ALLARD Fabien et HMADOUCH Anass

Projet ML&DL - M1 :

Analyse des commentaires de jeux vidéo sur Metacritic

## 1. Présentation du Dataset

Intérêt personnel concernant la thématique choisie

En tant qu'étudiants en Data Science mais aussi grands amateurs de jeux vidéo, nous avons voulu mêler passion et travail. Les notes et les commentaires des utilisateurs influencent grandement nos achats et ceux de nombreux joueurs. Par exemple, qui n'a jamais hésité à acheter un jeu après avoir lu des critiques négatives ? Comprendre les sentiments cachés derrière ces évaluations pourrait aider à déchiffrer ces tendances et même prévoir comment un jeu sera reçu. Pour ce projet, nous allons développer un modèle capable de prédire si un commentaire est positif, neutre ou négatif.

Contenu

Le dataset a été sélectionné sur la plateforme Kaggle, une référence pour l'accès à des données variées. Il est issu de Metacritic, un site populaire regroupant des critiques et commentaires de jeux vidéo. Ce dataset satisfait les exigences du projet avec une colonne contenant la labellisation des données (« Userscore »).

Deux colonnes principales :

- Userscore : Une note attribuée par l'utilisateur (de 0 à 10).
- Comment : Un texte écrit par l'utilisateur, expliquant ou non la note attribuée.

Nous avons choisi ce dataset car il est riche et varié : on y trouve des jeux de différents genres, des critiques passionnées, des joueurs exigeants… tout cela permet de construire un projet stimulant.

**Contexte du Dataset :**

Date de parution

Bien que la date exacte de création du dataset ne soit pas connue, les données semblent récente (2023), ce qui en fait un sujet d'étude pertinent pour comprendre les comportements récents.

Nombres de lignes de données

Le dataset initial contient près de 200 000 lignes, mais après un nettoyage rigoureux et un équilibrage des classes (pour éviter le biais), nous avons travaillé sur un sous-ensemble de 45 000 lignes, suffisant pour garantir des résultats significatifs.

Contexte d'utilisation

Ce travail pourrait bénéficier aux éditeurs de jeux vidéo en leur permettant d'anticiper les attentes des joueurs. Par exemple, une analyse approfondie pourrait aider à identifier les éléments qui déplaisent ou plaisent le plus.

Exploitation des résultats

Les résultats pourraient être utilisés dans des outils d'analyse pour détecter les tendances, les avis négatifs émergents, voire pour créer des recommandations plus personnalisées pour les joueurs.

- Identifier les points faibles et forts des jeux pour les studios.
- Améliorer les recommandations personnalisées pour les joueurs.
- Suivre l'évolution de la perception d'un jeu au fil du temps.

## 2. Nettoyage des Données  et Préparation des Données

Formatage

Nous avons commencé par importer les données dans un DataFrame Pandas, avec une indexation correcte et des en-têtes clairs. Les colonnes inutiles ont été supprimées pour simplifier l'analyse.

Suppression des valeurs manquantes et extrêmes

Un premier examen a révélé des lignes contenant des valeurs manquantes, qui ont été retirées. Les notes allant de 0 à 10 n'ont pas présenté d'erreurs.

Catégorisation des sentiments

Les scores ont été transformés en trois classes :

- 0 : Négatif (Userscore ≤ 3).
- 1 : Neutre (4 ≤ Userscore ≤ 5).
- 2 : Positif (Userscore ≥ 6).
-

Visualisation

Nous avons observé une surreprésentation des scores positifs. Pour rééquilibrer, nous avons rééchantillonné les données. En autre, Sachant qu'au maximum ~15 000 lignes avec comme sentiment  « neutre », nous avons donc décidé d'équilibrer le dataset en conséquence en limitant les autres sentiments à 15 000 lignes, pour un total 45 000 lignes.

