MODELISATION IA & ENERGIE

Travaux Pratiques

Première séance : 12/09/2023

Sommaire

Objectif : L'objectif de ce TP est de vous faire découvrir l'application de l'intelligence artificielle dans le domaine de la prédiction de l'énergie solaire photovoltaïque.

N.B : Lors de ce TP, vous serez amené à utiliser **Python 3** pour réaliser vos modèles prédictifs.

I. Introduction

1. Machine Learning : Définitions et Concepts

Le Machine Learning (ML) est une branche de l'intelligence artificielle qui vise à donner aux ordinateurs la capacité d'apprendre. L'ordinateur n'est pas intelligent, il effectue simplement des tâches. Nous lui décrivons sous forme de programme ce qu'il faut faire et comment le faire. C'est ce qu'on appelle la programmation.

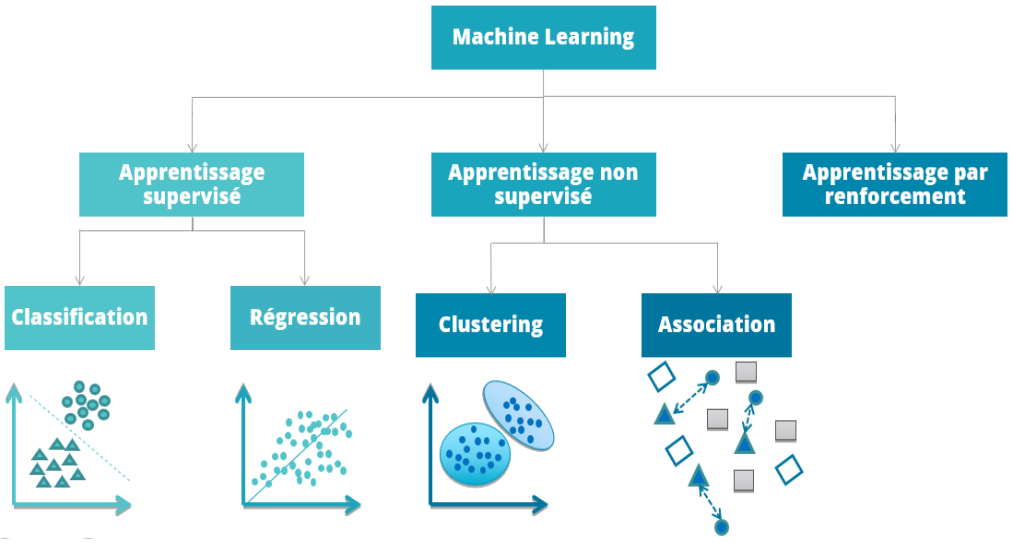
Le Machine Learning traite des sujets complexes où la programmation traditionnelle trouve ses limites. Construire un programme de conduite automobile sera très compliqué, voire impossible. Cela est dû au nombre infini de cas qui peuvent être traités.

Le Machine Learning gère ce problème différemment. Au lieu de décrire ce qu'il faut faire, le programme apprendra lui-même à conduire en « observant » l'expérience.

Machine Learning → Donner la possibilité à l'ordinateur d'apprendre sans être programmé.

On distingue trois principaux types de problèmes en ML :

- **Apprentissage Supervisé**
- Apprentissage Non supervisé
- Apprentissage par Renforcement



2. Apprentissage supervisé

C'est le type le plus récurrent. Il s'agit de fournir aux algorithmes d'apprentissages un jeu de données d'apprentissage *(Training Set)* sous forme de **(X, Y)** avec X les variables prédictives, et Y le résultat de l'observation. En se basant sur le *Training Set*, l'algorithme va trouver **une fonction mathématique** qui permet de transformer (au mieux) X vers Y. En d'autres termes, on l'algorithme va trouver une fonction tel que : F(X) ≃Y.

