

Rapport écrit

Nom du groupe : INFO groupes 2bis/3/3bis/PMMI

Nom de l'équipe : Vanilla

Codalab Login utilisé: vanilla

Membres de l'équipe : HOUMEL IDIR-MESSALI NASSIM (binôme 3), DEHLINGER CHARLES-ELYES KHERFI (binôme 2), ASSEME REGIS-CYRILLE N'GUESSAN-AISSATA BALDE (binôme 1)

Challenge URL : https://competitions.codalab.org/competitions/16151?secret_key=5d5dc04d-561a-47a3-b0d4-e9d50006569d

Github repo de l'équipe : <https://github.com/groupe-vanilla/mini-projet>

Numéro de la soumission Codalab: 399286

La performance sur le leaderboard: 0.2041

L'URL de la vidéo : <https://youtu.be/B0ZctjzgC8Q>

Brève présentation :

L'objectif du projet est de mettre en place un système de recommandation personnalisé basé sur les données du célèbre jeu Movielens et cela à l'aide d'outils d'apprentissage automatique. Le système doit prévoir pour un utilisateur et un film donné le score qui est le plus susceptible d'être attribué par l'utilisateur (score $\in [0 ; 5]$). Après étude des données fournies dans le starting kit on peut voir que chaque utilisateur et chaque film est identifié par un unique "id". De plus les données nous donnent des informations sur l'âge (répartis en 7 groupes), le métier et le genre de chaque utilisateur, de même pour les films qui sont répartis en 18 catégories [3]. Pour aborder notre problème nous avons décidé de nous séparer en 3 binômes chargés respectivement de l'interprétation graphique, du préprocessing et du regressor.

Représentation graphique :

Le binôme 1 était chargé d'abord d'effectuer une analyse exploratoire de l'ensemble de données en calculant la note moyenne (après avoir normalisé les notes) en fonction des caractéristiques des utilisateurs et du genre des films (cf Figures 1,2,3). Pour visualiser nos données le binôme 1 a opté pour des violinplot de la librairie seaborn qui sont adapté à la visualisation de variables catégorielles (qui est notre cas ici) mais aussi de simples histogrammes nous permettant de voir notamment quelles variables sont les plus représentatives. Ainsi par exemple sur la figure 2 et 4 on voit que le genre de la personne n'est pas très représentative et que l'utilisation des autres variables apparaît être un choix judicieux. Le binôme 1 avait d'abord opté pour l'utilisation de heatmaps mais la tentative s'est avérée infructueuse, car les représentations obtenues n'avait aucun sens et était impossible à déchiffrer. Sur les figures jointes en annexes sont représentés les votes des films en fonction de l'âge des utilisateurs. (cf figure 4). Les violinplots permettent de montrer la répartition des données quantitatives entre plusieurs niveaux d'une (ou plusieurs) variables catégorielles telles que ces distributions peuvent être comparées. Contrairement à un tracé de boîte, dans lequel toutes les composantes de tracé correspondent à des points de données réels, le tracé de violon comporte une estimation de densité de noyau de la distribution sous-jacente.

Preprocessing :

Le code preprocessor est un code qui a pour but de prendre les données et de les rendre plus lisibles, pour cela nous avons décidé d'utiliser un preprocessor déjà implémenté dans sickit-learn qui est le « One Hot Encoding ». Au début le binôme chargé du preprocessing a commencé à coder un ensemble de fonction ou méthodes permettant une meilleure manipulation de nos données comme par exemple :

* **réduire** : une méthode qui permet de retirer de la liste des n-uplets qui ne correspondent pas à certains critères.

* **rangs** : une méthode qui donne à une liste de noms de colonne les numéros correspondants.

* **Trier** : une méthode qui trie un ensemble de n-uplets selon différents critères.

