Enseignantes : A. NAJJAR-I. BEN OTHMEN - F. JENHANI **TP : Z. ZOUAGHIA - I. BEN AYCHA**	**TP3** **Machine Learning**	**Classe : 3ème GLSI**

Dans ce TP, nous utilisons la base "**pima-indians-diabetes.data.csv**".

1- Importer les données à partir du fichier.
2- Diviser l'ensemble de données en un ensemble d'apprentissage qui contient 75% des observations et un ensemble de test (25%).

Partie 1 : Régression Logistique binaire

1- Nous souhaitons étudier l'effet du choix des attributs sur la qualité de la régression. Pour cela déterminer, puis comparer, la précision du modèle de régression logistique binaire en utilisant chacun des attributs suivants :
   a. Tous les attributs
   b. Indice de masse corporelle (**BMI**)
   c. Age + Tension artérielle (**BloodP**) + Concentration du plasma en glucose (**PlGlcConc**)
2- Afficher les données et la courbe de régression, effectuer en utilisant l'attribut "BMI", sur un même graphe. Tester aussi les autres attributs. Conclure.

```
import pandas as pd            #data loading and manipulation
import matplotlib.pyplot as plt  #plotting
import seaborn as sns          #statistical plotting
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

#Question: Importer les données
diabetes = pd.read_csv("pima-indians-diabetes.data.csv")
Columns = ['NumTimesPrg', 'PlGlcConc', 'BloodP', 'SkinThick', 'TwoHourSerIns', 'BMI',
'DiPedFunc', 'age', 'HasDiabetes']
diabetes.columns=Columns
diabetes.head()
diabetes.info()
diabetes.describe()
```

```
#Partie1
#Question1
# a) Tous les attributs
feature_cols1 = ['NumTimesPrg', 'PlGlcConc', 'BloodP', 'SkinThick', 'TwoHourSerIns', 'BMI', 'DiPedFunc', 'age']
# b) Indice de masse corporelle (BMI)
feature_cols2 = [ 'BMI']
# c) Age + Tension artérielle (BloodP) + Concentration du plasma en glucose (PlGlcConc)
feature_cols3 = [ 'PlGlcConc', 'BloodP','age']
# Utilisation de la variable indépendante "BMI" comme input
X = diabetes[feature_cols1] # Features
y = diabetes.HasDiabetes # Target variable: utilisation de la variable dépendante HasDiabetes comme variable à prédire
#Découpage de l'ensemble des données: train set and test set
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
# instantiate the model (using the default parameters)
logreg = LogisticRegression()
# fit the model with data
logreg.fit(X_train,y_train)
# Faire la prédiction
y_pred=logreg.predict(X_test)
#Affichage des valeurs prédites
print("y_pred= ", y_pred)
#Evaluation de la qualité du modèle
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# Plot pedigree and diabetes and add the logistic fit
sns.regplot(x = "BMI", y = "HasDiabetes", data = diabetes,logistic = True)
# Display the plot
plt.show()
```

Partie 2 : Les réseaux de neurones artificiels

On souhaite entraîner et tester le Réseau de Neurones Artificiels de type **Perceptron Multi-Couches (MLP)**.

1- Essayer différents attributs et différentes combinaisons d'attributs
2- Essayer plusieurs architectures du MLP
   - Changer à chaque fois la fonction d'activation
   - Changer à chaque fois le nombre de couches
3- Discuter les résultats obtenus en calculons, à chaque fois, la précision de la prédiction.

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import pandas as pd
diabetes = pd.read_csv("pima-indians-diabetes.data.csv")
Columns = ['NumTimesPrg', 'PlGlcConc', 'BloodP', 'SkinThick', 'TwoHourSerIns', 'BMI', 'DiPedFunc', 'age', 'HasDiabetes']
diabetes.columns=Columns

feature_cols1 = ['NumTimesPrg', 'PlGlcConc', 'BloodP', 'SkinThick', 'TwoHourSerIns', 'BMI', 'DiPedFunc', 'age']
feature_cols2 = [ 'BMI']
feature_cols3 = [ 'PlGlcConc', 'BloodP','age']

X = diabetes[feature_cols3] # Features

y = diabetes.HasDiabetes # Target variable

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)

model = MLPClassifier(hidden_layer_sizes=150,activation='logistic',max_iter=1000)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Réseau de neurones artificiels

Inspiré du fonctionnement des neurones biologiques et est rapproché des méthodes statistiques.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

alpha : *float, default=0.0001*
L2 penalty (regularization term) parameter.

batch_size : *int, default='auto'*
Size of minibatches for stochastic optimizers. If the solver is 'lbfgs', the classifier will not use minibatch. When set to "auto", `batch_size=min(200, n_samples)`.

learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*
Learning rate schedule for weight updates.

Only used when `solver='sgd'`.

max_iter : *int, default=200*
Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*
Learning rate schedule for weight updates.