3. Apprentissage non supervisé

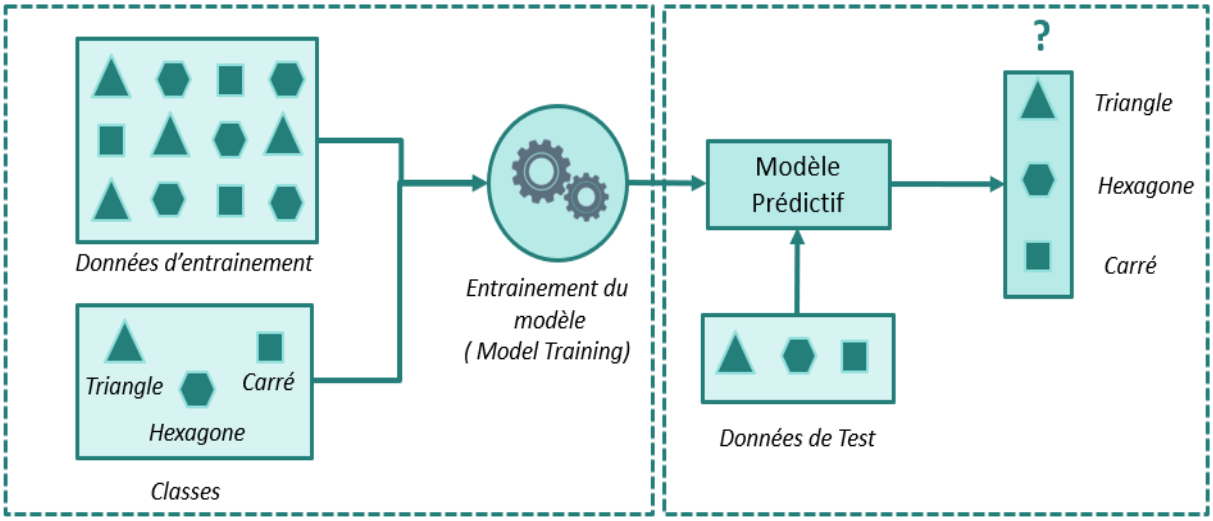
Dans ce type, on va donner à l'algorithme des données, éventuellement non structurées. Et on le laisse trouver une sorte de structure dans nos données. Cela peut être des regroupements de données (Clustering).

Dans ce TP, nous allons nous focaliser sur l'apprentissage supervisé. L'apprentissage supervisé peut être décliné en deux sous catégories :

- La classification
- La régression

4. Classification

Quand la variable à prédire prend une valeur discrète, on parle d'un problème de **classification**.



5. Régression

L'algorithme de régression permet de trouver un modèle (une fonction mathématique) en fonction des données d'entrainement. Le modèle calculé permettra de donner une estimation sur une nouvelle donnée non encore vue par l'algorithme (qui ne faisait pas partie des données d'entrainement).

Les algorithmes de régression peuvent prendre plusieurs formes en fonction du modèle qu'on souhaite construire. La régression linéaire est le modèle le plus simple : Il consiste à trouver la meilleure droite qui s'approche le plus des données d'apprentissage.

II. Problématique

1. Atouts de l'IA pour les défis énergétiques

Le premier atout de l'apprentissage automatique (*machine Learning*) est qu'il permet, à partir d'un comportement passé, de prédire ce comportement dans le futur. L'exemple sur lequel nous allons nous focaliser dans ce TP est celui des **énergies renouvelables**, particulièrement **l'énergie solaire photovoltaïque**.

L'énergie est essentielle dans notre société, car elle est le moteur de presque tous les secteurs. **Les combustibles fossiles** sont historiquement la source d'énergie la plus importante, représentant 80,2 % en 2019. Ces types d'énergie présentent différents problèmes, l'un d'entre eux étant leur **rareté**, puisqu'il s'agit de ressources **limitées** qui ont été exploitées pendant longtemps. Un autre problème critique est la **pollution** causée par la combustion et l'extraction de ces combustibles, qui est dangereuse pour les personnes et l'environnement.

