Bibliothèques Python pour l'IA

Bassam Kurdy Ph.D
<bassam.kurdy@apinum.fr>

Bibliothèques Python pour l'IA

- **Scikit Learn :**
  - Scikit Learn est une robuste bibliothèque d'apprentissage automatique pour Python.
  - Scikit Learn fournit une gamme d'algorithmes d'apprentissage supervisés et non supervisés via une interface cohérente en Python.
  - On peut faire de la classification, de la régression, choisir les hyperparamètres et du pre-processing.
  - Attention : Scikit Learn se concentre sur la modélisation des données et non pas la manipulation des données. (avec Numpy et Pandas)

Bibliothèques Python pour l'IA

- **TensorFlow :**
  - TensorFlow est une bibliothèque d'intelligence artificielle qui aide les développeurs à créer des réseaux neuronaux à grande échelle avec de nombreuses couches en utilisant des graphiques de flux de données.
  - TensorFlow est très efficace lorsqu'il s'agit de la classification, la perception, la compréhension, la découverte, la prévision et le création de données.
  - On peut faire de la reconnaissance de la voix et du son, l'analyse de sentiment, la reconnaissance faciale, la détection vidéo, etc.

Bibliothèques Python pour l'IA

- **Keras :**
  - Keras est l'API de haut niveau de Tensorflow pour le développement et la formation du code Deep Neural Network. Il s'agit d'une bibliothèque réseau neuronal open-source en Python. Avec Keras, la modélisation statistique, le travail avec les images et le texte est beaucoup plus facile surtout avec le codage simplifié pour l'apprentissage en profondeur.
  - Keras est une bibliothèque réseau neuronal Python tandis que Tensorflow est une bibliothèque open-source pour diverses tâches d'apprentissage machine.

Bibliothèques Python pour l'IA

- **PyTorch :**
  - Bibliothèque open source de ML qui s'appuie sur Torch développée par Facebook.
  - PyTorch permet d'effectuer les calculs tensoriels nécessaires notamment pour le DL.
  - PyTorch permet de manipuler les tableaux multidimensionnels et de calculer les gradients pour appliquer facilement des algorithmes d'optimisation par descente de gradient.

Veille individuelle 30min

Scikit-learn

- Comment les données sont représentées ? (Matrice de variables explicatives, vecteur de variable expliquée)
- Estimator API (construire un modèle, entraîner un modèle, prédire)
- Sauvegarder et charger un modèle

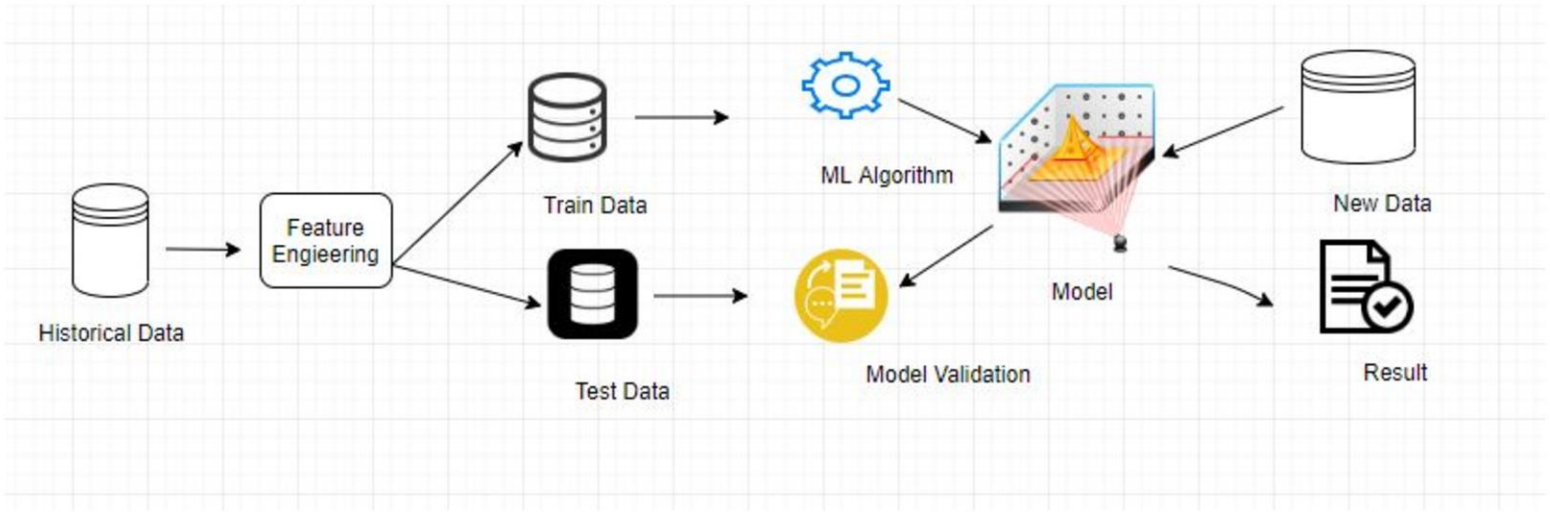
Ressources :

