TELECOM ParisTech **MDI343**
2016/2017 **Alexandre Gramfort, Joseph Salmon**

TP noté n° 2 : Bagging, Boosting, Random Forests

Pour ce travail vous devez déposer un **unique** fichier au format `nom_prenon.ipynb` sur le site pédagogique du cours (partie Rendus TP Bagging/Boosting).

Vous devez charger votre fichier sur Éole, avant le dimanche 22/01/2017 23h59.

La note totale est sur **20** points répartis comme suit :

— qualité des réponses aux questions : **15** pts,

— qualité de rédaction, de présentation et d'orthographe : **2** pts,

— indentation, style PEP8, commentaires adaptés, etc. : **2** pts,

— absence de bug : **1** pt.

Les personnes qui n'auront pas rendu leur devoir avant la limite obtiendront **zéro**.

Rappel : les exercices sont indépendants et aucun travail par mail ne sera accepté !

- Agrégation de modèles -

On considère un problème d'apprentissage supervisé standard. Étant donné un ensemble de points d'apprentissage $\mathcal{D} = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$, un estimateur/classifieur est une fonction $\hat{f}_{\mathcal{D}}$. Si les Y_i prennent leurs valeurs dans $\{1, \dots, K\}$ on parle de problème de classification multi-classe (avec K classes) et si les Y_i prennent leurs valeurs dans $\mathbb{R}$ on parle de problème de régression.

Une agrégation de modèles (classifieurs/estimateurs) consiste à combiner (linéairement) les prédictions individuelles de chaque modèle élémentaire. En régression, on s'intéresse au modèle $\hat{F}^L_{\mathcal{D}}$ obtenu par agrégation de L estimateurs $\hat{f}^l_{\mathcal{D}}, l = 1, \cdots, L$:

$$\hat{F}^L_{\mathcal{D}} = \sum_{l=1}^{L} w_l \hat{f}^l_{\mathcal{D}}$$

où les $w_l \geq 0$ sont les poids.

Pour la classification, l'agrégation peut se faire avec une procédure de vote (par exemple majoritaire), ou bien en moyennant la probabilité des classes. Si la prédiction d'un classifieur binaire $\hat{f}^l_{\mathcal{D}}$ en X correspond à $\text{sign}(\hat{f}^l_{\mathcal{D}}(X))$, alors le modèle agrégé peut prédire également en utilisant $\text{sign}(\sum_{l=1}^{L} w_l \hat{f}^l_{\mathcal{D}}(X))$.

- Bagging -

Le *Bagging* (acronyme venant de "Boostrap Aggregation") [Bre96] est une méthode classique pour combiner les modèles. Elle consiste à prendre une simple moyenne des prédictions, *i.e.*, $w_l = 1/L$. Afin de générer plusieurs estimateurs, on utilise plusieurs jeux de données générés aléatoirement en utilisant le *bootstrap*. Un échantillon *bootstrap* est un échantillon de n points d'apprentissage obtenus à partir de $\mathcal{D}$ par tirage aléatoire uniforme (avec remise).

1) Mettez en œuvre le Bagging avec des arbres de profondeur 1 (en Anglais *stumps*), puis avec des arbres plus profonds, en partant du code ci-dessous. On pourra utiliser `BaggingRegressor`.

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
import matplotlib.pyplot as plt
# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
```

```
y = np.sin(X).ravel()
y[::5] += 1 * (0.5 - rng.rand(16))

n_estimators = 10 # L in the text
tree_max_depth = 10
bagging_max_depth = 10
# TODO define the regressor by bagging stumps
# tree = ...
tree.fit(X, y)
# bagging = BaggingRegressor(...)
bagging.fit(X, y)
# Predict
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_tree = tree.predict(X_test)
y_bagging = bagging.predict(X_test)
# Plot the results
plt.figure(figsize=(12, 8))
plt.plot(X, y, 'o', c="k", label="data")
# TODO add plots for Bagging/Tree
plt.title("Decision Tree Regression")
plt.legend(loc=1, numpoints=1)
plt.show()
```

2) Illustrer graphiquement le rôle de L ainsi que de la profondeur des arbres (`max_depth`).
3) A quoi reconnaît-on que les estimateurs construits par les arbres sont biaisés et que le *bagging* réduit leur variance ?
4) En jouant sur le niveau de bruit mettez en évidence le sur-apprentissage.
5) Observer qu'on peut réduire ce phénomène en sous-échantillonnant aléatoirement (sans remise) au lieu de prendre des échantillons *bootstrap*.

Random Forests

Les forêts aléatoires (en : *Random Forests*) [Bre01], combinent l'idée du *Bagging*, l'échantillonnage par *bootstrap* et moyennage, avec une sélection aléatoire des variables à chaque nœud de la construction de l'arbre. Dans le cas de la classification, l'agrégation se fait par vote majoritaire.

6) Évaluez le score par 7-fold cross-validation des *Random Forests* sur les datasets `boston`, `diabetes`, `iris` et `digits`. Comparez ces performances avec celles d'un SVM linéaire. On pourra utiliser :

