

TD N° 1 Machine Learning & Text Mining

Application ML en Reconnaissance des nombres manuscrits

Considérons l'application de machine learning suivante :

```

1. import numpy as np
2. import matplotlib.pyplot as plt
3. import pandas
4. from sklearn.tree import DecisionTreeClassifier
5. data=pandas.read_csv("d:\\train.csv").as_matrix() } importer les modèles
6. clf=DecisionTreeClassifier()
7. x=data[0:21000,1:]
8. label=data[0:21000,0]
9. clf.fit(x,label)
10. xtest=data[21000:,1:]
11. actual_label=data[21000:,0]
12. p=clf.predict(xtest)
13. count=0
14. for i in range(0,21000):
15. [count+=1 if p[i]==actual_label[i] else 0
16. print("Accuracy=", (count/21000)*100)
17. d=xtest[5]
18. Nombre_de_pixels_errones=100
19. for i in range(Nombre_de_pixels_errones):
20. position=np.random.randint(0,784,1)[0]
21. bruit=np.random.randint(-200,200,1)[0]
22. d[position]+=bruit
23. d[position]=d[position]%255
24. print(clf.predict([d]))
25. d.shape=(28,28)
26. plt.imshow(255-d,cmap='gray')
27. plt.show()
    
```

Travail à faire :

1. Analyser le programme ci-dessus en identifiant (dataset, input, output, technique utilisée,...)
2. Identifier les étapes nécessaires à une application de Machine Learning (training data, testing data, testing....)
3. Quelle est la mesure de performance utilisée ?
4. Quelle est l'objectif des lignes de code (18 à 23)
5. Exécuter et tester le programme

TD N° 2 Machine Learning & Text Mining Techniques de regression

Considérons l'application de regression linéaire suivante :

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
df = pd.read_csv('univariate_linear_regression_dataset.csv')
X = df.iloc[:,0]
Y = df.iloc[:,1]
axes = plt.axes()
axes.grid()
plt.scatter(X,Y)
plt.show()
SL=stats.linregress(X, Y)
slope=SL.slope; intercept=SL.intercept; coef_correlation=SL.rvalue
def predict(x):
    return slope * x + intercept
axes = plt.axes()
axes.grid()
plt.scatter(X,Y)
fitLine = predict(X)
plt.plot(X, fitLine, c='r')
plt.show()
```

Travail à faire :

1. Analyser le programme ci-dessus en identifiant (dataset, input, output, technique utilisée,...)
2. Identifier les étapes nécessaires à une application de Machine Learning (training data, testing data, testing....)
3. Quelle est le rôle de iloc ?
4. Exécuter et tester le programme `predict()`
5. Afficher les valeurs de Slope, Intercept, coefficient de corrélation ?
6. Prédire une estimation pour x=22.5

TD N° 3 Machine Learning & Text Mining Techniques de régression (Régression logistique)

Considérons l'application de régression logistique suivante :

```
1. import numpy as np # Chargement de numpy
2. import matplotlib.pyplot as plt # import de Matplotlib
3. from sklearn import datasets
4. from sklearn.linear_model import LogisticRegression
5. from sklearn.model_selection import train_test_split
6. from sklearn.model_selection import cross_val_score
7. iris = datasets.load_iris()
8. X= iris.data[:, :2] # .....
9. # .....
10. y = []
11. for e in iris.target:
12.     if e==0:    y+=[0]
13.     else:    y+=[1]
14. y=np.array(y) plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g')
15. # En Vert les fleurs ayant l'étiquette 0
16. plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y')
17. # en Jaune les fleurs ayant l'étiquette 1
18. plt.legend(labels = ("classe 0", "classe 1"));
19. plt.xlabel("x")
20. plt.ylabel("y")
21. model = LogisticRegression(C=1e20)#tester avec 0.01
22. x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
23. #.....
24. #.....
25. #.....
26. model.fit(x_train, y_train)
27. score1=model.score(x_test, y_test)
28. score2=model.score(x_train, y_train)
29. print("score sur test-set =" +str(score1*100)+"%")
30. print("score sur train-set =" +str(score2*100)+"%")
31. theta0=model.intercept_; theta1=model.coef_[0][0]; theta2=model.coef_[0][1]
32. Iries_To_Predict = [ [5.5, 2.5], [7, 3], [3,2], [5,3]]
33. # demande de prédiction
34. print(model.predict(Iries_To_Predict))
35. score=cross_val_score(model,x_train, y_train, cv=5,scoring='accuracy')
36. print(score)
37. print(score.mean())
38. def fct_reg_logistic(x1,x2):
39.     z=(theta0)+theta1*x1+(theta2*x2)
40.     return(z)
41. print([fct_reg_logistic(e[0],e[1]) for e in Iries_To_Predict])
```

Travail à faire :

1. Consulter Iris dataset et identifier le type initial du problème ? régression ?classification multiclass ou binaire ?
2. Quel est l'objectif des instructions ligne 8 à la ligne 13 ?
3. Quel est l'intérêt de la fonction `train_test_split`? Chercher le rôle de ses paramètres : `test_size`? `Random_state`?
4. À quoi correspond le `score1` et le `score2` et quelle est la différence entre les deux ?
5. Réexecuter l'expérimentation en variant `test_size` par 0.3, 0.4 et 0.5 et noter les scores correspondants ?
6. Pourquoi la fonction `cross_val_score` est utilisé dans la ligne 35? Changer le paramètre `cv` et examiner les résultats obtenus ?

TD N° 4 Machine Learning & Text Mining

Techniques de représentation des données textuelles et applications

Considérons le programme suivant :

