TP2 Apprentissage supervisé

Régression et classification

Université Sorbonne Paris Nord

ING2 Instrumentation

Réalisé par :

MHADHEBI ZIED

AIT NOURI Rayane

Introduction :

Dans ce TP, nous avons exploré les concepts d'apprentissage supervisé en nous concentrant sur la régression et la classification. Nous avons utilisé diverses bibliothèques Python telles que numpy, scipy, sklearn, et matplotlib pour manipuler, analyser et visualiser les données. Les modèles de machine learning ont été implémentés à l'aide de la librairie Scikit-Learn. Ce compte-rendu présente les exercices effectués, les méthodes utilisées et les résultats obtenus.

Exercice 1 : Régression linéaire

1) Importation des modules nécessaires :

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Regression
```

LinearRegression est utilisé pour implémenter la régression linéaire.

LogisticRegression est utilisé pour implémenter la régression logistique (classification).

2) Construction d'un jeu de données formé de m échantillons :

```
np.random.seed(0)
m=100
X = np.linspace(0,10,m).reshape(m,1)
Y = X + np.random.random_sample((m,1))
```

X représente des données indépendantes.

Y représente les données dépendantes.

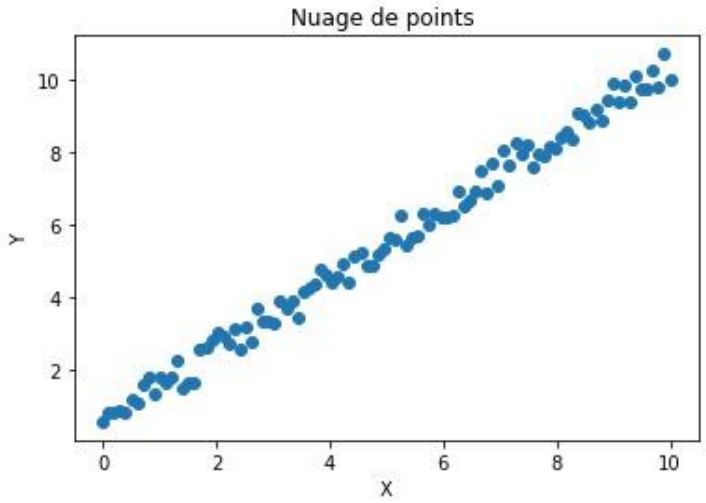
Donc X et Y sont les valeurs des points

3) Affichage de nuage des points :

Code utilisé :

```
plt.scatter(X, Y)
plt.title("Nuage de points ")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Nuage de points obtenu :



on a utilisé Y = X + np.random, ce qui nous donne une droite linéaire d'une pente égale à 1 (le random nous met des points autours de la droite, comme on peut le remarquer sur la figure ci-dessus)

Le bruit introduit une certaine dispersion des points autour de la ligne idéale Y=X. Cela reflète des variations dans des données réelles.

4) Représentation du nuage de points :

a) Entrainement du modèle (model.fit(X,y))

```
model = LinearRegression()
model.fit(X, Y)
```

`model.fit(X, Y)` ajuste le modèle linéaire aux données.

b) Evaluaion de modele :

```
score = model.score(X, Y)
print(f"Score pour la régression linéaire : {score:.2f}")
```

Et on a obtenu un score de 0.99 (99%)

```
Score pour la régression linéaire : 0.99
```

⇨ Le modèle de régression linéaire ajuste bien les données linéaires avec un score proche de 1.