```
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
```

Les *Random Forests*, tout comme le *Bagging*, peuvent être utilisées pour prédire une probabilité. Pour ce faire la probabilité d'être dans la classe k est la proportion des arbres qui prédisent la classe k.

7) En utilisant le dataset `iris` restreint aux deux premières variables explicatives afficher la probabilité de prédiction des classes. On partira du script suivant et on fera varier le nombre d'arbres aléatoires (variable `n_estimators`).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
# Parameters
n_estimators = 2
plot_colors = "bry"
plot_step = 0.02
```

```
# Load data
iris = load_iris()
X_unscaled, y = iris.data[:, :2], iris.target
# Standardize
X = preprocessing.scale(X_unscaled)
# RF fitting
model = RandomForestClassifier(n_estimators=n_estimators)
clf = model.fit(X, y)
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

plt.figure()
for tree in model.estimators_:
    # TODO use predict to obtain the probabilities you will store in Z
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, alpha=1. / n_estimators, cmap=plt.cm.Paired)
plt.axis("tight")
# Plot the training points
for i, c in zip(range(3), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=c, label=iris.target_names[i],
                cmap=plt.cm.Paired)
plt.legend(scatterpoints=1)
plt.show()
```

8) Comparez les scores par 6-fold cross-validation des *Random Forests* et des arbres de décisions pures (obtenus avec `DecisionTreeClassifier`), sur le dataset `iris` restreint aux deux premières variables explicatives. On fera varier le paramètre `max_depth` entre 1 et 30. Mettre en évidence le fait que les *Random Forests* permettent de réduire le sur-apprentissage, et ce même pour des arbres profonds.

Boosting

D'un point de vue historique, l'un des premiers algorithmes de *Boosting* à rencontrer un réel succès s'appelle "l'AdaBoost.M1" et a été proposé par Freund et Schapire [FS97]. Quelques informations supplémentaires sont disponibles dans [Fri01, FHRT00], [HTF09, Chapitre 10] [1].

- Théorie -

On se place dans le cadre de la classification à deux classes : les étiquettes ont deux valeurs possibles : -1 et 1. On cherche une 'bonne' fonction de régression $\hat{f} : x \mapsto \hat{f}(x) \in \mathbb{R}$, et on prendra comme classifieur associé $\hat{h}(x) = \text{sign}(\hat{f}(x))$. Rappelons que l'on note η la fonction de régression $\eta(x) \triangleq \mathbb{P}(Y = 1|X = x)$. Le coût de référence est le coût 0/1 qui s'écrit

$$\text{perte}(x, y, f) = \mathbb{1}_{\{-y\, f(x) \geq 0\}} = \varphi_0(-y\, f(x)),$$

où φ_0 est la fonction indicatrice $\varphi_0 = \mathbb{1}_{\mathbb{R}_+}$. On rappelle que le classifieur de Bayes associé au problème est $h^*_{\varphi_0} = \text{sign}(2\eta - 1)$. Par définition, le classifieur de Bayes minimise le risque de classification

$$R_{\varphi_0}(h) = \mathbb{P}(Y \neq h(X)) = \mathbb{P}(-Yf(X) \geq 0) = \mathbb{E}(\varphi_0\left[-Yf(X)\right]).$$

On notera selon le contexte $R_\varphi(f)$ ou $R_\varphi(h)$ pour désigner le même risque. Ainsi, $h^*_{\varphi_0} = \arg\min_h R_{\varphi_0}(h) = \text{sign}(2\eta - 1)$ où le min porte sur toute les fonction mesurables, ce qui s'écrit

$$h^*_{\varphi_0} = \underset{h:\mathbb{R}^d \mapsto \{-1,1\}}{\arg\min}\ R_{\varphi_0}(h), \text{ où } R_{\varphi_0}(h) = \mathbb{E}(\varphi_0\left[-Yf(X)\right]).$$

1. ainsi que sur la page de J. Friedman http://www-stat.stanford.edu/~jhf/R-MART.html

En pratique on n'a pas accès à la distribution inconnue de la loi jointe des observations. On cherche donc à optimiser la contrepartie empirique du risque. Le minimiseur du risque empirique est alors

$$\hat{f}_{n,\varphi_0} = \arg\min_f R_{n,\varphi_0}(f), \text{ où } R_{n,\varphi_0}(f) = \mathbb{E}_n(\varphi_0\left[-Yf(X)\right]) \triangleq \frac{1}{n}\sum_{i=1}^{n}\varphi_0(-Y_i f(X_i))$$

et le classifieur associé est donc $\hat{h}_{n,\varphi_0} = \text{sign}(\hat{f}_{n,\varphi_0})$.

