Package FunQuant part II

Charlie SIRE[1,2,3]

Supervisors: R. LE RICHE[3], D. RULLIERE[3], J. ROHMER[2], L. PHEULPIN[1], Y. RICHET[1]

[1]IRSN

[2]BRGM

[3]Mines Saint-Etienne and CNRS,LIMOS

September 21, 2023

Section 1

Structure of the metamodel

Metamodel for maps [Perrin et. al]

1. FPCA : Write every map $Y(x)$ as a linear combination of n_{pc} maps :

$$Y(x) = t_1(x)Y_1^{\mathrm{pca}} + \cdots + t_{n_{pc}}(x)Y^{\mathrm{pca}_{n_{pc}}}$$

2. Gaussian process regression potentially combined with classification on every axis to predict $(\hat{t}_1(x^\star), \ldots, \hat{t}_{n_{pc}}(x^\star))$ for a new $x^\star$

$\Longrightarrow$ Work with a large sample of predicted maps $(\hat{Y}(\tilde{X}^k))_{k=1}^n$

More on FPCA

- Find a decomposition of the training maps on a functional orthonormal basis $\Phi = (\Phi_1, \dots, \Phi_K)^\top$: $Y(x^i) = \Phi^\top \alpha(x^i)$
- Reduce the number of basis functions by keeping the $\tilde{K}$ most important ones, $\tilde{\Phi}$: $Y(x^i) = \tilde{\Phi}^\top \tilde{\alpha}(x^i)$
- Perform a PCA on the selected coordinates to get the principal components $(t(x^i))_{i=1,\dots,n_{\mathrm{train}}}$ with $t(x^i) = (t_1(x^i), \dots, t_{n_{\mathrm{pc}}}(x^i))^\top$ and the $n_{\mathrm{pc}} \times \tilde{K}$ projection matrix Ω : $\tilde{\alpha}(x^i) \simeq \Omega^\top t(x^i)$.

Optional classification

Problem: GPR ill-suited for forecasting phenomena that are inherently impossible below certain thresholds.

Then: Propose classification to distinguish between empty maps and the others

Summary:

- An optional classification that predict whether the output is zero or not
- A FPCA step to reduce the dimension of the output space
- Gaussian process regression on each FPCA axis

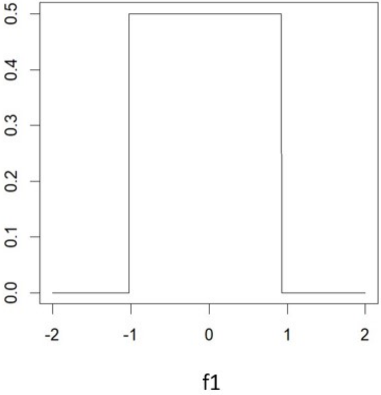
Hyperparameters

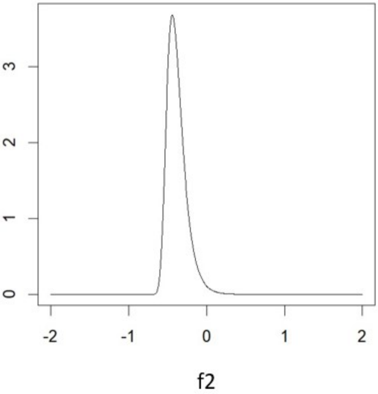
- If classification: The hyperparameters of the classifier
- $\tilde{K}$: Number of vectors in the basis to keep in the functional decomposition (denoted ncoeff in FunQuant)
- n_{pc} : Number of principal components

Section 2

Toy problem

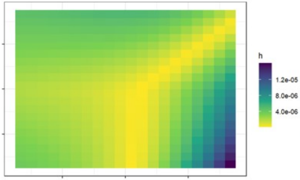
Inputs density



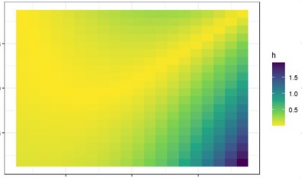


And we have $f_X = f_1 f_2$

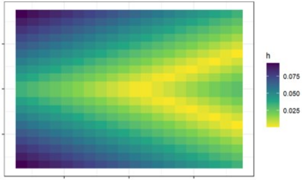
Output maps



$x_1 = 0.8, x_2 = -0.2$



$x_1 = 0.8, x_2 = 0.8$



$x_1 = 0, x_2 = 0.3$

Y is built such that $Y(x) \approx 0$ for $x_2 < -0.1$

Data : Design of experiments of 200 points with the sobol sequence in $[-1, 1]^2$

```
design = (sobol(200,2))*2-1
outputs = func2D(design)
```

Section 3

FPCA Tuning

FPCA tuning by RMSE

For every pair $(\tilde{K}, n_{\mathrm{pc}})$, we obtain m predicted maps by loo, k_fold, training_test

Then, we can compute the RMSE map:

$$y^{\mathrm{RMSE}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}.$$

FPCA tuning by RMSE II

Three tuning functions by rmse:

- rmse_loo
- rmse_k_fold
- rmse_training_test

Main arguments:

- outputs (_train and _test for training_test)
- design (_train and _test for training_test)
- ncoeff_vec
- npc_vec