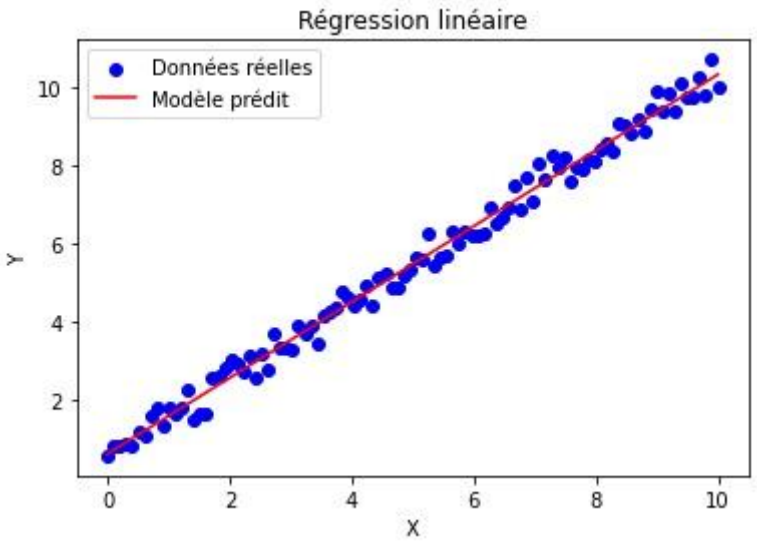
c) Sueprposition :

Code :

```
Y_pred = model.predict(X)
plt.scatter(X, Y, color='blue', label='Données réelles')
plt.plot(X, Y_pred, color='red', label="Modèle prédit")
plt.title("Régression linéaire")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```

Résultat obtenu :

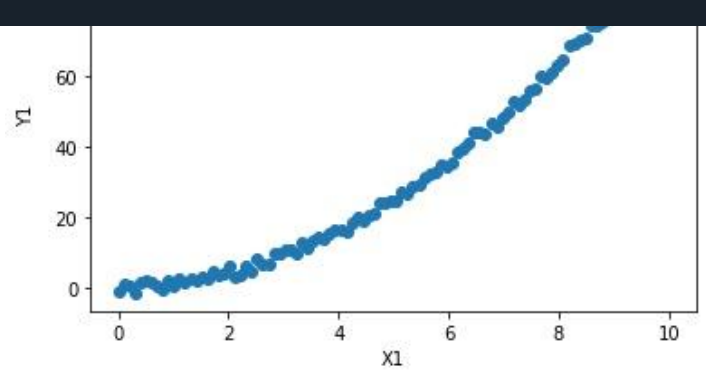
Le graphique montre une ligne droite qui passe bien à travers le nuage de points.



5) + 6) Construction et affichage du nouveau jeu de données :

Code :

```
X2 = np.linspace(0,10,m).reshape(m,1)
Y2 = X2**2 + np.random.randn(m,1)
plt.scatter(X2,Y2)
plt.title("Deuxieme Nuage de points ")
plt.xlabel("X2")
plt.ylabel("Y2")
plt.show()
```



Résultat :

La relation entre X et Y n'est plus linéaire.

Un modèle de régression linéaire ne pourra pas ajuster ces données correctement.

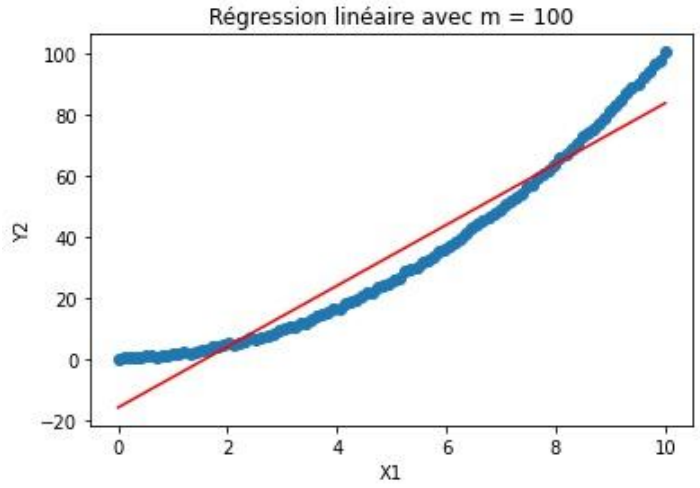
7) Représentation de ce nuage par un modèle de régression linéaire :

