TP 4 : arbres de décision CART, bagging et forêts aléatoires

1. Dans cet exercice, on souhaiterait trouver une règle de décision pour classer des courriers électroniques comme "spam" ou "non spam". On dispose pour cela d'une base de données contenant 4 601 courriers électroniques, classés dans deux catégories, "spam" ou "non spam". Pour chacun de ces courriers, on dispose de 57 variables explicatives, la plupart (54 variables) correspondant à la fréquence de certains mots ou caractères dans le texte du courrier (ex. fréquence d'apparition du caractère \$, ou du mot "money"), les trois dernières étant liées aux nombre de lettres majuscules dans le mot (nombre moyen de lettres dans les mots écrits en majuscules, nombre de lettres dans le plus long mot écrit en majuscules, et nombre total de lettres majuscules dans le texte).

1. Importer la base de données `spam` depuis la librairie `kernlab`.
2. Diviser la base en une partie apprentissage et test, de tailles respectives 75% et 25% de la base de données totale.
3. En utilisant la fonction `rpart` du package du même nom, construire un arbre de décision binaire de type CART à partir de la base d'apprentissage. Comparer les résultats obtenus pour différents seuils du coefficient de coût de complexité (option `cp=`). On pourra tracer un arbre donné avec la fonction `rpart.plot`. À noter que la fonction `prune` permet de construire un arbre correspondant à une valeur de `cp` choisie.
4. À partir de la plus grande séquence d'arbres emboîtées possible (i.e. avec `cp=0`), identifier la valeur du coefficient de coût de complexité à partir de laquelle le taux d'erreur se stabilise et en déduire l'arbre optimal associé. On pourra également utiliser la fonction `plotcp` pour choisir la valeur de `cp`.
5. Comparer le taux de mauvais classement obtenu, avec ce ou ces arbre(s) sélectionné(s), sur la base de test.

2. On s'intéresse à présent à la protéase du VIH, une enzyme qui possède un rôle majeur dans le cycle du virus du SIDA. Cette enzyme permet en effet le clivage de certaines protéines, ce qui a pour effet de libérer et de créer d'autres protéines néfastes à l'organisme et d'aider la propagation de la maladie. Pour mieux comprendre comment fonctionne la protéase du VIH, on dispose d'une base de données contenant un ensemble d'octamères (i.e. une séquence de 8 acides aminés), dont certains ont été clivés par la protéase et d'autres non. L'objectif de l'étude est de prédire, à partir de sa séquence d'acides aminés, si un octamère est clivé ou non par la protéase du VIH.

Les données se trouvent dans la base "1625Data.csv". Cette base de données contient deux variables : la première variable `Octamer` correspond à la séquence d'acides aminés composant l'octamère et la deuxième variable `Clived` vaut 1 si l'octamère a été clivé par la protéase, et -1 sinon.

1. À partir de la séquence de 8 lettres, construire 20 variables qualitatives prenant 2 modalités permettant de savoir si chaque lettre de la séquence (G,P,A,V,L,I,M,C,F,Y,W,H,K,R,Q,N,E,D,S,T) est présente dans l'octamère.
2. Recoder la variable `Clived` en un facteur valant 0 ou 1.
3. Quelle est la proportion d'octamères clivés ?
4. Construire une base d'apprentissage et de test.
5. Construire le meilleur classifieur **constant** (i.e. ne dépendant pas des covariables) à partir de la base d'apprentissage. Calculer son taux d'erreur sur la base de test ce qui constituera un référentiel à battre avec les autres méthodes.

Arbres de décision CART

6. Construire un arbre de décision avec la fonction `rpart`, en gardant les réglages par défaut. Calculer le taux d'erreur sur la base de test.
7. Construire un arbre de décision profond, en modifiant les options `maxdepth`, `minbucket` et `cp`. Calculer le taux d'erreur de cet arbre.

8. Calculer un arbre de décision de niveau "minimal" c'est à dire avec un nombre minimal de nœuds (mais avec au moins 1 nœud). Calculer le taux d'erreur de cet arbre.

Bagging

L'objectif des méthodes de bagging est de diminuer l'instabilité d'un algorithme, en diminuant sa variance. Avec les arbres de décision, une petite variation dans les données peut conduire à un prédicteur très différent. Dans la suite, on décrit le principe du bagging.

