
TP N° 5 : Arbres de Classification

Cadre décisionnel et notations

On se place dans le cadre de la classification multi-classe, avec les notations habituelles (pour le détails des notations, on pourra consulter l'énoncé du TP sur les k -plus proches voisins) : on suppose que les données peuvent être réparties dans K classes. L'ensemble d'apprentissage est de taille n : $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$ contenant les n observations (les \mathbf{x}_i) et leurs étiquettes (les y_i). Pour mémoire $\mathbf{x}_i = (x_1, \dots, x_p)^\top \in \mathcal{X} \subset \mathbb{R}^p$ est une observation, et dans le cas bidimensionnel $p = 2$.

Découverte du module tree de scikit-learn

Consulter les pages suivantes pour démarrer ou bien trouver quelques rappels

- *** <http://scikit-learn.org/stable/modules/tree.html>
- *** <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
- ** IPython Notebook :
<http://nbviewer.ipython.org/github/ipython/ipython/tree/1.x/examples/notebooks/>
- ** Plus de détails sur les arbres (notamment pour le langage R) :
<http://www.stat.cmu.edu/~cshalizi/350/lectures/22/lecture-22.pdf>

Génération artificielle de données

On peut utiliser de nouveau les fonctions qui génèrent les données des TP précédents (knn, perceptron notamment). Afin d'afficher quelques jeux de données, vous pouvez aussi utiliser ou modifier les fonctions `plot_2d` ou `plot_2d_simple` du fichier `tp_arbres_source.py`

Arbres de décision - Algorithme CART

On pourra consulter [2, Chapitre 9.2] pour plus de détails sur les arbres. La source la plus détaillée sur le sujet étant le livre fondateur [1].

Rappelons ici le fonctionnement d'un arbre décision (voir aussi les figures en dernière page). Notons qu'on ne considère que des arbres binaires par simplicité : un nœud ne peut avoir que deux enfants, sauf si c'est une feuille, auquel cas il n'en a aucun.

On associe à toute partition des données une représentation par arbre. Au départ l'arbre est restreint à un seul nœud, sa racine, qui représente l'espace \mathcal{X} tout entier. Récursivement, à chaque étape on choisit :

- une variable $j \in \{1, \dots, p\}$ (parmi les p possibles),
- un seuil $\tau \in \mathbb{R}$

et l'on partitionne l'espace des variables explicatives \mathcal{X} en deux sous-ensembles qui sont représentés par deux nœuds dans l'arbre $G(j, \tau) = \{x = (x_1, \dots, x_p)^\top \in \mathbb{R}^p : x_j < \tau\}$ et $D(j, \tau) = \{x = (x_1, \dots, x_p)^\top \in \mathbb{R}^p : x_j \geq \tau\}$. On incrémente donc à chaque étape le nombre de composantes de la partition, et de manière équivalente le nombre de feuilles de l'arbre. On répète le processus jusqu'à atteindre un critère d'arrêt, qui peut être :

- le fait que la profondeur de l'arbre dépasse un seuil prescrit,
- le fait que l'effectif d'un nœud (*i.e.*, le nombre d'observations qui tombent dans la partition correspondante) est inférieur à un seuil prescrit,
- le fait que le nombre de feuilles de l'arbre dépasse un seuil prescrit.
- etc.

Un exemple visuel d'une telle construction est donné à la Figure 1.

Il faut maintenant définir une règle pour décider où l'on doit faire la nouvelle découpe (*splitting*). Ce choix est crucial et n'est pas unique. Pour cela on utilise une fonction qui mesure "l'impureté", que l'on note H associée à une partition. On cherche alors la découpe (variable/seuil) qui produit une partition la plus pure possible selon le critère H .

Mathématiquement il s'agit de résoudre :

$$\arg \min_{j \in \llbracket 1, p \rrbracket, \tau \in \mathbb{R}} \hat{q}_{j, \tau} H(G(j, \tau)) + (1 - \hat{q}_{j, \tau}) H(D(j, \tau)), \quad (1)$$

où l'on a noté

$$\hat{q}_{j, \tau} = \frac{|\{i \in \llbracket 1, n \rrbracket : x_i \in G(j, \tau)\}|}{|\{i' \in \llbracket 1, n \rrbracket : x_{i'} \in G(j, \tau) \cup D(j, \tau)\}|}, \quad (2)$$

la proportion des observations qui tombent dans $G(j, \tau)$. Noter qu'ici $|\cdot|$ représente le cardinal d'un ensemble.

