Projet SY19 - Régression et classification

Tom Bourg, Guillaume Sabbagh, Jiali Xu

06/01/2021

1 Phoneme

1.1 Analyse exploratoire

Dans cette première partie, il s'agit d'une classification des phonemes. Le jeu de données contient 2250 observations avec 256 préditeurs et une seule variables de réponse. On observe que la classification est en 5 classes et tous les préditeurs sont quantitatives.

```
##  aa  ao dcl  iy  sh
## 343 555 357 560 435
```

On remarque ici que le nombre d'observation n'est pas balancé parmi les classes.

1.2 Analyse de Composantes Principales

On s'intéresse à une analyse de composantes principales afin de réduire le nombre de préditeurs sachant que 256 est assez nombreux. A l'aide de la fonction *prcomp()*, on trace la proportion de variance expliquée cumulée en fonction du nombre de composante dans la figure 1 dessous.

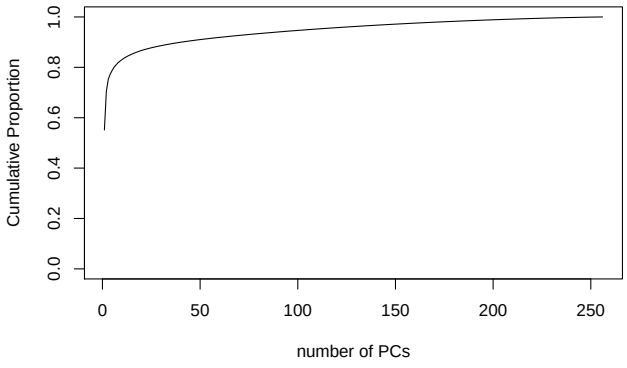

FIGURE 1 – Proportion de variance expliquée cumulée

On constate que les 107 premières composantes expliquent 95% de variance. Du fait que pour les réseaux neurones, le nombre de ports d'entrée de chaque couche est préférable d'être exponentiel de 2, on choisit ici 128 composantes pour les traitements suivants.

1.3 Apprentissage de modèles

1.3.1 LDA, QDA et Naive-Bayes

Le package **MASS** nous propose des fonctions pour LDA et QDA et le pakcage **naivebayes** pour un modèle Naive-Bayes. On observe que le taux d'erreur du modèle QDA est beaucoup plus haut que les deux autres. De ce

fait, on constate que la matrice de covariance pour chaque classe reste presque identique.

1.3.2 Regression Logistique

Afin de réaliser une regression logistique pour une classification multinomiale, on utilise le modèle *multinom()* dans le package **nnet**. A partir de cette méthode, on réalise également un modèle additive généralisé en ajoutant un *natural cubic spline*.

Ensuite, on teste la regression ridge et lasso. Le package **glmnet** nous permet de trouver *lambda* optimisé en faisant une validation croisée. Le modèle lasso nous donne un meilleur résultat.

1.3.3 Arbres de regression

Pour les méthodes basées sur arbre, on a essayé une arbre binaire de décision *rpart()* du package **rpart** et son élagage en choisissant une valeur **CP** qui minimise l'erreur de validation croisée. On a également réalisé un modèle **Bagging** et **RandomForest** en appliquant différentes valeurs de *mtry*.

1.3.4 SVM

La fonction *ksvm* du package **kernlab** est utilisée pour tester avec deux noyaux différents. Pour chaque noyau, on fait d'abord une validation croissée pour trouve une valeur **C** optimisée parmi un ensemble [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10e4].

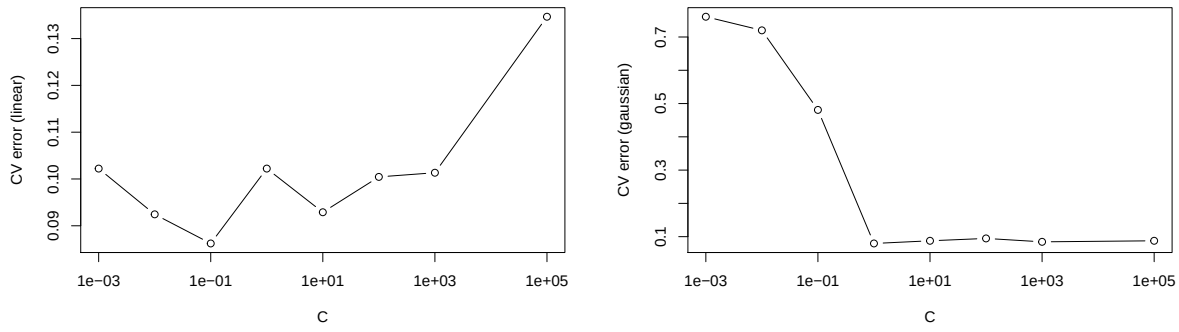

FIGURE 2 – Performance de modèles

1.3.5 Réseaux neurones

1.3.5.1 Réseau neurone artificiel On s'intéresse d'abord au réseau neurone artificiel avec deux couches. Sachant que nous sommes sur un problematique de classification, la fonction d'activation pour la couche de sortie est donc *softmax*. On paramètre *batch_size* à 128 et *epoch* à 30.