Cependant après recherche d'exemples sur sickit-learn, nous avons trouvé la technique du « One Hot Encoding » qui nous satisfaisait pleinement. En effet cette dernière a pour but d'encoder en entiers des variables catégoriels (ce qui est notre cas ici à l'aide d'un schéma one-hot appelé aussi one-of-K). L'entrée de ce transformateur devrait être une matrice d'entiers, en indiquant les valeurs prises par les fonctions catégoriques (professions des personnes, âges, types de films etc...). La sortie sera une matrice éparse où chaque colonne correspond à une valeur possible d'une caractéristique. On suppose que les fonctions d'entrée prennent des valeurs dans la plage $[0, n_values]$. L'avantage principal de cet encodage est que pour passer d'un état à un autre, seules deux transitions sont nécessaires: un chiffre passe de 1 à 0, un autre de 0 à 1 et qu'il nous permet de détecter facilement les valeurs aberrantes. Il est nécessaire pour alimenter des données catégoriques à de nombreux estimateurs scikit-learn, notamment des modèles linéaires (voir partie Regressor) et des SVM avec les noyaux standards). Le seul inconvénient que l'on peut souligner est qu'il faut au minimum n bits pour représenter n états, ce qui conduit à une augmentation linéaire du nombre de chiffres par rapport au nombre d'états, cependant ceci n'a pas influé sur notre travail.

Regressor :

Notre problème étant un problème de régression (en effet nous possédons un ensemble de variables quantitatives avec des plages continues). Pour ce faire nous avons créé une classe "Regressor" possède deux méthodes essentielles pour notre projet qui sont la méthode "fit" et "predict". La première méthode est nécessaire à la création de la méthode predict car elle permet aux données de s'adapter au modèle et elle permet "d'apprendre" au programme ce qu'il doit faire. La méthode "predict" quant à elle permet de renvoyer les étiquettes voulues (autrement dit les Y).

Pour la méthode fit nous avons décidé de faire appel au début à une méthode déjà implémentée dans sickit learn, qui est celle de "DecisionTreeRegressor". Son but est de créer un modèle qui prédit la valeur d'une variable-cible depuis la valeur de plusieurs variables d'entrée, cependant nous remarquons l'apparition de signes d'overfitting en effet le score sur la cross-Validation est supérieur au score sur les données d'entraînements (voir tableau 1 dans la partie bonus).

D'autres regressors ont été utilisés mais toutes les tentatives se sont avérées infructueuses

(pour le RandomForest, ce fut un cas qui marchait très bien cependant nous avons trouvé mieux).

Finalement, nous nous sommes tournés vers un linear regressor. Ce dernier adapte un modèle linéaire à l'ensemble de données en ajustant un ensemble de paramètres afin de rendre la somme des valeurs carrées du modèle aussi petite que possible (appelée méthode des moindres carrés). Tout d'abord nous avons implémenté une fonction init dans laquelle nous avons utilisé un pipeline qui nous permet de manipuler les données issues du preprocessing et du regressor comme une seule unité. Les fonctions fit et predict ont été expliquées un peu plus haut. Les deux dernières fonctions save et load pickle permettent de convertir un objet python (list, dict, etc.) en un flux de caractères. L'idée est que ce flux de caractères contient toutes les informations nécessaires pour reconstruire l'objet dans un autre script python.

On peut écrire un modèle linéaire comme suit:

$$\mathbf{f}(\mathbf{u};\mathbf{m}) = \mathbf{Xum} \cdot \boldsymbol{\theta}$$

où \mathbf{Xum} est le vecteur caractéristique de longueur 54 (nombre de features) représentant la paire de films utilisateur (u ; m) et $\boldsymbol{\theta}$ est l'ensemble des valeurs à apprendre.

L'évaluation de l'erreur peut être déterminée grâce à la fonction suivante :

$$F(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (f(u_i, m_i) - X_i \cdot \boldsymbol{\theta})^2 + \lambda \sum_{i=2}^N \theta^2_i$$


Où $N = 54$ est le nombre de features, λ l'hyper-paramètre de régularisation, X_i le vecteur caractéristique représentant l' i ème exemple d'entraînement. Le terme $\sum_{i=2}^N \theta^2_i$ permet de réduire l'overfitting (apprentissage « par cœur » des données).


Notons aussi que ce projet aurait pu être traité comme un problème de classification « multiclass » qui consiste à classer des instances en plus de deux classes (classification binaires).

La classification multiclass ne doit pas être confondue avec la classification multi-étiquettes, où plusieurs étiquettes doivent être prédites pour chaque instance. Cependant nous avons renoncé rapidement, par manque de bons résultats et aussi par le fait que comme les données sont fortement corrélées entre elles, il est très difficile d'implémenter ou d'utiliser ce genre d'algorithme.

