Coursework - Machine Learning

CID: 02091191

March 13, 2022

Abstract

We are presenting in this document the work conducted during the second Machine Learning Coursework of the year.

1 First question

In this question, we consider a provided dataset of dimensions $(500, 40)$ containing $p = 39$ biological features along with an additional column which corresponds to the log-ratio of the concentration of two proteins of interest and is the quantity of interest here. Let $n = 500$ denote the sample size. We may first focus on a short exploratory analysis of this dataset. Note that all features X and response y are continuous.

First, one may focus on correlations between the features $(X_1, \ldots, X_{39})$, as shown on Figure 1.1. This Pearson's correlation matrix heatmap shows that $(X_{35}, X_{36}, \ldots, X_{39})$ are relatively uncorrelated to the other variables. Moreover, y is highly correlated in magnitude to X_{29} and X_{31}. One can thus expect these features to be important when trying to model y from X.

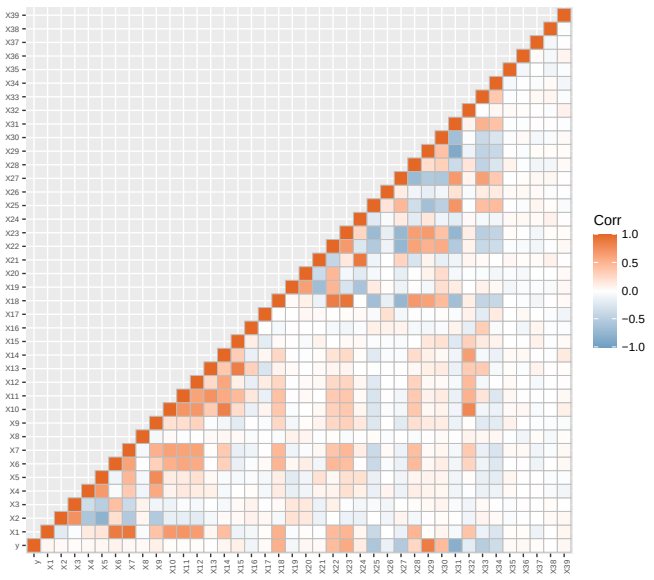


Figure 1.1: Correlation matrix between features $(X_1, \ldots, X_{39})$ and response y.

Then, since the number of features is to high to visualise every feature distribution, variables may be grouped according to their type of distribution. Note that features in this dataset are in general either distributed approximately as a normal distribution, a left-skewed distribution, a right-skewed distribution or a uniform distribution. This is illustrated on Figure 1.2, where these 4 types of distributions are plotted for some example variables, along with the distribution of y. Note that y seems either symmetrically

distributed or slightly right-skewed, but maybe not normally distributed. Note also that all features have different range of values.

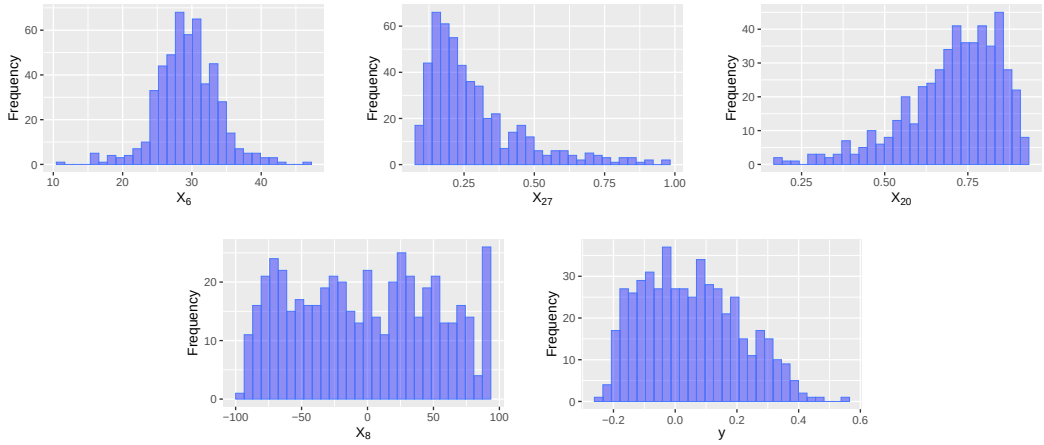


Figure 1.2: Examples of different variable distributions in the dataset.

Finally, it makes sense to give a low dimensional visualisation of the feature space X using a Principal Component Analysis (PCA) for example, as shown on Figure 1.3. This plot clearly shows that there exists a relationship between features X and the response y since we observe a trend between y and the first two principal components: y decreases when the two first components increase.

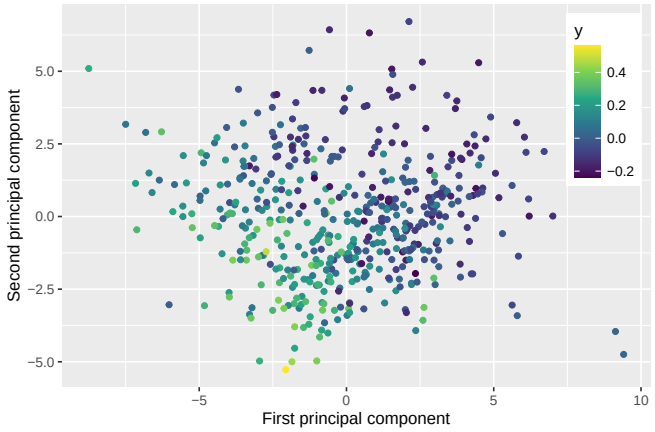


Figure 1.3: Scatter plot of the two first principal components issued from PCA. Points are coloured according to their value of y.

A - Unsupervised Learning

1. Hierarchical Clustering on features

Hierarchical Clustering is a popular method widely used for clustering data, with main advantage being that it does not require to specify the number of clusters in advance. Hierarchical Clustering depends on two hyper-parameters: the measure of dissimilarity between the groups of observations, based on the pairwise dissimilarities among the observations in the two groups, also called a linkage criterion; and a distance measure between pairs of observations (Lecture Notes [1]). Here, Hierarchical Clustering is used to cluster the 39 features, that is to check if some features are redundant, using 4 linkages criteria (single, complete, average and Ward) and 3 types of distance (Pearson's correlation, Manhattan and Euclidean). All linkages and distances formulas are available in Appendix A. Since all features have different ranges of values and that Hierarchical Clustering is a distance-based algorithm, one needs to centre and scale each feature in order to make them comparable. The results of this features clustering is shown on Figure 1.4 (note that the dendrograms for the single linkage criterion were not plotted since their results were not as relevant as the others). One may notice that the global *shape* of the dendrograms depend more on the

linkage criterion than the distance. The quality of clustering using an average linkage seems not as good as using complete or Ward. This plot first shows that the choice of hyper-parameters is essential since it results in completely different outcomes, and also that the linkage criteria seems to have more effect on the shape of the final dendrogram than the distance.

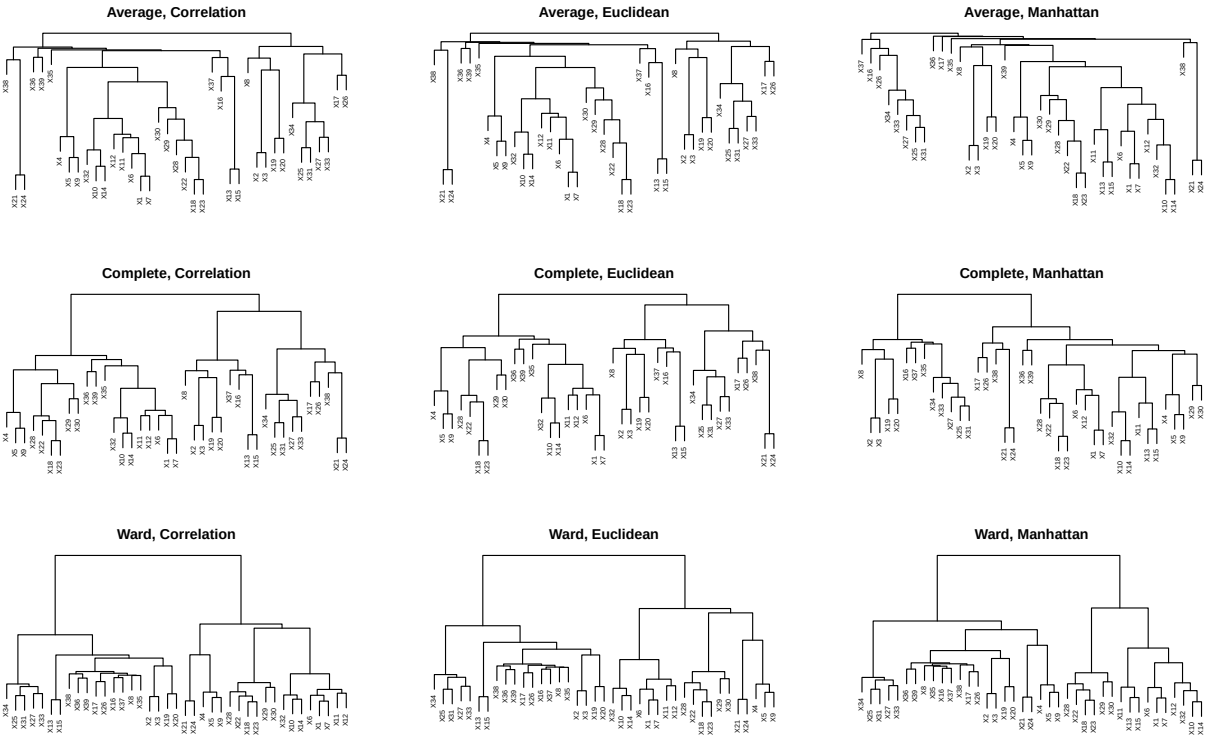
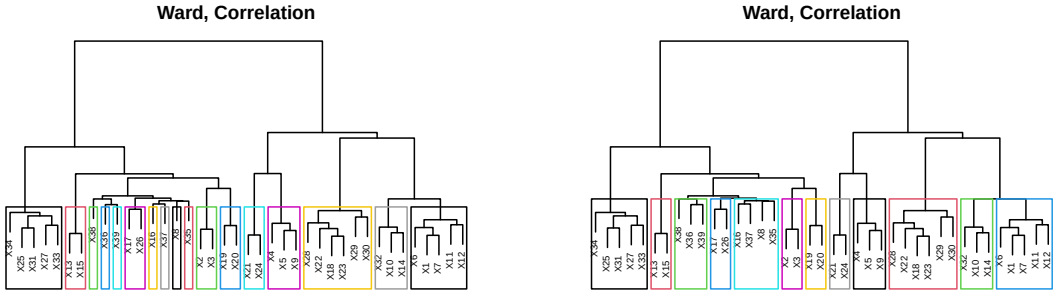


Figure 1.4: Dendrograms obtained from Hierarchical Clustering on all 39 features for different linkages and distances.

To determine the best linkage and distance parameters along with the best number of clusters, one may focus on the widely used silhouette measure and try to maximise it. To do so, for each linkage-distance possible couples (i.e. for each dendrograms shown on Figure 1.4 in addition to the single linkage criterion), the silhouette score is calculated for all possible number of clusters, that is between 2 and 38 since there are 39 features. Therefore, we obtain a grid of values of shape $(4 \times 3 \times 37)$ containing all the silhouette scores of each configuration (4 different linkages, 3 different distances, and $38 - 2 + 1 = 37$ numbers of clusters). Finding the best hyper-parameters simply consists in finding the linkage, distance and number of clusters that maximises the silhouette score among the grid: that is, here, a Ward linkage criterion, a correlation distance and 17 clusters which result in a silhouette score of 0.3773.



(a) $k = 17$ (b) $k = 12$

Figure 1.5: Final dendrograms obtained from Hierarchical Clustering on features using Ward linkage and correlation distance, with k clusters.

As shown on Figure 1.5a, the final features clustering seems consistent: features are gathered in groups when they share similarities among samples. However, one could note that the number of clusters $k = 17$ is relatively high compared to the total number of features, and in particular it can be seen that multiple features are alone in their cluster (X_{38}, X_{36}, X_{39} are some examples). This makes sense regarding the correlation matrix plotted on Figure 1.1, where these features are uncorrelated from the others. To investigate more on that effect, one could plot the silhouette score evolution with Ward linkage and correlation distance as a function of the number of clusters.

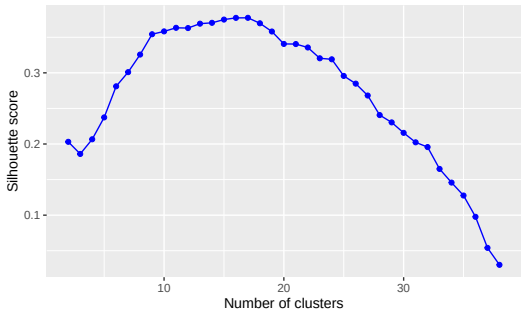


Figure 1.6: Evolution of silhouette score according to the number of clusters using Ward linkage and correlation distance.

Figure 1.6 shows that the silhouette score is therefore maximised for $k = 17$ clusters. However, this plot also shows that the silhouette score is almost constant for $k \in \{12, ..., 17\}$. Hence, considering $k = 12$ clusters instead of $k = 17$ would still make sense and not worsen the silhouette score that much, from 0.3773 to 0.3629. Therefore, one could cluster the features in 12 groups instead of 17, as shown on Figure 1.5b. The obtained groups merged the single within their cluster features in other clusters. This result may be more satisfying than the previous result using $k = 17$. Processing this Hierarchical Clustering technique may be useful for feature selection for example, in order to pick the best features for a supervised model without redundancy.

