Chapitre 4

Modélisation

Plan du chapitre

1. Introduction
2. Répartition du Dataset
3. Modélisation
4. Prédiction

01
Introduction

Introduction

- Une fois les données ont été rigoureusement prétraitées pour assurer leur qualité et leur cohérence, la phase suivante du processus englobe la répartition judicieuse du Dataset
- Cette étape implique la division méthodique des données en ensembles distincts, destinés respectivement à l'entraînement et aux tests
- Par la suite, vient l'étape de modélisation proprement dite
- Cela consiste à choisir un classificateur, basé sur une compréhension approfondie du problème à résoudre et qui convient le mieux au problème, en fonction de la nature des données et des objectifs

Introduction

- Le modèle est ensuite entraîné sur l'ensemble d'entraînement. Cela implique l'ajustement des paramètres du modèle pour minimiser la différence entre ses prédictions et les valeurs réelles dans l'ensemble d'entraînement

- Le modèle est ainsi prêt à être utilisé pour effectuer des prédictions sur de nouvelles données ou des données non vues précédemment

- La phase de prédiction est l'étape où le modèle utilise les connaissances acquises pendant l'entraînement pour générer des résultats sur de nouvelles observations

02

Répartition du Dataset

Répartition du Dataset

- Le découpage du jeu de données dans un projet de Machine Learning est une étape très importante qu'il ne faut pas négliger
- Faute de quoi, on risque de sur évaluer le modèle (surapprentissage : over-fitting)
- Cette étape est donc une étape préalable mais aussi une étape d'optimisation qu'il ne faut pas laisser de coté

Répartition du Dataset

Les données du Dataset doivent être divisées en des données d'entraînement et des données de test :

- **Entraînement (Train) :** C'est la partie du Dataset utilisée pour entraîner le modèle. Elle constitue la majorité du Dataset, généralement entre 70 % et 80 %
- **Test :** C'est la partie du Dataset utilisée pour évaluer la performance finale du modèle après l'entraînement. Elle représente entre 10 % à 20 % du Dataset

Remarques:

- Si le Dataset est déséquilibré (c'est-à-dire que certaines classes sont sous-représentées), il faut s'assurer de conserver la même répartition des classes dans chacune des partitions (train & test)
- Cela s'appelle la stratification et peut être important pour garantir une représentation adéquate de toutes les classes dans chaque ensemble

Répartition du Dataset

Pour le découpage des données du Dataset, nous avons 2 cas de figures :

- **Le Dataset est suffisamment grand :**
  Cela signifie qu'il contient un nombre important d'échantillons de données.

  ➔ Découper le Dataset en 2 sous-ensemble : Entrainement et Test

- **Le Dataset n'est pas suffisamment grand :**
  Le Dataset est limité en taille.

  ➔ Utiliser la validation croisée (cross-validation) pour maximiser l'utilisation des données disponibles

Entrainement/Test

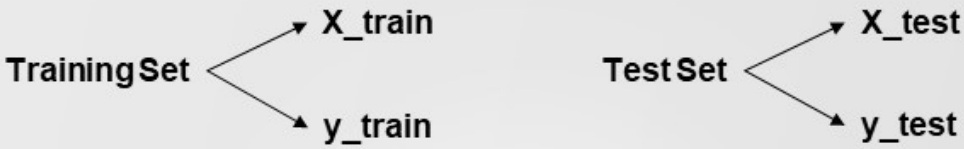
Découper le Dataset en 2 sous-ensemble :

1. **Training Set (~ 80%) :** sert à entraîner le modèle
2. **Test Set (~ 20%) :** sert à évaluer les performances réelles du modèle sur des données qu'il n'a jamais vues

Training Set :
==> utilisée pour l'apprentissage ;
==> fournie un modèle qui définie une correlation entre les variables indépendantes et dépendantes.

Test Set :
==> utilisée pour vérifier si le modèle maintient toujours ses corrélations ;
==> le modèle est testé sur une base de test, jamais vue en apprentissage.

