École Centrale Casablanca

Rapport

TP IA

Élèves :
Achraf El Faiz
Hamza El Othmani

Enseignant :
Redha Moulla

Table des matières

1 Introduction

Le domaine du Machine Learning (ML), ou apprentissage automatique en français, constitue une discipline clé au sein de l'intelligence artificielle (IA) qui a considérablement évolué ces dernières décennies. À l'intersection des domaines de la statistique, de l'informatique et de l'apprentissage automatique, le ML offre des approches novatrices pour permettre aux systèmes informatiques d'apprendre automatiquement des données et d'améliorer leurs performances sans une programmation explicite.

Après l'exécution d'un ensemble de travaux pratiques et d'un mini-projet, en exploitant les concepts fondamentaux du machine learning ainsi que les types d'algorithmes couramment utilisés, nous avons élaboré ce rapport pour interpréter les résultats obtenus lors des trois travaux pratiques .

2 TP 1

Dans le but de prédire la puissance de sortie d'une installation photovoltaïque , on a utilisé comme model la regression linéaire multiple . Ainsi , on a répondu aux questions de ce TP , en interprétant nos résultats .

2.1 le coefficient de détermination R^2 sur le jeu de test

En suivant les différentes étapes de modélisations (Part , proprety , assembly , Mesh ...) , et en spécifiant le nombre des élements sur 10 , nous avons obtenus les résultats suivants :

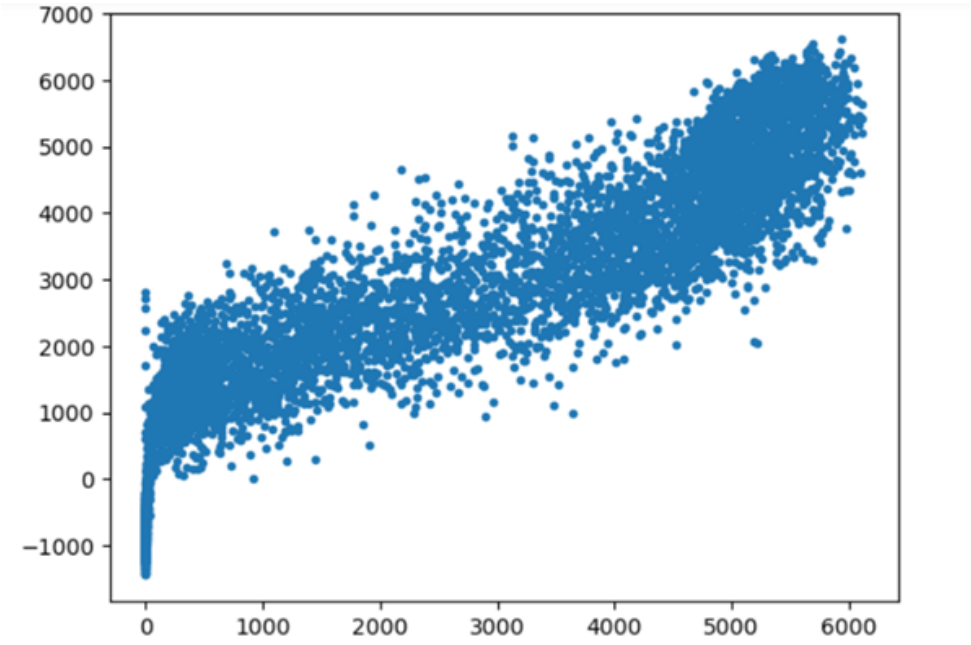
```
import sklearn.metrics as metrics
rsq=metrics.r2_score(yTrain,model.predict(XTrain))
print(f"coefficient de determination R**2 en apprentissage (train):{rsq:.2f}")
```

```
coefficient de determination R**2 en apprentissage (train):0.90
```