Résultats:

#	Submission Id	User	Phase	Date	Description	Prediction score	Duration	Likes	Downloads	Detailed results?
1	396903	gold	Development Phase	Mar 29 2017		2.5919	7.0154	👍👎 0	📄 (2 dls)	• View detailed results
2	396909	lemon	Development Phase	Mar 29 2017		0.7359	7.0098	👍👎 0	📄 (2 dls)	• View detailed results
3	397424	vanilla	Development Phase	Apr 07 2017		0.0088	4.0098	👍👎 0	📄 (0 dls)	• View detailed results
4	397477	lemon	Development Phase	Apr 07 2017		-1.7839	8.0164	👍👎 0	📄 (0 dls)	• View detailed results
5	399269	vanilla	Development Phase	Apr 29 2017		0.2041	4.0071	👍👎 0	📄 (0 dls)	• View detailed results
6	399274	lemon	Development Phase	Apr 29 2017		-1.7839	8.0114	👍👎 0	📄 (0 dls)	• View detailed results

 : score starting-kit

 : score final

Ainsi grâce au linear regresseur et aux données du preprocessing, on peut voir que l'on obtient un score bien plus supérieur à celui du starting-kit mais aussi par rapport aux autres regresseurs utilisés (voir tableau 1 du bonus).

Conclusion :

Cette UE qu'est « Mini-projet » a été une sorte de voyage pour nous, nous permettons de découvrir le monde aussi fascinant que complexe du data science. Mais aussi au-delà de cet aspect éducatif, il nous a permis de mettre en place une méthodologie de travail où la rigueur et le sérieux priment. Cependant, l'aspect qui en ressort fortement est celui du travail d'équipe qui est très important dans le domaine informatique, nous permettant ainsi de développer notre sens de la communication et de l'échange que ce soit entre binômes d'une même équipe et entre équipes différentes. La principale difficulté rencontrée fut l'utilisation du langage, si bien qu'aucun membre du groupe n'avait programmé en python auparavant, mais aussi le fait de réunir tout le monde, et surtout de faire travailler tout le monde. En effet à la fin du semestre, seulement la moitié du groupe (pourtant composé de 6 personnes) a vraiment mis la main à la pâte. Si nous avions eu plus de temps, nous aurions pu mieux exploiter les différentes possibilités de représentation graphique, mais aussi le fait de pouvoir combiner différents preprocessors et regressors pour améliorer un peu plus notre score. Enfin, pour finir nous aimerions dire aux étudiants de l'an prochain de ne pas négliger ce module et de commencer très tôt les recherches surtout pour les débutants en python, pour pouvoir ainsi se familiariser avec les différents outils d'anaconda que ce soit spyder ou jupyter-notebook. Quant au fonctionnement du groupe, je leur conseille tout d'abord de bien choisir leurs binômes et les autres membres du groupe, pour pouvoir travailler plus flexiblement et dans de meilleures conditions.

Figures

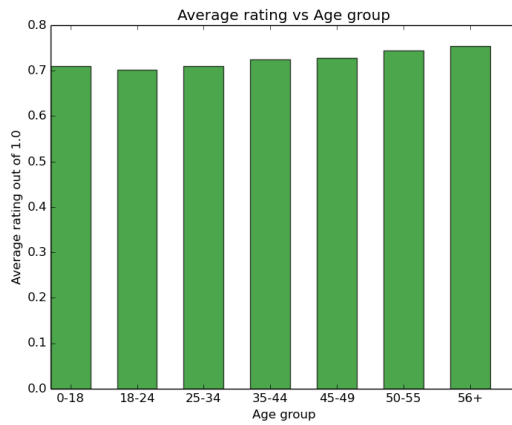


Figure 1-Notes moyennes en fonction de l'âge de l'utilisateur- [1]

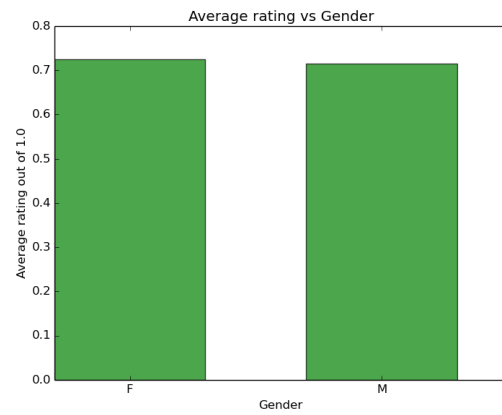


Figure 2-Notes moyennes en fonction du genre de la personne- [1]

