Machine Learning

Machine Learning c'est quoi ?

Le Machine Learning peut être défini comme étant une technologie d'intelligence artificielle permettant aux machines d'apprendre sans avoir été préalablement programmées spécifiquement à cet effet.

Le Machine Learning est explicitement lié au Big Data, étant donné que pour apprendre et se développer, les ordinateurs ont besoin de flux de données à analyser, sur lesquels s'entraîner. De ce fait, le Machine Learning, issu par essence du Big Data, a précisément besoin de ce dernier pour fonctionner. Le Machine Learning et le Big Data sont donc interdépendants.

Pour analyser de tels volumes de données, le Machine Learning se révèle bien plus efficace en termes de vitesse et de précision que les autres méthodologies traditionnelles.

À titre d'exemple, le Machine Learning est capable de déceler une fraude en une milliseconde, rien qu'en se basant sur des données issues d'une transaction (montant, localisation...), ainsi que sur d'autres informations historiques et sociales qui lui sont rattachées. En ce qui concerne l'analyse de données transactionnelles, de données issues de plateformes CRM ou bien des réseaux sociaux, là encore le Machine Learning se révèle désormais indispensable.

Suite des exercices de la doc de la Data Science avec vgsales.csv

Cette documentation décrit le processus de création d'un modèle de régression linéaire pour prédire les ventes globales de jeux vidéo en fonction des plateformes sur lesquelles ils sont disponibles.

Réviser et Préparer les Données :

1) Suppression des données

La suppression des colonnes non nécessaires est une étape cruciale dans la préparation des données pour le machine learning. Elle consiste à retirer de l'ensemble de données toutes les informations qui ne sont pas pertinentes ou qui pourraient fausser les prédictions.

Voici le code :

```
# Supprimez les colonnes textuelles non nécessaires et les colonnes de ventes régionales
columns_to_drop = ['Name', 'Rank', 'Genre', 'Publisher', 'JP_Sales', 'EU_Sales', 'NA_Sales', 'Other_Sales']
df = df.drop(columns=columns_to_drop, axis=1)
```

- **columns_to_drop** est une liste des noms de colonnes que nous voulons supprimer du DataFrame.
- **La méthode .drop()** est utilisée pour supprimer ces colonnes. L'argument **axis=1** indique que nous voulons supprimer des colonnes (par opposition à **axis=0**, qui supprimerait des lignes).

2) Traitement des Variables Catégorielles

Les variables catégorielles sont des variables qui représentent des types ou des catégories. Par exemple, la couleur d'une voiture (rouge, vert, bleu), le genre d'un livre (romance, thriller, science-fiction), ou les plateformes de jeu vidéo (PlayStation, Xbox, Nintendo) sont toutes des variables catégorielles. Ces variables peuvent être de type "nominal" (sans ordre inhérent) ou "ordinal" (avec un ordre inhérent, comme les notes 'faible', 'moyen', 'élevé').

En machine learning, la plupart des algorithmes s'attendent à des données d'entrée numériques, ce qui signifie que les variables catégorielles doivent être **converties en nombres** avant de pouvoir être utilisées pour entraîner un modèle. Le processus de conversion de variables catégorielles en variables numériques est connu sous le nom de "codage".

Le One-Hot Encoding est une méthode courante pour traiter les variables catégorielles. Elle consiste à créer une nouvelle colonne binaire pour chaque catégorie possible de la variable originale.

Voici le code :

```
# Appliquez One-Hot Encoding aux variables catégorielles restantes
df = pd.get_dummies(df, columns=['Platform'])
```

- **pd.get_dummies()** est une fonction qui réalise le One-Hot Encoding automatiquement pour les variables catégorielles.

- **columns=['Platform']** spécifie que nous voulons encoder la colonne **Platform**. Pour chaque plateforme unique, une nouvelle colonne est créée où la présence de cette plateforme est marquée par un **1** et l'absence par un **0**.

Séparation des features et de la cible :

Dans le contexte du machine learning, la séparation des features et de la cible est un processus fondamental pour préparer les données avant d'entraîner un modèle.

3) Features (X)

Les features sont les variables qui servent à prédire la variable cible. Dans le dataset, elles représentent toutes les informations disponibles sur chaque jeu vidéo qui pourraient potentiellement influencer les ventes globales, à l'exception des ventes elles-mêmes.

En termes de données, les features se composent de tout ce qui est contenu dans le DataFrame après avoir retiré la colonne cible (Global_Sales). Cela comprend les indicateurs de plateforme générés par le One-Hot Encoding ainsi que toute autre mesure quantitative ou catégorielle qui a été convertie en format numérique et qui est jugée utile pour le modèle.