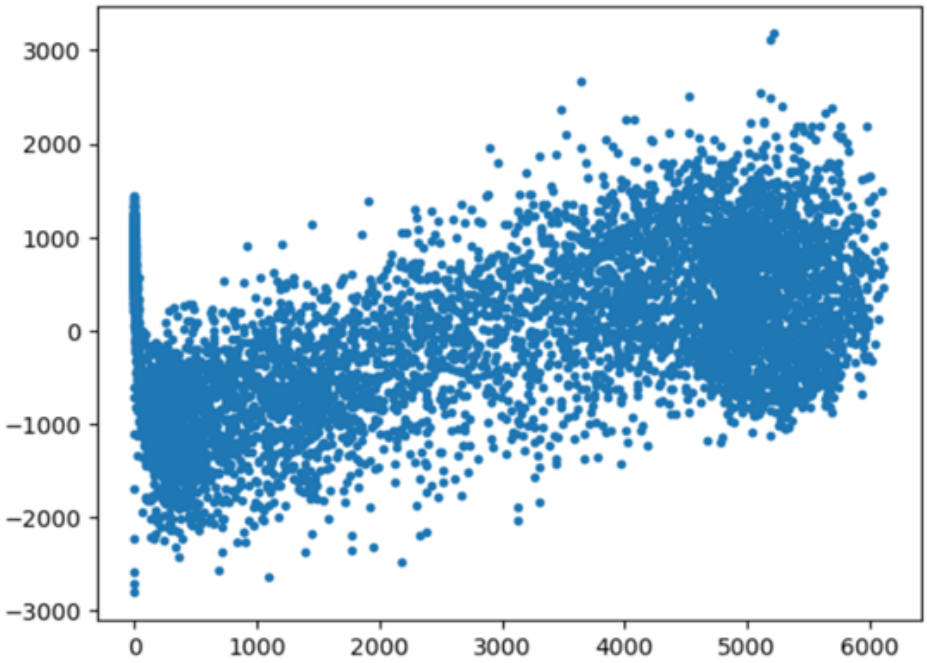
2.2 Interprétation des deux graphiques obtenus

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

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



Ce graphe présente un peu près une relation linéaire entre les vraies valeurs et les valeurs prédites . Ainsi , ce graphe justifie la valeur de coef de détermination ‘0.9 ‘ , vu que la plupart des points suivent une ligne diagonale .



Dans ce graphique, où les vraies valeurs sont tracées par rapport aux résidus (différences entre les vraies valeurs et les valeurs prédites),on observe les résidus sont répartis autour de zéro .

2.3 l'erreur quadratique moyenne

```python
import numpy as np


mse = np.mean(rsq)

# Calculez la racine carrée de la MSE pour obtenir le RMSE.
rmse = np.sqrt(mse)
print("RMSE:", rmse)
```

```
RMSE: 0.9480892073858305
```

2.4 le coefficient de détermination R^2 sur le jeu de test

```python
import sklearn.metrics as metrics
rsq=metrics.r2_score(yTest,model.predict(XTest))
print(f"coefficient de determination R**2 en apprentissage (train):{rsq:.2f}")
```

```
coefficient de determination R**2 en apprentissage (train):0.98
```

On remarque que le coef de détermination pour le test est élevé par rapport au précèdent ,ce qui explique que le modèle ait une très bonne capacité de généralisation aux nouvelles données .

2.5 la variable la plus importante

	importance
BHI	0.554609
Tm	0.173974
Eff	0.115231
GHI	0.067037
DHI	0.029424
Hour	0.023992
BNI	0.008898
Month	0.007571
Tamb	0.006209
TOA	0.004551
P	0.001926
RH	0.001872
WS	0.001713
WD	0.001578
Day	0.001415

BHI est la plus importante

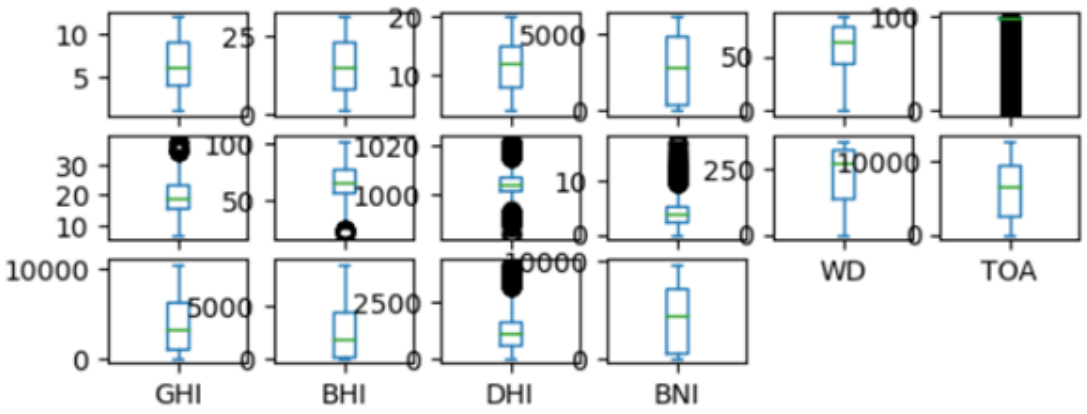
3 TP 2

Dans ce TP , on a exploré , visualisé et mis à l'échelle nos données . Après on a testé plusieurs modèles y compris regression polynomial , La régression à vecteur de support (SVR) et réseaux de neurones artificiels .