2. K-Means Clustering on samples

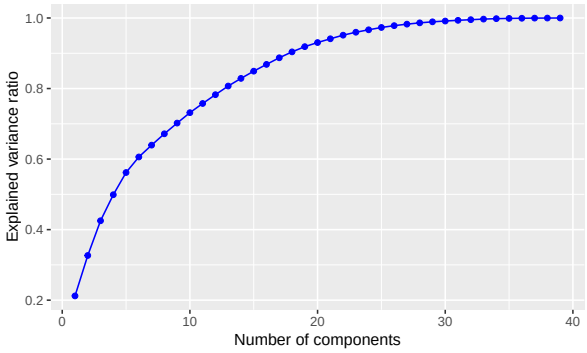
Now focus on clustering the data samples and not the features anymore. To recall, the dataset contains $X = (X_1, ..., X_{39}) \in \mathbb{R}^{n \times p}$ and $y \in \mathbb{R}^n$ where $n = 500$ and $p = 39$. One could apply K-Means algorithm to cluster the n cells in 2 groups using only the features X. To recall, the K-Means algorithm is an iterative clustering algorithm based on Euclidean distances between data points, which iteratively moves the k centroids m_k to minimise the loss

$$\varepsilon = \sum_{k=1}^{2} \sum_{i=1}^{n} z_{ki} \|x^{(i)} - m_k\|^2$$

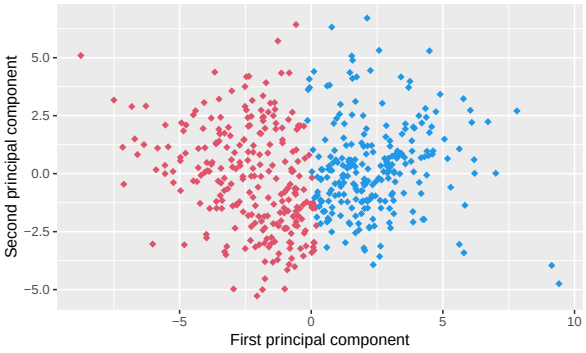
where z_{ki} is the binary indicator associated with each data point and cluster $z_{ki} \in \{0, 1\}$. The K-Means algorithm initialises the centroids m_k randomly. Then, at each iteration, it assigns each data point to its closest centroid and then updates the centroid position as the centre of gravity (the cluster means) of each point in its cluster. The process stops when the assignments don't change anymore, or when ε converges to a steady value.

One can fit a K-Means algorithm to the samples X in order to group them in 2 clusters. Then, to visualise the clustering results, one can use Principal Components Analysis to reduce the number of dimensions of the dataset from $p = 39$ to $p' = 2$ to help visualise the data in 2 dimensions. Centring and scaling the data is necessary prior to PCA since features have different ranges of values. Figure 1.7a shows the explained variance ratio as a function of the number of principal components. This plot shows that considering no more than 25 features is sufficient to explain almost the whole variance of the dataset (exactly 97.3%). However, plotting the data in 2 dimensions according to the two first components may not explain a lot of the total variance. On Figure 1.7b are plotted the data points in the first two components space, and each data point is coloured according to its cluster from K-Means algorithm. It does not seem trivial

to group the points in two clusters from this low dimensional visualisation since only a few variance is explained. Moreover, the K-Means clustering seems to divide the data vertically in two groups.



(a) Explained variance ratio from PCA.



(b) Cluster representation in a 2D plot made of the two first components from PCA.

Figure 1.7: Results of PCA and K-Means clustering on cells.

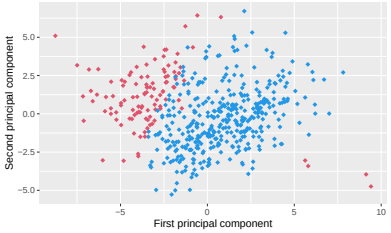
One can then use a slightly different approach to cluster the cells using the kernel K-Means algorithm, using a kernel function to transform the original data in another space and the kernel trick to make computation lower than mapping every data point from the original data space to the new data space. In the case of kernel K-Means, consider a transformation function $\phi(\cdot)$ which is applied to every $x^{(i)}$. Therefore, it can be shown (Lecture Notes [1]) that

$$\|\phi(x^{(i)}) - m_j\|^2 = K(x^{(i)}, x^{(i)}) - \frac{2}{N_j} \sum_{l \in C_j} K(x^{(i)}, x^{(l)}) + \frac{1}{N_j^2} \sum_{l,m \in C_j} K(x^{(l)}, x^{(m)})$$

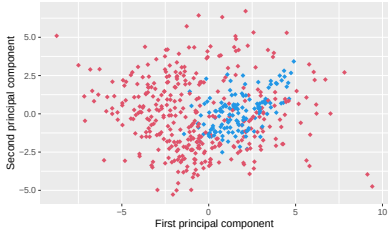
where $K(\cdot, \cdot)$ denotes the kernel function and N_j the number of observations in cluster j. Hence, plugging in this result into the loss function, it can be expressed purely in terms of inner products and as defined, the kernel K-Means algorithm aims to minimise this new loss function. Multiple kernel functions may be used, and we suggest here to try a polynomial kernel of degree 2, a RBF kernel and a hyperbolic tangent kernel, defined as

$$K_{\text{poly}}(x_1, x_2) = (a \cdot x_1^T x_2 + 1)^2, \; K_{\text{RBF}}(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{\sigma^2}\right), \; K_{\tanh}(x_1, x_2) = \tanh\left(\sigma x_1^T x_2 + r\right)$$

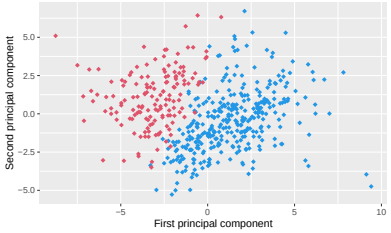
Each kernel function contains parameters such as scale, variance or offset. Fitting kernel K-Means algorithms with these three kernel functions would give different results than a simple K-Means.



(a) Polynomial, $a = 0.1$



(b) RBF, $\sigma = 0.1$



(c) Tanh, $\sigma = 0.2$, $r = -2$

Figure 1.8: Visualisation of kernel K-Means clusters for the three defined kernel functions.

Figure 1.8 shows how using a kernel function may change the clustering results compared to Figure 1.7. Only from a qualitative point of view, it seems that the hyperbolic tangent kernel K-Means separated quite well the data points. However, it is hard here to compare quantitatively the clustering quality of each model since no really distinct groups appear on the dataset. Hence, one can not clearly state that using a particular kernel is better than using another. However, we may compare these kernels K-Means

with the classic K-Means algorithm on multiple aspects, such as the number of points in each cluster, the within cluster sum of squares or the average value of y per cluster.

	Proportion		**WCSS**		**Mean WCSS**		**Mean y**	
	Blue	Red	Blue	Red	Blue	Red	Blue	Red
No kernel	0.49	0.51	8172	8545	33.36	33.51	-0.03	0.13
Polynomial	0.78	0.22	13488	6846	34.41	63.39	0.06	0.05
RBF	0.29	0.71	3872	16169	26.89	45.42	-0.02	0.08
Tanh	0.69	0.31	13079	7551	37.69	49.36	0.03	0.09

Table 1: Comparison of clustering results between K-Means and kernel K-Means for each kernel. Proportion is the proportion of samples in each cluster. WCSS is the within cluster sum of squares. Mean WCSS is the average WCSS. Mean y is the average value of y within each cluster.

Table 1 summarises quantitative results comparing each clustering algorithm used. It can be inferred for example that the polynomial kernel K-Means may not be the ideal clustering algorithm compared to the others: the proportion of samples in each cluster is very unbalanced and the mean WCSS is in the same time the highest for the red cluster, and finally both clusters did not manage to divide data in 2 groups with different y values (which is not directly the point of the question but may be useful in a regression setting). However, the RBF and hyperbolic tangent kernels seem to perform better on dividing the data in 2 groups: the proportion of samples in each cluster is more balanced than for the polynomial kernel and the mean WCSS are relatively small. Note also that the simple K-Means algorithm remains interesting too, and we can't assess that a clustering algorithm is better than another in this setting.

B - Supervised Learning

1. Modelling y from X

This part now focuses on fitting two regression models to predict y given $X = (X_1, ..., X_{39})$ and compare them. First, the original dataset is divided in a train set and a test set according to a 80%–20% split. First, a LASSO model is fitted to the data. To recall, a LASSO regression model aims to minimise the residuals along with a L_1 regularisation term, that is the loss function

$$L(\theta, \lambda) = \frac{1}{n}\sum_{i=1}^{n}(y - X\theta)^2 + \lambda\|\theta\|_1$$

where $\theta = (\theta_1, ..., \theta_{39})^T$ is the parameter vector and λ is the penalisation coefficient. Note that we do not consider any intercept term in this LASSO model and that data is scaled according to the training set in order to get a uniform range of values of θ estimates. The λ parameter can be optimised using cross-validation: here, we use a k-fold cross validation with 10 folds and searching for the best value of $\lambda \in [10^{-5}, 0.05]$ using the mean squared error (MSE), or equivalently the root mean squared error (RMSE) as the performance metric.

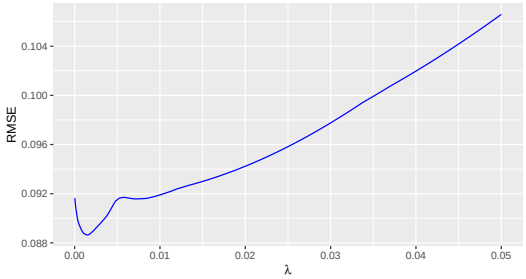


Figure 1.9: Evolution of the RMSE as a function of λ on the train set.

Figure 1.9 shows that the RMSE is minimised for $\lambda = 1.53 \times 10^{-3}$. Then, the final LASSO model is fitted using this optimal value of λ on the whole training set in order to infer the model parameters θ.

Then, a Random Forest regression model is fitted to the data. To recall, a Random Forest model consists in using a high number of regression trees with fixed depth to model subsets of y from X using bootstrap, each tree using randomly different features at each split, and therefore giving different outcomes. These trees are bagged together to form the Random Forest model. The advantage of this method is to prevent from overfitting which is often observed using regression trees. Therefore, bagging (bootstrap aggregation) all these trees tends to reduce variance. The main hyper-parameters of Random Forest are:

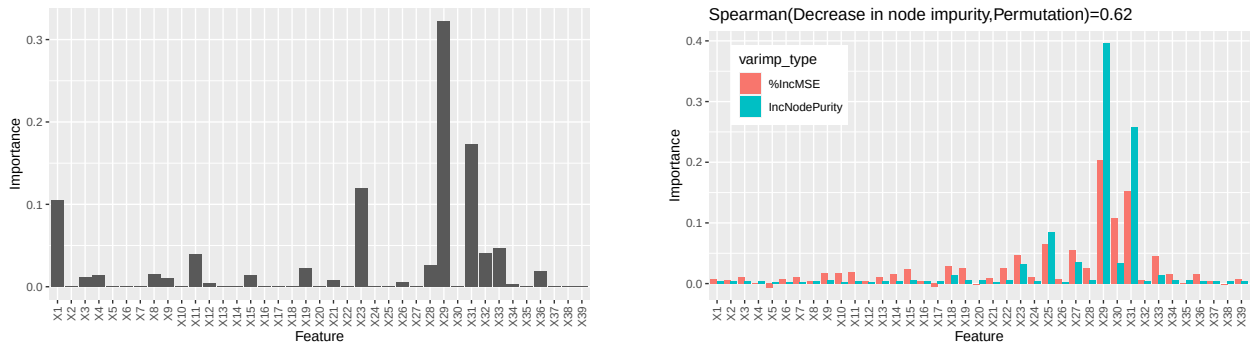
- `mtry`: the number of features considered at each split point,
- `ntree`: the number of trees in the forest,
- `maxnodes`: the maximum number of allowed leaf nodes in each tree (or equivalently, the number of examples being in leaf nodes in each tree).

These hyper-parameters can be optimised using cross-validation. However, since bootstrap consists in sampling with replacement, some data points may not be used at all during the training process. Then, one may use out-of-bag error instead of cross-validation in order to use these examples and to train the algorithm faster. Thus, we fit a Random Forest regression model to the training set using the out-of-bag mean squared error as the target metric to optimise the three hyper-parameters defined previously according to a predefined grid: $\texttt{mtry} \in \{5, 6, ..., 15\}$, $\texttt{ntree} \in \{500, 600, 700\}$ and $\texttt{maxnodes} \in \{50, 60, ..., 100\}$. This optimisation leads to the optimal parameters $\texttt{mtry} = 14$, $\texttt{ntree} = 600$ and $\texttt{maxnodes} = 70$. Note that data has not been scaled when fitting a Random Forest model since trees are invariant to scaling and do not take any notion of distance into account.

Both LASSO and Random Forest can be used to evaluate features importance to model the response y. In particular, LASSO is tailored for selecting important features due to its L_1 penalty term (importance here should be understood as global importance in the presence of other features and not individual feature importance). Indeed, the L_1 regularisation term makes some parameters coefficients of θ equal to 0: these features are therefore considered as not important in presence of the others. Then, feature importance of θ_i denoted as $\tilde{\theta}_i$ may be calculated as

$$\tilde{\theta}_i = \frac{|\theta_i|}{\sum_{j=1}^{39} |\theta_j|}, \ \forall i \in \{1, ..., 39\}$$

Feature importance can also be estimated when using a Random Forest model by two ways. The first (denoted as %IncMSE) consists in, for each tree in the forest, computing the difference between the MSE of out-of-bag set, and the MSE of out-of-bag set after permuting each predictor. This quantity is then averaged over all trees and normalised by the standard deviation of the differences. The second (denoted as IncNodePurity) is the total decrease in node impurities (residual sum of squares here) from splitting on the variable, averaged over all trees.



(a) LASSO

(b) Random Forest

Figure 1.10: Feature importance for the fitted LASSO and Random Forest models.

As shown on Figure 1.10, the Spearman correlation between the 2 sets of importance for Random Forest is 0.62, which means that the two sets of importance are relatively similar. One should take care when

interpreting the Random Forest importance measures since they are known to be biased. Important features from LASSO and Random Forest are often the same, such as X_{29} and X_{31}. However, some features selected by LASSO and are not important to Random Forest, such as X_1 or X_{23} and vice versa, X_{25} seems important to Random Forest but not to LASSO.

