Apprentissage de données massives avec H2O

Résumé

*Le succès de la science des données engendre un forte concurrence pour la production de librairies regroupant les algorithmes d'apprentissage applicables à des données massives distribuées sur un système Spark-*Hadoop *et exécutable à partir de R. Après RHadoop, voici une introduction à une autre librairie R (H2O) développée par la société Oxdata [1]. Concepts et types de données, classification supervisée et non supervisée, introduction à l'apprentissage profond ou* deep learning.

Organisation des tutoriels R.

- *Démarrer rapidement avec R*
- *Initiation à R*
- *Fonctions graphiques de R*
- *Programmation en R*
- *MapReduce pour le statisticien*
- *Apprentissage et données massives avec H2O*

1 introduction

1.1 Objectif

Introduction à l'utilisation de H2O à partir de R pour exécuter des algorithmes d'apprentissage dans différents environnements : simple poste avec plusieurs processeurs, serveur, cluster, cluster avec données distribuées (Spark, Hadoop, Amazon EC2). C'est doute la solution la plus simple à déployer pour l'utilisateur de R car il "suffit" d'installer la librairie éponyme. Bien entendu l'utilisation sur un cluster nécessite des compétences spécifiques pour le déploiement (cf. la documentation alors que H2O apporte des possibilités quasi implicites de parallélisation sur un seul poste à partir de R.

1. T. Hastie et R. Tibshirani, membres du conseil scientifique consultatif de cette société, ont sans doute une influence non négligeable sur la stratégie déployée.

1.2 Installation

Moyennant la dépendance vis-à-vis de certaines librairies (`RCurl`, `bitops`, `rjson`, `statmod`, `tools`), l'installation de la librairie R H2O à partir du CRAN ne semble pas poser de problème.

Une documentation "détaillée" est disponible ici dont est extrait le tuteuriel suivant qui apporte en plus un léger regard critique.

1.3 Fonctionnalités

Quelques uns des algorithmes et méthodes les plus classiques sont proposées : ACP, k-means, modèle linéaire général avec ou sans pénalisation (ridge, lasso), bayésien naïf, forêts aléatoires et boosting (GBM) ainsi qu'une version sans doute "édulcorée" de deep learning ; optimisation des paramètres de complexité des modèles sur une grille de valeurs. H2O est plus élaborée que `scikit-learn` de Pyhton ou MLlib de Spark, pour les aides à l'interprétation statistique, le calcul de scores (AIC, AUC, courbes ROC...). En revanche, d'autres algorithmes (NMF, modèles de mélange, arbre binaire de décision) présents dans Spark ne sont pas ou pas encore proposés dans H2O.

1.4 Type de données

Une table de données H2O, éventuellement massive et distribuée, est vue de R comme un objet "S4" de classe `H2OParsedData`. Cet objet S4 a une clée (`@key slot`) qui est la référence de l'objet "données massives" géré par le serveur H2O. La librairie H20 prend la main sur les commandes usuelles (`summary`, `+`...) pour les exécuter sur cette classe `H2OParsedData` d'objets et retourne une référence, une clef, stockée à son tour comme un objet S4 de R.

Un tel objet `H2OParsedData` (commande `as.dataframe`) peut être chargé en mémoire "dans" R comme un `data frame` classique à condition, évidemment, de ne pas déborder des capacités de la RAM de la machine utilisée.

2 Gestion des données

2.1 Les données

Les données utilisées dans les tuteuriels de la documentation sont entre autres celles du *Public Use Microdata Sample* (PUMS) de 2013. Elles sont considérées comme une mise à jour de celles plus connue de l'UCI Adult Data Set. Elles décrivent avec plus de 300 variables socio-économiques un échantillon de 313240 individus et il ne s'agit que de 1% de la table initiale.

