TP7 Khalis - Mekkoudi - Collas

Maïs

Chargement des données et des fonctions

```
source("functions.R")
data <- read.table(file = "../TP7-SY19/data/mais_train.csv", sep=',', header = T)
data$X <- NULL
RECHERCHE_HYPERPARAMETRE <- FALSE
```

Chargement des librairies

```
library(tidyr)
library(dplyr)
library(ggplot2)
library(randomForest)
library(keras)
library(kernlab)
```

Séparation des données en entrainement/test

Comme précédemment, nous séparons notre datset en deux ensembles: un ensemble d'entrainement et un ensemble de test. Nous centrons réduisons également nos données car elles ont des ordres de grandeur trés différents.

Méthode d'évaluation

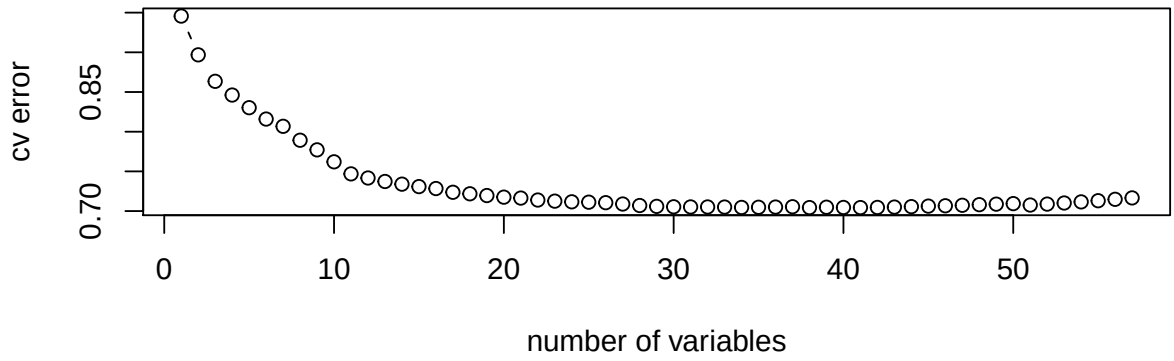
Nous avons choisi l'erreur quadratique moyenne comme mesure de l'erreur, elle sera estimée par validation croisée et par le "out-of-bag error" pour la forêt aléatoire.

Régression linéaire

Nous commençons par une régression linéaire, une méthode qui de par sa simplicité permet une meilleure interprétation.

Nous effectuons une "forward stepwise selection" et pour chaque modèle nous évaluons l'erreur grâce é une k-fold cross-validation. Le meilleur modèle correspond à celui comprenant les 38 premières variables sélectionnées

```
res <- ForwardCrossValidationLM(mais_train)
mais.mod <- lm(as.formula(paste("yield_anomaly ~", paste(res$vars[1:which.min(res$CV)],collapse="+"))),d
plot(res$CV, xlab="number of variables", ylab="cv error", type="b")
```



La régression linéaire ainsi que d'autres modèles linéaires tel que la régression Lasso n'arrivent pas à produire de meilleurs résultats. Nous avons opéré des transformations afin d'obtenir un modèle non linéaire en: - multipliant les variables par rapport à elle même - multipliant les variables par rapports aux autres du même mois - transformant la variable qui correspond au département en plusieurs variables binaires afin d'éliminer la relation d'ordre entre les départements (à priori, il n'y a pas de relation linéaire entre le numéro de département et le rendement de maïs) L'erreur quadratique moyenne ne s'est que faiblement amélioré, nous avons par conséquent écarté ses pistes.

Forêts aléatoires

Optimisation du nombre d'arbre

Comme pour le dataset astronomy, nous optimisons dans un premier temps le nombre d'arbre à utiliser en gardant le paramètre mtry fixe (ici p/3). L'erreur est calculé grâce au paramètre "out of bag error" qui est retourné par la fonction "randomForest".

```
scaled_test <- data.frame(t(apply(mais_test[,-which(names(mais_test)=="yield_anomaly")], 1, function(r)

nbtree <- optimizeNtreeAndPlotResults()
```

```
## Nombre d'arbres optimal:  956
```

Optimisation de 'mtry'

```
res <- optimizeMtryAndPlotResults()
```

```
## Paramètre mtry optimal:  34
```

```
test.err <- res$test.err
oob.err <- res$oob.err
mtry <- res$mtry
```

SVM à noyau

Noyau linéaire

Comme dit dans l'exercice précédent, pour des raisons de temps calculs nous choisirons d'optimiser le paramètre C uniquement avec le noyau linéaire.

Nous faisons varier ce paramètre et le calcul de l'erreur se fait par validation croisée

```
if(RECHERCHE_HYPERPARAMETRE){
  C <- (1:5)^5
  CV1 <- numeric(length(C)) # [1] 0.7318539 0.7338461 0.7423543 0.8082920 1.0046964

  for (c in 1:length(C)) {
    svmfit <- ksvm(yield_anomaly ~ .,data = mais_train, scaled=TRUE, type="eps-svr",
    kernel="vanilladot", C=C[c], cross=10)
    CV1[c] <- cross(svmfit)
  }
  c <- C[which.min(CV1)] #1
} else {
  c <- 1
}
```

Le paramètre C optimal trouvé vaut 1 et l'erreur est d'environ 0.73.