These two models being fitted, one may be interested in comparing their performances on test set. Since these models are regression models, one can use mean squared error (or equivalently root mean squared error) and R2 score to compare their performances.

	Train		**Test**	
	RMSE	R2	RMSE	R2
LASSO	0.0842	0.8126	0.0785	0.8248
Rand. For.	0.0305	0.9668	0.0635	0.8475

Table 2: Summary of RMSE and R2 scores on train and test sets for LASSO and Random Forest models

Table 2 shows that overall the Random Forest model has better performances for both measures than LASSO on both train and test sets. This may be explained since the Random Forest model is a more complex model than LASSO. However, the Random Forest regression model has much better performances on train set than on test set, which may be due to overfitting: the variance is high but the bias is low on train test. Regression trees are known to overfit data, thus considering more trees in the forest or less deep trees would help reduce variance and therefore increase bias on the train set but would maybe reduce bias on the test set, which would decrease the RMSE and increase the R2 score. Thus, Random Forest still performs slightly better on the test set than LASSO, but its computational cost is also higher than LASSO.

These models may be improved in different ways on the performance aspect and on the computational cost aspect. First, one may try to use different different features than the original dataset and in particular one may use the 39 features generated by PCA instead of $X = (X_1, ..., X_{39})$. This makes sense since Random Forest uses trees that split the dataset iteratively along each feature (however, a linear regression would be invariant to this transformation since PCA only creates linear transformations of the initial features space). In the context of predicting on test set, the PCA should only be run on the train set and be used to transform the data of the test set. Then, these 39 features would feed a Random Forest regression model with optimised parameters. This procedure results in a RMSE of 0.0414 on train set and 0.0928 on test set, which is higher than without transforming the data into the principal components space. Therefore, transforming the features space into the principal components space does not improve the model. In the context of transforming the data, one could also try to apply logarithmic or exponential transformations to skewed features to make them more *normally distributed*. However, it has not been explored in this report since there are 39 features and it would require some effort.

Another way to improve the models would be to use the strengths and weaknesses of both LASSO and Random Forest in order to create a more efficient global model. Thus, one could use the predictions of both models as new inputs for a final model (such as a linear regression for example): this final model would take as inputs the predictions of trained LASSO and Random Forest models, and would try to give a more accurate prediction of the response as output. This technique of ensemble learning is called stacking and is widely used in Machine Learning competitions. Hence, using the pre-trained LASSO and Random Forest models predictions as inputs of another linear regression model could improve the performances. In practice, training a such model results in a RMSE of 0.0215 on train set and 0.0669 on test set. Therefore, this has improved the performances on train set but worsen on test set.

A final idea to improve the models performances would be to use the work conducted previously focusing on variable importance and keep as inputs only the relevant features. Thus, based on the Random Forest importance plot on Figure 1.10, we keep as inputs of the model only features that have an importance value greater than 0.01, that is 8 features in total. A new Random Forest model is fitted and its parameters are optimised using out-of-bag error estimation. The final performance of this model is a RMSE of

0.0320 on train set and 0.0671 on test set. Therefore, it does not improve the previous models either in terms of RMSE. However, one may note that the performance on test set is relatively equal to previously using 39 features. Hence, considering only 8 features, this model performs as well as the one using all features: the advantage of this model is its lower computational cost, since the Random Forest has a time complexity which is linear in `mtry` which directly depends on the number of features of the model.

Therefore, for prediction task, one should recommend using the Random Forest model using only relevant features. Indeed, this model seems to be a good trade-off between computational cost and performance. Note that we did not try to use different models but only tried to improve the existing and considered models. More complex models could maybe have improved the performances in terms of prediction. However, note that using the LASSO model in a prediction setting would make the algorithm more explainable than a Random Forest: in order to prevent our model to be a *black box*, one could prefer a LASSO regression model as it has higher interpretability. Moreover and in general, in a prediction setting, the predicted output is very dependent to the input. In particular, if for some reason a new input may not be sampled from the same distribution as during training, the model could potentially return surprising predictions.

2. Effects of train-test split

The aim of this question is to focus and the effect of train-test split on both performance and feature importance for the two models, in particular on how the MSE and the features importance ratios vary when splitting the train and test sets differently. To do so, one may use the so-called nested cross validation algorithm for LASSO.

Algorithm 1: Nested cross-validation algorithm for LASSO model

Require k_{in} and k_{out} the outer and inner folds numbers
Require a dataset $\mathcal{D}$ of features X and response y
Require a grid $\mathcal{P}$ of λ values
for $i = 1$ *to* k_{out} **do**
  Divide $\mathcal{D}$ into $\mathcal{D}_i^{train}$ and $\mathcal{D}_i^{test}$ for the i^{th} split
  for $j = 1$ *to* k_{in} **do**
    Divide $\mathcal{D}_i^{train}$ into $\mathcal{D}_j^{train}$ and $\mathcal{D}_j^{test}$ for the j^{th} split
    for λ *in* $\mathcal{P}$ **do**
      Fit the LASSO model on $\mathcal{D}_j^{train}$ using λ penalty
      Compute test error ε_j^{test} using $\mathcal{D}_j^{test}$
    end
  end
  Select optimal value λ^* from $\mathcal{P}$, that is the one that minimises $\mathcal{D}_j^{test}$
  Fit LASSO using λ^* on $\mathcal{D}_i^{train}$ using λ^*
  Compute test error ε_i^{test} using $\mathcal{D}_i^{test}$
  Compute the features importance (as previously)
end
Average the k_{out} error values obtained
Average the k_{out} feature importance values obtained

Running Algorithm 1 gives random results since outer and inner folds are generated randomly. Thus, running this algorithm K times would give K different MSE values and K features importance values (for each feature). These values can then be compared to the ones obtained in **B.1**. Here, we run Algorithm 1 $K = 100$ times using $k_{out} = 10$ and $k_{in} = 5$ along a grid of λ values $\mathcal{P} = \{0, 0.001, 0.002, ..., 0.05\}$. Results are shown on Figure 1.11. The two left plots show that RMSE varies according to each split and that for some splits, it can be higher than expected. Moreover, the right plot shows that features importance values are relatively stable, in the sense that a feature does not become much more important than another depending on the train-test split, even if variations are not negligible.

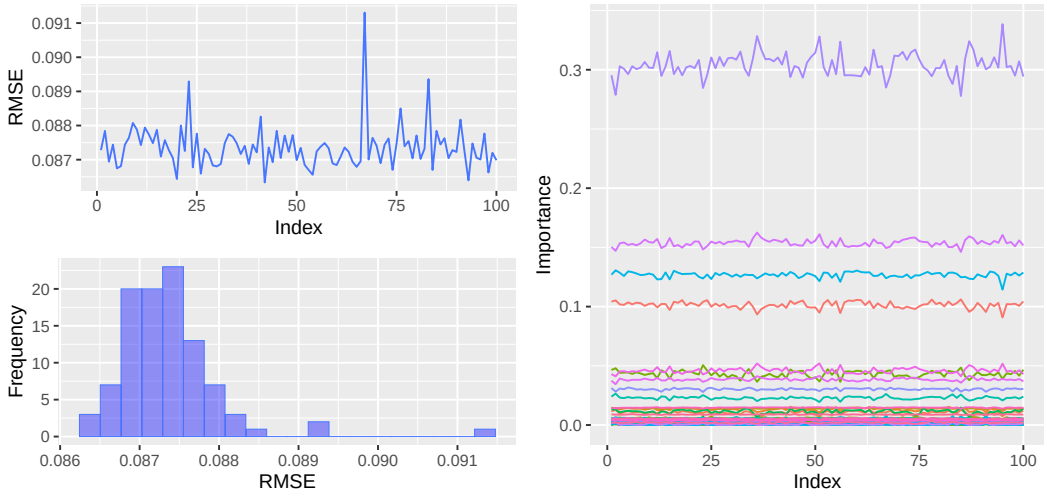


Figure 1.11: Results of running 100 nested cross-validation algorithm on LASSO. Top left: evolution of RMSE on all algorithm runs. Bottom left: histogram of RMSE values obtained. Right: evolution of features importance on all algorithm runs (a plot with legend can be found in appendix).

The same procedure can be applied for the Random Forest model. However, since we are using out-of-bag error instead of cross-validation, the inner fold procedure slightly changes.

Algorithm 2: Nested "cross-validation" algorithm for Random Forest model

Require k_{out} the outer folds numbers
Require a dataset $\mathcal{D}$ of features X and response y
Require a grid $\mathcal{P}$ of parameter values
for $i = 1$ *to* k_{out} **do**
  Divide $\mathcal{D}$ into $\mathcal{D}_i^{train}$ and $\mathcal{D}_i^{test}$ for the i^{th} split
  for p *in* $\mathcal{P}$ **do**
    Fit the Random Forest model on $\mathcal{D}_i^{train}$ using p parameters
    Compute the out-of-bag error using out-of-bag samples
  end
  Select optimal value p^* from $\mathcal{P}$, that is the one that minimises the out-of-bag errors
  Fit a Random Forest using p^* on $\mathcal{D}_i^{train}$ using p^*
  Compute test error ε_i^{test} using $\mathcal{D}_i^{test}$
  Compute the features importance (as previously)
end
Average the k_{out} error values obtained
Average the k_{out} feature importance values obtained

This algorithm is not really a nested cross-validation since there is not any inner split but the main principles are the same. Thus, this algorithm is run $K = 100$ times using $k_out = 10$ along a grid of values $\mathcal{P} = \{\texttt{mtry} = \{7, 8\}, \texttt{maxnodes} = \{50, 60\}, \texttt{ntree} = 800\}$. Results are shown on Figure 1.12. The RMSE seems more stable to train-test splits than for the LASSO algorithm since RMSE variations are smaller, and this is confirmed when comparing the RMSE standard deviations ($6.5 \cdot 10^{-4}$ for LASSO and $3.8 \cdot 10^{-4}$ for Random Forest). Moreover, the features importance are also less noisy than using the LASSO model. Therefore, one may conclude that both performances and features importance are sensible to train-test split as these nested cross-validation algorithms show. This procedure is thus a good practice to better estimate a model performance, but its computational cost is significantly larger than a simple k-fold cross-validation which therefore requires to restrict more the grid of parameter values. One could obtain a better estimate of the error rate on test set by computing the sample mean of the obtained K error values.

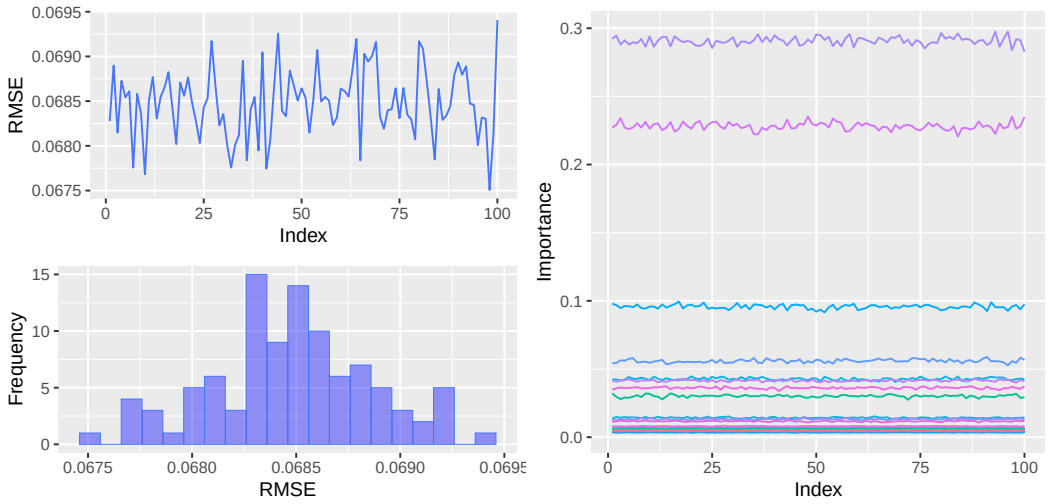
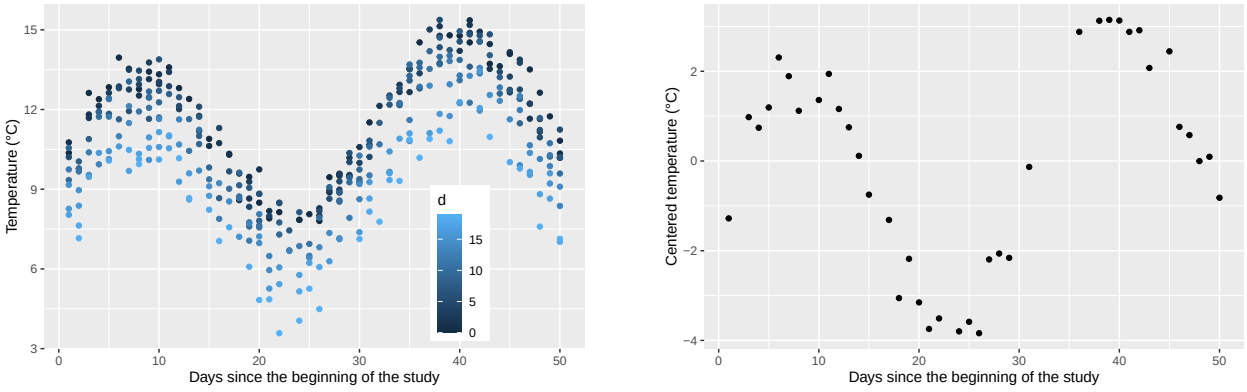


Figure 1.12: Results of running 100 nested cross-validation algorithm on Random Forest. Top left: evolution of RMSE on all algorithm runs. Bottom left: histogram of RMSE values obtained. Right: evolution of features importance on all algorithm runs (a plot with legend can be found in Appendix B).

2 Second question

In this question, we consider a provided dataset containing n water temperature measurements `temp` at different times `t` and at various distances `d` from the coast.

2.1 Modelling the water temperature when $d = 0$



(a) $d \in [0, 19]$ (b) $d = 0$

Figure 2.1: Evolution of the temperature (left) and centred temperature (right) as a function of the time.

