1 Introduction

Présentation de la librairie ''Orange'' pour la pratique du machine learning sous Python 3.

Je connais et apprécie le logiciel Orange, de l'Université de Ljubljana, en Slovénie, depuis longtemps. En recherchant dans mes archives, j'ai retrouvé une étude de 2005, où je me posais la question de l'intérêt et de la pertinence d'utiliser des logiciels libres pour l'enseignement du data mining dans nos formations (''Logiciels gratuits de data mining pour l'enseignement'', décembre 2005), Orange en faisait partie. Fait rare, j'ai même directement référencé une version du document de présentation du logiciel parce que je le trouvais particulièrement bien écrit (''Data Mining avec Orange'', octobre 2012). Plus récemment, il figurait parmi les outils que j'ai demandé aux étudiants du Master SISE d'explorer en montant des études de cas (''Etude des logiciels de data science'', octobre 2016).

Tous ces documents, et la plupart de ceux que l'on trouve sur le net, mettent en avant la version interactive du logiciel où nous définissons graphiquement les traitements sous la forme de flux d'opérations appliquées aux données (workflow). On parle moins en revanche de la possibilité d'utiliser les fonctions de la librairie Orange dans des programmes rédigés en Python. Pourtant, la fonctionnalité est disponible depuis longtemps, bien largement avant la vague Python dans la pratique du machine learning de ces dernières années. Mais elle est peu connue, peu utilisée à ma connaissance. Comme quoi, avoir raison (trop) tôt n'est pas un gage de succès.

Dans ce tutoriel, je montre son mode opératoire dans un problème simple d'apprentissage supervisé. Nous constaterons que le package Orange pour Python est assez simple d'utilisation et, dans ce cadre, se pose comme une alternative tout à fait valable aux libraires très populaires telles que ''scikit-learn'' ou le tandem ''tensorflow / keras''.

2 Installation et chargement du package

Je suis sous Windows 10 Education 64 bits. Je travaille avec Python 3.7.3 de la distribution Anaconda dans ce tutoriel. Le gestionnaire de package ''conda'' m'a permis d'installer le package Orange (https://orange.biolab.si/download/#windows). Tout s'est parfaitement déroulé. Je l'ai ensuite chargé et vérifié le numéro de version.

```
#importation de la librarie
import Orange
#version du package
print(Orange.__version__)
```

```
3.23.1
```

Nous disposons de la version **3.23.1** de la librairie Orange.

3 Importation des données

3.1 Données

Nous utilisons les données archi-connues ''Breast Cancer Wisconsin'' où l'objectif est de prédire la nature cancéreuse (classe = malignant) ou pas (classe = begnin) de cellules mamellaires à partir de leurs caractéristiques (épaisseur, taille, etc.). Nous connaissons parfaitement la teneur des résultats dans un schéma prédictif où nous utiliserons une régression logistique. Cela nous permettra de vérifier le bon fonctionnement de la libraire.

La base a été subdivisée en amont en échantillons d'apprentissage (''**breast_train.xlsx**'', 399 observations) et de test (''**breast_test.xlsx**'', 300). Voici les 6 premières lignes de la base d'apprentissage. Nous disposons de 9 descripteurs (''clump''…''mitoses'') ; la variable cible ''classe'' est placée en dernière position.

clump	ucellsize	ucellshape	mgadhesion	sepics	bnuclei	bchromatin	normnucl	mitoses	classe
4	1	1	1	2	1	1	1	1	begnin
3	3	2	1	3	1	3	6	1	begnin
10	9	7	3	4	2	7	7	1	malignant
10	10	10	7	10	10	8	2	1	malignant
10	8	8	4	10	10	8	1	1	malignant
2	1	1	1	2	1	1	1	1	begnin

Figure 1 - Les 6 premières lignes de l'échantillon d'apprentissage

3.2 Chargement des données

Dans notre programme Python, on chargeons le fichier d'apprentissage via le module Table du package Orange. Il peut prendre en charge différents formats de fichiers tabulaires, dont les fichiers CSV (texte) et XLSX (Excel).

```
#modifier le dossier de travail
import os
os.chdir("... votre dossier ...")

#charger le fichier de données
D1 = Orange.data.Table("breast_train.xlsx")

#type de l'objet
print(type(D1))
```

```
<class 'Orange.data.table.Table'>
```