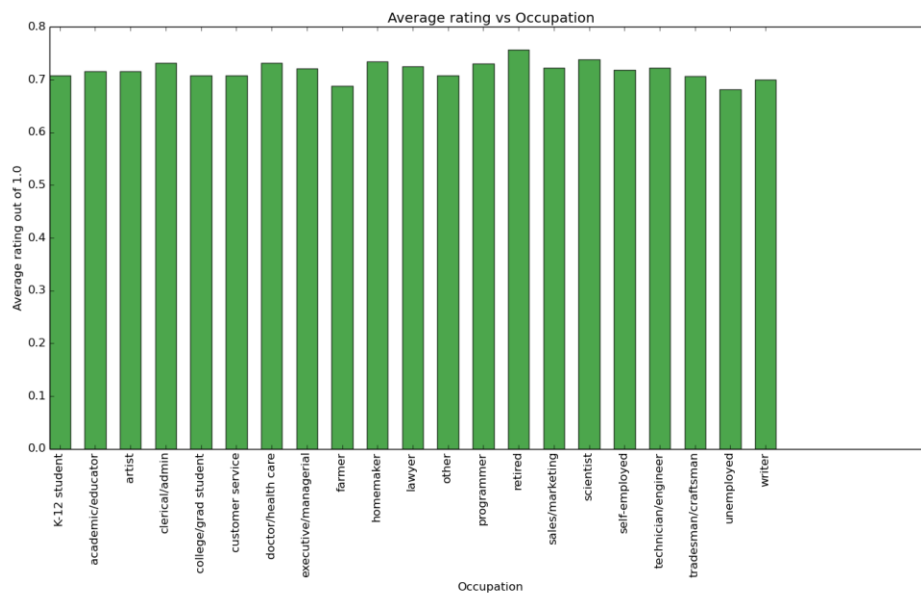


Figure 3-Notes moyennes en fonction du métier de l'utilisateur-[1]