The aim of this question is to model the water temperature near the coast as a function of time as shown on Figure 2.1b using a Gaussian Process model based on three different kernels for the covariance function. Therefore, let $\boldsymbol{y} \in \mathbb{R}^n$ be the one-dimensional response vector of observed temperature values, let $X \in \mathbb{R}^n$ the vector of times `t` values: one seeks a function f such that

$$\boldsymbol{y} = f(X) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_n^2)$$

Calculations in the Lecture Notes [1] show that, given a covariance function $k(\cdot, \cdot)$, $K = k(X, X)$, and new data samples X_*,

$$\begin{pmatrix} \boldsymbol{y} \\ f_* \end{pmatrix} \mid X, X_* \sim \mathcal{N}\left(\begin{pmatrix} m(X) \\ m(X_*) \end{pmatrix}, \begin{pmatrix} K + \sigma_n^2 I & k(X, X_*) \\ k(X_*, X) & k(X_*, X_*) \end{pmatrix} \right)$$

where $m(\cdot)$ denotes the mean function. Hence, one may derive the posterior predictive distribution $p(f_* \mid X, \boldsymbol{y}, X_*)$ at test inputs X_* obtained by Gaussian contitioning

$$
\begin{aligned}
f_* \mid X, \boldsymbol{y}, X_* &\sim \mathcal{N}\left(\mathbb{E}\left[f_* \mid X, \boldsymbol{y}, X_*\right], \mathbb{V}\left[f_* \mid X, \boldsymbol{y}, X_*\right]\right) \\
\mathbb{E}\left[f_* \mid X, \boldsymbol{y}, X_*\right] = m_{\text{post}}\ (X_*) &= \underbrace{m\left(X_*\right)}_{\text{prior mean}} + \underbrace{k\left(X_*, X\right)\left(K + \sigma_n^2 I\right)^{-1}}_{\text{“Kalman gain"}} \underbrace{\left(\boldsymbol{y} - m(X)\right)}_{\text{error}} \\
\mathbb{V}\left[f_* \mid X, \boldsymbol{y}, X_*\right] = k_{\text{post}}\ (X_*, X_*) &= \underbrace{k\left(X_*, X_*\right)}_{\text{prior variance}} - \underbrace{k\left(X_*, X\right)\left(K + \sigma_n^2 I\right)^{-1} k\left(X_*, X_*\right)}_{\geq 0}
\end{aligned}
\tag{1}
$$

Figure 2.1b shows that the temperature seems to be a periodic function of time (which is consistent), one may try to use a periodic covariance function k_{per}. Moreover, one may try to use a Gaussian (or RBF) covariance function k_{RBF}. Finally, it would make sense to combine these two covariance functions in a locally periodic kernel function k_{loc}. These three kernel functions are defined as

$$
k_{\text{per}}(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\frac{\kappa(x_i - x_j)}{2\pi}\right)}{\ell^2}\right), \quad k_{\text{RBF}}(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^T(x_i - x_j)}{\ell^2}\right)
$$

$$
k_{\text{loc}}(x_i, x_j) = k_{\text{per}}(x_i, x_j) \cdot k_{\text{Gauss}}(x_i, x_j)
$$

Since these kernel functions contain hyper-parameters such as σ_f, ℓ, κ and that σ_n is unknown, one needs to train the Gaussian Process to find a good set of hyper-parameters. To do so, one may try to maximise the marginal likelihood function (or maximum likelihood Type-II), defined as

$$
p(\boldsymbol{y} \mid X, \theta) = \int p(\boldsymbol{y} \mid f, X)\ p(f \mid X, \theta)\ df = \mathcal{N}(\boldsymbol{y} \mid 0, K + \sigma_n^2 I)
$$

where θ denotes the hyper-parameters set. Note that in practice the log-likelihood would be more convenient and one may implement a gradient-based approach to find the best hyper-parameters θ. However, we will here use a naive grid based approach as the marginal likelihood function may contain multiple local optima, which would make the gradient based approach not that robust.

In `R`, we then fit Gaussian Process models to the data $\boldsymbol{y}$ and X for each of the three kernels of interest and optimise their parameters using a grid-based optimisation approach. Note that one could have chosen a gradient-based optimisation approach coupled with MCMC methods for hyper-parameters tuning instead of a grid search. However, this additional effort seemed here not necessary since the grid-based approach gave relatively good results.

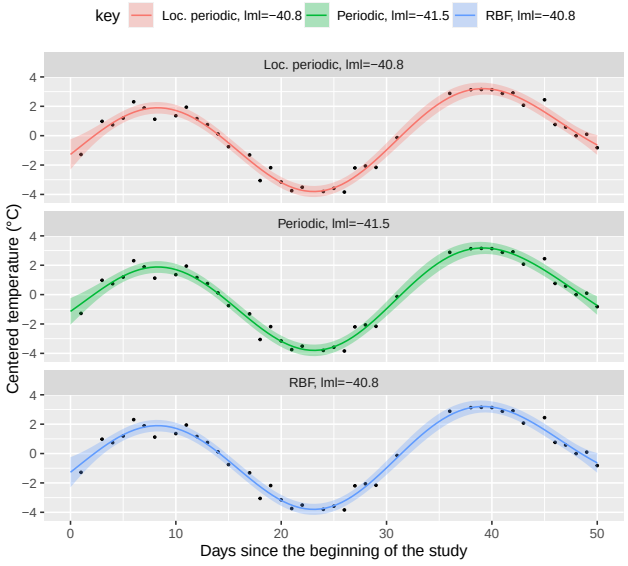


Figure 2.2: Gaussian Processes for each kernel function fitted to the temperature as a function of time. Coloured areas correspond to 95% confidence intervals for predictions.

		min	max	step	opt.
RBF	σ_f	10^{-3}	15	0.1	4
	ℓ	10^{-3}	20	0.1	9.8
	σ_n	0.1	2	0.1	0.5
Per.	σ_f	10^{-3}	7	0.1	4
	ℓ	0.1	5	0.1	1.1
	κ	30	70	1	64
	σ_n	10^{-3}	2	0.1	0.5
Loc. Per.	σ_f	3.8	4.2	0.1	4
	ℓ_{per}	1	10	0.2	10
	ℓ_{RBF}	9	11	0.2	9.8
	κ	62	68	1	68
	σ_n	0.4	0.5	0.1	0.5

Table 3: Range (min, max) of grids and step for each parameter optimisation for each kernel function. Optimal values are reported in the opt. column.

Table 3 reports the optimal values obtained, i.e. the hyper-parameters values that maximise the log-marginal likelihood. Gaussian processes fitted with these optimal values are plotted on Figure 2.2. We may first note that the obtained results look very similar and that all three models show qualitatively a good fit to the data. However, quantitatively, the obtained log-marginal likelihood values for the locally periodic and the RBF kernels are higher (-40.8) than for the periodic kernel (-41.5). Moreover, by computing the mean squared error (MSE) comparing the fitted values $\hat{\boldsymbol{y}}$ to the response $\boldsymbol{y}$, we obtain that the MSE of the RBF model is 0.1499, the MSE of the periodic model is 0.1519 and the MSE of the locally periodic model is 0.1498. Therefore, one should prefer using a RBF or a locally periodic kernel. As these two kernels present very similar likelihood values, we may choose one between both of them for the rest of the report. Here, one could notice that the water temperature may be periodic in time since the data points seem to follow a sinusoidal curve. Therefore, for common sense consistence, we would prefer here to use a locally periodic kernel function using the optimal values found. Also, one could take into account that the locally periodic model has a slightly smaller MSE.

Assume now that we are using the optimised locally periodic kernel as the best covariance function for this Gaussian Process model, and that we now want to predict the temperature when $d = 0$ at day $t = 35$. Using the posterior predictive distribution described in Equation (1), one can easily estimate the predicted water temperature for $t = 35$ days as the expected value of f_* given X, $\boldsymbol{y}$ and $X_* = 35$. Therefore, according to this Gaussian Process model, the water temperature at $t = 35$ days would be 13.83 degrees. As we can also easily compute the variance of the distribution of f_* at that point, we may provide a confidence interval for this predicted value. The estimated variance is 0.0766. Hence, a 99% confidence interval is computed as

$$\left[\mathbb{E}\left[f_* \mid X, \boldsymbol{y}, X_*\right] \pm q_{0.995}\sqrt{\mathbb{V}\left[f_* \mid X, \boldsymbol{y}, X_*\right]}\right]$$

where $q_{0.995}$ denotes the 0.995-quantile of a standard normal distribution. Here, the 99% confidence interval is $[13.12, 14.55]$, which is wider than the coloured area on Figure 2.2 since this area corresponds to a 95% confidence interval.

In order to estimate the probability that the water temperature would be greater than 13 degrees that day, one could notice in Equation (1) that posterior predictive distribution is normally distributed around the predicted value. Hence, $\mathbb{P}[f_* > m(X_*) - 13 \mid X, \boldsymbol{y}, X_*] = 1 - \mathbb{P}[f_* \leq m(X_*) - 13 \mid X, \boldsymbol{y}, X_*]$ where $f_* \mid X, \boldsymbol{y}, X_* \sim \mathcal{N}(2.187, 0.0766)$ and $m(X_*) = 11.65$. Thus, numerically we find that the probability that the water temperature would be greater than 13 degrees is 0.9917, which is consistent according to Figure 2.3: at $t = 35$, the confidence interval for the predicted value of the water temperature is located above 13 degrees and thus, it is very likely that the temperature on that day will exceed 13 degrees.

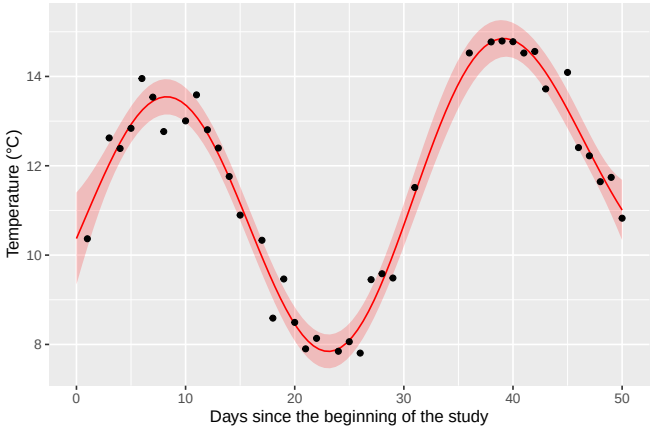


Figure 2.3: Temperature samples as a function of time when $d = 0$ (in black dots). Predictions on $t \in [0, 50]$ using a Gaussian Process model fitted to the data (red line) and 95% confidence interval on the prediction (red coloured area)

2.2 Modelling the water temperature as a function of d and t

Assume now a more complex model stating that the water temperature `temp` depends on both time `t` and distance `d`. One may no longer use exactly the same equations as previously for fitting such a model since the predictors' space is two-dimensional. However, to bypass this restriction, a usual trick is to decompose the problem, and first look independently at the evolution of the temperature with respect to time and to distance. As shown in **2.1**, the water temperature seems to oscillate with time and there also may be a drift, which is the reason why we chose to fit a Gaussian Process with a locally periodic kernel function. One now may look at the evolution of the water temperature as a function of distance.

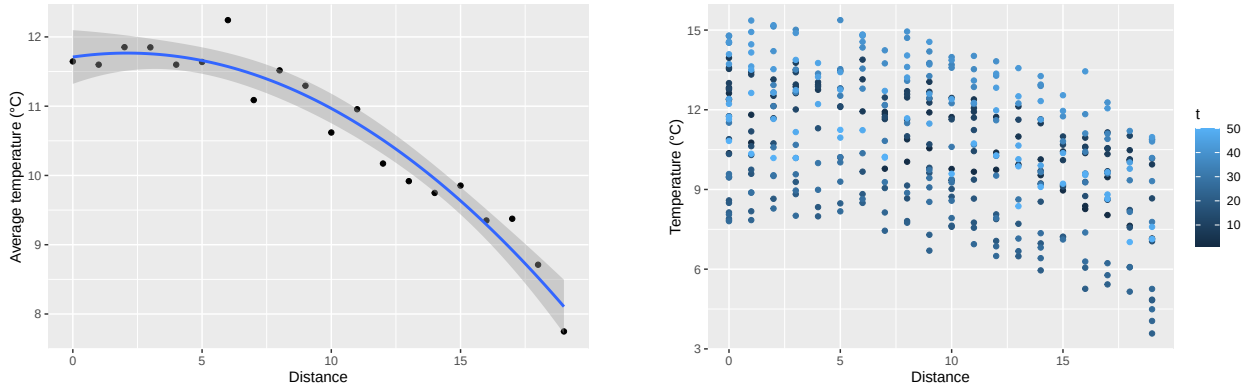


Figure 2.4: Evolution of (averaged with time on the left) water temperature with respect to distance to the coast. Fitted quadratic curve in blue (left plot).

As shown on Figure 2.4 and Figure 2.1a, the temperature decreases as distance to the coast increases, which is consistent. The fitted curve (blue) on Figure 2.1a is a quadratic model and seems to represent well the reality. Therefore, the water temperature oscillates along a trend with time, and decreases quadratically with distance. Hence, to fit a Gaussian Process model for the water temperature as a function of time and distance, one may use a custom kernel function defined as the product of two kernel functions, one for each predictor. This is consistent since the product of two kernel functions remains a kernel function (Lecture Notes [1]). The kernel function related to time will be a locally periodic kernel function, and the kernel function related to distance will be a quadratic kernel function, which can be expressed as the product of two linear kernel functions. For simplicity and for hyper-parameters optimisation convenience, we will use here the same linear kernel and square it. Therefore, the used kernel function is defined as

$$k(x_i, x_j, x'_m, x'_n) = k_{\text{loc}}(x_i, x_j) \cdot k_{quad}(x'_m, x'_n)$$

where x denotes the sequence of times `t` and x' denotes the sequence of distances `d`, with

$$k_{\text{loc}}(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\frac{\kappa(x_i - x_j)}{2\pi}\right)}{\ell_{\text{per}}^2}\right) \exp\left(-\frac{(x_i - x_j)^T(x_i - x_j)}{\ell_{\text{RBF}}^2}\right)$$

$$k_{quad}(x'_m, x'_n) = \left[k_{lin}(x'_m, x'_n)\right]^2 = \left[\sigma_b^2 + \sigma_\nu^2(x'_m - c)(x'_n - c)\right]^2$$

This kernel function k being defined, one needs to optimise the all 8 hyper-parameters σ_f, κ, ℓ_{per}, ℓ_{RBF}, σ_b, σ_ν, c and σ_n. The procedure used to optimise these hyper-parameters here is again a grid-based optimisation since computing the gradients of k with respect to all these 8 hyper-parameters would have been a tough task, and may have found a local optimum. Optimal found values are reported in Table 4 and result in a log marginal likelihood of -429.

