Université de Lille
M2 MFCA Data Sciences obouaziz@univ-lille.fr

TP. 4
CART et Forêts Aléatoires

Exercice 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 4601 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`, faire quelques statistiques descriptives et la diviser la base en une partie apprentissage et une partie test, de tailles respectives 75% et 25% de la base totale.
2. Construire un arbre de décision binaire de type CART, en utilisant la base d'apprentissage et en comparant les résultats obtenus pour différentes profondeurs différentes (option `max_depth`). On tracera pour chaque profondeur l'erreur de classification sur la base de données train et test. Commenter.
3. Quel est le taux de mauvais classement obtenu par le *meilleur* arbre ? Tracer la courbe ROC associée.
4. Comparer ce résultat avec le meilleur arbre obtenu par `GridSearchCV`.
5. Comparer ce classifieur avec celui obtenu par regression logistique.
6. Utiliser la fonction `RandomForestClassifier` pour construire un modèle de forêts aléatoires. Les deux paramètres à choisir sont B le nombre d'arbres dans la forêt (option `n_estimators`) et le nombre de variables parmi lesquelles sélectionner la meilleure division (option `max_features`). Faire varier ces paramètres et comparer les performances. Tracer la courbe ROC associée.
7. **Bagging** On va maintenant regarder la méthode de bagging fournie par la librairie `adabag`. Le bagging permet de réduire la variance d'un classifieur par aggrégation. Il est donc particulièrement adapté lorsque le classifieur de base a un faible biais et une forte variance. Nous allons tester cette propriété.
   (a) Construire un modèle de bagging à l'aide de la fonction `BaggingClassifier` avec 5 arbres puis 50. Comparer les taux d'erreurs associés.
   (b) Construire un modèle de bagging avec 50 arbres de décision à 1 nœud. Calculer le taux d'erreur.
   (c) Construire un modèle de bagging avec 50 arbres de décision profonds, et calculer le taux d'erreur associé.

(d) Comparer les taux d'erreur obtenus avec chaque méthode. En sélectionnant l'une des trois approches (justifier le choix effectué), 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.

8. **Boosting** On s'intéresse maintenant aux méthodes de boosting, qui contrairement aux méthodes de bagging, ne construisent pas des arbres de décision en parallèle, mais de façon séquentielle, chaque arbre se concentrant d'avantage sur les individus les moins bien classés par l'arbre précédent. Ces méthodes fonctionnent bien lorsque les classifieurs sont faibles (fort biais). Dans la version originelle de l'algorithme Adaboost, chaque observation est pondérée par un poids , et aucune source d'aléa n'est introduite contrairement aux méthodes de bagging. Les résultats de l'algorithme sont donc déterministes : si on lance plusieurs fois la méthode sur le même jeu de données, on obtient exactement les mêmes résultats.

   (a) Utiliser la fonction `AdaBoostClassifier` pour construire un modèle basé sur l'algorithme Adaboost. Calculer le taux d'erreur.

   (b) Construire un nouveau modèle utilisant Adaboost, mais basé sur des arbres de décision à 1 nœud. Faire de même en utilisant cette fois des arbres de décision profonds. Comparer les résultats obtenus.

   (c) Un algorithme populaire de boosting est XGBoost :
   `https://xgboost.readthedocs.io/en/stable/python/python_intro.html#scikit-learn-interface`
   Le mettre en œuvre sur les données et conclure.