Noyau gaussion

Nous faisons de même ici avec le paramètre sigma

```
if(RECHERCHE_HYPERPARAMETRE) {
  Sigma <- seq(0.0001, 0.0401, by=0.005)
  CV <- numeric(length(Sigma)) # [1] 0.8649978 0.6319195 0.5883565 0.5764376 0.5767431 0.5762012 0.5819

  for(s in 1:length(Sigma)) {
    svmfit <- ksvm(yield_anomaly ~ .,data = mais_train, scaled=TRUE, type="eps-svr",
    kernel="rbfdot", kpar=list(sigma=Sigma[s]), C=c, cross=10)
    CV[s] <- cross(svmfit)
  }
  s <- Sigma[which.min(CV)]
} else {
  s <- 0.0351
}
```

La valeur optimal trouvée 0.0351 et l'erreur estimée vaut 0.57

Noyau polynomial

```
if(RECHERCHE_HYPERPARAMETRE) {
  degree <- 2:5
  CV <- numeric(length(degree)) # [1] 3.226894 1.162719 1.394694 2.193799
  for(d in 1:length(degree)) {
    svmfit <- ksvm(yield_anomaly ~ .,data = mais_train, scaled=TRUE, type="eps-svr",
    kernel="polydot", kpar=list(degree=degree[d]), C=c, cross=10)
```

```
    CV[d] <- cross(svmfit)
  }
  d <- degree[which.min(CV)] # 3
} else {
  d <- 3
}
```

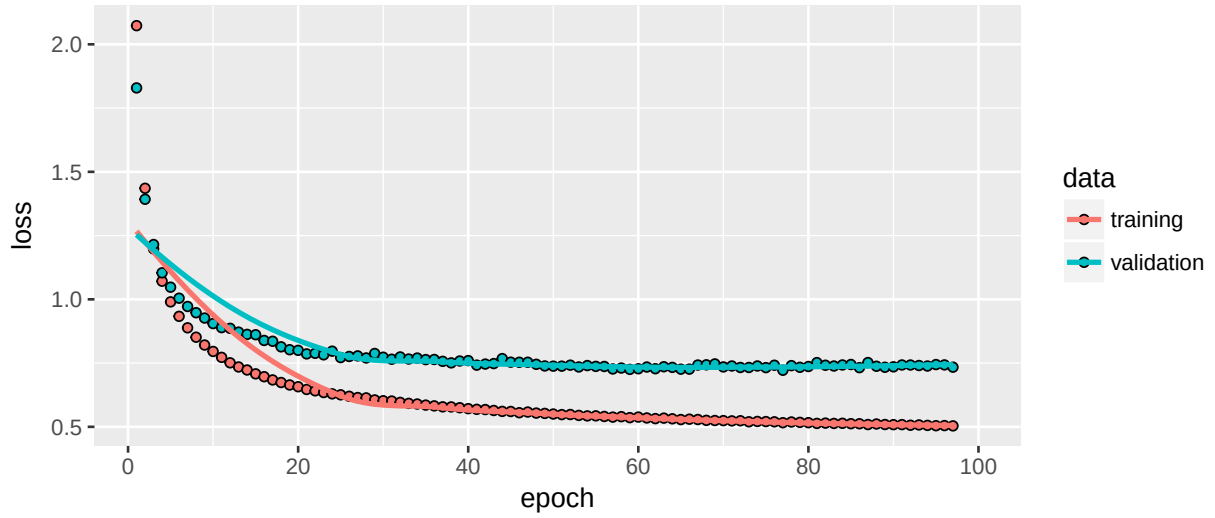
Le noyau polynomial n'offre quant à lui, pas un bon niveau de performance.

Réseau de neurone : construction d'un perceptron

Nous construisons un perceptron avec une couche cachée et 5 neurones dans cette dernière. Avec un nombre de neurones supérieur, notre réseau "surapprend" nos données d'entrainement. Il nous faudrait plus de données pour palier à ce problème étant donné le nombre élevé de variables.

```
history <- build_perceptron()
```

```
plot(history)
```



L'erreur se trouve aux alentours 0.75. Afin de réduire la dimension de notre dataset, nous avons tenté d'utiliser seulement les variables les plus importantes de la forêt aléatoire, nous avons obtenu un taux d'erreur similaire en utilisant plus de neurones.

Modèle retenu

Le modèle retenu est le SVM avec un noyau gaussien, il semble posséder la plus faible erreur même si la forêt aléatoire possède également de bonnes performances. Nous évaluons également l'erreur sur l'ensemble de test.

```
mais.mod <- ksvm(yield_anomaly ~ .,data = mais_train, scaled=TRUE, type="eps-svr", kernel="rbfdot", kpa
save(oob.err,test.err,centers, scales, mais.mod, file="env_mais.RData")
```

```
source(file="predicteurs.R")
pred <- classifieur_ble(mais_test[,-which(names(mais_test)=="yield_anomaly")])
mse <- mean((pred - mais_test$yield_anomaly)^2)
```

Nous obtenons une erreur d'environ 0.52 sur notre ensemble de test.