Pour résoudre ces problèmes, d'autres sources d'énergie peuvent être utilisées. Ces énergies alternatives, les **énergies renouvelables,** présentent deux avantages principaux. Tout d'abord, elles reposent sur des ressources **illimitées** qui ne s'épuiseront pas, même en cas d'exploitation intensive. Leur exploitation est également **non polluante**. Les investissements dans ces énergies ont augmenté ces dernières années.

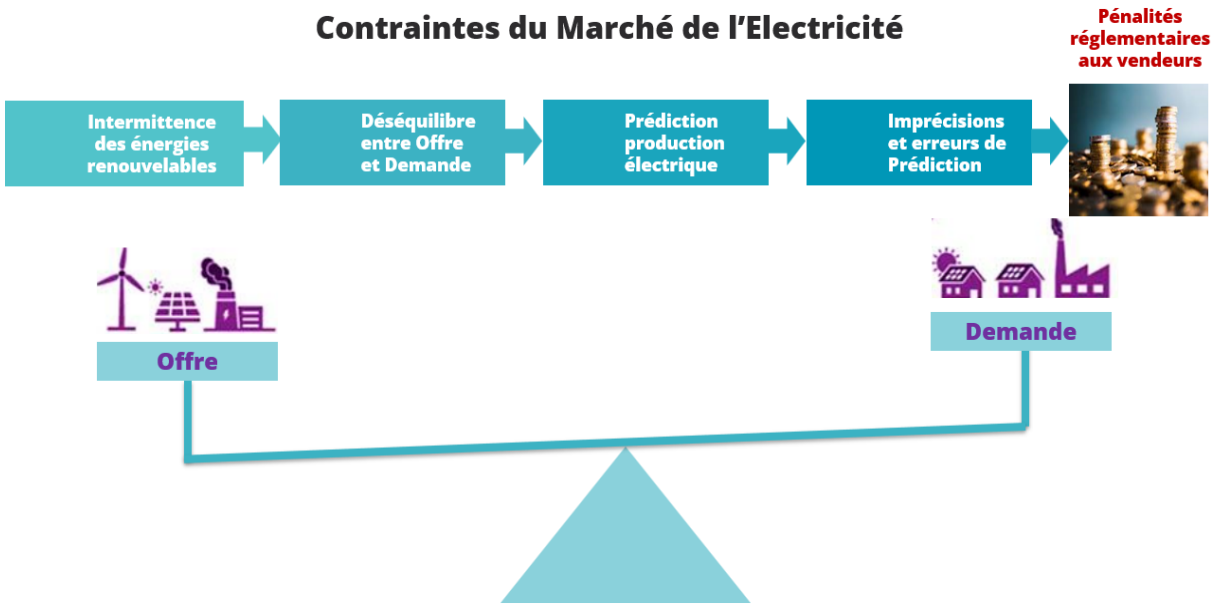
L'une des **énergies vertes** les plus importantes est **l'énergie solaire**. Cette énergie se compose de **l'énergie solaire**, thermique et **photovoltaïque (PV).** Cette dernière s'est avérée plus utile et plus rentable pour la production industrielle et a connu une croissance constante ces dernières années.

L'énergie photovoltaïque est produite par des modules photovoltaïques. Chaque module est composé de différentes sous-unités, appelées cellules solaires, qui absorbent l'énergie émise par le soleil. Les panneaux photovoltaïques sont généralement connectés en série les uns aux autres, c'est ce que l'on appelle une matrice photovoltaïque. Chaque panneau photovoltaïque est connecté à un onduleur qui contrôle la production et vérifie les performances du panneau. **Les installations photovoltaïques** font face à un grand nombre de problèmes. Les plus importants sont liés à l'optimisation de la production d'énergie, puisque c'est l'objectif final d'une installation énergétique.