L'outil retourne un type d'objet qui est spécifique à Orange, auquel sont associés un certain nombre de propriétés et méthodes.

```
#propriétés
print(dir(D1))
```

```
['DENSE', 'MISSING', 'SPARSE', 'SPARSE_BOOL', 'W', 'X', 'X_density', 'Y',
'Y_density', '_Table__determine_density', '_Y', '__abstractmethods__', '__class__',
'__contains__', '__delattr__', '__delitem__', '__dict__', '__dir__', '__doc__',
'__eq__', '__file__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__setattr__', '__setitem__',
'__sizeof__', '__slots__', '__str__', '__subclasshook__', '__weakref__',
'_abc_impl', '_check_all_dense', '_compute_basic_stats', '_compute_contingency',
'_compute_distributions', '_continuous_filter_to_indicator',
'_discrete_filter_to_indicator', '_filter_has_class', '_filter_is_defined',
'_filter_random', '_filter_same_value', '_filter_to_indicator', '_filter_values',
'_init_ids', '_next_instance_id', '_next_instance_lock',
'_range_filter_to_indicator', '_resize_all', '_set_row',
'_string_filter_to_indicator', '_values_filter_to_indicator', 'append',
'approx_len', 'attributes', 'checksum', 'clear', 'columns', 'concatenate', 'copy',
'count', 'domain', 'ensure_copy', 'extend', 'from_domain', 'from_file',
'from_list', 'from_numpy', 'from_table', 'from_table_rows', 'from_url',
'get_column_view', 'get_nan_frequency_attribute', 'get_nan_frequency_class',
'has_missing', 'has_missing_attribute', 'has_missing_class', 'has_weights', 'ids',
'index', 'insert', 'is_copy', 'is_sparse', 'is_view', 'metas', 'metas_density',
'n_rows', 'name', 'new_id', 'pop', 'remove', 'reverse', 'save', 'set_weights',
'shuffle', 'to_dense', 'to_sparse', 'total_weight', 'transform', 'transpose']
```

Nous pouvons afficher la liste des variables qui le compose.

```
#afficher la liste des variables
print(D1.domain)
```

```
[clump, ucellsize, ucellshape, mgadhesion, sepics, bnuclei, bchromatin, normnucl,
mitoses, classe]
```

Le nombre d'observations.

```
#dimension 1 -> nb. d'obs.
print(len(D1))
```

```
399
```

Le nombre de variables.

```
#dimension 2 -> nb. de colonnes
print(len(D1[0]))
```

```
10
```

Les premières lignes du tableau de données d'apprentissage…

```
#afficher les 6 premières lignes
print(D1[:6])
```

```
[[4.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, begnin],
 [3.0, 3.0, 2.0, 1.0, 3.0, 1.0, 3.0, 6.0, 1.0, begnin],
 [10.0, 9.0, 7.0, 3.0, 4.0, 2.0, 7.0, 7.0, 1.0, malignant],
 [10.0, 10.0, 10.0, 7.0, 10.0, 10.0, 8.0, 2.0, 1.0, malignant],
 [10.0, 8.0, 8.0, 4.0, 10.0, 10.0, 8.0, 1.0, 1.0, malignant],
 [2.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, begnin]]
```

… qui concordent bien, heureusement, avec la copie ci-dessus (Figure 1).

3.3 Organisation des données

Il faut spécifier les variables cibles et prédictives. Pour cela, nous définissons un nouveau ''domain'' si l'on se réfère à la terminologie Orange. Les 9 premières variables correspondent aux descripteurs, la 10ème (indice 9 puisque la numérotation commence à zéro) est la variable cible.

```
#re-définition du rôle des variables pour le problème à traiter
breast_domain = Orange.data.Domain(D1.domain[:9],D1.domain[9])
print(breast_domain)
```

```
[clump, ucellsize, ucellshape, mgadhesion, sepics, bnuclei, bchromatin, normnucl,
mitoses | classe]
```

La variable ''classe'' est toujours en dernière position dans la description, mais on observe l'apparition de l'opérateur '' | '' qui permet de la distinguer des autres.

