Prévision de la demande nette d'électricité en France *

Mohamed Amine GRINI, Marcos LAHOZ, Samuel MOLANO

Table des matières

1 Introduction

La prédiction de la demande électrique est clé pour assurer l'équilibre entre l'offre et la demande d'électricité. En effet un déséquilibre au sein de ce marché peut entraîner un gâchis d'électricité ou encore des *blackouts*.
Notre objectif est de construire des modèles statistiques permettant prédire de façon précise la demande nette énergétique française entre septembre 2022 et septembre 2023. Pour faire ceci, nous disposons de données allant de 2013 jusqu'à 2022.
Nos données comportent des variables calendaires (*time of year*, jour de la semaine, vacances, jours fériés), des variables météorologiques (température, vitesse du vent, nébulosité) et des données de production et de consommation passées (production solaire, éolienne et consommation totale).

2 Analyse et visualisation des données

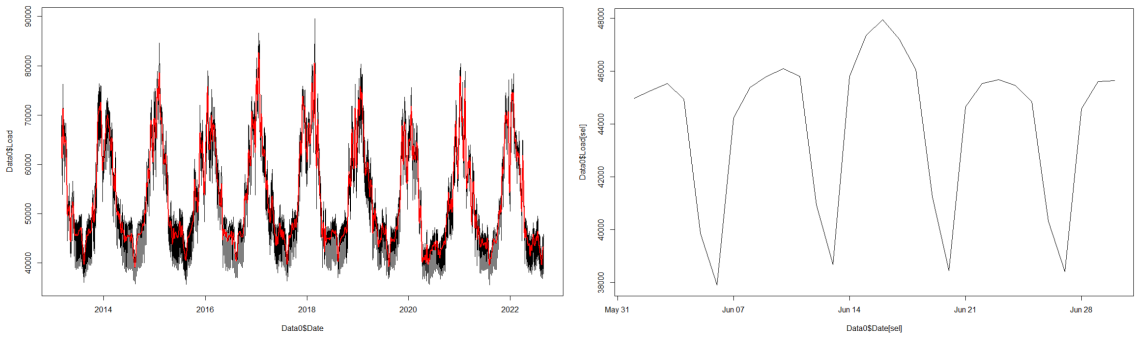
Notre jeu de données ne comporte pas de variables manquantes. Il comporte 40 variables (dont 35 qui peuvent être utilisées pour la prédiction) pendant une période de 3471 jours. Nous cherchons à prédire la variable `Net_demand`, qui correspond à la quantité d'électricité qui ne peut pas être fournie par la production des énergies solaires et éoliennes. On déduit alors la formule suivante :

```
Net_demand = Load - (Wind_power + Solar_power)
```

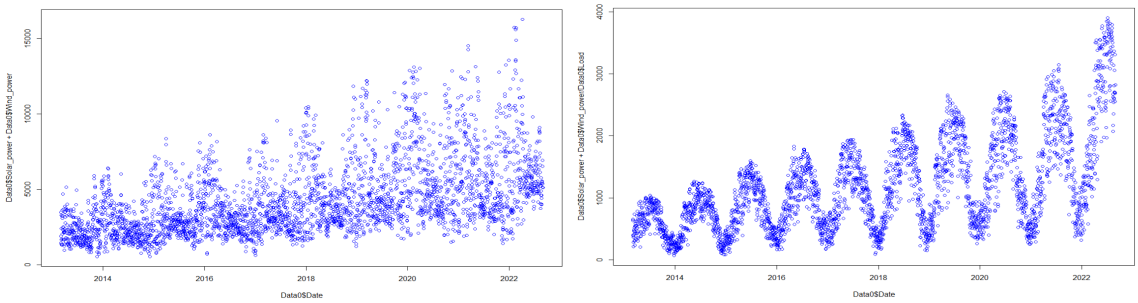
Les données d'électricité comportent des cycles annuels avec des demandes d'électricité plus élevées pendant l'hiver et plus basses pendant l'été et hebdomadaires, avec des consommations plus élevées en