2.2 Importation

Il n'est pas nécessaire de les décompresser avant de les charger dans H2O. Télécharger le fichier `adult_2013_full.csv` dans le répertoire courant de R ou RStudio.

```
# Charger la librairie préalablement installée
library(h2o)
# Lancer le serveur H2O ici sur un seul poste avec
# tous les processeurs sauf 1
h2oServer = h2o.init(nthreads = -1)
# Chemin absolu ou relatif d'accès      au fichier
datadir = "chemin" # à compléter !
# nom du fichier
csvfile = "adult_2013_full.csv.gz"
# Importation avec pour clef H2O le nom du fichier
adult_2013_full = h2o.importFile(h2oServer,
    path = file.path(datadir, csvfile),
    key = "adult_2013_full", sep = ",")
# Clef et correspondance
kvstore = h2o.ls(h2oServer)
kvstore
# Taille du fichier
kvstore$Bytesize[kvstore$Key==
   "adult_2013_full"]/1024^2
# Classe de l'objet
class(adult_2013_full)
# Dimensions
dim(adult_2013_full)
# 50 premières colonnes
head(colnames(adult_2013_full), 50)
# Extraction d'un sous-ensemble des variables
# individus sans valeur manquante et revenu positif
nms = c("AGEP", "COW", "SCHL", "MAR", "INDP","RELP",
    "RAC1P", "SEX","INTP", "WKHP", "POBP", "WAGP")
adult_2013 = adult_2013_full[!is.na
    (adult_2013_full$WAGP)&adult_2013_full$WAGP > 0,
    nms]
h2o.ls(h2oServer)
```

Les variables sélectionnées sont : age (AGEP), class of worker (COW), educational attainment (SCHL), marital status (MAR), industry employed (INDP), relationship (RELP), race (RAC1P), sex (SEX), interest/dividends/net rental income over the past 12 months (INTP), usual hours worked per week over the past 12 months (WKHP), place of birth (POBP) and wages/salary income over the past 12 months (WAGP).

Attention, `adult_2013` est un objet R qui nécessite une assignation pour en faire une clef valide. D'autre part la fonction ci-dessous supprime des clefs inutiles.

```
# Assignation de la nouvelle clef
adult_2013 = h2o.assign(adult_2013,
   key = "adult_2013")
# Fnnction pour effacer les clefs systèmes
rmLastValues = function(pattern = "Last.value.")
{
  keys = h2o.ls(h2oServer, pattern = pattern)$Key
  if (!is.null(keys))
     h2o.rm(h2oServer, keys)
  invisible(keys)
}
rmLastValues()
# Vérification des clefs
kvstore = h2o.ls(h2oServer)
kvstore
kvstore$Bytesize[kvstore$Key ==
   "adult_2013"] / 1024^2
```

2.3 Manipulations

Gestion élémentaire des données avec la construction de deux variables de pere ou gain de capital pour retrouver les variables du fichier UCI.

```
# Vérification des données
summary(adult_2013)
dim(adult_2013)
# Centiles des revenus
centiles = quantile(adult_2013$WAGP,
   probs = seq(0, 1, by = 0.01))
centiles
# perte ou gain en capital
capgain = ifelse(adult_2013$INTP > 0,
   adult_2013$INTP, 0)
caploss = ifelse(adult_2013$INTP < 0,
   - adult_2013$INTP, 0)
adult_2013$CAPGAIN = capgain
adult_2013$CAPLOSS = caploss
adult_2013 = adult_2013[,- match("INTP",
   colnames(adult_2013))]
```

NB : tous les résultats intermédiaires dont les vecteurs `capgain` et `caplost` sont stockés sur le serveur. Même si à ce niveau de l'étude, on peut considérer que les données nettoyées et prétraitées tiennent maintenant en mémoire.

Renettoyage des clefs.

```
adult_2013 = h2o.assign(adult_2013,
   key = "adult_2013")
h2o.ls(h2oServer)
rmLastValues()
h2o.ls(h2oServer)
```

Il est préférable de prendre le log des revenus et capitaux puis à nouveau nettoyer.

```
adult_2013$LOG_CAPGAIN=log(adult_2013$CAPGAIN+1L)
adult_2013$LOG_CAPLOSS=log(adult_2013$CAPLOSS+1L)
adult_2013$LOG_WAGP = log(adult_2013$WAGP + 1L)
```

```
# nettoyer
h2o.ls(h2oServer)
rmLastValues()
h2o.ls(h2oServer
```