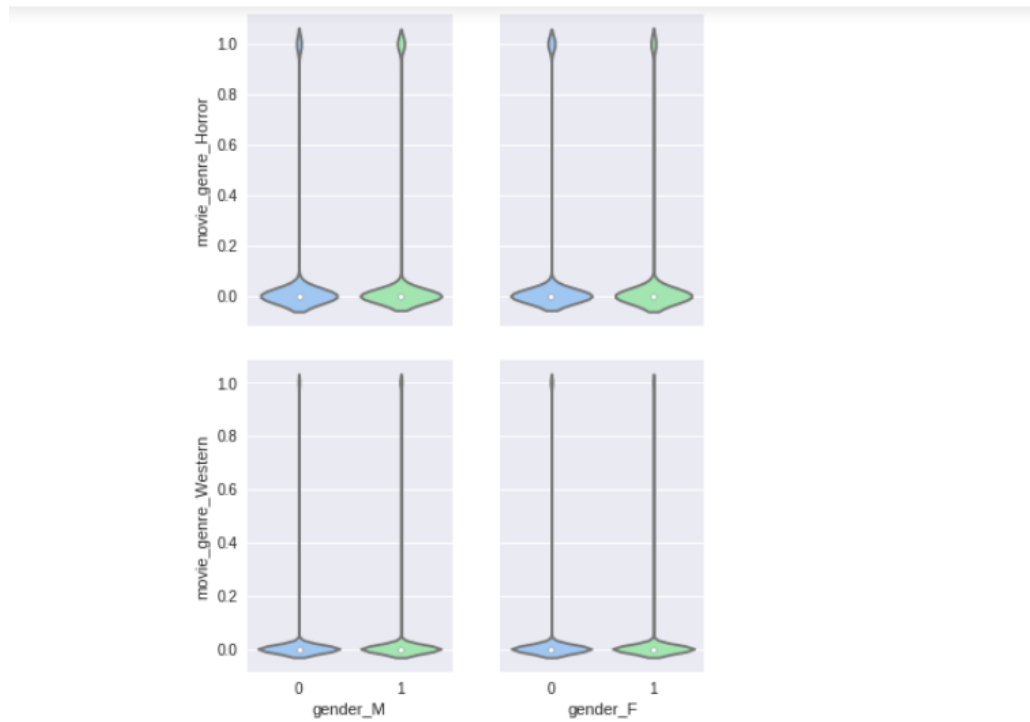
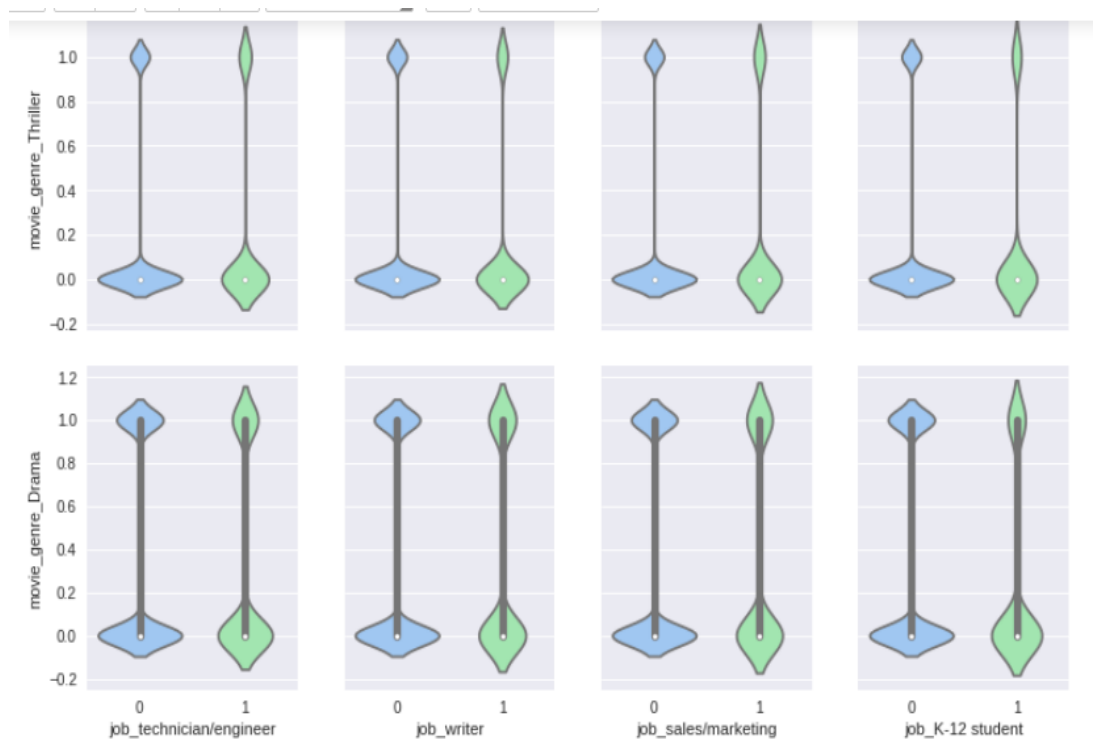


Figure 4- Violinplot représentant le type de films en fonction du genre de la personne-



