Machine Learning pentru Aplicatii Vizuale

1. Introducere in Python

Inainte de toate, pentru a lucra intr-un mediu care sa nu fie incarcat de prea multe biblioteci care pot conduce la conflicte, se recomanda crearea unui mediu virtual pentru Python. Din Anaconda Prompt:

conda create --name pytorch_env python=3.6

Dupa care, pentru activare la urmatoarele laboratoare rulati comanda:

activate pytorch_env

In [ ]:
```
# Definirea variabilelor
a = 5
b = 2
c = 10.0
d = 'sir de caractere'

# Operatii de baza
result1 = a+b
print(result1)

result2 = a+c
print(result2)

result3 = a*b
print(result3)

result4 = a/b
print(result4)

result5 = a//b
print(result5)

result6 = c/b
print(result6)

result_string = 'Acesta este un ' + d
print(result_string)

# Structuri de date - liste
l1 = [1,2,3,4]
```

Structurile conditionale si buclele sunt esentiale programare. O particularitate a limbajului Python este data de importanta indentarilor in scrierea codului, care forteaza programatorii sa scrie cod mai organizat. La intrarea in orice bloc de instructiuni (sturcturi conditionale, bucle, definiri de clasa, functii, etc.) codul trebuie indentat. Revenirea la indentarea anterioara semnifica iesirea din acel bloc.

In [ ]:
```
# Exemplu if
a = 5
if a > 4:
    print('OK')
else:
    print('Not OK')

# Exemplu for
for i in range(5):
    print(i)
```

2. Biblioteca Numpy

Numpy este biblioteca de facto in ceea ce priveste calculelor matematice in Python. Permite crearea matricelor si manipularea acestora prin intermediul algebrei liniare.

Pentru instalare, din consola Python rulati:
conda install numpy

In continuare se vor prezenta cateva aspecte legate de functionarea Numpy. Structura ei fundamentala de date (matricele de tip numpy.array) sunt similare in anumite aspecte cu liste implicite in Python. Dezavantajul major al listelor este ca operatiile obisnuite specifice algebrei liniare (adunari, scaderi, inmultiri, impartiri, produse scalare) nu sunt definite pentru aceasta sturctura de date.
Pentru ilustrarea facilitatilor de calcul elementar ale Numpy, se vor utiliza mai intai vectori, ca un caz particular al matricelor (linie sau coloana).

In [ ]:
```
# Importarea bibliotecilor relevante pentru laboratoare
import numpy as np

# Structuri de date speciale - Matrice Numpy
# Se defineste vectorul 'vect' si se aplica operatii de baza asupra lui
vect = np.array([1,2,3,4])
print('vect = ',vect)

# Adunarea unui scalar (asemanator pentru scadere)
add_vect = vect + 1

print('Adunarea unui scalar\n{} + 1 = '.format(vect),add_vect)

# Inmultirea cu o valoarea scalara (asemanator pentru impartire)
mult_vect = vect * 3.5
print('Inmultirea cu o valoare scalara\n{} * 3.5 = '.format(vect),mult_vect)

# Numpy permite distribuirea unei operatii asupra intregului vector.
# Pentru ilustrarea acestei facilitati, mai intai se creaza un al doilea vector, de aceeasi dimensiune ca primul.
vect_2 = np.array([11,12,13,14])
print('vect_2 = ',vect_2)

# Adunarea celui de-al doilea vector
add_vect = vect + vect_2
print('Adunarea vectorilor\n{} + {} = '.format(vect,vect_2),add_vect)

# Inmultirea a doi vectori de aceeasi dimensiune
mult_vect = vect * vect_2
print('Inmultirea a doi vectori de aceeasi dimensiune\n{} * {} = '.format(vect,vect_2),mult_vect)
# Se poate observa ca, in acest caz, operatia de inmultire reprezinta inmultirea element cu element a celor doi vectori

# Numpy permite calculul produsului scalar a doi vectori
dot_prod = np.dot(vect,vect_2)
print('Produsul scalar a doi vectori\n{} .* {} = '.format(vect,vect_2),dot_prod)
```

Din moment ce am stabilit notiunile de baza, acum putem extinde la cazul 2D (matrici):