4) Cible (Y)

La cible, également connue sous le nom de variable dépendante, est la quantité que je cherche à prédire. Dans ce cas, c'est la colonne Global_Sales, qui représente le nombre total d'unités vendues à travers le monde pour chaque jeu vidéo.

La cible est isolée des autres données car elle est ce que le modèle essaiera de prédire.

Pourquoi séparer les données ?

La séparation est cruciale car elle permet de structurer les données de manière à ce que le modèle de machine learning puisse clairement distinguer entre les entrées (features) et la sortie (cible) lors de l'apprentissage.

En fournissant les features au modèle sans variable cible pendant l'entraînement (apprentissage), le modèle peut apprendre à associer les patterns dans les features à la cible correspondante.

Lors de la prédiction, je fourni de nouvelles données de features au modèle, et en utilisant ce qu'il a appris, il prédit la valeur de la cible.

Voici le code :

```
# Séparation des features et de la cible
X = df.drop('Global_Sales', axis=1)
y = df['Global_Sales']
```

- **X** est un DataFrame qui contient toutes les colonnes sauf la cible que nous essayons de prédire.
- **y** est une série qui contient les valeurs de la variable cible, Global_Sales, que nous allons essayer de prédire avec notre modèle.

Division en Ensembles d'Entraînement et de Test :

La division en ensembles d'entraînement et de test est une étape critique dans le processus de développement d'un modèle de machine learning. Cette pratique permet de tester la performance du modèle sur des données qu'il n'a pas vues pendant la phase d'entraînement, ce qui fournit une estimation plus fiable de sa performance en situation réelle.

Voici le code :

```
# Division en ensembles 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)
```

train_test_split est une fonction de la bibliothèque scikit-learn qui permet de diviser de manière aléatoire les données en deux groupes : un pour l'entraînement du modèle et l'autre pour le tester.

La division aléatoire est importante car elle aide à garantir que les deux ensembles sont représentatifs de l'ensemble des données.

L'ensemble d'entraînement **(X_train, y_train)** est utilisé par le modèle pour apprendre. Il contient les features **(X_train)** et les cibles correspondantes **(y_train)**.

Le modèle ajuste ses paramètres pour minimiser l'erreur entre ses prédictions et les cibles réelles dans cet ensemble.

L'ensemble de test **(X_test, y_test)** sert à évaluer le modèle. Il est gardé séparé de l'ensemble d'entraînement et n'est pas utilisé pendant l'apprentissage.

Après l'entraînement, le modèle fait des prédictions sur cet ensemble. La comparaison de ces prédictions avec les valeurs réelles **(y_test)** fournit une mesure de la performance du modèle.

test_size=0.2 indique que **20%** des données totales doivent être réservées pour l'ensemble de test. Le reste, soit **80%**, constitue l'ensemble d'entraînement.

Cela me permet de conserver la majorité des données pour l'entraînement tout en ayant suffisamment de données pour tester le modèle de manière significative.

random_state=42 est un paramètre qui contrôle la randomisation de la division. En utilisant un nombre fixe (comme 42), je garantis que le processus de division est reproductible.

Cela signifie que si moi ou quelqu'un d'autre exécute le code avec le même état aléatoire sur les mêmes données, j'obtiendrai exactement la même division, ce qui est essentiel pour la validation et la comparaison des modèles.

Gestion des Valeurs Manquantes :

La gestion des valeurs manquantes est une étape essentielle dans le prétraitement des données pour le machine learning. Les valeurs manquantes peuvent fausser les résultats d'un modèle ou même empêcher son entraînement.

Voici le code :

```
# Gestion des valeurs manquantes avec imputation par la moyenne
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

SimpleImputer est une classe de la bibliothèque scikit-learn utilisée pour l'imputation des valeurs manquantes dans les ensembles de données.

L'imputation est le processus de remplacement des valeurs manquantes par d'autres valeurs. Elle est essentielle car de nombreux algorithmes de machine learning ne peuvent pas gérer directement les valeurs manquantes.

missing_values=np.nan : Cette option indique que les valeurs manquantes dans les données sont représentées par NaN (Not a Number). C'est une convention standard pour marquer les valeurs manquantes en Python.

strategy='mean' : Cette stratégie d'imputation remplace chaque valeur manquante dans une colonne par la moyenne de toutes les valeurs non manquantes dans cette colonne. C'est une méthode couramment utilisée car elle préserve la distribution générale des valeurs de la colonne.

imputer.fit_transform(X_train) : Cette ligne effectue deux opérations sur l'ensemble d'entraînement **(X_train)**. Tout d'abord, **fit** calcule les moyennes de chaque colonne. Ensuite, **transform** remplace les valeurs manquantes dans chaque colonne par la moyenne correspondante.

imputer.transform(X_test) : Cette ligne utilise les moyennes calculées à partir de l'ensemble d'entraînement pour remplacer les valeurs manquantes dans l'ensemble de test **(X_test)**. Il est crucial d'utiliser les mêmes statistiques pour les ensembles d'entraînement et de test afin de garantir la cohérence et d'éviter les fuites de données.

Après cette étape, les ensembles **X_train** et **X_test** ne contiennent plus de valeurs manquantes, ce qui permet au modèle de machine learning de fonctionner correctement sans erreurs dues à des données manquantes.

Entraînement du Modèle :

Voici le code :

