T. D. n° 1
La subset selection

Résumé

Ce document est le T.D. n° 1 du module Modèle pour la Data Science. Il reprend rapidement des éléments du cours et propose une mise en pratique des méthodes de sélection de variables.

1 Chargement du jeu de données

Nous travaillons ici avec un ensemble de données simulées contenant des informations sur 20 variables sur des joueurs de la ligue de baseball américaine durant les années 1986 à 1987. Vous allez essayer de prédire le salaire de ces joueurs grâce aux 19 variables restantes avec une régression linéaire multiple. Commencez par charger le jeu de données `Hitters` via le package `ISLR` sous `R`. [1]

```
install.packages("ISLR")
install.packages("leaps")
library(ISLR)
library(leaps)
```

Nous disposons de 20 variables dans ce jeu de données qui contient 322 observations :

1. `AtBat` : Number of times at bat in 1986
2. `Hits` : Number of hits in 1986
3. `HmRun` : Number of home runs in 1986
4. `Runs` : Number of runs in 1986
5. `RBI` : Number of runs batted in in 1986
6. `Walks` : Number of walks in 1986
7. `Years` : Number of years in the major leagues
8. `CAtBat` : Number of times at bat during his career
9. `CHits` : Number of hits during his career
10. `CHmRun` : Number of home runs during his career
11. `CRuns` : Number of runs during his career
12. `CRBI` : Number of runs batted in during his career
13. `CWalks` : Number of walks during his career

1. Ce T.D. est inspiré d'un article de Serge Nakache sur le site https ://lilbigdata-boy.wordpress.com.

14. `League` : A factor with two levels A and N indicating player's league at the end of 1986
15. `Division` : A factor with two levels E and W indicating player's division at the end of 1986
16. `PutOuts` : Number of put outs in 1986
17. `Assists` : Number of assists in 1986
18. `Errors` : Number of errors in 1986
19. `NewLeague` : A factor with two levels A and N indicating player's league at the beginning of 1987
20. `Salary` : 1987 annual salary on opening day in thousands of dollars.

L'année dernière, vous avez peut-être déjà étudié la régression linéaire multiple qui utilisait plusieurs variables pour la construction d'un modèle. Lorsque nous disposons de beaucoup de variables explicatives, il faut sélectionner les variables qui seraient utiles dans le modèle.

C'est à vous !

1. Supprimez les données manquantes du jeu de données initial à l'aide de la fonction `na.omit()`. Pourquoi devez-vous faire cela ?
2. Donnez deux raisons majeures pour lesquelles nous souhaitons réduire le nombre de variables.
3. Construisez un échantillon test composé de 180 observations sur les 263 restantes à l'aide de la fonction `sample()`.

2 La subset selection

Nous tentons de modéliser le salaire de l'année 1987 des joueurs de la ligue de baseball américaine à partir des 19 variables explicatives restantes. Pour chaque complexité (le nombre de paramètres d'un modèle) allant de 1 à 19, nous allons sélectionner le meilleur modèle grâce à une procédure `forward stepwise selection` réalisée avec le `training set`. Nous allons donc obtenir 19 modèles comprenant entre une et 19 variables et nous allons les stocker dans :

`best_models19_forward.`

La **best subset selection** teste toutes les combinaisons possibles de variables et regarde quel est le meilleur modèle. Cette technique exhaustive, a cependant un coût énorme. Pour plus d'informations, vous pouvez revenir au T.D. d'estimation de 4A consacré à la régression linéaire simple.

