département
Informatique

Université Claude Bernard Lyon 1

Année universitaire	**2025-2026**		
Département	**Informatique**	**Année**	**M1**
Matière	**Apprentissage et Analyse de données**		
Enseignants	**Haytham Elghazel et Ichrak Ennaceur**		
Intitulé TD/TP :	**Atelier 1 : Apprentissage supervisé avec Python**		
Contenu	• **Data Preprocessing (données hétérogènes, données manquantes, etc.)** • **Feature engineering** • **Feature selection** • **Classification** • **Evaluation de la qualité d'un classifieur**		
Rendu	**Rendu sur Moodle (Binôme ou Trinôme) : Un code factorisé, bien structuré et commenté**		

Dans cet atelier pratique, vous allez expérimenter des algorithmes de traitement de données pour répondre à différents problèmes liés à l'apprentissage supervisé avec le langage **Python**.

Python est un langage de programmation très polyvalent et modulaire, qui est utilisé aussi bien pour écrire des applications comme YouTube, que pour traiter des données scientifiques. Par conséquent, il existe de multiples installations possibles de Python. L'utilisateur débutant peut donc se sentir dérouté par l'absence d'une référence unique pour Python scientifique. Le plus simple pour ce TP est d'installer, la suite scientifique Anaconda développée par l'entreprise Continuum (http://continuum.io/downloads.html). Anaconda rassemble tout le nécessaire pour l'enseignement de Python scientifique : le langage Python et ses modules scientifiques. Sur le plan des packages Python, vous allez utiliser **Scikit-learn**. Cette libraire montre dans cette situation tout son intérêt. La plupart des techniques récentes d'apprentissage sont en effet expérimentées avec Scikit-learn et le plus souvent mises à disposition de la communauté scientifique.

Pour plus de détails concernant :

- le langage Python vous pouvez aller sur le site suivant : http://www.python-course.eu/index.php
- la librairie Scikit-learn vous pouvez aller sur le site suivant : http://scikit-learn.org

Pour lancer le notebook Python, il faut taper la commande **jupyter notebook** dans votre dossier de travail. Une fenêtre va se lancer dans votre navigateur pour ouvrir l'application Jupyter. Créer un nouveau notebook Python et taper le code suivant dans une nouvelle cellule :

