Apprentissage Statistique
TD 3 : Autour du Boosting
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 troisième et dernier TD est l'occasion de travailler sur une dernière méthode ensembliste qu'est le *boosting*.

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

Si précédemment les modèles ont été appris les uns indépendamment des autres, l'idée du boosting est de rompre cette indépendance.

On suppose que les modèles appris ont un faible pouvoir prédictif et on souhaite créer une combinaison de ces modèles afin que cette dernière soit plus performante. Pour faire cela, on va apprendre nos modèles de façon itérative et cela, de sorte que le modèle appris à l'itération actuelle soit capable de corriger les erreurs effectuées par le modèle précédent.

2 Adaboost

Le premier algorithme de boosting qui a été développé et qui répond à cette problématique est l'algorithme *Adaboost* [Freund et al., 1999].

La présentation de cette procédure est donnée par l'Algorithme 1 et se présente comme suit.

Principe du boosting On dispose initialement de notre échantillon S avec nos m exemples $(\mathbf{x}_i, y_i)$ et toutes les données ont **le même poids**, *i.e.* la même importance.

On va maintenant regarder comment une hypothèse h_{t+1} est apprise en fonction des performances du classifieur h_t.
Pour cela, plaçons nous à une étape t de notre algorithme où les exemples ont dont un poids égal à $w_i^{(t)}$. Une hypothèse h_t est alors apprise et nous pouvons évaluer son taux d'erreur en classification ε_t

$$\varepsilon_t = \sum_{i=1}^{m} w_i^{(t)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$$

A partir de cette erreur, nous allons déterminer une quantité α_t définie par :

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

et qui va permettre de quantifier l'importance de l'hypothèse h_t dans la décision finale, *i.e.* on va définir un poids sur le classifieur appris.

Le reste de la procédure consiste ensuite à trouver **une bonne repondération des exemples** de façon à ce que l'hypothèse qui sera apprise à l'itération suivante, puis se focaliser sur les erreurs commises par l'hypothèse actuelle, cela se fait par la mise à jour suivante :

$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t},$$

où Z_t est un facteur de normalisation permettant d'avoir une distribution sur les poids des exemples. Nous verrons plus tard que ce facteur de normalisation est donné par $Z_t = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}$. La fonction de repondération des exemples, va augmenter le poids des exemples mal classés et diminuer celui des exemples bien classés.

Algorithm 1: Adaboost algorithm [Freund et al., 1999]

Input: Echantillon d'apprentissage S de taille m,
un nombre T de modèles
Output: Un modèle $H_T = \sum_{t=0}^{T} \alpha_t h_t$
begin
  Distribution uniforme $w_i^{(0)} = \dfrac{1}{m}$
  for $t = 1, ..., T$ **do**
    Apprendre un classifieur h_t à partir d'un algorithme $\mathcal{A}$
    Calculer l'erreur ε_t de l'algorithme.
    if $\varepsilon_t > 1/2$ **then**
      Stop
    else
      Calculer $\alpha_{(t)} = \dfrac{1}{2} \ln \left(\dfrac{1 - \varepsilon_t}{\varepsilon_t} \right)$
      $w_i^{(t)} = w_i^{(t-1)} \dfrac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$
  Poser $H_T = \sum_{t=0}^{T} \alpha_t h_t$
  return H_T

1. Expliquer le lien entre les valeurs de α_t et les performances de l'algorithme.
2. Expliquer le fonctionnement de la repondération des exemples.
3. A quelle loss classique vous fait penser la loss qui est utilisée lors de la repondération des erreurs.

L'algorithme Adaboost peut illustrer graphiquement cette algorithme comme présenté en Figure 1 et vérifie la propriété suivante :

Proposition 2.1: Borne Erreur sur Adaboost

L'erreur empirique du classifieur obtenue à l'aide de Adaboost est bornée par

$$\mathcal{R}_S(H_T) \leq \exp\left[-2 \sum_{t=1}^{T} \left(\frac{1}{2} - \varepsilon \right)^2 \right].$$

De plus, si pour tout $t \in [\![1, T]\!]$, $\gamma \leq \left(\dfrac{1}{2} - \varepsilon_t \right)$, alors :

$$\mathcal{R}_S(H_T) \leq \exp(-2\gamma^2 T).$$

Cette proposition, et plus précisément la démonstration de cette dernière permet d'expliquer la pondération des différents exemples au cours des itérations, *i.e.* les $w_i^{(t)}$

mais aussi la façon dont est calculée l'importance d'un classifieur α_t pour ces mêmes itérations.

1. En regardant de plus près le résultat de cette proposition, que devrions nous observer, sur le risque, dans le cas où l'on apprend une infinité de modèles avec Adaboost ?
2. Est-ce faisable en pratique ? Qu'est-ce qui pourrait empêcher cela ?