```python
X2 = np.linspace(0,10,m).reshape(m,1)
Y2 = X2 ** 2 + np.random.randn(m,1)
plt.scatter(X2,Y2)
plt.title("Deuxieme Nuage de points ")
plt.xlabel("X2")
plt.ylabel("Y2")
plt.show()

model1 = LinearRegression()
model1.fit(X2, Y2)
score1 = model1.score(X2, Y2)
print(f"Score pour le deuxieme nuage : {score1:}")

for m in [100, 1000, 10000]:
    X1 = np.linspace(0, 10, m).reshape(m, 1)
    Y1 = X1 ** 2 + np.random.random((m, 1))
    model.fit(X1, Y1)
    plt.scatter(X1, Y1)
    plt.plot(X1, model.predict(X1), color='red')
    plt.xlabel('X1')
    plt.ylabel('Y2')
    plt.title(f'Régression linéaire avec m = {m}')
    plt.show()
```

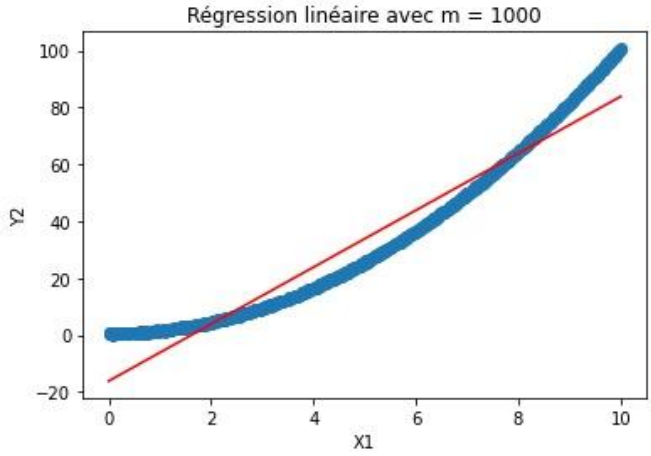
L'apprentissage est réussi par ce que :

```
Score pour le deuxieme nuage : 0.9353934515416071
```

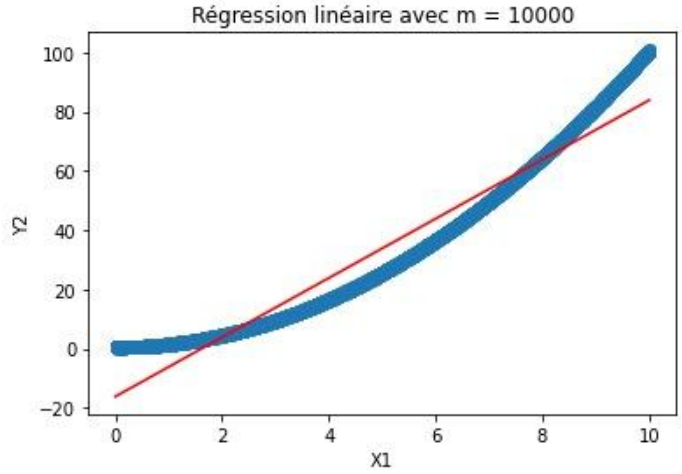
Régression avec m = 100 :



Régression avec m = 1 000 :



Régression avec m = 10 000 :



On a changé plusieurs fois le m on remarque une petite infiltration et malheureusement on n'a pas eu le bon résultat alors on conclue que le model2.predict() qui n'est pas adéquat, alors on doit le changé

8) Modèle SVR :
   SVR cherche à trouver une fonction qui minimise l'erreur de prédiction tout en maximisant la marge entre les points de données te la fonction de régression contrairement à la regression linéaire.