σ_f	κ	ℓ_{per}	ℓ_{RBF}	σ_b	σ_ν	c	σ_n
0.5	0.5	15	13	4	0.1	1	0.9

Table 4: Optimal values for hyper-parameters obtained from grid-based optimisation.

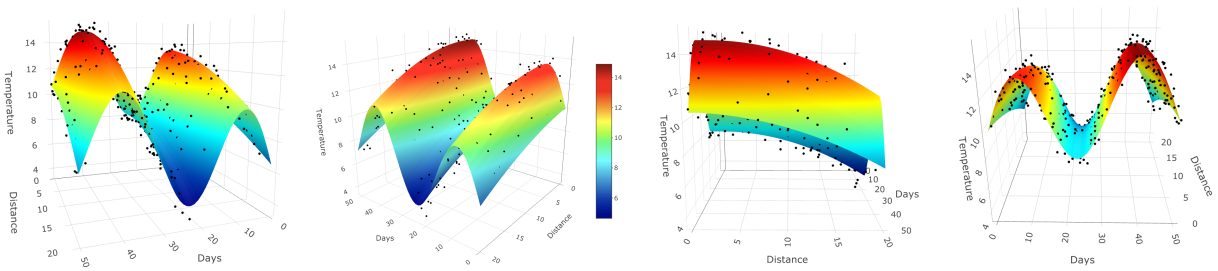


Figure 2.5: Three dimensional plots of the fitted Gaussian Process (coloured surface) along with data points (in black).

Figure 2.5 shows that the Gaussian Process fits relatively well to the data since the black data points are close to the fitted surface. On the middle-right plot, one can clearly show the quadratic curve of temperature with respect to distance. The colour scale chosen in these plots represents the fitted temperature: as the scale shows on the middle-left plot, the fitted temperature is very close to the real data temperatures. Therefore, we may assess a good quality of fit but it could be improved by considering a wider grid of values or considering a gradient-based optimisation.

Now that the Gaussian Process model has been fitted to the data, it can be used to make predictions. Here, we are interested in predicting the water temperature at day $t = 55$ as a function of the distance d. We can then easily estimate the posterior distribution using the optimised kernel function and result in Equation 1, using a X_* matrix defined as

$$X_* = \begin{bmatrix} 55 & 0 \\ 55 & 1 \\ \vdots & \vdots \\ 55 & 20 \end{bmatrix}$$

since we fix time $t = 55$ and make d vary from 0 to 20 (same range of values as in the original dataset). The predictions result is shown in Figure 2.6. The values of temperatures obtained are consistent with Figure 2.1a since the samples temperatures seem to locally decrease with time on this plot. Moreover, the global trend of the obtained curve is clearly quadratic in d, which is consistent since this is the assumption made when choosing our kernel function. Note that the confidence interval area is wider than the one plotted on Figure 2.3 since the dataset does not contain any sample at day $t = 55$, which makes the prediction more difficult to make.

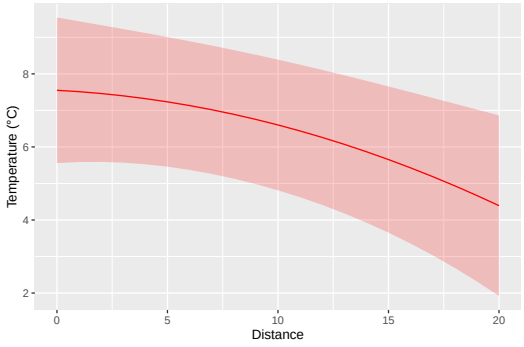


Figure 2.6: Predicted temperature on day $t = 55$ as a function of d (red line) along with a 95% confidence interval (coloured area).

References

[1] Dr Sarah Filippi (January 2022) *MATH70091 - Machine Learning lecture notes*, Imperial College London MSc Statistics resources

A Linkages and distances definitions

$$d_{\text{Eucl.}}(a,b) = \|a-b\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$$

$$d_{\text{Manh.}}(a,b) = \|a-b\|_1 = \sum_i |a_i - b_i|$$

$$d_{\text{Corr.}}(a,b) = 1 - \frac{\sum_i (a_i - \bar{a}) \sum_i (b_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2 (b_i - \bar{b})^2}}$$

$$l_{\text{single}}(A,B) = \min\{d(a,b) : a \in A, b \in B\}$$

$$l_{\text{complete}}(A,B) = \max\{d(a,b) : a \in A, b \in B\}$$

$$l_{\text{average}}(A,B) = \frac{1}{|A| \cdot |B|} \sum_{a \in A, b \in B} d(a,b)$$

$$l_{\text{Ward}}(A,B) = \sum_{x \in A \cup B} d(x, m_{A \cup B})^2 - \left(\sum_{a \in A} d(a, m_A)^2 - \sum_{b \in B} d(b, m_B)^2 \right)$$

where $m_\cdot$ denotes the center of cluster $\cdot$

B Additional plots

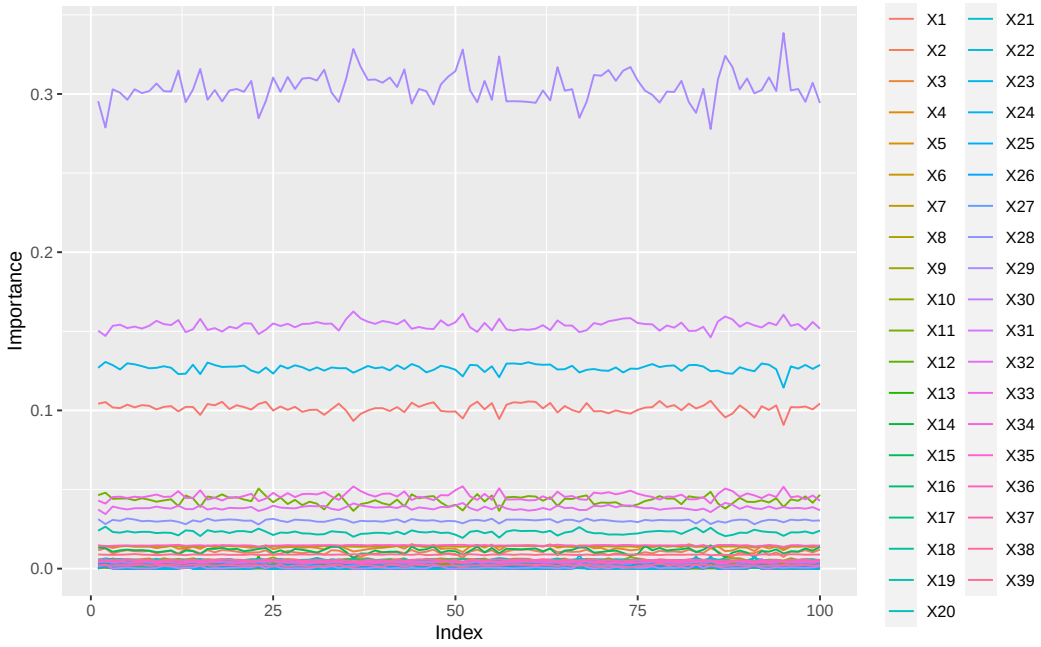


Figure B.1: Feature importance evolution using LASSO with different train-test splits

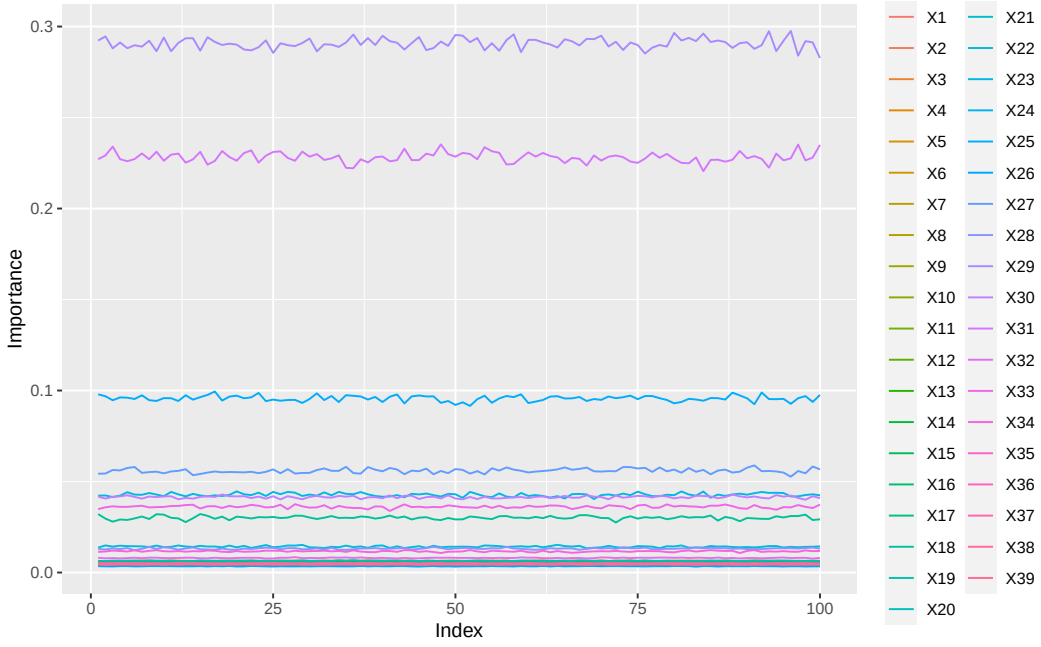


Figure B.2: Feature importance evolution using Random Forest with different train-test splits

