

Da Silva Jean-Charles

Mebarki Seif Eddine

2025

### Fouille de Textes

#### Analyse de la notation des jeux-vidéo.

#### Conditions d'analyse :

Pour cette analyse, nous utilisons **Steam** et **Metacritic**, le premier étant le site de référence concernant les jeux-vidéo sur PC et le second, le site de référence agglomérant de nombreuses notations de sites différents pour le jeu-vidéo. Comme Steam, ne nous permet pas de classer les jeux par notation, on a décidé d'utiliser le système de notation de Metacritic (de 0 à 100) afin de sélectionner les jeux. Puis, rechercher les reviews des jeux sélectionnés sur Steam.

Evidemment, cette méthode a des défauts car tous les jeux ne sont pas disponibles sur Steam (même si Steam est la plateforme la plus populaire). Il existe de multiples plateformes comme Epic Game Store, GoG, etc. Obligatoirement, cette méthode retire également de notre analyse, tous les jeux exclusifs aux consoles (PlayStation, Nintendo, Xbox et autres).

Toutes les compilations, rééditions sont exclus également, pour la variété de notre analyse.

### Tailles et Requis :

Le dossier pèse 5,90 Mo décompressé et 5,66 Mo en format ZIP.

Modules requis pour les scripts Python :

- **PySide6**
- **Requests**
- **Pandas**
- **Matplotlib**
- **Scipy**
- **Langdetect**
- **Textblob**
- **Textblob\_fr**
- **Nltk**
- **vaderSentiment**
- **scikit-learn**
- **seaborn**
- **numpy**

Le script **requirements.py** permet de vérifier l'environnement python afin de savoir si les modules sont déjà présents sur la machine. Si ce n'est pas le cas, il les installe automatiquement.

Le dossier d'entrée demandé dans la grande majorité des scripts est le dossier racine du Projet. (**Projet\_Steam\_Reviews**). Le dossier de sortie est au choix de l'utilisateur.

### Préparation :

Autant l'obtention des notes Metacritic était particulièrement simple, autant la récupération était plus fastidieuse mais pas pour autant difficile. Pour obtenir les reviews Steam, on a créé et utilisé un script Python (**SteamReviewDownloader.py**) avec l'aide de l'API Steam. On a utilisé **PySide6** pour créer une interface graphique afin de ne pas passer sur le terminal à chaque demande. On se retrouve avec un code qui nous demande un **AppID** (le code du jeu sur Steam), dans l'exemple ci-dessous, le code serait 316790.

**store.steampowered.com/app/316790/Grim\_Fandango\_Remastered/**

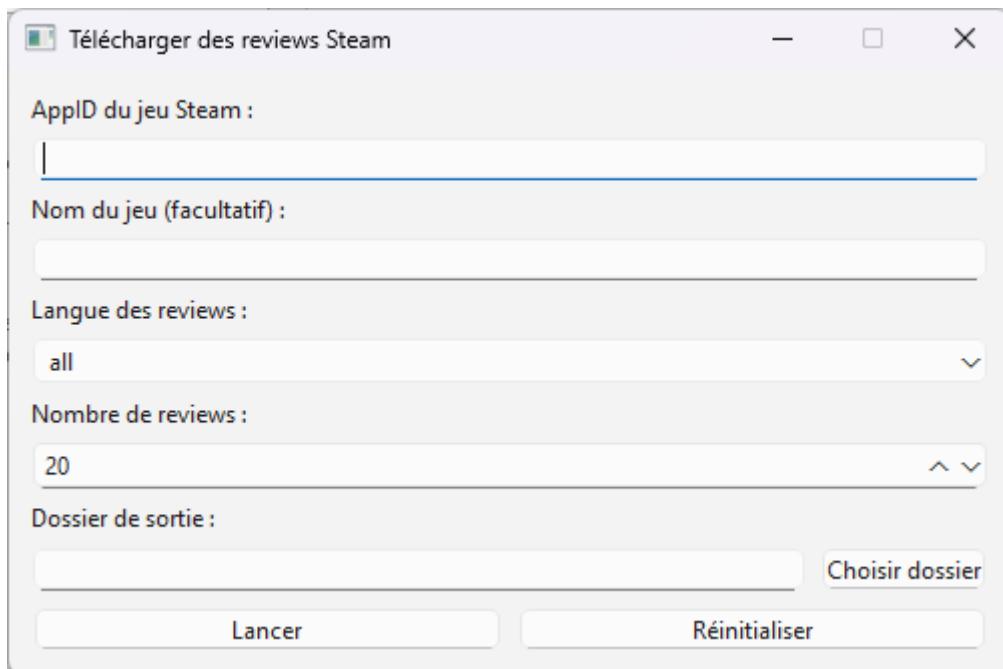
**Le nom du jeu**, qui nous permet de créer un dossier regroupant les reviews à l'intérieur.

**La langue** des reviews. (Majoritairement, nous avons choisis le français mais un certain nombre de jeu nous obligeait à mettre français et anglais pour respecter le nombre de reviews minimum que l'on s'était fixé)

**Le nombre** de reviews. (Ici, 20 maximum. En utilisant l'API Steam et en ayant un nombre plutôt faible de requêtes, on ne risquait pas d'être bloqué.)

### Le chemin du dossier de sortie.

Un bouton pour lancer le processus et un autre pour réinitialiser les champs.