Normalisation

Pour les commentaires, un prétraitement complet a été effectué :

- Suppression de la ponctuation et des caractères spéciaux.
- Conversion en minuscule.
- Tokenization, suppression des mots vides (« the », « and », etc.) et lemmatisation.

Statistiques essentielles

```
Moyenne des scores : 7.62
Écart-type : 3.13
Médiane : 9.00
```

## 3. Modélisation Machine Learning

Choix du Modèle

Nous avons utilisé un Random Forest Classifier, adapté pour des problèmes de classification avec des données hétérogènes comme ici. Les commentaires ont été transformés en vecteurs grâce à TF-IDF.

Pourquoi utiliser TF-IDF ?

Le TF-IDF (Term Frequency-Inverse Document Frequency) est une méthode mathématique qui permet de transformer un texte en vecteurs numériques pour son traitement par des modèles de Machine Learning. Elle est particulièrement adaptée pour représenter des données textuelles tout en pondérant l'importance des mots en fonction de leur fréquence.

Cela se traduit par :

TF-IDF(t,d) = TF(t,d) × IDF(t)

TF (Term Frequency) : Fréquence d'un mot dans le document.

TF(t,d) = Nombre d'occurrences de t dans d / Nombre total de mots dans d

« t » étant un terme (mot) dans un document (commentaire)

« d » étant un document (commentaire dans notre cas)

IDF (Inverse Document Frequency) : Pondération de l'importance globale du mot.

IDF(t) = log ( N / Nombre de documents contenant t )

N étant le nombre total de documents.

L'équation suivante décrit les opérations que Scikit-Learn applique pour calculer les valeurs d'idf :

$$idf_i = ln[\,(N+1)/\,df_i] + 1$$

Une fois idfi calculé, tf-idfi est tfi multiplié par idfi.

$$tf\text{-}idf_i = tf_i \times idf_i$$

Entraînement

- Division des données en ensembles d'entraînement (80%) et de test (20%).
    - Vectorisation des textes avec TF-IDF (5 000 mots), le résultat est une matrice de taille m×n.
        - m est le nombre de documents (lignes dans le dataset).
        - n est le nombre de mots sélectionnés dans le vocabulaire (ici, n = 5000)
- Optimisation avec 100 arbres. Le modèle Random Forest construit un ensemble de 100 arbres de décision.

Résultat

```
Rapport de classification (ML) :
              precision    recall  f1-score   support

           0       0.75      0.75      0.75      3006
           1       0.72      0.72      0.72      2988
           2       0.79      0.79      0.79      3006

    accuracy                           0.75      9000
   macro avg       0.75      0.75      0.75      9000
weighted avg       0.75      0.75      0.75      9000
```

## 4. Modélisation Deep Learning

Choix d'architecture

Le modèle de réseau neuronal était simple mais efficace :

- Une couche dense (128 neurones, ReLU).
    - Dense signifie que chaque neurone de la couche est connecté à tous les neurones de la couche précédente.
    - ReLU (Rectified Linear Unit) est utilisée comme fonction d'activation ($f(z)=max(0,z)$).
- Une couche cachée (64 neurones, ReLU).
- Une sortie (3 neurones, softmax).
    - Chaque neurone correspond à une classe (négatif, neutre, positif).
    - La fonction softmax transforme les sorties en probabilités, ce qui permet une meilleur interprétation des prédiction.
- L'optimisation est réalisée avec l'algorithme Adam, une version améliorée de la descente de gradient, qui ajuste les poids pour minimiser la perte.
- La perte utilisée est la « Sparse Categorical Crossentropy », qui mesure la différence entre les prédictions du modèle et les labels réels.

Optimisation

L'algorithme Adam ($\alpha=0.001$) a été utilisé avec la fonction de perte sparse_categorical_crossentropy. Le dropout a été fixé à 0.5 pour limiter le surapprentissage.

