TP7 Khalis - Mekkoudi - Collas

```
source(file="cv_classif.R")
```

Astronomy

Chargement des données

```
astronomy <- read.table('data/astronomy_train.csv', sep = ",", header = TRUE)
```

Nous commençons par charger les données du jeu de données 'astronomy.csv'. Ce problème est un problème de classification comportant 3 classes. Ce jeu de données comporte 5000 individus décrits par 17 variables. Toutes les variables sont quantitatives. La proportion du nombre d'individus pour chaque classe est la suivante:

	Proportions
GALAXY	0.5038
QSO	0.0836
STAR	0.4126

Séparation des données en entrainement/test

Nous séparons les données en un ensemble d'entrainement sur lequel nous allons tester tout nos modèles et un ensemble de test qui nous permettra d'évaluer l'erreur pour le modèle que nous aurons retenu. Afin de conserver la même séparation des données à chaque execution du notebook nous fixons le générateur de nombres aléatoires (seed).

```
set.seed(42)
train_size <- nrow(astronomy) * (2/3)
train <- sample(1:nrow(astronomy), size = train_size)
astronomy_train <- astronomy[train,]
astronomy_test <- astronomy[-train,]
ncol(astronomy_train)
```

```
## [1] 18
```

```
nrow(astronomy_train)
```

```
## [1] 3333
```

```
nrow(astronomy_test)
```

```
## [1] 1667
```

Transformations des données d'entrainement

Nous appliquons plusieurs transformations à l'ensemble d'entrainement:

Les variables 'objid' et 'rerun' n'ont qu'une seule valeur. Nous les retirons puisqu'elles ne nous servirons pas à discriminer les 3 classes.

Nous centrons et réduisons les variables. En effet certaines variables comme 'specobjid' sont en 10^18 alors que d'autres variables sont en 10^-4 comme 'redshift'.

Nous affichons 3 individus à titre d'exemple:

ra	dec	u	g	r	i	z	run	camcol	field	specobjid	redshift	plate	mjd	fiberid	y
-0.3	-0.5	0.3	-0.3	-0.6	-0.7	-0.7	-0.8	1.4	0.0	-0.5	-0.2	-0.5	-0.6	-0.7	GALAXY
1.6	-0.6	-2.0	-1.7	-1.3	-1.0	-0.8	-0.9	0.8	1.9	-0.6	-0.4	-0.6	-0.8	0.9	STAR
0.5	-0.6	1.1	1.0	0.8	0.8	0.7	1.3	0.2	-1.7	3.4	-0.4	3.4	2.5	2.6	STAR

Méthode d'évaluation

Nous évaluons chaque modèle sur l'ensemble d'entrainement à l'aide d'une validation croisée (avec 10 parties). Nous calculons l'erreur moyenne et l'écart type. Nous choisissons le modèle ayant la plus faible erreur de classification pour notre modèle final. Enfin, nous estimons l'erreur de notre modèle final sur l'ensemble de test.

Validation croisée

Nous avons implémenté une fonction qui calcule l'erreur de classification et l'écart type par validation croisée pour l'ensemble des modèles que nous utiliserons par la suite. Cette fonction s'appelle 'CV_eval'.

NB: Nous utilisons la constante 'RECHERCHE_HYPERPARAMETRES' afin de ne pas rechercher les hyperparamètres à chaque execution de ce notebook.

```
RECHERCHE_HYPERPARAMETRES <- TRUE
```

Analyses discriminantes

```
set.seed(123)
#validation croisée sur l'ensemble d'entrainement
res <- data.frame("Erreur"=rep(0,3), "Ecart type"=rep(0,3))
res[1,] <- CV_eval('adl', astronomy_train)
res[2,] <- CV_eval('adq', astronomy_train)
res[3,] <- CV_eval('bayesien_naif', astronomy_train)
```

Nous obtenons les erreurs suivantes:

	Erreur	Ecart type
ADL	0.0906091	0.0124512
ADQ	0.0276028	0.0091547
Bayesien naif	0.0732073	0.0192217

Nous ne testons pas la régression la régression logistique puisque la'analyse discriminante linéaire donne de mauvaises performances.