Appliqué aux données, nous observons maintenant le statut particulier de la dernière colonne :

```
#nouveau dataset avec les rôles redéfinis
DTrain = Orange.data.Table(breast_domain,D1)

#vérification
print(DTrain[:6])
```

```
[[4.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0 | begnin],
 [3.0, 3.0, 2.0, 1.0, 3.0, 1.0, 3.0, 6.0, 1.0 | begnin],
 [10.0, 9.0, 7.0, 3.0, 4.0, 2.0, 7.0, 7.0, 1.0 | malignant],
 [10.0, 10.0, 10.0, 7.0, 10.0, 10.0, 8.0, 2.0, 1.0 | malignant],
 [10.0, 8.0, 8.0, 4.0, 10.0, 10.0, 8.0, 1.0, 1.0 | malignant],
 [2.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0 | begnin]]
```

Nous pouvons accéder explicitement à la variable cible…

```
#vérif. variable cible
print(DTrain.domain.class_var)
```

```
classe
```

… et en afficher les valeurs.

```
#liste des modalités de la cible
print(DTrain.domain.class_var.values)
```

```
['begnin', 'malignant']
```

La première modalité (n°**0**, c'est important pour la suite) est 'begnin' ; la seconde (n°**1**) est 'malignant'. Nous calculons la distribution de fréquences des valeurs.

```
#comptage des modalités de la cible
import numpy
numpy.unique(DTrain.Y,return_counts=True)
```

```
(array([0., 1.]), array([267, 132], dtype=int64))
```

Nous remarquons les codes associés aux modalités. Il y a 267 observations 'begnin' (codé **0**) et 132 'malignant' (codé **1**) dans la base d'apprentissage.

Nous pouvons aussi vérifier la liste des variables explicatives.

```
#liste des variables explicatives
print(DTrain.domain.attributes)
```

```
(ContinuousVariable(name='clump', number_of_decimals=1),
ContinuousVariable(name='ucellsize', number_of_decimals=1),
ContinuousVariable(name='ucellshape', number_of_decimals=1),
ContinuousVariable(name='mgadhesion', number_of_decimals=1),
ContinuousVariable(name='sepics', number_of_decimals=1),
ContinuousVariable(name='bnuclei', number_of_decimals=1),
ContinuousVariable(name='bchromatin', number_of_decimals=1),
ContinuousVariable(name='normnucl', number_of_decimals=1),
ContinuousVariable(name='mitoses', number_of_decimals=1))
```

Orange fournit en sus le type de chaque variable.

4 Modélisation

Nous souhaitons utiliser la régression logistique. Comme sous ''scikit-learn'', nous travaillons en deux temps : (1) il faut instancier l'objet d'apprentissage (le learner), …

```
#instancier une régression logistique
learner = Orange.classification.LogisticRegressionLearner()
print(learner)
```

```
logistic regression
```

… (2) qu'il faut ensuite appliquer aux données pour obtenir le classifieur (classifier).

```
#apprentissage sur data
classifier = learner(DTrain)
print(classifier)
```

```
LogisticRegressionClassifier(skl_model=LogisticRegression(C=1.0,  class_weight=None,
dual=False, fit_intercept=True,
                  intercept_scaling=1, l1_ratio=None, max_iter=100,
                  multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                  solver='liblinear', tol=0.0001, verbose=0, warm_start=False))  #
params={'penalty': 'l2', 'dual': False, 'tol': 0.0001, 'C': 1.0, 'fit_intercept':
True, 'intercept_scaling': 1, 'class_weight': None, 'random_state': None, 'solver':
'liblinear', 'max_iter': 100, 'multi_class': 'ovr', 'verbose': 0, 'n_jobs': 1}
```

L'objet affiche les propriétés de l'algorithme d'apprentissage. La description des paramètres est accessible en ligne, nous notons qu'une régression ridge est proposée par défaut (penalty = 'l2') (https://docs.biolab.si//3/data-mining-library/reference/classification.html#logistic-regression).

Nous pouvons afficher les membres de l'objet :

```
#membres de l'objet
dir(classifier)
```

```
['Probs',
 'Value',
 'ValueProbs',
 '__call__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_repr_pretty_',
 '_reprable_fields',
 '_reprable_items',
 '_reprable_module',
 '_reprable_omit_param',
 'coefficients',
 'domain',
 'intercept',
 'name',
 'original_data',
 'original_domain',
 'params',
 'predict',
 'predict_storage',
 'skl_model',
 'supports_multiclass',
 'supports_weights',
 'used_vals']
```

S'agissant de la régression logistique, nous remarquons en particulier les champs 'intercept' et 'coefficients' qui portent les coefficients estimés du modèle.