Le principal problème de la fonction φ_0 est qu'elle n'est pas convexe, ce qui rend difficile son optimisation. On utilisera donc des "substituts convexes", c'est-à-dire des fonctions φ bien choisies, convexes, et "proches" de φ_0.

9) Démontrez la propriété suivante : Le minimiseur de la fonction $f \to R_{\exp}(f) = \mathbb{E}(\exp(-Yf(x)))$ est atteint en $f^*_{\exp} = \frac{1}{2}\log(\frac{\eta(x)}{1-\eta(x)})$ [FHRT00, p. 215].

10) En déduire que le classifieur de Bayes associé au risque $R_{\exp}$ est le même que le classifieur de Bayes associé au risque 0/1, R_{φ_0}.

- AdaBoost -

Cette méthode consiste à chercher les solutions d'un problème d'optimisation où l'on restreint les choix possibles de candidats. Supposons pour commencer que l'on ait M classifieurs experts disponibles, ou de manière équivalente que l'on dispose de $f_1, \ldots, f_M$. Un objectif naturel est alors de chercher la meilleure combinaison de classifieurs possible, donc de résoudre le programme suivant :

$$\hat{f}^M_{n,\varphi} = \underset{f\in\text{Conv}(f_1,\ldots,f_M)}{\arg\min} R_{n,\varphi}(f), \text{ où } \quad \varphi_0 \leq \varphi \text{ et } \varphi \text{ est convexe.}$$

avec la notation : $\text{Conv}(f_1,\ldots,f_M) = \{f : \exists(\alpha_1,\ldots,\alpha_M) \in \mathbb{R}_+, \sum_{j=1}^M \alpha_j = 1, \text{ t.q. } f = \sum_{j=1}^M \alpha_j f_j\}$.

L'algorithme AdaBoost repose sur ce principe (à ceci près que les classifieurs experts sont eux aussi mis à jour au cours de l'algorithme). Il vise à minimiser une version convexifiée $R_{n,\exp}$ du risque empirique correspondant à la fonction de perte exponentielle.

Pour $w \in \mathbb{R}^n$, on utilisera la notation $\mathbb{P}_{n,w}$ pour noter la probabilité discrète à poids w : $\mathbb{P}_{n,w} = \frac{1}{n}\sum_{i=1}^n w_i\delta_{X_i}$, et $\mathbb{E}_{n,w}$ l'espérance associée, par exemple $\mathbb{E}_{n,w}(f(X)) = \frac{1}{n}\sum_{i=1}^n w_i f(X_i)$.

Algorithme : AdaBoost

Données : les observations et leurs étiquettes $\mathcal{D}_n = \{(X_i, Y_i) : 1 \leq i \leq n\}$, le nombre d'étapes M, un vecteur poids $w^0 \in \mathbb{R}^n$ (généralement on choisit $w_i^0 = \frac{1}{n}$ pour tout $i \in [\![1, n]\!]$)

Résultat : un classifieur $\hat{h}^M_{\text{boost}}$

pour $m = 1, \ldots, M$ **faire**

- Ajuster un classifieur $\hat{h}_m$ avec une distribution des observations suivant le vecteur de poids $w^{m-1} = (w_1^{m-1}, \ldots, w_n^{m-1})$
- Calculer :

$$\mathbb{P}_{w^{m-1}}(Y \neq \hat{h}_m(X)) = \sum_{i=1}^n w_i^{m-1}\mathbb{1}_{\{Y_i\neq\hat{h}_m(X_i)\}}$$

$$\mathbb{P}_{w^{m-1}}(Y = \hat{h}_m(X)) = \sum_{i=1}^n w_i^{m-1}\mathbb{1}_{\{Y_i=\hat{h}_m(X_i)\}}$$

$$c_m = \frac{1}{2}\log\left[\frac{\mathbb{P}_{w^{m-1}}(Y = \hat{h}_m(X))}{\mathbb{P}_{w^{m-1}}(Y \neq \hat{h}_m(X))}\right]$$

- Mettre à jour les poids : $\begin{cases} w_i^{int} &= w_i^{m-1}\exp\left(2\cdot c_m\cdot\mathbb{1}_{\{Y_i\neq\hat{h}_m(X_i)\}}\right) \\ w_i^m &= \frac{w_i^{int}}{\sum_{j=1}^n w_j^{int}} \end{cases}$

$\hat{h}^M_{\text{boost}} = \text{sign}(\sum_{m=1}^M c_m\hat{h}_m)$

Remarque : l'algorithme augmente le poids des observations qui sont mal classées (*i.e.*, $Y_i \neq h_m(X_i)$) par le m^{e} expert afin que l'on prête plus attention à elles à l'étape suivante.