C R code

```
### Question 1

library(ggplot2)
library(NbClust)
library(cluster)
library(purrr)
library(clValid)
library(kableExtra)
library(kernlab)
library(factoextra)
library(ggcorrplot)

data <- read.csv("02091191.csv")
head(data)
X <- data[,-1]
y <- data[,1]

# Data exploration
ggplot() +
  geom_point(aes(x=X_pca[,1], y=X_pca[,2], colour=y)) +
  scale_colour_viridis_c() +
  xlab("First principal component") + ylab("Second principal component") +
  theme(legend.position=c(0.92, 0.75))

corr <- cor(data)
ggcorrplot(corr, hc.order = FALSE, type = "lower",
           ggtheme = ggplot2::theme_gray, show.diag=TRUE,
           colors = c("#6D9EC1", "white", "#E46726"), tl.cex=6, tl.srt=90)

ggplot() +
  geom_histogram(aes(x=y), bins=30, fill="blue", alpha=0.4,
                 colour="royalblue1", lwd=.2) +
  labs(x=expression(y), y="Frequency")

# Part A

# 1. Hierarchical clustering
X_scaled <- scale(X)
X_features <- t(X_scaled)

linkages <- c("single", "complete", "average", "ward")
distances <- c("manhattan", "euclidean", "correlation")
nClustMax <- 38

results <- array(NA, dim=c(length(linkages), length(distances), nClustMax-1),
                 dimnames=list(linkages, distances, 2:nClustMax))

for(linkage in linkages){
  for(distance in distances){
    for(k in 2:nClustMax){
      intern <- clValid(X_features, nClust=k, clMethods ="hierarchical",
                        metric=distance,
```

```
                        method=linkage,
                        validation="internal")
      results[linkage, distance, k-1] <- optimalScores(intern)["Silhouette",][["Score"]]
    }
  }
}

bests <- which(results == max(results), arr.ind=TRUE)
cat(" Linkage:", linkages[bests[,1]], "\n",
    "Distance:", distances[bests[,2]], "\n",
    "# of clusters:", bests[,3]+1, "\n")


# dist_data <- dist(X_features, method = "manhattan")
dist_data <- as.dist(1 - cor(t(X_features)))
res.hc <- hclust(dist_data, method = "ward" )
grp <- cutree(res.hc, k = 17)
plot(res.hc, cex = 0.6)
rect.hclust(res.hc, k = 17, border = 2:17)

plot(results["ward", "correlation",], type="l")

ggplot() +
  geom_line(aes(x=2:38, y=results["ward", "correlation",]), col="blue") +
  geom_point(aes(x=2:38, y=results["ward", "correlation",]), col="blue") +
  xlab("Number of clusters") + ylab("Silhouette score")

#dist_data <- dist(X_features, method = "euclidean")
dist_data <- as.dist(1 - cor(t(X_features)))
res.hc <- hclust(dist_data, method = "ward" )
grp <- cutree(res.hc, k = 12)
par(mar = c(2, 1, 2, 1))
plot(res.hc, cex = 0.6, main="Ward, Correlation", xlab="", ylab="", xaxt='n',
     yaxt='n', lwd=1.5)
par(lwd=1.5, mar=c(0,0,0,0))
rect.hclust(res.hc, k = 12, border = 1:12)


# 2. KMeans Clustering

km <- kmeans(X_scaled, 2)
pca <- prcomp(X_scaled, center=FALSE, scale=FALSE)

explained_variance <- cumsum(summary(pca)$importance["Proportion of Variance",])

ggplot() +
  geom_line(aes(x=1:39, y=explained_variance), col="blue") +
  geom_point(aes(x=1:39, y=explained_variance), col="blue") +
  xlab("Number of components") + ylab("Explained variance ratio")

X_pca <- get_pca_ind(pca)$coord

colors_km <- (km$cluster-1)*2 + 2
ggplot() +
```

```
  geom_point(aes(x=X_pca[,1], y=X_pca[,2]), colour=colors_km, shape=18, size=2) +
  xlab("First principal component") + ylab("Second principal component")


kernel = "rbfdot"
kpar = list(sigma=.1)

kernel = "polydot"
kpar = list(degree=2, scale=.1)

kernel = "tanhdot"
kpar = list(scale=.2, offset=-2)

kernel_km <- kkmeans(X_scaled, centers=2, kernel=kernel,
                     kpar=kpar)
colors_kkm <- (kernel_km@.Data-1)*2 + 2

ggplot() +
  geom_point(aes(x=X_pca[,1], y=X_pca[,2]), colour=colors_kkm, shape=18, size=2) +
  xlab("First principal component") + ylab("Second principal component")


# Part B

library(caret)
library(glmnet)

train_ix <- createDataPartition(y=data$y, p=0.8, list=FALSE)
train <- data[train_ix,]
test <- data[-train_ix,]
X_train <- train[,-1]
y_train <- train[,1]
X_test <- test[,-1]
y_test <- test[,1]

# LASSO model
trControl <- trainControl(method  = "cv",
                          number = 10,
                          savePredictions = 'final')

lasso.cv <- train(y ~ .,
                  method     = "glmnet",
                  family     = "gaussian",
                  intercept  = FALSE,
                  tuneGrid   = expand.grid(alpha=c(1),
                                           lambda=seq(1e-5, .05, by=1e-5)),
                  trControl  = trControl,
                  metric     = "RMSE",
                  data       = train,
                  preProcess = c("center","scale"))

ggplot() +
  geom_line(aes(x=lasso.cv$results$lambda, y=lasso.cv$results$RMSE), colour="blue") +
  xlab(expression(lambda)) + ylab("RMSE")
```

```
lasso.cv$bestTune

lasso <- train(y ~ .,
               method     = "glmnet",
               family     = "gaussian",
               intercept  = FALSE,
               tuneGrid   = expand.grid(alpha=c(1),
                                        lambda=c(lasso.cv$bestTune$lambda)),
               trControl  = trControl,
               metric     = "RMSE",
               data       = train,
               preProcess = c("center","scale"))

# Returns . for non-selected features
lasso_importance <- coef(lasso$finalModel, lasso$bestTune$lambda)
lasso_importance <- data.frame(matrix(lasso_importance),
                               row.names = lasso_importance@Dimnames[[1]])
lasso_importance <- lasso_importance[-1,]
lasso_ratio_importance <- abs(lasso_importance)/sum(abs(lasso_importance))

summary_lasso_importance <- cbind(colnames(data)[-1], lasso_ratio_importance)
colnames(summary_lasso_importance) <- c("Feature", "Importance")
summary_lasso_importance <- as.data.frame(summary_lasso_importance)
summary_lasso_importance$Importance <- as.numeric(summary_lasso_importance$Importance)
summary_lasso_importance$Feature <- factor(summary_lasso_importance$Feature,
                                           levels=summary_lasso_importance$Feature)
ggplot(data=summary_lasso_importance, aes(x=Feature, y=Importance)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle = 90, vjust = 0.5, hjust=1))

# RMSE on train and test sets
RMSE(pred=predict(lasso, train), obs=y_train)
RMSE(pred=predict(lasso, test), obs=y_test)
R2(pred=predict(lasso, train), obs=y_train)
R2(pred=predict(lasso, test), obs=y_test)

# Random Forest model
library(randomForest)

mse <- function(y_true, y_obs) { mean((y_true-y_obs)**2.0) }

hparam_grid <- as.data.frame(expand.grid(mtry=seq(5, 15, by=3),
                                         maxnodes=seq(50, 100, by=10),
                                         ntree=seq(500, 700, by=100)))
oob_mses <- rep(0.0, nrow(hparam_grid))

for(hparam_idx in 1:nrow(hparam_grid)) {
  this_mtry <- hparam_grid[hparam_idx, 1]
  this_maxnodes <- hparam_grid[hparam_idx, 2]
  this_ntree <- hparam_grid[hparam_idx, 3]
  print(c(this_mtry, this_maxnodes, this_ntree))
  rf <- randomForest(X_train, y_train, mtry=this_mtry, maxnodes=this_maxnodes,
                     ntree=this_ntree)
```

```
  oob_mses[hparam_idx] <- mse(y_train, predict(rf))
}

best_hparam_set <- hparam_grid[which.min(oob_mses),]

rf_final <- randomForest(X_train, y_train,
                         mtry=best_hparam_set$mtry,
                         maxnodes=best_hparam_set$maxnodes,
                         ntree=best_hparam_set$ntree,
                         importance=TRUE)

# RMSE on train and test sets
RMSE(pred=predict(rf_final, train), obs=y_train)
RMSE(pred=predict(rf_final, test), obs=y_test)
R2(pred=predict(rf_final, train), obs=y_train)
R2(pred=predict(rf_final, test), obs=y_test)

# Variable importance
library(tibble)
library(tidyr)
library(dplyr)

rf_importance <- cbind(importance(rf_final, type=1), importance(rf_final, type=2)) %>%
  as.data.frame() %>%
  rownames_to_column("variable")

rf_importance$variable <- factor(rf_importance$variable,
                                 levels=rf_importance$variable)

rf_importance %>%
  mutate_if(is.numeric, function(x) x/sum(x)) %>%
  pivot_longer(-variable, names_to="varimp_type", values_to="Importance") %>%
  ggplot(aes(x=variable, y=Importance, fill=varimp_type)) +
  geom_col(position="dodge") +
  ggtitle(sprintf("Spearman(Decrease in node impurity,Permutation)=%.2f",
                  cor(rf_importance$IncNodePurity, rf_importance$`%IncMSE`,
                      method="spearman"))) +
  theme(axis.text.x=element_text(angle = 90, vjust = 0.5, hjust=1),
        legend.position=c(0.15, 0.8)) +
  xlab("Feature")

# Default random forest
rf_default <- randomForest(X_train, y_train)

# RMSE on train and test sets
RMSE(pred=predict(rf_default, train), obs=y_train)
RMSE(pred=predict(rf_default, test), obs=y_test)


# Improve the model: fit a linear regression at the end for balancing and using both models

y_pred_train_rf <- predict(rf_final, train)
y_pred_train_lasso <- predict(lasso, train)
```

```
outputs <- data.frame(cbind(y_pred_train_lasso, y_pred_train_rf))
outputs$true <- y_train

final_lm <- lm(true ~ y_pred_train_lasso + y_pred_train_rf, data=outputs)
summary(final_lm)

#final_lm <- glmnet(outputs[,1:2], outputs[,3], family="gaussian", alpha=0, lambda=0.01)

# Predictions for test set
y_pred_test_rf <- predict(rf_final, test)
y_pred_test_lasso <- predict(lasso, test)
outputs_test <- data.frame(cbind(y_pred_test_lasso, y_pred_test_rf))
final_pred <- predict(final_lm, data.frame(y_pred_train_lasso=y_pred_test_lasso,
                                           y_pred_train_rf=y_pred_test_rf))

# RMSE on train and test sets
RMSE(final_lm$fitted.values, y_train)
RMSE(final_pred, y_test)

# Plots for train and test sets: noise reduction
plot(y_train, y_pred_train_lasso, pch=16, col="blue")
points(y_train, y_pred_train_rf, pch=17, col="red")
points(y_train, final_lm$fitted.values, pch=18, col="green")
lines(y_train, y_train, lwd=2, col="black")

plot(y_test, y_pred_test_lasso, pch=16, col="blue")
points(y_test, y_pred_test_rf, pch=17, col="red")
points(y_test, final_pred, pch=18, col="green")
lines(y_test, y_test, lwd=2, col="black")

# Other way to improve: use the PCA coordinates as inputs
# Not helpful compared to the optimized RF, worse than default RF

new_data <- data.frame(cbind(y, X_pca))
new_train <- new_data[train_ix,]
new_test <- new_data[-train_ix,]
X_train <- new_train[,-1]
y_train <- new_train[,1]
X_test <- new_test[,-1]
y_test <- new_test[,1]

hparam_grid <- as.data.frame(expand.grid(mtry=seq(5, 15, by=3),
                                         maxnodes=seq(50, 100, by=10),
                                         ntree=seq(500, 700, by=100)))
oob_mses <- rep(0.0, nrow(hparam_grid))

for(hparam_idx in 1:nrow(hparam_grid)) {
  this_mtry <- hparam_grid[hparam_idx, 1]
  this_maxnodes <- hparam_grid[hparam_idx, 2]
  this_ntree <- hparam_grid[hparam_idx, 3]
  print(c(this_mtry, this_maxnodes, this_ntree))
  rf <- randomForest(X_train, y_train, mtry=this_mtry, maxnodes=this_maxnodes,
                     ntree=this_ntree)
```

```
  oob_mses[hparam_idx] <- mse(y_train, predict(rf))
}

best_hparam_set <- hparam_grid[which.min(oob_mses),]

rf_new <- randomForest(X_train, y_train, mtry=11, maxnodes=100, ntree=700)

# RMSE on train and test sets
RMSE(pred=predict(rf_new, new_train), obs=y_train)
RMSE(pred=predict(rf_new, new_test), obs=y_test)

# Other way to improve: remove the components that seems irrelevant for RF model

rf_importance <- rf_importance[,-c(2)]
rf_importance[,2] <- rf_importance[,2] / sum(rf_importance[,2])
rf_importance[rf_importance[,2] > 0.01,]
new_data <- data[, rf_importance[rf_importance[,2] > 0.01,]$variable]
new_data <- cbind(y, new_data)
new_train <- new_data[train_ix,]
new_test <- new_data[-train_ix,]
X_train <- new_train[,-1]
y_train <- new_train[,1]
X_test <- new_test[,-1]
y_test <- new_test[,1]

hparam_grid <- as.data.frame(expand.grid(mtry=seq(5, 8, by=3),
                                         maxnodes=seq(50, 100, by=10),
                                         ntree=seq(500, 700, by=100)))

oob_mses <- rep(0.0, nrow(hparam_grid))

for(hparam_idx in 1:nrow(hparam_grid)) {
  this_mtry <- hparam_grid[hparam_idx, 1]
  this_maxnodes <- hparam_grid[hparam_idx, 2]
  this_ntree <- hparam_grid[hparam_idx, 3]
  print(c(this_mtry, this_maxnodes, this_ntree))
  rf <- randomForest(X_train, y_train, mtry=this_mtry, maxnodes=this_maxnodes,
                     ntree=this_ntree)

  oob_mses[hparam_idx] <- mse(y_train, predict(rf))
}

best_hparam_set <- hparam_grid[which.min(oob_mses),]

rf_new <- randomForest(X_train, y_train, mtry=5, maxnodes=80, ntree=500)

# RMSE on train and test sets
RMSE(pred=predict(rf_new, new_train), obs=y_train)