```
1. from sklearn.feature_extraction.text import CountVectorizer
2. texte = ["La vie est douce","La vie est tranquille, est belle, est douce"]
3. vect = CountVectorizer()
4. T= vect.fit_transform(texte)
5. dictionnaire_des_mots=vect.vocabulary_
6. print("dictionnaire_des_mots :", dictionnaire_des_mots)
7. liste_des_mots=list(dictionnaire_des_mots.keys())
8. print("liste_des_mots :", liste_des_mots)
9. Matrice_sparse_correspondante=T.toarray()
10. print("Matrice_sparse_correspondante:\n",Matrice_sparse_correspondante)
```

Travail à faire :

1. Exécuter le programme précédent

2. Identifier le rôle de la fonction `fit_transform()` :

.....

3. Identifier le rôle de l'attribut `vocabulary_`:

.....

4. Identifier le rôle de la fonction `keys ()` :

.....

5. Quel est le résultat final de ce programme : (changer le texte en cas de besoin)

.....

.....

6. Appeler la fonction `CountVectorizer` avec l'argument (`binary=True`) dans la ligne 3, analyser le résultat du programme et identifier le rôle du paramètre `binary`

TD N° 6 Machine Learning & Text Mining

Les arbres de décision

Considérons le programme suivant pour le calcul de l'entropie

```
1. from numpy import *
2. import pandas
3. df = pandas.DataFrame([[0,1,'C1'],[0,0,'C1'],
4. [1,1,'C2'],[1,0,'C2']],columns=['A', 'B', 'Classes'])
5. df=df.to_numpy()
6.
7. def Entropie(S):
8.     classes_distinctes=set(S[:, -1])
9.     classes=list(S[:, -1])
10.    s=0
11.    for c in classes_distinctes:
12.        p=classes.count(c)/len(classes)
13.        s+=p*log2(p)
14.    return(-1.0*s)
15. print(Entropie(df))
```

Travail à faire :

1. Quel est le rôle de la méthode `to_numpy()` ?

.....

2. Quel est le résultat de `set(S[:, -1])` ?

.....

3. Quel est le résultat de `list(S[:, -1])` ?

.....

4. Exécuter le programme et noter l'entropie :

.....

5. Changer le DataFrame df de tel sorte à avoir une entropie égale à 0 et une entropie supérieure à 1.

.....

.....

Considérons le programme suivant pour le calcul du gain de l'information

```
16. def Gain_d_information(S,numero_colon_attribut):  
17.     classes=list(set(S[:, -1]))  
18.     valeurs_attribut=list(set(S[:, numero_colon_attribut]))  
19.     Si=[[ ] for i in range(len(valeurs_attribut))]  
20.     for e in S:  
21.         e=list(e)  
22.         val_attribut_pour_e=e[numero_colon_attribut]  
23.         numero_sous_ensemble=valeurs_attribut.index(val_attribut_pour_e)  
24.         Si[numero_sous_ensemble].append(e)  
25.     Si=array(Si) ; som=0  
26.     for sous_ensemble in Si:  
27.         som+=(len(sous_ensemble)/len(S))*Entropie(sous_ensemble)  
28.     return(Entropie(S)-som)
```

Travail à faire :

6. Analyser et exécuter le programme en identifiant l'objectif des lignes :