Dans la `forward stepwise selection` pour un nombre fixé k de variables dans le modèle (k est inférieur au nombre total de variables explicatives), nous construisons d'abord un modèle M_0, qui ne contient que l'`intercept`. Puis nous construisons

ensuite un modèle M_1, qui est le meilleur modèle à une variable explicative selon le coefficient de détermination R^2. Nous construisons ensuite un modèle M_2, qui est le meilleur modèle avec la variable explicative de M_1 ET une autre variable. Nous réitérons l'opération jusqu'au modèle M_k. Voici les lignes de commande :

```
> best_models19_forward=regsubsets(Salary~.,data=Hitters[train,],
nvmax=19,method='forward')
# Pour accéder aux coefficients du modèle 5, nous appelons
la fonction coef().
# Pour accéder au RSS des modèles, nous lançons la fonction summary().
> coef(best_models19_forward,5)
 (Intercept)        Walks        CRuns       CWalks    DivisionW
  63.0001969    6.3210070    1.1833383   -0.8077783 -157.3429617
  PutOuts
  0.3606914
> summary(best_models19_forward)$rss
 [1] 26087576 22818891 21189931 20102279 19064320 18680019
 [7] 18215542 17767946 17494128 17269461 17154340 16995989
 [13]16928512 16863474 16833614 16802615 16801619 16800998
 [19]16800985

# Nous avons 19 modèles. Il faut choisir le meilleur.
# Pour ce faire, nous allons appliquer chacun des modèles sur le test
#set et calculer la MSE (Mean Square Error) des modèles.
> mse=rep(NA,19)
> test=model.matrix(Salary~.,data=Hitters[-train,])
> for(i in 1:19)
{coefi=coef(best_models19_forward,id=i)
+pred=test[,names(coefi)]%*%coefi
+mse[i]=mean((Hitters$Salary[-train]-pred)^2)}
```

Si le modèle est de la forme $a * x + b$, `Intercept` correspond ici à l'ordonnée à l'origine : le coefficient b de la droite.

C'est à vous !

4. Dans le modèle 5, quelle variable a le coefficient le plus élevé ? Le plus bas ?
5. Interprétez le modèle 5.
6. Définissez la RMSE. Quelle est la différence entre la RMSE et la MSE ?
7. Affichez les RMSE des 19 modèles sur le `training set` et sur le `test set`. Vous devriez obtenir la figure ci dessous.
8. Concluez sur la modélisation.

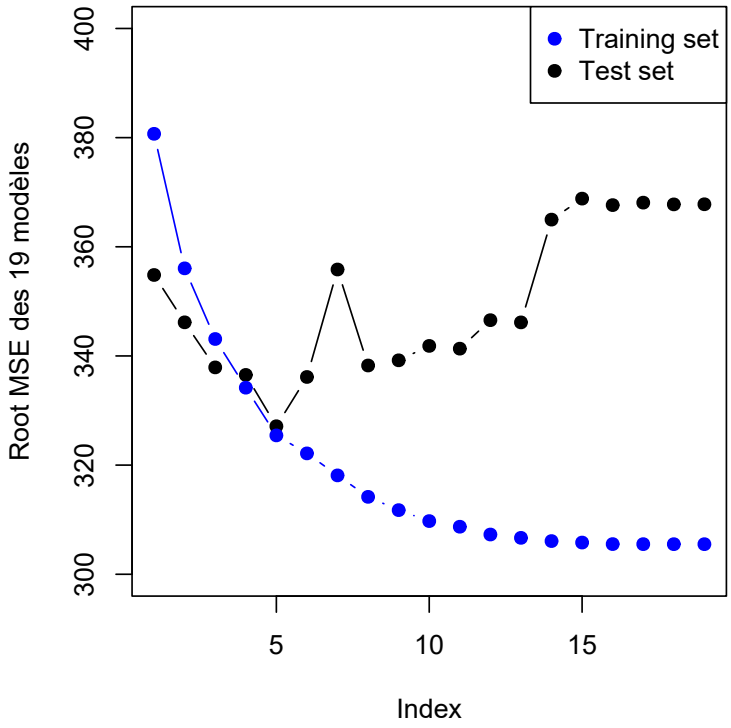

FIGURE 1 – RMSE des 19 modèles sur le `training set` et sur le `test set`.