- Introducing Scikit-Learn | Python Data Science Handbook (jakevdp.github.io)
- Travaux pratiques - Introduction à Scikit-learn — Cnam - UE RCP208
- An introduction to machine learning with scikit-learn — scikit-learn 1.3.2 documentation
- scikit-learn: machine learning in Python — scikit-learn 1.3.2 documentation

Premier modèle de ML

Etapes clés

7 étapes vers l'apprentissage automatique
Source : 7 Steps to Machine Learning: How to Prepare for an Automated Future



Etapes de création d'un modèle ML
Source : Machine Learning Workflow

Etape de construction d'un modèle

1. Importer les données
2. Séparation des données en sous ensemble d'entraînement et un sous ensemble de test.
3. construction du modèle
4. Entrainement du modèle avec le sous ensemble d'entraînement
5. Prédictions
6. Evaluation du modèle

1 -> dataloader - dataframe

2-> train_test_split

3-> estimator (hyperparamètres)

4->estimator.fit(X_train,y_train)

 # modèle entraîné

5-> predict(X_test)

6-> score(X_test,y_test)

Premier exemple

```
# Regression Linéaire Simple

# Importer les librairies
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importer le dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Diviser le dataset entre le Training set et le Test set
# on utilise la fonction train_test_split de sklearn.model_selection
# généralement, on utilise le ratio 30% pour le test set et 70% pour le train set
# on reviendra sur les paramètres ultérieurement !
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1.0/3, random_state = 0)

# Construction du modèle
from sklearn.linear_model import LinearRegression
# Notre modèle est dans la variable regressor et c'est une instanciation de la classe LinearRegression
regressor = LinearRegression()

# Entrainer le modèle
# pour entrainer le modèle, on  utilise la variables explicatives X_train et la variable expliquée y_train
# L'objectif c'est que le modèle apprenne des ces données pour trouver la droite qui donne le meilleur ajustement
#de y_train en fonction de X_train
regressor.fit(X_train, y_train)

# Faire de nouvelles prédictions
# avec un vecteur X_test ! à ce stade, on test notre modèle sur des données connu pour pouvoir évaluer ses performances.
# on calcule alors des prédictions avec notre modèles qu'on peut ensuite comparer avec les valeurs y_test connues
y_pred = regressor.predict(X_test)
# prédiction du salaire de quelqu'un avec 15ans d'expérience
y = regressor.predict(np.array([[15]]))

# Visualiser les résultats
plt.scatter(X_test, y_test, color = 'red')
plt.scatter(X_train, y_train, color = 'blue')
plt.title('Salaire vs Experience')
plt.xlabel('Experience')
plt.ylabel('Salaire')
plt.show()
```

Pratiquer

créer des modèles de classification Naïve Bayesienne avec scikit-learn

- 1.9. Naive Bayes — scikit-learn 1.3.2 documentation
- Tuto Python & Scikit-learn: la classification Naïve Bayésienne - Tutoriel Python (cours-gratuit.com)