1.3.5.2 Réseau neurone profond régularisé Un réseau neurone profond est un réseau neurone artificiel avec plusieurs couches. Ici, on réalise un réseau neurone profond avec 4 *layer_dense* et 2 *layer_dropout*. On fixe le coefficient à 0.5 pour les couche de régulation afin d'éviter un surapprentissage. On applique une fonction d'activation *relu* pour tous les couches sauf celui de sortie. Les hyperparamètres à optimiser sont donc *units* des deux couches au milieu. Après des essais en utilisant une validation croisée, on n'observe pas une amélioration très évidente pour les différentes valeurs.

1.3.5.3 Réseau neurone convolutionnel Enfin, on effectue un essai sur un réseau neurone convolutionnel. Pour ce faire, il nous faut d'abord convertir les données en matrice 3D. Dans ce cas, la dimension de matrice de notre données est 2250*128 (après extraction de caractéristique). On choisit de la convertir en dimension 2250*16*8. La structure de modèle est comme ci-dessous.

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## conv1d (Conv1D)              (None, 14, 16)            400
## _________________________________________________________________________
## average_pooling1d (AveragePooling1D (None, 7, 16)              0
## _________________________________________________________________________
## conv1d_1 (Conv1D)            (None, 5, 32)             1568
## _________________________________________________________________________
## average_pooling1d_1 (AveragePooling (None, 2, 32)              0
## _________________________________________________________________________
## dropout (Dropout)            (None, 2, 32)             0
## _________________________________________________________________________
## flatten (Flatten)            (None, 64)                0
## _________________________________________________________________________
## dense (Dense)                (None, 128)               8320
## _________________________________________________________________________
## dropout_1 (Dropout)          (None, 128)               0
## _________________________________________________________________________
## dense_1 (Dense)              (None, 5)                 645
## =========================================================================
## Total params: 10,933
## Trainable params: 10,933
## Non-trainable params: 0
## _________________________________________________________________________
```