*https://github.com/SamuelMolano/ModelisationPredictive

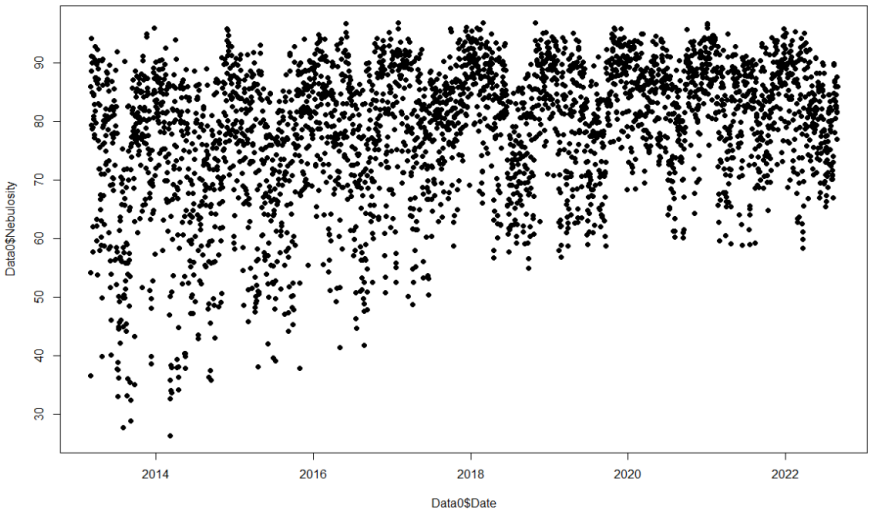
semaine que pendant les weekends. Toutefois, nos données comportent une tendance : les consommations électriques peuvent varier d'une année à l'autre. Depuis l'arrivée du COVID, on observe des consommations électriques plus faibles.



La demande nette d'électricité varie aussi en fonction la production d'énergies renouvelables, qui est à la hausse mais qui continue à être insuffisante pour satisfaire les besoins énergetiques de la France, particulièrement pendant l'hiver.



La nebulosité est une variable météorologique qui connait une évolution particulière, avec une tendance observée qui ne semble pas correspondre à la réalité. Il s'agit probablement d'un changement d'instruments de mesure qu'on devra prendre en compte par la suite.



Toutes les variables étant complètes, on décide de les garder telles qu'elles sont pour nos modèles.

3 Modélisation

Nous entrainons plusieurs modèles qui prédisent la demande nette à partir d'autres covariables. Pour chaque modèle, nous utilisons une technique d'estimation différente et aussi un jeu de covariables différent, adaptés aux tests de significativité associés au modèle donnée. Notre objectif final est alors de combiner ces différentes approches de façon à pouvoir obtenir finalement les meilleures prédictions possibles. Pour pouvoir tester nos modèles, on les ajuste sur les 3471 premiers jours (`sel_a`) et on transforme les 395 dernier jours en données de validation (`sel_b`). Les graphiques tracés sont ceux des prédictions effectuées sur les données de validation. La fonction de perte qu'on essaie de minimiser est la pinball loss évaluée au quantile 0.8.

3.1 Generalized Additive Model (GAM) avec filtre de Kalman

Notre première approche consiste à ajuster un GAM (Generalized Additive Model) à nos données en utilisant un grand nombre de covariables. On inclut des variables météorologiques, calendaire et de consommation passées. Toutes les variables incluses sont statistiquement significatives [1].

```
equation <- Net_demand ~ s(toy,k=30, bs='cc') + s(Temp,k=10, bs='cr') +
s(Net_demand.1, bs='cr', by = as.factor(WeekDays))+ s(Net_demand.7, bs='cr') +
WeekDays + BH + s(Temp_s99_min, k = 10, bs = 'cr') + s(Temp_s99_max, k = 10, bs = 'cr') +
s(Wind) + Christmas_break + te(as.numeric(Date), Nebulosity, k=c(4,10)) + s(Time, k = 10, bs =
    'cr') +
s(Load.7, k = 5)
gamn <- gam(equation, data = Data0[sel_a])
```