```
# Entraînement du modèle
model = LinearRegression()
model.fit(X_train, y_train)
```

model = LinearRegression() : Cette ligne crée une instance du modèle de régression linéaire.

LinearRegression est une classe de la bibliothèque scikit-learn qui implémente un modèle de régression linéaire.

La régression linéaire est une méthode statistique qui vise à modéliser la relation entre une variable dépendante (cible) et une ou plusieurs variables indépendantes (features) par une fonction linéaire.

model.fit(X_train, y_train) : Cette ligne entraîne le modèle sur l'ensemble d'entraînement.

.fit() est une méthode qui applique l'algorithme de régression linéaire aux données d'entraînement.

X_train contient les features d'entraînement, tandis que **y_train** contient les valeurs correspondantes de la variable cible.

Pendant l'entraînement, l'algorithme cherche à trouver les meilleurs coefficients pour chaque feature qui minimisent l'erreur entre les valeurs prédites et les valeurs réelles de la variable cible dans l'ensemble d'entraînement.

Le but de cette étape est de permettre au modèle d'apprendre suffisamment des données d'entraînement pour pouvoir faire des prédictions précises sur de nouvelles données non vues.

Le modèle essaie de comprendre comment chaque feature influence la variable cible (ici, les ventes globales de jeux vidéo) et de quantifier cette influence sous forme de coefficients.

Une fois l'entraînement terminé, le modèle est prêt à être utilisé pour prédire la variable cible pour de nouvelles observations.

Évaluation du Modèle :

L'évaluation du modèle donne une compréhension de la performance du modèle dans des conditions réelles, c'est-à-dire lorsqu'il est confronté à de nouvelles données. C'est une étape essentielle pour valider la fiabilité et la validité du modèle avant de l'utiliser pour des applications.

Voici le code :

```
# Évaluation du modèle
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"MSE: {mse}")
```

y_pred = model.predict(X_test) : Cette ligne utilise la méthode **.predict()** du modèle entraîné pour faire des prédictions sur l'ensemble de test **(X_test)**.

Le modèle utilise les coefficients et l'interception qu'il a appris pendant la phase d'entraînement pour calculer les valeurs prédites pour chaque observation dans l'ensemble de test.

y_pred contient les prédictions du modèle, c'est-à-dire les ventes globales estimées pour chaque jeu vidéo dans l'ensemble de test.

mse = mean_squared_error(y_test, y_pred) : Cette ligne calcule le Mean Squared Error (MSE) entre les valeurs prédites **(y_pred)** et les valeurs réelles **(y_test)**.

Le MSE est une mesure courante pour évaluer la performance des modèles de régression. Un MSE plus faible indique que les prédictions du modèle sont proches des valeurs réelles.

Un MSE faible suggère que le modèle a une bonne performance, car les erreurs entre les prédictions et les valeurs réelles sont en moyenne petites.

Un MSE élevé indique que le modèle fait des erreurs plus importantes dans ses prédictions, ce qui peut être dû à un manque de généralisation, à un surajustement sur les données d'entraînement, ou simplement à la complexité du problème de prédiction.

print(f"MSE: {mse}") affiche le MSE calculé, fournissant une indication claire de la qualité des prédictions du modèle.

Voici mon MSE :

```
MSE: 4.147919180685654
```

Il est faible ce qui indique que le modèle à une bonne performance.

Analyse des Coefficients :

L'analyse des coefficients dans un modèle de régression linéaire est une étape importante pour comprendre comment chaque feature influence la variable cible. Dans ce cas, cette étape vise à déterminer l'impact de chaque plateforme de jeu sur les ventes globales.

Voici le code :