D'autres typres de transformations et analyses sont facilement exécutables sans utiliser la RAM.

```
# Discrétisation de la variable revenus
cutpoints = centiles
cutpoints[1L] = 0
adult_2013$CENT_WAGP = h2o.cut(adult_2013$WAGP,
   cutpoints)
adult_2013$TOP2_WAGP=adult_2013$WAGP>centiles[99L]
# gestion de tables
centcounts = h2o.table(adult_2013["CENT_WAGP"],
   return.in.R = TRUE)
round(100 * centcounts/sum(centcounts), 2)
top2counts = h2o.table(adult_2013["TOP2_WAGP"],
   return.in.R = TRUE)
round(100 * top2counts/sum(top2counts), 2)
relpxtabs = h2o.table(adult_2013[c("RELP",
   "TOP2_WAGP")], return.in.R = TRUE)
relpxtabs
round(100 * relpxtabs/rowSums(relpxtabs), 2)
schlxtabs = h2o.table(adult_2013[c("SCHL",
   "TOP2_WAGP")], return.in.R = TRUE)
schlxtabs
round(100 * schlxtabs/rowSums(schlxtabs), 2)
# nettoyage
h2o.ls(h2oServer)
rmLastValues()
h2o.ls(h2oServer)
```

Remplacer comme dans R les codes entiers par des niveaux de facteurs.

```
for (j in c("COW", "SCHL", "MAR", "INDP", "RELP",
   "RAC1P", "SEX", "POBP"))
```

```
adult_2013[[j]] = as.factor(adult_2013[[j]])
h2o.ls(h2oServer)
rmLastValues()
h2o.ls(h2oServer)
```

Création d'une variable "interaction" entre deux facteurs.

```
inter_2013 = h2o.interaction(adult_2013,
   factors = c("RELP", "SCHL"),
pairwise = TRUE, max_factors = 10000,
min_occurrence = 10)
adult_2013 = cbind(adult_2013, inter_2013)
adult_2013 = h2o.assign(adult_2013,
   key = "adult_2013")
colnames(adult_2013)
```

La préparation s'achève avec la création des échantillons d'apprentissage (75%) et de test (25%).

```
# tirage des échantillons
rand = h2o.runif(adult_2013, seed = 1185)
adult_2013_train = adult_2013[rand <= 0.75, ]
adult_2013_train = h2o.assign(adult_2013_train,
   key = "adult_2013_train")
adult_2013_test = adult_2013[rand > 0.75, ]
adult_2013_test = h2o.assign(adult_2013_test,
   key = "adult_2013_test")
# Vérification des effectifs
nrow(adult_2013)
nrow(adult_2013_train)
nrow(adult_2013_test)
# Vecteur colonne de la cible de l'échantillon test
actual_log_wagp = h2o.assign(
   adult_2013_test[, "LOG_WAGP"],
   key = "actual_log_wagp")
rmLastValues()
```

3 Exploration et Modélisation

3.1 Régression

Il s'agit de trouver le meilleur modèle de prévision de la variable `LOG_WAGP`.

Modèle linaire général

Cette partie est bien détaillée dans la librairie et la documentation avec les modèles : gaussien, binomial, Poisson... ainsi que les pénalités *Elastic net* avec optimisation des paramètres sur une grille.

Le modèle de la variable `LOG_WAGP` est un modèle gaussien de régression, sélection et optimisation sont traitées par *Elastic Net*. Il n'y a semble-t-il pas de procédure automatique de sélection de modèle par mis toutes les interactions d'ordre 2 mais la comparaison des modèles ci-dessous incite à considérer l'interaction entre `RELP` et `SCHL`.