1.4 Comparaison de modèle

A la fin, on fait une comparaison de tous les modèles réalisés. Voir les performance sur figure 3 et 4

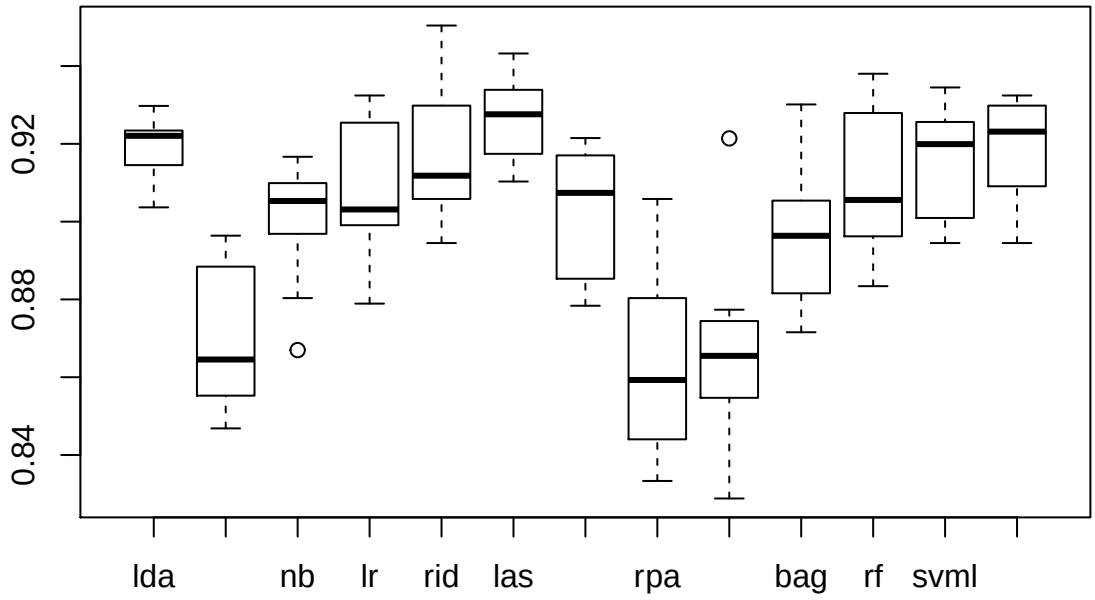

FIGURE 3 – Performance de modèles

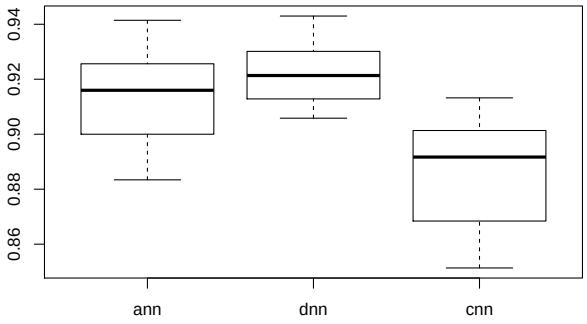

FIGURE 4 – Performance de réseau neurone

On constate que le modèle réseau neurone profond nous donne une précision la plus haute et une variance plus petite.

2 Bike rental dataset

2.1 Travail préliminaire

Le jeu de donnée contient 12 variables explicatives pour une variable à prédire. On cherche à prédire le nombre de vélo loué une journée en fonction de la saison, des conditions météorologiques, des jours ouvrés etc. Il s'agit d'une tâche de régression. Nous disposons pour cela de 365 observations ce qui est relativement peu. Pour un petit dataset comme celui-là, nous avons décidé d'éliminer d'office les réseaux neuronaux qui nécessitent de grands jeux de données pour converger. Nous essaieront donc les principaux algorithmes d'apprentissage statistique nécessitant peu de données pour fonctionner.

De plus, 7 des 12 prédicteurs sont des variables qualitatives ! Il nous faudra donc des méthodes capables de gérer des variables qualitatives en même temps que des variables quantitative. On pense directement aux arbres de décision qui sont plutôt versatiles. On pouvait penser que certaines variables serait corrélées (saison et mois par exemple) mais puisque la plupart des variables sont qualitatives, une ACP ne fonctionnerait pas. On va donc compter sur les modèles pour sélectionner les bonnes variables. Encore une fois les arbres de décision devraient bien s'en sortir.

2.2 KNN regression

En premier modèle, nous avons testé le modèle KNN, sans nous attendre à ce qu'il performe bien puisque ce dernier ne peut gérer les variables qualitatives. Le paramètre k a été trouvé par nested cross-validation. Comme prévu, le modèle ne performe pas particulièrement et notre meilleure erreur quadratique moyenne est de 483209.3 pour k=7.

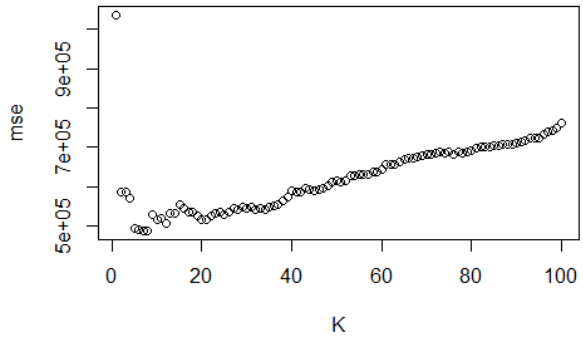

FIGURE 5 – Modèle KNN : Erreur quadratique moyenne en fonction de k, le minimum est atteint en k=7.

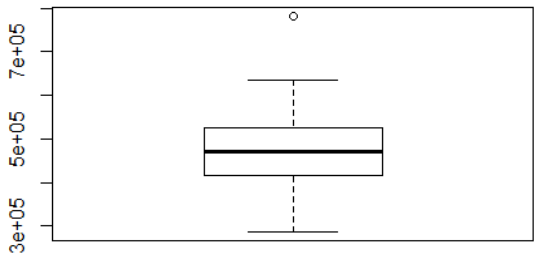

FIGURE 6 – Boxplot de l'erreur quadratique moyenne pour le modèle de régression KNN.

2.3 Elastic-net regression