L'analyse discriminante quadratique est l'algorithme le plus performant des trois modèles testés précédemment. Les modèles plus complexes fonctionnent donc bien sur ce jeu de données. Nous testons donc SVM avec différents noyaux.

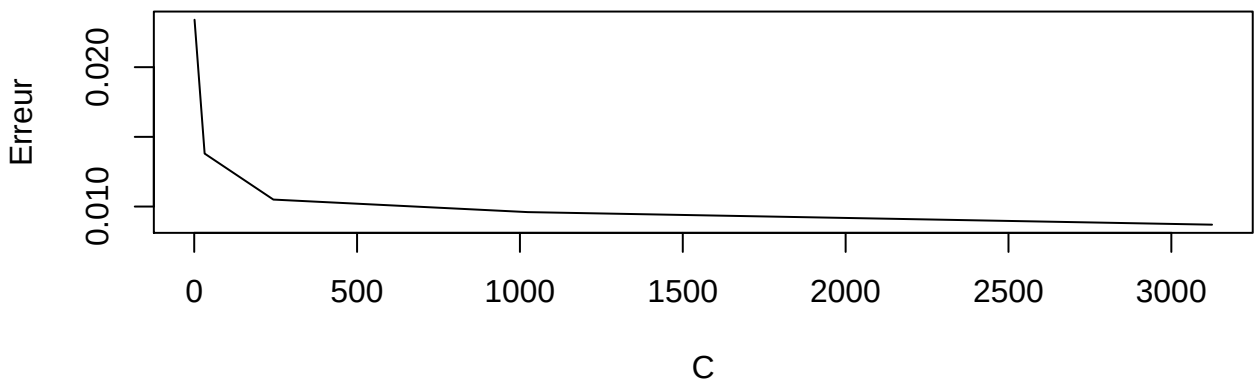
SVM

Optimisation C

Les SVM ont l'hyperparamètre C: si C est faible alors le SVM aura une large marge quitte à mal classer certains points (on peut dans ce cas se retrouver dans un cas de sous entrainement). De manière opposée, si C est grand alors le SVM classifira bien beaucoup de points mais aura une faible marge et donc risque un surentrainement. Nous cherchons le meilleur compromis.

Nous cherchons l'hyperparamètre C minimisant l'erreur de classification pour un noyau linéaire. Nous testons différents C: 1, 32, 243, 1024 et 3125. Pour chaque valeur nous calculons l'erreur sur le jeu d'entrainement par validation croisée.

```
if (RECHERCHE_HYPERPARAMETRES){
  set.seed(123)
  C_list <- (1:5)^5
  erreurs_C <- rep(0, length(C_list))
  for (i in 1:length(C_list)){
    hyperparametres <- list(kernel="vanilla", C=C_list[i], kpar=list())
    erreurs_C[i] <- CV_eval('svc', astronomy_train, hyperparametres)[1]
  }
  plot(C_list, erreurs_C, type="l",xlab="C", ylab="Erreur")
  C <- C_list[which.min(erreurs_C)]
}else{
  C <- 3125
}
```



La valeur de C minimisant l'erreur est 3125.

SVM à noyaux

Les différents noyaux vont nous permettre de trouver des fontières de décision non linéaires.

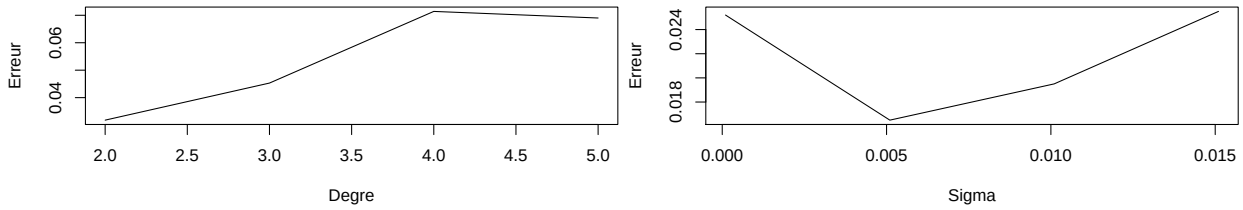
Nous testons trois noyaux pour les SVM:

1. noyau linéaire ('vanilladot') qui n'a pas d'hyperparamètres
2. noyau polynomial pour lequel il faut régler le degré du noyau. Nous testons cet hyperparamètre pour des valeurs de 2 à 5. Nous retenons le degré donnant la plus faible erreur par validation croisée.
3. noyau gaussien pour lequel il faut régler le paramètre sigma. Nous testons cet hyperparamètre pour des valeurs de 0.0001 à 0.02 par pas de 0.005. Nous retenons le sigma donnant la plus faible erreur par validation croisée.