Créer un nouveau notebook Python et taper le code suivant dans une nouvelle cellule :

```
import numpy as np
np.set_printoptions(threshold=10000, suppress=True)
import pandas as pd
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

I. Apprentissage supervisé : Feature engineering et Classification

L'objectif dans cette partie est de construire un bon classifieur sur un jeu de données de *crédit scoring* du fichier **"credit_scoring.csv"**.

1. ***Chargement des données et préparation*** **:** Dans un premier temps nous allons importer le jeu de données et analyser ses caractéristiques.
    - Importer ce jeu de données avec la librairie **pandas** (*c.f.* ***read_csv***)

- Transformer votre jeu de données issue de **pandas** qui sera de type **Data Frame** en **numpy Array** (*c.f.* ***values***) et séparer ensuite les variables caractéristiques de la variable à prédire (**status**) en deux tableaux différents.
- Analyser les propriétés de vos données : taille de l'échantillon (*c.f.* ***shape***), pourcentage d'exemples positifs et négatifs.
- Pour éviter d'avoir un résultat biaisé du classifieur que nous allons construire, séparer les données en deux parties (de taille **50%** chacune) une dite d'apprentissage qui servira à l'apprentissage du classifieur et l'autre dite de test qui servira à son évaluation (*c.f.* ***train_test_split*** avec un **random_state=1**).

2. ***Apprentissage et évaluation de modèles :*** Utiliser ensuite sur votre jeu de données les algorithmes d'apprentissage supervisé suivants :
    - Un arbre CART **(random_state=1)**
    - k-plus-proches-voisins avec **k=5**
    - MultilayerPerceptron à **deux couches de tailles respectives 40 et 20** et **random_state=1**

L'objectif est à présent de comparer les résultats obtenus à l'aide de ces trois simples algorithmes sur ce jeu de données. Cette comparaison doit être faite dans une **fonction python** et s'appuiera sur :

- la ***matrice de confusion***,
- l'estimation de ***la moyenne de l'accuracy et le meilleur critère*** entre le ***Rappel*** et la ***Précision*** *(rappel ou précision)* dont vous pensez qu'il est le plus adéquat pour ce cas métier de *crédit scoring*,
- la courbe ROC et l'AUC affichée sur le graphique.

Dans cette partie, vous devez définir :

- Un ***dictionnaire*** contenant les algorithmes supervisés utilisés.
- Une ***fonction d'évaluation*** permettant :
    - d'afficher la matrice de confusion et la courbe ROC avec l'AUC,
    - de calculer l'accuracy, le rappel **ou** la précision et de retourner un score final donné par la moyenne entre l'accuracy et le meilleur des deux critères selon ce cas métier (accuracy + meilleur critère) / 2.
- Une ***fonction d'apprentissage et de test*** (***run_classifiers_train_test***) permettant :
    - d'entraîner chaque algorithme de votre dictionnaire,
    - d'afficher les résultats d'évaluation,
    - de comparer les scores obtenus,
    - et de retourner le meilleur modèle selon le score final (accuracy + meilleur critère) / 2.

3. ***Normalisation des variables continues*** **:** Certains algorithmes d'apprentissage supervisé fonctionneront mieux si les données sont normalisées (centrées autour de 0) pour que toutes les variables caractéristiques auront le même poids dans la phase d'apprentissage. Utiliser le module ***StandardScaler*** de Scikit-learn pour normaliser vos données (il est possible d'utiliser à la place le module ***MinMaxScaler***). Exécuter à nouveau votre fonction sur vos données une fois normalisées. Interpréter les résultats obtenus en les comparant avec ceux de la question précédente. **Attention à bien utiliser la bonne méthodologie pour normaliser la base de train et la base de test.**

4. ***Création de nouvelles variables caractéristiques par combinaisons linéaires des variables initiales*** **:** Il est parfois utile pour certains classifieurs de faire une réduction de dimensions sur les données afin de déceler et créer certaines combinaisons linéaires dans les variables descriptives et augmenter ainsi le pouvoir discriminant du classifieur. Appliquer une **ACP** (module ***PCA*** de Scikit-learn) sur vos données et garder les **3** premières nouvelles dimensions en les concaténant à vos données normalisées de l'étape précédente. Exécuter à nouveau votre code sur vos nouvelles données. Que se passe-t-il ? **Attention à bien utiliser la bonne méthodologie pour calculer les variables de l'ACP sur la base de train et la base de test.**

A partir de cette étape il faut noter quel est le meilleur algorithme que vous allez garder pour la suite et quelle est la meilleure stratégie de préparation de données. Faut-il ou pas normaliser ? Faut-il ou pas inclure les nouvelles variables issues de l'ACP ?

5. ***Sélection de variables*** **:** Même si vous utilisez le meilleur algorithme d'apprentissage, la présence de

variables bruitées pourra avoir un impact négatif sur les résultats d'apprentissage. La sélection de variables est un processus très important en apprentissage supervisé. Il consiste à sélectionner le sous-ensemble de variables les plus pertinentes (en enlevant le bruit et la redondance) à partir de la série de variables candidates permettant de mieux expliquer et prédire votre target. Dans la suite, vous allez utiliser la méthode **Random Forest** de Scikit-learn pour déterminer quelles sont les meilleures variables pour prédire si une personne va payer son crédit ou pas. Attention :

Ainsi, il faut afficher un histogramme des importances des variables.

Attention : dans la suite de cette partie, pour X_train et X_test, il faut utiliser les données issues de votre propre choix effectué avant l'étape 5.
Les éléments affichés en vert doivent être ajustés en fonction de vos résultats précédents et de ce qui est demandé dans le TP.

Il est demandé d'inclure les blocs de code ci-dessous dans trois fonctions Python distinctes, afin de structurer votre travail et de favoriser sa réutilisation dans la suite du TP.

Première fonction : Importance des variables
Cette fonction doit permettre de calculer et d'afficher l'importance relative des variables à l'aide d'un modèle Random Forest :

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=1000,random_state=1)
clf.fit(Xtrain, Ytrain)
importances=clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],axis=0)

sorted_idx = np.argsort(importances)[::-1]

features =nom_cols
print(features[sorted_idx])

padding = np.arange(Xtrain.size/len(Xtrain)) + 0.5
plt.barh(padding, importances[sorted_idx],xerr=std[sorted_idx], align='center')
plt.yticks(padding, features[sorted_idx])
plt.xlabel("Relative Importance")
plt.title("Variable Importance")
plt.show()
```