Après avoir finaliser ce programme, il nous fallait un moyen de regrouper les informations importantes de notre analyse, autrement que chercher dans les dossiers créés par le programme, un par un. Nous avons décidé d'utiliser une base de données SQL qui regroupera toutes les informations nécessaires. De la même manière que la récupération des reviews, on a créé et utilisé un programme python (**BDD\_Steam.py**) utilisant PySide6 et cette-fois-ci, le module **sqlite3** afin qu'on puisse directement entrer nos informations **comme le nom du jeu, l'AppID, la note metacritic, l'évaluation Steam, le nombre de reviews capturés, le nombre de reviews positives, le nombre de reviews négatives et s'il y a eu un cas de Review-Bombing ou autre controverse**. Le programme crée une base de données et ajoute les informations ou ajoute les informations données dans une BDD existante (si vous donnez une BDD lors de l'exécution du script).

Base de données Steam - Jeux Vidéo

Nom:	<input type="text"/>		
AppID:	<input type="text"/>		
Note Metacritic:	<input type="text"/>		
Évaluation Steam:	<input type="text"/>		
Reviews Capturés:	<input type="text"/>		
Reviews Positives:	<input type="text"/>		
Reviews Négatives:	<input type="text"/>		
Review-Bomb / Controverses:	<input type="text"/>		
<input type="button" value="Ajouter"/> <input type="button" value="Réinitialiser"/>			
ID	Nom	AppID	Metacritic
1	Disco Elysium - ...	632470	97
2	Half-Life 2	220	96
3	Red Dead ...	1174180	93
4	Baldur's Gate 3	1086940	96
5	Half-Life	70	96

Exemple de review :

Note :   
 One of the greats. I bought this when it came out and I still go back to it every few years. It's cute, it's funny, and the puzzles are clever without being crazy difficult. Also, the atmosphere (day of the dead + film noir) really works.

Chaque review possède une ligne (la première) ayant une « note » :  ou 

Et quelques lignes constituant l'avis du joueur.

C'est avec cela que l'on a pu commencer notre projet.

## Contenu :

Un jeu par note Metacritic. [76 jeux différents]

Un nombre de reviews globalement supérieur à 10 par jeu (il existe cependant des exceptions compte tenu de la popularité ou encore du manque de reviews). [1275 reviews au total]

Les reviews de jeux rangés dans des dossiers nommés par le nom du jeu.

Ces mêmes jeux sont classés par tranche de note (0-10, 10-20, 20-30, ..., 90-100).

7 bases de données SQL. (**BDD\_Review.db**) contenant les données principales.

(**resultat\_analyse\_sentiment.db**) contenant les analyses de sentiments

(**resultats\_scores\_evaluation.db**) contenant les analyses de mesure d'évaluations (f-mesure, rappel et précision) et 4 (**resultats\_classifieurs.db**), chacun contenant les analyses des classifieurs pour les tranches respectives analysés (30-40, 90-100, global et global sans oversampling)

4 graphiques d'analyse concernant les mesures statistiques et de sentiment

21 graphiques d'analyse concernant les mesures de classifications (Matrice de confusions, graphiques de comparaison de classifieurs, f-mesure des classifieurs)

1 rapport d'analyse statistique en format .txt et .csv (**rappor\_reviews\_steam**)

4 rapports en .txt (**resultats\_classifieurs**) contenant les mêmes résultats que ceux de la base de données (resultats\_classifieurs.db) et une statistique sur les fichiers utilisés en plus.

8 scripts :

- **Requirements** (permet de vérifier et d'installer les modules nécessaires au projet)
- **SteamReviewDownloader** (permettant de récupérer par l'intermédiaire de l'API Steam les avis nécessaires à notre analyse)
- **BDD\_Steam** (permettant de remplir et/ou créer une base de données avec les éléments soutirés des avis Steam)
- **Analyse\_stat** (script dédié à l'analyse statistique)
- **Analyse\_sentiment** (script dédié à l'analyse du sentiment des avis Steam)
- **Analyse\_eval\_score** (script calculant les mesures d'évaluations comme la f-mesure, la précision et le rappel pour les avis Steam par tranche Metacritic)
- **Analyse\_classifieurs** (script dédié à la classification et à l'analyse des classifieurs utilisés)
- **F\_score\_graph** (script créant un graphique des f-scores des classifieurs analysés)

### Améliorations possibles – Récupération des Reviews :

Alors, notre script est censé récupérer les reviews Steam considérés comme « pertinentes » par ce dernier mais il suffit de regarder certaines reviews récupérées pour se poser la question sur la véracité de la pertinence de certaines d'entre elles. Il est sûrement possible d'affiner notre programme pour avoir des reviews de qualité supérieure (ou également taper sur les doigts sur Steam et leur évaluation des avis pertinents).

Il est sûrement possible de pouvoir automatiser la recherche et le téléchargement de nos données plutôt que de devoir rechercher les AppID et les noms des 76 jeux de notre corpus afin d'avoir nos données.

Dans notre recherche de reviews, nous avons omis certaines informations comme le type de jeu analysé, les développeurs/éditeurs, la date de sortie. Ces informations peuvent être ajoutées dans une recherche future.

Nous n'entrons pas dans les détails d'améliorations de l'interface car ce n'est pas réellement notre intérêt principal mais lui aussi est améliorable.

