ESIEE [it]
School of Digital Intelligence

Lab 2 – Introduction Intelligence Artificielle

JEAN-PIERRE Matthieu
23/01/2025

Objet du Lab2 : Comprendre Apprentissage Supervisé et apprentissage non supervisé

Contexte :

Le domaine de l'apprentissage automatique est l'un des domaines les plus dynamique de l'intelligence artificielle.

1ère étape il faut prétraiter les données pour cela on va utiliser les imports la module numpy as np et un importer un module du package sklearn « processing ».

```
import numpy as np
from sklearn import preprocessing
```

Nous allons créer une liste de liste chaque échantillon da la sous liste va être représenter par 3 caractéristiques. Ce va permettre à l'ordinateur de comprendre qu'il travaille sur une matrice

```
input_data = np.array([[5.1, -2.9, 3.3],
                       [-1.2, 7.8, -6.1],
                       [3.9, 0.4, 2.1],
                       [7.3, -9.9, -4.5]])
```

La binarisation de la matrice c'est dire insérer des 0 et des 1 dans matrice pour cela on va utiliser le mot clé processing.binarser avec un seuil à 2 qui est apposer on va lui mettre comme paramètre notre variable précédente input_data. Et pour finir on va appeler la variable data_banarized afin de print le résultat.

Enlèvement moyen :

L’enlèvement est un est technique qui permet de centrée chaque caractéristique sur zéro. Une fois les données centrées on récupère les meilleur donnée grâce à cette près technique de prétraitement.

Le data_scaled.mean va afficher les donnée après centrage proche de 0.

Et après on recupere Std deviation

Le code « Before » va afficher la moyenne et l’écart type avant transformation.

Le code « After » va afficher la moyenne et le l’écart type qui va être de 1. Ces techniques de normalisation vont permettre a d’autre algorithme de converger plus rapidement.

Mise a l’échelle :

On utilise MiniMaxScaler pour que les données se situe sur une plage entre 0 et 1. On applique minimax sur les donnée brut input data, puis on va transformer les données et enfin afficher les donner pour un résultat.

```
Min max scaled data:
 [[ 0.74117647  0.39548023   1.        ]
 [ 0.           1.          0.        ]
 [ 0.6          0.5819209   0.87234043]
 [ 1.           0.          0.17021277]]
```

Normalisation :

Data_normalized_l1 fait référence aux moindres écarts absolus et permet de s’assurer que la somme des avaleur absolues est égale à 1.

Data_normalizes_l2 va faire référence au carrée et va s’assurer que la somme des carrés est égale à 1.

Data_normalizes_l2 est un meilleur choix s’il y a beaucoup de valeur aberrante.

```
# Normalize data
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nL1 normalized data:\n", data_normalized_l1)
print("\nL2 normalized data:\n", data_normalized_l2)
```

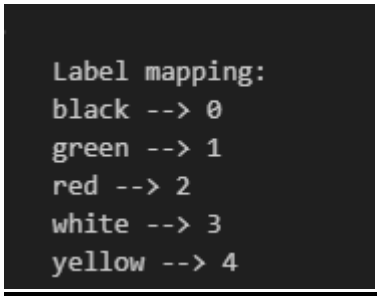
Codage des étiquettes :

On va définir une liste sous forme de texte qui sera par la suite encodé pour être sous format numérique qui sera compris par l'ordinateur.

LabelEncoder est utiliser pour convertir des étiquettes réelles en valeur numérique

Encoder.fit va entrainer les étiquettes crée précédemment input_labels.

Remarque : a la sortie du code python on observe que la liste indexée avec la boucle for ne retourne pas dans l'ordre de l'étiquette input_label. C'est parce que python va trier alphabétiquement le premier caractère de chaque élément de la liste donc black sera le premier élément de label mapping :



Le décodage on va commencer par crée une liste (encoded_values) qui va contenir le résultat de l'encodage de la première étiquette.