Nous faisons ensuite de l'online learning à l'aide du filtre de Kalman avec espérance-maximisation. Le modèle s'ajuste itération par itération. Nous faisons bien attention à ce qu'on n'utilise pas les valeurs réelles au temps t pour prédire ce même état.

```
X <- predict(gamn, newdata=Data0, type='terms')
###scaling columns
for (j in 1:ncol(X)){
  X[,j] <- (X[,j]-mean(X[,j])) / sd(X[,j])}
X <- cbind(X,1)
d <- ncol(X)
y <- Data0$Net_demand
ssm <- viking::statespace(X, y)
ssm_em <- viking::select_Kalman_variances(ssm, X[sel_a], y[sel_a], method = 'em', n_iter = 100,
    Q_init = diag(d), verbose = 10, mode_diag = T)
ssm_em <- predict(ssm_em, X, y, type='model', compute_smooth = TRUE)
gamn.forecast <- ssm_em$pred_mean%>%tail(length(sel_b))
rmse(gamn.forecast, Data0$Net_demand[sel_b])

rmse(gamn.forecast, Data0$Net_demand[sel_b]) ## 1629
pinball_loss(Data0$Net_demand[sel_b], gamn.forecast) ## 537
```

Ce modèle nous donne les résultats les plus performants.

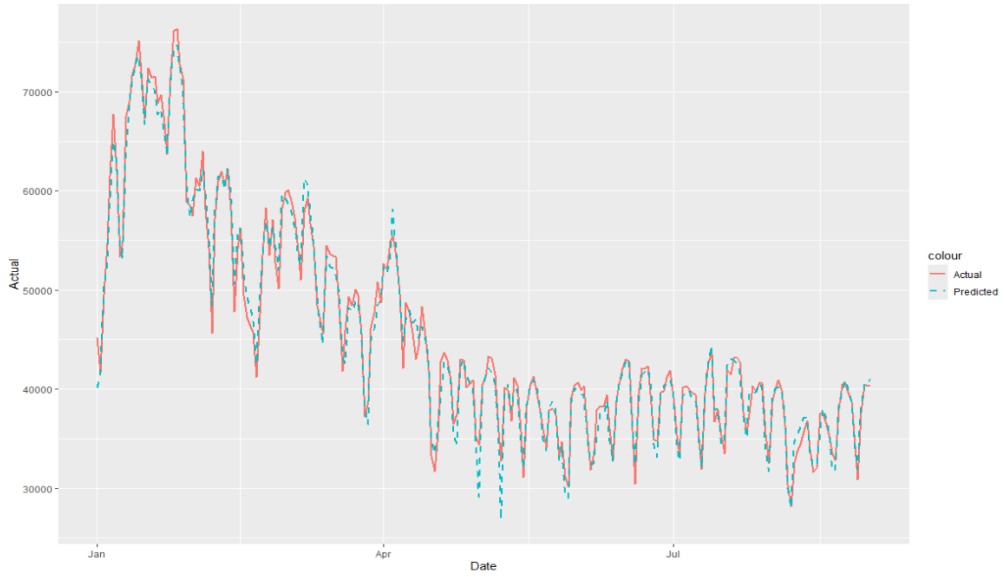
1. Pour tous les *summary* : `https://github.com/SamuelMolano/ModelisationPredictive/tree/main/Final_Project`