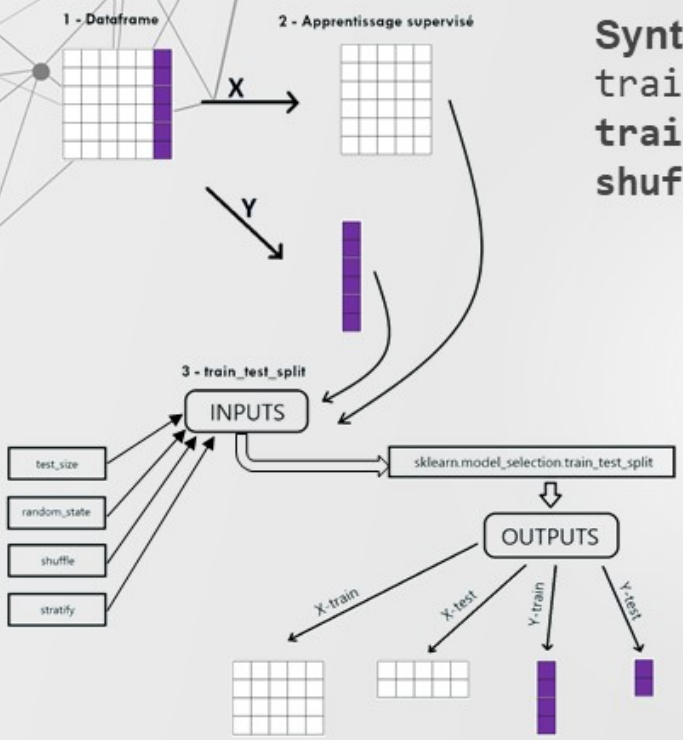
Entrainement/Test



➔ Le découpage se fait grâce à la fonction train_test_split()de la bibliothèque Scikit-Learn.

➔ Cette fonction renvoit les données (Observation=X, résultat=y) découpées en jeux d'entraînement et test. en ajustant les paramètres random_state et shuffle on peut même régler le degré de choix (aléatoire) de distribution dans l'un ou l'autre jeu de données.

Entrainement/Test



Syntaxe:

```
train_test_split(*arrays, test_size=None,
train_size=None, random_state=None,
shuffle=True, stratify=None)
```

- **Apprentissage supervisé :**

  Ces arrays sont l'input array **X**, constitué des variables explicatives en colonnes, et l'output array **y**, constitué de la variable cible (c'est-à-dire les labels)

- **Apprentissage non-supervisé :**

  L'unique array en argument est l'input array **X**, constitué des variables explicatives en colonnes

Entrainement/Test

La taille du test set (test_size) et la taille du training set (train_size).

- La taille de chaque ensemble est soit un nombre décimal compris entre 0 et 1 représentant une proportion du jeu de données, soit un nombre entier représentant un nombre d'exemples du jeu de données

Remarque :
Il est suffisant de définir un seul de ces arguments, le deuxième lui étant complémentaire.

Entrainement/Test

random state

- Le random state est un nombre qui contrôle la façon dont le générateur pseudo-aléatoire divise les données.

Remarque :
Choisir un nombre entier comme random state permet de séparer les données de la même manière à chaque appel de la fonction.

➔ Cela rend donc le code reproductible.

Entrainement/Test

shuffle

- Le shuffle est un booléen qui choisit si les données doivent être mélangées ou non avant d'être séparées
- Dans le cas où elles ne sont pas mélangées, les données sont donc séparées en fonction de l'ordre où elles étaient initialement

Remarque :
La valeur par défaut est True

Entrainement/Test

stratify

- Le paramètre stratify choisit si les données sont séparées de façon à garder les mêmes proportions d'observations dans chaque classe dans les ensembles train et test que dans le Dataset initial

Remarques :
Ce paramètre est particulièrement utile face à un Dataset non équilibrées
La valeur par défaut est None

Entrainement/Test

```
# Import de la fonction train_test_split
from sklearn.model_selection import train_test_split

# X représente vos caractéristiques et y les étiquettes

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
```


Entrainement/Test

Exemple

X_train

Pays			Age	Salaire
0	1	0	40	63777.778
1	0	0	37	67000
0	0	1	27	48000
0	0	1	38.7778	52000
1	0	0	48	79000
0	0	1	38	61000
1	0	0	44	72000
1	0	0	35	58000

Y_train

Achat
1
1
1
0
1
0
0
1

X_test

Pays			Age	Salaire
0	1	0	30	54000
0	1	0	50	83000

Y_test

Achat
0
0

Entrainement/Test

Avantages :

- **Simplicité :** Facile à mettre en œuvre et à comprendre.
- **Gain de temps :** Plus rapide que la validation croisée car le modèle n'est entraîné qu'une seule fois

Inconvénient :

- **Sensibilité :** Les performances du modèle peuvent dépendre fortement de la manière dont les données sont réparties entre les ensembles d'entraînement et de test
- **Biais dans l'évaluation :** Ne convient pas à un ensemble de données non équilibré

