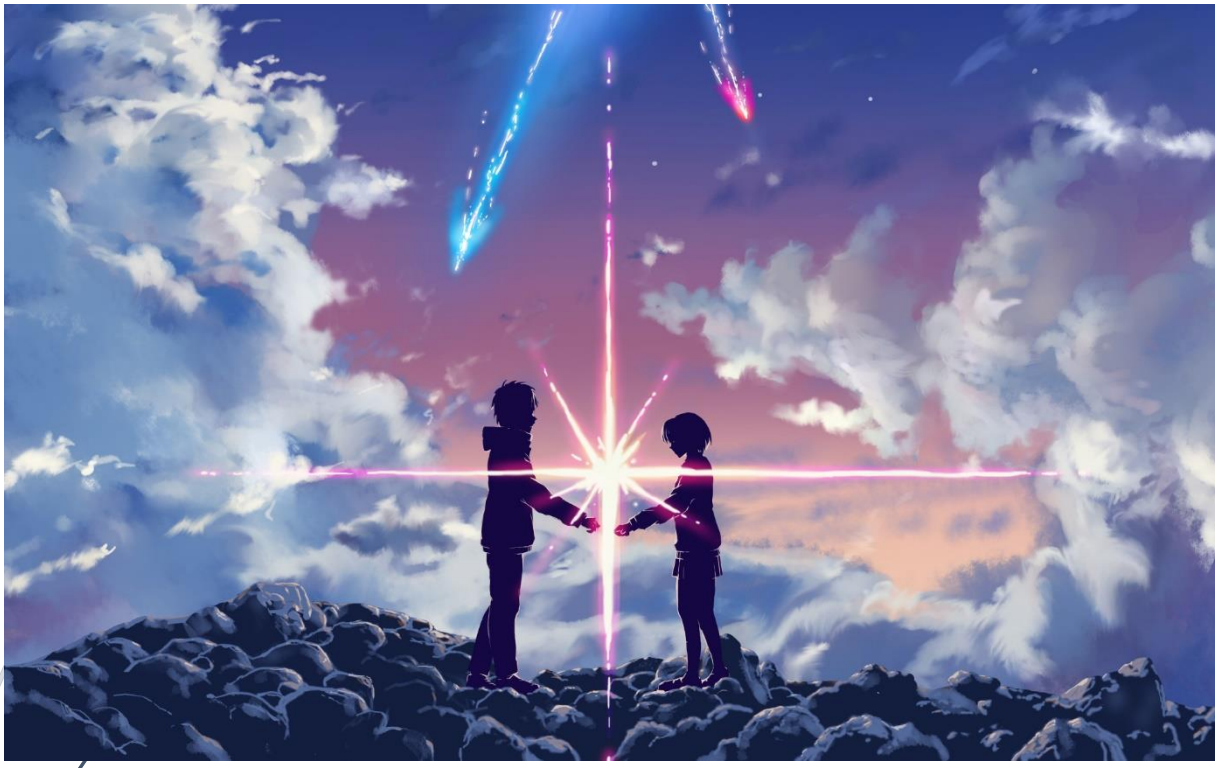
20/01/2023

Modélisation et Résolution de problèmes

Projet MyAnimeList



BOUBOU Jean-Philippe, HACQUARD Grégorie,
MASSET Eliot

IMT Mines Alès
École Mines-Télécom

Table des matières

I. Définition générale des tâches

Nous voulons, à partir d'un jeu de données, prédire le score de popularité qu'aura un anime en fonction de différentes caractéristiques telles que :

- Score
- Le nombre d'épisode
- Le genre
- La durée d'un épisode
- Le studio d'animation
- La source
- Le type d'anime

Ces caractéristiques nous servirons à chercher une corrélation entre certaines données disponibles avant la sortie d'un média d'animation, à sa note moyenne, donc son appréciation générale.

II. Analyse du jeu de données correspondant.

Nous avons récupéré le jeu de données suivant sur le site Kaggle à ce lien :

- anime-recommendation-database-2020

Ce jeu de données a été récupéré à partir des données présentes dans la base de données de MyAnimeList. C'est un site web et un réseau social de catalogue d'animes[1] et de mangas. Il permet à chaque utilisateur de l'application, de noter un anime/manga et de l'ajouter dans différentes catégories. Le site répertorie ainsi une base de données immense à l'origine de notre jeu de données.