```
model_SVR = SVR(C=100)
model_SVR.fit(X2, Y2.ravel())  # .ravel() pour convertir Y en vecteur 1D
score_SVR = model_SVR.score(X2, Y2.ravel())
print(f" pour SVR avec C=100 : {score_SVR:}")
```

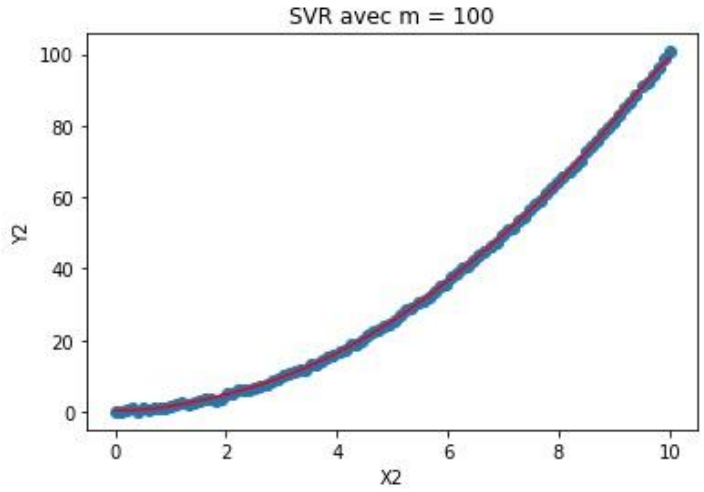
C : Ce paramètre contrôle la pénalité pour les erreurs de prédiction, lorsque C est grand, il pénalise fortement les erreurs.

Maintenant on peut afficher le SVR avec m=100, 1000 ou 10000 grâce a ce code qu'on trouve ci-dessous :

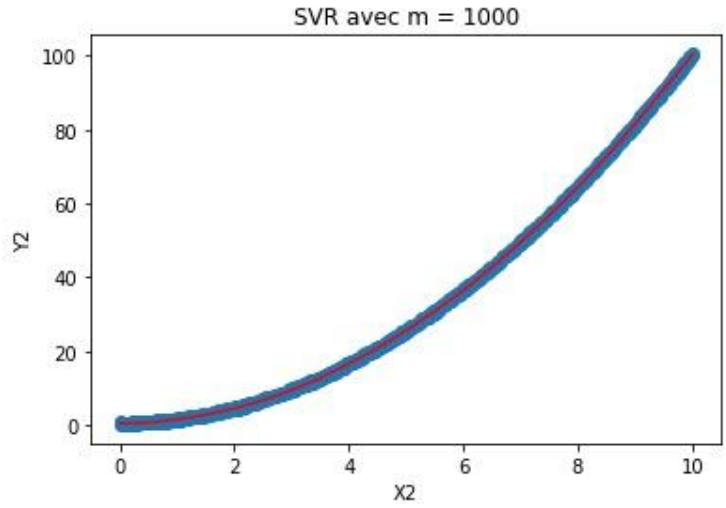
```
for m in [100, 1000, 10000]:
    X2 = np.linspace(0, 10, m).reshape(m, 1)
    Y2 = X2 ** 2 + np.random.random((m, 1))
    model_SVR = SVR(C=100)
    model_SVR.fit(X2, Y2.ravel())
    plt.scatter(X2, Y2)
    plt.plot(X2, model_SVR.predict(X2), color='red')
    plt.xlabel('X2')
    plt.ylabel('Y2')
    plt.title(f'SVR avec m = {m}')
    plt.show()
```

Résultat :

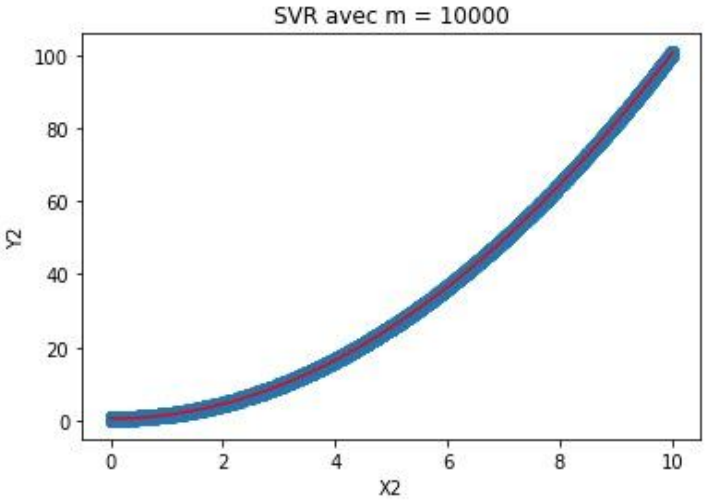
m=100 :



m=1000 :



m=10 000 :



L'ajustement s'améliore avec l'augmentation des échantillons, ce qui montre la robustesse de SVR face aux grandes quantités de données, on conclue que le model3.predict() est bon ,il suit le nuage des points.

Exercice 2 : Classification KNN

L'objectif de cet exo est d'utiliser et travailler avec des réelles du Titanic.

1) Importation des librairies :

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
```