2. Prédire pour anticiper et s'adapter

Il existe plusieurs variantes du problème de prédiction qui se pose dans le domaine du **photovoltaïque** : les prédictions météorologiques, les prédictions d'irradiation solaire et les prédictions de production d'énergie, qui consistent à estimer la production d'énergie du système. Cette dernière est très importante pour optimiser la gestion en temps réel des systèmes qui utilisent ce type d'énergie (villes intelligentes, villages, etc.).

Ce problème est prioritaire pour les compagnies d'électricité, car elles souhaitent disposer d'un système plus robuste et plus fiable pour prévoir les changements dans les charges et les demandes d'énergie.



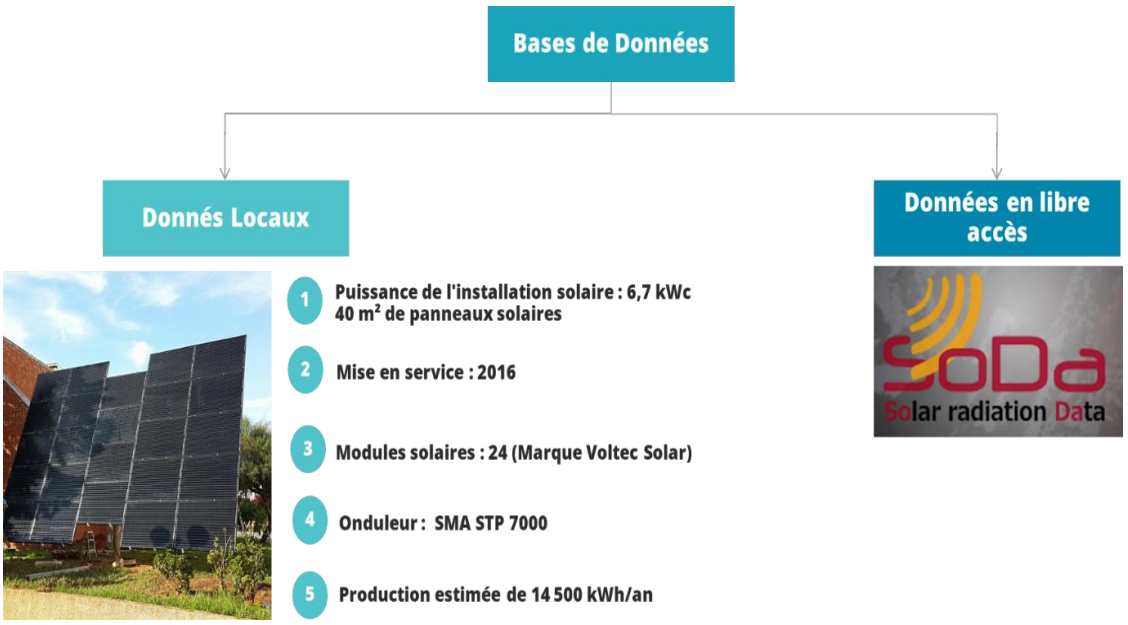
Un autre aspect important est la durée de la prédiction.

- **Les prédictions à court terme** portent généralement sur une période allant d'une heure à une semaine et sont utilisées pour programmer le transfert d'énergie, la répartition économique de la charge et la réponse à la demande.
- **Les prédictions à moyen terme** sont généralement envisagées entre 1 mois et 1 an à l'avance, généralement pour planifier les plans d'énergie à venir et pour montrer la dynamique du système dans cet intervalle.
- **Les prédictions à long terme** sont envisagées entre 1 an et 10 ans. Sa fonction est de planifier la production de la centrale électrique de manière à satisfaire les besoins futurs et la rentabilité.

III. Base de données

1. Définition des données d'entrées et de sortie

Le problème que nous allons résoudre est une tâche de régression dans le but de prédire la puissance de sortie d'une installation photovoltaïque.



