TD1-exo Introduction Python (sept 2016)

September 20, 2016

1 TD 1 - Septembre 2016

1.0.1 Introduction à python, numpy, matplotlib (et un peu de machine-learning)

mon mail : thomas.schmitt#inria.fr

In [ ]:

tao.lri.fr : activités : Courses : Module Apprentissage, TD 1

In [ ]:

Python est un langage : - interprété : les instructions sont lues et traduites en langage machine ligne à ligne. - non typé : Il n'est pas utile de préciser le type des variables : il sera directement assigné par l'interpréteur.

C'est un langage dont l'usage est extrêmement répandu dans les milieux académique et industriel.

In [ ]:

IDE utilisé : jupyter notebook (aussi appelé : IPython) Après avoir téléchargé Anaconda (distribution python incluant les package populaires :http://continuum.io/downloads)
lancer jupyter : $ jupyter notebook

In [ ]:

Raccourcis clavier :
- ctrl(shift) + enter # execute la cellule
- escape, b (ou le plus en haut à gauche) # nouvelle cellule
- shift + tab # ouvre la doc d'une fonction

2 I - fonctions de bases

```
In [ ]: a = 4
```

```
In [ ]: a
```

In [ ]:

2.1 1) List et slice

```
In [ ]: l = [10,11,12,13,14,15]
```

```
In [ ]: l
```

```
In [ ]: l[0]
```

```
In [ ]: l.append(16) # ajoute un élément en fin de liste
        l

In [ ]: l.count(14) # compte le nombre d'élements (==14)

In [ ]: l[3:7] #slice entre 3 inclus et 7 exlus

In [ ]: l = []
        for i in range(10): #r ange : créer une liste d'entier
            l.append(i) # une tabulation (ou 4 espaces) à l'intérieur d'une boucle ou d'une fonctions
        l

In [ ]: #méthode pythonic
        ll = [i for i in range(10)]
        ll

In [ ]: def my_function(x):
            """ docstring : description de ma fonction """

            if x < 0:
                z = 'it is a negatif number'
            else:
                z = x*10

            return z

In [ ]: my_function(-3)

In [ ]: my_function(9)

In [ ]: def prod(x, y = 5):
            """ x *y """
            return x*y

In [ ]: prod(10)

In [ ]: prod(10,y = 10)
```

2.2 2) Array (tableau)

1 dimension

```
In [ ]: import numpy as np # import la librairie numpy (calcul scientifique)
        from __future__ import print_function, division # compatibilite python 2 et python 3

In [ ]: a = np.arange(5,15) # liste entre 5 et 15
        print(a)

In [ ]: a[0]

In [ ]: a[(a%2==0)] # rend les termes paire

In [ ]: b = np.arange(0,1,0.1)  #shift + tab to read the doc inside the parentheses
        b

In [ ]: b[[2,3,4]] # rend les élements de positions 2,3 et 4

In [ ]: a + b # somme terme à terme

In [ ]: b * 3
```

2 dimension

```
In [ ]: A = np.array([[0,1,2],[3,4,5],[6,7,8]])
        print(A)
```

```
In [ ]: A.shape
```

```
In [ ]: I = np.eye(3,3)
        I
```

```
In [ ]: Z = np.zeros((5,2))
        O = np.ones((5,2))
        R = np.random.rand(5,2) # tire des nombre aléatoire uniforme (entre 0 et 1)
```

```
In [ ]: R
```

```
In [ ]: R.shape
```

```
In [ ]: (Z + 1) * 3
```

```
In [ ]: R * Z # /!\ produit terme à terme (sinon voir np.dot)
```

```
In [ ]: v = np.array([0,1,10])
        print('A =')
        print(A)
        print('v =')
        print(v)
```

```
In [ ]: print('A.v =',np.dot(A,v)) #produit matriciel
```

```
In [ ]: np.dot(A,I)
```

2.3 3) plot et scatter

```
In [ ]: import matplotlib.pyplot as plt # librairie pour afficher les graphs.
        # Doc : http://matplotlib.org/gallery.html, http://matplotlib.org/api/pyplot_api.html
        %matplotlib inline
        # Affiche dans la fenêtre courante.
```

```
In [ ]: x = np.arange(-6,6,0.1)
        y = np.cos(x)
        plt.plot(x,y)
```

```
In [ ]: x = np.arange(-6,6,0.1)
        y = np.cos(x)
        z = np.sin(x)
        plt.plot(x,y, label = 'cos')
        plt.plot(x,z, label = 'sin')
        plt.title('cosinus')
        plt.xlabel('abscisses')
        plt.ylabel('ordonnées')
        plt.legend()  #affiche la legend
```

```
In [ ]: x = np.random.rand(100) * 10
        y = np.random.rand(100) + 10
        plt.scatter(x,y) #point bleu
        plt.scatter(np.mean(x),np.mean(y), c = 'red', s=100)
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

3 II - Exercices

3.0.1 0) Mean

Écrire une fonction qui calcul la moyenne d'une liste.