### Amélioration possibles – Base de données SQL :

Ici, la base de données est créée dans un second temps (dans un programme entièrement différent de celui de la récupération des reviews). On pourrait faire en sorte que lors de la recherche et du téléchargement des reviews, une option à cocher dans le script Steam permettant d'ajouter les informations récupérées dans une base de données existante ou en crée une si elle n'existe pas.

### Hypothèse :

En lançant le projet, la question que l'on s'est posé est : **Est-ce que les notes de la presse sont-elle majoritairement en harmonie avec celles des joueurs ?** A la recherche d'une corrélation.

### Analyses statistiques :

Nous avons quelques analyses statistiques. Pour s'y faire, nous avons utilisé un autre script python (**analyse\_stat.py**) qui, nous permettent d'obtenir des informations intéressantes. Ici, la subtilité est que les informations sont prises dans notre base de données par le script et sont analysés par la suite pour nous ressortir des résultats. En utilisant cette fois-ci, **pandas**, **matplotlib** et **pearsonr**, on a déterminé et calculé :

Concernant Metacritic,

Le nombre de notes **(76)**, la moyenne **(62,1)**, l'écart-type (standard deviation/std) **(22,2)**, la note minimale **(19)**, 25% **(43,75)**, 50% **(62,5)**, 75% **(81,25)** et la note maximale **(97)**.

Concernant les évaluations Steam,

Le nombre d'évaluation **(76)**, le nombre d'évaluation uniques **(8)**, l'évaluation la plus représenté **(Très positives)** et sa fréquence **(22)**.

Comme on peut le remarquer, les évaluations Steam sont, à l'origine, des chaînes de caractères, donc il a fallu **les représenter numériquement**, ce qui a été fait dans le script par ce passage en particulier :

```
# Nettoyage des évaluations Steam
steam_eval_map = {
    "extrêmement négatives": -4,
    "très négatives": -3,
    "négatives": -2,
    "plutôt négatives": -1,
    "moyennes": 0,
    "positives": 1,
    "plutôt positives": 2,
    "très positives": 3,
    "extrêmement positives": 4
}
```

Un calcul de la corrélation entre **la notation Metacritic (l'avis de la presse)** et **l'évaluation Steam (l'évaluation des joueurs)** a également été faite par le biais du module pearsonr et donc de la **corrélation de Pearson**.

Ce résultat est de : 0.84. Ce qui signifie **une forte corrélation entre les deux systèmes de notations**. (Plus tu te rapproches de 1, plus ta corrélation est forte et inversement, plus tu te rapproches de 0, ta corrélation devient de plus en plus faible).

Il y a également un résumé des informations déjà connus concernant les notes Metacritic par tranche mais en rajoutant le ratio de notes positives. Ces informations sont disponibles dans le fichier : « **Rapport\_review\_steam.txt** ».

Ces informations sont normalement, aussi disponible en format csv (**rapport\_review\_steam.csv**) mais ce dernier dysfonctionne et les informations ne sont pas écrits correctement.

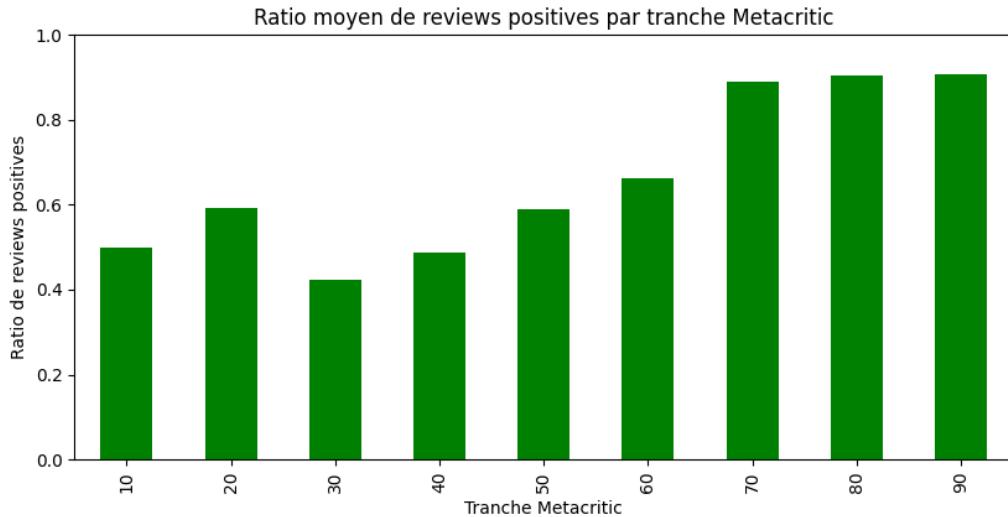
**PS** : Après s'être penché sur ce problème, nous l'avons finalement corrigé (l'erreur n'était visible que lors de l'ouverture du fichier avec Excel, la séparation des colonnes était mal interprétée). En rajoutant « `sep= ';'`  », le problème s'est résolu et nous avons donc un csv fonctionnel.

```

csv_path, _ = QFileDialog.getSaveFileName(self, "Exporter le résumé en CSV", "", "CSV Files (*.csv)")
if csv_path:
    summary.to_csv(csv_path, index=False, sep=";")
    self.text_output.append(f"Résumé exporté : {csv_path}")

```

Revenons sur le ratio de notes positives, voici le graphique (généré par notre script) :



On remarque **que plus l'on augmente la tranche, plus le ratio de reviews positives augmente**. Ce qui semble logique. Toutefois, on peut remarquer une anomalie dans cette logique, la tranche 10 et 20. Pourquoi cela ? Cela s'explique par le nombre de jeu et de reviews présent dans ces catégories. (0 jeu/review entre 0-10, 1 jeu et 2 reviews entre 10-20, 5 jeux et 72 entre 20-30). Comme ces catégories sont plus lacunaires, ces deux éléments sont des anomalies causées par nos conditions d'analyse. Notre hypothèse semble être juste mais continuons notre recherche pour le confirmer.