Résultat

Rapport de classification (DL) :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.75 | 0.74 | 0.74 | 3006 |
| 1 | 0.69 | 0.75 | 0.72 | 2988 |
| 2 | 0.85 | 0.79 | 0.82 | 3006 |
| accuracy | | | 0.76 | 9000 |
| macro avg | 0.76 | 0.76 | 0.76 | 9000 |
| weighted avg | 0.76 | 0.76 | 0.76 | 9000 |

## 5. Analyse et Conclusion

Conclusion

Ce projet a été enrichissant et a montré que les avis utilisateurs sont prédictibles avec une bonne précision. Bien que le ML ait été plus rapide, le DL pourrait être amélioré pour des données plus complexes.

### L'Importance de l'Équilibrage des Données

Pour cette partie, nous avons voulu démontrer l'impact de l'équilibrage des données sur la performance des modèles de Machine Learning. Le modèle Random Forest a été utilisé sur deux versions du dataset : une version non équilibrée où les classes sont déséquilibrées, et une version équilibrée avec 15 000 exemples par classe

```python
initial_distribution = filter_data['sentiment'].value_counts(normalize=True)

balanced_distribution = balanced_data['sentiment'].value_counts(normalize=True)

print("Distribution initiale des classes (en %):")
print((initial_distribution * 100).round(2))
print("\nDistribution après équilibrage (en %):")
print((balanced_distribution * 100).round(2))
```

✓ 0.0s

```
Distribution initiale des classes (en %):
sentiment
2    79.36
0    14.27
1     6.36
Name: proportion, dtype: float64

Distribution après équilibrage (en %):
sentiment
0    33.33
1    33.33
2    33.33
Name: proportion, dtype: float64
```

Version sans balance des données :

```
Rapport de classification (ML) :
              precision    recall  f1-score   support

           0       0.79      0.41      0.54      8200
           1       0.77      0.02      0.04      3636
           3       0.85      0.99      0.91     44956

    accuracy                           0.84     56792
   macro avg       0.80      0.47      0.50     56792
weighted avg       0.83      0.84      0.80     56792
```

Le modèle semble bien prédire la classe positive, mais au détriment des classes négative et neutre. La classe neutre est particulièrement mal prédite avec un rappel très faible (0.02), ce qui signifie que le modèle ignore presque complètement cette catégorie.

Version avec balance des données :

```
Rapport de classification (ML) :
              precision    recall  f1-score   support

           0       0.75      0.76      0.76      3006
           1       0.73      0.73      0.73      2988
           3       0.80      0.79      0.79      3006

    accuracy                           0.76      9000
   macro avg       0.76      0.76      0.76      9000
weighted avg       0.76      0.76      0.76      9000
```

Avec un dataset équilibré, le modèle parvient à mieux prédire les classes négative et neutre, tout en maintenant une performance raisonnable pour la classe positive. Bien que l'accuracy globale soit plus faible (à cause d'un meilleur traitement des classes minoritaires), le modèle est plus équitable et fiable.

| Critère | Modèle Non Équilibré | Modèle Équilibré |
|---|---|---|
| **Accuracy** | **84%** | **76%** |
| **F1-Score Classe 0** | **0.54** | **0.76** |
| **F1-Score Classe 1** | **0.04** | **0.73** |
| **F1-Score Classe 3** | **0.91** | **0.79** |

- La classe neutre (1) a montré une très grande amélioration du F1-score (de 0.04 à 0.73).
- La classe négative (0) a vu une progression significative (F1-score de 0.54 à 0.76).
- La classe positive (3) a subi une légère baisse, mais reste très performante (de 0.91 à 0.79).

En conclusion, cela met en évidence l'importance de l'équilibrage des données dans les problèmes de classification. Un dataset déséquilibré peut biaiser le modèle en faveur des classes majoritaires, ce qui le rend peu fiable pour les classes minoritaires. En revanche, l'équilibrage permet d'améliorer la performance globale tout en assurant une prédiction plus équitable entre les classes.