Donnée de sortie (Variable cible à prédire)

La variable cible considérée est la puissance solaire photovoltaïque horaire (PAC). Elle est dérivée d'une installation photovoltaïque d'une capacité totale de 6 KW en fonctionnement depuis 2016 à l'Ecole d'Ingénieurs de Mohammadia à Rabat.

Données d'entrée

Dans le cadre de ce TP, nous allons utiliser des données d'entrée simples. Celles-ci proviennent de SoDa, une source de données gratuite qui offre des services d'énergie solaire et météorologiques. Ces données sont présentées comme suit :

Paramètre	Description	Unité	Symbole
Heure	Heure de la journée	-	Hour
Température des modules	La température du module solaire PV	°C	Tm
Température ambiante	Température à 2 m au-dessus du sol	°C	Tamb
Rayonnement au sommet de l'atmosphère	Rayonnement à la hauteur de l'atmosphère	Wh/m^2	TOA
Le rayonnement horizontal global	Rayonnement global au niveau du terrain	Wh/m^2	GHI

2. Importation des bibliothèques et chargement des données

Nous allons utiliser la bibliothèque **pandas** afin d'importer les données. Cette bibliothèque est comprise dans **Anaconda**. Python propose via sa librairie Pandas des classes et fonctions pour lire divers formats de fichiers. La fonction **read_table()** est conçue pour lire le type de données textuelles où chaque colonne est séparée par une (ou plusieurs) colonnes d'espace. Ci-dessous donc le code pour importer les données :

```
#importation des données
import pandas
D = pandas.read_table("data.txt",sep=",",header=0)
#liste des variables
print(D.info())
```

Vous constaterez que vous n'aurez pas de données manquantes et que vous aurez 5 variables explicatives et une cible.

3. Découpage des données

Comme expliqué précédemment, nous allons prendre comme variable dépendante **Y**, la puissance de sortie photovoltaïque **(PAC)** et comme variables indépendantes **X** toutes les autres variables. Il faudra donc séparer les données d'entrée et la cible dans deux variables Python.

```
Y=D.PAC
X=D.drop('PAC',axis=1)
print(Y)
```

Nous générons maintenant un premier découpage entre **données d'apprentissage** et **données de test**.

Ce découpage est aléatoire et s'effectue à l'aide de la fonction **train_test_split** de la **bibliothèque scikit-learn**. Le paramètre **test_size** permet de spécifier le pourcentage du jeu de données qui sera contenu dans le jeu de test.

```
import sklearn
from sklearn import model_selection
XTrain,XTest,yTrain,yTest = model_selection.train_test_split(X,Y,test_size=2348,random_state=100)
```

IV. Construction des modèles prédictifs : Apprentissage supervisé

1. Régression Linéaire Multiple

Généralités :

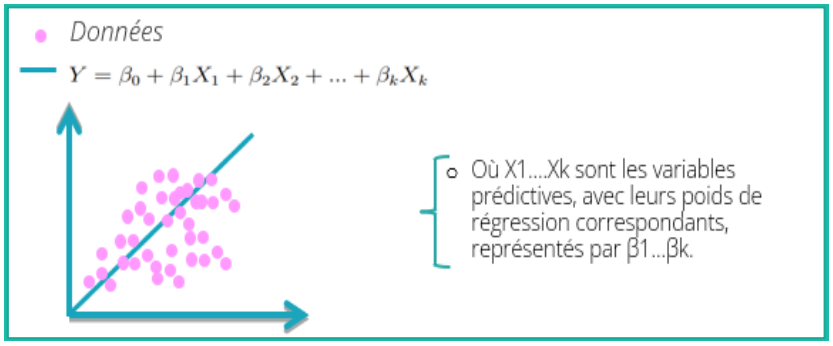
Pour commencer ce premier TP, nous examinerons l'algorithme de régression linéaire multiple. Celui-ci est représenté comme suit :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

Où $X_1 \ldots X_K$ sont les variables prédictives, avec leurs poids de régression correspondants, représentés par $\beta_1 \ldots \beta_k$. Les coefficients de régression $\beta_1 \ldots \beta_k$ dans l'équation sont inconnus et doivent être calculés.

