TME 1 - Arbres de décision, sélection de modèles

L'essentiel sur les arbres de décision

Un arbre de décision est un modèle de classification hiérarchique : pour des exemples sous la représentation $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, à chaque nœud de l'arbre est associé un test sur une des dimensions x_i de la forme $x_i \leq s$ avec s une valeur réelle. Ce test indique le nœud fils qui doit être sélectionné (par exemple pour un arbre binaire, le fils gauche quand le test est vrai, le fils droit sinon). À chaque feuille de l'arbre est associée une étiquette. Ainsi, la classification d'un exemple consiste en une succession de tests sur les valeurs des dimensions de l'exemple, selon un chemin dans l'arbre de la racine à une des feuilles. La feuille atteinte donne la classe prédite.
L'apprentissage de l'arbre s'effectue de manière récursive gloutonne top-down : à chaque nœud, l'algorithme doit choisir un test optimal, c'est-à-dire à la fois sur quelle dimension faire le test et quel seuillage appliqué (ce qu'on appelle un *split*). La mesure d'optimalité est en général une mesure d'homogénéité sur la partition obtenue, usuellement l'entropie de Shanon ou l'index de Gini : l'entropie d'une partition est d'autant plus petite qu'une classe prédomine dans chaque sous-ensemble de la partition, elle est nulle lorsque la séparation est parfaite (un seul label présent dans chacune des partitions) et maximale lorsque l'ensemble est le plus désordonné possible (équiprobabilité des labels dans chaque partition).
Pour calculer le split optimal, chaque dimension de l'espace de description est considérée itérativement ; pour une dimension i, les exemples sont triés par rapport à la valeur de l'attribut de cette dimension x_i, puis pour chaque split possible, le calcul de l'homogénéité des deux partitions obtenues (en termes de label) par ce split est effectué. Le split ayant la meilleure homogénéité est alors choisi.
Bien que l'algorithme pourrait continuer récursivement jusqu'à n'obtenir que des feuilles contenant un ensemble pur d'exemples (d'une seule classe), on utilise souvent des critères d'arrêts (pourquoi ? - nous y reviendrons lors de ce TP). Les plus utilisés sont le nombre d'exemples minimum que doit contenir un nœud pour être divisé (critère local) et la profondeur maximale de l'arbre (critère global).

Exercice 1 – Entropie

Le but de cet exercice est d'implémenter les fonctions utiles au calcul du partitionnement optimal, i.e. les fonctions de calcul d'entropie. *Ne passer pas trop de temps sur cette partie ! Passer si vous n'y arrivez pas. Il n'est pas nécessaire de coder ces fonctions pour la suite du TME.*

Q 1.1 Soit un objet itérable `vect` (une liste ou un vecteur `numpy` par exemple) qui contient une liste de label. Coder une fonction `entropie(vect)` qui calcule l'entropie de ce vecteur : $H(Y) = -\sum_{y \in Y} p_y log(p_y)$, p_y correspond à la probabilité du label y dans le vecteur `vect`. Penser à utiliser l'objet `Counter` du module `collections` qui permet de faire un histogramme des éléments d'une liste.

Q 1.2 L'entropie conditionnelle permet de calculer l'homogénéité de la partition obtenue. Dans le cas général d'un split n-aire en n partitions $P = \{P_1, \ldots, P_n\}$, l'entropie conditionnelle à P s'écrit $H(Y|P) = \sum_i p(P_i) H(Y|P_i)$, avec $H(Y|P_i) = -\sum_{y \in Y} p(y|P_i) log(p(y|P_i))$ l'entropie des labels conditionnée à la partition considérée, et $p(P_i) = \frac{|P_i|}{\sum_j |P_j|}$, la proportion d'éléments dans P_i. Il s'agit en fait de la moyenne pondérée des entropies des sous-ensembles obtenus.
Coder la fonction `entropie_cond(list_vect)` qui à partir d'une liste de listes de labels (la partition des labels), calcule l'entropie conditionnelle de la partition.

Q 1.3 Le code suivant permet de charger un extrait de la base imdb (à télécharger sur le site de l'ue).

```
import pickle
import numpy as np
# data : tableau (films,features), id2titles : dictionnaire id -> titre,
# fields : id feature -> nom
[data, id2titles, fields]=pickle.load(open("imdb_extrait.pkl","rb"))
# la derniere colonne est le vote
datax=data[:,:32]
datay=np.array([1 if x[33]>6.5 else -1 for x in data])
```

Chaque ligne du tableau `data` correspond à la description d'un film (le titre dans `id2titles`, chaque colonne à un attribut (dont la signification est donnée par `fields`). La plupart sont des genres (action, comédie, ...), la valeur 1 indique l'appartenance du film au genre, 0 sinon. Les dernières colonnes concernent l'année de production, la durée du film, le budget, le nombre de vote et la note moyenne attribuée au film. On binarise la note moyenne afin d'avoir deux classes, les films de note supérieure à 6.5, et les autres (vecteur `datay`).
Calculer pour chaque attribut binaire l'entropie et l'entropie conditionnelle du vote selon la partition induite par l'attribut (les exemples dont la valeur de l'attribut est 1 vs les autres). Calculer également la différence entre l'entropie et l'entropie conditionnelle pour chaque attribut. A quoi correspond une valeur de 0 ? une valeur de 1 ? Quel est le meilleur attribut pour la première partition ?

