Université de Lille 1 2018-2019
M2 ISN/MRA/FC - Machine Learning 24 octobre 2018

Machine Learning, Travaux Pratiques

Ce TP fera l'objet d'un rapport, pouvant être réalisé individuellement ou par groupe de deux, que vous devrez rendre le *20 décembre 2018 à minuit* au plus tard. Ce rapport, au format PDF, comprendra deux partie, la première répondant aux questions de la première partie de ce TP (en illustrant les réponses de graphiques), et la seconde présentant le travail que vous avez effectué sur de nouvelles données. Le code associé (notebook ou simplement code Python) devra être envoyé également.

Adresse pour l'envoi des documents : `emilie.kaufmann@univ-lille.fr`. Merci de nommer vos fichiers de manière pertinente (par exemple Rapport_NOM1_NOM2.pdf, Code_NOM1_NOM2.ipynb).

1 Mise en oeuvre des méthodes d'ensemble

L'objectif de cette première partie est de mettre en œuvre les méthodes d'ensemble vues en cours (Bagging et Boosting) sur la base de données `census.csv` issue d'un recensement, ainsi que l'approche par validation croisée pour sélectionner les paramètres de ces algorithmes. On pourra se servir d'éléments déjà donnés dans le notebook `ExampleML.ipnyb` disponible sur la page web du cours.

Validation croisée. On va d'abord explorer cette méthode de sélection de paramètre sur le cas des arbres de décision.

1. Préparation des données : effectuer le pre-processing nécessaire pour supprimer les valeurs manquantes, et remplacer les variables catégorielles par les "dummy variables" correspondantes. Séparer la base de données en une base d'apprentissage contenant 80% des données, et une base de test.
2. Pour ce jeu de données, quel est le meilleur classifieur constant ?
   Calculer son erreur de test, qui fournira un référentiel à battre.
3. Construire et visualiser (à l'aide de la fonction `graphviz` par exemple) un arbre de décision pas trop grand, calculer son erreur de test. En faisant varier les paramètres de construction de l'arbre (notamment `max_depth`) on obtient des résultats différents.
4. Implémenter une validation croisée pour sélectionner le ou les paramètres pertinent pour l'algorithme CART sur ce jeu de données. On pourra se servir de la fonction `GridSearchCV` du package `sklearn.model_selection`. Elle prend en entrée une méthode de classification et une grille de paramètres à tester, représentée par un dictionnaire. Par exemple si l'on veut tester les profondeurs 5,10,15 et deux critères d'impuretés, on peut utiliser

```
param_grid = {
    'max_depth': [5,10,15],
    'criterion': ["entropy","gini"]
}
```

   Déterminer ainsi le meilleur paramètre de profondeur dans une grille bien choisie. On pourra visualiser l'erreur de validation croisée en fonction du paramètre de profondeur de l'arbre. Evaluer le classifieur optimisé ainsi obtenu.

Bagging et Random Forest. La fonction `RandomForestClassifier` permet de constuire un classifieur par la méthode des forêts aléatoires. Les paramètres cruciaux de cette méthode sont le nombre d'arbres aggregés (`n_estimators`, B dans le cours) et le nombre de variables prises en compte lors de chaque séparation (`max_features`, p dans le cours).

4. Comment se servir de la fonction `RandomForestClassifier` pour faire du Bagging ? Implémenter le Bagging pour un nombre B d'arbres. Commenter la complexité et la performance de la méthode lorsque B augmente.
5. En choisissant une valeur pour p, construire maintenant un classifieur de type forêt aléatoire. Calculer son erreur Out-Of-Bag (qui vient en attribut du classifieur `clf` : `clf.oob_score_`), et la comparer à l'erreur de test.
6. En se fixant une valeur de B raisonnable, effectuer une validation croisée sur le paramètre p pour construire un classifeur Random Forest optimisé.