Nous pouvons générer des prédictions une fois que nous avons utilisé nos données d'apprentissage pour obtenir des estimations pour les coefficients du modèle $\hat{\beta}_1 \ldots \hat{\beta}_k$.

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \cdots + \hat{\beta}_n X_n$$



Implémentation en Python :

Pour générer votre premier modèle de régression linéaire, vous allez utiliser la fonction **LinearRegression** de **scikit-learn**. La bibliothèque **scikit-learn** va chercher la fonction affine $f: X \to Y = \beta_0 + \beta_1 X$ en déterminant les paramètres β_0 et β_1 à l'aide de la méthode des moindres carrés.

Les résultats sont évalués ici à travers le coefficient de détermination, qui est le rapport entre la variance expliquée par le modèle et la variance totale (de la variable expliquée).

Commençons par instancier un modèle de régression linéaire. Nous créons un objet **model** qui est le conteneur de notre modèle de régression multiple. Une fois l'objet créé en utilisant la bibliothèque scikit-learn, nous ajustons le modèle **(fit)** en utilisant nos données.

Tous les modèles implémentés dans scikit-learn suivent la même interface et présentent au moins les trois méthodes suivantes :

- **.fit()** : permet d'apprendre le modèle en estimant ses paramètres à partir d'un jeu de données,
- **.predict()** : permet d'appliquer le modèle à de nouvelles données,
- **.score():** permet d'évaluer le modèle selon un critère prédéfini (taux de bonne classification, coefficient de détermination, etc.) sur un jeu de test.

Par exemple, dans notre cas, la méthode **.fit(X, Y)** permet de déterminer les paramètres de la régression linéaire à partir de nos observations X et de notre vérité terrain Y**.**

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# optimisation du modèle: détermination des paramètres de la régression linéaire par la méthode des moindres carré
model.fit(XTrain, yTrain)
```

Une fois le modèle ajusté, nous affichons, la constante et les coefficients du modèle.

```
print(model.intercept_)
print(model.coef_)
```

Nous pouvons évaluer à quel point le modèle « colle » aux données à l'aide du coefficient de détermination **R^2** calculé sur les exemples du jeu d'apprentissage.

Le **R^2** est une mesure statistique dans un modèle de régression qui détermine la proportion de la variance de la variable dépendante qui peut être expliquée par la variable indépendante.

En d'autres termes, le **R^2** indique dans quelle mesure les données correspondent au modèle de régression (qualité de l'ajustement). Le **R^2** peut prendre n'importe quelle valeur entre 0 et 1 et se calcule comme suit :

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Où $\hat{y}$ représente la prédiction de la valeur de y et $\bar{y}$ représente la valeur moyenne de y.

```
import sklearn.metrics as metrics
rsq=metrics.r2_score(yTrain,model.predict(XTrain))
print(f"Coefficient de détermination R² en apprentissage (train) : {rsq:.2f}")
```

Question 1 Calculer le coefficient de détermination R^2 sur le jeu de test. Que constatez-vous ?

Vous pouvez ensuite, à partir des coefficients qui ont été estimés, tracer votre modèle de régression. Vous obtiendrez deux graphiques représentant : les valeurs de Y en fonction des valeurs prédites avec le modèle de régression linéaire et les valeurs de Y en fonction des résidus.