### Analyse de sentiment :

Pour l'instant, on s'était plutôt concentré sur des statistiques obtenues par ce qui entoure les reviews plutôt que les avis en elle-même. Maintenant, on va se concentrer sur les mots des reviews et ce que l'on peut en soutirer. Premièrement, on va rechercher dans les reviews **le sentiment**, c'est-à-dire, soutirer à partir des mots un ressenti que ce soit positif, négatif ou neutre, que l'on peut également quantifier. On peut faire une analyse par jeu mais également par tranche de note Metacritic (par ex : 70-80). Cela nous permettrait de connaître l'engouement du joueur d'une autre manière que celle utilisé par nos analyses statistiques.

Pour s'y atteler, nous avons utilisé un script (**analyse\_sentiment.py**) utilisant toujours **PySide6** (interface graphique), **matplotlib** (pour la représentation graphique de nos analyses) mais aussi **langdetect** (afin de détecter la langue utilisée dans nos avis Steam donc soit anglais soit français), **textblob**, **textblob\_fr** et **nltk** (les trois modules sont utilisés pour l'analyse de sentiment dont **textblob\_fr** pour le français). Comme nos autres scripts, on enregistre les résultats dans une base de données (**resultat\_analyse\_sentiment.db**) et on

génère également un graphique résumant nos analyses (en réalité 2 car il y a 2 analyses différentes, **le graphique du sentiment moyen par jeu et le graphique du sentiment moyen par tranche Metacritic**).

La BDD contient **les tranches metacritic** (0-10, 10-20, ...), **le sentiment moyen** (plus l'on se rapproche de -1, plus l'avis est négatif et inversement, plus l'on se rapproche de 1, plus l'avis est positif) [calculé par la moyenne des scores obtenus par les reviews du jeu analysé sémantiquement et quantifiés], **le nombre de reviews** et **l'écart Metacritic/sentiment** (calculé par : (note metacritic /100) – sentiment\_moyen))

[Si l'écart est proche de 0, l'avis des joueurs est en lien avec le Metacritic. Si l'écart est positif (on se rapproche de 1 et plus), Metacritic est plus généreux que les joueurs. Si l'écart est négatif (inférieur à 0), les joueurs sont plus positifs que Metacritic]

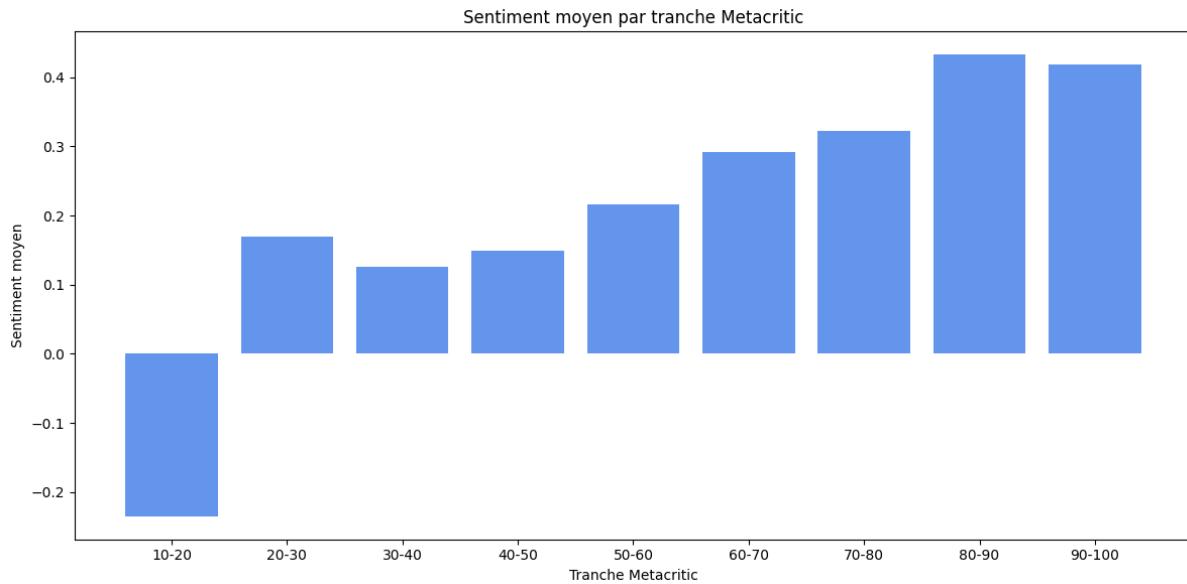
Avec cela, on peut trier l'écart Metacritic afin de déterminer la différence de ressenti entre la presse (Metacritic étant un agrégateur de note de la presse) et les joueurs. Exemple :

67	Baldur's Gate 3	90-100	0.4041	20	0.5459
68	Bioshock	90-100	0.5646	20	0.3854
69	Disco Elysium - The Final Cut	90-100	0.182	13	0.768

Dans cet exemple, c'est la dernière colonne qui nous intéresse. Ici, Baldur's Gate 3 récolte 0,54 d'écart, Bioshock 0,38 et Disco Elysium 0,76. Comment l'interpréter ? Déjà, les trois jeux ont un écart positif, Bioshock se rapproche le plus de 0 comparé aux deux autres. C'est donc lui qui possède l'avis des joueurs le plus proche de celui de Metacritic. Baldur's Gate et particulièrement Disco Elysium montre que Metacritic est plus généreux (une meilleure note) que les joueurs.

