stid_example_motivant

January 13, 2020

1 Exemple d'utilisation en science des données

[9]:
```
import pandas as pd
import numpy as np

import json

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

import pymongo
```

1.1 1. Gestion de jeux de données

1.1.1 Insérer le dataset Iris

[2]:
```
df = pd.read_csv('./data/iris.csv')
```

Le dataframe est en RAM

[6]:
```
df
```

[6]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
..           ...          ...          ...          ...          ...
145          6.7          3.0          5.2          2.3    virginica
146          6.3          2.5          5.0          1.9    virginica
147          6.5          3.0          5.2          2.0    virginica
148          6.2          3.4          5.4          2.3    virginica
149          5.9          3.0          5.1          1.8    virginica

[150 rows x 5 columns]
```

Le stocker dans Mongo

[10]:
```
client = pymongo.MongoClient('localhost', 27017)
stid_db = client["stid"]
datasets_coll = stid_db["datasets"]

json_string = df.to_json()
json_object = json.loads(json_string)
datasets_coll.insert_one(json_object)
```

[10]:
```
<pymongo.results.InsertOneResult at 0x7f2905bd5500>
```

1.1.2 Insérer un autre dataset

[12]:
```
df = pd.read_csv('./data/gdp-life-exp-2007.csv')
df
```

[12]:
```
     Unnamed: 0             country continent  population  life expectancy  \
0            11         Afghanistan      Asia  31889923.0           43.828
1            23             Albania    Europe   3600523.0           76.423
2            35             Algeria    Africa  33333216.0           72.301
3            47              Angola    Africa  12420476.0           42.731
4            59           Argentina  Americas  40301927.0           75.320
..          ...                 ...       ...         ...              ...
137        1655             Vietnam      Asia  85262356.0           74.249
138        1667  West Bank and Gaza      Asia   4018332.0           73.422
139        1679         Yemen, Rep.      Asia  22211743.0           62.698
140        1691              Zambia    Africa  11746035.0           42.384
141        1703            Zimbabwe    Africa  12311143.0           43.487

     gdp per capita
0         974.580338
1        5937.029526
2        6223.367465
3        4797.231267
4       12779.379640
..               ...
137      2441.576404
138      3025.349798
139      2280.769906
```

```
140     1271.211593
141      469.709298

[142 rows x 6 columns]
```

[11]:
```
df = pd.read_csv('./data/gdp-life-exp-2007.csv')
json_string = df.to_json()
json_object = json.loads(json_string)
datasets_coll.insert_one(json_object)
```

[11]:
```
<pymongo.results.InsertOneResult at 0x7f29064d2aa0>
```

[ ]:
```
cursor = datasets_coll.find()
datasets = list(cursor)
iris_dataset = datasets[0]
gdp_dataset = datasets[1]
iris_dataset
```

{'_id': ObjectId('5e0f4a2b17214ed279dabf87'), 'sepal_length': {'0': 5.1, '1': 4.9, '2': 4.7, ... , 'sepal_width': {'0': 3.5, '1': 3.0, ... }

1.1.3 Comment les retrouver par un critère pratique comme un nom ?

[20]:
```
datasets_coll.update_one( {"country" :  {"$exists" : True} }  , { "$set" : {␣
↪"name" : "GDP"} })
```

[20]:
```
<pymongo.results.UpdateResult at 0x7f2905b50d70>
```

[21]:
```
datasets_coll.update_one( {'sepal_length' :  {"$exists" : True} }  , { "$set" :␣
↪{ "name" : "IRIS"} })
```

[21]:
```
<pymongo.results.UpdateResult at 0x7f2905e51730>
```

[ ]:
```
o = datasets_coll.find_one({"name" : "GDP"})
o
```

1.1.4 Ajouter d'autres métadonnées (date de crétion, source, etc.)

[25]:
```
datasets_coll.update_one({"name" : "GDP"} , { "$set" : { "creation_date" : "03/
↪01/19"}})
datasets_coll.update_one({"name" : "GDP"} , { "$set" : { "source" : "./data/iris.
↪csv"}})
```

[25]:
```
<pymongo.results.UpdateResult at 0x7f2905e514b0>
```

1.2 2. Gestion d'expériences

Le preprocessing consiste a : - mettre a part la colonne target 'species' qui est sauvegardee das une variable y - creer un train et un test dataset

[12]:
```
# df.drop('Id', axis=1, inplace=True)

X = df.iloc[:,:-1].values
y = df['species']
```

Create the train and test sets y_train sera utilise pour entrainer le classifieur et y_test pour evaluer ses predictions.

120 individus pour entrainer et 30 pour tester

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=27)
```

[21]:
```
SVC_model = SVC( gamma='scale') # sum.SVC
# KNN model requires you to specify n_neighbors,
# 5 points will be looked at to determine what class a test point belongs to
KNN_model = KNeighborsClassifier(n_neighbors=5)
```

Entrainer

[22]:
```
SVC_model.fit(X_train, y_train)
KNN_model.fit(X_train, y_train)
```

[22]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[ ]:
```
Predire
```

[26]:
```
svm_y_predicted = SVC_model.predict(X_test)
knn_y_predicted = KNN_model.predict(X_test)
```

[ ]:
```
Evaluer les predictions
```

[32]:
```
print(accuracy_score(svm_y_predicted, y_test))
print(accuracy_score(knn_y_predicted, y_test))

print(confusion_matrix(svm_y_predicted, y_test))
print(classification_report(svm_y_predicted, y_test))
# lignes = ground-truth et colonnes = predictions
# pour virginica :
# la 3eme ligne [ 0  1 11] montre le rappel  TP / (TP+FN) = 11 / (11+1)
# et la 3eme colonne [ 0  1 11]^T montre la precision TP / (TP+FP) 11 / (11+1)
# Le 1 de la ligne 2 est un faux positif et le 1 de la ligne 3 est un faux␣
↪negatif
```

```
0.9333333333333333
0.9666666666666667
[[ 7  0  0]
 [ 0 10  1]
 [ 0  1 11]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
```

```
  versicolor       0.91      0.91      0.91        11
   virginica       0.92      0.92      0.92        12

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```

1.2.1 Stocker la description de l'expérimentation

[27]:
```
iris_experiment = {"dataset_name" : "GDP",
                   "preprocessing" : {},
                  "classifier" : {"name" : "klearn.neighbors.KNeighborsClassifier",
                                  "parameters" : {"n_neighbor" : "5"},
                                   "test_size" : 0.20
                                  },
                   "results" : {"accuracy" : 0.933}
                 }
```

[28]:
```
stid_db = client["stid"]
experiments_coll = stid_db["experiments"]
experiments_coll.insert_one(iris_experiment)
```

[28]:
```
<pymongo.results.InsertOneResult at 0x7f2905e6f1e0>
```