```
In [ ]: def my_mean(X):
            """ calcul la moyenne de la liste (ou tableau) X"""

            #todo
```

```
In [ ]: X = [10,11,12,13]

        if np.mean(X) == my_mean(X):
            print('ok, mean =', my_mean(X))
        else:
            print('erreur, mean = ', my_mean(X))
```

3.1 1) Produit scalaire

On rappelle que, pour $ x = (x_1,...,x_n)$ et $y = (y_1, \ldots, y_n)$ dans $\mathbb{R}^n$, on définit le produit scalaire de x et y par $< x, y >= \sum_{i=1}^{n} x_i y_i = xyT$.

Écrire deux fonctions prenant en entrée deux vecteurs (list ou array) :

- La première devra calculer leur produit scalaire comme produit matriciel
- La seconde devra calculer leur produit scalaire de manière itérative. (*len* permet de connaitre la longueur d'une liste ou d'un tableau 1D).

Tester le temps de calcul de chaque méthode (%timeit my_function(x,y))

```
In [ ]: def prod_scal_mat(x,y):
            """ rend le produit scalaire comme produit matriciel """

            #todo

        def prod_scal_iter(x,y):
            """ rend le produit scalaire de manière itérative """

            #todo
```

```
In [ ]: x = np.arange(10000)
        y = np.arange(10000)

        z1 = prod_scal_mat(x,y)
        z2 = prod_scal_iter(x,y)

        if z1 == z2 :
            print('ok, z=', z1)
        else:
            print('z1 != z2', z1, z2)
```

```
In [ ]: %timeit prod_scal_mat(x,y)
```

```
In [ ]: %timeit prod_scal_iter(x,y)
```

3.2 2) Approximation de $\frac{\pi}{4}$ par Monte-Carlo

On désire approximer $\frac{\pi}{4}$ par méthode de Monte-Carlo. Un point tiré uniformément dans $[0,1] \times [0,1]$ est dans le quart de disque vérifiant $x2 + y2 \leq 1$ avec probabilité $\frac{\pi}{4}$.

a) Écrire une fonction qui :

- prend en argument un entier N
- tire N points indépendemment et uniformément sur $[0,1] \times [0,1]$
- affiche en rouge, avec la commande *plt.scatter*, les n_1 points dans le quart de disque de rayon 1
- affiche en bleu les autres points
- retourne le rapport $\frac{n_1}{N}$

b) Afficher l'évolution de $\frac{n_1}{N}$ en fonction de N. (on peut utiliser la fonction np.linspace(start, end,step))

```
In [ ]: def MonteCarlo(N=1000, scatter = True):
            """ rend une approximation de pi/4
            si scatter, alors affiche les points rouges et bleus """

            #todo
```

```
In [ ]: MonteCarlo(1000) * 4
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

4 III - Machine Learning

4.1 Classification

4.1.1 Exercices

- explorer les donnees (X.shape, plt.scatter,. . .)
- faire une fonction de score de classification (= nb points correctement classé / nb de points)
- faire un classifieur naif (sur une composante de X par exemple)
- tester son score
- améliorer ce score ! (optimiser le seuil de la valeur de séparation par exemple).

```
In [ ]: def generate_data_gauss(n = 1000):
            """ generate X,y for classification """

            np.random.seed(42)
            X = np.random.randn(n,2)
            X[:int(n/2)] += 3
            y = [0] * int(n/2) + [1] * (len(X) - int(n/2))

            return X,y
```

```
In [ ]: X,y = generate_data_gauss()

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: def score_classif(y_pred,y_true):
            """ score de classification
            = #{y_pred == y_true} / #{y_pred} """

            #todo

        # plus pythonic (sans boucle for):
        def score_classif_py(y_pred, y_true):
            """ score de classification
            = #{y_pred == y_true} / #{y_pred} """

            return None

In [ ]: def decision0(x, seuil = 2):

            """ classe x suivant sa première composante """

            #todo

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```

5 with scikit-learn

http://scikit-learn.org/

```
In [ ]: import sklearn # if not installed : £ pip install sklearn
```

5.1 Decision Tree

http://scikit-learn.org/stable/modules/tree.html

5.2 Exercices

- à l'aide de la fonction "plot_boundary" ci dessous et de la classe "tree.DecisionTreeClassifier", entrainer un classifieur sur les données et afficher les zones de prédictions.