Nous ne mettrons pas d'image du graphique traitant des sentiments par jeu, étant donné que sa taille est plutôt imposante, cela rendrait le graphique plutôt difficile à lire dans un document (*graphique\_sentiment\_par\_jeu.png*). Cependant, nous pouvons ajouter le graphique traitant du sentiment par tranche de note Metacritic que voici :

(*graphique\_sentiment\_par\_tranche.png*)



De la même manière que le graphique des ratios moyen de reviews positives comparé à la tranche metacritic, les deux premières colonnes sont biaisées par le manque de reviews et de jeu. Cependant la tendance globale de ce graphe (calculé avec des arguments purement quantitatifs) est la même que celle du sentiment moyen (calculé avec des arguments purement textuels), plus la tranche augmente, plus les joueurs sont en harmonie avec la presse et les reviews sont positives. Logique. **Les joueurs sont globalement en accord avec la presse.** (En tout cas dans notre cas d'analyse)

Après avoir vu les sentiments, on devait également traiter les mesures d'évaluation. Cela nous permettrait de déterminer si l'analyseur de sentiment que l'on a utilisé est de bonne qualité. (Si l'analyse est précise, ne fait pas d'erreurs ou oubli). Pour s'y prendre, vous l'aurez deviné, un autre script (**analyse\_eval\_score.py**), lui, dédié aux mesures d'évaluations (**f-mesure, précision et rappel**). Ici, on utilise, en dehors des modules déjà utilisé dans d'autres scripts, **sklearn.metrics** pour calculer les mesures voulues. Pour que le calcul fonctionne, il nous faut **une valeur de référence** à comparer avec celle que l'on va recevoir de l'analyse. Pour s'y faire, on utilise la première ligne des reviews comme token (positif ou négatif), on calcule le nombre d'avis positifs et négatifs par tranche et par jeu avec cette ligne. Finalement, la prédiction, elle, se fait simplement qu'avec l'analyse des sentiments (donc en excluant la première ligne) :

```

note_line = lines[0].strip()
review_text = " ".join(line.strip() for line in lines[1:] if line.strip())

if "👍" in note_line:
    y_true.append(1)
elif "👎" in note_line:
    y_true.append(0)
else:
    continue

try:
    lang = detect(review_text)

    if lang == 'fr':
        sentiment = TextBlob(review_text).sentiment.polarity
        y_pred.append(1 if sentiment > 0 else 0)

    elif lang == 'en':
        score = analyzer.polarity_scores(review_text)["compound"]
        y_pred.append(1 if score > 0 else 0)

    else:
        y_pred.append(0)

except Exception:
    y_pred.append(0)

```

```

if y_true and y_pred:
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    scores_par_tranche[tranche] = (precision, recall, f1)

    global_y_true.extend(y_true)
    global_y_pred.extend(y_pred)

    c.execute("INSERT INTO resultats VALUES (?, ?, ?, ?)", (tranche, precision, recall, f1))
    conn.commit()

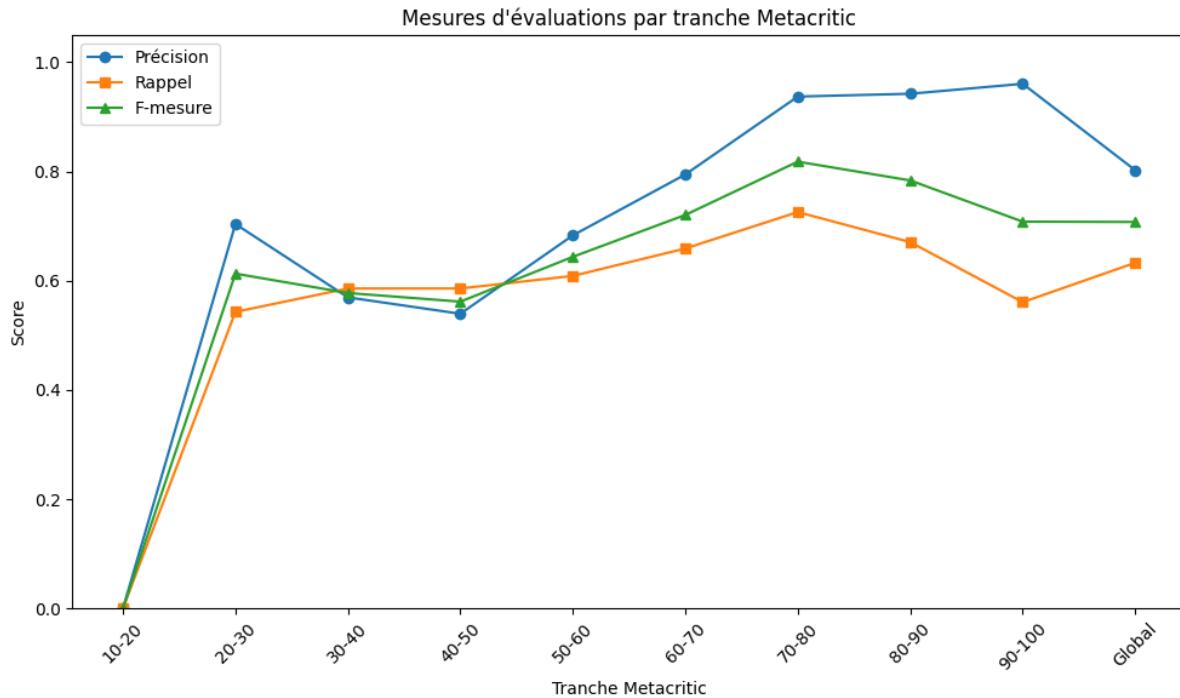
# Résultats globaux
if global_y_true and global_y_pred:
    precision = precision_score(global_y_true, global_y_pred)
    recall = recall_score(global_y_true, global_y_pred)
    f1 = f1_score(global_y_true, global_y_pred)
    scores_par_tranche["Global"] = (precision, recall, f1)
    c.execute("INSERT INTO resultats VALUES (?, ?, ?, ?)", ("Global", precision, recall, f1))
    conn.commit()

conn.close()

self.show_scores_chart(scores_par_tranche)