```
import matplotlib.pyplot as plt
plt.plot(yTrain, model.predict(XTrain),'.')
plt.show()

plt.plot(yTrain, yTrain-model.predict(XTrain),'.')
plt.show()
```

Question 2 Quelle est votre interprétation des deux graphiques obtenus ?

Question 3 Calculez l'erreur quadratique moyenne du modèle sur les données d'apprentissage et ensuite sur les données de test. Vous pouvez l'implémenter vous-même à l'aide de NumPy ou bien consulter la documentation du module sklearn.metrics.

2. Forêts Aléatoires

Généralités :

La régression linéaire est un outil pratique mais quelque peu limité : les relations entre les variables explicatives et la variable à prédire sont rarement linéaires en pratique. D'où l'utilisation d'autres algorithmes tels que les forêts aléatoires.

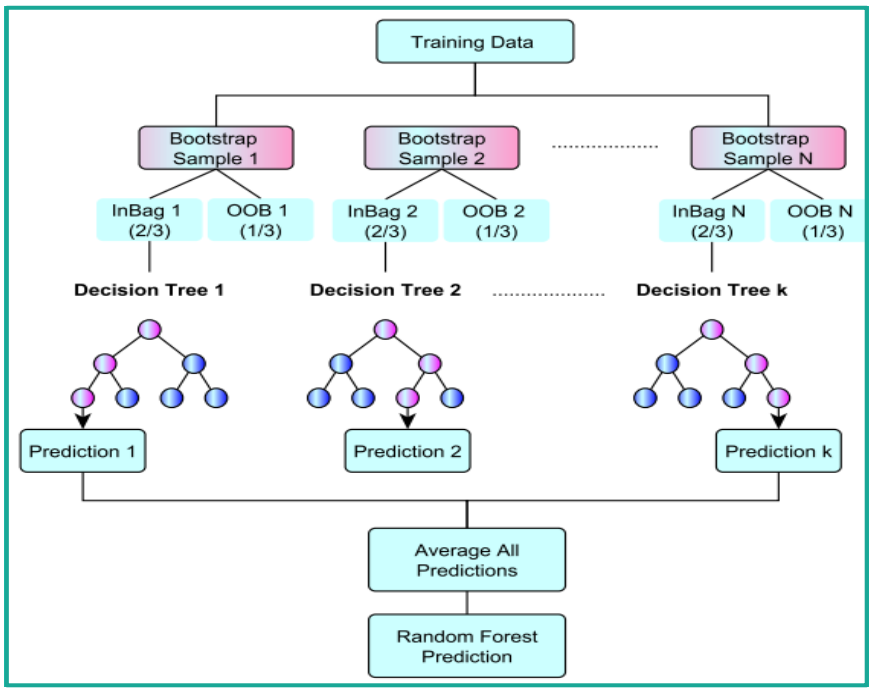
Une forêt aléatoire consiste en une collection d'arbres de décision. L'algorithme sélectionne aléatoirement des échantillons des données de formation et des variables et crée un arbre de décision à partir de chaque échantillon. La prédiction s'effectue en faisant la moyenne ou en rapprochant les prédictions de chaque arbre. Ce modèle utilise deux concepts clés qui lui donnent le nom aléatoire :

- Échantillonnage aléatoire des points de données d'entraînement lors de la création d'arbres
- Sous-ensembles aléatoires d'entités pris en compte lors de la division des nœuds

Lors de l'entraînement, chaque arbre d'une forêt aléatoire apprend à partir d'un échantillon aléatoire des points de données.

Les échantillons sont dessinés avec remplacement, appelé **bootstrap**, ce qui signifie que certains échantillons seront utilisés plusieurs fois dans un même arbre. L'idée est qu'en entraînant chaque arbre sur différents échantillons, bien que chaque arbre puisse avoir une variance élevée par rapport à un ensemble particulier de données d'apprentissage, dans l'ensemble, la forêt entière aura une variance plus faible, mais pas au prix d'une augmentation du biais.

Cette procédure de formation de chaque apprenant individuel sur différents sous-ensembles **bootstrap** de données, puis de calcul de la moyenne des prédictions est connue sous le nom de **bagging**, abréviation de **bootstrap agrégation**.



Implémentation en Python :

Nous allons utiliser la fonction **RandomForestRegressor** de la bibliothèque **scikit-learn** pour construire notre forêt aléatoire. Nous allons définir ce qu'on appelle les hyperparamètres du modèle. Il s'agit de paramètres tels que le nombre d'arbres dans la forêt qui doivent être définis en amont de l'apprentissage. Les hyperparamètres les plus importants sont :

- **n_estimators** : Nombre d'arbres dans la forêt
- **random_state** : divise les ensembles de données de test et de formation de manière aléatoire.
- **bootstrap** : paramètre pour utiliser du bootstrap, si il est à False, le même échantillon est pris pour chaque arbre

Après avoir lu les données, nous pouvons instancier et entraîner une forêt aléatoire comme suit :