Le jeu de données possède **X** entrées avec **Y** caractéristiques.

Les caractéristiques sont les suivantes :

- Identifiant (de 1 à nombre d'entrées)
- Nom
- Score
- Genres
- Nom anglais
- Nom Japonais
- Type
- Nombre d'épisodes
- Diffusé
- Date de la Première[2]
- Producteurs
- Licences
- Studios d'animation
- Source
- Durée
- Note moyenne attribuée par les membres
- Rang (de 1 à nombre d'entrées)
- Popularité
- Nombre de membres l'ayant noté
- Nombre de membres l'ayant mis en favori
- Nombre de membre l'ayant regardé
- Nombre de membre l'ayant regardé en entier
- Nombre de membre le regardant
- Nombre de membre l'ayant lâché en cours de route
- Nombre de membre souhaitant le regarder
- Score-10 (Nombre de membre ayant mis ce score)
- Score-9
- Score-8
- Score-7
- Score-6
- Score-5
- Score-4
- Score-3
- Score-2
- Score-1

1 : Tout type d'animation arborant les codes d'animation japonais, souvent adapté d'un manga.

2 : Correspond à la première date de diffusion

III. Définition du problème tel qu'un problème de « Machine Learning »

On peut donc voir que certaines de certaines de ces données ne sont pas pertinentes. En effet, Les score une grande partie de ces données, sont soit :

- Directement relié à la note (exemple : Nombre de note à 10)
- Définissable après la sortie dudit « animé » (exemple : Nombre de membres l'ayant noté)
- Peu pertinentes dans leur note moyenne (exemple : Nom Japonais)

On choisit alors de garder les caractéristiques suivantes :

- Score (donnée de sortie)
- Genres
- Type
- Nombre d'épisodes
- Studios d'animation
- Source

Nous avons donc extrait du jeu de données uniquement les données nous intéressants. Il s'agit donc d'un problème de prédiction de la note, basé sur des données d'entrée telles que les caractéristiques de l'anime, qui est un exemple typique de Machine Learning.

En utilisant les techniques de Machine Learning, nous allons entraîner un modèle en utilisant les caractéristiques sélectionnées comme données d'entrée et les notes comme données de sortie. Le modèle pourra alors utiliser cette relation pour prédire les notes des animes en utilisant les caractéristiques fournies. Il peut s'agir d'un problème de régression, car nous cherchons à prédire une note appartenant au domaine du continu.

Enfin, une fois le modèle entraîné, il pourra être utilisé pour prédire les notes des animes qui n'ont pas encore été notés, ce qui permettra potentiellement aux utilisateurs de mieux sélectionner les anime qu'ils vont regarder en fonction de leur préférence.

IV. Définition de la méthodologie et du protocole utilisé pour s'attaquer au problème

1. Recherche linéaire multiple

La régression linéaire est un choix populaire pour commencer à travailler sur un problème de prédiction. Elle suppose que la relation entre les variables d'entrée et la variable de sortie est linéaire, c'est-à-dire qu'elle peut être décrite par une fonction linéaire de la forme $Y = ax + b$.

La régression linéaire utilise une méthode appelée moindres carrés pour estimer les valeurs de a et b qui minimisent la somme des carrés des erreurs entre les valeurs prévues et les valeurs réelles.

En outre, il est facile de visualiser les résultats de la régression linéaire en traçant les données et la droite de régression. Cela permet de vérifier rapidement si la relation entre les variables d'entrée et la variable de sortie est réellement linéaire ou s'il est nécessaire d'utiliser des modèles plus complexes.

Pour résoudre ce problème nous avons donc commencé par réaliser une régression linéaire sur notre jeu de données, afin de définir un lien entre la moyenne des notes utilisateurs attribuées sur un « animé » et des données d'entrées.

Une fois notre modèle défini, nous avons entrainé notre modèle avec 80% de nos observations. Ainsi, une fois le modèle entrainé, nous avons pu le tester avec les 20% des observations restantes.

Pour le traitement de certaines colonnes, nous avons décidé d'opter pour l'utilisation du « One-Hot Encoding ». En effet, Des attributs tels que le Genre ou les studios d'animations sont des caractéristiques multi-catégorielles. Cela veut dire qu'une entrée peut avoir plusieurs catégories.

