Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université de Carthage Institut National des Sciences Appliquées et de Technologie		Année Universitaire : 2024 - 2025 Module : Machine Learning Audience : 4ème RT Enseignante : Hazar MLIKI

TP 4 : Apprentissage Supervisée

Objectifs

- Construire des modèles de prédiction en utilisant des algorithmes basés sur l'apprentissage supervisée.
- Evaluer et optimiser les modèles générés.

K-plus proche voisin (KNN) (Reconnaissance de chiffres)

MNIST est une base de données étiquetée. Pour chaque chiffre, une représentation sous forme d'image (de taille 8x8 pixels) ainsi que son étiquette sont fournies. Le jeu de données MNIST comporte 1797 chiffres que nous diviserons en deux sous-ensembles : Apprentissage et test.

La fonction *load_digits* charge le jeu de données MNIST. Par la suite, on crée un *DataFrame* "dig" alimenté par ce jeu de données.

Exemple

```
from sklearn.datasets import *
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

digit = load_digits()
dig = pd.DataFrame(digit['data'][0:1700])
print(dig.head())
print(digit.keys())
```

On remarque que le DataFrame contient 64 colonnes (de 0 à 63). Ce qui représente l'ensemble des valeurs de la matrice de taille 8×8 (pixels).

Dans notre cas, nous sommes intéressés par les colonnes : data et target. La variable "data" est la représentation en GrayScale d'un chiffre, et le target, représente son étiquette. Pour afficher quelques images de la base de données MNIST, remplacez-le (i) par le chiffre que vous voulez l'afficher.

```
plt.imshow(digit['images'][i], cmap='Greys_r')
```

```
plt.show()
```

Pour diviser le jeu de données en 75% de Training Set et 25% de Testing Set, on utilise la fonction **`train_test_split ()`**.

```
train_x = digit.data # input variables
train_y = digit.target # les étiquettes (output variable)
#découpage du jeu de données (0.25 pour indiquer 25%)
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,test_size=
0.25)
```

Enfin, pour entraîner un K-NN, on utilise la librairie SickitLearn qui propose la classe **KNeighborsClassifier**

```
### Train
KNN = KNeighborsClassifier(7) # on utilise 7 voisins
KNN.fit(x_train, y_train)
```

Pour mesurer la performance (exactitude) de notre classifieur à 7 voisins, on utilise les 25% des observations du jeu de test.

```
###Test
print(KNN.score(x_test,y_test))
```

Question 1 : Variez la valeur de k et étudiez la performance de KNN.

Bayésien Naïf (SPAM Filter)

La base de données « emails » est composée des mails Spam et Non-spam (Ham). En utilisant les modules « OS » de Python, on charge la base de données.

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
print(os.listdir("C:/DB"))
df = pd.read_csv('C:/DB/spam.csv', encoding='latin-1')[['v1', 'v2']]
df.columns = ['label', 'message']
print(df.head())
```

Question 2 : Décrire l'ensemble de données « emails » et visualiser le nombre de ham/spam.

Pour nettoyer et normaliser le texte des emails, il faut :

- Supprimer les ponctuations
- Supprimer tous les mots vides
- Appliquer la technique de « stemming » (obtenir la forme normale du mot). Par exemple, « driving car» et « drives car» deviennent « drive car »

Pour ce faire, il faut installer le toolkit de Natural Language « **nltk** », comme suit :

```
import nltk
nltk.download('stopwords')
```

La méthode « **process** » permet de retourner un texte normalisé sous forme de jetons (lemmes)

```
import string
from nltk.corpus import stopwords #Natural Language Toolkit
from nltk import PorterStemmer as Stemmer
def process(text):
    # lowercase it
    text = text.lower()
    # remove punctuation
    text = ''.join([t for t in text if t not in string.punctuation])
    # remove stopwords
    text = [t for t in text.split() if t not in
stopwords.words('english')]
    # stemming
    st = Stemmer()
    text = [st.stem(t) for t in text]
    # return token list
    return text
# Testing
print(process('It\'s holiday and we are playing cricket. Jeff is playing
very well!!!'))
# Test avec notre dataset
print(df['message'][:20].apply(process))
```

Convertir chaque message en vecteurs en utilisant le modèle « de sac de mots » (bag of words). TfidfVectorizer transforme une collection de documents texte (corpus SMS) en une matrice 2D : une dismension qui représente les documents et l'autre dimension représente chaque mot unique dans le corpus. Ainsi, pour chaque message, nous extrayons les vecteurs normalisés ayant une longueur égale à la taille du vocabulaire (nombre de termes uniques de l'ensemble du corpus SMS)

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidfv = TfidfVectorizer(analyzer=process)
data = tfidfv.fit_transform(df['message'])
mess = df.iloc[2]['message']
print(mess)
print(tfidfv.transform([mess]))
#visualiser
j = tfidfv.transform([mess]).toarray()[0]
print('index\tidf\ttfidf\tterm')
for i in range(len(j)):
    if j[i] != 0:
       print(i, format(tfidfv.idf_[i], '.4f'), format(j[i], '.4f'),
tfidfv.get_feature_names_out()[i],sep='\t')
```