Deuxième fonction – Sélection du nombre optimal de variables
Cette fonction doit permettre de déterminer combien de variables il est pertinent de conserver, en mesurant l'évolution de la performance du **meilleur algorithme** choisi précédemment.

```
KNN=KNeighborsClassifier(n_neighbors=5) // Ici c'est votre Meilleur algorithme
scores=np.zeros(Xtrain.shape[1])
for f in np.arange(0, Xtrain.shape[1]):
    X1_f = Xtrain[:,sorted_idx[:f+1]]
    X2_f = Xtest [:,sorted_idx[:f+1]]
    KNN.fit(X1_f,Ytrain)
    YKNN=KNN.predict(X2_f)
    scores[f]=np.round(accuracy_score(Ytest,YKNN),3)
plt.plot(scores)
plt.xlabel("Nombre de Variables")
plt.ylabel("Accuracy")
plt.title("Evolution de l'accuracy en fonction des variables")
plt.show()
```

Troisième fonction – Explicabilité avec SHAP

Cette fonction doit intégrer une étape **d'explicabilité** à l'aide de la bibliothèque **SHAP** (*SHapley Additive exPlanations*). Elle doit permettre d'interpréter le modèle en identifiant les variables qui influencent le plus les prédictions, aussi bien de manière globale que locale.

6. **Paramétrage des classifieurs** : Dans cette partie, vous allez utiliser la méthode **GridSearchCV** de scikit-learn afin de tuner les paramètres de votre meilleur algorithme. Vous allez choisir les meilleurs paramètres qui optimisent au mieux le critère suivant : **(accuracy+précision)/2** utilisant le nombre optimal de variables. Votre code de choix des meilleurs paramètres doit être inclus dans une **fonction Python**

7. **Création d'un pipeline :** Dans cette partie vous allez automatiser l'enchainement des traitements effectués précédemment (Normalisation ou non, ACP ou non et Construction du classifieur et sélection de variables) dans un pipeline (module ***pipeline.Pipeline*** de Scikit-learn). Sauvegarder ensuite votre pipeline dans un **pickle** afin de pouvoir l'utiliser dans une phase de scoring. Votre code de création de pipeline doit être inclus dans une **fonction Python**.

8. **Orchestration :** Après avoir développé séparément les différentes fonctions suivantes qu'il faut toutes intégrées dans **un fichier utils.py :**
    - une fonction d'apprentissage et de test des algorithmes supervisés,
    - une fonction d'apprentissage et de test des algorithmes supervisés sur des données normalisées,
    - une fonction d'apprentissage et de test des algorithmes supervisés sur des données incluant des nouvelles variables,
    - une fonction de sélection des variables les plus pertinentes,
    - une fonction de recherche et de choix des meilleurs paramètres,
    - une fonction de création du pipeline d'apprentissage,

    Vous allez maintenant concevoir une fonction principale *pipeline_generation_train_test_split* capable d'orchestrer l'ensemble de ces étapes qui fournira en sortie le pipeline dans un pickle.

9. **Création d'une API FastAPI pour exécuter le pipeline :**

    Créer une API avec FastAPI permettant :

    - de charger le pipeline depuis le pickle,
    - de recevoir en entrée un individu de test (ses valeurs de features) via un endpoint **/predict/,**
    - de renvoyer la prédiction et, si possible, les probabilités associées.

    Vous devrez tester votre API sur un ou plusieurs individus de test et vérifier que les prédictions correspondent à vos attentes.

10. Après avoir construit votre pipeline et testé quelques algorithmes sur une seule base de test, il est nécessaire d'étendre la comparaison à plusieurs classifieurs afin de sélectionner le modèle le plus performant de manière robuste en utilisant une validation croisée.

    Comparaison de plusieurs algorithmes *d'apprentissage* : Utiliser ensuite sur votre jeu de données les algorithmes d'apprentissage supervisé suivants :
    - Un arbre CART
    - Un arbre ID3
    - Un Decision Stump
    - MultilayerPerceptron à deux couches de tailles respectives 20 et 10 par exemple
    - k-plus-proches-voisins avec k=5 par exemple
    - Bagging avec 200 classifieurs par exemple
    - AdaBoost avec 200 classifieurs par exemple
    - Random Forest avec 200 classifieurs par exemple
    - Un XGboost avec 200 classifieurs par exemple
    - Etc.