```
valeurs_attribut=list(set(S[:, numero_colon_attribut]))
```

-
.....
.....
.....
.....
.....
.....
.....
7. Calculer le gain de l'information des différents attributs:

-
.....
.....
.....
.....
8. Calculer les gains d'information concernant cette dataset :

	A	B	C
1	Douleur	Inanime	Infarctus
2	poitrine	oui	oui
3	ailleurs	oui	oui
4	poitrine	non	oui
5	poitrine	oui	oui
6	ailleurs	non	oui
7	poitrine	non	non
8	ailleurs	non	non
9	poitrine	oui	non
10	ailleurs	non	non
11	ailleurs	non	non

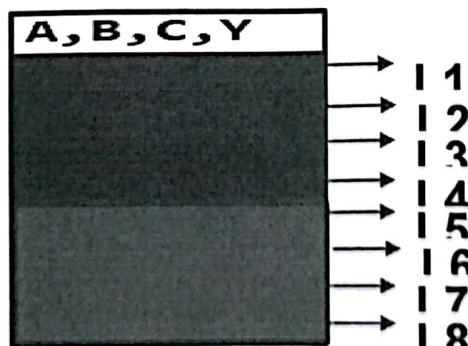
9. En se basant sur les fonctions précédentes :
- Ecrire une fonction **GINI(S)** qui calcule et retourne l'indice de GINI d'un ensemble
 - Ecrire une fonction **Mesure_desordre(S,numero_colon_attribut)** qui calcule et retourne le désordre selon un attribut de partitionnement
 - Utiliser le programme obtenu pour calculer Indice de GINI et le désordre des DataFrame précédents.

Considérons la fonction suivante qui calcule le désordre pour les valeurs continues :

```

22 def Mesure_desordre(S,numero_colon_attribut):
23     Bonne_mesure_desordre=+inf
24     Bonne_valeur_de_repartition=0
25     classes=list(set(S[:, -1]))
26     valeurs_attribut=list(set(S[:, numero_colon_attribut]))
27     valeurs_attribut.sort()
28     Gauche=[]; Droite=[]
29     for i in range(len(valeurs_attribut)-1):#Le dernier est le max !
30         v=valeurs_attribut[i]; Gauche.clear(); Droite.clear()
31         for e in S:
32             val_attribut_pour_e=e[numero_colon_attribut]
33             if val_attribut_pour_e<=v:
34                 Gauche.append(e)
35             else:
36                 Droite.append(e)
37             pGau=len(Gauche)/len(S)
38             pDr=len(Droite)/len(S)
39             mesure=(pGau)*GINI(Gauche)+(pDr)*GINI(Droite)
40             if mesure<Bonne_mesure_desordre:
41                 Bonne_mesure_desordre=mesure
42                 Bonne_valeur_de_repartition=v
43     return(Bonne_mesure_desordre,Bonne_valeur_de_repartition)

```



10. Expliquer comment la fonction calcule et retourne la valeur de répartition ?
-
.....
.....

11. Comment peut-on sélectionner le meilleur attributs à utiliser le partitionnement et la meilleure valeur ?

Considérons le programme suivant :

```
1 def meilleur_attribut_meilleure_valeur(S):  
2     meilleur_attribut=0  
3     meilleure_val=0  
4     meilleure_mesure=+inf  
5     for num_attribut in range(len(S[0])-1):  
6         mesure,valeur=Mesure_desordre(S,num_attribut)  
7         if mesure<meilleure_mesure:  
8             meilleure_val=valeur  
9             meilleure_mesure=mesure  
10            meilleur_attribut=num_attribut  
11    return meilleur_attribut,meilleure_val  
12 print(meilleur_attribut_meilleure_valeur(Data_train))
```

12. Exécuter ce programme et noter les résultats concernant le meilleur attributs à utiliser le partitionnement et la meilleure valeur ?

Considérons le programme suivant :