RMSE(pred=predict(rf_new, new_test), obs=y_test)


# B.2

trials_results <- c()
```

```
trials_results_sel <- c()

for(num_trial in 1:100){

  results_rmse <- c()
  results_selection <- c()

  k_out <- 10
  k_in <- 5

  flds_out <- createFolds(data$y, k = k_out, list = TRUE, returnTrain = FALSE)

  for(i in 1:k_out){
    train_set <- data[-flds_out[[i]],]
    test_set <- data[flds_out[[i]],]

    trControl = trainControl(
      method = "cv",
      number = k_in
    )

    lasso.nested <- train(y ~ .,
                          method     = "glmnet",
                          family     = "gaussian",
                          intercept  = FALSE,
                          tuneGrid   = expand.grid(alpha=c(1),
                                                   lambda=seq(1e-5, .05, by=1e-3)),
                          trControl  = trControl,
                          metric     = "RMSE",
                          data       = train_set,
                          preProcess = c("center","scale"))

    rmse <- RMSE(predict(lasso.nested, test_set),
                 test_set$y)

    results_rmse <- c(results_rmse, rmse)

    lasso_nested_importance <- coef(lasso.nested$finalModel,
                                    lasso.nested$bestTune$lambda)
    lasso_nested_importance <- data.frame(matrix(lasso_nested_importance),
                                      row.names=lasso_nested_importance@Dimnames[[1]])
    lasso_nested_importance <- lasso_nested_importance[-1,]
    lasso_nested_ratio_importance <- abs(lasso_nested_importance)/
      sum(abs(lasso_nested_importance))

    results_selection <- cbind(results_selection, lasso_nested_ratio_importance)
  }

  trControl = trainControl(
    method = "cv",
    number = k_in
  )

  lasso.classic <- train(y ~ .,
```

```
                          method     = "glmnet",
                          family     = "gaussian",
                          intercept  = FALSE,
                          tuneGrid   = expand.grid(alpha=c(1),
                                                   lambda=seq(1e-5, .05, by=1e-3)),
                          trControl  = trControl,
                          metric     = "RMSE",
                          data       = data,
                          preProcess = c("center","scale"))

  non_nested_result <- min(lasso.classic$results$RMSE)

  lasso_classic_importance <- coef(lasso.classic$finalModel,
                                   lasso.classic$bestTune$lambda)
  lasso_classic_importance <- data.frame(matrix(lasso_classic_importance),
                                         row.names=lasso_classic_importance@Dimnames[[1]])
  lasso_classic_importance <- lasso_classic_importance[-1,]
  lasso_classic_ratio_importance <- abs(lasso_classic_importance)/
    sum(abs(lasso_classic_importance))

  trials_results <- cbind(trials_results, c(mean(results_rmse), non_nested_result))

  trials_results_sel <- cbind(trials_results_sel, c(rowMeans(results_selection),
                                                    lasso_classic_ratio_importance))

  print(num_trial)
}

p1 <- ggplot() +
  geom_line(aes(x=1:length(trials_results[1,]), y=trials_results[1,]),
            colour="royalblue1") +
  xlab("Index") + ylab("RMSE")

p2 <- ggplot() +
  geom_histogram(aes(x=trials_results[1,]), bins=20, fill="blue", alpha=0.4,
                 colour="royalblue1", lwd=.2) +
  labs(x="RMSE", y="Frequency")

library(ggpubr)
p <- ggarrange(p1, p2, ncol = 1, nrow = 2)

rownames(trials_results_sel) <- c(paste(rep("X", 39), 1:39, sep=""),
                                  paste(rep("X", 39), 1:39, sep=""))

library(zoo)
q <- autoplot(zoo(t(trials_results_sel[1:39,])), facet = NULL) +
  ylab("Importance") +
  theme(legend.position="none")

q_bis <- autoplot(zoo(t(trials_results_sel[1:39,])), facet = NULL) +
  ylab("Importance")

ggarrange(p, q, ncol = 2, nrow = 1)
```

```
q_bis



library(proxyC)
library(caret)
library(tibble)

#Random Forest
trials_results <- c()
trials_results_sel <- c()

for(num_trial in 1:100){

  folds <- createFolds(data$y, k = k_out)

  y = data$y
  x <- unname(as.matrix(data[,2:40]))

  outer_rmse = rep(NA,k_out)
  outer_importance = array(NA, dim=c(39, k_out),
                           dimnames=list(colnames(data[,2:40]), 1:k_out))

  for (i in 1:k_out){

    test_idx = unlist(unname(folds[i]))
    x_train <- x[-test_idx,]
    y_train <- y[-test_idx]

    x_test <- x[test_idx,]
    y_test <- y[test_idx]

    hparam_grid <- as.data.frame(expand.grid(mtry = c(7,8),
                                             maxnodes = c(50,60)))

    oob_mse <- rep(0.0, nrow(hparam_grid))

    for (hparam_idx in 1:nrow(hparam_grid)){
      this_mtry = hparam_grid[hparam_idx,1]
      this_maxnodes = hparam_grid[hparam_idx,2]
      rf <- randomForest(x_train, y_train, mtry = this_mtry,
                         maxnodes = this_maxnodes, ntree=800)
      oob_mse[hparam_idx] <- mse(y_train, predict(rf))
    }

    best_hparam_set <- hparam_grid[which.min(oob_mse),]
    rf_final <- randomForest(x_train, y_train,
                             mtry = best_hparam_set$mtry,
                             maxnodes = best_hparam_set$maxnodes,
                             importance = TRUE, ntree=800)

    yhat <- predict(rf_final, newdata = x_test)
    outer_rmse[i] <- sqrt(mse(y_test, yhat))
    rf_importance <- importance(rf_final, type = 2) %>%
```

```
      as.data.frame(.) %>%
      rownames_to_column("variable")
    outer_importance[,i] <- abs(rf_importance[,2]) / sum(abs(rf_importance[,2]))
  }

  trials_results <- c(trials_results, mean(outer_rmse))

  trials_results_sel <- cbind(trials_results_sel, rowMeans(outer_importance))

  print(num_trial)

}

p1 <- ggplot() +
  geom_line(aes(x=1:length(trials_results), y=trials_results), colour="royalblue1") +
  xlab("Index") + ylab("RMSE")

p2 <- ggplot() +
  geom_histogram(aes(x=trials_results), bins=20, fill="blue", alpha=0.4,
                 colour="royalblue1", lwd=.2) +
  labs(x="RMSE", y="Frequency")

p <- ggarrange(p1, p2, ncol = 1, nrow = 2)

q <- autoplot(zoo(t(trials_results_sel[1:39,])), facet = NULL) +
  ylab("Importance") +
  theme(legend.position="none")

q_bis <- autoplot(zoo(t(trials_results_sel[1:39,])), facet = NULL) +
  ylab("Importance")

ggarrange(p, q, ncol = 2, nrow = 1)

q_bis

plot(trials_results[1,], type="l", col="blue", ylim=c(min(trials_results),
                                                      max(trials_results)))
lines(trials_results[2,], col="red")


# Question 2: Gaussian Processes

library(ggplot2)
library(dplyr)
library(data.table)

data <- read.csv("dataQ2.csv")

# Plot all the data points
ggplot(data, aes(x=t, y=temp, colour=d)) +
  geom_point() +
  xlab("Days since the beginning of the study") + ylab("Temperature (°C)") +
  theme(legend.position = c(0.77, 0.25))
```

```
# Part 1: d = 0
data_0 <- data[data$d==0,]
T <- nrow(data_0)
# Define the design matrix
X <- matrix(data_0$t)
# Define the response vector
y <- matrix(data_0$temp)

# Center the response vector
mu <- mean(y)
y <- y - mu

# Plot centered data
ggplot() +
  geom_point(aes(x=X, y=y)) +
  xlab("Days since the beginning of the study") + ylab("Centered temperature (°C)")

# Defines the kernel functions
rbf_kernel <- function(X1, X2=NULL, sigma_f=1.0, l=1.0) {
  # Calculate a kernel matrix using the radial basis function (RBF)
  # kernel function
  #
  # Args:
  # - X1: matrix with shape n1 x 1
  # - X2: matrix with shape n2 x 1 or NULL (default), in which case X2=X1
  # - sigma_f: output stddev (hyperparameter)
  # - l: lengthscale (hyperparameter)
  #
  # Returns an n1 x n2 kernel matrix
  if(is.null(X2)) X2 <- X1
  n1 <- nrow(X1)
  n2 <- nrow(X2)
  K <- matrix(0, n1, n2)
  for(i in 1:n1) {
    for(j in 1:n2) {
      K[i,j] <- -(X1[[i]]-X2[[j]])**2.0
    }
  }
  return(sigma_f**2.0*exp(K/(2.0*l**2.0)))
}

periodic_kernel <- function(X1, X2=NULL, sigma_f=1.0, l=1.0, period=1.0) {
  # Calculate a kernel matrix using the periodic kernel function
  #
  # Args:
  # - X1: matrix with shape n1 x 1
  # - X2: matrix with shape n2 x 1 or NULL (default), in which case X2=X1
  # - sigma_f: output stddev (hyperparameter)
  # - l: lengthscale (hyperparameter)
  # - period: time period (hyperparameter)
  #
  # Returns an n1 x n2 kernel matrix
  if(is.null(X2)) X2 <- X1
  n1 <- nrow(X1)
```

```
  n2 <- nrow(X2)
  K <- matrix(0, n1, n2)
  for(i in 1:n1) {
    for(j in 1:n2) {
      K[i,j] <- -2.0*sin((pi*abs(X1[[i]]-X2[[j]]))/period)**2.0
    }
  }
  return((sigma_f**2.0)*exp(K/(l**2.0)))
}

locally_periodic_kernel <- function(X1, X2, sigma_f, l_rbf, l_per, period) {
  K_rbf <- rbf_kernel(X1, X2, 1.0, l_rbf)
  K_per <- periodic_kernel(X1, X2, 1.0, l_per, period)
  return(sigma_f**2.0 * K_rbf * K_per)
}

# Define each kernel
SIGMA_N <- 0.5
kernel_fns <- list(
  "RBF"=function(x1, x2=NULL) rbf_kernel(x1, x2, sigma_f=4, l=9.8),
  "Periodic"=function(x1, x2=NULL) periodic_kernel(x1, x2, sigma_f=4, l=1.1,
                                                   period=64),
  "Loc. periodic"=function(x1, x2=NULL) locally_periodic_kernel(x1, x2, sigma_f=4,
                                                                l_rbf=9.8,
                                                                l_per=10, period=68)
)

# Define fit and likelihood functions
fit_gp_posterior <- function(X, y, X_star, k_fn, sigma_n) {
  # Returns the mean and variance of the GP posterior
  # for data (X,y) and kernel function k_fn at test points X*
  # sigma_n is the noise std

  # compute kernels
  K_xx <- k_fn(X) # K(X,X)
  K_xxs<- k_fn(X, X_star) # K(X,X*)
  K_xsxs <- k_fn(X_star) # K(X*,X*)

  # calculate posterior
  sigma_n_sq <- sigma_n**2.0
  mu <- t(K_xxs) %*% solve(K_xx + sigma_n_sq*diag(nrow(K_xx)), y)
  Sigma <- K_xsxs - t(K_xxs) %*% solve(K_xx + sigma_n_sq*diag(nrow(K_xx)), K_xxs)

  return(list(mu=mu, Sigma=Sigma))
}

log_marg_likelihood <- function(X, y, kernel_fn, sigma_n) {
  # calculate the log marginal likelihood given X,y, noise
  # std sigma_n and a kernel function
  Kxx <- kernel_fn(X)
  A <- Kxx + sigma_n**2.0*diag(nrow(Kxx))
  lml <- -0.5*(t(y) %*% solve(A, y) + log(det(A) + 1e-9) + nrow(X)*log(2.0*pi))
  return(lml)
}
```

```
# Calculate the GP posterior for each model
gp_posterior <- lapply(
  kernel_fns,
  function(fn) fit_gp_posterior(X, y, matrix(seq(0, 50, length.out=100), 100, 1),
                                fn, SIGMA_N)
)

# Same for MSE estimation
gp_posterior_mse <- lapply(
  kernel_fns,
  function(fn) fit_gp_posterior(X, y, X, fn, SIGMA_N)
)
mean((gp_posterior_mse$RBF$mu - y)^2)
mean((gp_posterior_mse$Periodic$mu - y)^2)
mean((gp_posterior_mse$`Loc. periodic`$mu - y)^2)

# Calculate the log marginal likelihood for each model
all_lmls <- lapply(kernel_fns,
                   function(fn) log_marg_likelihood(X, y, fn, SIGMA_N)
) %>%
  lapply(as.data.frame) %>%
  rbindlist(idcol="kernel") %>%
  rename(lml=V1)

all_lmls

# Plot the final fitted GP
lapply(gp_posterior,
       function(gpp) data.frame(mean_f=gpp$mu, std_f=diag(gpp$Sigma)**0.5,
                                t=seq(0, 50, length.out=100))) %>%
  rbindlist(idcol="kernel") %>%
  left_join(all_lmls, by="kernel") %>%
  mutate(key=sprintf("%s, lml=%.1f", kernel, lml)) %>%
  ggplot(aes(x=t, y=mean_f)) +
  geom_point(data=data.frame(x=X, y=y), aes(x=x, y=y), size=0.5) +
  geom_line(aes(colour=key)) +
  geom_ribbon(aes(ymin=mean_f-1.96*std_f, ymax=mean_f+1.96*std_f, fill=key),
              alpha=0.3) +
  theme(legend.position="top") +
  facet_wrap(~key, ncol=1) +
  xlab("Days since the beginning of the study") + ylab("Centered temperature (°C)")

# Optimize kernel parameters

# RBF
# Define the grid
sigma_f_s <- seq(0.001, 15, by=0.1)
l_s <- seq(0.001, 20, by=0.1)
SIGMA_N_s <- seq(0.1, 2, by=0.1)
all_lmls <- array(NA, dim=c(length(sigma_f_s), length(l_s),length(SIGMA_N_s)))

for (i in 1:length(sigma_f_s)){
```

```
  for (j in 1:length(l_s)){
    for (k in 1:length(SIGMA_N_s)){
      sigma_f <- sigma_f_s[i]
      l <- l_s[j]
      SIGMA_N <- SIGMA_N_s[k]
      kernel_fn <- function(x1, x2=NULL) rbf_kernel(x1, x2, sigma_f=sigma_f, l=l)
      all_lmls[i, j, k] <- log_marg_likelihood(X, y, kernel_fn, SIGMA_N)
    }
  }
  print(i)
}

max_index <- which(all_lmls == max(all_lmls), arr.ind = TRUE)
sigma_f_s[max_index[1]] # 4
l_s[max_index[2]] # 9.8
SIGMA_N_s[max_index[3]] # 0.5
max(all_lmls)


# Periodic
# Define the grid
sigma_f_s <- seq(0.001, 7, by=0.1)
l_s <- seq(0.1, 5, by=0.1)
SIGMA_N_s <- seq(0.001, 2, by=0.1)
period_s <- seq(30, 70, by=1)
all_lmls <- array(NA, dim=c(length(sigma_f_s), length(l_s), length(SIGMA_N_s),
                            length(period_s)))