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

Cette fonction dépend de plusieurs arguments, mais on se contentera uniquement des présentés ci-dessous :

- `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

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

```
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
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 = AdaBoostClassifier(estimator=SVC(), n_estimators=10, random_state=0, algorithm =
↪  'SAMME').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 AdaBoostClassifier
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 = AdaBoostClassifier(n_estimators=10, random_state=0).fit(X, y)
```

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

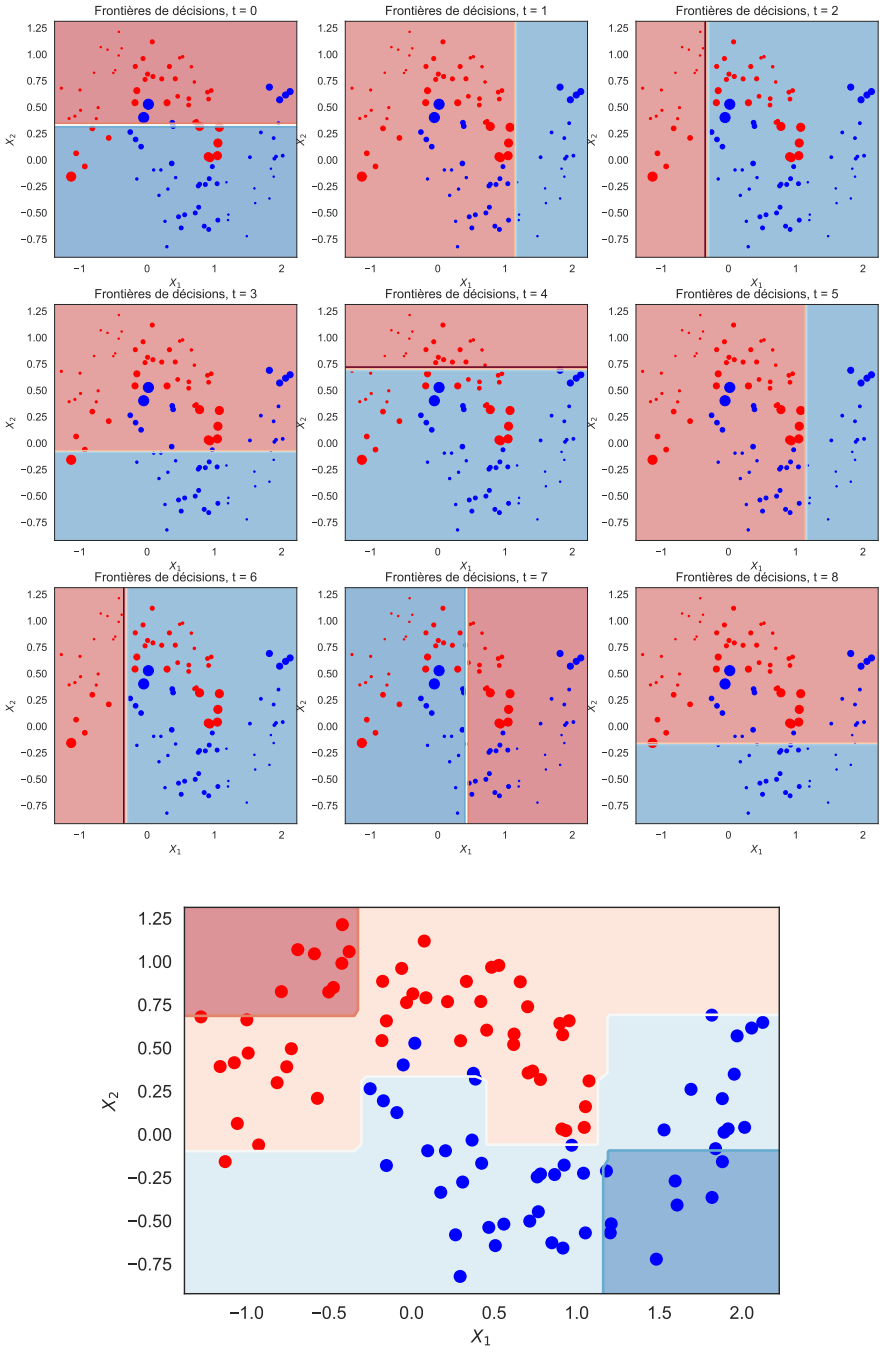

Figure 1: Illustration des différentes itérations de l'algorithme Adaboost lorsque l'on apprend un ensemble de 9 hypothèses