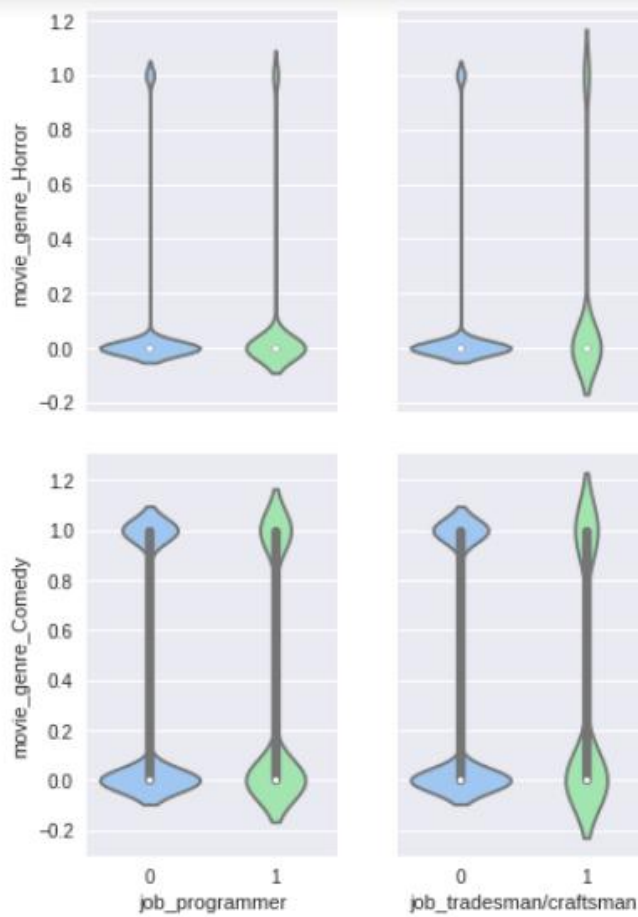


Figure 5- Violinplot représentant le type de films en fonction de la profession de la personne-

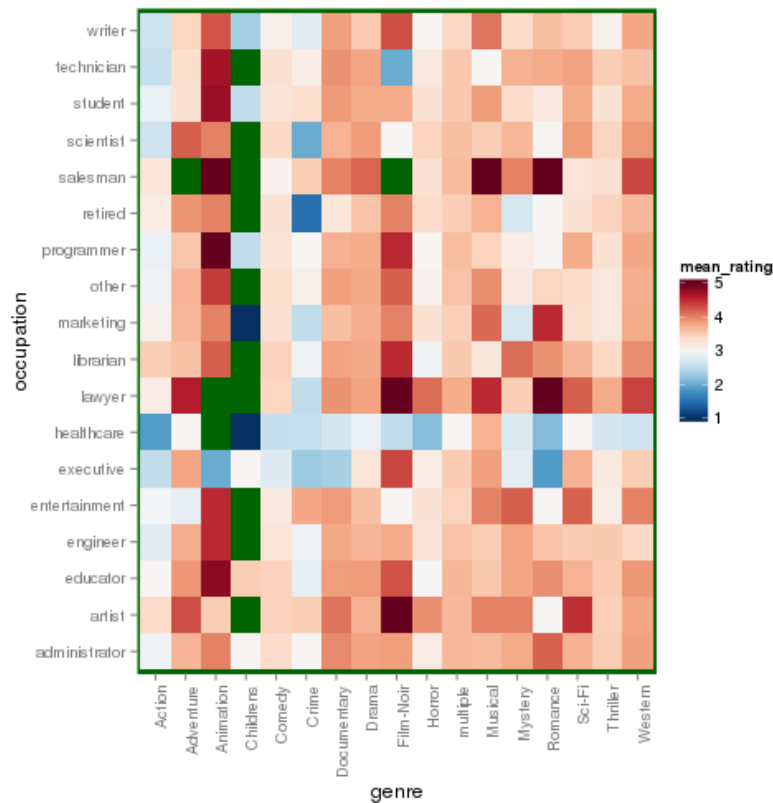


Figure 5-Heatmap-

Code :

DataManager :

```
class DataManager(data_manager.DataManager):

#   def __init__(self, basename="", input_dir=""):
#       ''' New constructor.'''
#       DataManager.__init__(self, basename, input_dir)
#       # So something here

    def toDF(self, set_name):
        ''' Change a given data subset to a data Panda's frame.
            set_name is 'train', 'valid' or 'test'.'''
        DF = pd.DataFrame(self.data['X_'+set_name])
        # For training examples, we can add the target values as
        # a last column: this is convenient to use seaborn
        # Look at http://seaborn.pydata.org/tutorial/axis\_grids.html for other ideas
        if set_name == 'train':
            Y = self.data['Y_train']
            DF = DF.assign(target=Y)
        return DF

    def DataStats(self, set_name):
        ''' Display simple data statistics'''
        DF = self.toDF(set_name)
        print DF.describe()

    def ShowViolinplot(self, dn, x1, y1):
        # "age_25-34"      "age_45-49", "age_50-55", "age_56+"
        g = sns.PairGrid(dn,
                        x_vars=[x1],
                        y_vars=[y1],
                        aspect=.75, size=3.5)
        g.map(sns.violinplot, palette="pastel");
```

Son test :