```
In [ ]: def generate_data_lines():
            # Make data
            rng = np.random.RandomState(13)
            d1 = np.asarray(((np.arange(10) + rng.rand(10),np.arange(10)+ rng.rand(10)))).T
            d2 = np.asarray(((np.arange(3,13)+ rng.rand(10),np.arange(10)+ rng.rand(10)))).T
            X = np.vstack((d1,d2))
            y = [0] * d1.shape[0] + [1] * d2.shape[0]

            return X,y

        # Ne pas s'attarder sur cette fonction pour le moment.
        def plot_boundary(clf, X, y):
            """Affiche la carte avec les zones de chaque catégorie"""

            plt.figure()
            plot_step = 0.02
            x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
            y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
            xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                                 np.arange(y_min, y_max, plot_step))
            Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
            Z = Z.reshape(xx.shape)
            cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
            plt.scatter(X[:,0],X[:,1], c = y)
            # plt.title('score = ' + str(clf.score(X,y)))

In [ ]: X,y  = generate_data_lines()

In [ ]: plt.scatter(X[:,0],X[:,1], c = y, s = 50)

In [ ]: from sklearn import tree
        #http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

In [ ]: clf = tree.DecisionTreeClassifier() # clf : une instance de la classe 'DecisionTreeClassifier'

In [ ]: clf = clf.fit(X, y) # fit : s'entraine sur les données

In [ ]: y_pred = clf.predict(X) # predict : prédit le label

In [ ]: score_classif(y_pred, y)

In [ ]: from sklearn.metrics import accuracy_score
        accuracy_score(y_pred, y)

In [ ]: clf.feature_importances_

In [ ]: plot_boundary(clf, X, y)

In [ ]: # fait varier max_depth
        for max_depth in np.arange(1,10):
            clf = tree.DecisionTreeClassifier(max_depth=max_depth)
            clf = clf.fit(X, y)
            plot_boundary(clf, X,y)

            if clf.score(X,y) == 1.:
                print('break, score = 1')
                break
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
from sklearn.datasets import make_moons
X,y = make_moons(n_samples=200, noise=.1, random_state=14)
plt.scatter(X[:,0],X[:,1], c = y, s=90)
```

6 Regression

data : 1 dimension
valeur objectives : scalaire

6.0.1 Exercices

- explorer les donnees (X.shape, plt.scatter,...)
- faire une fonction d'erreur de regression : moindre carrée par exemple : err $= 1/N \sum_i |ypred_i 2 - y_i 2|2$
- faire un regresseur naif
- tester son erreur

In [ ]:
```
# Create a noisy sinus dataset
def regression_data():

    rng = np.random.RandomState(1)
    X = np.sort(5 * rng.rand(80, 1), axis=0)
    y = np.sin(X).ravel()
    y[::5] += 3 * (0.5 - rng.rand(16)) # ajout de bruit

    X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
    y_test = np.sin(X_test).ravel()
    return X,y,X_test,y_test
```

In [ ]:
```
X,y, X_test,y_test = regression_data()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
def score_regression(y_pred, y_true):
    """ return 1/N \sum_i |ŷ_i^2 -y_i^2|^2"""

    #todo
```

In [ ]:
```
def score_regression_py(y_pred, y_true):
    """ return y_pred^2 - y_true^2"""

    return None
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

6.1 Decision Tree

In [ ]:
```
from sklearn import tree

clf_1 = tree.DecisionTreeRegressor(max_depth=None)
clf_1.fit(X, y)
score_train = clf_1.score(X,y)
print('score_train = ', score_train)

# Predict on test set
y_1 = clf_1.predict(X_test)
print('score_test =')

# Plot the results
plt.figure()
plt.scatter(X, y, c="k", label="data")
plt.plot(X_test, y_1, c="g", label="", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("DT Regression. score train = " + str(np.round(score_train,3)) +\
        " score test = ")
plt.legend()
plt.show()
```

6.2 Exercices

a) Evaluer le score du modèle sur un jeu de donnée test

b) Dans une boucle, faire varier le paramètre 'max_depth' et afficher les predictions

c) Afficher sur un même graphe l'évolution du score d'entrainement et du score de test

d) À l'aide d'un ensemble de validation, et sans utiliser l'ensemble de test, estimer la meilleur valeur pour 'max_depth'

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# Ce que l'on peut faire avec Jupyter :
# https://github.com/rhiever/Data-Analysis-and-Machine-Learning-Projects/blob/master/example-da
```