Estimation de trois modèles :

```
# Une seul vartiable explicative RELP
log_wagp_glm_relp = h2o.glm(x = "RELP",
   y = "LOG_WAGP",
   data = adult_2013_train,
   key = "log_wagp_glm_relp",
   family = "gaussian",
   lambda = 0)
# Autre variable: SCHL
log_wagp_glm_schl = h2o.glm(x = "SCHL",
    y = "LOG_WAGP",
    data = adult_2013_train,
    key = "log_wagp_glm_schl",
    family = "gaussian",
    lambda = 0)
# Interaction entre les deux : RELP_SCHL
log_wagp_glm_relp_schl = h2o.glm(x = "RELP_SCHL",
   y = "LOG_WAGP",
   data = adult_2013_train,
   key = "log_wagp_glm_relp_schl",
```

```
    family = "gaussian",
    lambda = 0)
```

La qualité prédictive de chaque modèle est évaluée par le critère AIC.

```
log_wagp_glm_relp@model$aic
log_wagp_glm_schl@model$aic
log_wagp_glm_relp_schl@model$aic
```

Il est opportun de prendre en compte l'interaction mais ce n'est pas la façon de procéder la plus optimale ! Sans vouloir étudier toutes les interactions d'ordre 2, un arbre de régression aurait pu mettre l'accent sur le couple de variables le plus pertinent à prendre en compte.

Le modèle finalement estimé prend en compte toutes les variables sauf celle issues du revenu par discrétisation et l'interaction pour estimer le log du revenu. Le paramètre alpha] règle l'équilibre entre pénalisation Lasso et *ridge* (4 valeurs possibles) tandis que celui lambda règle l'importance globale de la pénalisation; 10 valeurs sont automatiquement générées. C'est donc un ensemble de 50 modèles qui sont estimés pour les différentes valeurs des paramètres de la grille.

```
# Liste de variables
addpredset = c("COW", "MAR", "INDP", "RAC1P",
    "SEX", "POBP", "AGEP",
    "WKHP", "LOG_CAPGAIN", "LOG_CAPLOSS")
# Estimation du modèle gaussien
log_wagp_glm_grid = h2o.glm(x = c("RELP_SCHL",
    addpredset), y = "LOG_WAGP",
    data = adult_2013_train,
    key = "log_wagp_glm_grid",
    family = "gaussian",
    lambda_search = TRUE,
    nlambda = 10,
    return_all_lambda = TRUE,
    alpha = c(0, 0.25, 0.5, 0.75, 1))
# Le smodèles sont stockés en listes
```

```
class(log_wagp_glm_grid)
getClassDef("H2OGLMGrid")
class(log_wagp_glm_grid@model[[1L]])
getClassDef("H2OGLMModelList")
length(log_wagp_glm_grid@model[[1L]]@models)
class(log_wagp_glm_grid@model[[1L]]@models[[1L]])
```

Il s'agit ensuite de sélectionner le meilleur couple (alpha, lambda) mais la procédure proposée dans le tuteuriel officiel est "douteuse" car l'échantillon test est utilisé comme échantillon de validation pour optimisé ce choix. Ce même échantillon est ensuite réutilisé pour comparer avec forêt aléatoire, boosting et apprentissage profond. Néanmoins, comme le modèle linéaire est moins performant c'est sans enjeu mais un point important à signaler quant à la limitation des algorithmes de sélection de variables avec données massives.

```
log_wagp_glm_grid@model[[1L]]@models[[1L]]
   @model$params$alpha # ridge
log_wagp_glm_grid@model[[2L]]@models[[1L]]
   @model$params$alpha
log_wagp_glm_grid@model[[3L]]@models[[1L]]
   @model$params$alpha
log_wagp_glm_grid@model[[4L]]@models[[1L]]
   @model$params$alpha
log_wagp_glm_grid@model[[5L]]@models[[1L]]
   @model$params$alpha # lasso
log_wagp_glm_grid_mse =
  sapply(log_wagp_glm_grid@model,
       function(x)
         sapply(x@models, function(y)
           h2o.mse(h2o.predict(y, adult_2013_test),
                   actual_log_wagp)))
log_wagp_glm_grid_mse
```

```
log_wagp_glm_grid_mse == min(log_wagp_glm_grid_mse)
log_wagp_glm_best =
   log_wagp_glm_grid@model[[5L]]@models[[10L]]
```

Question: quelle est la régression optimale retenue?