3 Forward stepwise avec cross-validation

La procédure `forward stepwise`, montre des bons résultats lorsque nous avons suffisamment de données pour découper le jeu de données en deux sous-ensembles (`training set` et `test set`). Si nous ne souhaitons pas diviser le jeu de données initial, nous pouvons utiliser la méthode de `cross-validation`. Le principe est le suivant :

a) Division du jeu de données en K^* sous-ensembles de taille égale (K souvent égal à 10).

b) Construction des x modèles sur le premier $K-1$ sous-ensemble.

c) Application des modèles sur le Kème sous-ensemble. Ce Kème sous-ensemble sert de mini `test set`.

d) Enregistrement des MSE test des x modèles sur le Kème sous-ensemble.

e) Nous recommençons le principe pour les 9 autres $K-1$ sous-ensembles.

f) Nous moyennons les erreurs des 10 Kème sous-ensembles pour les x modèles.

```
# Nous construisons la fonction predictFUN() car il n'existe pas
```

```
# de fonction predict() dans le package leaps.
> predictFUN=function(object,newdata,id,...){
  form=as.formula(object$call[[2]])
  mat=model.matrix(form,newdata)
  coefi=coef(object,id=id)
  mat[,names(coefi)]%*%coefi
}

#
# Ici nous construisons manuellement une 10-folds cross validation
# pour sélectionner le meilleur modèle.
# Pour chacun des 10*k-1 subsets, nous allons produire 19 meilleurs
# modèles, un pour chaque complexité.
# Nous allons calculer la MSE des 19 modèles sur chacun des 10k
# subsets.
# Enfin, nous moyennons cette MSE pour chacun des 19 modèles.

> folds=sample(rep(1:10,length=nrow(Hitters)))
> RSS_training=matrix(NA,10,19)
> for(k in 1:10){
  models19_training_cv=regsubsets(Salary~.,data=Hitters[folds!=k,],
                                  nvmax=19,method='forward')
  for(i in 1:19){
    pred=predictFUN(models19_training_cv,Hitters[folds==k,],id=i)
    RSS_training[k,i]=mean( (Hitters$Salary[folds==k]-pred)^2)
  }
}
```

C'est à vous !

9. Affichez les moyennes des 10 RMSE des 19 modèles. Vous devriez obtenir la figure ci-dessous.
10. Listez les modèles équivalents.
11. Quel modèle choisissez-vous ? Justifiez votre réponse.

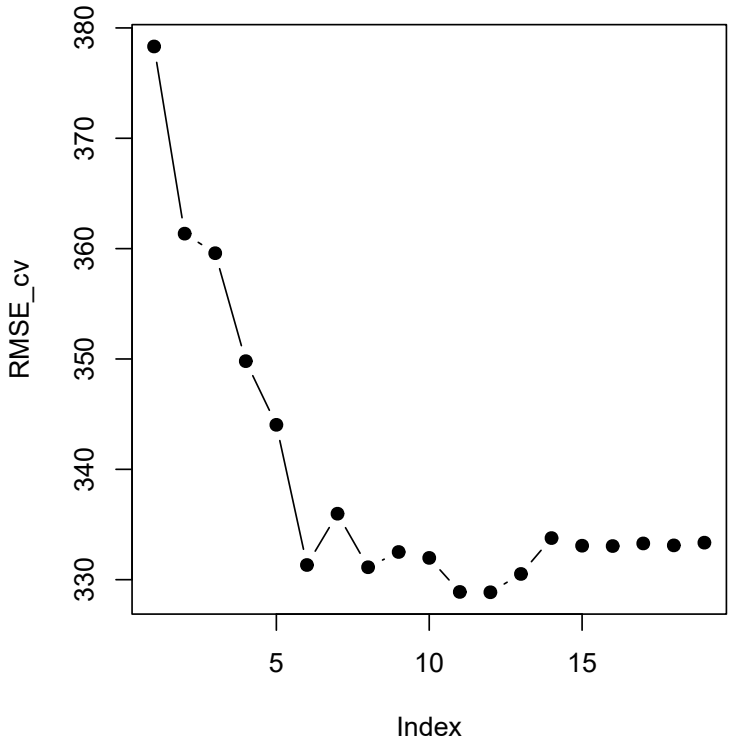

FIGURE 2 – Moyennes des 10 RMSE des 19 modèles avec la cross-validation.