Voici donc la constante (intercept) de l'équation de régression estimée, …

```
#constante
print(classifier.intercept)
```

```
[-5.66466353]
```

… et les coefficients.

```
#coefficients
print(classifier.coefficients)
```

```
[[ 0.29026781  0.15415504  0.23569909  0.06830692 -0.12276859  0.31506304
   0.17951231  0.21365977  0.12810439]]
```

Pouvoir associer ces coefficients aux variables, c'est mieux. Nous passons par un data.frame Pandas pour ce faire.

```
#coefficients avec les noms des variables
cn = {'variable': [d.name for d in DTrain.domain.attributes],
'coefficient':classifier.coefficients[0]}

#affichage
import pandas
print(pandas.DataFrame.from_dict(cn))
```

```
     variable  coefficient
0       clump     0.290268
1   ucellsize     0.154155
2  ucellshape     0.235699
3  mgadhesion     0.068307
4      sepics    -0.122769
5     bnuclei     0.315063
6  bchromatin     0.179512
7    normnucl     0.213660
8     mitoses     0.128104
```

5 Evaluation sur l'échantillon test

L'étape suivante consiste à évaluer la qualité de la prédiction sur l'échantillon test.

5.1 Chargement et préparation

Nous chargeons le fichier test, nous redéfinissons le domaine de l'ensemble de données pour être raccord avec le schéma prédictif de la phase d'apprentissage (prédire ''classe'' à l'aide des autres variables). Nous comptabilisons les modalités de la variable cible pour vérification.

```
#chargement de l'échantillon test
D2 = Orange.data.Table("breast_test.xlsx")

#re-définition du rôle des variables en accord avec la phase d'apprentissage
DTest = Orange.data.Table(breast_domain,D2)

#comptage des modalités de la cible
numpy.unique(DTest.Y,return_counts=True)
```

```
(array([0., 1.]), array([191, 109], dtype=int64))
```

L'échantillon test comporte 300 (191 + 109) observations, avec 191 'begnin' et 109 'malignant'.

5.2 Prédiction en test

Nous appliquons le classifieur sur cet échantillon test.

```
#prédiction
prediction = numpy.array([classifier(DTest,0)])
print(prediction)
```

```
[[0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0.
  1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1.
  0. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1.
  0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 1.
  0. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0.
  1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1.
  1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
  1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0.
  0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0.
  0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1.
  1. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0.]]
```

Le processus a fonctionné, aucun doute là-dessus. Mais nous n'avons aucune expertise sur les résultats à ce stade.

Déjà, dénombrons les prédictions 'begnin' et 'malignant'.

```
#comptage des prédictions 0 et 1
print(numpy.unique(prediction[0],return_counts=True))
```

```
(array([0., 1.]), array([194, 106], dtype=int64))
```

194 observations ont été prédites 'begnin', 106 'malignant'. Ces informations, ainsi que les fréquences des classes dans l'échantillon test ci-dessus, seront précieuses lorsque nous aurons à inspecter la matrice de confusion confrontant les classes observées et prédites.

5.3 Matrice de confusion et indicateurs de performances

Nous calculons la matrice de confusion avec la fonction crosstab() du package Pandas.

```
#matrice de confusion
mc = pandas.crosstab(DTest.Y,prediction[0])
print(mc)
```

```
col_0  0.0  1.0
row_0
0.0    188    3
1.0      6  103
```

La présentation n'est pas très avenante certes, mais nous avons les bons résultats. En effet, les sommes en ligne (classes observées) et en colonne (prédites) sont tout à fait cohérentes avec ce que nous avons pu calculer ci-dessus. Ce genre de vérifications est toujours bon à prendre dans

les études. Les erreurs de manipulations arrivent si facilement. Un résultat doit toujours être croisé avec un autre pour s'assurer de sa légitimité.

Orange propose un outil dédié pour la confrontation entre classes observées et prédites. Nous n'évaluons qu'un seul classifieur ici, mais son utilisation est plus large et permet d'éprouver la qualité de plusieurs prédictions (issues de plusieurs classifieurs).

```
#Orange - confrontation classes observées et prédites
res = Orange.evaluation.testing.Results(actual=DTest.Y,predicted=prediction,domain=breast_domain)
```

L'objet ''res'' qui en découle peut être utilisé pour obtenir les indicateurs usuels de performance.

Le taux de succès (taux de reconnaissance) :

```
#taux de reconnaissance
print(Orange.evaluation.CA(res))
```

```
[0.97]
```

Qui correspond à (188 + 103) / 300.

Remarque : Ce résultat est conforme à ce que l'on observe dans la littérature lorsqu'on applique la régression logistique sur ce jeu de données. C'est toujours rassurant de le savoir.