```
from sklearn.ensemble import RandomForestRegressor # for building the model
# Create the model with 10 trees
model = RandomForestRegressor(n_estimators = 10, random_state = 10, bootstrap = True)
```

Nous avons donc construit un modèle forêt aléatoire, nous devons maintenant effectuer l'apprentissage

```
# Fit on training data
model.fit(XTrain, yTrain)
```

Nous évaluerons maintenant le coefficient de détermination $\mathbf{R^2}$ durant la phase d'apprentissage.

```
import sklearn.metrics as metrics
rsq=metrics.r2_score(yTrain,model.predict(XTrain))
print(rsq)
```

Le modèle est maintenant prêt à faire des prédictions sur les données de test.

Question 4 Calculer le coefficient de détermination R^2 sur le jeu de test. Comparez-le au score obtenu précédemment en utilisant la régression linéaire multiple. Que constatez-vous ?

Vous pouvez visualiser la courbe représentant les valeurs prédites vs les valeurs réels de puissance photovoltaïque en appliquant le code ci-dessous :

```
import seaborn as sns
width = 12
height = 10
plt.figure(figsize=(width, height))
ax1 = sns.distplot(yTest, hist=False, color="r", label="Actual Value")
sns.distplot(y_pred, hist=False, color="b", label="Fitted Values" , ax=ax1)
plt.title('Actual vs Fitted Values for PV Power')
plt.xlabel('PV Power Length')
plt.show()
plt.close()
```

Vous pouvez également afficher l'importance des variables de notre modèle de forêt aléatoire comme suit :

```
pandas.DataFrame(model.feature_importances_,
                 index = XTrain.columns,
                 columns = ["importance"]).sort_values(
    "importance",
    ascending = False)
```

Question 5 Quelle est la variable la plus importante (au sens de son importance dans les arbres générés) ainsi que celle la moins importante ?

Question 6 Le paramètre n_estimators définit le nombre d'arbres dans la forêt. Essayez plusieurs valeurs pour voir l'impact de l'utilisation d'un nombre important ou faible de nombre d'arbres. Que constatez-vous ?

Une autre étape consiste à optimiser la forêt aléatoire que nous pouvons faire. **GridSearchCV** de **Scikit-Learn** est une technique permettant de rechercher les meilleures valeurs de paramètre à partir de l'ensemble donné de la grille de paramètres. Celle-ci est implémentée comme suit :

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
parameters = {
    'n_estimators': [10, 50, 100, 200, 300]
}
regr = RandomForestRegressor(random_state=10,bootstrap = True)

model = GridSearchCV(regr, parameters)
model.fit(XTrain, yTrain)
```

Question 7 **Après application de la fonction GridSearchCV, calculez les coefficients de détermination R^2 sur les données d'entrainement et de test. Que constatez-vous ?**

V. Résumé

Dans cette première séance de TP, vous vous êtes initiés à l'application de l'IA au domaine énergétique. Vous avez implémenté en Python deux algorithmes basiques de la prédiction dans le domaine énergétique à savoir la régression linéaire multiple et les forêts aléatoires. Vous avez appris à créer des modèles prédictifs et évaluer les résultats de prédiction.

Dans la prochaine séance de TP, vous allez découvrir deux autres méthodes de prédiction populaires dans le domaine des énergies renouvelables.