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from DataManager import DataManager
input_dir = "../public_data"
output_dir = "../res"

basename = 'movierec'
D = DataManager(basename, input_dir)
print D

D.DataStats('train')
D.ShowViolinplot('data', "age_18", "movie_genre_Romance")
```

Preprocessor :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from outils import *
from sys import argv
from sklearn.base import BaseEstimator
from DataManager import DataManager
from sklearn.preprocessing import OneHotEncoder

class Preprocessor:
    def __init__(self):
        self.table = []
        self.categories = []
        self.transformer = OneHotEncoder();
    def fit(self, X, y=None):
        return self.transformer.fit(X, y)

    def fit_transform(self, X, y=None):
        return self.transformer.fit_transform(X)

    def transform(self, X, y=None):
        return self.transformer.transform(X)
```

|

Son Test :

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from DataManager import DataManager
from zPreprocessor import Preprocessor
input_dir = "../public_data"
output_dir = "../res"

basename = 'movierec'
D = DataManager(basename, input_dir) # Load data
print("**** Original data ****")
print D

Prepro = Preprocessor()

# Preprocess on the data and load it back into D
D.data['X_train'] = Prepro.fit_transform(D.data['X_train'], D.data['Y_train'])
D.data['X_valid'] = Prepro.transform(D.data['X_valid'])
D.data['X_test'] = Prepro.transform(D.data['X_test'])

# Here show something that proves that the preprocessing worked fine
print("**** Transformed data ****")
print D

# Preprocessing gives you opportunities of visualization:
# Scatter-plots of the 2 first principal components
# Scatter plots of pairs of features that are most relevant
import matplotlib.pyplot as plt
X = D.data['X_train']
Y = D.data['Y_train']
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.colorbar()
plt.show()
```

Regressor :

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from sklearn.base import BaseEstimator
from sklearn.preprocessing import Preprocessor
from sklearn import linear_model
from sklearn.pipeline import Pipeline
import pickle

class Regressor(BaseEstimator):
    def __init__(self):
        regressor = Pipeline([
            ('preprocessing', Preprocessor()),
            ('regression', linear_model.LinearRegression())])
        self.clf = regressor

    def fit(self, X, y):
        self.clf.fit(X, y)

    def predict(self, X):
        return self.clf.predict(X)

    def get_classes(self):
        return self.clf.classes_

    def save(self, path="./"):
        pickle.dump(self, open(path + '_model.pickle', "w"))

    def load(self, path="./"):
        self = pickle.load(open(path + '_model.pickle'))
        return self
```

Son test :

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from DataManager import DataManager
from Regressor import Regressor
from sklearn.metrics import accuracy_score
# from sklearn.model_selection import cross_val_score

input_dir = "../public_data"
output_dir = "../res"

basename = 'movierec'
D = DataManager(basename, input_dir) # Load data
print D

myRegressor = Regressor()

# Train
Ytrue_tr = D.data['Y_train']
myRegressor.fit(D.data['X_train'], Ytrue_tr)

# Making predictions
Ypred_tr = myRegressor.predict(D.data['X_train'])
Ypred_va = myRegressor.predict(D.data['X_valid'])
Ypred_te = myRegressor.predict(D.data['X_test'])

