Amusons nous en faisant de la data science en Python

Python est devenu un langage de programmation pour de nombreuses applications de science des données communément appelé "data science". Il combine la puissance des langages de programmation à usage général avec la facilité d'utilisation de langages de script spécifiques au domaine scientifique comme MATLAB ou R. Il a des bibliothèques pour le chargement des données, la visualisation, les statistiques, le traitement du langage naturel, traitement de l'image et bien plus ... . Cette vaste boîte à outils fournit aux scientifiques des données un large éventail de fonctionnalité. L'un des principaux avantages de l'utilisation de Python est sa capacité d'interagir directement avec le code, en utilisant un terminal ou d'autres outils comme le Jupyter Notebook. L'apprentissage automatique et l'analyse des données sont fondamentalement des processus itératifs, dans lesquels les données pilotent l'analyse. C'est essentiel pour ces processus d'avoir des outils qui permettent une itération rapide et une interaction facile. En tant que langage de programmation généraliste, Python permet également la création de interfaces graphiques (GUI) et services Web complexes, et est facile à intégrer dans un systèmes existants.

Si on désire installer plusieurs package Python, on utilise pip pour les installer dans la console d'anaconda. exemple: on désire enstaller "numpy, scipy, matplotlib, ipython, scikit-learn et pandas" $ pip install numpy scipy matplotlib ipython scikit-learn pandas

NumPy NumPy est l'un des paquets fondamentaux pour l'informatique scientifique en Python. Il contient des fonctionnalités pour les tableaux multidimensionnels, les fonctions mathématiques de haut niveau tels que les opérations d'algèbre linéaire, la transformée de Fourier, le générateurs de nombres pseudo-aléatoire.

Dans scikit-learn, le tableau NumPy est la structure de données fondamentale. Scikit-learn prend en données des tableaux de type NumPy. Ainsi donc, toutes les données que vous utilisez devront être converties en un tableau NumPy. La fonctionnalité principale de NumPy est la classe ndarray: c'est un tableau multidimensionnel (n-dimensionnel) et tous les éléments du tableau doivent être du même type. Un tableau NumPy ressemble à ceci:

In [9]:

```
import numpy as np # on importe la library "numpy" puis dans un objet nomm
é "np"

x = np.array([[1, 2, 3], [4, 5, 6]]) # on spécifie la library "numpy" à tr
avers son objet créé "np" suivie d'un "point" puis on appelle la fonction
 "array" qui permet de créer des matrices. ensuite, on saisie notre matric
e
# on ecrit la matrice en ecrivant chaque ligne dans une même acolales

print(x) # afficher juste le contenu du tableau "x"
```

```
[[1 2 3]
 [4 5 6]]
```

In [11]:

```
print("x_afficher :\n{}".format(x)) # afficher x en le précéder d'un nom
 "x_afficher"
```

```
x_afficher :
[[1 2 3]
 [4 5 6]]
```

SciPy SciPy est une collection de fonctions pour l'informatique scientifique en Python. Il offre, entre autres fonctionnalités, les routines d'algèbre linéaire avancées, fonction mathématique d'optimisation, de traitement du signal, fonctions mathématiques spéciales et distributions statistiques.

scikit-learn puise dans la collection de fonctions de SciPy pour la mise en œuvre de ses algorithmes. La partie la plus importante de SciPy pour nous est scipy.sparse: ceci fournit des matrices creuses, qui sont une autre représentation utilisée pour les données dans scikit-learn. Les matrices fragmentées sont utilisées chaque fois que nous voulons stocker un tableau en 2D contenant principalement des zéros:

In [12]:

```
from scipy import sparse # on entre dans le package "scipy" puis on import
e les fonctions de la library "sparse"

# Créons un tableau NumPy en 2D avec une diagonale de uns, et des zéros pa
rtout ailleurs
matrice_1_zero = np.eye(4)

print("NumPy array:\n{}".format(matrice_1_zero))
```

```
NumPy array:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [13]:

```
# Convertir le tableau NumPy en une matrice SciPy sparse au format CSR

# Seules les entrées non nulles sont stockées "c'est à dire seulement les
 "1" avec les indications de leurs indices de positionnement dans la matri
ce initiale

sparse_matrix = sparse.csr_matrix(matrice_1_zero)

print("\nSciPy sparse CSR matrix:\n{}".format(sparse_matrix))
```

```
SciPy sparse CSR matrix:
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
```

Voici une autre méthodes en utilisant le format COO:

In [14]:

```
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))

print("COO representation:\n{}".format(eye_coo))
```

```
COO representation:
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
```

matplotlib matplotlib est la principale bibliothèque de traçage scientifique en Python. Il fournit des fonctions pour la réalisation de visualisations de qualité telles que les graphiques linéaires, les histogrammes, la dispersion etc ... .

Lorsque vous travaillez à l'intérieur du carnet Jupyter, vous pouvez afficher les chiffres directement dans l'interface jupyter en utilisant les commandes "%matplotlib notebook" et "%matplotlib inline" .

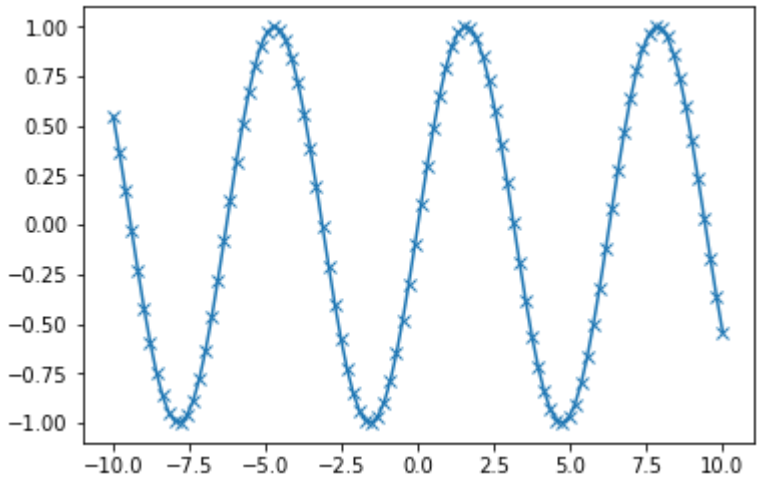
si vous n'écrivez pas dans un notebook mais plutot dans une console python directement, nous vous recommandons d'utiliser "%matplotlib notebook" , qui fournit un environnement interactif en vous ouvrant une boite de dialogue ou nouvelle fenetre contenant le graphique Mais si vous êtes dans un notebook, alors nous vous conseillons d'utiliser le "%matplotlib inline" qui va créer le graph dans votre notebook en cours d'utilisation sans ouvrire une nouvelle boite de dialogue. Comme ici, on travaille dans jupyter qui est un notebook de python, alors, on va utiliser le "inline"

In [15]:

```
%matplotlib inline

import matplotlib.pyplot as plt

# Génère une séquence de 100 nombres allant de -10 à 10

x = np.linspace(-10, 10, 100)

# Créer un deuxième tableau en utilisant sinus

y = np.sin(x)

# La fonction plot crée un graphique linéaire d'un tableau par rapport à u
n autre

plt.plot(x, y, marker="x")
```

Out[15]:

```
[<matplotlib.lines.Line2D at 0xd1c6908>]
```



pandas pandas est une librairie Python pour la recherche et l'analyse de données. Il est construit autour d'une donnée structure appelée DataFrame qui est modélisée après le R DataFrame. Tout simplement, un pandas DataFrame est une table, similaire à une feuille de calcul Excel. pandas fournit une bonne gamme de méthodes pour modifier et opérer sur cette table; en particulier, il permet les requêtes de jointures de tables "SQL-like". Contrairement à NumPy, qui exige que toutes les entrées dans

un "array" soit du même type "une matrice", les pandas permettent à chaque colonne d'avoir un type distinct (pour exemple, entiers, dates, nombres à virgule flottante et chaînes) comme les data.frame dans R. Un autre outil précieux fourni par pandas est sa capacité à ingérer une grande variété de formats de fichiers et de bases de données, comme les fichiers SQL, Excel et les fichiers CSV (valeurs séparées par des virgules). Exemple de création d'un DataFrame en utilisant un dictionnaire:

In [17]:

```
import pandas as pd

# créer un jeu de données simple de personnes

data = {'Name': ["John", "Anna", "Peter", "Linda"],

'Location' : ["New York", "Paris", "Berlin", "London"],

'Age' : [24, 13, 53, 33]

}

data_pandas = pd.DataFrame(data)

# IPython.display permet une "jolie impression" de dataframes dans le note book de Jupyter
from IPython.display import display

display(data_pandas)
```

	Age	Location	Name
0	24	New York	John
1	13	Paris	Anna
2	53	Berlin	Peter
3	33	London	Linda

Il existe plusieurs manières possibles d'interroger cette table. Exemple:

In [19]:

```
# Sélectionnez toutes les lignes dont la colonne d'âge est supérieure à 30
table_age_sup_30 = data_pandas[data_pandas.Age > 30]

display(table_age_sup_30)

print(table_age_sup_30)


# ou

display(data_pandas[data_pandas.Age > 30])
```

	Age	Location	Name
2	53	Berlin	Peter
3	33	London	Linda

```
   Age Location   Name
2   53   Berlin  Peter
3   33   London  Linda
```

	Age	Location	Name
2	53	Berlin	Peter
3	33	London	Linda

voir les Version de quelques library utilisé (compatibilité python 2 (2.7) ou 3(3.5), etc ...)

In [26]:

```
import sys

print("Python version: {}".format(sys.version))

import pandas as pd

print("pandas version: {}".format(pd.__version__))

import matplotlib

print("matplotlib version: {}".format(matplotlib.__version__))

import numpy as np

print("NumPy version: {}".format(np.__version__))

import scipy as sp

print("SciPy version: {}".format(sp.__version__))

import IPython

print("IPython version: {}".format(IPython.__version__))

import sklearn

print("scikit-learn version: {}".format(sklearn.__version__))
```

```
Python version: 3.6.3 |Anaconda custom (64-bit)| (default,
Oct 15 2017, 03:27:45) [MSC v.1900 64 bit (AMD64)]
pandas version: 0.20.3
matplotlib version: 2.1.0
NumPy version: 1.14.0
SciPy version: 0.19.1
IPython version: 6.1.0
scikit-learn version: 0.19.1
```

I/ Exploration des données et

manipulation de table

TP 1 : Exemple sur les pétales de fleurs d'IRIS (Exploration des données et manipulation de table)

on charges des données d'IRIS :

In [27]:

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

L'objet iris renvoyé par load_iris est un objet Bunch, ce qui est très similaire à un dictionnaire. Il contient des clés et des valeurs:

In [28]:

```
print("Keys of iris_dataset: \n{}".format(iris_dataset.keys()))
```

```
Keys of iris_dataset:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'fea
ture_names'])
```

La valeur de la touche DESCR est une courte description de l'ensemble de données. Nous montrons le début de la description :

In [32]:

```
print(iris_dataset['DESCR'][:193] + "\n...")
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classe
s)
    :Number of Attributes: 4 numeric, predictive att
...
```

La valeur de la clé target_names est un tableau de chaînes de charactère contenant les nom des espèces de fleur que nous voulons prédire (cible) :

In [30]:

```
print("Target names: {}".format(iris_dataset['target_names'])) # afficher
 les différente modalité de la variable "target_names".  Dans R i correspo
nd à la fonction "unique()"
```

```
Target names: ['setosa' 'versicolor' 'virginica']
```

In [35]:

```
print("Feature names: \n{}".format(iris_dataset['feature_names'])) # affic
her les différente modalité de la variable "feature_namess"
```

```
Feature names:
['sepal length (cm)', 'sepal width (cm)', 'petal length (c
m)', 'petal width (cm)']
```

In [34]:

```
print("Type of data: {}".format(type(iris_dataset['data']))) # afficher le
 type de notre table "data". Dans R i correspond à la fonction "class()"
```

```
Type of data: <class 'numpy.ndarray'>
```

In [36]:

```
print("Shape of data: {}".format(iris_dataset['data'].shape)) # afficher l
es dimensions de la table "data". affiche le nombre de ligne et le nombre
 de colonne de la table. Dans R, correspond à la fonction "dim()".
```

```
Shape of data: (150, 4)
```

Nous voyons que le tableau contient des mesures pour 150 fleurs différentes. Rappellons que les éléments individuels sont appelés des échantillons dans l'apprentissage automatique, et leurs propriétés sont appelées caractéristiques. La forme du tableau de données est le nombre d'échantillons multiplié par le nombre de fonctionnalités. Ceci est une convention dans scikit-learn, et vos données seront toujours être supposé être dans cette forme. Voici les valeurs des caractéristiques pour les cinq premiers échantillons (5 premiers individus) :

In [37]:

```
print("First five columns of data:\n{}".format(iris_dataset['data'][:5]))
# afficher les 5 premiers individus avec toutes les colonnes
```

```
First five columns of data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

In [38]:

```
print("Type of target: {}".format(type(iris_dataset['target']))) # R: clas
s(variable)
```

```
Type of target: <class 'numpy.ndarray'>
```

In [39]:

```
print("Shape of target: {}".format(iris_dataset['target'].shape)) # R: dim (variable)
```

```
Shape of target: (150,)
```

In [40]:

```
print("Target:\n{}".format(iris_dataset['target'])) # afficher la variable cible
```

```
Target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Les significations des nombres sont données par le tableau iris ['target_names']: 0 signifie setosa, 1 signifie versicolor et 2 signifie virginica.

Extration de la base d'entrainement et de la base test. La Fonction "train_test_split" de scikit-learn mélange l'ensemble de données et le scinde pour nous. Cette fonction extrait 75% des lignes des données en tant qu'ensemble d'apprentissage, avec les étiquettes correspondantes pour ces données. Les 25% restants des données, avec les étiquettes restantes forment l'ensemble de test.

In [43]:

```
from sklearn.model_selection import train_test_split # on importe la fonct
ion "train_test_split" de a library "sklearn.model_selection"
# on crée les table splité X_train, X_test, y_train, y_test
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
iris_dataset['target'], random_state=0)
```

In [44]:

```
print("X_train shape: {}".format(X_train.shape)) # on affiche les dimensio
ns de la table

print("y_train shape: {}".format(y_train.shape)) # on affiche les dimensio
ns de la table
```

```
X_train shape: (112, 4)
y_train shape: (112,)
```

In [46]:

```
print("X_test shape: {}".format(X_test.shape)) # on affiche les dimensions
 de la table

print("y_test shape: {}".format(y_test.shape)) # on affiche les dimensions
 de la table
```

```
X_test shape: (38, 4)
y_test shape: (38,)
```

In [49]:

```
print("Type of X_train: {}".format(type(X_train))) # l'objet table x_train
 est de type "ndarray" de numpy
```

```
Type of X_train: <class 'numpy.ndarray'>
```

In [50]:

```
print("iris_dataset.feature_names: {}".format(iris_dataset.feature_names))
```

```
iris_dataset.feature_names: ['sepal length (cm)', 'sepal w
idth (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [51]:

```
print("Type of iris_dataset.feature_names: {}".format(type(iris_dataset.feature_names))) # l'objet iris_dataset.feature_names est de type "list"
```

```
Type of iris_dataset.feature_names: <class 'list'>
```