```
1 X = data[:8,0:3]; Y = data[:8,3] ; X_test=data[8:,0:3]; y_test=data[8:,3]  
2 clf_gini = DecisionTreeClassifier(criterion = "gini")  
3 clf_gini.fit(X, Y)  
4 clf_entropy = DecisionTreeClassifier(criterion = "entropy")  
5 clf_entropy.fit(X, Y)  
6 print("Results Using Gini Index:")  
7 y_pred_gini = clf_gini.predict(X_test)  
8 print("Confusion Matrix: ",confusion_matrix(y_test, y_pred_gini))  
9 print(X_test,y_test,y_pred_gini)  
10 print ("Accuracy : ",accuracy_score(y_test,y_pred_gini)*100)  
11 print("Results Using Entropy:")  
12 y_pred_entropy = clf_entropy.predict(X_test)  
13 print("Confusion Matrix: ",confusion_matrix(y_test, y_pred_entropy))  
14 print ("Accuracy : ",accuracy_score(y_test,y_pred_entropy)*100)
```

13. Combien y a-t-il de données dans l'ensemble d'apprentissage et dans l'ensemble de test ?
14. Quel est le meilleur critère à utiliser dans l'arbre de décision ? entropie ou l'indice de GINI ?
15. Avec l'indice GINI ? quel est le nombre de Fp et Fn ?

Pour visualiser l'arbre de décision générer utiliser le code suivant :

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

svg_graph = graph.create_svg()
svg = open('gini_tree.svg', 'wb')
svg.write(svg_graph)
svg.close()
Image(graph.create_png())
```

TD N° 7 Machine Learning & Text Mining Les arbres de décision – Application Crédit Scoring

Etude de cas :

Le processus d'acceptation d'un crédit à la consommation doit être très rapide en particulier lorsque les clients ont besoin d'un crédit en magasin au moment de leur achat (exemple : paiement en trois fois sans frais, etc.).

Depuis très longtemps, les banques et les organismes de crédit ont développé des méthodes automatiques et instantanées de notation de la solvabilité des clients et de leur capacité à rembourser le crédit. Ces outils sont très utiles pour une acceptation rapide d'octroi de crédit, mais également pour déterminer le montant de l'enveloppe d'autorisation de crédit ou pour effectuer le suivi du risque des clients et des portefeuilles de prêts.

Le score de crédit est une fonction qui attribue une valeur de qualité de crédit à un client ou un prêt en fonction de variables explicatives telles que le ratio d'endettement de l'emprunteur, son comportement de compte ou tout autre grandeur qui est corrélée au défaut de l'emprunteur.

Tavail à faire :

1. Analyser le data set ? Combien de données ? de features ? quel est la variable dépendante ?
2. Y a-t-il des problèmes dans le data set ?
3. Analyser gain d'information ou indice de GINI de chaque attribut ?
4. Y a-t-il des problèmes dans le data set ?
5. Créer un modèle de prédition en utilisant les arbres de décision ? la taille de data testing doit être 0.33

TD N° 8 Machine Learning & Text Mining Les arbres de régression

Considérons le programme suivant qui utilise les arbres de décision pour un problème de regression.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.tree import DecisionTreeRegressor
4 # Créer les données d'apprentissage
5 np.random.seed(0)
6 X = np.sort(5 * np.random.rand(200, 1), axis=0)
7 y = np.sin(X)
8 plt.plot(X, y)
9 plt.title("Courbe de répartition sinusoïdale:")
```

```
from sklearn.metrics import mean_squared_error
reg = DecisionTreeRegressor(max_depth=2)
reg.fit(X, y)
X_test = np.sort(5 * np.random.rand(80, 1), axis=0)
y_predicted = reg.predict(X_test)
y_test = np.sin(X_test)
plt.figure()
plt.scatter(X, y, color="b", label="Exemples d'apprentissage")
plt.plot(X_test, y_predicted, color="r", label="Prédiction", linewidth=3)
plt.xlabel("x"); plt.ylabel("y")
plt.title("Régression par un arbre de décision")
plt.legend(); plt.show()
a=mean_squared_error(y_test, y_predicted); print("mean_squared_error: ",a)
```

Travail à faire :

1. Analyser le programme ci-dessus en le exécutant ?
2. En consultant les paramètres de DecisionTreeRegressor : **max_depth** et **min_samples_leaf**.
 - a. Changer la valeur de **max_depth** par 200 au lieu de 2 en premier temps ? est ce que vous remarquez ?
 - b. Changer la valeur de **min_samples_leaf** ? et noter vos remarques concernant les performances ?

TD N° 10 Machine Learning & Text Mining SVM

Activité 1 : Compréhension du SVM Linéaire

Considérons les programmes suivants :

