M1-ISL : option Initiation à la Recherche. Résultats de recherche : code, analyse et reproductibilité

Jean-François Couchot

Université de Franche-Comté, UFR-ST

Plan

Publication de code

Codes pour la recherche en IA

Extensions pour la recherche

Plan

Google Colaboratory

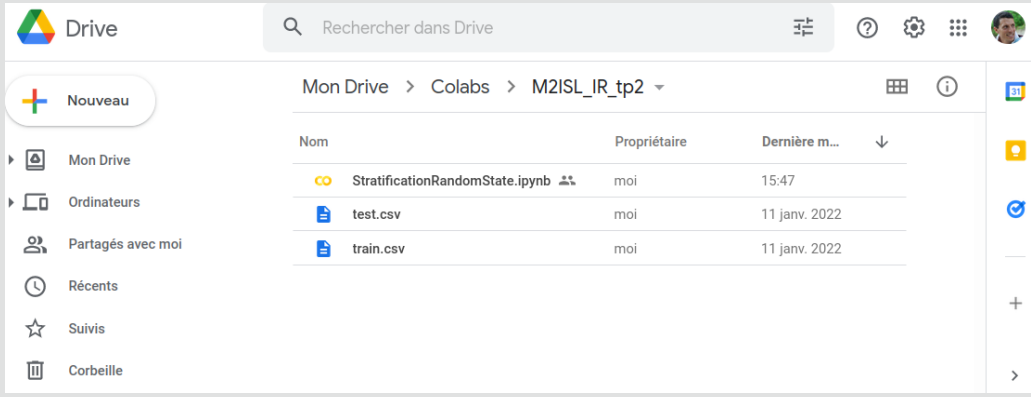
Obj. : Publier des codes de recherche

- Tout code python
- Important ou pas des bibliothèques externes
- Avec résultats ou pas d'évaluation
- Avec possibilité de partage
- Mais nécessité d'avoir un compte gmail
- Mais partage (code et ev. jeu de données) avec Google.

Lien avec Google Drive

Préalable : données dans un répertoire de Google Drive

- Par ex : contenu de `Colabs/M2ISL_IR_tp2/`



Import des données depuis Google Drive

```
from google.colab import drive
import pandas as pd

drive.mount('/content/drive')
path = "/content/drive/MyDrive/Colabs/M2ISL_IR_tp2/"
train_all = pd.read_csv(path+'train.csv')
test_all = pd.read_csv(path+'test.csv')
```

Du code séparé en cellules

Une cellule de commentaires

```
**Pour importer des données depuis Google Drive**

On monte le dossier drive, puis on lit les fichiers csv
```

Des cellules de code et leurs sorties

```
#Cell 2                                                  # première distribution
train_all.Survived.value_counts()/train_all.shape[0]     frequence des survivants/total
0    0.616162                                            62\% de morts
1    0.383838                                            38\% de survivants

#Cell 3
from sklearn.model_selection import train_test_split
X = train_all.drop('Survived', axis = 1)
y = train_all.Survived
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = 163035)

#Cell 4
y_train.value_counts() / y_train.shape[0]                # distrib. aléatoire dans y_train
0    0.577247                                            58\% de morts
1    0.422753                                            42\% de survivants

#Cell 5
y_test.value_counts() / y_test.shape[0]                  # distrib. aléatoire dans y_test différente
0    0.77095                                             77\% de morts
1    0.22905                                             23\% de survivants
```

Plan

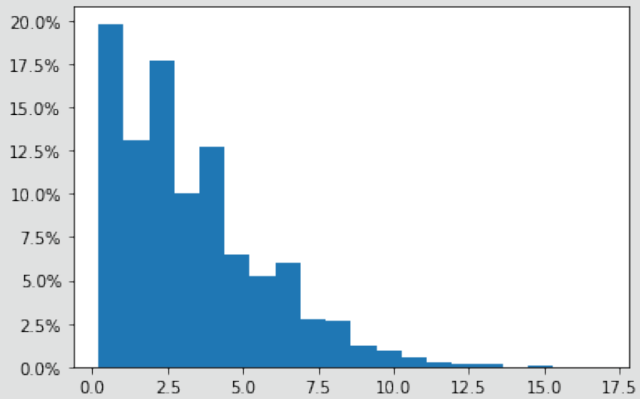
Ensembles efficaces/honnêtes pour l'apprentissage