4 Les modèles de contraction des coefficients

Jusqu'ici, les modèles utilisés dans les techniques de `subset` étaient des modèles qui cherchaient à minimiser la SSE, *i*.e. la somme des carrés des erreurs. Avec les modèles de contraction, c'est un peu plus subtile. Nous allons rajouter à la quantité que nous cherchons à minimiser un coefficient de pénalité de sorte à empêcher que les coefficients des paramètres du modèle ne soient pas trop grands (nous prouvons que des coefficients trop grands favorisent l'`overfitting`). Il existe de grands modèles de contraction : RIDGE et LASSO. On ne va s'intéresser qu'à LASSO dont la structure permet de faire de la sélection de variables.
Le modèle LASSO cherche à minimiser la quantité suivante :

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{i,j}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = R.S.S + \lambda\sum_{j=1}^{p}|\beta_j| \qquad (1)$$

$\sum_{j=1}^{p}|\beta_j|$ est la **Shrinkage Penalty**. Plus cette valeur est élevé plus le modèle a tendance à vouloir réduire la taille des coefficients les moins pertinents jusqu'à zéro. C'est comme ça que LASSO fait de la sélection de variables.

Le paramètre λ, est le **Tuning Parameter**. Il représente à quel point on veut pénaliser la taille des coefficients. Si λ vaut zéro alors on ne veut rien pénaliser, et notre modèle devient un modèle classique des moindres carrés. Si λ est élevé alors le modèle va avoir tendance à donner des poids extrêmement faible aux paramètres (proche de zéro ou égal à zéro). Le choix du paramètre λ est crucial. On peut construire autant de modèles LASSO qu'il existe de valeurs pour λ, soit une infinité. Alors comment fait-on en pratique ? Comme dans le cas des techniques de subset : soit en divisant le sets en deux (training et test set) soit par cross validation. On commence par la cross validation qui est la méthode la plus utilisée.
La fonction `cv.glmnet` du package `glmnet` permet de lancer une cross validation sur un set de modèles LASSO pour une range de lambda.On obtient une centaine de modèles LASSO associé à une valeur de λ différente. En utilisant la fonction plot on obtient directement le graphique représentant les MSE des modèles.

```
> library(glmnet)
> x=model.matrix(Salary~.-1,data=Hitters)
> y=Hitters$Salary
> lasso_model_cv=cv.glmnet(x,y,alpha=1)
> plot(lasso_model_cv)
> numero_du_best_model=which(lasso_model_cv$lambda==
   lasso_model_cv$lambda.min)
> lasso_model_cv$glmnet.fit$beta[,numero_du_best_model]
        AtBat          Hits         HmRun          Runs
          RBI         Walks         Years        CAtBat
          CHits        CHmRun
-1.547343e+00  5.660897e+00  0.000000e+00  0.000000e+00  0.000000e
   +00  4.729691e+00 -9.595837e+00  0.000000e+00  0.000000e+00
   5.108207e-01
        CRuns          CRBI        CWalks       LeagueA
          LeagueN     DivisionW       PutOuts       Assists
                 Errors    NewLeagueN
 6.594856e-01  3.927505e-01 -5.291586e-01 -3.206508e+01  4.302928e
    -14 -1.192990e+02  2.724045e-01  1.732025e-01 -2.058508e+00
    0.000000e+00
>
```

À vous !

a) Détaillez la fonction `cv.glmnet()`

b) Quel est le meilleur modèle ?

c) Quel est la valeur de son λ ?

d) Pourquoi obtenons-nous 71 modèles et pas 100 ?

e) Quelles sont les variables retenues par le meilleur modèle ?

FIGURE 3 – 71 modèles pour 71 valeurs du paramètre lambda