```

On choisit la ou les tranches Metacritic contenant les reviews à analyser et on obtient une base de données (**resultats\_scores\_evaluation.db**) contenant les résultats par tranche et un graphique les représentant :



Si l'on prend les valeurs globales, c'est-à-dire, toutes les tranches, on obtient :

**Précision : 0.80**

**Rappel : 0.63**

**F-mesure : 0.71**

Que peut-on en déduire ?

Concernant la précision, on peut en déduire que le modèle d'analyse de sentiment est **plutôt précis**, il fait peu d'erreurs mais cela ne veut pas dire qu'il est entièrement correct.

Le rappel, lui, est **plutôt faible**, ce qui montre effectivement que le modèle ne réussit pas à obtenir toutes les informations correctement. Cela peut être due par le niveau de langue, les tons utilisés dans les reviews ou encore le fait de ne pas avoir de mots/phrases dans la review en question.

La f-mesure, quant à lui, est **plutôt moyenne**. Pas mauvaise mais pas excellent non plus, plutôt bonne compte tenu des conditions d'analyse et du sujet. Le modèle est plutôt équilibré mais a une tendance à accentuer la précision.

L'analyse de sentiment est donc **globalement fiable mais n'est pas sans faille et commet des erreurs** (rappel « faible »). Il faudra donc faire attention à ne pas généraliser les résultats de ce dernier.

Avec nos données, l'analyse statistique et l'analyse de sentiment, on arrive à une conclusion, effectivement, **la presse est globalement en accord avec les joueurs sur la notation des jeux-vidéos**. Notre hypothèse se révèle juste (en tout cas sur notre base de données).

Toutefois, nous ne nous arrêtons pas ici. Comme vous l'aurez pu le remarquer, afin d'obtenir toutes ces analyses, un long processus a dû se mettre en place (pas toujours pratique, d'ailleurs). Utilisons des **classifieurs** afin de d'avoir **une deuxième analyse « automatique »** (autre que l'analyse de sentiment), vérifions d'abord s'ils sont fonctionnels, entraînons certains modèles et analysons leurs résultats.

Mais avant cela, abordons rapidement les améliorations possibles de nos analyses statistiques et de sentiment.

PS : Normalement, lorsque l'on effectue le calcul des mesures d'évaluations avec le script (et toutes les tranches d'activités), une phrase s'affiche sur le terminal : « Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero\_division` parameter to control this behavior.