Boosting

L'algorithme de *Gradient Boosting Model* pour la régression ou la classification est retenu avec les paramètres suivants:

```
args(h2o.gbm)
```

Ceux les plus importants: (interaction.depth et surtout schrinkage sont ceux trouvés dans la fonction R de la librairie gbm avec, comme précédemment, un paramètre nbins=20 mais pas de taux d'échantillonnage.

La procédure considère une grille avec 3 nombres d'itérations et 3 valeurs de rétrécissement.

```
log_wagp_gbm_grid = h2o.gbm(x = c("RELP",
   "SCHL", addpredset),
   y = "LOG_WAGP",
   data = adult_2013_train,
   key = "log_wagp_gbm_grid",
   distribution = "gaussian",
   n.trees = c(10, 20, 40),
   shrinkage = c(0.05, 0.1, 0.2),
   validation = adult_2013_test,
   importance = TRUE)
```

Attention, le tuteuriel officiel "triche" à nouveau en utilisant ci-dessus l'échantillon test comme échantillon de validation ce qui biaise l'optimisation en la rendant optimiste. En laissant le paramètre validation en blanc et fixant le paramètre nfolds=10 on retrouve la procédure de validation croisée de la fonction gbm de R mais il est plus judicieux dans le cas de données massives de fixer par exemple holdout.fraction=2/5 une fraction de l'échantillon d'apprentissage comme échantillon de validation.

```
log_wagp_gbm_grid = h2o.gbm(x = c("RELP",
   "SCHL", addpredset),
   y = "LOG_WAGP",
   data = adult_2013_train,
   key = "log_wagp_gbm_grid",
   distribution = "gaussian",
   n.trees = c(10, 20, 40),
   shrinkage = c(0.05, 0.1, 0.2),
   holdout.fraction=2/5,
   importance = TRUE)
```

Forêt aléatoire

L'algorithme des forêts aléatoires est classique mais avec deux paramètres supplémentaires en plus du nombre d'arbres (ntree défaut 50) et du nombre mtries (défaut usuel) de variables tirées.

```
# liste des paramètres avec valeurs par défaut
args(h2o.randomForest)
```

- Le premier est sample.rate=2/3 par défaut qui règle donc la taille de chaque échantillon tiré pour réduire les temps de calcul. Pour une valeur de 1, on retrouve la procédure *bootstrap* usuelle mais il n'est pas explicité la façon de tirer chaque échantillon. On peut penser que chacun est tiré avec remise avec un effectif de $2n/3$ ce qui rendrait donc les arbres plus "indépendants" mais "moins bien ajustés".
- Le deuxième qui passe presque inaperçu est nbins=20 par défaut. C'est le nombre max de modalités considérées pour une variables qualitative, les moins fréquentes sont agrégées en une seule et aussi le nombre de valeurs possibles et donc le nombre de classes d'une variable quantitative qui est discrétisée d'autorité. C'est à nouveau pour réduire drastiquement les temps de calcul.

Il est important de noter que ces paramètres règles le curseur entre qualité de l'optimisation et temps de calcul. C'est le réglage du troisième terme d'erreur avec celui d'approximation (biais) et d'estimation (variance) bien connus en

apprentissage. Les valeurs entrées ont été rendues plus favorable aux forêts aléatoires par rapport au tuteuriel "officiel" qui privilégie donc GBM.

```
log_wagp_forest = h2o.randomForest(x =
   c("RELP", "SCHL", addpredset),
   y = "LOG_WAGP",
   data = adult_2013_train,
   key = "log_wagp_forest",
   classification = FALSE,
   nbins=100,
   depth = 20,
   ntree = 200,
   validation = adult_2013_test,
   seed = 8675309,
   verbose=TRUE,
   type = "BigData")
```

Il serait opportun d'optimiser `mtries` mais la procédure n'est pas documentée et même avec `verbose=TRUE, oobee=TRU` les erreurs *out of bag* ne sont pas listées. Une insuffisance de l'implémentation.