Par exemple, un anime peut être du genre « Action » et aussi « Humour ». Pour pallier ce problème, le « One-Hot Encoding » est parfaitement adapté comme le schéma ci-dessous l'explique :

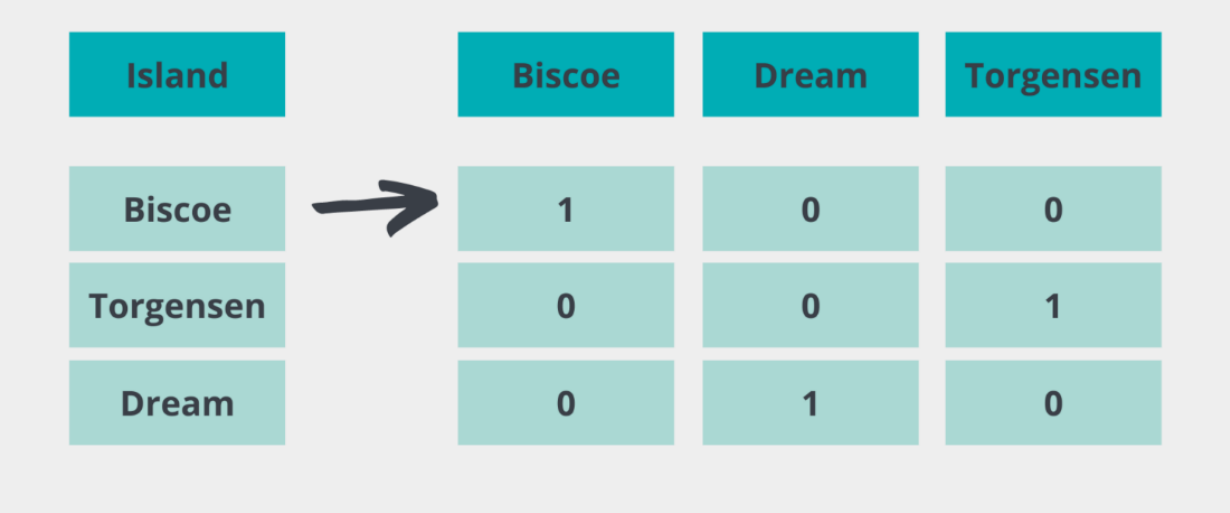


Figure 1 - Schémas explicatif du "One-Hot Encoding"

D’autres données comme le type et la source ne nécessiterons que la simple utilisation du « label Encoding ». Effectivement, ces données ne pouvant avoir qu’une seule catégorie, cette méthode est plus adaptée.

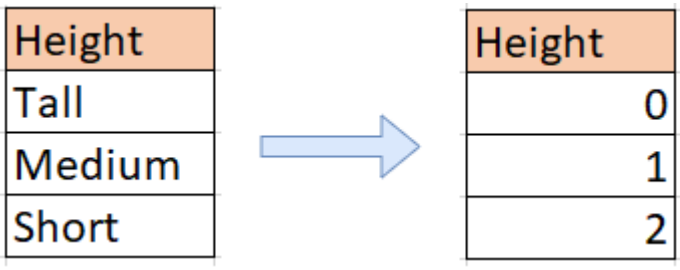


Figure 2 - Schémas explicatif du "Label Encoding"

Après avoir exécuter notre fichier linear.py (cf. Annexe 1), les résultats obtenus a été le suivant :

```
C:\Users\grego\Documents\Cours Mines\IA\testia>py reg.py
Scores de cross-validation:  [0.00819866 0.00608119 0.01353746 0.00495591 0.00720241]
Moyenne des scores:  0.0079951255968987
```

La validation croisée est une technique utilisée pour évaluer la performance d'un modèle de machine learning en utilisant une portion de données d'entraînement pour entraîner le modèle et une autre portion pour tester le modèle. Elle permet de mesurer l'erreur de généralisation, c'est-à-dire la capacité du modèle à généraliser à des données qu'il n'a pas vues lors de l'entraînement.

Ce résultat est très insuffisant. On peut en déduire que la corrélation entre nos entrées et la note moyenne de l’anime n’est pas démontrable avec une méthode linéaire. Nous allons donc tester avec d’autres méthodes, avec pour objectif de trouver des résultats plus satisfaisants.