3.2 Modèle GAM Ridge

Un autre modèle GAM est construit en utilisant une régularisation Ridge, ce qui permet de réduire l'overfitting qui peut avoir lieu dans le cas où les splines utilisés sont trop lisses (grand nombre de noeuds). La régularisation est appliquée à la formule de base de la demande nette. Aussi le paramétre `select=TRUE` permet d'enlever les termes de lissages qui ne sont pas nécessaires. Cette double pénalisation permet d'avoir un modèle frugal.

```
equation <- Net_demand ~ s(as.numeric(Date),k=3, bs='cr') + s(toy,k=30, bs='cc') + s(Temp,k=10,
    bs='cr') + s(Load.1, bs='cr')+ s(Load.7, bs='cr')+
s(Temp_s99,k=10, bs='cr') + WeekDays +BH +
s(Wind) +
te(as.numeric(Date), Nebulosity, k=c(4,10)) +
s(Net_demand.1, bs='cr') + s(Net_demand.7, bs='cr')
gam_ridge <- gam(as.formula(equation), data=Data0, method="REML", select=TRUE)
saveRDS(gam_ridge, "Models/gam_ridge.RDS")
gam_ridge.forecast <- predict(gam_ridge, newdata = Data1)

rmse(gam_ridge.forecast, Data0$Net_demand[sel_b]) ## 1353
pinball_loss(Data0$Net_demand[sel_b], gam_ridge.forecast) ## 562
```



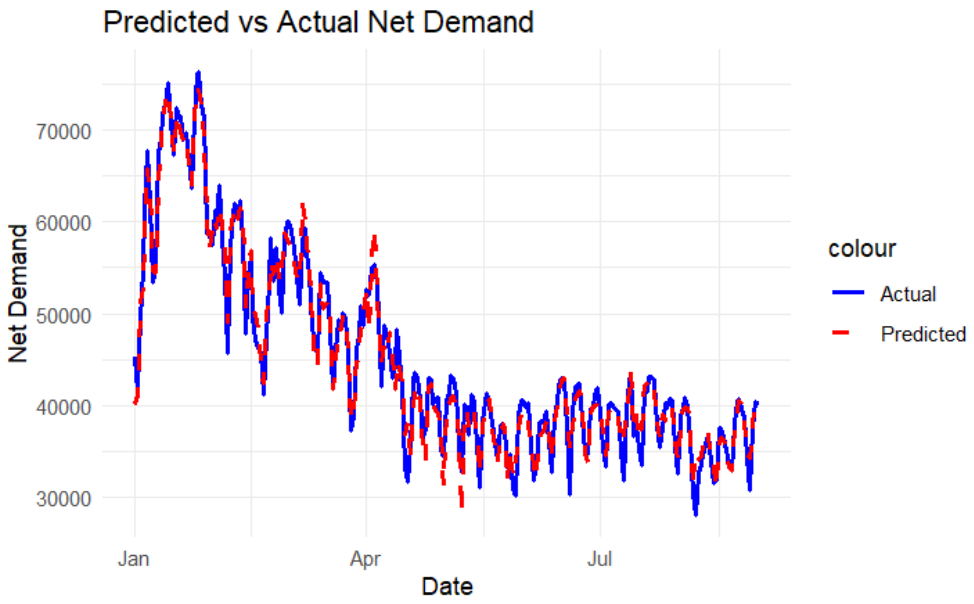
3.3 GAM et RF sur les erreurs

Nous ajustons cette fois-ci une GAM et ensuite une RandomForest aux erreurs. Nous faisons dans l'espoir de capturer certaines interactions ou non-linéarités qui auraient pu échapper aux autres modèles, tout en ayant conscience du risque de redondance, car les variables qu'on utilise sur le RandomForest était comprises dans la formule initiale.

```
### GAM et RF sur les erreurs

equation <- Net_demand ~ s(toy,k=30, bs='cc') + s(Temp,k=10, bs='cr') + s(Net_demand.1, bs='cr',
    by = as.factor(WeekDays))+ s(Net_demand.7, bs='cr') +
  WeekDays + BH + s(Temp_s99_min, k = 10, bs = 'cr') + s(Temp_s99_max, k = 10, bs = 'cr') +
  s(Wind) + Christmas_break + te(Solar_power.1, Wind_power.1, by = as.factor(WeekDays), k =
      c(3,3), bs = 'cr') +
  te(as.numeric(Date), Nebulosity, k=c(4,10))
gam <- gam(equation, data=Data0[sel_a1,])
res_gam <- (Data0$Net_demand[sel_a2,] - predict(gam, newdata = Data0[sel_a2,]))
res_tab <- Data0[sel_a2, c("Temp", "toy", "Time", "WeekDays")]
res_tab[,"Res"] <- res_gam
rf_err <- randomForest(Res ~ Temp + toy + WeekDays + Time, data=res_tab)
gam.predict <- predict(gam, newdata = Data0[sel_b,])
rf.predict <- predict(rf_err, newdata = Data0[sel_b,])
final_pred <- gam.predict + rf.predict

rmse(final_pred, Data0$Net_demand[sel_b]) ## 1754
pinball_loss(Data0$Net_demand[sel_b], final_pred) ## 716
```