for (i in 1:length(sigma_f_s)){
  for (j in 1:length(l_s)){
    for (k in 1:length(SIGMA_N_s)){
      for (p in 1:length(period_s)){
        sigma_f <- sigma_f_s[i]
        l <- l_s[j]
        SIGMA_N <- SIGMA_N_s[k]
        period <- period_s[p]
        kernel_fn <- function(x1, x2=NULL) periodic_kernel(x1, x2, sigma_f=sigma_f,
                                                           l=l, period=period)
        all_lmls[i, j, k, p] <-  log_marg_likelihood(X, y, kernel_fn, SIGMA_N)
      }
    }
  }
  print(i)
}

max_index <- which(all_lmls == max(all_lmls), arr.ind = TRUE)
sigma_f_s[max_index[1]] # 4
l_s[max_index[2]] # 1.1
SIGMA_N_s[max_index[3]] # 0.5
period_s[max_index[4]] # 64
max(all_lmls)


# Locally periodic
```

```
# Define the grid
sigma_f_s <- seq(3.8, 4.2, by=0.1)
l_rbf_s <- seq(9, 11, by=0.2)
l_per_s <- seq(1, 10, by=0.2)
SIGMA_N_s <- seq(0.4, 0.5, by=0.1)
period_s <- seq(62, 68, by=1)
all_lmls <- array(NA, dim=c(length(sigma_f_s), length(l_rbf_s), length(l_per_s),
                            length(SIGMA_N_s), length(period_s)))

for (i in 1:length(sigma_f_s)){
  for (j in 1:length(l_rbf_s)){
    for (k in 1:length(l_per_s)){
      for (l in 1:length(SIGMA_N_s)){
        for (m in 1:length(period_s)){
          sigma_f <- sigma_f_s[i]
          l_rbf <- l_rbf_s[j]
          l_per <- l_per_s[k]
          SIGMA_N <- SIGMA_N_s[l]
          period <- period_s[m]
          kernel_fn <- function(x1, x2=NULL) locally_periodic_kernel(x1, x2,
                                                                     sigma_f=sigma_f,
                                                                     l_rbf=l_rbf,
                                                                     l_per = l_per,
                                                                     period=period)
          all_lmls[i, j, k, l, m] <-  log_marg_likelihood(X, y, kernel_fn, SIGMA_N)
        }
      }
    }
  }
  print(i)
}

max_index <- which(all_lmls == max(all_lmls), arr.ind = TRUE)
sigma_f_s[max_index[1]] # 4
l_rbf_s[max_index[2]] # 9.8
l_per_s[max_index[3]] # 10
SIGMA_N_s[max_index[4]] # 0.5
period_s[max_index[5]] # 68
max(all_lmls)


# Best model: RBF or locally periodic
loc_rbf <- function(x1, x2=NULL) locally_periodic_kernel(x1, x2, sigma_f=4,
                                                         l_rbf=9.8, l_per=10,
                                                         period=68)

# Predict temperature at t = 35
Xs <- matrix(35)
# Calculate GP posterior at the test points
gp_posterior_with_pred <- fit_gp_posterior(X, y, Xs, loc_rbf, SIGMA_N)

pred_temp_35 <- gp_posterior_with_pred$mu + mu
upper_pred_temp_35 <- gp_posterior_with_pred$mu + mu +
  qnorm(0.995)*sqrt(gp_posterior_with_pred$Sigma)
```

```
lower_pred_temp_35 <- gp_posterior_with_pred$mu + mu -
  qnorm(0.995)*sqrt(gp_posterior_with_pred$Sigma)

print(c(lower_pred_temp_35, pred_temp_35, upper_pred_temp_35))

# Probability more than 13 degrees
1 - pnorm(mu-13, mean=gp_posterior_with_pred$mu, sd=sqrt(gp_posterior_with_pred$mu))

# Plot fitted GP
gp_posterior_with_pred <- fit_gp_posterior(X, y, matrix(seq(0, 50, length.out = 100),
                                                        100, 1),
                                           loc_rbf, SIGMA_N)

x <- seq(0, 50, length.out = 100)
mu_ <- gp_posterior_with_pred$mu
upper <- gp_posterior_with_pred$mu+qnorm(0.975)*diag(gp_posterior_with_pred$Sigma)**0.5
lower <- gp_posterior_with_pred$mu-qnorm(0.975)*diag(gp_posterior_with_pred$Sigma)**0.5

X <- matrix(data_0$t[1:T])
y <- matrix(data_0$temp[1:T])

data_pred <- data.frame(X=x, mu=mu_+mu, lower=lower+mu, upper=upper+mu)

ggplot(data=data_pred, aes(X, mu)) +
  geom_ribbon(aes(ymin=lower, ymax=upper), fill= "red", alpha = 0.2) +
  geom_line(aes(X, mu), colour="red") +
  geom_point(data=data_0, aes(x=t, y=temp)) +
  xlab("Days since the beginning of the study") + ylab("Temperature (°C)")



library(ggplot2)
library(dplyr)
library(data.table)
library(plotly)

# Import data
data <- read.csv("dataQ2.csv")
T <- nrow(data)
# Create design matrix and response vector
X <- as.matrix(data[,1:2])
y <- matrix(data$temp)

# Define useful functions (fit, likelihood, kernels)
fit_gp_posterior <- function(X, y, X_star, k_fn, sigma_n) {
  # Returns the mean and variance of the GP posterior
  # for data (X,y) and kernel function k_fn at test points X*
  # sigma_n is the noise std

  # compute kernels
  K_xx <- k_fn(X) # K(X,X)
  K_xxs<- k_fn(X, X_star) # K(X,X*)
  K_xsxs <- k_fn(X_star) # K(X*,X*)
```

```
  # calculate posterior
  sigma_n_sq <- sigma_n**2.0
  mu <- t(K_xxs) %*% solve(K_xx + sigma_n_sq*diag(nrow(K_xx)), y)
  Sigma <- K_xsxs - t(K_xxs) %*% solve(K_xx + sigma_n_sq*diag(nrow(K_xx)), K_xxs)

  return(list(mu=mu, Sigma=Sigma))
}

log_marg_likelihood <- function(X, y, kernel_fn, sigma_n) {
  # calculate the log marginal likelihood given X,y, noise
  # std sigma_n and a kernel function
  Kxx <- kernel_fn(X)
  A <- Kxx + sigma_n**2.0*diag(nrow(Kxx))
  lml <- -0.5*(t(y) %*% solve(A, y) + log(det(A) + 1e-9) + nrow(X)*log(2.0*pi))
  return(lml)
}

rbf_kernel <- function(X1, X2=NULL, sigma_f=1.0, l=1.0) {
  # Calculate a kernel matrix using the radial basis function (RBF)
  # kernel function
  #
  # Args:
  # - X1: matrix with shape n1 x 1
  # - X2: matrix with shape n2 x 1 or NULL (default), in which case X2=X1
  # - sigma_f: output stddev (hyperparameter)
  # - l: lengthscale (hyperparameter)
  #
  # Returns an n1 x n2 kernel matrix
  if(is.null(X2)) X2 <- X1
  n1 <- nrow(X1)
  n2 <- nrow(X2)
  K <- matrix(0, n1, n2)
  for(i in 1:n1) {
    for(j in 1:n2) {
      K[i,j] <- -(X1[[i]]-X2[[j]])**2.0
    }
  }
  return(sigma_f**2.0*exp(K/(2.0*l**2.0)))
}

periodic_kernel <- function(X1, X2=NULL, sigma_f=1.0, l=1.0, period=1.0) {
  # Calculate a kernel matrix using the periodic kernel function
  #
  # Args:
  # - X1: matrix with shape n1 x 1
  # - X2: matrix with shape n2 x 1 or NULL (default), in which case X2=X1
  # - sigma_f: output stddev (hyperparameter)
  # - l: lengthscale (hyperparameter)
  # - period: time period (hyperparameter)
  #
  # Returns an n1 x n2 kernel matrix
  if(is.null(X2)) X2 <- X1
  n1 <- nrow(X1)
  n2 <- nrow(X2)
```

```
  K <- matrix(0, n1, n2)
  for(i in 1:n1) {
    for(j in 1:n2) {
      K[i,j] <- -2.0*sin((pi*abs(X1[[i]]-X2[[j]]))/period)**2.0
    }
  }
  return((sigma_f**2.0)*exp(K/(l**2.0)))
}

locally_periodic_kernel <- function(X1, X2, sigma_f, l_rbf, l_per, period) {
  K_rbf <- rbf_kernel(X1, X2, 1.0, l_rbf)
  K_per <- periodic_kernel(X1, X2, 1.0, l_per, period)
  return(sigma_f**2.0 * K_rbf * K_per)
}

linear_kernel <- function(X1, X2=NULL, sigma_b=0, sigma_v=1, c=0) {
  # Returns an n1 x n2 kernel matrix
  if(is.null(X2)) X2 <- X1
  n1 <- nrow(X1)
  n2 <- nrow(X2)
  K <- matrix(0, n1, n2)
  for(i in 1:n1) {
    for(j in 1:n2) {
      K[i,j] <- sigma_b^2 + sigma_v^2 * (X1[[i]]-c) * (X2[[j]]-c)
    }
  }
  return(K)
}

product_kernel <- function(X1, X2=NULL, sigma_f, l_rbf, l_per, period,
                           sigma_b, sigma_v, c) {
  if (is.null(X2)){
    K_loc_rbf <- locally_periodic_kernel(matrix(X1[, 1]), NULL, sigma_f, l_rbf,
                                         l_per, period)
    K_linear <- linear_kernel(matrix(X1[, 2]), NULL,  sigma_b, sigma_v, c)

  }
  else {
    K_loc_rbf <- locally_periodic_kernel(matrix(X1[, 1]), matrix(X2[, 1]),
                                         sigma_f, l_rbf, l_per, period)
    K_linear <- linear_kernel(matrix(X1[, 2]), matrix(X2[, 2]),  sigma_b,
                              sigma_v, c)
  }
  return(K_loc_rbf * K_linear * K_linear)
}

# We notice that T decreases with d and is locally periodic wrt t
mean_temp <- data %>%
  group_by(d) %>%
  summarise(mean_T=mean(temp))

ggplot(data=mean_temp, aes(x=d, y=mean_T)) +
  geom_point() +
  stat_smooth(method = "lm", formula = y ~ I(x^2) + x, size = 1) +
```

```
  ylab("Average temperature (°C)") + xlab("Distance")

ggplot(data=data, aes(x=d, y=temp, colour=t)) +
  geom_point() +
  ylab("Temperature (°C)") + xlab("Distance")

# Hence, we try to find a kernel regression using the product of linear and
# locally periodic kernels
# Define the grid (linear)
sigma_f_s <- seq(0.5, 1.5, 0.5)
l_rbf_s <- seq(14, 16, 1)
l_per_s <- seq(16, 18, 1)
period_s <- seq(35, 45, 5)
sigma_b_s <- seq(0, 20, 10)
sigma_v_s <- seq(0.1, 0.3, 0.1)
c_s <- seq(20, 30, 10)
SIGMA_N_s <- seq(0.7, 0.9, 0.1)

# Define the grid (quadratic)
sigma_f_s <- seq(0.4, 0.6, 0.1)
l_rbf_s <- seq(13, 16, 1)
l_per_s <- seq(13, 16, 1)
period_s <- seq(0.5, 1.1, 0.2)
sigma_b_s <- seq(3, 5, 1)
sigma_v_s <- seq(0.1, 0.5, 0.2)
c_s <- seq(0, 1, 1)
SIGMA_N_s <- seq(0.8, 1, 0.1)

all_lmls <- array(NA, dim=c(length(sigma_f_s), length(l_rbf_s), length(l_per_s),
                            length(period_s), length(sigma_b_s), length(sigma_v_s),
                            length(c_s), length(SIGMA_N_s)))

for(i in 1:length(sigma_f_s)){
  for(j in 1:length(l_rbf_s)){
    for(k in 1:length(l_per_s)){
      for(l in 1:length(period_s)){
        for(m in 1:length(sigma_b_s)){
          for(n in 1:length(sigma_v_s)){
            for(o in 1:length(c_s)){
              for(p in 1:length(SIGMA_N_s)){
                sigma_f <- sigma_f_s[i]
                l_rbf <- l_rbf_s[j]
                l_per <- l_per_s[k]
                period <- period_s[l]
                sigma_b <- sigma_b_s[m]
                sigma_v <- sigma_v_s[n]
                c <- c_s[o]
                SIGMA_N <- SIGMA_N_s[p]

                kernel_fn <- function(x1, x2=NULL) product_kernel(x1, x2, sigma_f,
                                                                  l_rbf, l_per,
                                                                  period, sigma_b,
                                                                  sigma_v, c)
```

```
                all_lmls[i, j, k, l, m, n, o, p] <-  log_marg_likelihood(X, y,
                                                                         kernel_fn,
                                                                         SIGMA_N)

                print(c(i, j, k, l, m, n, o, p))
              }
            }
          }
        }
      }
    }
  }
}

max_index = which(all_lmls == max(all_lmls), arr.ind=TRUE)

sigma_f_s[max_index[1]]
l_rbf_s[max_index[2]]
l_per_s[max_index[3]]
period_s[max_index[4]]
sigma_b_s[max_index[5]]
sigma_v_s[max_index[6]]
c_s[max_index[7]]
SIGMA_N_s[max_index[8]]
max(all_lmls)

# Optimal parameters
# Linear
sigma_f <- 0.5
l_rbf <-  15
l_per <- 16
period <- 40
sigma_b <- 20
sigma_v <- 0.3
c <- 30
SIGMA_N <- 0.9

# Quadratic
sigma_f <- 0.5
l_rbf <-  15
l_per <- 13
period <- 0.5
sigma_b <- 4
sigma_v <- 0.1
c <- 1
SIGMA_N <- 0.9

kernel_fn <- function(x1, x2=NULL) product_kernel(x1, x2, sigma_f,
                                                  l_rbf, l_per,
                                                  period, sigma_b,
                                                  sigma_v, c)

log_marg_likelihood(X, y, kernel_fn, SIGMA_N)
```

```
# Plot the fitted values
d_star <- matrix(0, 51, 1)
t_star <- seq(0, 50, 1)
X_star = cbind(t_star, d_star)

temp_star <- fit_gp_posterior(X, y, X_star, kernel_fn, SIGMA_N)$mu

data_pred <- data.frame("d"=d_star, "t"=t_star, "temp"=temp_star)

d_s = seq(0, 19, by=1)
for (i in 1:length(d_s)){
  d <- d_s[i]
  d_star <- matrix(d, 51, 1)
  t_star <- seq(0, 50, 1)
  X_star <- cbind(t_star,d_star)
  temp_star <- fit_gp_posterior(X, y, X_star, kernel_fn, SIGMA_N)$mu
  data_ <- data.frame("d"=d_star,"t"=t_star, "temp"=temp_star)
  data_pred <- rbind(data_pred, data_)
}

plot_3D <- plot_ly(x=0:50, y=0:20, z=matrix(data_pred$temp, nrow=21, ncol=51,
                                            byrow=TRUE)) %>%
  add_surface(colorscale="Jet") %>%
  add_trace(x=data$t, y=data$d, z=data$temp, name = 'trace 0', mode = 'markers',
            marker = list(
              color = 'rgb(0, 0, 0)',
              size = 2)) %>%
  layout(scene = list(xaxis = list(title = "Days"), yaxis = list(title = "Distance"),
                      zaxis = list(title = "Temperature")))

plot_3D

# Plot the temperature at day 55 as a function of d
d_star <- seq(0, 20, 1)
t_star <- matrix(55, length(d_star), 1)
X_star <- cbind(t_star, d_star)

temp_star <- fit_gp_posterior(X, y, X_star, kernel_fn, SIGMA_N)

mean_55 <- temp_star$mu
sd_55 <- diag(temp_star$Sigma)**0.5

data_ <- data.frame(temp=mean_55, d=d_star, lower=mean_55-qnorm(0.975)*sd_55,
                    upper=mean_55+qnorm(0.975)*sd_55)

ggplot(data_, aes(x=d, y=temp)) +
  geom_line(colour="red") +
  geom_ribbon(aes(ymin=lower, ymax=upper), alpha=0.2, fill="red") +
  xlab("Distance") + ylab("Temperature (°C)")
```