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result)) »
```

Cela est dû à la tranche 0-10 étant vide, il n'est donc pas possible de calculer quelque chose. On aurait pu simplement ne pas la proposer au calcul mais, on n'y avait pas pensé.

#### Améliorations possibles – Analyses statistiques et sentiment :

Dans les analyses statistiques, il me semble que l'on a abordé les principaux éléments mais d'autres recherches étaient possibles. Premièrement, comme vous l'aurez remarqué, on aurait pu faire une analyse de corrélation entre les notes, reviews et controverses (si on en avait plus qu'une seule !). On pouvait également déduire avec nos éléments si on était en présence de jeu « sous-coté » ou « surcoté », c'est-à-dire, un jeu ayant un score Metacritic haut mais un sentiment faible (pour le cas du surcoté) et inversement pour le sous-coté.

Du côté de l'analyse de sentiment en elle-même, il y avait la possibilité de soutirer des reviews, les sujets qui reviennent le plus dans les avis positifs et négatifs et les confronter. On pouvait également créer un nuage de mots mettant en valeur les mots les plus employés par tranche ou même par langue ou ressenti. En ajoutant les catégories de jeu aux informations principales, on aurait pu analyser les différences de notations, sentiments par type de jeu. En rajoutant la date de la review, on aurait pu analyser l'évolution des reviews et notations selon les périodes, la corrélation entre les événements liés aux jeux et le sentiment et notation. En rajoutant le studio ou éditeur, on aurait pu évaluer les notations et sentiment par studio/éditeur et faire une analyse comparative entre les gros studios et les indépendants. En rajoutant le prix, on aurait pu voir par les reviews en elle-même et la notation si le prix a un impact sur l'appréciation du jeu.

Il y a également des éléments que l'on a effleuré et que l'on pouvait élaborer plus en détail. Dans l'analyse de sentiment, qui manque de nuance, on a majoritairement le choix entre positif ou négatif dans notre analyse. On n'aborde pas le sentiment par review mais seulement le sentiment moyen par jeu et on ne calcule pas la variance du sentiment, c'est-à-dire le fait de vérifier si les avis sont divergents ou homogènes. On peut évidemment croiser plus en détail les BDD que nous avons à disposition afin d'avoir des résultats supplémentaires. Concernant les mesures d'évaluations, rendre plus efficace la détection de la langue et rajouter une matrice de confusion est déjà une bonne amélioration.

### Classificateurs évalués :

Pour l'analyse avec classificateurs, nous avons décidé d'en utiliser et comparer **4** :

- **Naïve Bayes**
- **SVM**
- **Random Forest**
- **Logistic Regression**

**Naïve Bayes** est un algorithme d'apprentissage se reposant sur l'hypothèse d'indépendance des caractéristiques (ici, mot) afin d'estimer la probabilité qu'un élément appartienne à une classe.

**SVM** (Support Vector Machine ou Machine à vecteurs de support) : Les SVMs sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie. (Définition extraite de dataanalyticspost.com)

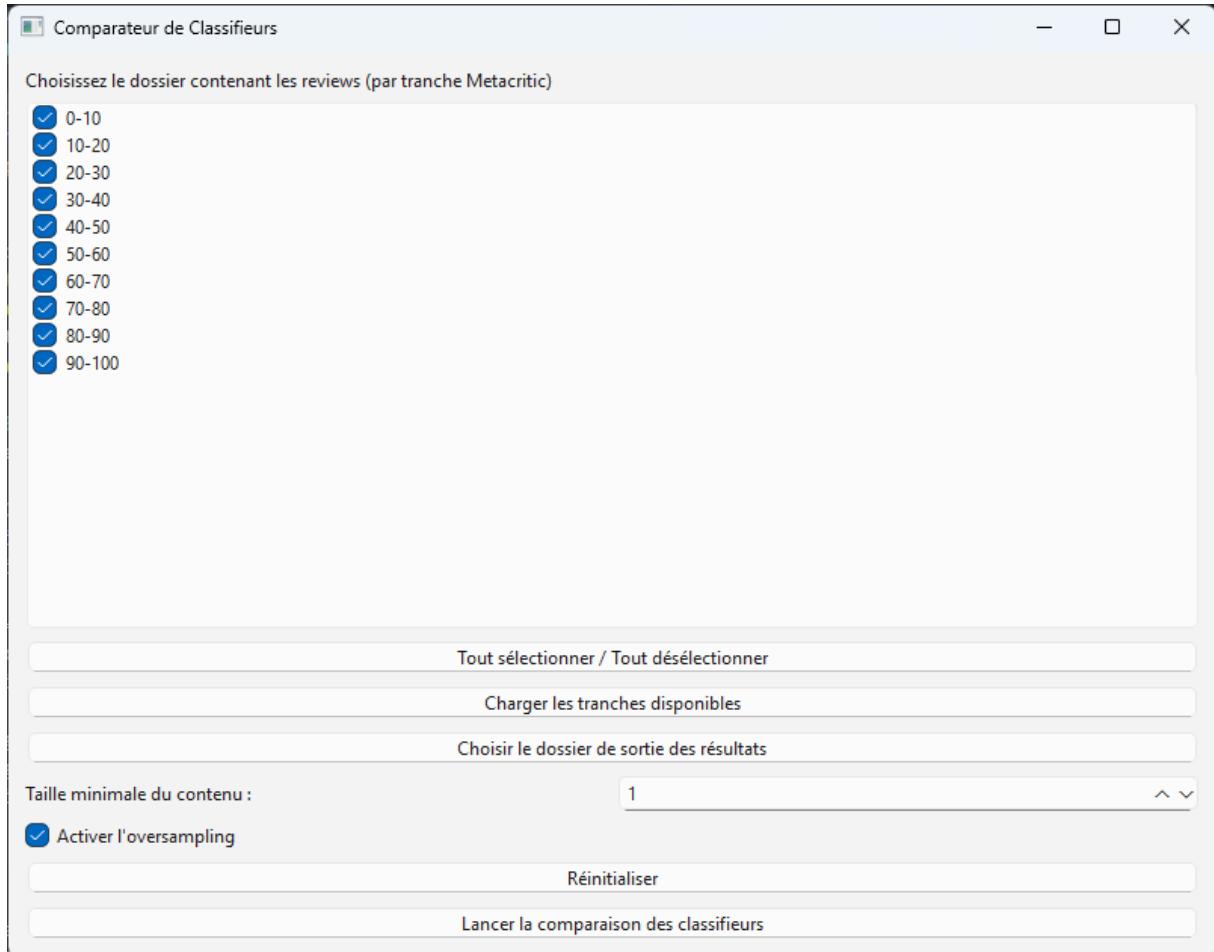
**Random Forest** est un modèle reposant sur les arbres de décisions. Il combine les résultats de ses arbres afin d'avoir un résultat plus fiable.

**Logistic Regression** est un modèle statistique qui permet d'étudier les relations. C'est un algorithme de classification binaire prédisant la probabilité qu'un élément appartienne à une classe.

### Classification et Analyse :

Nous devons utiliser les quatre classificateurs cités ci-dessus afin de les entraîner sur nos données, pour ensuite les tester, analyser les résultats et en tirer une information. Pour s'y prendre, nous avons un script (**analyse\_classificateurs.py**) utilisant **PySide6** (interface graphique), **sqlite3** (BDD), **numpy** (calculs), **matplotlib** (visualisation graphique), **scikit-learn**

(pour tous les modèles de classifieurs, la séparation des données de train et test, l'oversampling et les analyses) et **seaborn** (visualisation).



Voici à quoi ressemble le programme lorsque l'on charge le dossier racine du projet. Il permet de **choisir les tranches Metacritic à évaluer**, **le dossier de sortie des résultats**, **la taille minimale du contenu** (c'est-à-dire, le nombre de caractères attendu dans les reviews analysés, ici, au moins 1), **l'oversampling** (qu'on élaborera par la suite), **le bouton réinitialiser** (pour remettre à 0 l'app, au lieu de la fermer à chaque fois).

Les modèles sont ici :