In [ ]:
```
# Importarea numpy
import numpy as np

# Definirea matricei
mat = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("mat =\n",mat)

# Aceleasi operatii cu un scalar pot fi definite si pe o matrice.
add_mat = mat + 1
mult_mat = mat * 2
print('Adunare\n{} + 1 = \n'.format(mat),add_mat)
print('Inmultire\n{} * 2 = \n'.format(mat),mult_mat)

# Cand adunam/inmultim doua matrici, rezultatul este o adunare/inmultire element cu element.
# Totusi, operatia np.dot duce la o inmultire intre doua matrici.
mat2 = np.array([[1,1,1],[2,2,2],[3,3,3]])
dot_prod = np.dot(mat,mat2)
print('Inmultire matriceala:\n{}.*\n{} = \n'.format(mat,mat2),dot_prod)

print(mat*mat2)
```

Uneori, programatorul are nevoie doar de anumite elemente ale unei matrici. In Python, indexarea matricelor incepe de la **0** ! De asemenea, cand specificam un interval de selectie, valoarea care se afla dupa **":"** nu este inclusa in interval. Aceste reguli pot fi aplicate si listelor, nu doar matricelor Numpy.
Cateva exemple de accesarea datelor dintr-o matrice:

In [ ]:
```
# Importare numpy
import numpy as np

mat = np.array([[1,2,3],[4,5,6],[7,8,9]])
# Obtinerea valorii de pe a 2a linie, a 2-a coloana (5):
print(mat[1,1]) # indexarea incepe de la 0

# Obtinerea valorilor de pe ultimele 2 linii si coloane (5,6,8,9):
print(mat[1:3,1:3]) # se observa ca elmentul dupa : (adica 3) nu este inclus in interval

# Obtinerea tuturor valorilor de pe prima linie (1,2,3):
print(mat[0,:])
```

In anumite situatii, este nevoie ca toate valorile care indeplinesc un criteriu sa fie modificate. Nu este nevoie sa iteram prin toata matricea, daca ne folosim de metodele de indexare din Numpy. Exemplu:

In [ ]:
```
# Importare numpy
import numpy as np

mat = np.array([[1,2,3],[4,5,6],[7,8,9]])
mat[mat >= 5] = 0
print(mat)
```

Alte functii si metode utile din Numpy:

In [ ]:
```
# Importare numpy
import numpy as np

# Definire vector
vect = np.array([0,1,2,3,4])

# Maxim/minim dintr-un vector
print(vect.max())
print(vect.min())

# Crearea unei matrici plina cu valori de 0
mat = np.zeros([2,2])
print(mat)
```

3. Aspecte suplimentare

Primul laborator de MLAV presupune si o trecere rapida prin biblioteca scikit-learn. In continuare vor fi prezentate cateva functii ajutatoare pentru aceasta sectiune a laboratorului:

- Functia de generare a baze de date de exercitiu *Moons*: **sklearn.datasets.make_moons(n_samples=100, shuffle=True, noise=None, random_state=None)**
- Functia de afisare a punctelor bazei de date: **matplotlib.pyplot.scatter(x1, x2, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, data=None)**
- Functia de afisare pe ecran a graficului creat: **matplotlib.pyplot.show()**
- Functia de generare a unui Random Forest: **rf = sklearn.ensemble.RandomForestClassifier(n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None)** ; **toate argumentele sunt initializate, deci puteti sa nu pasati nici unul**
- Functia de antrenare a clasificatorului: **rf.fit(X,Y)**
- Functia de predictie a clasificatorului antrenat: **rf.predict(X_test)**
- Functia care calculeaza preciza clasificatorului pe punctele prezise: **acc = sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)**

Mai intai, trebuie instalate bibliotecile in mediul nostru de Python:

conda install -c anaconda scikit-learn
conda install -c anaconda matplotlib

Desigur, bibliotecile si modulele trebuie importate. Va fi nevoie de urmatoarele:

from sklearn import datasets,ensemble,metrics
import matplotlib.pyplot as plt

Evident, datorita modului in care am importat modulele/functiile se schimba si apelurile. Ex: **matplotlib.pyplot.show()** devine **plt.show()**, **sklearn.datasets.make_moons()** devine **datasets.make_moons()** etc.

Exercitii

1. Generati 1200 de esantioane din setul de date `make_moons`. Reprezentati grafic aceste puncte, folosind `Matplotlib`.
2. Pastrati primele 1000 de esantioane ca set de antrenare si restul ca set de testare. Antrenati un clasificator de tip Random Forest si verificati performanta acestuia.
3. Inlocuiti tipul clasificatorului cu un SVM, iar apoi un MLP (trebuie sa vedeti in documentatia `scikit-learn` in ce module gasiti clasificatoarele).
4. Testati performantele si pentru alte seturi de date, fie generate, fie salvate in `scikit-learn`