BUT3-STID - Ressource 5-02

Data Mining - Pratique sous R

Sophie Lambert-Lacroix

On reprend le jeu de données "*Fleurs d'iris*". Rappelons que ce jeu de données comprend un total de 150 observations, réparties de manière égale entre trois espèces de fleurs d'iris (*setosa*, *virginica* et *versicolor*). Quatre caractéristiques sont mesurées pour chaque observation : la longueur (X_1) et la largeur (X_2) du sépale et la longueur (X_3) et la largeur (X_4) du pétale, en centimètres. Dans cet exemple, on souhaite prédire l'espèce d'iris à partir de la longueur et la largeur du sépale et du pétale. Ces données sont directement accessibles sous `R` sous le nom `iris`.

1 La méthode k-NN

Cette méthode est implémentée pour la distance euclidienne dans la librairie `class` est s'appelle `knn`.

1. Décrire comment fonctionne cette procedure; on décrira à quoi correspondent les entrées et la nature de la sortie.

2. Tester cette procédure avec comme échantillon d'apprentissage les 25 premières iris de chaque espèce et comme échantillon test celles qui restent et avec $k = 3$. Construire la matrice de confusion associée aux données "test" en utilisant la fonction `table` (attention à l'ordre dans lequel vous entrez les deux vecteurs : prédictions et observations). En déduire les calculs du taux d'erreur et des précisions associées aux trois modalités.

3. Programmer une fonction qui, à partir d'un vecteur réponse `y` de longueur n, de la matrice des prédicteurs associés `X` de taille $n \times p$ et d'un vecteur contenant plusieurs valeurs de k (nombre de voisins), retourne le nombre optimal de voisins. Pour choisir ce nombre de voisins optimal, on utilisera une validation croisée de type 10-fold afin de minimiser l'erreur estimée que l'on commet. Pour le tirage aléatoire des individus on utilisera la fonction `sample` qui permet d'extraire de manière aléatoire un sous-échantillon (de taille donnée) d'indice parmi un vecteur d'indices de taille plus grande.

4. Tester votre fonction sur le jeu entier des données `iris` avec une grille de k égale à $(3, 5, 7)$.

5. Programmer l'étude de ré-échantillonnage. Les données d'entrée sont un vecteur réponse `y` de longueur n, la matrice des prédicteurs associés `X` de taille $n \times p$,

un vecteur contenant plusieurs valeurs de k (nombre de voisins) et un nombre de répétitions (découpage "échantillon d'apprentissage - échantillon test") N. On stockera les indices des N échantillons d'apprentissage dans une matrice de taille $N \times$ (70% du nombre d'individus). Afin de rendre reproductible vos résultats on utilisera la procédure `set.seed` pour fixer la graine du générateur aléatoire avant de générer cette matrice. Les données de sorties sont les N valeurs du nombre optimal de voisins associées, ainsi que les N taux d'erreurs et les N précisions pour chaque modalité résultants des N découpages "échantillon d'apprentissage - échantillon test".

6. Tester votre programme sur le jeu entier des données `iris` avec une grille de k égale à $(3, 5, 7, 9, 11, 13)$ et avec $N = 500$. Tracer les boxplots associés aux taux d'erreurs et aux trois précisions (4 boxplots en tout) et déterminer la répartition du nombre optimal de voisins. Conclure sur l'utilisation de la méthode k-NN sur ce jeu de données.

2 La méthode LDA

Cette méthode est implémentée dans la librairie `MASS` est s'appelle `lda`.

1. On se propose de tester cette procédure avec comme échantillon d'apprentissage les 25 premières iris de chaque espèce et comme échantillon test celles qui restent. Exécuter les lignes de commandes suivantes :

```
ind.train <- c(1:25,51:75,101:125)
attach(iris)
res <- lda(formula=Species ~ ., data = iris, prior = c(1,1,1)/3,
subset = ind.train)
respredict <- predict(res, iris[-ind.train, ])
```

   Décrire à quoi correspondent les entrées `formula`, `data`, `prior` et `subset`, les sorties `respredict$class` et `respredict$posterior`.