La classe **MultinomialNB** (NB pour Naïve Bayes) permet de lancer le processus d'apprentissage avec le bayésien Naïf.

```
########### TRAIN ############
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
spam_filter = Pipeline([
    ('vectorizer', TfidfVectorizer(analyzer=process)), # messages to
weighted TFIDF score
    ('classifier', MultinomialNB())   # train on TFIDF vectors with NB
])
x_train, x_test, y_train, y_test = train_test_split(df['message'],
df['label'], test_size=0.20, random_state = 21)
spam_filter.fit(x_train, y_train)

######## TEST ##############
predictions = spam_filter.predict(x_test)
count = 0
for i in range(len(y_test)):
    if y_test.iloc[i] != predictions[i]:
        count += 1
print('Total number of test cases', len(y_test))
print('Number of wrong of predictions', count)
```

Question 3 :

a) Quel est le nombre de prédictions erronées ?

b) Visualisez les prédictions erronées qui ont été classées comme du ham ?

c) Vérifiez que le message suivant est un ham ou un spam : *'Your cash-balance is currently 500 pounds - to maximize your cash-in now, send COLLECT to 83600.*'

```
#Calculez Rappel  Précision F1 score
from sklearn.metrics import classification_report
print(classification_report(predictions, y_test))
```

Régression Linéaire (Prédiction du gain)

Supposons que vous êtes le chef de direction d'une franchise de camions ambulants (Food Trucks). Vous envisagez différentes villes pour ouvrir un nouveau point de vente. La chaîne a déjà des camions dans différentes villes et vous avez des données pour les bénéfices et les populations des villes. Vous souhaitez utiliser ces données pour choisir la ville où il est plus opportun d'ouvrir un nouveau point de vente.

Ce problème est de type **apprentissage supervisé modélisable par un algorithme de régression linéaire**. Il est de type **supervisé** car pour chaque ville ayant un certain

nombre de population (variable prédictive X), on a le gain effectué dans cette dernière (la variable qu'on cherche à prédire : Y).

Population de la ville en 10 000	Profit en
6.1101	17.592
5.5277	9.1302
8.5186	13.662

Pour résoudre ce problème, il faut prédire le profit (la variable Y) en fonction de la taille de la population (la variable prédictive X). On commence par charger les données contenues dans le fichier CSV. La fonction *read_csv()*, renvoie un DataFrame de deux dimensions contenant, respectivement, la taille de population et les profits effectués.

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
df = pd.read_csv("C:\\BD_ML\\univariateLinearRegression.csv")
print(len(df))
```

Pour pouvoir utiliser les librairies de régression de Python (**stats.linregress**), il faut séparer les deux colonnes dans deux variables.

- La fonction **len()** permet d'obtenir la taille d'un tableau.
- La fonction **iloc** permet de récupérer une donnée via sa position.
- **iloc[0:len(df),0]** permet de récupérer toutes les données de la ligne 0 à la ligne 97 se trouvant à la colonne d'indice 0.

```
#selection de la première colonne de la base (la taille de la population)
X = df.iloc[0:len(df),0]
#selection de la deuxième colonne de la base (le profit effectué)
Y = df.iloc[0:len(df),1]
```

Question 4 : Visualiser les données avec un scatter plot. Interprétez le résultat.

Pour appliquer la **régression linéaire à une variable (univariée),** on utilise le module **scipy.stats** qui dispose de la fonction **`linregress`**, qui permet de faire la régression linéaire.

```
#linregress() renvoie plusieurs variables de retour. On s'interesse au
slope et intercept
slope, intercept, r_value, p_value, std_err = stats.linregress(X, Y)
```

En utilisant la fonction F(x) = ax+b (F(x) = slope*x+intercept), on peut effectuer une prédiction sur nos 97 populations.