À partir de l'échantillon d'apprentissage initial $(X_1, Y_1), \ldots, (X_n, Y_n)$, on tire avec remise B échantillons bootstrap de taille n, que l'on note $\mathcal{D}_b = \{(X_1^b, Y_1^b), \ldots, (X_n^b, Y_n^b)\}$. Sur chacun des échantillons bootstrap $\mathcal{D}_b$, on construit un classifieur. Dans le cas des arbres de décision, on construit l'arbre maximal $A_{\max}^b$, associé à sa règle de décision. Enfin, on définit le classifieur final en faisant appel aux B arbres à l'aide d'un vote majoritaire.

De façon générale, les méthodes de bagging permettent de construire un classifieur qui aura le même biais que les classifieurs qui le composent, mais une variance plus faible. Comme il n'est pas possible d'avoir un biais et une variance faible, il faut donc faire attention au choix des classifieurs. On aura ainsi intérêt à choisir des classifieurs avec un biais faible, pour obtenir un classifieur final de faible biais, mais la variance des classifieurs ne doit pas être trop élevée, sinon il faudra aggréger un grand nombre de classifieurs pour espérer diminuer la variance de façon significative.

9. Charger le package `adabag`. Construire un modèle de bagging à l'aide de la fonction `bagging` avec `mfinal`= 20 arbres, en laissant les autres réglages par défaut. Calculer le taux d'erreur associé.
10. Construire un modèle de bagging avec 20 arbres de décision à 1 nœud. Calculer le taux d'erreur.
11. Construire un modèle de bagging avec 20 arbres de décision profonds et calculer le taux d'erreur associé.
12. Comparer les taux d'erreur obtenus avec chaque méthode. En sélectionnant l'une des trois approches, tester l'effet du nombre d'arbres, en calculant le taux d'erreur moyen sur 10 répétitions, pour des valeurs de `mfinal` égales à 1, 2, 5, 10, 20 et 50. Commenter.

Forêts aléatoires

L'algorithme des forêts aléatoires est une méthode de bagging appliquée aux arbres CART (pour la régression ou pour la classification). L'idée principale est la même que celle du bagging présentée dans la section précédente, mais on ajoute une part d'aléatoire dans le processus, en ne sélectionnant pour chaque échantillon bootstrap, qu'un sous-ensemble de taille m des variables explicatives.

On commence par choisir un nombre d'échantillons bootstrap B et un nombre de variables m, puis pour $b = 1, \ldots, B$, on tire un échantillons bootstrap de taille $\mathcal{D}_b$ à partir de l'échantillon initial, puis on tire au hasard m variables explicatives parmi les p variables initiales et enfin on construit un arbre CART A_b sur l'échantillon $\mathcal{D}_b$ en utilisant les m variables choisies précédemment. La prédiction finale s'obtient en aggrégeant les arbres A_b à l'aide d'un vote majoritaire.

En pratique, comme dans le cas général du bagging, il faut faire un compromis biais-variance pour le choix de m. Lorsque m est petit, il y a moins de variables en compétition pour la division des nœuds et on se rapproche donc d'une procédure au hasard, basée uniquement sur le tirage des m variables parmi p. Seul le point en lequel on coupe la variable en deux sera basée sur l'échantillon. Les différents arbres construits sur les échantillons bootstrap seront donc peu corrélés, ce qui entraîne une baisse de la variance globale après aggrégation. Par contre, les classifieurs ainsi obtenus seront plus fortement biaisés, car moins bien ajustés au jeu de données. À l'inverse, si on augmente le nombre de variables m, on obtiendra des arbres bootstrap avec une variance globale après aggrégation plus élevée mais un plus faible biais. En pratique, on construit des arbres dont les nœuds terminaux sont de petite taille, ce qui implique un faible biais mais une plus forte variance, que l'on compense par une faible valeur de m.

13. Charger le package `randomForest`. Construire un modèle de forêt aléatoire avec `ntree`= 20 arbres profonds, chaque arbre étant construit avec `mtry` égal à $1, 2, 5, 8, 10, 12, 14, 16, 18$ variables. Tracer l'évolution du taux d'erreur moyen calculé sur 10 forêts aléatoires pour chaque valeur de `mtry`. En déduire une valeur de `mtry` optimale.
14. Tester l'effet du nombre d'arbres en traçant l'évolution du taux d'erreur moyen calculé sur 10 forêts aléatoires, chacune construite avec la valeur optimale de `mtry` obtenue à la question précédente, en fonction de `ntree`, avec `ntree` égal à $1, 2, 5, 10, 20, 50, 100, 200$ ou 500.