3.4 Forêt aléatoire (Random Forest)

Un modèle de forêt aléatoire est également entraîné pour prédire la demande nette, en utilisant un ensemble de variables explicatives larges. La méthode Random Forest est adaptée pour capturer les relations complexes non linéaires dans les données, c'est une méthode qui a du mal a overfitter. On a entraîné plusieurs random forests avec différentes valeures de `mtry` pour optimizer ce paramètre. On choisit `mtry = 7` .

```
rf=randomForest(Net_demand ~Load.1 + Load.7 + Temp + Temp_s95 + Temp_s99 +
Temp_s95_min + Temp_s95_max + Temp_s99_min + Temp_s99_max +
Wind + Wind_weighted + Nebulosity + Nebulosity_weighted + toy +
WeekDays + BH_before + BH + BH_after + Year + Month + DLS +
Summer_break + Christmas_break + Holiday + Holiday_zone_a +
Holiday_zone_b + Holiday_zone_c + BH_Holiday +
Solar_power.1 + Solar_power.7 + Wind_power.1 + Wind_power.7 +
Net_demand.1 + Net_demand.7, data=Data0[sel_a.], mtry=7)
rf.forecast <- predict(rf, newdata = Data0[sel_b,])

rmse(rf.forecast, Data0$Net_demand[sel_b]) ## 2064
pinball_loss(Data0$Net_demand[sel_b], rf.forecast) ## 465
```

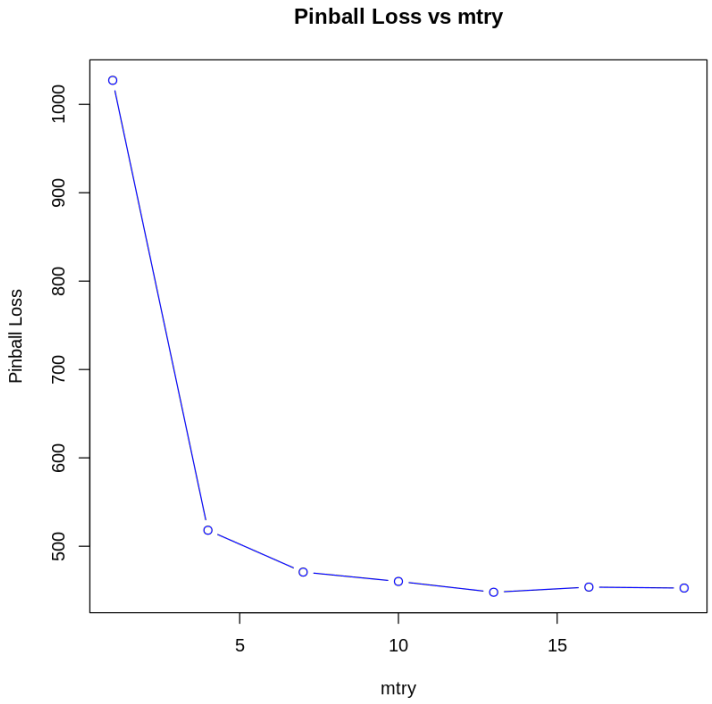


FIGURE 1 – Plot pinball loss selon la valeur de mtry

On voit les variables avec une plus grande importance : `Net_demand.1, Load.1` et les données de température.