2. Construire la matrice de confusion associée aux données "test". En déduire les calculs du taux d'erreur et des précisions associées aux trois modalités. Comparer ces résultats avec ceux obtenus avec la méthode k-NN.

3. La procédure `greedy.wilks` du package `klaR` permet d'effectuer une sélection de variables via la méthode ascendente (forward), basée sur le Λ de Wilks. Procéder à une sélection des variables les plus discriminantes en utilisant la commande :

```
disc.forward <- greedy.wilks(Species ~.,data=iris,niveau=0.01)
```

Décrire à quoi correspond le champ `niveau` et les objets retournés par cette fonction. En particulier que permet d'obtenir `disc.forward$results[,1]` ?

4. La commande

```
res.lda <- lda(disc.forward$formula, data = iris)
```

permet d'exécuter la méthode LDA en utilisant que les variables sélectionnées. Adapter les lignes de commandes données à la Question 1 pour effectuer une sélection de variables. Comparer les résultats avec et sans sélection de variables.

5. Programmer l'étude de ré-échantillonnage. On reprendra les lignes de commande obtenues pour k-NN pour déterminer les indices des N échantillons d'apprentissage. Les données de sorties sont les N taux d'erreurs et les N précisions pour chaque modalité résultants des N découpages "échantillon d'apprentissage - échantillon test". On stockera également dans une liste de taille N les variables retenues pour chacun de ces découpages.

6. Tester votre programme sur le jeu entier des données `iris` avec $N = 500$. Tracer les boxplots associés aux taux d'erreurs et aux trois précisions (4 boxplots en tout). Conclure sur l'utilisation de la méthode LDA sur ce jeu de données.

7. Construire à partir de la liste contenant les variables retenues à chaque découpage, le tableau de contingence qui croise variable sélectionnée et sa position de sélection. Conclure sur la pertinence des 4 variables `Sepal.Length`, `Sepal.Width`, `Petal.Length` et `Petal.Width`.

3 Les Arbres

La réalisation d'arbre de décision avec `R` peut se faire grâce à l'utilisation de différents packages. Nous utiliserons ici les packages `rpart` et `rpart.plot`, ce dernier permettant un affichage plus lisible de l'arbre. La fonction `rpart()` permet de construire un arbre de décision avec la possibilité d'utiliser deux critères de segmentation : l'indice de Gini (Méthode CART) et l'entropie (Méthode C4.5). Ce choix doit être indiqué dans le paramètre `split` de l'argument `parms` de la fonction `rpart` : `parms=list(split='...')`. Si `split = 'information'`, la fonction utilise l'entropie si `split='gini'` (par défaut), elle utilise l'indice de Gini. Nous utiliserons ici l'indice de Gini.

3.1 Arbre avec les paramètres par défaut

On se propose de tester cette procédure avec comme échantillon d'apprentissage les 25 premières iris de chaque espèce et comme échantillon test celles qui restent. Exécuter les lignes de commandes suivantes :

```
X <- iris[,1:4]
y <- iris[,5]
ind.train <- c(1:25,51:75,101:125)
y.train <- y[ind.train]
X.train <- X[ind.train,]
dat.train <- data.frame(cbind(X.train,y.train=y.train))
arbre<-rpart(y.train~., data=dat.train)
```

Expliquer les différents éléments de la sortie `print(arbre)`. Exécuter la commande `rpart.plot(arbre)`. Commentez le graphique et les différentes quantités affichées.