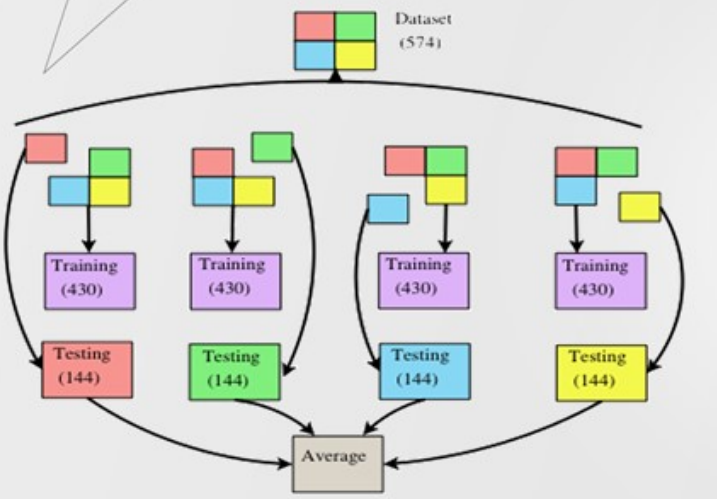
Validation croisée

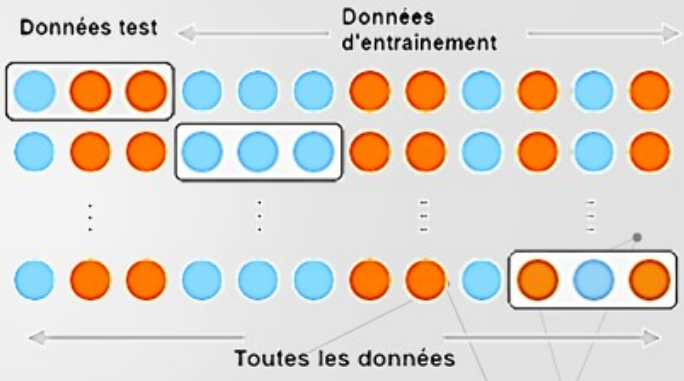
Pour utiliser la validation croisée (cross-validation ou bien K-Folds):

- On commence par une séparation de l'ensemble de données de façon aléatoire en K sous ensembles appelés Folds
- L'ajustement du modèle est réalisé avec les Folds K-1, puis le modèle est validé en utilisant le K-Fold restant. On répète le même processus jusqu'à ce que tous les K-Fold servent dans l'ensemble d'entraînement
- On calcule ensuite la moyenne des scores enregistrés qui représente la métrique de performance du modèle

Validation croisée

Exemple: 4-Fold





Validation croisée

Avantages :

- **Utilisation optimale des données :** Permet d'utiliser l'ensemble des données pour l'entraînement et la test
- **Évaluation robuste :** Fournit une évaluation plus robuste des performances du modèle
- **Réduction de la dépendance à la répartition des données :** Réduit le risque de biais d'évaluation dû à une répartition particulière des données

Validation croisée

Inconvénients :

- **Coût computationnel :** Plus gourmand en ressources computationnelles car le modèle est entraîné et évalué plusieurs fois. Il est inadéquat pour les grands ensembles de données

- **Complexité :** Plus difficile à mettre en œuvre et à interpréter, surtout pour des ensembles de données volumineux

03

Modélisation

Modélisation

- Lors de cette étape on doit choisir le ou les algorithmes de Machine Learning qui convient le mieux au problème
- Selon du type du problème traité (classification, régression, etc.), on a le choix entre plusieurs algorithmes pour la formation du modèle (modélisation) tels que la régression linéaire, les arbres de décision ou les réseaux de neurones
- Au cours du processus de formation, les modèles apprennent des données et établissent des relations entre les caractéristiques d'entrée et la variable cible

Modélisation

- La difficulté n'est pas dans ce choix de l'algorithme mais plutôt dans l'ajustement des hyperparamètres afin d'obtenir un modèle performant
- De nombreux modèles comportent des hyperparamètres, qui sont des paramètres qui régissent le comportement et les performances du modèle mais qui ne sont pas tirés des données
- Ces hyperparamètres sont raffinées pour optimiser les performances du modèle et obtenir les meilleurs résultats possibles
- Des techniques telles que la recherche par grille (Grid Search), la recherche aléatoire (Random search) ou l'optimisation bayésienne (Bayes Search) sont couramment utilisées pour identifier la combinaison optimale de valeurs d'hyperparamètres.

Modélisation

Exemple 1 : Modèle de régression logistique

```
# Import des bibliothèques
from sklearn.linear_model import LogisticRegression

# Initialisation du modèle
model = LogisticRegression()

# Entraînement du modèle sur l'ensemble d'entraînement
model.fit(X_train, y_train)
```


Modélisation

Exemple 2 : Modèle de réseau de neurones

```
# Import des bibliothèques
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Initialisation du modèle
model = Sequential()
model.add(Dense(units=64, activation='relu',
input_dim=input_size))
model.add(Dense(units=1, activation='sigmoid'))
```

Modélisation

Exemple 2 : Modèle de réseau de neurones (suite)

```
# Compilation du modèle
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy’])

# Entraînement du modèle sur l'ensemble d'entraînement
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))
```


04

Prédiction

Prédiction

- Après avoir achevé les étapes de modélisation et d'entraînement d'un modèle en Machine Learning, la phase suivante est celle de la prédiction

- Cette étape consiste à utiliser le modèle entraîné pour faire des prédictions sur de nouvelles données ou des données non vues auparavant

- Lors de la prédiction, les caractéristiques des nouvelles instances sont alimentées dans le modèle, qui applique les relations et les patterns appris pendant la phase d'entraînement pour générer des prédictions

Prédiction

```
# Prédiction
y_pred = model.predict(X_test)
```