**Analyse des Résultats : Modèle Deep Learning**

Changement des Labels

Lors de l'entraînement du modèle Deep Learning, une erreur a été remonté en raison du choix initial des labels : 0 (négatif), 1 (neutre) et 3 (positif). La fonction de perte utilisée, sparse_categorical_crossentropy, exige que les labels soient numérotés de manière continue à partir de 0, c'est-à-dire 0, 1, 2 pour un problème à trois classes. Par conséquent, un ajustement a été effectué pour convertir le label 3 (positif) en 2 (positif).

Ce changement est purement technique et n'affecte en rien les résultats des modèles ou leur interprétation. Les prédictions finales restent cohérentes, et les performances des modèles sont calculées de la même manière avec les nouvelles valeurs des labels.

Les résultats obtenus pour le modèle Deep Learning montrent une performance globale solide avec une accuracy de 76%, ce qui reflète un modèle bien équilibré dans la gestion des trois classes (négatif, neutre et positif). Voici une analyse détaillée des performances :

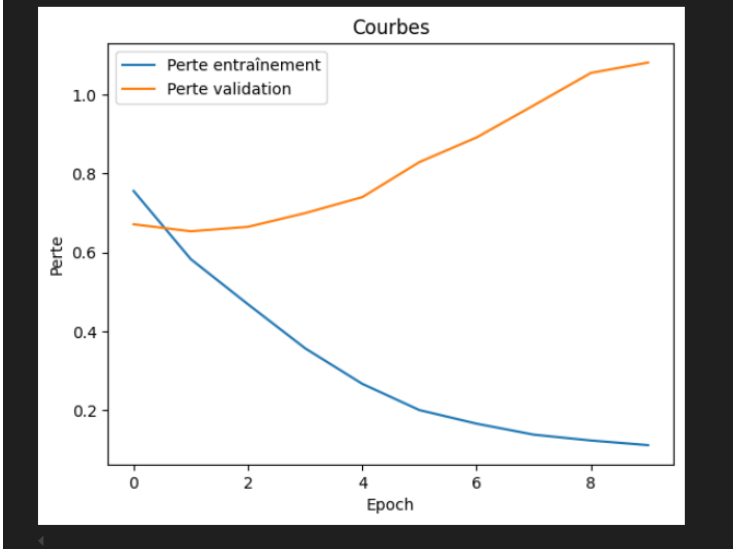
Rapport de classification (DL) :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.75 | 0.74 | 0.74 | 3006 |
| 1 | 0.69 | 0.75 | 0.72 | 2988 |
| 2 | 0.85 | 0.79 | 0.82 | 3006 |
| accuracy | | | 0.76 | 9000 |
| macro avg | 0.76 | 0.76 | 0.76 | 9000 |
| weighted avg | 0.76 | 0.76 | 0.76 | 9000 |

Autres analyses :

```python
# Analyse de perte
plt.plot(history.history['loss'], label='Perte entraînement')
plt.plot(history.history['val_loss'], label='Perte validation')
plt.title("Courbes")
plt.xlabel("Epoch")
plt.ylabel("Perte")
plt.legend()
plt.show()
```

✓ 0.1s




Observations

- Perte d'entraînement :
  - La perte diminue rapidement au fil des époques, ce qui est attendu. Cela indique que le modèle apprend bien sur les données d'entraînement.
- Perte de validation :
  - Contrairement à la perte d'entraînement, la perte de validation commence à augmenter après quelques époques. Cela suggère que nôtre modèle commence à surapprendre (overfitting) sur les données d'entraînement.
- Écart entre les deux courbes :
  - L'écart significatif entre les pertes d'entraînement et de validation signifie qu'il y a overfitting.

L'une des solutions possible pour réduire le cas d'un overfitting, est l'augmentation de dropout, la reduction du nombre d'epochs comme par exemple dans notre cas, le réduire à 5-6 epochs ou encore l'augmentation des données.