L'idéal serait d'optimiser C pour différents noyaux. Cependant chaque noyau possède d'autres hyperparamètres et essayer toutes les combinaisons se révèle trop couteux en calculs. Par conséquent nous conservons la valeur trouvée avec le noyau linéaire.

Nous observons que le noyau polynomial n'améliore pas les performances de notre classifieur: plus le degré du noyau est élevé plus les performances se dégradent.

De même le noyau gaussien n'améliore pas les perfomances par rapport à un noyau linéaire.



Nous obtenons les erreurs suivantes:

	Erreur	Ecart type
vanilla	0.0081008	0.0040398
poly	0.0339034	0.0094051
rbf	0.0159016	0.0088564

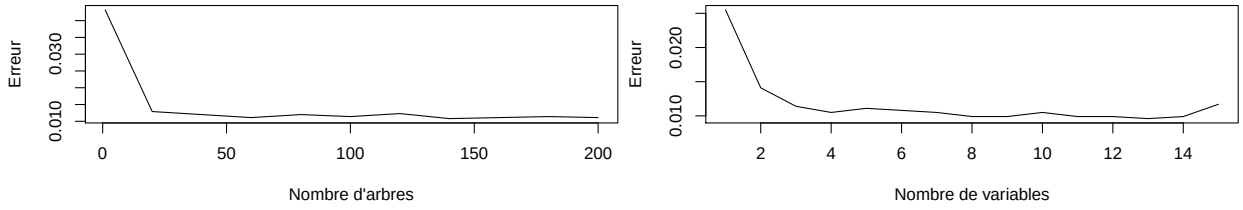
Forêts aléatoires

Nous essayons maintenant les forêts aléatoires.

Nous devons tout d'abord nous assurer que notre fôret aléatoire contient suffisamment d'arbres. En effet, si le nombre d'arbres est trop faible certains individus ne seront pas utilisés (à cause du bootstrap). Si le nombre d'arbres est élévé la recherche de l'hyperparamètre 'mtry' (que nous faisons ensuite) sera très longue. Pour cela nous calculons l'erreur de classification par validation croisée pour des fôrets ayant entre 1 et 200 arbres (par pas de 20).

Ensuite, nous avons un hyperparamètre à optimiser qui est 'mtry' dans la bibliothèque 'randomForest'. Cet hyperparamètre correspond au nombre de variables sélectionnées, par échantillonnage, comme candidates pour chaque noeud. Plus il y a de variables meilleure sera la séparation des données à chaque noeud mais les arbres seront similaires et donc l'agrégation des arbres améliorera peu les performances (par rapport à un arbre seul).

Le nombre de variables minimisant l'erreur de classification est 9.



Nous obtenons l'erreur suivante:

Erreur	Ecart.type
0.0108011	0.005559

Modèle retenu

Le modèle retenu est le SVM avec un noyau linéaire puisqu'il correspond au modèle ayant la plus faible erreur de classification. Nous évaluons l'erreur sur l'ensemble de test (réservé à cet effet au départ): cette erreur nous donne une estimation non biaisée de l'erreur de classification.

```
astr.mod <- ksvm(y~., data=astronomy_train, type="C-svc", kernel='vanilla', C=3125, cross=0)
```

```
##  Setting default kernel parameters
```

```
save(centers, scales, astr.mod, file="env_astronomy.RData")
```

```
rm(list=setdiff(ls(), "astronomy_test"))
astronomy_test_y <- astronomy_test$y
astronomy_test_X <- astronomy_test
astronomy_test_X$y <- NULL
source(file="predicteurs.R")
y_pred <- classifieur_astronomie(astronomy_test_X)
```

Nous obtenons la matrice de confusion suivante sur le jeu de test:

	GALAXY	QSO	STAR
GALAXY	838	10	0
QSO	0	138	0
STAR	3	0	678

La précision est de 99.2%