Choix aléatoire des ensemble train/test : disparité possible entre eux

```
from matplotlib import pyplot as plt
from matplotlib.ticker import PercentFormatter
import numpy as np

mn, idx, l= 0,-1,[]
for rs in range(10000):
  X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size = 0.2,random_state = rs)
  dif = abs((y_train.value_counts() / y_train.shape[0])[0]
            - (y_test.value_counts() / y_test.shape[0])[0])
  l.append(dif)
  if mn < dif :
    idx,mn = rs,dif

plt.hist(l, bins=20,weights=np.ones(len(l)) / len(l))
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))
plt.show()
```



Solution : ensembles stratifiés avec random state

```
    # Split avec distribution équilibrées
    X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size = 0.2,random_state = 20220115,stratify = y)
```

Mise en place de l'apprentissage

Prétraitement usuel des données

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
X = X[['Pclass', 'Sex', 'SibSp', 'Fare']]

# The "Sex" variable is a string and needs to be one-hot encoded
X['gender_dummy'] = pd.get_dummies(X.Sex)['female']
X = X.drop(['Sex'], axis = 1)
X_train, X_test, y_train, y_test =
  train_test_split(X, y, test_size = 0.2, random_state = 20220115, stratify = y)
```

Apprentissage reproductibe (random_state fixe)

```
clf = RandomForestClassifier(n_estimators = 50, random_state = 0)
clf = clf.fit(X_train, y_train)
preds = clf.predict(X_test)
round(accuracy_score(y_true = y_test, y_pred = preds), 3)
# 0.793, mais quelle est la moyenne, la variance?
```

Evaluation de l'apprentissage

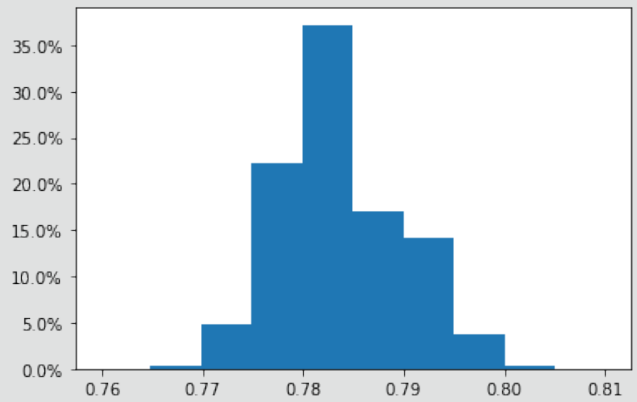
Statistiques d'apprentissage

```
mne,idxe, leclf= 0,-1,[]
for rs in range(int(1E4)):
  clf = RandomForestClassifier(n_estimators = 50,
       random_state = rs)
  clf = clf.fit(X_train, y_train)
  preds = clf.predict(X_test)
  sc = accuracy_score(y_true = y_test, y_pred = preds)
  leclf.append(sc)
  if sc > mne :
  idx,mne = rs,sc

from matplotlib import pyplot as plt
from matplotlib.ticker import PercentFormatter
import numpy as np

plt.hist(leclf,weights=np.ones(len(leclf)) / len(leclf))
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))
plt.show()

res_series = pd.Series(leclf)
res_series.describe()
# résultats à droite ->
```




```
count    10000.000000
mean         0.783520
std          0.006885
min          0.759777
...
max          0.810056
```

Plan

Des biblothèques meilleures que notre code

Installation (d'une bibliothèque de PVP)

```
!pip install diffprivlib
```

Exemple d'exploitation de cette bibliothèque

```
import diffprivlib as dp
dpclf = dp.models.RandomForestClassifier(n_estimators = 50, random_state = 0,epsilon=0.1)
dpclf = dpclf.fit(X_train, y_train)
preds = dpclf.predict(X_test)
round(accuracy_score(y_true = y_test, y_pred = preds), 3)
#0.666
```

Évaluer des fonctions numériques

Fonction à évaluer

```
def une_eval_avec_dp_random_forest(rs):
  dpclf = dp.models.RandomForestClassifier( n_estimators = 50, epsilon=0.1,random_state=rs)
  dpclf = dpclf.fit(X_train, y_train)
  preds = dpclf.predict(X_test)
  return accuracy_score(y_true = y_test, y_pred = preds)
```

Schéma d'évaluation

```
def evaluation(func,nbexp):
  s = pd.Series(np.arange(nbexp))
  sp = s.apply(func)
  return sp

sp = evaluation(une_eval_avec_dp_random_forest,100)
sp.describe()
...
mean       0.665531
std        0.070180
...
```

Export de code

De Google Colab à Github

- File > Save a copy in Github