3.1 Description du Data

A partir de l'instruction ''D.describe()" , j'ai remarqué qu'il y a une grande différence entre la valeur minimale et maximale de mes données .

3.2 Graphe obtenu



3.3 Justification de l'utilisation de l'outil de normalisation

Les données présentent des amplitudes différentes et que ces différences d'amplitude ne sont pas considérées comme des informations pertinentes pour le modèle.

3.4 Comparaison de la valeur des moyennes avant et après normalisation

```
XTrainStd = scaler.fit_transform(XTrain)
XTestStd = scaler.fit_transform(XTest)
print(numpy.mean(XTrain,axis=0))
```

```
Month       6.351263
Day        15.519018
Hour       11.884048
Tm         56.435012
Eff        85.111175
Tamb       19.552282
RH         67.323443
P        1004.418618
WS          3.924581
WD        228.997543
TOA      6283.139553
GHI      3789.532906
BHI      2595.584642
DHI      1194.763148
BNI      4145.743985
dtype: float64
```

```
print(numpy.mean(XTrainStd,axis=0))
```

```
[0.48647847 0.48396726 0.49275298 0.64818161 0.85111175 0.40444785
 0.56847428 0.54344249 0.22788656 0.63614719 0.48559411 0.36170081
 0.2875168  0.28964594 0.43384047]
```

Remarque : la valeur des moyennes après normalisation est plus petit qu'avant .

3.5 Entrainement du modèle pour les valeurs 2, 3, et 4 du degré du polynôme (Régression Polynomiale)

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import sklearn.metrics as metrics
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)
X_train_poly = polynomial_features.fit_transform(XTrainStd)
X_test_poly = polynomial_features.fit_transform(XTestStd)
model = LinearRegression()
model.fit(X_train_poly,yTrain)
rsq=metrics.r2_score(yTrain,model.predict(X_train_poly))
print(rsq)
rsq=metrics.r2_score(yTest,model.predict(X_test_poly))
print(rsq)
```

```
0.9413853212673228
-10.67754364782113
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=3)
X_train_poly = polynomial_features.fit_transform(XTrainStd)
X_test_poly = polynomial_features.fit_transform(XTestStd)
model = LinearRegression()
model.fit(X_train_poly,yTrain)
rsq=metrics.r2_score(yTrain,model.predict(X_train_poly))
print(rsq)
rsq=metrics.r2_score(yTest,model.predict(X_test_poly))
print(rsq)
```

```
0.964026621665625
-18462.9934555529
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=4)
X_train_poly = polynomial_features.fit_transform(XTrainStd)
X_test_poly = polynomial_features.fit_transform(XTestStd)
model = LinearRegression()
model.fit(X_train_poly,yTrain)
rsq=metrics.r2_score(yTrain,model.predict(X_train_poly))
print(rsq)
rsq=metrics.r2_score(yTest,model.predict(X_test_poly))
print(rsq)
```

```
0.990922575740136
-3155584955734.6763
```

Remarque : Une évolution de R^2 du training et du testing avec l'évolution du degré de la regression polynomiale

3.6 la meilleure configuration d'unités, de couches, d'epochs et de taille du batch

La réponse de cette question est inclus dans le mini projet , ou on a utilisé aussi comme modèle les réseaux de neurones .

4 TP 3

4.1 Repartition de la puissance en 6 classes

La première étape à réaliser durant ce TP est de répartir la puissance photovoltaïque PAC en 6 classes

```
import numpy as np
D['PAC'] = np.where ((D.PAC <= 1000) | (D.PAC==0) ,1, D.PAC)
D['PAC'] = np.where ((D.PAC <= 2000) & (D.PAC>1000) ,2, D.PAC)
D['PAC'] = np.where ((D.PAC <= 3000) & (D.PAC>2000) ,3, D.PAC)
D['PAC'] = np.where ((D.PAC <= 4000) & (D.PAC>3000) ,4, D.PAC)
D['PAC'] = np.where ((D.PAC <= 5000) & (D.PAC>4000) ,5, D.PAC)
D['PAC'] = np.where ((D.PAC <= 6500) & (D.PAC>5000) ,6, D.PAC)
D.PAC
```

4.2 Décompte des classes

```
print (pd.value_counts (D.PAC))
```

```
1.0    3106
6.0    1939
5.0    1762
2.0    1029
4.0     863
3.0     695
Name: PAC, dtype: int64
```

Après le décompte des classes , on remarque que la première classe apparaît 3106 fois, indiquant qu'elle est la catégorie la plus fréquente parmi toutes les valeurs.

4.3 Affichage d'une partie des résultats obtenus

