Rapport TP 10 - Apprentissage à partir de trois jeux de données réelles

Assila Yassine, Gabet Joseph, Sébastien Dam

A21

Letter recognition

L'objectif est d'identifier à partir d'un grand nombre de pixels rectangulaires en noir et blanc l'une des 26 lettres majuscules de l'alphabet anglais.

Analyse exploratoire

Letter recognition est aussi un problème de classification puisqu'il s'agit de prédire une lettre parmi les lettres de l'alphabet. Les fréquences des lettres sont assez bien réparties.

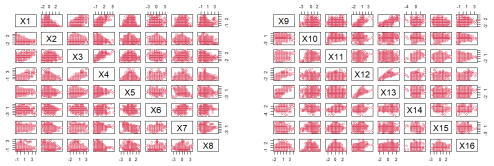
Analyse des données

En faisant une analyse rapide du jeu de données, nous obtenons les caractéristiques :

- Pas de dimension temporelle *connue*
- Nombre d'observations : 10 000
- Nombre de variables : 16 prédicteurs + 1 variable à prédire
- Variables connues : X1 à X16
- Variable à prédire : y
- Aucune valeur manquante
- Variables continues : X1 à X16
- Variable discrète : y $\in$ [1, 26]

Analyse des prédicteurs

En premier lieu, on regarde s'il existe de possibles correlations entre les prédicteurs.



A priori, il ne semble pas y avoir de liens évidents qui nous permettraient d'éliminer certaines variables, hormis entre X3 et X4, et X12 et X13 qui semblent suivre une tendance linéaire.

Sélection de variables

Au vu du faible nombre de prédicteurs, il ne semble pas nécessaire d'appliquer des méthodes de sélection de variables. Cependant il est toujours intéressant de diminuer la dimensionalité si cela est possible et permet d'améliorer les résultats de notre classification.

L'ACP semble être une mauvaise idée. En effet pour obtenir une explication de la variance supérieure proche de 100% il faut utiliser l'ensemble des prédicteurs. Il n'y a pas un "gap" qui permette, avec un sous-ensemble de composantes, d'expliquer une grande proportion de la variance.

Une autre idée est de sélectionner les variables qui ont la plus grande importance dans en utilisant l'importance des variables avec la méthode des forêts aléatoires.

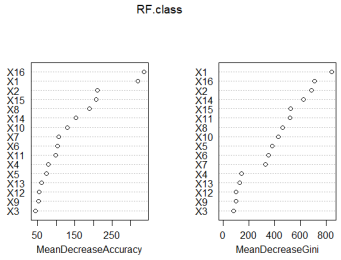


Figure 1: Importances des variables, forêt aléaroire

Cette idée est notamment renforcée par le fait que les modèles basés sur la forêt aléatoire se trouvent être les plus performants (entre 2% et 3% de taux d'erreur).

Enfin nous testons les modèle de sélection de variables ridge et lasso. Ceux-ci ne présentent pas de très bonnes performances (relativement aux autres méthodes testées). Cela conforte l'idée qu'une sélection de variable ne semble pas être très pertinente ici.

Entraînement des modèles

Nous appliquons les algorithmes de classifications suivants : Naive Bayes (NB), K plus proches voisins (KNN), Analyse Discriminante Linéaire (LDA), Analyse Discriminante Quadratique (QDA), Analyse Discriminante Factorielle (FDA), Régression Logistique (RL), Ridge, Lasso, arbre de classfication (TREE), arbre de classification élagué (pTREE), Bagging, Forêt Aléatoire (RF), Forêt Aléatoire avec recherche du paramètre optimal mtry, Mélange Aléatoire Gaussien (GMM), Mélange Aléatoire Gaussien avec Analyse Discriminante (GMM_EDDA), Séparateur à Vaste Marge (KSVM) avec plusieurs noyaux et les réseaux de neurones (nn).

Le résaux de neurones à été genéré grâce au package "nnet" de la façon suivante :

```
nnet(as.factor(Y) ~ ., data=train_data, size=5, linout = TRUE)
```

Les mauvais résultats (relatif) ne nous pousserons pas à approfondire ce modèle.