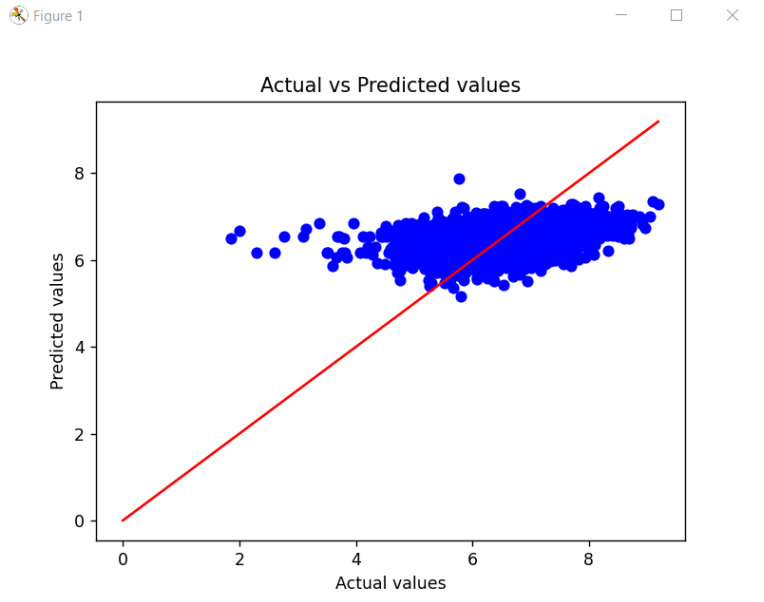


Figure 3 - Schémas représentatif des valeurs prédites et valeurs actuelles

Grâce au graphique que l'on à générer ci-dessus, on peut en déduire que le modèle prédit des données sur une plage de valeurs bien plus petite (de 5 à 7,5) que les valeurs actuelles (de 2 à 9,5). Cela est explicable par le fait que l'on a peu d'animé ayant une note en dessous de 5.

2. Recherche polynomiale

La régression polynomiale est un choix approprié si la régression linéaire échoue car elle permet de modéliser des relations non linéaires entre les variables d'entrée et la variable de sortie. Elle s'appuie sur le même principe que la régression linéaire, mais en utilisant des termes polynomiaux (c'est-à-dire des puissances) des variables d'entrée pour décrire la fonction de prédiction.

Il est important de noter que la régression polynomiale peut entraîner des problèmes tels que le surajustement (overfitting) si le degré du polynôme est trop élevé. C'est pourquoi nous avons commencé avec un degré relativement bas pour ensuite l'augmenter progressivement jusqu'à ce que des résultats satisfaisants soient obtenus.

Ainsi nous avons testé une recherche polynomiale (cf. Annexe 2) avec un degré de 2 :

```
C:\Users\grego\Documents\Cours Mines\IA\testia>py poly.py
Scores de cross-validation:  [0.02487109 0.03659455 0.0397641  0.02551553 0.03760344]
Moyenne des scores:  0.03286974235819991
```

Dans nos résultats ci-dessus, la moyenne des scores de cross-validation est de 0.03 environs. Ils sont près de 4 fois supérieurs à la régression linéaire. C'est encore insuffisant, mais encourageant.

Avec un degré de 3, nous avons un résultat clairement moins satisfaisant :

```
Scores de cross-validation:  [ 0.02829868  0.04571714  0.05510449  0.04553081 -0.12195809]
Moyenne des scores:  0.010538606067778367
```

Et pour le degré 4 :

```
Scores de cross-validation:  [-2.98067019  0.06713456  0.06703963  0.06581733  0.04803365]
Moyenne des scores:  -0.5465290028006125
```

Ainsi le degré 2 est le mieux entrainer, on peut ainsi en déduire que le modèle est trop entrainé avec un degré supérieur ou égale 3.

3. Recherche par arbre de décision

La régression par arbre de décision est un bon choix pour continuer la recherche si la régression linéaire et polynomiale sont infructueuses car elle permet de modéliser des relations non linéaires de manière plus flexible et plus robuste. Elle utilise une structure d'arbre pour découper les données en sous-ensembles plus petits en utilisant des conditions de test basées sur les variables d'entrée.