Dans une variable decoded_values nous allons inverser les processus d'encodage avec les mots clés inverse_transform(encode_values). Et pour finir nous allons afficher

```
# Decode a set of values using the encoder
encoded_values = [3, 0, 4]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list(decoded_list))
```

d

```
Label mapping:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels = ['green', 'red', 'black']
Encoded values = [np.int64(1), np.int64(2), np.int64(0)]

Encoded values = [3, 0, 4]
Decoded labels = [np.str_('white'), np.str_('black'), np.str_('yellow')]
```

Classification

```
# Define sample input data
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5], [6, 5], [5.6, 5], [3.3,
0.4], [3.9, 0.9], [2.8, 1], [0.5, 3.4], [1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])
```

Ce script va créer un tableau x qui contenir des valeurs par deux. Et la deuxième y va être un tableau qui va classer les éléments de x par classe. Exemple [3.1, 7.2] correspond à l'échantillon de la classe 0.

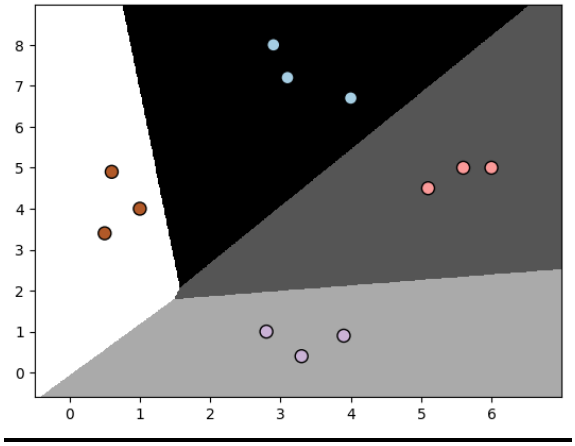
Cela permet d'entrainer un modèle d'apprentissage relier des caractéristiques (taille, poids)

Création de l'objet solveur, libliear provient langage C, c=1 va limiter a 1 si on dépasse le 1 on aura moins de régularisation ce qui évitera le surapprentissage et donc minimiser l'erreur d'entrainement.

```
# Create the logistic regression classifier
classifier = linear_model.LogisticRegression(solver='liblinear', C=1)
```

On va entrainer le classificateur avec le fit, et enfin nous allons l'afficher visualize_classifier qui va prendre en paramètre classifier, plus les tableaux X et Y.

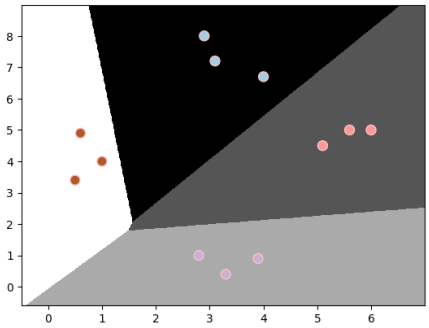
Résultat final !!!



Le Fichier régression logistiques 2 :

Ce script est similaire au script précédent, dans ce code python nous allons créer une fonction qui va faire la classification.

1- Importation de bibliothèque et modules nécessaire pour notre script.
2- Définition de notre fonction qui prend en entrée un classifier et des données x, y.
3- Définition de la grille de maillage pour limiter notre graphe.
4- Nous allons définir le pas de maillage plus le pas sera important et moin la précision sera à l'inverse plus le pas est petit plus la précision sera grand dans notre cas le pas est petit.
5- Combinaison de deux tableaux en un prédire avec la bibliothèque numpy.
6- La construction de la figure se fais avec clés de matplot. Figure().
7- Plt.xlim et plt.ylim vont fixer la limite des axes x et y . xticks et yticks vont accentuer les graphes pour une meilleurs visibilité



Classificateur Naïves Baise

Méthode probabiliste, qui permet de calculer un la probabilité d'une classe donnée en fonction des caractéristiques observés.

