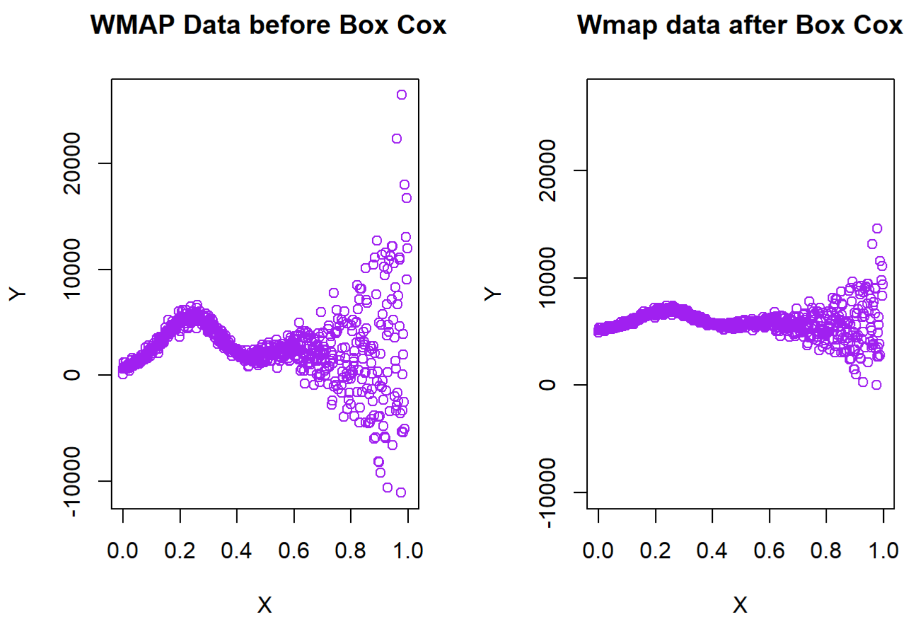
So we proceed applying the Box Cox trasformation:

Box Cox transformation

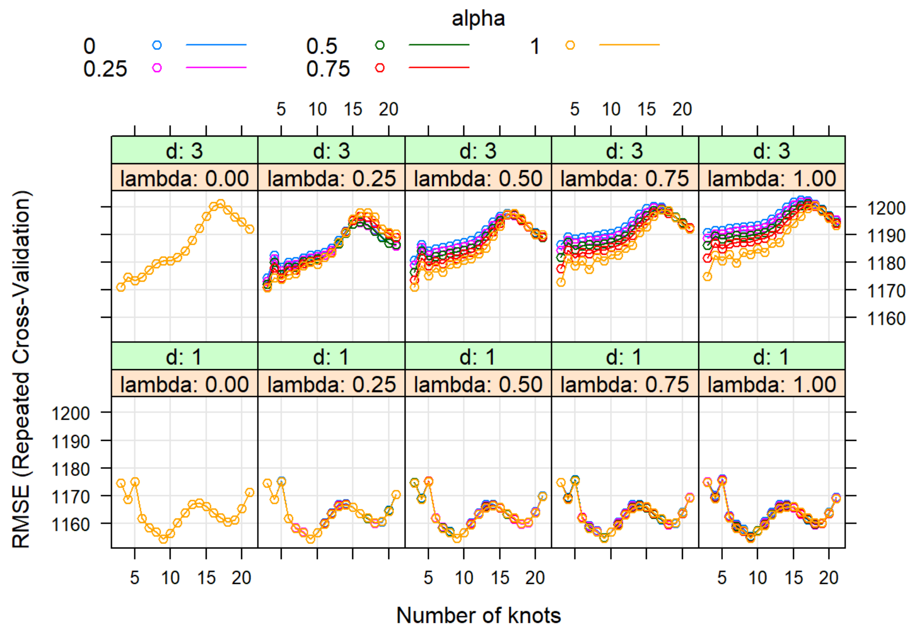
$$y(\lambda) = \begin{cases} \dfrac{(y+C)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(y+C) & \text{if } \lambda = 0 \end{cases}$$

- The data is adjusted for negative or zero values by adding a constant value C to shift the data. The constant C is determined by taking the absolute value of the minimum value in the data (y) and adding 1. $C = min(y) + 1$
- The λ value corresponding to the highest likelihood is identified using the “boxcox” function in R of the library MASS. This λ value represents the optimal λ for the Box-Cox transformation.

Let’s see the differences between the data before and after the transformation

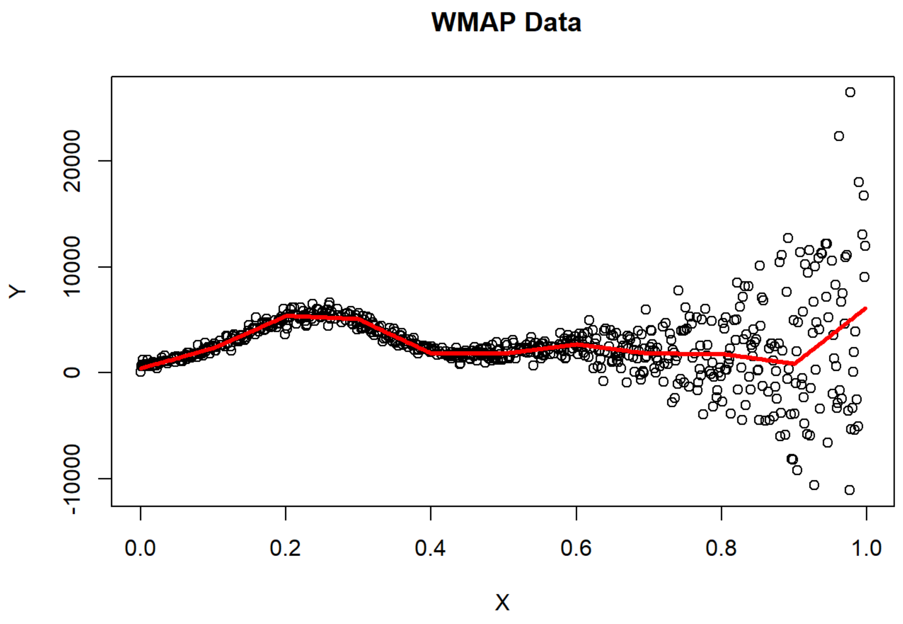


We can see that the box cox transformation has succeeded in limiting the variability in the data and in this way even the presence of outliers is handled, in what way controlled and will not affect the linear model negatively.



```
##      d q alpha lambda     RMSE  Rsquared      MAE  RMSESD RsquaredSD    MAESD
## 151 1 9     0      0 1154.418 0.1697595 606.5198 214.128  0.1067966 98.10142
```

Constructing the model after using the Box Cox on the y and standardizing the features after embedding we get as a model that the minor rmse the one with degree = 1, knots = 9, alpha and lambda = 0, so no penalty. So we proceed with the estimation of this model and we have to remember to perform the inverse box cox function in order to return to the original y scale.



This is how the model fits on the data.

It should be noted that this transformation is really useful in case we want to fit obviously non-linear data like WMAP data with a linear model. however, it is clear that this is not the objective of this assignment, which refers to the development of models with splines, which have the purpose of reducing a non-linear regression model to a linear model. Actually the $BoxCox$ could also be seen as an alternative to splines, but this in any case is not strictly related to this assignment.