Apprentissage Statistique

TD 2 : Autour du Bagging
BUT 3

Guillaume Metzler
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France
guillaume.metzler@univ-lyon2.fr

Ce deuxième TD est de vous faire travailler sur une méthode ensembliste particulière qu'est le *bagging*.

On se propose d'étudier des algorithmes dans le cadre de ce TP, en travaillant plus spécifiquement sur avec des algorithmes comme les SVM ou encore les arbres de décision.

1 Introduction

Travailler avec des modèles simples ne permet pas de résoudre des tâches complexes, comme des problèmes de classification non linéaires par exemple.

Nous avons déjà pu mettre en oeuvre des algorithmes qui permettent de de résoudre des tâches complexes en utilisant des approches non linéaires, comme les méthodes à noyaux, *i.e.* SVM à noyau gaussien, pour résoudre de telles tâches (voir Figure 1).

On souhaite, à travers ce TD, savoir s'il existe une façon, à partir de modèles simples (comme les modèles linéaires) de construire des modèles beaucoup plus complexes et qui sont capables de résoudre des tâches plus complexes, comme des problèmes non linéairement séparables.

Pour faire cela, nous allons explorer des solutions du côté des *méthodes ensembllistes*.

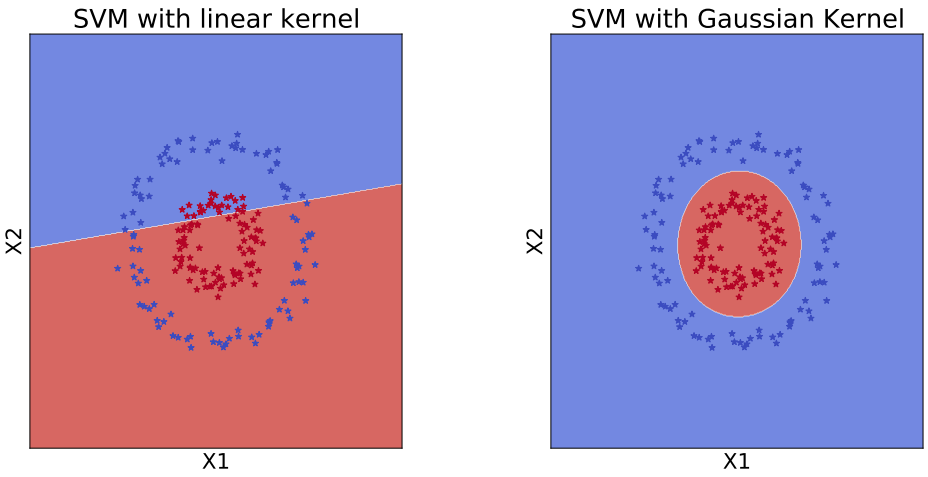

Figure 1: Illustration du pouvoir prédictif d'un modèle linéaire versus celui d'un modèle non linéaire

2 Autour du bagging

Les méthodes ensemblistes sont des méthodes qui consistent à combiner plusieurs modèles afin de pouvoir générer un nouveau modèle qui soit potentiellement plus *performant* mais aussi plus *robuste*. Cette combinaison s'effectue généralement de deux façons :

- via du *bagging*
- ou du *boosting*

On se concentrera uniquement sur le bagging dans le cadre de ce TP.

Présentation Un algorithme emblématique reposant du bagging est celui des forêts aléatoires. En quelques mots, les méthodes de bagging consistent à combiner des modèles qui sont appris à partir d'informations différentes, *i.e.* à partir d'échantillons différents. Ces modèles sont ensuite combinés, généralement sommés, afin de créer un seul et uniquement modèle performant.

On considère un échantillon d'apprentissage S et on effectue la procédure suivante T fois, où T représente le nombre de modèles que l'on souhaite apprendre : (i) on tire un échantillon S_t avec remise à partir de S, (ii) on apprend un modèle h_t en en utilisant l'échantillon S_t.