Deep learning

L'algorithme connexionniste d"apprentissage profond" n'est pas, ou pas encore, décrit sur le site wikistat mais très bien détaillé sur celui de Wikipédia en anglais. L'implémentation dans H2O est plus précisément documentée dans une brochure. Celle-ci montre la grande complexité engendrée par la procédure d'apprentissage et le nombre de choix / paramètres à régler pour optimiser un critère :

```
args(h2o.deeplearning)
```

Comme pour le boosting, le tuteuriel officiel "triche" en utilisant l'échantillon test pour optimiser le *deep learning*. Le paramètre est modifié pour utiliser en lieu et place de l'échantillon test une portion de l'échantillon d'apprentissage.

```
log_wagp_dl = h2o.deeplearning(x = c("RELP",
   "SCHL", addpredset),
   y = "LOG_WAGP",
   data = adult_2013_train,
   key = "log_wagp_dl",
   classification = FALSE,
   holdout_fraction=2/5)
```

sans plus d'expérience, il serait difficile de tenter de régler les autres paramètres

Prévision de l'échantillon test

```
h2o.mse(h2o.predict(log_wagp_glm_best,
   adult_2013_test),actual_log_wagp)
h2o.mse(h2o.predict(log_wagp_gbm_best,
   adult_2013_test),actual_log_wagp)
h2o.mse(h2o.predict(log_wagp_forest,
   adult_2013_test),actual_log_wagp)
h2o.mse(h2o.predict(log_wagp_dl,
   adult_2013_test),actual_log_wagp)
```

Comparer les erreurs de prévision, et juger de la méthode la plus facile à optimiser. Le tuteuriel officiel classe en premier le *deep learning* [2] suivi du boosting, de la forêt aléatoire et enfin du modèle gaussien [3].

3.2 Classification

La même démarche peut être mise en œuvre sur un problème de classification. Très voisine de ce qui vient d'être fait, elle n'est pas reprise car il suffit de suivre la documentation. Il s'agit de modéliser la présence ou non d'un revenu parmi les 2% les plus élevés.

- Régression logistique : la qualité de prévision des modèles est obtenue par le tracé de la courbe ROC ou l'évaluation de l'AUC équivalente à une concentration de Gini. La pénalisation *Elastic Net* est encore optimisée sur l'échantillon test.
- GBM et même problème d'utilisaiton de l'échantillon test pour optimiser rétrécissement et nombre d'arbres.
- Forêt aléatoire sans chercher à optimiser les paramètres (`mtries`) et une profondeur réduite des arbres.

2. dans lequel la société Oxdata a beaucoup investi.
3. Ne pas y voir le résultat d'un conflit d'intérêt avec les conseillers scientifiques ou commerciaux.

- *Deep learning* optimisé sur l'échantillon test.

Conclusion

Autres exemples.

- Le tuteuriel officiel utilise un modèle d"apprentissage profond" (avec confusion validation / test) pour la reconnaissance (classification supervisée) des chiffres manuscrits des célèbres données MNIST. Elles seront abordées dans un autre scénario.
- Des exemples d'utilisation des autres algorithmes sont décrits : k-means, ACP, ainsi qu'une procédure intéressante de détection d'atypiques (*outliers*) multidimensionnels dans les données de reconnaissance de caractères ; détection des chiffres les plus "mal écrits".
- Deep learning (resp. Random forest) de H2O est comparé avec `nnet` (resp. la version) de R sur les données MNIST (resp. de la KDDCup98).

Remarques :

- Se méfier des comparaisons proposées dans un tuteuriel produit par une société commerciale (H2O *vs.* R). Il n'est pas sûr que celle-ci soient strictement objectives.
- Ce tuteuriel souligne les principaux enjeux émergents avec l'analyse de données massives :
  - Impossibilité de sélectionner automatiquement des interactions dans un modèle linéaire avec des algorithmes implémentés pour des données massives.
  - Même chose pour la bonne optimisation des paramètres de complexité quelque soit la méthode utilisée.
  - Attention à l'optimisation du compromis : temps de calcul et précision des résultats en particulier en échantillonnant les données pour réduire les temps de calcul afin de mieux optimiser les modèles.
  - Beaucoup de méthodes ne sont pas implémentées (SVM, PLS, KRLS...).
- Rigueur et déontologie ne peuvent, à moyen terme, qu'être bénéfiques.