Les arbres ont été crées via la fonction "rpart" avec l'indice de gini. Pour ce qui est de l'arbre élagué nous avons utilisé la fonction "prune".

```
fit <- rpart(Y ~ ., data = train_data, method="class", parms = list(split = 'gini'))
pruned_tree<-prune(fit,cp=fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])
```

Ici Bagging prend comme paramètre `mtry = 16` (le nombre de prédicteurs), forêt aléatoire prend comme paramètre `mtry = 4` (racine carrée du nombre de prédicteurs). Enfin, grâce à la fonction `tuneRF` de la librarie `randomForest`, on recherche le paramètre mtry‘ optimal.

```
tuneRF(data_class.train[,2:17], data_class.train$Y, stepFactor=1.5, improve=1e-5, ntree=500)
```

Dans un premier temps, ces algorithmes sont testés avec l'ensemble des prédicteurs. Cette première étude nous permet de voir si des modèles se distinguent déjà fortement des autres, ce qui nous permettra de réaliser une étude approfondie.

Pour chaque modèle, nous réalisons une validation croisée à 10 plis. Pour certains classifieurs nécessitant de spécifier un hyperparamètre, nous utilisons une validation croisée imbriquée pour obtenir l'hyperparamètre optimal.

Table 1: Valeur des hyperparamètres optimaux pour chaque modèle

Modèle	KNN	Ridge	Lasso	RF mtry
Hyperparamètre	K = 1	$\bar{\lambda}$ = 0.008674308	$\bar{\lambda}$ = 8.615844e-05	mtry = 2.4 (en moyenne)

KSVM

Au cours de nos tests, nous essayons d'appliquer un SVM avec le noyau "Radial Basis Function". Pour ce faire, on calcule l'erreur obtenue pour différentes valeurs de C : 0.001, 0.01, 1, 10, 100, 1 000, 100 000. On s'aperçoit alors de l'efficacité du modèle qui a une erreur minimale de 4,7% pour $C = 10$. Ce résultat très encourageant au vu des performances des autres modèles (seuls les modèles de forêt aléatoire font mieux), nous pousse à approfondire nos recherches sur ce type de solution. Ainsi pour chaque type de noyaux, nous essayons de déterminer le taux d'erreur en fonction de l'hyperparamètre C.

Table 2: Valeur des hyperparamètres optimaux pour chaque noyau du SVM

Noyaux	RBF	Laplace	Bessel	polynome	tanh
taux d'erreur miniaml	0.423	0.0445	0.269	0.1484	> 0.8
C correspondant	100	10	0.1	1	0.01

On constate que les deux modèles qui se distinguent des autres par leur bonne performance sont ceux qui utilisent le noyau "Laplace" ainsi que le noyau "Radial Basis Function". Il est normal que ces deux modèles aient des résultats très proches car les fonctions de noyaux sont similaires. Ainsi KSVM donne de très bonnes performances de 4.42% de taux d'erreur avec le noyau "Laplace".

Enfin puisque le SVM se distingue des autres pour ces deux noyaux, nous approfondissons nos recherches en essayant d'appliquer une sélection de variables en retirant celles qui sont moins importantes au sens de la forêt aléatoire. Cependant on constate que chaque prédicteur enlevé augmente le taux d'erreur.

Résultats

Après avoir testé tous les modèles, nous obtenons la figure 5 résumant le taux d'erreur selon chaque modèle. Nous constatons que les modèles sont globalement performants avec des écart-types inégaux mais relativement faibles. En effet étant donné qu'il y a 26 classes, le taux d'erreur d'une classification aléatoire est de 25/26 et ici, le pire modèle à un taux d'erreur inferieur à 50%.

Les pires modèles sont l'arbre et l'arbre élagué (qui en fait ici est toujours le même arbre). Ensuite le modèle avec le moins bon taux d'erreur est le réseau de neurones. Cela n'est pas surprenant car un réseau de neurones performe une transformation de l'espace des prédicteurs. Or ici, les prédicteurs sont relativement peu nombreux et représente déjà une tranformation des informations initiales (les images de lettres).

