tp3

Unknown Author

May 7, 2015

1 TP : Arbres de décision et fôrets aléatoires

1.1 Résumé

Un arbre de décision est un modèle de classification hiérarchique : à chaque noeud 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 noeud 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 noeud, 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 les suivants :

- le nombre d'exemples minimum que doit contenir un noeud
- la profondeur maximale de l'arbre
- la différence de gain de la mesure d'homogénéité entre le noeud père et les noeuds fils

1.2 Prise en main sklearn, données artificielles

scikit-learn est un des modules de machine learning les plus populaires (installation : pip install scikit-learn –user). Il contient les algos que nous avons déjà vu (knn, noyaux, perceptron, regression), et bien d'autres outils et algorithmes.

In []:
```
import numpy as np # module pour les outils mathématiques
import matplotlib.pyplot as plt # module pour les outils graphiques
import tools # module fourni en TP1
from sklearn import tree # module pour les arbres
from sklearn import ensemble # module pour les forets
from sklearn import cross_validation as cv
from IPython.display import Image
import pydot

%matplotlib inline
```

Tous les modeles d'apprentissage sous scikit fonctionnent de la manière suivante :

- création du classifieur (ici **cls=Classifier()**)
- réglage des paramètres (par exemple la profondeur maximale, le nombre d'exemples par noeud)
- apprentissage du classifieur par l'intermédiaire de la fonction **cls.fit(data,labels)**
- prediction pour de nouveaux exemples : fonction **cls.predict(data)**
- score du classifieur (précision, pourcentage d'exemples bien classés) : fonction **cls.score(data,labels)**

Pour un arbre de déciion, la classe est **tree.DecisionTreeClassfier()**. Dans le cas des arbres de décisions, nous avons aussi la possibilité d'obtenir l'importance des variables, un score qui est d'autant plus grand que la variable est "utile" pour la classification.

In []:
```
#Initialisation
data,y=tools.gen_arti()
mytree=tree.DecisionTreeClassifier() #creation d'un arbre de decision
mytree.max_depth=8 #profondeur maximale de 5
mytree.min_samples_split=1 #nombre minimal d'exemples dans une feuille
#Apprentissage
mytree.fit(data,y)

#prediction
pred=mytree.predict(data)
print "precision : ", (1.*pred!=y).sum()/len(y)

#ou directement pour la precision :
print "precision (score) : "  +` mytree.score(data,y)`

#Importance des variables :
plt.subplot(1,2,2)
plt.bar([1,2],mytree.feature_importances_)
plt.title("Importance Variable")
plt.xticks([1,2],["x1","x2"])

#Affichage de l'arbre
with file("mytree.dot","wb") as f:
    tree.export_graphviz(mytree,f)
```

Sur différents jeux de données artificielles (des tps précédents) :

- *observer les frontières de décision en fonction de la taille de l'arbe.*
- *faites varier les différents paramètres disponibles (hauteur de l'arbre, nombre d'exemples dans les noeuds par exemple) et tracer la précision en fonction de ces paramètres. Que remarquez vous sur la précision ?*
- *Est-ce que cette valeur de précision vous semble une estimation fiable de l'erreur ? Pourquoi ?*

1.3 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 paramètrisation du problème, une estimation de l'erreur empirique du classifieur appris est faîte selon la procédure suivante :

- l'ensemble d'apprentissage E_{app} est partitioné 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

Ci-dessous quelques fonctions utiles pour la sélection de modèle :

In []:
```
#permet de partager un ensemble en deux ensembles d'apprentissage et de test
data_train,data_test,y_train,y_test=cv.train_test_split(data,y,test_size=0.3)
mytree.fit(data_train,y_train)
print "precision en test (split 30 %) : ", mytree.score(data_test,y_test)

#permet d'executer une n-validation croisée et d'obtenir le score pour chaque tentativ
print "precision en test (10-fold validation) : ",cv.cross_val_score(mytree,data,y,cv=

#alternative : obtenir les indices et itérer dessus
kf= cv.KFold(y.size,n_folds=10)
res_train=[]
res_test=[]
for cvtrain,cvtest in kf:
    mytree.fit(data[cvtrain],y[cvtrain])
    res_train+=[mytree.score(data[cvtrain],y[cvtrain])]
    res_test+=[mytree.score(data[cvtest],y[cvtest])]
print "ou de maniere analogue : "
print "precision en train : ",res_train
print "precision en test : ",res_test
print "moyenne train : ",np.mean(res_train)," (", np.std(res_train),")"
print "moyenne test : ",np.mean(res_test)," (",np.std(res_test),")"
```