1- Import les modules des packages python.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

2- Charger les donner avec un fichier .txt

```
# Input file containing data
input_file = 'data_multivar_nb.txt'
```

3- Délimiter les données.
4- Crée la Naïve Bayes classifier avec GaussianNB()

```
# Create Naïve Bayes classifier
classifier = GaussianNB()
```

5- Entrainer le 'classifier' avec fit qui va utiliser deux paramètres x et y. X va sélectionner toute les ligne et colonnes sauf la dernière colonne. La variable y va sélectionner toutes les lignes et la dernière colonne du fichier txt.

```
# Train the classifier
classifier.fit(X, y)
```

6- Utiliser le modelé entrainer pour prédire comment le modèle s'adapte aux données.

```
# Predict the values for training data
y_pred = classifier.predict(X)
```

7- Fais le calcul de la précision avec un pourcentage avec un arrondie a 2. Et afficher le pourcentage à l'aide du print(« »).

```
# Compute accuracy
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naïve Bayes classifier =", round(accuracy, 2), "%")
```

8- Appel de la fonction visual_classifier pour voir notre résultat.

```
# Visualize the performance of the classifier
visualize_classifier(classifier, X, y)
```

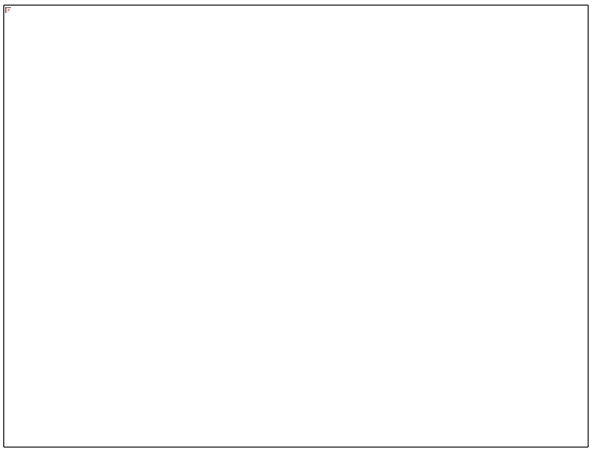
9- Séparer les donnée (traint_test_split), création du modèle GaussianNB(), entrainement (.fi)t , prédiction (.predict).

```
# Split data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)
```

10- Affichage du pourcentage du classifier en pourcent à 2 chiffres après la virgule.

```python
# compute accuracy of the classifier
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")
```

```
...   Accuracy of Naïve Bayes classifier = 99.75 %
```



Matrice de confusion

La matrice de confusion est une figure ou un tableau utilisé pour décrire les performances d'un classificateur. Une matrice de confusion est une sorte de matrice qui nous permets de voir la comparaison entre le une classe prédite et une classe réelle. En comparant nos différentes classes on peut avoir nous pouvons tomber sur plusieurs scénarios.

Exemples :

Prédiction Vrai >> Réel Faux

Prédiction Faux >> Réel Faux

Prédictions Vrais >> Réel Vrai

Prédiction Faux >> Réel Faux

Remarque ces scénario sont entraînent des conséquences sur les pourcentages. L'exigence du pourcentage de réussite de la matrice dépendra du secteur d'activé. On demandera un meilleur pourcentage dans le domaine spatial que dans la grande distribution ;

Voici les étapes détailler pour crée une matrice de confusion

1- On importe les modules

```
numpy as np
matplotlib.pyplot as plt
clearn.metrics import confusion_matrix
clearn.metrics import classification_report
```

2- Créations des étiquettes.

```
# Define sample labels
true_labels = [2, 0, 0, 2, 4, 4, 1, 0, 3, 3, 3]
pred_labels = [2, 1, 0, 2, 4, 3, 1, 0, 1, 3, 3]
```

3- Construction de la matrice de confusion avec 'confusion_matrix'

```
# Create confusion matrix
confusion_mat = confusion_matrix(true_labels, pred_labels)
```

4- Visualisation de la matrice avec ses personnalisations comme le nom de la matrice la couleur et les dimensions.

```
# Visualize confusion matrix
plt.imshow(confusion_mat, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Confusion matrix')
plt.colorbar()
ticks = np.arange(5)
plt.xticks(ticks, ticks)
plt.yticks(ticks, ticks)
plt.ylabel('True labels')
plt.xlabel('Predicted labels')
plt.show()
```

Construction d'un régresser à une variable.

La régression est une technique qui modélise et à prédire une valeur continue en fonction d'une variable d'entrée : exemple le prix de l'essence dépend du prix du baril de pétrole donc on pourrait avoir la prédiction de prix aux station essence en fonction d'une variable le prix du baril de pétrole.