# We can compute the training success rate
acc_tr = accuracy_score(Ytrue_tr, Ypred_tr)
# But it might be optimistic compared to the validation and test accuracy
# that we cannot compute (except by making submissions to Codalab)
# So, we can use cross-validation:
# acc_cv = cross_val_score(myRegressor, D.data['X_train'], Ytrue_tr, cv=5, scoring='accuracy')
```

Références :

[1] CSE 255 Assignment 1 : Movie Rating Prediction using the MovieLens dataset.

Yashodhan Karandikar

[2] Scikit-learn: Machine Learning in Python, < http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression>

[3] The Movie Recommendation Challenge

<https://competitions.codalab.org/competitions/16151#learn_the_details-overview> [4] GitHub : Madclam <https://github.com/madclam/L2_iris>

[5] Cross-validation: evaluating estimator performance

<http://scikit-learn.org/stable/modules/cross_validation.htm>

[6] Violinplot : < <http://seaborn.pydata.org/generated/seaborn.violinplot.html>>

Bonus :

1).

Table 1 : Statistiques des données

Dataset	Num. Examples	Num. Variables/Features	Sparsity	Has categorical Variables ?	Has missing data ?	Num. Examples in each class
Training	105000	54	//	NON	NON	54
CV	15000	54	//	NON	NON	54
Valid(ation)	30000	54	//	NON	NON	54

Table 2 : Résultats préliminaires

Méthodes	Decision Tree Regressor	Linear Regression	Random Forest Regressor
Training	0.913318221857	0.784841236388	0.945085687996
CV	0.955996955951	0.702992533516	0.923515337324
Valid(ation)	0.008762164967	0.204063204042	0.028910302908

2). Brèves explications :

• **Decision Tree Regressor** : C'est une technique d'apprentissage supervisé : on utilise un ensemble de données pour lesquelles on connaît la valeur de la variable-cible afin de construire l'arbre (données dites étiquetées), puis on extrapole les résultats à l'ensemble des données de test.

• **Linear Regression**: Le but de ce modèle est de prédire des scores sur une variable à partir des scores sur une deuxième variable. La variable que nous voulons prédire est appelée variable de critère et est appelée Y. La variable sur laquelle nous reposons nos prédictions est appelée variable prédictive et est appelée X. Lorsqu'il n'y a qu'une seule variable prédictive, la méthode de prédiction est appelée Régression simple. En régression linéaire simple, les prévisions de Y lorsqu'elles sont tracées en fonction de X forment une ligne droite.

• **Random Forest Regressor** : Une forêt aléatoire est un méta-estimateur qui correspond à un certain nombre d'arbres de décision utilisés sur différents sous-échantillons de l'ensemble de données et utilise la moyenne pour améliorer la précision prédictive et contrôler l'overfitting. La taille de l'échantillon secondaire est toujours identique à la taille

d'échantillon d'entrée d'origine.

3). Métrique utilisée :

Les soumissions sont évaluées grâce à la métrique suivante : $a_metric = 1 - \frac{MAE}{MAD}$. Cette dernière est obtenue en retranchant à 1, le quotient entre l'erreur moyenne absolue (MAE) et la déviation moyenne absolue (MAD). L'erreur moyenne absolue ou MAE est une quantité utilisée pour mesurer la proximité des prévisions avec les résultats éventuels, la déviation moyenne absolue ou MAD quant à elle est la moyenne des déviations absolues des éléments (différence absolue entre ces éléments et un point donné) par rapport à la moyenne.

4). Définition de Cross-Validation :

La *Cross-Validation* désigne le processus qui permet tester la précision prédictive d'un modèle dans un échantillon test (parfois aussi appelé *échantillon de Cross-Validation*) par rapport à la précision prédictive de l'échantillon d'apprentissage à partir duquel le modèle a été développé. Dans l'idéal, avec un échantillon suffisamment grand, une certaine proportion d'observations (disons la moitié ou les deux-tiers) peut être affectée à l'échantillon d'apprentissage et les observations restantes sont affectées à l'échantillon test. Le modèle peut être construit en utilisant les observations de l'échantillon d'apprentissage, et la puissance prédictive peut être testée en utilisant les observations de l'échantillon test. Si le modèle s'exécute aussi bien sur l'échantillon test que sur l'échantillon d'apprentissage, nous pouvons dire que la *validation croisée* est bonne ou plus simplement, qu'il y a *validation croisée*.

5).Explication du sur-apprentissage :

Le sur-apprentissage ou « overfitting » en anglais, s'interprète comme un apprentissage "par cœur" des données. Il résulte souvent d'une trop grande liberté dans le choix du modèle. Il est en général provoqué par un mauvais dimensionnement de la structure utilisée pour classifier. De par sa trop grande capacité à stocker des informations, une structure dans une situation de sur-apprentissage aura de la peine à généraliser les caractéristiques des données. Elle se comporte alors comme une table contenant tous les échantillons utilisés lors de l'apprentissage et perd ses "pouvoirs" de prédiction sur de nouveaux échantillons. L'excès de fonctionnement se produit lorsqu'un modèle est excessivement complexe, comme le fait d'avoir trop de paramètres par rapport au nombre d'observations. Un modèle qui a été "overfitté" a des performances prédictives médiocres, car il réagit de manière excessive aux fluctuations mineures des données de formation.