A contrario, les modèles basés sur les SVM, précédés par les modèles de forêt aléatoire semblent particulièrement performants (moins de 5% de taux d'erreur). De plus ces modèles sont ceux qui présentent les variances les plus faibles. Enfin, au vu de la grande différence des taux d'erreur observés suivant ces modèles, ce critère semble suffisamment pertinent pour distinguer les modèles.

Table 3: taux d'erreurs moyen (validation croisée 10 plis)

Meilleurs modèles	RF mtry	RF	Bagging	SVM_rbf	SVM_laplace
taux d'erreur	0.01810013	0.02550019	0.02599946	0.04550196	0.04550196

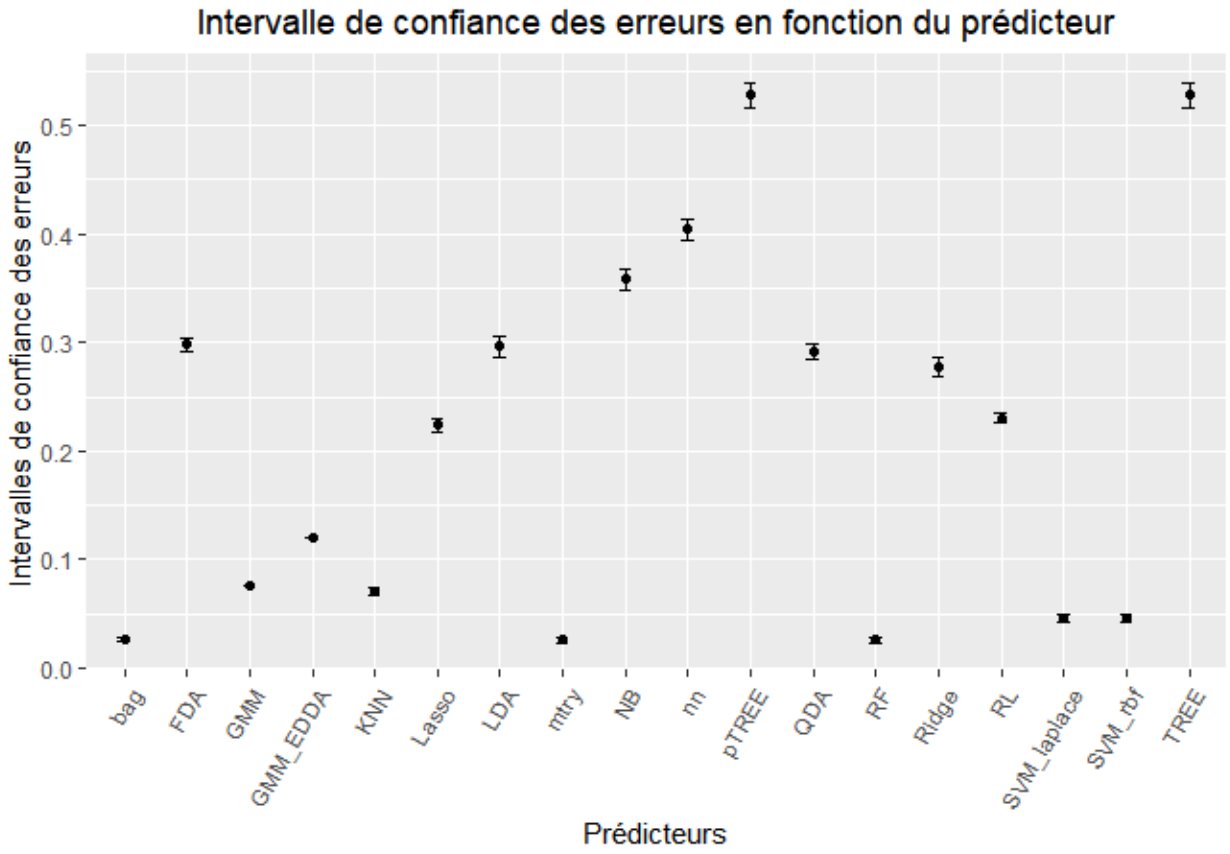


Figure 2: Résultats des modèles pour le dataset `letter`

Ainsi, en conclusion, nous choisissons le modèle de la **forêt aléatoire avec mtry = 2**.