Mise en pratique

1. En considérant un jeu de données de votre choix (ou celui qui est donné ci-dessous) 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 Boosting. Décrivez vos observations.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=400, noise=0.2)
```

2. Recoder vous même l'algorithme Adaboost en vous basant sur la présentation donnée en Algorithme 1.

3 Gradient Boosting et XGBoost

Utilisez les lignes de code suivante pour votre première implémentation de l'algorithme XGBoost et tester ensuite ses performances sur les différents jeux de données à votre disposition.

Un exemple sur un jeu de classification

```
import xgboost
from xgboost import XGBClassifier
from sklearn.datasets import make_classification

# Création d'un jeu de données
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
↪  n_redundant=5, random_state=7)
# Information sur ce dernier
print(X.shape, y.shape)

# Définition du modèle
model = XGBClassifier()
```

Un exemple pour la régression

```
from sklearn.datasets import make_regression
from xgboost import XGBRegressor

# Création d'un jeu de données
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1,
↪  random_state=7)

# Définition du modèle
model = XGBRegressor()
```

Vous pourrez ensuite vous renseigner sur les différents paramètres de la fonction et chercher à les optimiser par cross-validation.

4 Exercices

Exercice 1

Cet exercice se concentre sur l'étude du boosting. On va considérer que l'on dispose du jeu d'entraînement S suivant

Individu	x_1	x_2	y
1	-2	1	1
2	1	-1	1
3	3	4	1
4	0	-3	-1
5	1	-2	-1
6	-1	-4	-1

1. Décrire les différentes étapes de l'algorithme Adaboost ci-dessous :

Input: Echantillon d'apprentissage S de taille m,
un nombre T de modèles
Output: Un modèle $H_T = \sum_{t=0}^{T} \alpha_t h_t$
begin
  Distribution uniforme $w_i^{(0)} = \dfrac{1}{m}$
  for $t = 1, ..., T$ **do**
    Apprendre un classifieur h_t à partir d'un algorithme $\mathcal{A}$
    Calculer l'erreur ε_t de l'algorithme.
    if $\varepsilon_t > 1/2$ **then**
      Stop
    else
      Calculer $\alpha_{(t)} = \dfrac{1}{2} \ln\left(\dfrac{1-\varepsilon_t}{\varepsilon_t}\right)$
      $w_i^{(t)} = w_i^{(t-1)} \dfrac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$
  Poser $H_T = \sum_{t=0}^{T} \alpha_t h_t$
  return H_T

---

2. Quelle est loss que l'on cherche à minimiser avec l'algorithme adaboost ?

   On considère le modèle suivant :

$$H(\mathbf{x}) = sign(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x})),$$

où $(\alpha_1, \alpha_2, \alpha_3) = (1, 2, 3)$ et $h_1(\mathbf{x}) = sign(2), h_2(\mathbf{x}) = sign(2x_1)$ et $h_3(\mathbf{x}) = sign(-x_1 + x_2 - 1)$.

(a) En repartant de la description de l'algorithme Adaboost, quel est le *weak learner* qui a l'erreur la plus faible ?

(b) Prédire l'étiquette des données $\mathbf{x}_1, \mathbf{x}_2$ et $\mathbf{x}_3$ de l'ensemble S défini à l'exercice précédent.

Exercice 2

Considérons que l'on mette en place l'algorithme Adaboost avec un séparateur linéaire de la forme $h(\mathbf{x}) = \langle \boldsymbol{\theta}, \mathbf{x} \rangle + b$ de sorte à ce que ce dernier renvoie une valeur dans l'ensemble $\{-1, 1\}$. Les hypothèses sont apprises sur le jeu de données suivant

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

Au cours de la première itération de notre algorithme, les paramètres du modèle sont donnés par $\boldsymbol{\theta} = (1, 1)$ et $b = -1$ et chaque exemple a un poids égal à $\frac{1}{m}$ où m désigne le nombre d'exemples.

1. Evaluer l'erreur global de votre modèle
2. En déduire le poids du modèle nouvellement appris
3. Déterminer la pondération des exemples pour la prochaine itération de l'algorithme Adaboost.

On considère le modèle suivant :

$$H(\mathbf{x}) = \text{sign}(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x})),$$

où $(\alpha_1, \alpha_2, \alpha_3) = (2, 3, 1.5)$ et $h_1(\mathbf{x}) = \text{sign}(3x_1 + 2), h_2(\mathbf{x}) = \text{sign}(2x_1 + 3x_2 - 4)$ et $h_3(\mathbf{x}) = \text{sign}(-x_1 + 2x_2 - 1)$.

4. En repartant de la description de l'algorithme Adaboost, quel est le *weak learner* qui a l'erreur la plus faible ?
5. On considère les individus

$$\mathbf{x}'_1 = (1, 0),\ \mathbf{x}'_2 = (2, 3) \quad \text{et} \quad \mathbf{x}'_3 = (-3, 2)$$

Prédire l'étiquette de ces différents individus par votre méthode ensembliste.

References

[Freund et al., 1999] Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.