Notre prochain modèle va permettre de faire de la sélection de variable. Le modèle Elastic-net, et ses cas particuliers ridge et lasso, permettent de faire une régression en sélectionnant les variables les plus utiles à la prédicitions. Cependant, ils ne se comportent pas bien avec les données qualitatives. Et malheureusement cela rend les modèles peu précis tout comme le modèle KNN (cf. les boxplots d'erreurs).

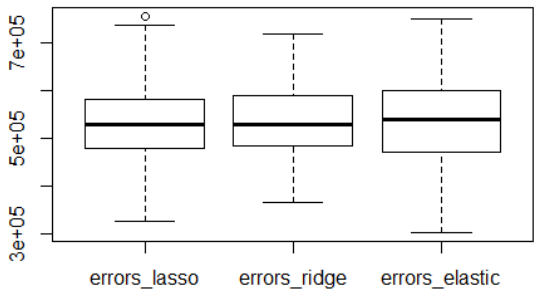

FIGURE 7 – Boxplot des erreurs quadratiques moyennes pour les méthodes lasson ridge ainsi que elastic net.

2.4 SVM

Un SVM pourrait correctement traiter nos données avec un kernel adéquat sachant que nos données ne sont pas linéaires.

L'astuce est de trouver le bon kernel, ce qui n'est pas simple.

Avec un kernel radial, on trouve par nested cross-validation que le paramètre gamma doit être 1/7. On obtient alors une erreur quadratique moyenne d'environ 250000.

Avec un kernel polynomial, on trouve par nested cross-validation les paramètres du kernel polynomial et on obtient finalement une erreur quadratique moyenne d'environ 800000.

Le kernel linéaire donne de très mauvais résultats comme prévu (mse supérieure à 500000).

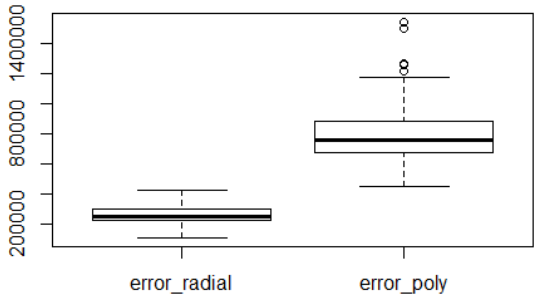

FIGURE 8 – Boxplots des erreurs quadratiques moyennes pour les SVM avec kernel radial et polynomial.

2.5 Decision tree et random forest

C'est la méthode qui semble la plus prometteuse depuis l'analyse préliminaire. En effet, elle est robuste aux données qualitatives et permet aussi de sélectionner les prédicteurs les plus discriminants.

L'arbre de décision seul a tendance à overfit, on préférera donc les random forest qui généralisent mieux.

Comme prévu, l'arbre de décision seul overfit et donne de mauvais résultats : l'erreur quadratique moyenne est d'environ 450000.

La forêt quant à elle n'overfit pas et donne de bons résultats, l'erreur quadratique moyenne est d'environ 250000. Elle égalise donc les performances du svm avec le kernel radial.

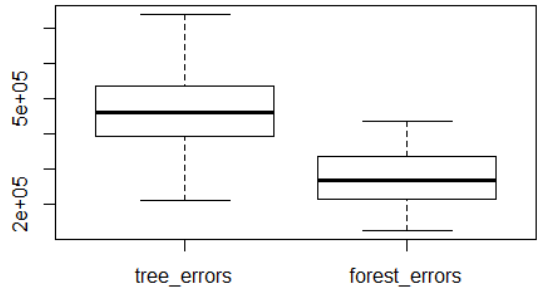

FIGURE 9 – Boxplots des erreurs quadratiques moyennes pour les arbres et forêts de régression.

2.6 Sélection de modèle

Voilà un récapitulatif général des mse obtenues :

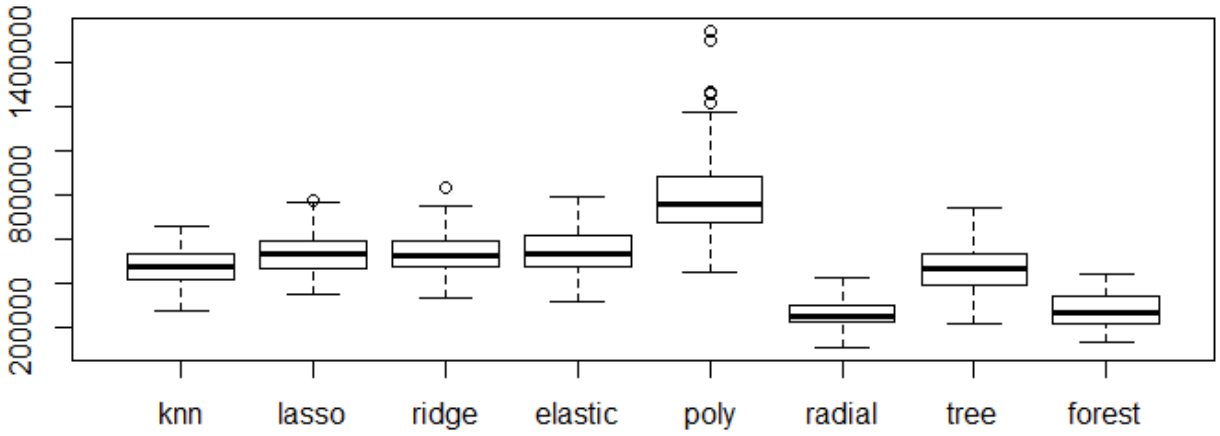

FIGURE 10 – Boxplots des erreurs quadratiques moyennes pour toutes nos méthodes.

Nous avons donc décidé de retenir la SVM avec kernel radial.