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 : à chaque nœu d de l'arbre est associé un test sur une des dimensions x_i de la forme $x_i\{\leq, >, =\}s$ (s une valeur réelle) qui 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). A 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 top-down : à chaque nœud, l'algorithme choisit le split vertical (seuillage d'une variable) qui optimise 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).
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 communs sont le nombre d'exemples minimum que doit contenir un nœud pour être divisé et la profondeur maximale de l'arbre.

1. Coder une fonction `entropie(vect)` qui calcule l'entropie d'un vecteur ($H(x) = -\sum_i p_i \log(p_i)$, p_i correspond aux probabilités des différentes valeurs du vecteur). Pensez à utiliser l'objet `Counter` du module `collections` qui permet de faire un histogramme des éléments d'une liste.
2. Coder une fonction `entropie_cond(vect1,vect2` qui calcule l'entropie conditionnelle de la variable représentée par le premier vecteur par rapport à celle représentée par le deuxième ($H(Y|X) = \sum_i p_j H(Y|X = i)$).
   Le code suivant permet de charger la base imdb.

```
# data : tableau (films,features), id2titles : dictionnaire id -> titre film, fields : i
[data, id2titles, fields]=cPickle.load(file("imdb_extrait.pkl"))
# 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 attribué 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`).
3. Calculer pour chaque attribut l'entropie et l'entropie conditionnelle du vote binarisé par rapport à l'attribut. Calculez également la différence entre ces 2 valeurs pour chaque attribut. A quoi correspond une valeur de 0 ? une valeur de 1 pour les attributs binaires ?

Quelques expériences préliminaires

Tous les modèles d'apprentissage que nous étudierons seront sur le même schema : soit par exemple la classe `Classifieur` :

- création du classifieur : mon_classifieur=Classifier()
- réglage des paramètres, par exemple dans la suite la profondeur maximale : `mon_classifieur.max_ = 5`
- apprentissage du classifieur : `mon_classifieur.fit(data,labels)`, `data` un tableau de taille (nombre d'exemple, nombre de dimension) et `labels` un vecteur de labels
- prediction pour de nouveaux exemples : `mon_classifieur.predict(data)`
- score du classifieur (précision, pourcentage d'exemples bien classés) :`mon_classifieur.score(dat`

Télécharger l'archive sur le site de l'UE. Le code suivant permet de créer, d'apprendre un arbre de décision et de l'utiliser :

```
from decisiontree import
import cPickle
[data, id2titles, fields]=cPickle.load(file("imdb_extrait.pkl"))
datax=data[:,:32]
datay=np.array([1 if x[33]>6.5 else -1 for x in data])
dt = DecisionTree()
dt.max_depth = 5 #on fixe la taille de l'arbre a 5
dt.fit(datax,datay)
dt.predict(datax[:5,:])
print dt.score(datax,datay)
dt.to_pdf("/tmp/test_tree.pdf",fields) # dessine l'arbre dans un fichier pdf
```

1. Sur la base de données imdb, apprenez quelques arbres de profondeurs différentes. Visualisez-les. Que remarquez-vous quant au nombre d'exemples séparés à chaque niveau en fonction de la profondeur ? est-ce normal ?
2. Calculez les scores de bonnes classification. Comment ils évoluent en fonction de la profondeur ? Est-ce normal ?
3. 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.

1. Pour différents partitionnement, (faites varier le nombre d'exemples de chaque ensemble, par exemple un partage $(0.2/0.8), (0.5, 0.5), (0.8, 0.2)$), tracez les courbes de l'erreur en apprentissage et de l'erreur en test en fonction de la profondeur du modèle.

2. 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 ?
3. Vos résultats vous semble-t-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} 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

Refaites les expériences précédentes avec cette fois de la validation croisée (pensez à la fonction `np.random.shuffle`).