Etapes de construction d'une régression linéaire :

1- Utilisation d'un fichier source qui est 'data_singlevar_regr.txt'

```
# Input file containing data
input_file = 'data_singlevar_regr.txt'
```

2- Nous allons séparer par des virgules les documents puis nous nous allons créer tableau de x, y grâce a la technique du slising qui nous permets de manipuler des tableaux. X contient toutes les lignes sauf la dernière colonne et y contient seulement la dernière colonne de ma matrice.

```
# Read data
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

3- Séparer les partie test et la train avec slipt.

```
# Training data
X_train, y_train = X[:num_training], y[:num_training]

# Test data
X_test, y_test = X[num_training:], y[num_training:]
```

4- Création de l'objet regressor :

```
# Create linear regressor object
regressor = linear_model.LinearRegression()
```

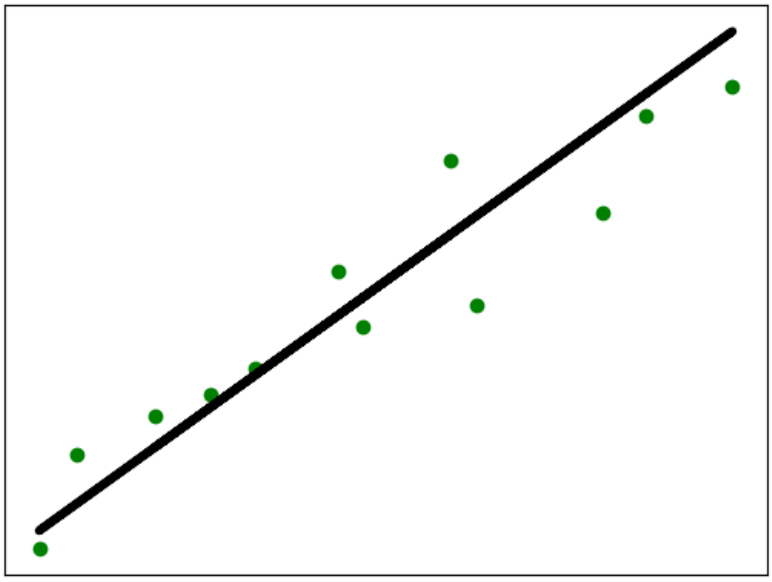
5- Entrainement du modèle avec fit et prédiction de x_test.

```
# Train the model using the training sets
regressor.fit(X_train, y_train)

# Predict the output
y_test_pred = regressor.predict(X_test)
```

6- Traçage de la fonction avec des point grâce à la fonction plt.scatter qui permet notamment de créer des nuages de point en python :

```
# Plot outputs
plt.scatter(X_test, y_test, color='green')
plt.plot(X_test, y_test_pred, color='black', linewidth=4)
plt.xticks(())
plt.yticks(())
plt.show()
```



Construction d'un régresser multivariable :

Les étapes de la régression multivariables se calquent sur la régression simple. La raison est que la régression multivariable va prédire en fonction de plusieurs variables.

Exemple : le prix de l'immobilier peut varier de la surface d'habitation et de sa localisation géographique.

Prenons l'exemple de la maison :

1- Importation des paquets

```
import numpy as np
from sklearn import datasets
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle
```
✓ 16.3s

2- Chargement des données plus mélange des données :

```
# chargement des données
data = datasets.load_boston()

# mélange des données
X, y = shuffle(data.data, data.target, random_state=7)
```

3- La séparation des données afin d'avoir un ratio 80/20

```
# On split les donnée de sorte à avoir 80/20
num_training = int(0.8 * len(X))
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]
```

4- Création du modelé SVR il utilise un noyau linéaire, une variable représentant la pénalité pour l'erreur d'apprentissage. Plus cette variable est élevée et plus elle dérivé dans un surapprentissage

5- Entrainement .fit()

```
# Entrainement du modèle
sv_regressor.fit(X_train, y_train)
```
✓ 16.3s

6- Problème lors de la visualisation des résultats de la régression multivariable il y a un message indiquant que le Dataset a été supprimer pour des questions éthiques raciaux au US.

Mettre une capture pour preuve

J’ai essayé des charger des données depuis kaggle mais j’ai souvent des erreur ne maitrisant pas bien kaggle et les datasets j’en suis resté à là.

FIN