```
Out[21]: array([[0., 0., 0., 1., 0., 0.],
                [0., 0., 0., 1., 0., 0.],
                [0., 0., 0., 0., 0., 1.],
                ...,
                [1., 0., 0., 0., 0., 0.],
                [0., 1., 0., 0., 0., 0.],
                [0., 0., 0., 0., 1., 0.]], dtype=float32)
```

Le résultat est une matrice avec 6 colonnes, contenant des valeurs binaires (0 ou 1). Étant donné que la variable 'PAC' est divisée en 6 classes, cette matrice attribue à chaque valeur son appartenance à la classe correspondante. Chaque colonne représente une classe, et la valeur 1 dans une colonne indique que la valeur donnée appartient à la classe associée, tandis que la valeur 0 indique qu'elle n'appartient pas à cette classe.

4.4 Comparaison de résultat obtenu au nombre d'occurrence de classe calculé précédemment

```
Entrée [22]: #verification
             import numpy as np
             print(np.sum(y,axis=0))

             [3106. 1029.  695.  863. 1762. 1939.]
```

La sortie consiste en une liste qui représente le nombre de valeurs incluses dans chaque classe. Il est observé que ces nombres correspondent à ceux déjà mentionnés précédemment.

4.5 Subdivision des données

```
from sklearn.model_selection import train_test_split
XTrain,XTest,yTrain,yTest = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.6 Normalisation des données

```
import numpy
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
XTrainStd = scaler.fit_transform(XTrain)
XTestStd = scaler.fit_transform(XTest)
print(numpy.mean(XTrain,axis=0))
```

```
Hour      11.865469
Tm        56.322178
Tamb      19.606982
TOA     6273.807769
GHI     3771.936256
dtype: float64
```

4.7 Construction du modèle Prédictif en utilisant les Réseaux de Neurones

```
# modèle de réseau de neurones
from keras.models import Sequential
from keras.layers import Activation,Dense
from sklearn.metrics import r2_score, mean_absolute_percentage_error
from tensorflow.keras.optimizers import Adam
#instanciation du modèle
model = Sequential()
#architecture
model.add(Dense(15,input_dim=5,activation= 'relu'))
model.add(Dense(10,input_dim=5,activation= 'relu'))
model.add(Dense(6, activation='softmax'))
opt=keras.optimizers.Adam(learning_rate=0.1)
#compilation - algorithme d'apprentissage
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
#apprentissage
history = model.fit(XTrain, yTrain, epochs=100, batch_size=10, verbose=0)
```

4.8 Affichage et interprétation de la synthèse du modèle

```
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_33 (Dense)            (None, 15)                90

 dense_34 (Dense)            (None, 10)                160

 dense_35 (Dense)            (None, 6)                 66

=================================================================
Total params: 316 (1.23 KB)
Trainable params: 316 (1.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Voici une interprétation de l'architecture du modèle :

- Couche Dense 1 (dense"="33) : Cette couche a 15 neurones (unités) avec une fonction d'activation ReLU. La sortie de cette couche est de forme (None, 15), indiquant que pour chaque exemple dans le batch, il y a 15 valeurs produites par cette couche.

- Couche Dense 2 (dense"="34) : Cette couche a 10 neurones avec une fonction d'activation ReLU. La sortie de cette couche est de forme (None, 10).

- Couche Dense 3 (dense"="5) : C'est la couche de sortie avec 6 neurones (unités) représentant le nombre de classes dans votre problème. Elle utilise une fonction d'activation softmax, ce qui est approprié pour une tâche de classification multiclasse. La sortie de cette couche est de forme (None, 6)

4.9 Evaluation du modèle Prédictif et interprétation

```
#Evaluation
score=model.evaluate(XTestStd,yTest)
print(score)
print("test_lost:", 100*score[0])
print("test_Accuracy:", 100*score[1])
print('Model errore rate : %.2f%%' % (100-score[1]*100 ))
```

```
59/59 [==============================] - 4s 1ms/step - loss: 1.6743 - accuracy: 0.3294
[1.674330234527588, 0.32943055033683777]
test_lost: 167.4330234527588
test_Accuracy: 32.94305503368378
Model errore rate : 67.06%
```

- la précision sur l'ensemble de test est d'environ 32.94

- Le taux d'erreur du modèle sur l'ensemble de test est d'environ 67.06

Ces résultats suggèrent que le modèle pourrait nécessiter des ajustements ou des améliorations pour obtenir de meilleures performances, comme l'optimisation des hyperparamètres, la modification de l'architecture du modèle, ou l'augmentation de la taille de l'ensemble d'entraînement.

4.10 Vérification de la performance du modèle