3.2 Le pré-élagage

Les paramètres de pré-élagage sont dans l'argument `control=rpart.control( )`. Ils permettent de fixer les règles d'arrêt dans la construction de l'arbre :

- `maxdepth` : la profondeur de l'arbre
- `minsplit` : le nombre minimum d'individus présents à l'étape d'un nœud pour envisager une coupure
- `minbucket` : le nombre minimum d'individus présents à l'étape d'un nœud qu'engendrerait la coupure du nœud parent
- `cp` : paramètre qui traduit la complexité de l'arbre (plus cp prend une grande valeur, plus la complexité est pénalisée). Il peut éviter si besoin de déterminer des arbres trop profonds.

Par défaut, l'argument est paramétré de la façon suivante :

```
   control = rpart.control(maxdepth = 30, minsplit = 20,
minbucket = minsplit/ 3, cp = 0.01)
```

Pour obtenir l'arbre de taille maximum on choisit `cp = 0`. Construire l'arbre de taille maximal et le comparer à l'arbre précédant.

3.3 Le post-élagage

La phase d'élagage peut permettre de se ramener à un arbre de taille raisonnable mais pas nécessairement optimale. Tout au long de la construction de l'arbre, la

fonction `rpart()` calcule plusieurs quantités pour chaque arbre emboîté de la racine à celui qui a la complexité maximum demandée (par défaut `cp = 0.01`). Ces quantités sont stockées dans la table `arbre$cptable`. Dans cette table, `nsplit` donne le nombre de coupures de l'arbre emboîté correspondant, la colonne `rel error` renvoie l'erreur relative d'ajustement associée à ces arbres emboîtés ; il s'agit de l'erreur de classification calculée sur l'échantillon d'apprentissage normalisée par l'erreur obtenue à la racine. Cette erreur diminue avec la profondeur des arbres emboîtés : "moins on coupe, mieux on ajuste". On ne peut donc pas utiliser cette erreur pour trouver le `cp` "optimal". La fonction `rpart()` effectue alors une validation croisée 10-folds pour estimer le risque de chaque arbre emboîté. Le risque estimé moyen est donné dans la colonne `xerror` (toujours en erreur relative par rapport à la racine) ainsi que son écart-type empirique dans la colonne `xstd`. Ceux sont ces quantités qui vont permettre d'estimer la complexité optimale avec la règle du pouce qui consiste à choisir l'arbre le moins profond dont la complexité est inférieure à la valeur minimum de "`xerror+xstd`". Cela peut s'obtenir avec les lignes de commandes suivantes :

```
seuil <- min(arbre$cptable[,4]+arbre$cptable[,5])
cpadm <- arbre$cptable[arbre$cptable[,4]<=seuil,1]
cpopti <- cpadm[1]
```

La fonction `prune()` permet alors de procéder au post-élagage de l'arbre à partir d'une complexité donnée. Reprendre l'arbre de taille maximum obtenu à la section 3.2. A partir du champ `$cptable`, retrouver où se trouve l'arbre le plus élagué ? Quel est la valeur de `cp`? La commande `plotcp()` permet de tracer l'évolution des erreurs estimées en fonction de la taille de l'arbre. Quel arbre le moins complexe minimise cette erreur ? En déduire la valeur optimale de `cp`. Retrouver cette valeur par le calcul. Utiliser la fonction `prune()` pour obtenir l'arbre optimal. Le comparer avec les deux arbres précédents.

3.4 Prédire avec `rpart`

La commande `print(arbre)` fournit les règles de décision. Pour appliquer ces règles à un jeu de données, on utilise comme précédemment la fonction `predict()`. Utiliser cette fonction pour prédire les sorties de l'échantillon test pour les trois arbres ajustés précédemment. Commentez les résultats en terme d'indicateur de performance.