Une fois que les T modèles sont appris, on se retrouve avec un modèle global H_T que l'on écrira :

$$H_T = \frac{1}{T}\sum_{t=1}^{T} h_t.$$

L'exemple le plus simple est celui de la combinaisons d'arbres pour construire une *forêt aléatoire* [Breiman, 2001].

1. Que peut-on dire des échantillons S_t vis-à-vis de S ?
2. Quel est l'avantage de cette procédure d'apprentissage de modèle sur le plan algorithmique, si on prend en compte le fait que chaque modèle est indépendant des autres ?
3. Quel est l'avantage d'apprendre des modèles avec des informations différentes ?

Implémentation Vous pouvez apprendre un modèle ensembliste en utilisant une méthode de *Bagging* à partir en utilisant la librairie `sklearn.ensemble` et plus précisément la fonction `BaggingClassifier`.

Cette fonction dépend de plusieurs arguments

- `base_estimator` : choix du modèle de base à partir duquel on effectuera notre combinaison, on pourra choisir n'importe quel classifieur ou régresseur (voir exemple après)
- `n_estimators` : nombre de modèles que l'on souhaite apprendre, il faut donc spécifier un nombre entier
- `max_features` : on pourra spécifier quelle proportion des variables employer pour l'apprentissage d'un modèle
- `oob_score` : *True* : calcul ce que l'on appelle l'*out of bag error* qui permet d'avoir une idée des performances du modèle en généralisation.

1. Quel est l'intérêt de faire de l'échantillonnage sur les variables ?
2. A quoi pourrait bien servir, sur le plan pratique, le paramètre `oob_score` dans la fonction `BaggingClassifier` de Python.

Exemples On donne ici un exemple qui permet d'apprendre un classifieur ensembliste, fondé sur la bagging, à partir de SVM linéaires[1].

[1] Cet exemple est directement extrait de l'exemple présenté à l'adresse suivante : https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html.

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=100, n_features=4, n_informative=2,
    n_redundant=0,random_state=0, shuffle=False)
clf = BaggingClassifier(base_estimator=SVC(), n_estimators=10, random_state=0).fit(X, y)
```

Un autre exemple où l'on effectue cette fois-ci une combinaison d'arbres de décisions *DecisionTreeClassifier* (ce qui correspond au paramètre par défaut de `base_estimator`).

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=100, n_features=4, n_informative=2,
    n_redundant=0,random_state=0, shuffle=False)
clf = BaggingClassifier(n_estimators=10, random_state=0).fit(X, y)
```

Mise en pratique

1. En considérant un jeu de données de votre choix et en prenant un classifieur de base qui soit un SVM et/ou arbre de décision, représenter votre d'erreur en entraînement et en test en fonction du paramètre `n_estimators` de la méthode de Bagging. Décrivez vos observations.

3 Quelques exercices

Exercice 1 : Arbres de décisions et forêts aléatoires

On repart du jeu de données présenté dans l'exercice précédent que l'on rappelle ci-dessous:

y	-1	1	-1	-1	1	1	1	-1
x_1	-2	3	-1	1	-4	-2	3	2
x_2	0	2	1	-3	-1	-2	4	-3

On s'intéresse ici à une tâche de classification binaire, on cherche donc à prédire la valeur de $y \in \{-1, 1\}$ en fonction des caractéristiques $\mathbf{x}$.

1. Evaluer la valeur de l'indice de Gini à la racine de l'arbre.
2. Evaluer la valeur de l'entropie à la racine de l'arbre.

On souhaite maintenant étudier deux modèles induit par deux splits différents : (i) pour le premier modèle, le premier split se fait avec la règle $x_1 < 2.5$; (ii) pour le deuxième modèle le split se fait à l'aide du critère $x_2 \geq 0$.

3. Pour les différents splits :

   (a) représenter les arbres obtenus : on indiquera quels sont les exemples que l'on trouvera dans chaque feuille

   (b) évaluer la valeur de l'indice de Gini dans les différentes feuilles

   (c) déterminer les performances de classification de ces deux arbres

   (d) calculer le gain de gini pour les deux splits. Quel est le split réellement effectué par l'arbre lors de l'apprentissage ?