Chaque nœud de l'arbre représente une condition de test sur une des variables d'entrée, et chaque branche représente la réponse à cette condition (soit vraie ou fausse). Les feuilles de l'arbre représentent les valeurs prédites pour la variable de sortie.

Il est important de noter que les arbres de décision peuvent également entraîner des problèmes tels que le surajustement (overfitting) si l'arbre est trop profond et complexe.

Après avoir réalisé le script (cf. Annexe 3), l'exécution de celui-ci nous a fourni ces résultats :

```
C:\Users\grego\Documents\Cours Mines\IA\testia>py main.py
Scores de cross-validation:  [0.08063753 0.09424537 0.10099392 0.09814403 0.0739524 ]
Moyenne des scores:  0.08959464833139621
```

Cette méthode a tout de suite été plus fructueuse. Les résultats obtenus sont plus de 10 fois supérieurs à la régression linéaire, malheureusement ils sont encore loin d'être probants.

4. Modification des entrées

Si on a décidé de garder les valeurs comme « Unknown », c'est parce qu'on a conclu que ces valeurs sont cohérentes. En effet, Un studio d'animation Unknown à de forte chance d'être soit indépendant, soit très peu connus, et donc cela peut jouer sur le note moyenne.

Une solution testée a tout de même été d'enlever toutes les lignes d'entrée ayant des valeurs aberrantes, tels que « Unknown ». Malheureusement les résultats en sorties étaient un petit peu mieux mais pas satisfaisant.

Loin d'être satisfait, nous avons opté pour une autre solution : modifier nos méthodes de « mapping » sur nos données. Enfin un résultat un minimum intéressant est sortie. La solution a été de retirer le « One-Hot Encoder » des caractéristiques excepté du genre (cf. Annexe 4). Après cela, nous avons obtenu un meilleur résultat.

```
Modèle linéaire
Performance du modèle linéaire en apprentissage :  0.1592016698599188
Performance du modèle linéaire Entrainé :  0.16416217598778715

Modèle polynomial
score modèle polynomiale de degré 4: 0.20168949126475366
MSE : 0.6470407811224526
Mean Absolute Error:  0.6281158653930468

Arbre de décision
score arbre de décision: -80653.36561855316
MSE : 65371.38514366198
Mean Absolute Error:  241.24685714285712
```

Pour aller plus loin, nous avons essayé d'utiliser des entrées différentes, et après plusieurs tentatives, une caractéristique est sortie du lot : « le studio d'animation ». En effet, en utilisant seulement cette caractéristique dans nos recherches, les résultats obtenus étaient les même jusqu'à 10 chiffres après la virgule. Contrairement aux autres attributs qui isolés, nous donnait des résultats

très bas. On peut en déduire, que ce qui joue le plus sur l'appréciation d'un animé, est le studio chargé de l'animation.

V. Conclusion

Pour conclure sur nos résultats, il a été assez difficile de trouver des résultats satisfaisants. Cela peut être dû au fait que nos données ont probablement très peu de corrélation avec nos sorties. En fait plusieurs autres arguments pourraient influer sur la note, comme des données complexes, tels que les scénarii ou la lenteur d'une anime. En outre, il est important de rappeler que les données utilisées dans cette étude ne sont qu'une partie des nombreux facteurs qui peuvent influencer la popularité et la qualité d'un anime. Il y a de nombreux autres aspects tels que la promotion, le marketing, les critiques et les opinions des fans qui peuvent également jouer un rôle important. Malgré cela, nous avons tout de même attend des résultats démontrant un début de corrélation, faible mais existant.

Il est aussi plus sûr de faire confiance au studio d'animation dans le choix d'un anime, qu'au genre ou au type. Bien que ce soit très loin d'être absolue. Il est important de noter que ces résultats ne sont pas absolus et qu'il est toujours important de considérer d'autres facteurs avant de choisir un anime.

Enfin, il est important de noter que cette étude est un début de recherche et qu'il y a encore beaucoup de travail à faire pour comprendre les corrélations entre les différents facteurs qui influencent la popularité et la qualité des animes. Il serait donc intéressant de continuer à explorer ces questions à l'avenir pour une meilleure compréhension.

