Etude Pratique (4) – Master Informatique – IAA – 2021
Université d'Aix-Marseille

Cécile Capponi – QARMA, LIS – AMU
Cecile.Capponi@lis-lab.fr

17 mars 2021

Objectifs de la séance :
- comprendre le boosting
- visualiser les effets des algorithmes
- comprendre les cas extrêmes et l'impact du bruit
- comparaison avec d'autres algorithmes connus

Etude du boosting

1 Visualisation d'Adaboost avec $\mathcal{X} = \mathbb{R}^2$

Le fichier `boostutil.py` contient une classe et des fonctions basiques qui seront utiles pour visualiser, itération après itération, le comportement de l'algorithme AdaBoost. L'idée est que lorsqu'Adaboost fonctionne avec des stumps, les frontières de décision des classifieurs faibles sont des droites parallèles aux composantes : de fait, *in fine*, les frontières de décision de l'ensemble des classifieurs faibles rendu par AdaBoost définissent des surfaces de décision qui sont des rectangles.

— La classe `Rectangle` permet de créer 4 points d'un rectangle dans un plot via les paramètres de `__init__`. A chaque rectangle une classe pourra être associée.

— La fonction `getStump(clf,t` permet de renvoyer quelques informations nécessaires pour la visualisation : sur l'indice de quelle composante de $\mathcal{X}$ le stump a-t-il porté, et quelle est la valeur seuil pour cette axe. Si la valeur pour cette composante est inférieure à cette valeur seuil pour un exemple x, alors x va dans le noeud gauche, sinon il va dans le noeud droit.

— La fonction `generateZones(clf,limitsx, limitsy, T, process_all=1)` permet de générer la liste des rectangles dans $\mathcal{X} = \mathbb{R}^2$ que le classifieur `clf` a permis de créer, au sein des limites des axes des composantes. Cette fonction crée cet ensemble de rectangles soit pour les `T` itérations (par défaut), soit pour la seule itération `T` lorsque `process_all` vaut 0.

1.1 Questions basiques, pour la compréhension

1. Soient l'échantillon X,Y, le classifieur CL appris par AadaBoost, et R l'ensemble des rectangles produits par CL. Indiquer les instructions permettant d'affecter une classe à chaque rectangle de R.
2. Expliquer l'algorithme de la fonction `generateZones`
3. Programmer la fonction `getStump` (la compréhension de la fonction l'utilisant est nécessaire, ainsi que toute la doc en ligne sur les arbres de décision et adaboost sous `sklearn`).

1.2 Visualisation d'AdaBoost : cas général

L'objectif ici est de voir, pas à pas et dans $\mathcal{X} = \mathbb{R}^2$, le comportement d'AdaBoost en classification binaire. L'utilisation de la documentation de `sklearn` est requise.

1. Soit l'appel :

```
X,Y = make_classification
              (n_samples=100, n_features=2, n_informative=2,
               n_redundant=0, n_repeated=0, class_sep=0.5, random_state=72)
```

2. Afficher le dataset produit par l'instruction ci-avant, avec `scatter` de `pyplot`, en adaptant la couleur des points à leur classe.
3. Que fait cette fonction ? Il faut ici comprendre – avec la doc – les paramètres valués, quitte à les changer pour mieux comprendre.
4. Apprendre un classifieur via Adaboost (base learner = stump) sur ce jeu de données, et indiquer ses performances.
5. Estimer – avec moyenne et écart-type – les performances d'AdaBoost sur ce jeu de données (base learner = stump), avec le nombre d'itérations à 20, 50, 70, 90, 150 et 300. Indiquer aussi, pour chaque apprentissage, le temps de calcul de l'apprentissage.
6. Utiliser les fonctionnalités de la section précédente pour visualiser pas à pas le comportement d'AdaBoost. Pour cela, faire varier intelligemment de 2 à 500 le nombre d'itérations d'AdaBoost.

1.3 Cas extrêmes d'AdaBoost

Ici, l'objectif est de comprendre les écueils d'AdaBoost sur des cas simples mais extrêmes. Le fichier `databoost.py` contient trois jeux de données.

1. Pour chacun de ces jeux de données, expliquer les résultats d'AdaBoost avec stump (en faisant varier le nombre d'itérations – la visualisation aide à comprendre).
2. Expliquer le différentiel entre le jeu B et le jeu C.
3. Introduire un bruit de classification : pour β allant de 10 à 50 par pas de 5, inverser les étiquettes de $\beta\%$ des exemples de l'ensemble d'apprentissage produit par le générateur `make_classification` ci-avant. Tracer la courbe des erreurs estimées par validation croisée en fonction du taux de bruit. Qu'observe-t-on ? Produire une visualisation pour $\beta = 25\%$.
4. Imaginer, toujours dans $\mathcal{X} = \mathbb{R}^2$ et avec $\mathcal{Y} = \{0, 1\}$, un autre type de jeu de données pour lequel AdaBoost fonctionnerait mal. Justifier le choix de ce jeu de données au regard du comportement attendu d'AdaBoost.

2 Observation du Gradient Boosting en régression

Nous nous plaçons dans un problème de régression. Soit la fonction à apprendre : $f(x) = x^2\sin(2x - 1)$, avec $x \in \mathbb{R}$.

— Ecrire une fonction qui génère et renvoit, entre $x = -5$ et $x + 5$, les données $(x, f(x))$.
— Donner une fonction pour l'affichage de ces données avec, en rouge, la fonction génératrice.
— Lancer un apprentissage du GradientBoosting sur ces données, avec des stumps comme base learners, et avec 5 itérations. Ajouter au plot précédent la courbe obtenue par cet apprentissage (cf courbe vue en cours). Qu'observe-t-on ?
— Faire varier la taille de l'ensemble de régresseurs de 1 à 20 par pas de 3. Afficher chaque estimateur sur la même courbe, avec les données et la fonction génératrice. Qu'observe-t-on ?
— Ajouter du bruit gaussien dans les données : définir une fonction qui modifie chaque x vers $x \pm \sigma$, avec σ pris aléatoirement entre 0.1 et 0.7. Recommencer les expériences de l'item précédent. Qu'observe-t-on ? Produire une image qui illustre ce qui est observé.
— Commenter toutes les observations, et conclure.