   On s'intéresse maintenant à la combinaison de ces deux arbres (celui obtenu avec la règle (i) et celui obtenu avec la règle (ii) pour former une forêt aléatoire.

4. Prédire l'étiquette des données suivantes

$$\mathbf{x}'_1 = (2 \quad 3),\ vx'_2 = (-4 \quad 6),\ \text{et}\ \mathbf{x}'_3 = (2 \quad 6).$$

   En affectant un poids de 0.6 au premier arbre et de 0.4 au deuxième arbre.

Exercice 2 : Une autre forêt aléatoire

Quelques questions de cours

1. En utilisant vos propres mots, expliquez la signification du terme présenté ci-dessous ainsi que le rôle de ce dernier dans un contexte de *bagging*.

$$\frac{1}{T} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X},y\sim\mathcal{Y}} \left[\sum_{t=1}^{T} (h_t(\mathbf{x}) - y)^2 \right] \geq \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X},y\sim\mathcal{Y}} \left[(y - H_T(\mathbf{x}))^2 \right],$$

   où $H_T = \sum_{t=1}^{T} h_t$, $y \in \mathbb{R}$ représente la valeur prédite par la combinaison de modèles, $\mathbf{x} \in \mathbb{R}^d$ désigne un individu et h_t sont les apprenants.

2. Expliquer le principe du double échantillonnage qui est couramment utilisé dans les modèles de forêts aléatoires. On précisera notamment l'intérêt de ce dernier sur les plans théorique et pratique.

3. Expliquer comment est effectué la validation des modèles de forêts aléatoires. On prendra soin de préciser quel est, en moyenne, le pourcentage des exemples utilisés pour la validation de ces derniers.

Construction d'une forêt aléatoire

On considère le jeu de données de classification suivant :

y	-1	-1	-1	-1	-1	-1	+1	+1	+1	+1
x_1	-3	-4	2	4	-1	0	1	6	5	8
x_2	-4	-2	1	3	4	7	-6	-3	-5	9

Notre objectif est de construire une forêt constituée de deux arbres ayant chacun une profondeur de 1, *i.e.*, une seule séparation (ou split) sera effectué.

A. Construction d'un premier arbre Pour ce premier arbre, seule la **première** variable est utilisée (x_1). De plus, on considérera uniquement les exemples $\mathbf{x}_i$ pour $i \in [\![2, 9]\!]$ pour entraîner notre arbre et les autres exemples serviront à la validation.

1. Evaluer l'indice de Gini à la racine de cet arbre.
2. Déterminer la split optimal et évaluer le gain de Gini (on pourra s'aider d'un dessin pour trouver le split optimal)
3. Déterminer les performances, en *accuracy*, de ce modèle h_1, sur l'ensemble d'apprentissage et sur l'ensemble de validation.

B. Construction d'un deuxième arbre Pour ce premier arbre, seule la **deuxième** variable est utilisée (x_2). De plus, on considérera uniquement les exemples $\mathbf{x}_i$ pour $i \in [\![1, 8]\!]$ pour entraîner notre arbre et les autres exemples serviront à la validation.

1. Evaluer l'indice de Gini à la racine de cet arbre.
2. Déterminer la split optimal et évaluer le gain de Gini (on pourra s'aider d'un dessin pour trouver le split optimal)
3. Déterminer les performances, en *accuracy*, de ce modèle h_2, sur l'ensemble d'apprentissage et sur l'ensemble de validation.

C. Performance de la forêt On considère la forêt aléatoire défini par $H_T = \frac{1}{2}(h_1 + h_2)$ et le jeu de données suivant :

y	-1	-1	+1	+1
x_1	-1	2	3	4
x_2	-5	6	-1	-2

Evaluer les performances, en terme d'accuracy, de la forêt aléatoire sur ce jeu de données.

References

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.