VI. Annexe

1. Annexe 1

reg.py

```
import numpy as np
import pandas as pd
import sys
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

df = pd.read_csv('anime.csv', delimiter=';')

ohe = OneHotEncoder(sparse_output=False)
le = LabelEncoder()
df['Type'] = le.fit_transform(df['Type'])
df['Source'] = le.fit_transform(df['Source'])
```

```
df['Genres']                                                                    =
ohe.fit_transform(df['Genres'].str.get_dummies(sep=',')).astype(int)
df['Studios']                                                                   =
ohe.fit_transform(df['Studios'].str.get_dummies(sep=',')).astype(int)

matrix = df.to_numpy()

Y = matrix[:,0]

#Séparer les données en données d'entraînement et de test
X_train,   X_test,   y_train,   y_test   =   train_test_split(matrix[:,1:],   Y,
test_size=0.2, random_state=42)

#Instancier un modèle de régression linéaire
reg = LinearRegression()

#Entraîner le modèle
reg.fit(X_train, y_train)

#Prédire les scores
y_pred = reg.predict(X_test)

y_pred_train = reg.predict(X_train)

#Evaluer la performance du modèle
print(r2_score(y_test, y_pred))
print(r2_score(y_train, y_pred_train))
```

2. Annexe 2

poly.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score
#Evaluer la performance du modèle
from sklearn.metrics import r2_score

df = pd.read_csv('anime2.csv', delimiter=';', encoding = "ISO-8859-1")

ohe = OneHotEncoder(sparse_output=False)
le = LabelEncoder()
df['Type'] = le.fit_transform(df['Type'])
```

```
df['Source'] = le.fit_transform(df['Source'])

df['Genres']                                                                =
ohe.fit_transform(df['Genres'].str.get_dummies(sep=',')).astype(int)
df['Studios']                                                               =
ohe.fit_transform(df['Studios'].str.get_dummies(sep=',')).astype(int)

df = df.sample(frac=1)
matrix = df.to_numpy()

Y = matrix[:,0]

#Séparer les données en données d'entraînement et de test
X_train,   X_test,   y_train,   y_test   =   train_test_split(matrix[:,1:],   Y,
test_size=0.2, random_state=42)

# Instancier un modèle de régression polynomiale
reg = PolynomialFeatures(degree=2)
X_poly = reg.fit_transform(X_train)
reg = LinearRegression().fit(X_poly, y_train)

# Effectuer la cross-validation en utilisant la métrique R²
scores = cross_val_score(reg, X_poly, y_train, cv=5, scoring='r2')

# Afficher les scores pour chaque itération de la cross-validation
print("Scores de cross-validation: ", scores)

# Afficher la moyenne des scores
print("Moyenne des scores: ", np.mean(scores)
```

3. Annexe 3

tree.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import cross_val_score

#Evaluer la performance du modèle
from sklearn.metrics import r2_score

df = pd.read_csv('anime_genre.csv', delimiter=';', encoding = "ISO-8859-1")

#shuffle df
```

```
ohe = OneHotEncoder(sparse_output=False)
le = LabelEncoder()
df['Type'] = le.fit_transform(df['Type'])
df['Source'] = le.fit_transform(df['Source'])

df['Genres']                                                    =
ohe.fit_transform(df['Genres'].str.get_dummies(sep=',')).astype(int)
df['Studios']                                                   =
ohe.fit_transform(df['Studios'].str.get_dummies(sep=',')).astype(int)

df = df.sample(frac=1)
matrix = df.to_numpy()
Y = matrix[:,0]

#Séparer les données en données d'entraînement et de test
X_train,   X_test,   y_train,   y_test   =   train_test_split(matrix[:,1:],   Y,
test_size=0.2, random_state=42)

three = DecisionTreeRegressor()

# Entraîner le modèle
three.fit(X_train, y_train)

y_pred = three.predict(X_test)
y_pred_train = three.predict(X_train)

scores = cross_val_score(three, X_train, y_train, cv=5, scoring='r2')
print("Scores de cross-validation: ", scores)
print("Moyenne des scores: ", np.mean(scores))
```