```python
# Analyse des coefficients
coefficients = pd.DataFrame(model.coef_, X_train.columns, columns=['Coefficient'])
coefficients = coefficients.sort_values(by='Coefficient', key=abs, ascending=False)
print(coefficients)
```

model.coef_ contient les coefficients estimés pour chaque feature par le modèle de régression linéaire.

Chaque coefficient représente l'effet attendu sur la variable cible (ici, les ventes globales) d'une augmentation d'une unité de la feature correspondante, tout en gardant les autres features constantes.

pd.DataFrame(model.coef_, X_train.columns, columns=['Coefficient']) crée un DataFrame pandas à partir des coefficients.

coefficients.sort_values(by='Coefficient', key=abs, ascending=False) trie les coefficients par leur valeur absolue en ordre décroissant.

Cette opération permet d'identifier rapidement quelles features ont le plus grand impact sur la variable cible, indépendamment de la direction de cet impact (positif ou négatif).

Un coefficient **positif** indique qu'une augmentation de cette feature est associée à une augmentation des ventes globales.

Un coefficient **négatif** indique qu'une augmentation de cette feature est associée à une diminution des ventes globales.

La magnitude du coefficient (sa valeur absolue) indique la force de l'association : plus le coefficient est élevé, plus l'impact est important.

print(coefficients) affiche les coefficients triés, permettant une analyse visuelle rapide de l'importance relative des différentes features.

Cette analyse des coefficients est essentielle pour comprendre non seulement les facteurs qui influencent le plus les ventes globales, mais aussi la nature de cette influence (positive ou négative). Elle fournit des insights précieux sur les données et le modèle, qui peuvent être utilisés pour informer les décisions commerciales ou stratégiques, comme se concentrer sur certaines plateformes de jeu.

Le résultat :

	Coefficient
Platform_GB	2.114890e+00
Platform_NES	1.402166e+00
Platform_GG	-1.023360e+00
Platform_SCD	-8.966656e-01
Platform_PCFX	-8.479715e-01
Platform_TG16	-8.443186e-01
Platform_NG	-8.321268e-01
Platform_PS4	8.296559e-01
Platform_SAT	-6.850314e-01
Platform_XOne	6.258054e-01
Platform_WiiU	5.878336e-01
Platform_2600	-5.813562e-01
Platform_PS3	5.519131e-01
Platform_X360	5.028717e-01
Platform_WS	-4.553139e-01
Platform_3DS	4.268466e-01
Platform_DC	-4.070430e-01
Platform_Wii	3.039099e-01
Platform_XB	-2.115822e-01
Platform_N64	-1.782798e-01
Platform_GC	-1.631287e-01
Platform_PS	-1.286336e-01
Platform_GBA	-1.271984e-01

Conclusion :

L'analyse des coefficients du modèle de régression linéaire révèle des informations significatives sur l'impact des différentes plateformes de jeux vidéo sur les ventes globales. D'après le modèle, certaines plateformes se démarquent par leur association positive ou négative avec les ventes globales.

La plateforme **NES** montre le coefficient le plus élevé positivement, suggérant que les jeux sortis sur cette plateforme ont tendance à avoir des ventes globales significativement plus élevées, avec un impact estimé à environ 1.40 unités de vente supplémentaires pour chaque jeu. Ceci pourrait refléter une tendance rétro, où les jeux **NES** bénéficient d'une popularité et d'une nostalgie qui boostent leurs ventes.

Inversement, la plateforme **GG (Game Gear)** présente un coefficient fortement négatif, indiquant que les jeux associés à cette plateforme ont tendance à voir leurs ventes globales diminuer d'environ 1.02 unités de vente pour chaque jeu. Cela pourrait refléter une désaffection ou une obsolescence du support de jeu.

Il est intéressant de noter que la plupart des plateformes modernes comme **PS4** et **XOne** ont des coefficients positifs, bien que plus modestes, ce qui suggère qu'elles contribuent favorablement aux ventes globales, mais pas aussi fortement que la plateforme **NES** historiquement populaire.

Cependant, il est important de noter que ces conclusions sont basées sur les données historiques et les tendances du marché au moment de la collecte des données (la base de données s'arrête à 2020). Les préférences des consommateurs et les dynamiques du marché évoluent constamment.

Ressources

La chaîne youtube de Machine Learnia : FORMATION MACHINE LEARNING (2019) - ML#1

Introduction Au machine Learning : Introduction au Machine Learning

Régression linéaire : Prédire les prix des maisons à Boston en utilisant Python et la r...

MSE : MÉTRIQUES de RÉGRESSIONS en DATA SCIENCE (Coefficient de Déterminati...

FAQ