Dpt Informatique

Année universitaire	**2018-2019**		
Filière	**Intelligence Artificielle**	**Année**	**M2**
Matière	**Machine Learning**		
Enseignant	**Haytham Elghazel**		
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**		

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**.

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

```
import numpy as np
np.set_printoptions(threshold=np.nan,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***), nombre d'exemples positifs et négatifs (*c.f.* ***hist***).
   - Pour éviter d'avoir un résultat biaisé du classifieur que nous allons construire, séparer les données en deux partie 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***).
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

L'objectif est à présent de comparer les résultats obtenus à l'aide de ces deux simples algorithmes sur ce jeu de données. Cette comparaison s'appuiera sur l'estimation de l'***accuracy*** et le ***meilleur critère*** entre le ***Rappel*** et la ***Précision*** dont vous pensez qu'il est le plus adéquat pour ce cas d'application.

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. Vous pouvez également tester le module ***MinMaxScaler***. Exécuter à nouveau votre code sur vos données une fois normalisées. Interpréter les résultats obtenus en les comparant avec ceux de la question précédente.

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 *k* 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 ?

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.

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

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X1_scale, Y1)
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(X_train_scale.size/len(X_train_scale)) + 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()
```

   Déterminer ensuite le nombre de variables à garder en exécutant le code suivant

```
KNN=KNeighborsClassifier(n_neighbors=5)
scores=np.zeros(X1_scale.shape[1]+1)
for f in np.arange(0, X1_scale.shape[1]+1):
   X1_f = X1_scale[:,sorted_idx[:f+1]]
   X2_f = X2_scale[:,sorted_idx[:f+1]]
   KNN.fit(X1_f,Y1)
   YKNN=KNN.predict(X2_f)
   scores[f]=np.round(accuracy_score(Y2,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()
```

6. **Paramétrage des classifieurs** : Dans cette partie, vous allez utiliser la fonction **GridSearchCV** de scikit-learn afin de tuner les paramètres des deux algorithmes *k-plus proches voisins* et *Arbre de décision*.

7. **Création d'un pipeline :** Dans cette partie vous allez automatiser l'enchainement des traitements effectués précédemment (Normalisation, ACP et Construction du classifieur) dans un pipeline (module ***pipeline.Pipeline*** de Scikit-learn)

8. **Comparaison de plusieurs algorithmes *d'apprentissage* :** Utiliser ensuite sur votre jeu de données les algorithmes d'apprentissage supervisé suivants :
    - NaiveBayesSimple
    - Un arbre CART
    - Un arbre ID3
    - 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 50 classifieurs par exemple
    - AdaBoost avec 50 classifieurs par exemple
    - Random Forest avec 50 classifieurs par exemple
    - Etc.

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 5 fold cross-validation du **critère qu'il vous semble le plus pertinent** entre le ***rappel*** et la ***précision*** pour cette tâche du crédit scoring. Il faut afficher aussi la moyenne et l'écart type.
- le temps d'exécution de l'algorithme d'apprentissage (*c.f.* ***time.time()***)

Note :

Afin de mieux comparer plusieurs algorithmes sur une même validation croisée, il est préférable de passer par un dictionnaire dans lequel vous mettez la liste des algorithmes à comparer et d'utiliser le code suivant :

```
clfs = {
  'RF': RandomForestClassifier(n_estimators=50),
  '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)))
```

Créer une fonction Python ***run_classifiers*** qui permettra de lancer la comparaison des algorithmes de classification supervisée et qui prendra en paramètre un dictionnaire **clfs**, le tableau de données caractéristiques **X** et la target **Y**. (c.f. ***def***).

Exécuter cette fonction et interpréter les résultats obtenus.

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 ensembles 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 quant 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.

III. *Apprentissage supervisé sur des données textuelles* : Feature engineering et Classification

L'objectif dans cette partie est de faire une étude comparative entre plusieurs algorithmes d'apprentissage supervisé sur un jeu de données textuelles de SMS ("SMSSpamCollection.data") pour prédire si un message est un spam ou pas. Pour plus d'informations sur ce jeu de données, vous pouvez visiter le lien suivant (https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection).
Chaque ligne dans ce fichier correspond à la classe d'un SMS ('***spam'*** ou '***ham***') et le texte du SMS. Avant de pouvoir appliquer votre fonction ***run_classifiers*** sur ce jeu de données, il faut faire un pré-traitement de ces données afin de transformer le texte de chaque SMS en un vecteur de données (on parle ici du ***Text Mining***). Vous allez utiliser pour cela la représentation *bag-of-words*. Une fois les données chargées, les étapes suivantes doivent être exécutées dans l'ordre en suivant la documentation dans la page officielle du package ***sklearn.feature_extraction.text*** de Scikit-learn (http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction) :

- ***CountVectorizer*** **:** pour splitter chaque texte en différents mots clés (termes), supprimer les mots clés vides (stopwords) et calculer la matrice de co-occurences. Il est possible de garder que les mots clés les plus fréquents. Exécuter ensuite votre fonction ***run_classifiers*** sur vos données et

interpréter les résultats obtenus.

- ***Tf–idf term weighting*** **:** une mesure statistique utilisée pour la normalisation et la pondération de l'importance d'un terme contenu dans un document, relativement à toute la collection des documents (ici les SMS). Exécuter ensuite votre fonction ***run_classifiers*** sur vos données et interpréter les résultats obtenus en les comparant à ceux obtenus à l'étape précédente.

- ***TruncatedSVD :*** une méthode de réduction de dimensions pour les matrices très creuses (sparses en anglais) qui permettra d'améliorer la représentation vectorielle des textes des SMS par une indexation sémantique latente. Cette dernière permet d'établir des relations entre un ensemble de documents et les termes qu'ils contiennent, en construisant des "concepts" liés aux documents et aux termes. Elle permettra entre autre de résoudre les problèmes de synonymie (plusieurs mots avec un seul sens) et de polysémie (un seul mot avec plusieurs sens). La SVD est bien documentée en Scikit-learn. Exécuter ensuite votre fonction ***run_classifiers*** sur vos données et interpréter les résultats obtenus en les comparant à ceux obtenus à l'étape précédente.

- ***Pipeline*** **:** Automatiser l'enchainement des traitements effectués précédemment dans un pipeline

- ***Application sur un autre jeu de données*** **:** Appliquer votre pipeline sur le jeu de données Yelp (yelp-text-by-stars.csv) contenant un ensemble d'avis participatifs notés sur des lieux (restaurants, hôtels, etc.) donnés par des utilisateurs sur le site Yelp.com