4. Annexe 4

mainWithMapping.py

```
import numpy as np
import pandas as pd
import json

from matplotlib import pyplot as plt
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, LabelEncoder,
PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv('anime2.csv', delimiter=';')

mappings = {}

mappings['Type'] = pd.factorize(df.loc[:, 'Type'])[1].tolist()
```

```
df.loc[:, 'Type'] = pd.factorize(df.loc[:, 'Type'])[0]

mappings['Studios'] = pd.factorize(df.loc[:, 'Studios'])[1].tolist()
df.loc[:, 'Studios'] = pd.factorize(df.loc[:, 'Studios'])[0]

mappings['Source'] = pd.factorize(df.loc[:, 'Source'])[1].tolist()
df.loc[:, 'Source'] = pd.factorize(df.loc[:, 'Source'])[0]

# Convert the dictionary to a JSON object
json_obj = json.dumps(mappings)

# Write the JSON object to a file
with open('mapping2.json', 'w') as f:
    f.write(json_obj)


ohe = OneHotEncoder(sparse_output=False)
le = LabelEncoder()
df['Genres'] =
ohe.fit_transform(df['Genres'].str.get_dummies(sep=',')).astype(int)

#genreDf = pd.DataFrame({'Genres': [["Action", "Adventure", "Avant Garde",
"Award Winning", "Boys Love", "Comedy", "Drama", "Fantasy", "Girls Love",
"Gourmet", "Horror", "Mystery", "Romance", "Sci-Fi", "Slice of Life",
"Sports", "Supernatural", "Suspense", "Ecchi", "Erotica", "Hentai", "Adult
Cast", "Anthropomorphic", "CGDCT", "Childcare", "Combat Sports",
"Crossdressing", "Delinquents", "Detective", "Educational", "Gag Humor",
"Gore", "Harem", "High Stakes Game", "Historical", "Idols (Female)", "Idols
(Male)", "Isekai", "Iyashikei", "Love Polygon", "Magical Sex Shift", "Mahou
Shoujo", "Martial Arts", "Mecha", "Medical", "Military", "Music",
"Mythology", "Organized Crime", "Otaku Culture", "Parody", "Performing
Arts", "Pets", "Psychological", "Racing", "Reincarnation", "Reverse Harem",
"Romantic Subtext", "Samurai", "School", "Showbiz", "Space", "Strategy
Game", "Super Power", "Survival", "Team Sports", "Time Travel", "Vampire",
"Video Game", "Visual Arts", "Workplace", "Demographics", "Josei", "Kids",
"Seinen", "Shoujo", "Shounen"]]})

column = ['Genres', 'Type', 'Episodes', 'Studios', 'Source']
Y = df['Score']
X = df[column]

#Séparer les données en données d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

#Instancier un modèle de régression linéaire
reg = LinearRegression()

#Entraîner le modèle
reg.fit(X_train, y_train)

#Prédire les scores
y_pred = reg.predict(X_test)

y_pred_train = reg.predict(X_train)

#Evaluer la performance du modèle
print("\nModèle linéaire")
print("Performance du modèle linéaire en apprentissage : ",
r2_score(y_train, y_pred_train))
print("Performance du modèle linéaire Entrainé : ", r2_score(y_test,
```

```
y_pred))

xline = np.linspace(0, max(y_test), 100)
yline = xline
plt.plot(xline, yline, color="red")
plt.scatter(y_test, y_pred, color="blue")
plt.xlabel("Actual values")
plt.ylabel("Predicted values")
plt.title("Actual vs Predicted values")
plt.show()
#Modèle polynomial
print("\nModèle polynomial")
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(X_train)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y_train)
X_test_poly = poly_reg.transform(X_test)
y_pred = lin_reg_2.predict(X_test_poly)
r2_score(y_test, y_pred)
mean_squared_error(y_test, y_pred)
print("score modèle polynomiale de degré 4:", r2_score(y_test, y_pred))
print("MSE :", mean_squared_error(y_test, y_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))



#Arbre de décision
print("\nArbre de décision")
# Initialiser l'arbre de décision
clf = DecisionTreeClassifier()
# Entraîner l'arbre de décision avec des données d'entraînement
label_encoder = preprocessing.LabelEncoder()
y_train = label_encoder.fit_transform(y_train)

clf = clf.fit(X_train, y_train)
# Utiliser l'arbre de décision pour prédire des résultats sur des données
de test
y_pred = clf.predict(X_test)
# Evaluer la performance de l'arbre de décision
print("score arbre de décision:", r2_score(y_test, y_pred))
print("MSE :", mean_squared_error(y_test, y_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred))
```