Boosting. On va expérimenter le Boosting à l'aide de la fonction `GradientBoostingClassifier`. Vérifier que vous disposez bien d'une version de scikit-learn supérieure à 0.20, car certaines fonctionnalités intéressantes ne sont disponibles qu'à partir de cette version.

7. A partir de la documentation en ligne, identifiez les paramètres cruciaux de l'algorithme Gradient Boosting. Quels jeux de paramètres correspondent à l'algorithme Adaboost ?
8. Le nombre nombre total d'arbres agregés B doit être sélectionné avec attention pour éviter le surapprentissage. Une implémentation de l'"early stopping" est possible dans scikit-learn à l'aide des paramètres `validation_fraction` et `n_iter_no_change`. A λ (taux d'apprentissage) et p (paramètre d'interaction) fixés, proposer un choix de B.
9. Sélectionner par l'approche de votre choix un "bon" algorithme de Gradient Boosting, qui sera votre classifieur Gradient Boosting optimisé. Le comparer à l'arbre de décision optimisé et au classifieur Random Forest optimisé obtenus précédemment.

Sélection de variables. Les méthodes d'aggrégation d'arbres perdent le caractère interprétable d'un seul arbre de décision, dont on peut observer les coupes successives pour interpréter une décision. Toutefois, il est possible de déduire de toute méthode de construction ou d'aggrégation d'arbres des "variables importantes" (dont la significativité statistique est moindre par rapport aux tests que l'on peut construire dans des modèles de régression).

10. Les trois méthodes de classification implémentées disposent d'un attribut calculant un score d'importance pour les variables explicatives, `clf.feature_importances_`. Essayer de comprendre comment ces scores sont calculés, puis repérer les variables jugées importantes par chacun de vos classifieurs optimisés.

Evaluation : courbe ROC. Les méthodes de classification qui produisent un *score* appartenant à $[0, 1]$ plutôt qu'une classe $\{0, 1\}$ peuvent être optimisées en choisissant un seuil potentiellement différent de $1/2$ pour convertir le score en prédiction. A chaque seuil possible est associé un taux de vrais positifs (TPR) et un taux de faux positifs (FPR). La qualité intrinsèque du score peut alors être évaluée via une courbe ROC, qui représente pour chaque seuil possible le TPR en fonction du FPR.

11. Comment prédire un score avec Random Forest ou Gradient Boosting ?

12. A l'aide de la fonction `sklearn.metrics.roc_curve` tracer sur le même graphique une courbe ROC pour les trois classifieurs optimisés obtenus précédemment.
13. Calculer l'AUC (aire sous la courbe ROC) pour ces trois classifieurs.

2 Expérimentation sur un nouveau jeu de données

L'objectif de cette partie et d'utiliser une méthode machine learning de votre choix pour effectuer une tâche de prédiction sur un jeu de données qui vous intéresse. Vous pouvez en trouver un vous même sur internet (par exemple Kaggle regorge de possibilités). Quelques suggestions ci-dessous :

— prédiction de la sévérité de sinistres pour l'assurance automobile. Il s'agit d'une tâche de **régression**.

— segmentation de marché dans le domaine de l'assurance : les données sont décrites ici, elles sont issues d'un challenge organisé par une compagnie d'assurance américaine. Il s'agit d'une tâche de **classification multi-classe**.

— détection de fraude à la carte de crédit : les données sont décrites ici. Il s'agit d'une tâche de **classification binaire où les deux classes sont déséquilibrées** (il y a beaucoup moins d'exemples de transactions frauduleuses que de transactions non-frauduleurses).

Les trois jeux de données décrits ci-dessus sont téléchargeables sur le site du cours, mais vous êtes libres d'en choisir un autre.

Une fois les données choisies, il y aura évidemment des problèmes spécifiques à résoudre (préparation des données, tâche d'apprentissage différente, classes déséquilibrées...). Il s'agira d'être débrouillards (une bonne recherche Google – si possible en anglais – vous aidera plus que de m'envoyer un mail dès que vous ne savez pas faire quelque chose avec Python) et de décrire soigneusement l'approche suivie dans votre rapport.