Le rappel ou sensibilité, correspondant à 103 / (103 + 6) car 'malignant' est la modalité codée 1.

```
#print rappel
print(Orange.evaluation.Recall(res))
```

```
[0.94495413]
```

La précision avec 103 / (103 + 3),

```
#print précision
Orange.evaluation.Precision(res)
```

```
array([0.97169811])
```

Et le F1-Score.

```
#F1-Score
print(Orange.evaluation.F1(res))
```

```
[0.95813953]
```

6 Autres algorithmes – Induction de règles

Les algorithmes d'induction de règles sont les grands absents des packages récents de machine learning qui, pour la plupart, font une fixette quasi-hystérique sur le deep learning. Pourtant ils (ces algorithmes) répondent à un vrai besoin. Les règles prédictives présentent l'avantage, entres autres, d'être facilement interprétables.

Orange propose l'algorithme CN2 (Clark et Niblett, 1989) qui permet de produire une base de règles ordonnées (les règles sont testés séquentiellement en prédiction) ou non-ordonnées (toutes les règles sont testées) avec un procédé relativement simple mais efficace. Les variables prédictives quantitatives sont automatiquement discrétisées.

Nous appliquons CN2 (règles non-ordonnées) sur notre base d'apprentissage en spécifiant un nombre minimal de 20 individus par règles pour éviter la sur-spécialisation. L'objet classifieur possède un certain nombre de propriétés…

```
#algorithme CN2
cn2_learner = Orange.classification.CN2UnorderedLearner()
#paramétrage - nombre d'observations min. couvertes par chaque règle
cn2_learner.rule_finder.general_validator.min_covered_examples = 20

#apprentissage
cn2_classifier = cn2_learner(DTrain)
print(dir(cn2_classifier))
```

```
['Probs', 'Value', 'ValueProbs', '__call__', '__class__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', '_repr_pretty_', '_reprable_fields',
'_reprable_items', '_reprable_module', '_reprable_omit_param', 'domain', 'name',
'ordered_predict', 'original_data', 'original_domain', 'predict',
'predict_storage', 'rule_list', 'supports_multiclass', 'supports_weights',
'unordered_predict', 'used_vals']
```

… dont la liste des règles que nous nous empressons d'afficher.

```
#affichage des règles
for rule in cn2_classifier.rule_list:
    print(rule)
```

```
IF bnuclei<=2.0 AND ucellsize<=4.0 AND sepics<=5.0 THEN classe=begnin
IF clump<=4.0 AND ucellsize<=4.0 AND bnuclei<=5.0 THEN classe=begnin
IF clump>=7.0 AND bnuclei>=2.0 THEN classe=malignant
IF bchromatin>=4.0 AND bnuclei>=9.0 THEN classe=malignant
IF ucellshape>=3.0 AND normnucl>=4.0 THEN classe=malignant
IF TRUE THEN classe=begnin
```

N'y connaissant rien, je ne peux rien dire en ce qui me concerne. Mais j'imagine qu'un expert du domaine saura tirer profit du caractère explicite des règles de désignation des classes, et pourra les interpréter en les mettant en relation avec les connaissances métiers. C'est tout l'intérêt des modèles prédictifs basés sur des règles logiques.

Pour évaluer la qualité du modèle, nous calculons les prédictions en test et mesurons le taux de reconnaissance via les concordances entre classes observées et prédites.

```
#prédiction en test
cn2_pred = cn2_classifier(DTest)

#taux de reconnaissance - concordance obsv. vs. préd.
print(numpy.mean(DTest.Y == cn2_pred))
```

Il est de **95%**, légèrement moins bon sur cet exemple que la régression logistique (97%).

7 Conclusion

''Qu'importe le flacon pourvu que l'on ait l'ivresse'', j'ai toujours aimé ce dicton (on se demande pourquoi). Les packages telles que ''scikit-learn'' ou ''tensorflow'' font référence et occupent le devant de la scène sous Python. Très bien. Mais il ne faut pas oublier qu'il existe des alternatives, qui sont aussi de qualité. Il ne tient qu'à nous que de les explorer.

Dans ce tutoriel, nous avons testé le paquetage ''Orange'' pour Python 3. Nous constatons qu'il propose des fonctionnalités en adéquation avec la pratique usuelle du data mining. Elles sont assez similaires à celles de ''scikit-learn'' finalement. Ce n'est pas vraiment étonnant. Il n'y a pas 10.000 manières de mener un processus de data mining.

8 Références

Demsar J, Curk T, Erjavec A, Gorup C, Hocevar T, Milutinovic M, Mozina M, Polajnar M, Toplak M, Staric A, Stajdohar M, Umek L, Zagar L, Zbontar J, Zitnik M, Zupan B (2013), ''Orange: Data Mining Toolbox in Python'', Journal of Machine Learning Research 14(Aug): 2349−2353.