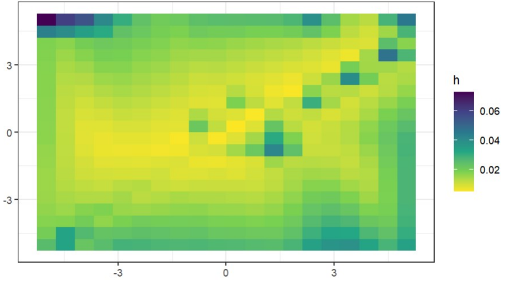
+ parameters of the wavelets decomposition

+ parameters of the kriging objects

Example with rmse_loo

```
list_rmse_loo = rmse_loo(
  outputs = outputs,
  design = design,
  npc_vec = 3:6,
  ncoeff_vec = c(100,250,400))

list_rmse_loo[[1]]
```



FPCA tuning with membership probabilities

For every pair $(\tilde{K}, n_{\mathrm{pc}})$ and for each fold k, we obtain m predicted maps by loo, k_fold or training_test

Then, for a given set of prototype maps, we can compare p the membership probabilities with the true maps and $\hat{p}$ the membership probabilities with the predicted maps

The tuning functions return $\frac{p-\hat{p}}{p}$

FPCA tuning with membership probabilities II

- probas_training_test
- probas_loo
- probas_k_fold

Same argument that rmse tuning

+ prototypes: a set of prototypes
+ density_ratio: the density ratio computed for the training maps, often associated to uniform distribution
+ distance_func: the distance between two maps

Example with probas_loo

```
list_probas_loo = probas_loo(
  outputs = outputs,
  design = design,
  density_ratio = density_ratio
  prototypes = prototypes_apriori,
  npc_vec =3:6,
  ncoeff_vec =c(100,250,400))

list_probas_loo$probas_pred[1,]

list_probas_loo$error[1,]
```

	ncoeff	npc	1	2	3	4	5	6
1	100	3	0.9601188	0.0473694	0.0009766928	0.003282139	0.0003660801	0.001624763

	ncoeff	npc	1	2	3	4	5	6
1	100	3	0.006125355	0.4305926	0.06802295	0.715503	0	0.008887564

Section 4

Tuning classification

All classification tuning functions

- rf_rmse_training_test
- rf_rmse_k_fold
- rf_probas_training_test
- rf_probas_k_fold
- rf_classif_training_test
- rf_classif_k_fold

Example with rf_rmse_k_fold I

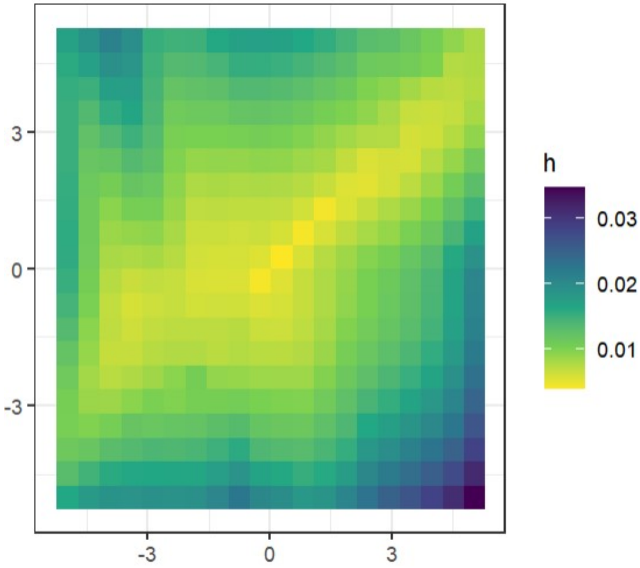
```
df_search = expand.grid(classwt1 = seq(0.2,0.8,0.3),
                        nodesize = c(1,3,5))
list_search = list("nodesize" = as.list(df_search[,2]),
                   "classwt" = lapply(1:nrow(df_search),
                    function(i){
                    c(df_search[i,1], 1-df_search[i,1])}))
```

Example with rf_rmse_k_fold II

```
list_rf_rmse_k_fold = rf_rmse_k_fold(
  design = design,
 outputs = outputs,
 threshold_classification = 0.01,
  threshold_fpca = 0.01,
 list_search = list_search,
 nb_folds = 10,
 ncoeff = 250,
 npc = 6)

 list_rf_rmse_k_fold$error
```

Example with rf_rmse_k_fold III



Example with rf_classif_k_fold

```r
sum_depth = Vectorize(function(i){
  sum(outputs[,,i])})(1:dim(outputs)[3])

list_rf_classif_k_fold = rf_classif_k_fold(
  design = design,
     outputs = as.factor(sum_depth > 0.01,
 list_search = list_search,
 nb_folds = 10)
)

 list_rf_classif_k_fold[[1]]
```

```
  [1] 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2
 [48] 1 2 2 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 2 1
 [95] 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2
[142] 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 2 2 1
[189] 2 1 2 1 2 1 2 1 2 1 2 1
```

Section 5

Fit metamodel and predict

Fit metamodel

```
mm = fit_metamodel(
design_train = design,
outputs_train = outputs,
ncoeff = 250,
npc = 6,
classification =TRUE,
control_classification = list("nodesize" = 1,
                                "classwt" = c(0.5,0.5)),
threshold_classification = 0.01)}
threshold_fpca = 0.01)}
```

The arguments kernel, regmodel, normalize, optim, objective of the Kriging function of rlibkriging can be provided

Maps prediction

```
predict_outputs(
  metamodel_fitted = mm,
  design_test =design_test)
```

If the metamodel is not provided, predict_outputs needs the arguments of the function fit_metamodel to build the metamodel