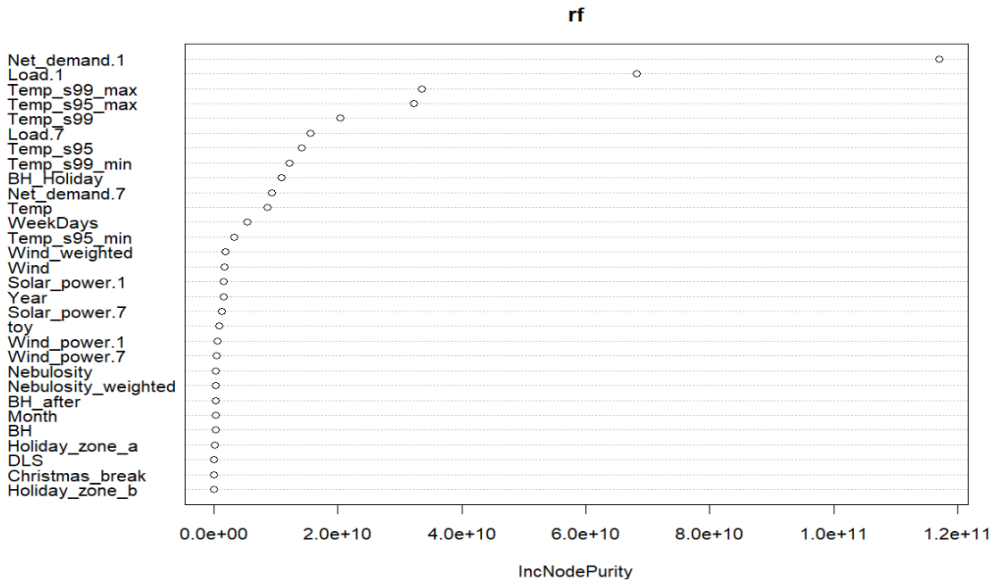
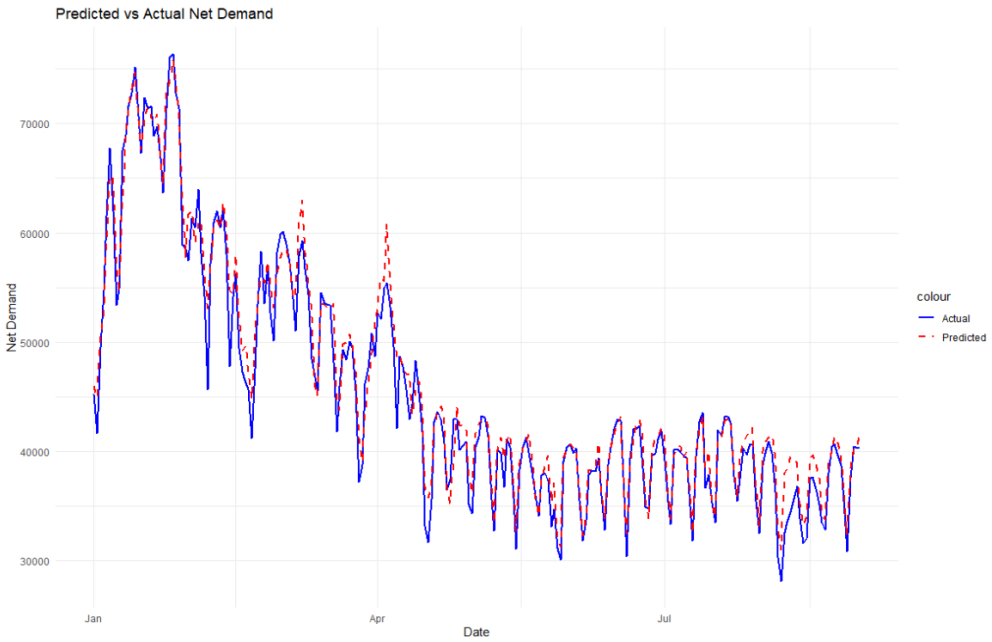


FIGURE 2 – Plot d'importance des variables



La prédiction faite par la forêt aléatoire est parmi les meilleures, nous avons cependant une tendance claire à la sur-estimation.