3.5 Score d'importance des prédicteurs

La visualisation de l'arbre est clairement une étape importante pour interpréter le jeu de données. Notamment elle permet d'identifier les variables concernées dans le découpage de chaque nœud. Elle ne permet néanmoins pas de hiérarchiser complétement les variables en fonction de leur importance. En effet les variables

en haut de l'arbre ne sont pas forcément plus importantes que les autres. Pour les arbres, il existe une mesure, appelée score d'importance, qui va noter chaque variable en fonction de son utilité dans la construction de l'arbre. Rappelons que chaque coupure dans l'arbre est sélectionnée en maximisant le gain d'impureté entre un nœud et ses deux nœuds fils. Plus le gain d'impureté est important, meilleure est la coupure. Le score d'importance est défini en sommant les gains d'impureté engendrées par chaque variable. Ce score est stocké dans le champs `$variable.importance`. Donner pour les trois arbres ces scores. Commentez les résultats obtenus.

3.6 Etude de ré-échantillonnage

Programmer l'étude de ré-échantillonnage (la calibration de l'arbre devra comprendre les étapes de pré et post-élagage). On reprendra les lignes de commande obtenues pour k-NN pour déterminer les indices des N échantillons d'apprentissage. Les données de sorties sont les N taux d'erreurs et les N précisions pour chaque modalité résultants des N découpages "échantillon d'apprentissage - échantillon test". On stockera également dans un vecteur de taille N les `cp` optimaux et dans une liste de taille N les scores d'importance retenues pour chacun de ces découpages.

Tester votre programme sur le jeu entier des données `iris` avec $N = 500$. Tracer les boxplots associés aux taux d'erreurs et aux trois précisions (4 boxplots en tout). Conclure sur l'utilisation des arbres sur ce jeu de données. Tracer l'histogramme associé aux `cp` optimaux. Conclure.

Déterminer les scores d'importance moyens et conclure sur la pertinence des 4 variables `Sepal.Length`, `Sepal. Width`, `Petal.Length` et `Petal.Width`. Comparer ces résultats avec ceux obtenus avec LDA.

4 Comparaison des trois approches

Tracer sur un même graphique les trois boxplots liés aux erreurs assoicées aux trois méthodes. Même chose pour les précisions. Selon vous quelle est la méthode la plus adaptée au jeu de données `iris` ?

5 Complément : l'importance des variables

En général on cherche le modèle qui prédit avec précision, mais on souhaite également en plus du pouvoir prédictif, interpréter le modèle. L'"importance des variables" est une mesure d'interprétation utile pour ce faire. La première façon pour calculer l'importance des variables et celle utilisée par les algorithmes de type arbres de

décision qui est calculée comme la diminution des impuretés (indice de Gini) comme cela a été décrit dans la partie "Score d'importance des prédicteurs".

La méthode "permutation importance" est une technique qui, elle, peut s'appliquer à n'importe quelle méthode de classification supervisée. Cette méthode consiste à mesurer la diminution ou l'augmentation d'un score (par exemple le taux d'erreur) du modèle après la permutation des valeurs d'un prédicteur X_j. En effet si le prédicteur est important on s'attend à une augmentation du taux d'erreur lorsqu'on "mélange" ses valeurs.

On se propose ici d'estimer l'importance pour chaque prédicteur comme suit. Dans l'étude de ré-échantillonnage, une fois le modèle calibré sur l'échantillon d'apprentissage et le calcul de l'erreur sur l'échantillon test effectué, pour chaque prédicteur X_j, on effectuera le calcul de l'erreur sur l'échantillon test après avoir permuté les valeurs de X_j. L'importance du prédicteur X_j sera estimé (pour ce tirage) par la différence entre les deux taux d'erreur aprés et avant permutation. Plus cette différence sera élevée plus le prédicteur devrait être important.

Reprendre les trois études de ré-échantillonnage en y insérrant les calculs des importances de chaque prédicteur. On pourra les stocker dans une matrice de taille $N \times p$. Afin de visualiser l'importance de chaque prédicteur, on tracera sur un même graphique (pour un algorithme donné) les p boxplots des importances de chaque prédicteur associé aux N tirages. Illustrer cela sur les données d'`iris`. Vos conclusions ? On comparera ces résultats avec ceux obtenus pour LDA et les arbres.