Remarque : Si vous ne connaissez pas le fonctionnement précis de l'un de ces algorithmes, n'hésitez pas à consulter la documentation de Scikit-learn. Attention, la plupart de ces algorithmes ont des paramètres qu'il vous faudra prendre en compte lors de vos expérimentations. Vous en choisirez quelques-uns que vous ferez varier afin d'observer l'effet pratique de ces paramètres sur les résultats obtenues (par exemple ***k*** pour les ***k-plus-proches-voisins*** ou le nombre d'arbres dans un Random Forest).

L'objectif est à présent de comparer les résultats obtenus à l'aide des différents algorithmes donnés ci-dessus sur votre jeu de données. Ces comparaisons s'appuieront sur :

- l'estimation de l'***accuracy*** et de l'***AUC*** (Aire sous la courbe ROC) par 10 fold cross-validation (*c.f.* ***cross_val_score***). Il faut afficher la moyenne et l'écart type.
- l'estimation aussi par 10 fold cross-validation du **critère (accuracy+précision)/2**. Il faut afficher aussi la moyenne et l'écart type.
- le temps d'exécution de l'algorithme d'apprentissage (*c.f.* ***time.time()***)

Note :

Pour comparer plusieurs classifieurs sur la même validation croisée, il est recommandé d'utiliser un dictionnaire Python regroupant les classifieurs, par exemple :

```
clfs = {
  'RF': RandomForestClassifier(n_estimators=200, random_state=1),
  'KNN': KNeighborsClassifier(n_neighbors=10),
 liste à compléter
}
kf = KFold(n_splits=10, shuffle=True, random_state=0)
for i in clfs:
   clf = clfs[i]
   cv_acc = cross_val_score(clf, X, Y, cv=kf)
   print("Accuracy for {0} is: {1:.3f} +/- {2:.3f}".format(i, np.mean(cv_acc), np.std(cv_acc)))
```

Vous créerez ensuite une fonction Python ***run_classifiers_cv*** qui automatisera cette comparaison pour tout dictionnaire de classifieurs **clfs** et jeux de données donnés (**X** et sa target **Y**).

- Exécuter cette fonction sur le jeu de données original, sur les données normalisées et sur les données incluant des variables issues d'une ACP.
- Identifier le meilleur algorithme selon vos critères.
- Refaire ensuite les étapes de sélection de variables, recherche de paramètres et création du pipeline avec ce meilleur algorithme afin de le mettre en production.
- Concevoir une fonction principale ***pipeline_generation_cv*** capable d'orchestrer ces étapes et de sauvegarder le pipeline final dans un fichier pickle.

II. Apprentissage supervisé : Données hétérogènes

L'objectif dans cette partie est de faire une étude comparative entre plusieurs algorithmes d'apprentissage supervisé sur un nouveau jeu de données de *crédit scoring*. Pour plus d'informations sur ce jeu de données, vous pouvez visiter le lien suivant (https://archive.ics.uci.edu/ml/datasets/Credit+Approval).

Le fichier **"credit.data"** comporte 688 instances décrites par 15 variables caractéristiques (6 numériques, 9 catégorielles) et la variable à prédire "classe" (la dernière colonne du fichier) de nature nominale possédant un nombre fini de valeurs (ici deux valeurs "+" et "–"). Il ne s'agit pas d'une tâche de régression, mais de classification. Les exemples de ce jeu de données représentent des personnes (positifs et négatifs) pour lesquels un crédit a été accordé ou non.

1. Dans un premier temps nous allons considérer que les caractéristiques continues (numériques) de ce jeu de données sans les données manquantes.

    - ***Chargement des données et préparation*** **:**

- Importer ce jeu de données avec la librairie **pandas** (*c.f.* ***read_csv***)
- Transformer votre jeu de données issue de **pandas** qui sera de type **Data Frame** en **numpy Array** (*c.f.* ***values***) et séparer ensuite les variables caractéristiques de la variable à prédire (target) en deux tableaux différents.
- Créer un sous-ensemble de vos données en gardant que les variables numériques et en remplaçant les valeurs manquantes (?) par des **nan**. N'oublier de typer votre tableau en type **float** (*c.f.* ***astype***).
- Supprimer les individus dans vos données contenant des **nan** sur au moins une variable.
- Analyser les propriétés de vos données : taille de l'échantillon (*c.f.* ***shape***), nombre d'exemples positifs et négatifs (*c.f.* ***hist***).
- Binariser votre target (+ en 1 et – en 0).