```
def F(x):
    return slope * x + intercept
#la variable fitLine est un tableau de valeurs prédites depuis la tableau
de variables X
fitLine = F(X)
plt.scatter(X,Y)
plt.plot(X, fitLine, c='r')
plt.show()
```

Pour effectuer la prédiction du gain d'une population de taille 20.27, appliquons la fonction **F*()*** qu'on a défini précédemment :

```
print (F(20.27))
```

Régression multivariée (Prédiction du prix)

Supposons que vous souhaitez vendre votre maison qui se trouve à Tunis. Pour estimer le bon prix, vous disposez de l'historique des prix de vente effectuées à Tunis. Ainsi, pour chaque maison vendu (une observation), on dispose des données suivantes :

- La taille de la superficie en pieds² *(*Un pied² fait environ 0,092 m²)
- Le nombre de chambre de la maison
- Le prix

TAILLE EN PIEDS²	NOMBRE DE CHAMBRES	PRIX DE VENTE EN $
2104	3	329900
1600	3	329900
2400	3	369000
1416	2	232000

On commence par charger les données contenues dans le fichier Excel. La fonction ***read_excel(),*** renvoie un DataFrame de trois dimensions contenant respectivement : la taille de la maison, le nombre de chambres et le prix auquel elle a été vendu.

```
import pandas as pd
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
import numpy as np

from mpl_toolkits.mplot3d import Axes3D
```

```
import matplotlib.pyplot as plt
df = pd.read_excel("C:\BD_ML\multivariateLinearRegression.xlsx")
print(df.head())
```

Puis, on va récupérer le prix : les valeurs observées pour la variable Cible et les variables prédictives : La superficie en pieds² et le nb chambre.

```
#Récupérer le prix : les valeurs observées pour la variable Cible
Y = df["prix"]
#Récupérer les variables prédictives : La superficie en pieds² et le nb
chambre
X = df[['taille_en_pieds_carre','nb_chambres']]
```

Question 5 : Visualiser les données avec un scatter plot. Interprétez le résultat.

Dans cette base, on note que les variables prédictives ont des ordres de grandeurs très différents. En effet, le nombre de chambre d'une maison est généralement compris entre 1 et 10 alors que la superficie se compte en quelques milliers de pieds^2. Pour appliquer l'algorithme Multivariate Regression, il est nécessaire que les variables prédictives soient du même ordre de grandeur**.** Généralement, il faut que la valeur de chaque variable prédictive soient compris (approximativement) entre **-1 et 1**. Le package sklearn.preprocessing propose la classe **`StandardScaler`** qui permettra de faire du features scaling sur toutes les variables prédictives.

```
scale = StandardScaler()
X_scaled = scale.fit_transform(X[['taille_en_pieds_carre',
'nb_chambres']].as_matrix())
print(X_scaled)
```

Enfin, pour appliquer l'algorithme de régression linéaire multivariée, on utilise ici l'**Ordinary Least Squares (OLS).** En effet, OLS est une méthode pour estimer une variable cible dans un modèle de régression linéaire. Ainsi, OLS va minimiser la somme des carrés des différences entre les réponses observées (de notre Training Set) et ceux prédits par la fonction linéaire appliquée à nos variables prédictives.

```
model = sm.OLS(Y, X).fit()
print(model.summary())
```

La fonction prédictive obtenue :

$$prix\_estimé = F(superficie, nb\_chambres) = \varepsilon + \alpha * superficie + \beta * nb\_chambres$$

Pour tester notre modèle, on va lui fournir la taille de la maison et le nombre chambre.

```
def predict_price_of_house(taille_maison, nb_chambre):
    return 140.8611 * taille_maison + 1.698e+04 * nb_chambre
print(predict_price_of_house(4500,5))
```

Régression polynomiale (Prédiction du salaire)

Nous allons utiliser les données de salaire des employés. Dans cet ensemble de données, nous avons trois colonnes : Position, Level and Salary. On commence par charger les données contenues dans le fichier Excel.

```
# Importez les libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

df_sal = pd.read_csv(' C:\BD_ML\Position_Salaries.csv')
df_sal.head()
```

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

Question 6 : Récupérez une description de cette base de données. Visualisez la distribution des données avec Seaborn plot. Dressez un scatter plot pour examiner la relation entre le salaire et le niveau. Interprétez le résultat.

Divisez l'ensemble de données en variables dépendantes/indépendantes : L'expérience (X) est une variable indépendante ; le salaire (y) dépend de l'expérience.

```
X = df_sal.iloc[:, 1:-1].values  # independent
y = df_sal.iloc[:, -1].values  # dependent

# Train linear regression model
lr = LinearRegression()
lr.fit(X, y)

# Train polynomial regression model
pr = PolynomialFeatures(degree = 4)
X_poly = pr.fit_transform(X)
lr_2 = LinearRegression()
lr_2.fit(X_poly, y)
```

Pour tester une valeur de y (salaire) en fonction de X (poste, niveau) avec le modèle généré.

```
# TEST
y_pred_lr = lr.predict(X)           # Linear Regression
y_pred_poly = lr_2.predict(X_poly)  # Polynomial Regression
```

Question 7

a) Dressez les plots de prédiction pour visualisez les données réelles avec la régression linéaire ainsi que la régression polynomiale.

b) Soit un level égal à 7.5, examinez quel salaire les modèles générés prédisent et à quel point ils ont précis.