```

1 import pandas as pd; from numpy import *
2 import matplotlib.pyplot as plt
3 from sklearn.metrics import confusion_matrix
4 donnees= pd.read_csv("D:\\pointsSVM.csv",sep=';')
5 donnees=array(donnees); x=donnees[:,0:2]; y=donnees[:,2];
6 axes = plt.axes(); axes.grid()
7 plt.scatter(x[y ==+1][:, 0], x[y ==+1][:, 1], color='r')
8 plt.scatter(x[y ==-1][:, 0], x[y == -1][:, 1], color='b')
9 def f(X,Beta):    return Beta[0]+sum(X.T*Beta[1:])
10 def classer(X,Beta):   return int(sign(f(X,Beta)))
11 from sklearn.svm import LinearSVC, SVC
12 lsvm = LinearSVC(C=10000); X_train=donnees[:,0:2]; y_train=donnees[:,2]
13 lsvm.fit(X_train, y_train); score = lsvm.score(X_train, y_train)
14 print(score); Beta=list(lsvm.intercept_)+list(lsvm.coef_[0]); print(Beta)
15 plt.show()

```

```

1 axes = plt.axes(); axes.grid()
2 plt.scatter(x[y ==+1][:, 0], x[y ==+1][:, 1], color='r')
3 plt.scatter(x[y ==-1][:, 0], x[y == -1][:, 1], color='b')
4 Bleu=[]; Rouge=[]; milieu=[]
5 for abscisseX in range(1,13):
6     y=-1.0*(Beta[0]+Beta[1]*abscisseX+1)/Beta[2]; Bleu+=[y]
7     y=-1.0*(Beta[0]+Beta[1]*abscisseX-1)/Beta[2]; Rouge+=[y]
8     y=-1.0*(Beta[0]+Beta[1]*abscisseX)/Beta[2]; milieu+=[y]
9 plt.plot(arange(1,13),Bleu,color='b')
10 plt.plot(arange(1,13),Rouge,color='r')
11 plt.plot(arange(1,13),milieu,color='g')
12 plt.show()
13 print(Beta)
14 print('[5,1]==>',f(array([5,1]),Beta));print('[2,1]',f(array([2,1]),Beta))
15 print('[6,9]==>',f(array([6,9]),Beta));print('[12,2]',f(array([12,1]),Beta))

```

Activité 2 : Compréhension du SVM non Linéaire

Considérons le programme suivant :

```
1 import pandas as pd; from numpy import *
2 import matplotlib.pyplot as plt
3 donnees= pd.read_csv("D:\\pointsSVM2.csv",sep=';')
4 donnees=array(donnees); x=donnees[:,0:2]; y=donnees[:,2];
5 axes = plt.axes(); axes.grid()
6 plt.scatter(x[y ==+1][:, 0], x[y ==+1][:, 1], color='r')
7 plt.scatter(x[y ==-1][:, 0], x[y == -1][:, 1], color='b')
8 def f(X,Beta):    return Beta[0]+sum(X.T*Beta[1:])
9 def classer(X,Beta):    return int(sign(f(X,Beta)))
10 from sklearn.svm import LinearSVC, SVC,NuSVC
11 X_tr=donnees[:,0:2]; y_tr=donnees[:,2]
12 lsvm= SVC(gamma='auto');lsvm.fit(X_tr, y_tr);
13 score = lsvm.score(X_tr, y_tr); print(score);
14 lsvm= NuSVC(gamma='auto');lsvm.fit(X_tr, y_tr);
15 score = lsvm.score(X_tr, y_tr); print(score);
16 lsvm= LinearSVC(C=1000);lsvm.fit(X_tr, y_tr);
17 score = lsvm.score(X_tr, y_tr); print(score);
18 Beta=list(lsvm.intercept_)+list(lsvm.coef_[0]); print(Beta)
19 plt.show()
```

Activité 3 : Application du SVM en Sentiment Analysis

Considérons le programme suivant :