Pour cet exercice, nous avons commencé par importer les librairies nécessaires

- Panda : Pour manipuler et analyser les données
- Seaborn : Pour visualiser les données.

Ces librairies sont essentielles pour l'analyse et la visualisation des données du jeu de données du Titanic

2) Téléchargement du jeu de données du Titanic en utilisant seaborn :

```
titanic = sns.load_dataset('titanic')
print(titanic.shape)
print(titanic.head())
```

Nous avons téléchargé le jeu de données du Titanic à l'aide de la librairie seaborn. Ce jeu de données contient des informations sur les passagers du Titanic, telles que leur survie, leur classe, leur sexe et leur âge, qu'on pourrait les visualiser dans la question suivante.

3) Analyse du jeu de données en appuyant sur titanic.head() :

```
(891, 15)
   survived  pclass     sex   age  ...  deck  embark_town  alive  alone
0         0       3    male  22.0  ...   NaN  Southampton     no  False
1         1       1  female  38.0  ...     C    Cherbourg    yes  False
2         1       3  female  26.0  ...   NaN  Southampton    yes   True
3         1       1  female  35.0  ...     C  Southampton    yes  False
4         0       3    male  35.0  ...   NaN  Southampton     no   True

[5 rows x 15 columns]
```

- Analyse :

Nous avons affiché les premières lignes du jeu de données du Titanic pour avoir un aperçu de sa structure et de son contenu, le jeu de données contient 891 lignes et 15 colonnes, incluant des informations telles que la survie des passagers « survived », leur classe « pclass », leur sexe, leur âge, et d'autres caractéristiques telles que si il était seuls ou pas.

Cet aperçu nous permet de comprendre les types de données disponibles et de planifier les étapes de nettoyage et d'analyse qui vont suivre.

4) Nettoyage du jeu de données afin de récupérer uniquement les données sur lesquelles nous souhaiterons travailler :

Pour préparer le jeu de données du Titanic, nous avons d'abord sélectionné uniquement les colonnes « survived, pclass, sex, age », ensuite, nous avons supprimé les données manquantes pour garantir la complétude des données.

Enfin, nous avons numérisé le jeu de données en remplaçant ‘male’ par 0 et ‘female’ par 1, et pour réaliser toutes ces étapes, nous avons utilisé le programme suivant :

```python
titanic = titanic[['survived', 'pclass', 'sex', 'age']] # Sélection des colonnes

titanic.dropna(axis=0, inplace=True) # Suppression des données manquantes

titanic['sex'].replace(['male', 'female'], [0, 1], inplace=True) #  Numérisation du jeu de données
print(titanic.head())
```

Et on a obtenu :

	survived	pclass	sex	age
0	0	3	0	22.0
1	1	1	1	38.0
2	1	3	1	26.0
3	1	1	1	35.0
4	0	3	0	35.0

Ce processus nous a permis d'obtenir un jeu de données propre et prêt à être utilisé pour l'entraînement de notre modèle de classification

5) Entrainement du modele pour savoir si un passager survivra au naufrage : :

a. Division du tableau en deux taleaux X et Y :

```
Y = titanic['survived']
X = titanic.drop('survived', axis=1)
```

Y contient les étiquettes de survie (0 ou 1).

X contient les caractéristiques (features) des passagers (pclass, sex, age)

Cette division permet au modèle de classification d'apprendre à partir des caractéristiques des passagers pour prédire leur survie.

Sélection du modèle de classification KNeighborsClassifier, puis entraînement du modèle avec model.fit(X, y) et évaluation du score avec model.score(X, y).