```
# Predict with linear regression
print(f'Linear Regression result : {lr.predict([[7.5]])}')
# Predict with polynomial regression
print(f'Polynomial Regression result :
{lr_2.predict(pr.fit_transform([[7.5]]))}')
```

Régression logistique (classification des fleurs d'iris)

Pour charger le jeu de données Iris, on utilise la méthode **load_iris()** du package *datasets.*

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
#chargement de base de données iris
iris = datasets.load_iris()
```

Sachant que le *dataset Iris* se compose de quatre features (variables explicatives), on n'utilisera que les deux premières features à savoir : *Sepal_length et Sepal_width.*

Également, le jeu IRIS se compose de trois classes, les étiquettes peuvent donc appartenir à l'ensemble {0, 1, 2}. Puisque la régression logistique est un algorithme de classification binaire, on va **re-étiqueter** les fleurs ayant le label 1 et 2 avec le label 1. Ainsi, on se retrouve avec un problème de classification binaire.

- La variable X comporte l'ensemble des observations qui serviront à l'apprentissage.
- La variable y comporte l'étiquette de chacune de ces observations.

```
# choix de deux variables : Utiliser les deux premières colonnes afin d'avoir un problème de classification binaire
X = iris.data[:, :2]
y = (iris.target != 0) * 1 # re-étiquetage des fleurs
```

Question 8 : Visualisez les données dans ce jeu de données. Interprétez le résultat.

L'instruction **model.fit(X, Y)** de la bibliothèque Scikit Learn permet d'entraîner le modèle via une régression logistique.

```
## TRAIN
model = LogisticRegression(C=1e20)
model.fit(X, y)
```

Pour prédire la classe de fleurs d'IRIS, on fournit des valeurs pour les deux variables ("Sepal Length" et "Sepal Width") et on demande au modèle prédictif de nous indiquer la classe de la fleur.

Question 9 : Testez les valeurs suivants [5.5, 2.5], [7, 3], [3,2] et [5,3] et déterminez les classes attribuées par le modèle. Vérifiez les résultats obtenus sur le plot scatter dressé au début.

Arbre de décision

Installer les packages suivants.

```
pip install scikit-learn
pip install graphviz # ou conda install python-graphviz
pip install pydotplus
pip install six
```

Importer le jeu de données « diabetes.csv » à partir de Kaggle data set « https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database/»

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
import matplotlib.pyplot as plt

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']
diabetes_data= pd.read_csv("diabetes.csv", header=None, names=col_names)
```

Question 10 : Prétraitement des données

a) Dressez les graphes nécessaires pour mieux comprendre vos données.

b) Quel est le type du problème de classification ?

c) Divisez les colonnes en deux types de variables : cibles (Y) et caractéristiques (X).

Question 11 : Construction du modèle

a) Pour construire le modèle, divisez l'ensemble de données en un ensemble d'apprentissage et un ensemble de test en utilisant la fonction « `train_test_split()` ».

b) Créez un modèle d'arbre de décision en utilisant la fonction « DecisionTreeClassifier() » du package Scikit-learn qui fait recours à l'indice de Gini pour calculer l'impureté des nœuds.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Question 12 : Évaluez le modèle en calculant son Accuracy (metrics.accuracy_score) et en affichant la matrice de confusion (confusion_matrix).

Question 13 : Pour visualiser l'arbre de décision générée, la fonction « export_graphviz » convertit le modèle d'arbre de décision en un fichier dots, et la fonction « pydotplus » convertit ce fichier en une image png.

a) Quelles informations affichées au niveau de chaque nœud ?

b) L'arbre affiché est-il élagué ?

```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Question 14 : Pour optimiser la performance des arbres de décision, il est possible de modifier les paramètre de la fonction « DecisionTreeClassifier() » :

- **critère** : (par défaut="gini") « gini » pour l'indice de Gini et « entropie » pour le gain d'information.
- **splitter** : (default="best"). Ce paramètre nous permet de choisir la stratégie de split à savoir « best » pour choisir la meilleure répartition et « random » pour choisir la meilleure répartition aléatoire.

- **max_degree** : (par défaut=Aucun). Ce paramètre permet de définir la profondeur maximale d'un arbre. Si « aucun » est défini, les nœuds sont développés jusqu'à ce que toutes les feuilles contiennent moins d'échantillons que min_samples_split. Une valeur plus élevée de la profondeur maximale provoque unover-fittingt, et une valeur inférieure entraîne un under-fitting.

  a) Appliquez l'entropie pour le calcul de l'impureté des nœuds et fixez la profondeur à 3.

  b) Générez de nouveau le graphique de l'arbre de décision. Que constatez-vous ?

  c) Cette optimisation est considérée comme un pré-élagage ou un post-élagage ?