Manipuler sur les différents types de génération artificielle ces fonctions afin de trouver les meilleurs paramètres selon le problème. Tracer l'erreur d'apprentissage et l'erreur de test en fonction des paramètres étudiés. Que se passe-t-il pour des profondeurs trop élevées des arbres ?

1.4 Classification données USPS

Tester sur les données USPS (en sélectionnant quelques sous-classes). Observer l'importance des variables. Afficher la matrice 2D de la variable importance de chaque pixel de l'image (avec **plt.imshow(matrix)**). Les résultats semble-t-ils cohérents ? Utiliser l'algorithme du perceptron fourni par sklearn (**linear_model.Perceptron**) ou le votre et comparer les résultats obtenus pour les poids.

Sur quelques exemples, comparer les performances des arbres, des knns (**neighbors.KNeighborsClassifier**) et du Perceptron en utilisant la validation croisée pour calibrer au mieux vos modèles.

Expérimenter également les fôrets aléatoires : c'est une méthode de baging très utilisée, qui consiste à considérer un ensemble d'arbres appris chacun sur un échantillonage aléatoire de la base d'exemples; la classification se fait par vote majoritaire (**enemble.RandomForestClassifier()**).

1.5 Classification sur la base movielens

Introduction

La base movielens est une base de données issue d'imdb, qui contient des informations sur des films (le genre, l'année de production, des tags) et des notes attribuées par les utilisateurs. Elle est utilisée généralement pour la recommendation de films. Nous allons l'utiliser dans le cadre de la classification, afin de prédire si un film est bon ou mauvais, dans deux contextes :

- en prenant en compte uniquement l'information sur le film et le score moyen du film
- en prenant en compte l'information de l'utilisateur qui score le film

Télécharger l'archive suivante

Le bloc de code suivant est utilisé pour charger et prétraiter les données.

In []:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from numpy import random


### liste des professions
occupation=[ "other", "academic/educator", "artist", "clerical/admin", "college/grad s
"executive/managerial", "farmer", "homemaker", "K-12 student", "lawyer", "programmer",
"self-employed", "technician/engineer", "tradesman/craftsman", "unemployed", "writer"]

### liste des genres
genre=['unknown','Action','Adventure','Animation',"Children's",'Comedy','Crime','Docum
'Horror','Musical','Mystery','Romance','Sci-Fi','Thriller','War','Western']

dicGenre=dict(zip(genre,range(len(genre))))

dicOccup=dict(zip(occupation,range(len(occupation))))


def read_mlens(fname,sep='::'):
    """ Read a generic .dat file
            - fname : file name
            - sep : separator
        return : list of lists, each list contains the fields of each line read """

    def toint(x):
        if x.isdigit():
            return int(x)
        return x
    f=open(fname,'r')
    tmp= [ s.lstrip().rstrip().split(sep) for s in f.readlines()]
    #lstrip et rstrip enleve les blancs de debut et de fin
    f.close()
    return [ [toint(x) for x in y] for y in tmp ]
def read_movies(fname="movies.dat"):
    """ Read movies information, reindexing movies
        return : res: binary matrix nbmovies x genre, res[i,j]= 1 if movie i has the g
                 movies2id : original idMovie to reindexed id
                 dic2Movies : reindexed id to list of movie information (id, title, ge
    """
    movies=read_mlens(fname)
    dicMovies=dict()
    movies2id=dict()
    res=np.zeros((len(movies),len(genre)))
    for i,m in enumerate(movies):
        dicMovies[i]=m
        movies2id[m[0]]=i
        for g in m[2].split('|'):
            res[i,dicGenre[g]]=1
    return res,movies2id,dicMovies

def read_users(fname="users.dat"):
    """ Read users informations
        return : nbusers * 3 : gender (1 for M, 2 for F), age, occupation index"""
    users=read_mlens(fname)
    res=np.zeros((len(users),3))
    for u in users:
        res[u[0]-1,:]=[u[1]=='M' and 1 or 2, u[2],int(u[3])]
    return res

def read_tags(fname="tags.dat"):
    """ Read tags informations
        return : dictionary : idTag->(label,number of uses)
        """
    tags=read_mlens(fname,"\t")
    res=dict()
    for t in tags:
```