Voici le code que nous avons utilisé pour cette partie :

```
model = KNeighborsClassifier()

model.fit(X, Y)

score = model.score(X, Y)
print(f"Score du modèle : {score}")
```

Et cela nous a permis d’obtenir le score du modele qui est de 84.17%

```
Score du modèle : 0.8417366946778712
```

Ce score indique que le modèle a une bonne performance pour prédire la survie des passagers en fonction des caractéristiques fournies.

6) Prédiction de survie d'un passager :

Construction d'une fonction « survie » qui prend en parametre « pclass », « sex » et « age » :, et prendre les parametres pclass=3, sex=0, age=26, pour voir si ce passager a survecu, et aussi, calculer la probabilité pour appartenir à la classe 0 et à la classe 1 en ajoutant à la fonction « print(model.predict_proba(x)) :

- Code utilisé pour faire cette partie :

```
def survie(model, pclass=3, sex=0, age=26):
    x = np.array([pclass, sex, age]).reshape(1, 3)
    prediction = model.predict(x)
    probabilities = model.predict_proba(x)
    print(f"Prédiction de survie : {prediction[0]}")
    print(f"Probabilités : {probabilities}")
    return prediction, probabilities


survie(model, pclass=3, sex=0, age=26)
```

- Résultat obtenu :

```
Prédiction de survie : 0
Probabilités : [[0.8 0.2]]
```

La prédiction de survie = 0, donc le passager n'a pas survecu

La probabilité est de 0.8 pour la classe 0 et 0.2 pour la classe 1.

Ces résultats montrent que le modèle prédit que le passager n'a pas survécu, avec une probabilité de 80% pour la non-survie et de 20% pour la survie.

7) Variation du nombre de voisins de 1 à 10 du modele KNeighborsClassifier :

- Programme utilisé :

```python
best_score = 0
best_n_neighbors= 0

for n_neighbors in range(1, 11):

    model = KNeighborsClassifier(n_neighbors=n_neighbors)
    model.fit(X, Y)
    score = model.score(X, Y)
    print(f'Score avec {n_neighbors} voisins : {score}')
    if score > best_score:
        best_score = score
        best_n_neighbors = n_neighbors

print(f'Meilleur nombre de voisins : {best_n_neighbors} avec un score de {best_score}')



best_model = KNeighborsClassifier(n_neighbors=best_n_neighbors)
best_model.fit(X, Y)
survie(best_model, pclass=3, sex=0, age=26)
```

Pour améliorer notre modèle KNeighborsClassifier, nous avons testé différents nombres de voisins (de 1 à 10) et avons constaté que le meilleur nombre de voisins est 1, avec un score de 0.8739, comme les résultats ci-dessous nous montrent :

```
Score avec 1 voisins : 0.8739495798319328
Score avec 2 voisins : 0.84593837535014
Score avec 3 voisins : 0.8627450980392157
Score avec 4 voisins : 0.8403361344537815
Score avec 5 voisins : 0.8417366946778712
Score avec 6 voisins : 0.8221288515406162
Score avec 7 voisins : 0.8207282913165266
Score avec 8 voisins : 0.7941176470588235
Score avec 9 voisins : 0.8095238095238095
Score avec 10 voisins : 0.8025210084033614
Meilleur nombre de voisins : 1 avec un score de 0.8739495798319328
```

Résultat des probabilités avec le meilleur nombre de voisins (k=1) :

```
Prédiction de survie : 0
Probabilité de non-survie : 1.0, Probabilité de survie : 0.0
```

Conclusion :

En conclusion, ce TP nous a permis de nous familiariser avec les concepts de régression linéaire et de classification à l'aide de l'algorithme K-Nearest Neighbors (KNN). Dans le premier exercice, nous avons appris à ajuster des modèles de régression linéaire à des données et à évaluer leur performance. Dans le second exercice, nous avons travaillé avec un jeu de données réel du Titanic pour prédire la survie des passagers en utilisant le modèle KNN. Nous avons également optimisé ce modèle en variant le nombre de voisins pour améliorer la précision des prédictions.