Pour tout ensemble $R \subset \mathbb{R}^p$ et toute étiquette k on note $\hat{p}_k(R)$ la proportion d'observations qui ont k comme étiquette (numérotées de 1 à K), *i.e.*,

$$\hat{p}_k(R) = \frac{|\{i \in \llbracket 1, n \rrbracket : x_i \in R \text{ et } y_i = k\}|}{|\{i \in \llbracket 1, n \rrbracket : x_i \in R\}|} \quad (3)$$

On considérera dans CART les mesures d'impureté H suivantes :

- l'indice de Gini : $\sum_{k=1}^K \hat{p}_k(R)(1 - \hat{p}_k(R))$
- l'entropie : $-\sum_{k=1}^K \hat{p}_k(R) \log(\hat{p}_k(R))$

- 1) Dans le cadre de la régression (*i.e.*, quand on cherche à prédire une valeur numérique pour Y et non une classe), proposez une autre mesure d'homogénéité. Justifier votre choix.

Avec `scikit-learn` on peut construire des arbres de décision grâce au package `tree`. On obtient un classifieur avec `tree.DecisionTreeClassifier`.

```
from sklearn import tree
```

- 2) Simulez avec `rand_checkers` des échantillons de taille $n = 456$ (attention à bien équilibrer les classes). Créez deux courbes qui donnent le pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre (une courbe pour Gini, une courbe pour l'entropie). On laissera les autres paramètres à leur valeurs par défaut.
- 3) Afficher la classification obtenue en utilisant la profondeur qui minimise le pourcentage d'erreurs obtenues avec l'entropie (utiliser si besoin les fonctions `plot_2d` et `frontiere` du fichier source).
- 4) Exporter un graphique de l'arbre obtenu à la question précédente en format pdf. On pourra par exemple utiliser la fonction `export_graphviz` du module `tree`.
- 5) Créez $n = 160 = 40 + 40 + 40 + 40$ nouvelles données avec `rand_checkers`. Pour les arbres de décision obtenus en question 2, calculer la proportion d'erreurs faites sur cet échantillon de test. Commenter.
- 6) Reprendre la question 2 pour le dataset `ZIPCODE`. Ce jeu de données est disponible dans le module `sklearn.datasets`. On peut l'importer avec la fonction `load_digits` du dit module (ou voir pour plus de détails http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html) :

```
digits = datasets.load_digits()
```

Méthodes de choix de paramètres - Sélection de modèle

Il est rare de disposer en pratique d'un ensemble de test (on préfère inclure le plus grand nombre de données dans l'ensemble d'apprentissage), c'est au praticien de garder une partie de ces données à cet effet. Pour sélectionner un modèle ou un paramètre tout en considérant le plus grand nombre d'exemples possibles pour l'apprentissage, on utilise généralement une sélection par validation croisée. Pour chaque paramètre utilisé, une estimation de l'erreur empirique du classifieur est obtenue selon la procédure suivante. On fixe d'abord un entier N , souvent $N = 5$ ou $N = 10$, qui représente le nombre de blocs (en anglais : *fold*) et un jeu de paramètre du modèle :

- l'ensemble d'apprentissage est partitionné en N blocs de taille n/N
- pour chaque sous-ensemble possible, on mesure l'erreur obtenue par le classifieur (pour un jeu de paramètres fixé) appris sur les $N - 1$ blocs restants.
- l'erreur estimée est la moyenne de l'erreur des classifieurs appris.

On peut répéter cette procédure sur toute la grille des paramètres. Cela permet d'obtenir une mesure d'erreur pour chaque paramètre, et finalement on choisit le paramètre minimisant cette quantité.

Pour plus de détails sur les variantes de ce principe implémentées dans `scikit-learn`, consultez la page http://scikit-learn.org/stable/modules/cross_validation.html.

- 7) Utiliser la fonction `sklearn.cross_validation.cross_val_score` et tester la sur le jeu de données ZIPCODE en faisant varier la profondeur de l'arbre de décision. On pourra se servir de cette fonction pour choisir la profondeur de l'arbre.
- 8) En s'inspirant de http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html afficher la courbe d'apprentissage (en : *learning curve*) pour les arbres de décisions sur le même jeu de données¹.

- RÉGRESSION LOGISTIQUE -

Importer le paquet `sklearn.linear_model` qui contient en particulier la classe `LogisticRegression` qui nous servira d'exemple dans la suite.

```
from sklearn import linear_model
```