```
        res[t[0]]=(t[1],t[2])
    return res


def read_files(mname="movies.dat",uname="users.dat",rname="ratings.dat",tname="tags.da
    """ Read all files
        return :
            * movies: binary matrix movies x genre
            * users : matrix users x (gender, age, occupation index)
            * ratings : matrix movies x users, with score 1 to 5
            * movies2id : dictionary original id to reindexed id
            * dicMovies : dictionary reindexed id to movie information
            * tags : dictionary idTag -> (name,popularity)
            * tagrelevance : matrix movies x tags, relevance
    """
    print "Reading movies..."
    movies,movies2id,dicMovies=read_movies(mname)
    print "Reading users..."
    users=read_users(uname)
    print "Reading ratings..."
    rtmp=read_mlens(rname)
    ratings=np.zeros((movies.shape[0],users.shape[0]))
    for l in rtmp:
        ratings[movies2id[l[1]],l[0]-1]=l[2]
    print "Reading tags..."
    tags=read_tags(tname)
    tagrelevance=np.zeros((movies.shape[0],len(tags)))
    print "Reading tags relevance..."
    with open(trname) as f:
        for i,ltag in enumerate(f):
            if i % 100000 == 0:
                print str(i /100000)+" 00k lines"
            ltag=ltag.rstrip().split("\t")
            if int(ltag[0]) in movies2id:
                tagrelevance[movies2id[int(ltag[0])],ltag[1]]=float(ltag[2])

    return movies,users,ratings,tags,tagrelevance,movies2id,dicMovies


###lire les fichiers
movies,users,ratings,tags,tagrelevance,movies2id,dicMovies=read_files()
```

Les informations suivantes sont stockées :

- movies: une matrice binaire, chaque ligne un film, chaque colonne un genre, 1 indique le genre s'applique au film
- users : une matrice, chaque ligne un utilisateur, et les colonnes suivantes : sexe (1 masculin, 2 feminin), age, index de la profession
- ratings : une matrice de score, chaque ligne un film, chaque colonne un utilisateur
- movies2id : dictionnaire permettant de faire la correspondance entre l'identifiant du film à l'identifiant réindexé
- dicMovies : dictionnaire inverse du précédent
- tags : dictionnaire des tags, identifiant vers le couple (nom,popularité)
- tagrelevance : matrice, chaque ligne un film, chaque colonne un tag, chaque case un score entre 0 et 1

Classification à partir de l'information unique du film

Notre matrice **movies** ne contient que les informations du genre du film. Il vous faut y ajouter également l'année de production du film. Nous allons considérer le problème de classification binaire si un film est bon ou non, en considérant comme bon les filmes de score supérieur à 3.

- *Transformer les données et fabriquer les labels. Utiliser les arbres de décisions et le perceptron pour cette tache d'apprentissage.*
- *Sur quelques paramètres, que remarquez vous sur l'erreur d'apprentissage et de test ?*
- *La taille de l'ensemble de test joue-t-elle un rôle ?*
- *Tracer les courbes de ces deux erreurs en fonction de la profondeur. Que remarquez vous ? Quels sont les meilleurs paramètres pour l'erreur en apprentissage et en test ?*
- *Quelles sont les variables les plus importantes ?*

Classification avec les informations utilisateurs

Proposer une manipulation des données qui permettent d'intégrer les informations utilisateurs dans le processus de classification. Tester (meme questions que précedement).

- *Etudier comme précédement les erreurs en test et en apprentissage. Comparer ces erreurs si vous réduisez les utilisateurs par tranche d'age. Remarquez-vous des tranches d'age plus stable ?*
- *Comparer vos résultats si vous utilisez une fôret aléatoire (aggrégation d'arbres).*