Quelques expériences préliminaires

Nous utiliserons les conventions de `scikit-learn` dans la suite : les modèles d'apprentissage sont munis d'un constructeur (qui permet d'initialiser les paramètres, variables d'instance de l'objet), d'une méthode `fit(data,labels)` qui permet d'apprendre le modèle sur les données en paramètre, d'une méthode `predict(data)` qui permet d'obtenir un vecteur de prédiction pour les données passées en paramètre, et d'une méthode `score(data,labels)` qui permet de renvoyer le pourcentage de bonne classification des données par rapport aux labels passés en paramètre.
Le code suivant permet de créer, d'apprendre un arbre de décision et de l'utiliser :

```
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier as DTree
import pydotplus

id2genre = [x[1] for x in sorted(fields.items())[:-2]]
dt = DTree()
dt.max_depth = 5 #on fixe la taille max de l'arbre a 5
dt.min_samples_split = 2 #nombre minimum d'exemples pour spliter un noeud
dt.fit(datax,datay)
dt.predict(datax[:5,:])
print(dt.score(datax,datay))
# utiliser http://www.webgraphviz.com/ par exemple ou https://dreampuf.github.io/Graphviz
export_graphviz(dt, out_file="/tmp/tree.dot",feature_names=id2genre)
# ou avec pydotplus
tdot = export_graphviz(dt,feature_names=id2genre)
pydotplus.graph_from_dot_data(tdot).write_pdf('tree.pdf')
```

Q 1.4 Sur la base de données imdb, apprendre quelques arbres de profondeurs différentes. Visualiser-les. Que remarquez-vous quant au nombre d'exemples séparés à chaque niveau en fonction de la profondeur ? est-ce normal ?

Q 1.5 Calculer les scores de bonne classification. Comment ils évoluent en fonction de la profondeur ? Est-ce normal ?

Q 1.6 Ces scores sont-ils un indicateur fiable du comportement de l'algorithme ? Comment obtenir un indicateur plus fiable ?

Sur et sous apprentissage

Pour obtenir une meilleure estimation de l'erreur du classifieur appris, il est usuel d'utiliser deux ensembles d'exemples étiquetés :

- l'ensemble d'apprentissage : l'apprentissage du classifieur ne se fait que sur ce sous-ensemble d'exemples ;
- l'ensemble de test : cet ensemble sert à évaluer l'erreur du classifieur.

Ces deux sous-ensembles sont tirés de manière aléatoire en faisant une partition en 2 parties des exemples disponibles. L'erreur faîte sur l'ensemble d'apprentissage s'appelle l'erreur d'apprentissage, celle sur l'ensemble de test l'erreur de test.

Q 1.7 Pour différents partitionnement, par exemple des partages en $(0.2, 0.8), (0.5, 0.5), (0.8, 0.2))$, tracer les courbes de l'erreur en apprentissage et de l'erreur en test en fonction de la profondeur du modèle.

Q 1.8 Que remarquez vous quand il y a peu d'exemples d'apprentissage ? Comment progresse l'erreur ? De même quand il y a beaucoup d'exemples d'apprentissage. Est-ce le même comportement pour les deux erreurs ?

Q 1.9 Vos résultats vous semblent ils fiables et stables ? Comment les améliorer ?

Validation croisée : 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). Pour sélectionner un modèle tout en considérant le plus grand nombre d'exemples possible pour l'apprentissage, on utilise généralement une procédure dite de sélection par validation croisée. Pour chaque paramétrisation de l'algorithme, une estimation de l'erreur empirique du classifieur appris est faîte selon la procédure suivante :

- l'ensemble d'apprentissage E_{app} est partitionné en n ensembles d'apprentissage $\{E_i\}$
- Pour $i = 1..n$
  - ▶ l'arbre est appris sur $E_{app} \backslash E_i$
  - ▶ l'erreur en test $err(E_i)$ est évaluée sur E_i (qui n'a pas servi à l'apprentissage à cette itération)
  - ▶ l'erreur moyenne $err = \frac{1}{n} \sum_{i=1}^{n} err(E_i)$ est calculée, le modèle sélectionné est celui qui minimise cette erreur

Refaire les expériences précédentes avec cette fois de la validation croisée (soit faîte à la main, soit en utilisant les fonctions de `scikit-learn`). Vous pouvez également étudier les autres hyper-paramètres du modèle (nombre minimal d'exemples par nœud, gain d'entropie minimal ...)