L'objectif est à présent de comparer les résultats obtenus à l'aide des différents algorithmes**.** Exécuter votre fonction ***run_classifiers*** en rajoutant l'***AUC*** (Aire sous la courbe ROC) comme critère d'évaluation et interpréter les résultats obtenus.

- ***Normalisation des variables continues*** **:** Certains algorithmes d'apprentissage supervisé fonctionneront mieux si les données sont normalisées (centrées autour de 0) pour que toutes les variables caractéristiques auront le même poids dans la phase d'apprentissage. Utiliser le module ***StandardScaler*** de Scikit-learn pour normaliser vos données. Vous pouvez également tester le module ***MinMaxScaler***. Exécuter votre fonction ***run_classifiers*** sur vos données une fois normalisées. Interpréter les résultats obtenus en les comparant avec ceux de la question précédente.

2. Nous allons maintenant considérer la totalité de la base originale comportant les 15 variables continues et catégorielles mais aussi les données manquantes.

- ***Traitement de données manquantes*** **:** La non utilisation des données manquantes peut impacter la phase d'apprentissage suite à la perte d'information susceptible d'être pertinente et/ou informative, surtout quand la proportion des données manquantes dans l'échantillon est forte. Pour traiter les données manquantes deux méthodologies sont possibles : Soit (1) d'utiliser une technique d'imputation de valeurs manquantes, ou (2) non en intégrant des indicateurs de données manquantes dans l'échantillon par l'ajout par exemple d'une modalité à votre variable catégorielle incomplètement remplie (remplacer par exemple le '?' dans une variable sexe avec deux modalités 'Femme' et 'Homme' par 'Inconnu'). Dans la suite, vous allez utiliser la première méthodologie. Pour imputer des données manquantes, différentes stratégies sont possibles, certaines sont supervisées (utilisant la variable target pour remplir les valeurs manquantes dans les autres variables) et d'autres non supervisées. Pour plus de détails considérant ces approches, les liens suivants pourront vous intéresser http://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-idm.pdf et http://cybertim.timone.univ-mrs.fr/Members/rgiorgi/DossierPublic/Enseignement/TraitementNA-PDF-RG/docpeda_fichier.
  Imputer les valeurs manquantes dans votre jeu de données en utilisant les stratégies non supervisées suivantes du module ***Imputer*** de Scikit-learn (voir code ci-dessous) :
  - ***mean*** pour les variables continues
  - ***most_frequent*** pour les variables catégorielles

  Attention : Imputer ne considère que des données sous format de **float**. Il faut d'abord transformer les colonnes catégorielles en valeurs numériques (par exemple ['a', 'b', 'a'] en [1,2,1]. Il faut utiliser pour cela le code suivant sur votre tableau de données avec les données catégorielles (ici c'est X_cat) :

Pour les variables catégorielles

```
X_cat = np.copy(X[:, col_cat])
for col_id in range(len(col_cat)):
   unique_val, val_idx = np.unique(X_cat[:, col_id], return_inverse=True)
   X_cat[:, col_id] = val_idx
```

```
imp_cat = Imputer(missing_values=0, strategy='most_frequent')
X_cat[:, range(5)] = imp_cat.fit_transform(X_cat[:, range(5)])
```

Pour les variables numériques

```
X_num = np.copy(X[:, col_num])
X_num[X_num == '?'] = np.nan
X_num = X_num.astype(float)

imp_num = Imputer(missing_values=np.nan, strategy='mean')
X_num = imp_num.fit_transform(X_num)
```

- ***Traitement de variables catégorielles*** **:** Pour pouvoir utiliser les variables catégorielles dans les algorithmes d'apprentissage supervisé de votre fonction ***run_classifiers***, une solution consiste à transformer chaque variable catégorielle avec ***m*** modalités en ***m*** variables binaires dont une seule sera active. Pour cela utiliser le module ***OneHotEncoder*** de Scikit-learn pour encoder les 9 variables catégorielles de votre jeu de données.

  ```
  X_cat_bin = OneHotEncoder().fit_transform(X_cat).toarray()
  ```

- ***Construction de votre jeu de données*** **:** Construire votre jeu de données en concaténant à la fois les données catégorielles transformées et les données continues normalisées. L'objectif étant de bien préparer les données originales afin d'obtenir le meilleur jeu de données sur lequel vos classifieurs seront appris. Exécuter maintenant votre fonction ***run_classifiers*** sur vos nouvelles données.