On peut expliquer le choix des coefficients de pondération de la façon suivante : ayant déjà à disposition un régresseur $\hat{F}_{m-1} = \sum_{k=1}^{m-1} c_k \hat{h}_k$, le classifieur obtenu si l'on s'arrêtait à l'étape m serait

$$\hat{h}^m_{\text{boost}} = \text{sign}\left(\sum_{k=1}^{m-1} c_k \hat{h}_k + c_m \hat{h}_m\right),$$

On cherche à construire un nouveau poids c pour la nouvelle contribution $\hat{h}_m$ de manière à minimiser le exp-risque de $\hat{h}^m_{\text{boost}}$. On veut donc obtenir la solution du programme suivant

$$c^*_m = \arg\min_{c\in\mathbb{R}} \mathbb{E}\left[\exp\left(-Y\left(\sum_{k=1}^{m-1} c_k \hat{h}_k(X) + c\cdot \hat{h}_m(X)\right)\right)\right].$$

Mais comme $\mathbb{E}$ est inconnue, on cherche plutôt une reformulation utilisant la contrepartie empirique :

$$c^*_m = \arg\min_{c\in\mathbb{R}} \mathbb{E}_n\left[\exp\left(-Y\left(\sum_{k=1}^{m-1} c_k \hat{h}_k(X) + c\cdot \hat{h}_m(X)\right)\right)\right].$$

En notant $\hat{F}_{m-1} = \sum_{k=1}^{m-1} c_k \hat{h}_k$, le problème devient

$$\arg\min_{c\in\mathbb{R}} \mathbb{E}_n\left[\exp\left(-Y\left(\hat{F}_{m-1}(X) + c\cdot \hat{h}_m(X)\right)\right)\right] = \arg\min_{c\in\mathbb{R}} \mathbb{E}_{\omega^{m-1}}\left[\exp(-c\cdot Y\cdot \hat{h}_m(X))\right]$$

où $\omega_i^{m-1} \propto \exp\left(-Y_i \hat{F}_{m-1}(X_i)\right)$.

11) Montrer que la solution du dernier programme d'optimisation est : $c_m = \frac{1}{2}\log\left[\frac{\mathbb{P}_{w^{m-1}}(Y=\hat{h}_m(X))}{\mathbb{P}_{w^{m-1}}(Y\neq\hat{h}_m(X))}\right]$.

12) Montrer que les poids $\omega_i^m \propto \omega_i^{m-1}\cdot\exp(-c^*_m\cdot Y_i\cdot\hat{h}_m(X_i))$ (où c^*_m est défini ci-dessus) et les poids $w_i^m \propto w_i^{m-1}\cdot\exp(2\cdot c_m\cdot \mathbb{1}_{\{Y_i\neq\hat{h}_m(X_i)\}})$ (où c_m est défini dans l'algorithme `AdaBoost`) sont identiques, avec la convention $\hat{F}_0 = 0$ et $w^0 = \omega^0 = (\frac{1}{n},\cdots,\frac{1}{n})$.

13) Mettre en œuvre AdaBoost avec des arbres de profondeur 1, puis 2, puis 10, sur le jeu de données `digits`. On calculera notamment la précision obtenue par 6-fold cross-validation. On pourra utiliser par exemple

```
from sklearn.ensemble import AdaBoostClassifier
```

14) Appliquer AdaBoost sur les données `digits` découpées en deux échantillons : apprentissage (75%) et test (25%). Tracer les erreurs (0/1) d'apprentissage et de test en fonction du nombre d'itérations.

15) Que remarquez vous ? Que se passe-t-il si la profondeur des arbres de classification est grande ?

16) (Question Bonus) : Implémenter vous-même l'algorithme AdaBoost.

Références

[Bre96] L. Breiman. Stacked regressions. *Mach. Learn.*, 24(1) :49–64, 1996. 1

[Bre01] L. Breiman. Random Forests. *Mach. Learn.*, 45(1) :5–32, 2001. 2

[FHRT00] J. Friedman, T. Hastie, and Robert R. Tibshirani. Additive logistic regression : a statistical view of boosting. *Ann. Statist.*, 28(2) :337–407, 2000. http://www-stat.stanford.edu/~jhf/ftp/boost.pdf. 3, 4

[Fri01] J. Friedman. Greedy function approximation : a gradient boosting machine. *Ann. Statist*, 29(5) :1189–1232, 2001. http://www-stat.stanford.edu/~jhf/ftp/trebst.pdf. 3

[FS97] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1) :119–139, 1997. 3

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. http://www-stat.stanford.edu/~tibs/ElemStatLearn/. 3