3.5 Modèle QGAM

Nous faisons des GAM quantiles pour borner nos prédictions par des valeurs extrêmes. On ajuste les modèles pour prédire les quantiles 0.2 et 0.8. Les variables incluses sont toutes statistiquement significatives.

```
equation <- Net_demand ~s(toy,k=30, bs='cc') + s(Temp,k=10, bs='cr') +
  s(Load.1, bs='cr', by= as.factor(WeekDays))+
  s(Load.7, bs='cr') + as.factor(WeekDays) +BH +
  s(Time, k = 10, bs = 'cr')
equation_var <- ~ s(Temp,k=10, bs='cr') +
s(Load.1) + as.factor(WeekDays) + s(toy, k = 30, bs = 'cc') +
s(Time, k = 10, bs='cr')
gqgam08 <- qgam(list(equation, equation_var), data=Data0[sel_a,], qu=0.8)
gqgam08.forecast <- predict(gqgam08, newdata = Data0[sel_b,], qu = 0.8)
gqgam02 <- qgam(list(equation, equation_var), data=Data0[sel_a], qu=0.2)
gqgam02.forecast <- predict(gqgam02, newdata = Data0[sel_b,], qu = 0.2)
saveRDS(gqgam09, "Models/qgam09.RDS")
```

4 Agrégation d'experts

Les différentes prévisions générées par les modèles étudiés seront comparées par une agrégation d'experts, où de façon similaire au filtre de Kalman, on doit faire attention à ne pas utiliser l'information inconnue au temps t pour prédire l'état à ce même moment. Ceci fait qu'on ait un problème pour la dernière valeur à prédire, qu'on prédira par convention en utilisant le GAM pénalisé.

On entraîne nos modèles, cette fois-ci sur la totalité des données et on fait l'agrégation d'experts en ligne, en utilisant l'algorithme MLpol.

On a

$$\hat{y}_t = \sum_{i=1}^{6} p_{i,t} \cdot \hat{y}_{i,t}$$

avec

$$\sum_{i=1}^{6} p_{i,t} = 1, \quad \forall t$$

Nous faisons de l'agrégation d'experts avec deux fonctions de perte : la pinball loss de quantile 0.8 et la RMSE. On s'attend à obtenir un meilleur score avec la pinball loss car il s'agit de la fonction de perte du challenge.

```
library(opera)

experts1 <- cbind(gqgam08.forecast[-395],gqgam02.forecast[-395], gam_ridge.forecast[-395],
    gamn.forecast, final_pred[-395], rf.forecast[-395])
experts <- experts1
colnames(experts) <- c("gam08","gam02","gam_ridge", "GAM_Kalman", "gam_rf_erreurs", "rf")
MLPol <- mixture(Y=Data1$Net_demand.1[-1], experts=experts, model = "MLpol")
MLPol_pinball <- mixture(Y=Data1$Net_demand.1[-1], experts=experts, loss.type =
    list(name='pinball', tau=0.8), model = "MLpol")
```

On obtient les poids suivants :

— $p_{qgam08} = 0.02$
— $p_{qgam02} = 0.02$
— $p_{GAMRidge} = 0.32$
— $p_{GAMKalman} = 0.31$
— $p_{GAMRFErr} = 0.31$
— $p_{RF} = 0.02$

5 Conclusion

Nous avons vu plusieurs méthodes pour faire des prévisions, des modèles additifs jusqu'aux forêts aléatoires, en passant par des modèles additifs quantiles. Nous avons même vu des combinaisons de plusieurs de ces méthodes. L'apprentissage en ligne, quand il est possible, produit des meilleurs résultats en adaptant les modèles aux nouvelles données.