In [63]:

```
# on crée une data.frame à partir de l'objet x_train qui est de type ndarray
# on choisi les libellé de la liste "iris_dataset.feature_names" comme nom de colonne

iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)

# créer des  scatter matrix sur la dataframe, et faire les paires de graphiques puis les colorer selon les modalités de "y_train"

# on charge les fonction de la library mglearn qui est util pour faire certains types des graphiques et photo
# import numpy as np
# import matplotlib.pyplot as plt
# import pandas as pd
# import mglearn
# l'importation des fonctions de mglearn situé à l'adresse https://github.com/amueller/introduction_to_ml_with_python/tree/master/mglearn ne marche pas.

# donc, on ne poura pas voir le graph

# grr = pd.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
#                         hist_kwds={'bins': 20}) #, s=60, alpha=.8, cmap=mglearn.cm3)
```

TP 2 : Classification supervisé avec les K-NN :

K-nn (k-Nearest Neighbors) : les K-plus proches voisins (en francais : kmean)

In [66]:

```
from sklearn.neighbors import KNeighborsClassifier # on importe les fonction de la library "KNeighborsClassifier" pour faire le k-nn
k=1 # on a choisi par exemple k=1 voisin, on pourais choisr plus, 3 plus proches voisins, 5 plus proches voisins, etc ...
knn = KNeighborsClassifier(n_neighbors=k) # avant de lancer le knn, on initialise ses paramètres d'abords, notemment le "k"
```

La méthode fit retourne l'objet knn lui-même (et le modifie en place), donc nous obtenons une représentation de chaîne de notre classificateur. La représentation nous montre quels paramètres ont été utilisés dans la création du modèle. Presque tous sont les valeurs par défaut, mais on peu voir aussi n_neighbors = 1, qui est le paramètre que nous avons passé lors de l'initialisation des parametres du modèle plus haut. La plupart des modèles scikit-learn ont de nombreux paramètres, mais la majorité d'entre eux sont soit des optimisations de vitesse ou des paramètres pour des cas d'utilisation très spéciaux.

In [67]:

```
knn.fit(X_train, y_train) # on lance le knn avec le knn.fit en insérant des tables numpy.ndarray bi-dimensionnel
```

Out[67]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Prédiction (predict) avec les knn

In [68]:

```
# exemple de données à prédire
X_new = np.array([[5, 2.9, 1, 0.2]])

print("X_new.shape: {}".format(X_new.shape))
```

```
X_new.shape: (1, 4)
```

In [70]:

```
prediction_exemple = knn.predict(X_new) # predict sur l'exemple créé
print("Prediction_exemple: {}".format(prediction_exemple))
print("Predicted_exemple target name: {}".format(iris_dataset['target_name
s'][prediction_exemple]))

prediction_test = knn.predict(X_test) # predict sur la table test
print("Prediction_test: {}".format(prediction))
print("Predicted_test target name: {}".format(iris_dataset['target_names']
[prediction_test]))
```

```
Prediction: [0]
Predicted target name: ['setosa']
Prediction: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0
0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
Predicted target name: ['virginica' 'versicolor' 'setosa'
'virginica' 'setosa' 'virginica'
 'setosa' 'versicolor' 'versicolor' 'versicolor' 'virginic
a' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'setosa' 'versicol
or' 'versicolor'
 'setosa' 'setosa' 'virginica' 'versicolor' 'setosa' 'seto
sa' 'virginica'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'setosa' 'vir
ginica'
 'versicolor' 'setosa' 'virginica' 'virginica' 'versicolo
r' 'setosa'
 'virginica']
```

prediction de l'echantillon test

In [71]:

```
y_pred = knn.predict(X_test)

print("Test set predictions:\n {}".format(y_pred))
```

```
Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1
1 0 2 1 0 2 2 1 0
 2]
```

evaluation du modèle

In [72]:

```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test))) # 97% de
 bonne prédiction
```

```
Test set score: 0.97
```

In [73]:

```
# on peut aussi coder comme ceci sans avoir besoins de lancer un code a pa
rt pour la prédiction:
print("Test set score: {:.2f}".format(knn.score(X_test, y_test))) # 97% de
 bonne prédiction
```

```
Test set score: 0.97
```

resumé du code pour la procédure des k-nn

In [74]:

```
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
iris_dataset['target'], random_state=0)
k=1
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```