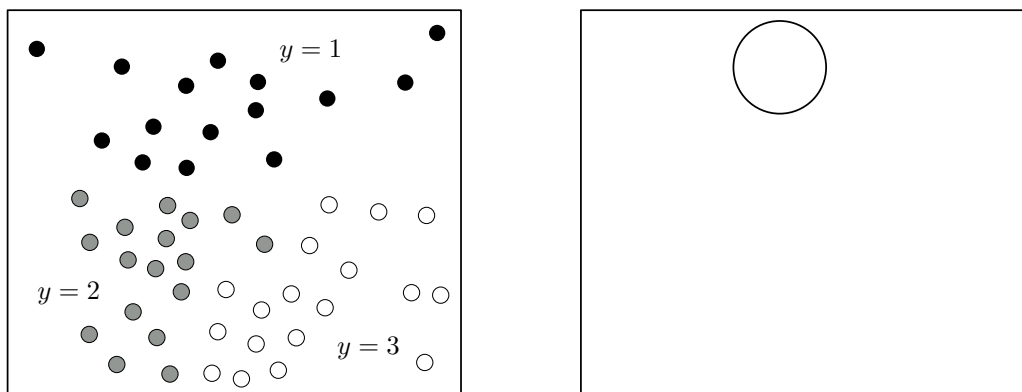
On pourra s'inspirer de l'exemple suivant pour utiliser de l'exemple : http://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html

- 9) Reprendre le jeu de données obtenu par `rand_checkers`. Effectuer une régression logistique sur ce jeu de données et donner la proportion d'erreur commise en apprentissage. Comparer la performance par rapport à la méthode par arbre de décision. Commenter.
- 10) À quoi correspond la variable `coef_` du modèle ? `intercept_` ?
- 11) Que vous retourne l'attribut `score` ?
- 12) Proposer trois manières d'appliquer la classification par régression logistique aux données issues de la base ZIPCODE. Détailler votre démarche. Testez-en deux, par exemple en utilisant des courbes d'apprentissage comme ci-dessus.

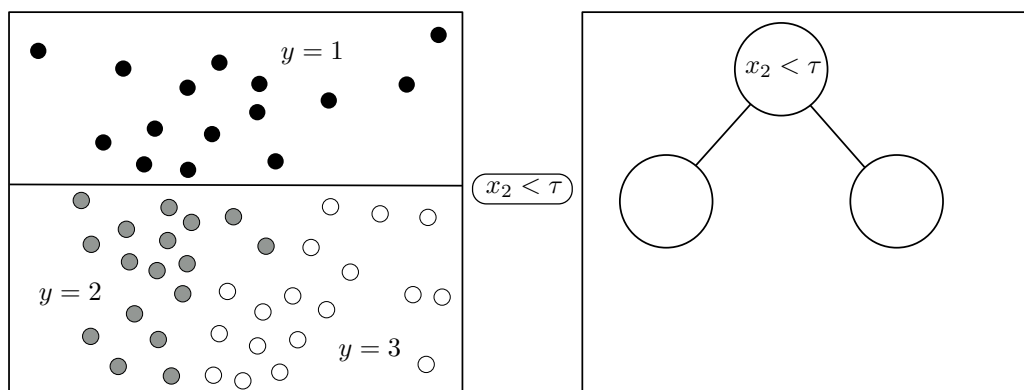
Références

- [1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth Statistics/Probability Series. Wadsworth Advanced Books and Software, Belmont, CA, 1984. 1
- [2] 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/>. 1

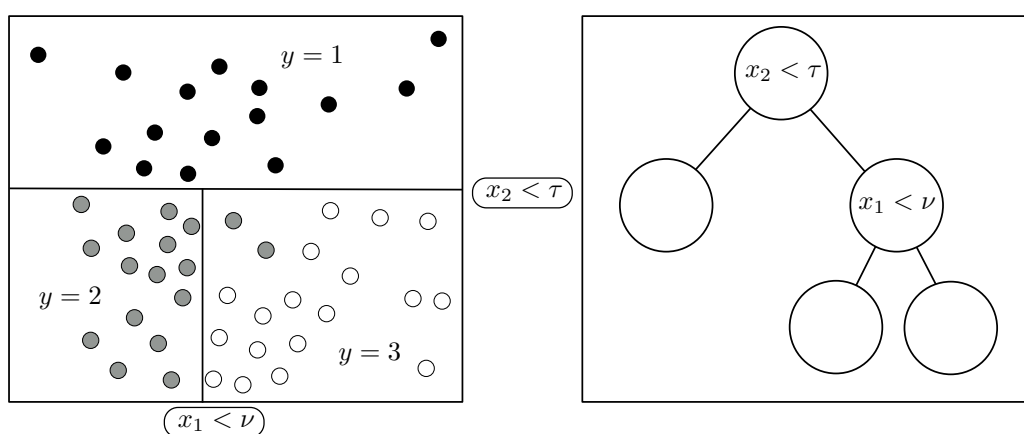
¹. pour une version de `sklearn` > version 0.18, voir http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html



(a) zéro découpe



(b) une découpe



(c) deux découpes

FIGURE 1 – Exemple de fonctionnement de la méthode de la création d'un arbre de décision