```
1 with open("D:\\reviews.txt") as f:  
2     reviews = f.read().split("\\n")  
3 with open("D:\\labels.txt") as f:  
4     labels = f.read().split("\\n")  
5 reviews_tokens = [review.split() for review in reviews]  
6 from sklearn.preprocessing import MultiLabelBinarizer  
7 MLB = MultiLabelBinarizer()  
8 MLB.fit(reviews_tokens)  
9 from sklearn.model_selection import train_test_split  
10 X_train, X_test, y_train, y_test = train_test_split(\  
11     reviews_tokens, labels, test_size=0.25, random_state=None)  
12 from sklearn.svm import LinearSVC  
13 lsvm = LinearSVC(); lsvm.fit(MLB.transform(X_train), y_train)  
14 score = lsvm.score(MLB.transform(X_test), y_test); print(score*100,'%')
```

85.44 %

Travail à faire :

1. Comprendre, analyser et exécuter le programme ci-dessous ? en précisant le type du SVM utilisé ?
2. Chercher le meilleur modèle SVM en menant une étude empirique sur le type de SVM et les hyperparamètres afin d'avoir le meilleur Score ?

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('univariate_linear_regression_dataset.csv')
x= df.iloc[:,0]; n_samples = len(x); x=np.array(x).reshape((n_samples, 1)) #96
y= df.iloc[:,1]; y=np.array(y).reshape((n_samples, 1))
x=np.array(x).reshape((n_samples, 1))
plt.scatter(x, y) # afficher les résultats. X en abscisse et y en ordonnée
plt.show()
# ajout de la colonne de biais a X
X = np.hstack((x, np.ones(x.shape)))
# création d'un vecteur parametre theta
theta = np.random.randn(2, 1)
print(theta)

def f(X, theta):
    return X.dot(theta)
def erreur_somme_des_distances(X, y, theta):
    m = len(y)
    return (1/(2*m))*np.sum((f(X, theta)-y)**2)
def gradient(X, y, theta):
    m = len(y)
    return 1/m * X.T.dot(f(X, theta) - y)
def gradient_descent(X, y, theta, learning_rate, n_iterations):
    # création d'un tableau de stockage pour enregistrer l'évolution des erreurs
    historique_des_erreurs = np.zeros(n_iterations)
    for i in range(0, n_iterations):
        theta = theta - learning_rate * gradient(X, y, theta) # mise a jour du theta
        historique_des_erreurs[i] = erreur_somme_des_distances(X, y, theta)
    return theta, historique_des_erreurs
# Example de test :
n_iterations = 10000
learning_rate = 0.01 # modifier pour voir l'effet
theta_final, historique_des_erreurs=gradient_descent(X, y, theta, learning_rate, n_iterations)
print(theta_final) # theta une fois que la machine a été entraînée
# création d'un vecteur prédictions qui contient les prédictions de notre modèle final
predictions = f(X, theta_final)
# Affiche les résultats de prédictions (en rouge) par rapport a notre Dataset (en bleu)
plt.scatter(x, y)
plt.plot(x, predictions, c='r')
plt.show()
plt.figure()
plt.plot(range(n_iterations), historique_des_erreurs)
plt.show()
print(f(np.array([22.5,1]), theta_final))

```

1. Analyser le programme ci-dessus en identifiant (dataset, input, output, technique utilisée,...)
2. Exécuter et tester le programme
3. Quel est le rôle n_iterations et learning_rate ? modifier la valeur et identifier l'impact sur le modèle ?

Considérons l'application de régression polynomiale suivante :

```
def afficher_pol(LPol):
    L=LPol[::-1]; L.reverse()
    L=[round(e,2) for e in L]
    for i in range(len(L)):
        print('('+str(L[i])+'*X^'+str(i)+')',end=' + ')
def Evaluer_pol(LPol,x):
    s=0
    L=LPol[::-1]
    L.reverse()
    for i in range(len(LPol)):
        s+=L[i]*(x**i)
    return(s)
import numpy as np
import matplotlib.pyplot as plt
import pandas
data=pandas.read_csv('dataset_polynomial3.csv').values
x=data[1:1001,0]
y=data[1:1001,1]
degre_polynome=2
p= np.poly1d(np.polyfit(x, y,degre_polynome))
LPol=list(p)
afficher_pol(LPol)
les_x= np.linspace(min(x), max(x), 100)
plt.scatter(x, y)
plt.plot(les_x, p(les_x), c='r')
plt.show()
```

Travail à faire :

1. Analyser le programme ci-dessus en identifiant (dataset, input, output, technique utilisée,...)
2. Quel est le rôle des fonctions afficher_pol et evaluer_pol ?
3. Exécuter et tester le programme
4. Est-ce que le polynôme modélise correctement la relation entre la variable prédictive et la variable cible ? sinon chercher une polynôme adéquate ?