```
models = {
    "Naive Bayes": MultinomialNB(),
    "SVM": SVC(),
    "Random Forest": RandomForestClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=1000)
}
```

Les données sont vectorisées par le TF-IDF :

```
vectorizer = TfidfVectorizer(max_features=5000) # Vectorisation par le TF-IDF
```

Puis séparer dans un ratio 80/20 (80% pour le train et 20% pour le test) :

```
X_train, X_test, y_train, y_test = train_test_split(X_vect, y, test_size=0.2, random_state=42)
```

De la même manière que dans le calcul des mesures d'évaluations précédemment, on sépare la première ligne des reviews afin de ne pas fausser l'entraînement des modèles et les résultats : (La ligne correspondant à la note étant l'index 0)

```
content = " ".join(lines[1:]).strip()
```

Après l'entraînement du modèle et le test, le programme enregistre les résultats dans une base de données (**resultats\_classifieurs.db**) et un fichier texte. Il crée des matrices de confusion pour chaque algorithme testé (donc 4) et un graphique comparant les mesures d'évaluations (précision, rappel, f-mesure) des classifieurs. (**comparaison\_classifieurs.png**).

Cependant nous nous sommes confrontés à un problème. Les résultats de nos analyses étaient étranges. Sur une analyse de l'intégralité des tranches, nous n'avions que **très peu de détections d'avis négatifs**. C'est un problème qui nous a pris un bon moment pour enfin comprendre comment « essayer » de le contourner. Pourtant, les raisons sont plutôt évidentes. **Notre corpus est majoritairement dominé par des avis positifs** donc la classe « positive » est clairement majoritaire comparé à la classe « négative » et influe sur nos valeurs et résultats. Pour y remédier, nous avions plusieurs possibilités, **récupérer plus de reviews négatives** (mais non seulement, cela nous obligeait à reprendre toutes nos analyses depuis le début mais compte tenu de nos conditions d'élaboration de notre corpus, cette idée n'est pas possible), **créer artificiellement des reviews négatives** (soit à la main, soit « automatiquement » par le biais de la génération synthétique mais cela aurait créé des exemples possiblement peu réalistes pour notre corpus, biaise d'autant plus l'intérêt de notre recherche étant donné que l'on voulait analyser les avis de joueurs donc humain. De plus, étant donné qu'une grande partie de nos données sont en français, on n'avait aucune certitude que cette méthode fonctionne à cause de la langue). Il nous restait que l'**oversampling**. **L'oversampling, c'est le fait de dupliquer aléatoirement des éléments de la classe minoritaire afin d'équilibrer les classes**. C'est pratique et plutôt simple cependant les données (ici négatives) perdront en diversité et le **risque de surapprentissage est plus grand**.

**Le surapprentissage, c'est le fait que le modèle apprend intensément les données d'entraînement et finit par être performant que lors qu'il est utilisé pour ces mêmes données. Il perd sa capacité à généraliser.**

Partie concernant l'oversampling dans le script :

```
if self.oversampling_checkbox.isChecked(): # Oversampling (au cas où une classe à évaluer est clairement minoritaire, ici les avis négatifs)
    X_train_array = X_train.toarray()
    y_train_array = np.array(y_train)

    class_0_indices = np.where(y_train_array == 0)[0]
    class_1_indices = np.where(y_train_array == 1)[0]

    if len(class_0_indices) > 0 and len(class_1_indices) > 0:
        if len(class_0_indices) > len(class_1_indices):
            maj, min_ = class_0_indices, class_1_indices
        else:
            maj, min_ = class_1_indices, class_0_indices

        min_upsampled = resample(min_, replace=True, n_samples=len(maj), random_state=42)
        indices_final = np.concatenate((maj, min_upsampled))
        X_train = X_train_array[indices_final]
        y_train = y_train_array[indices_final]
```

Maintenant que le problème est réglé, on a choisi plusieurs tranches spécifiques à analyser :

- **30-40**
- **90-100**
- **Global (signifie toutes les tranches)**
- **Global (toutes les tranches mais sans oversampling)**

**30-40**, car c'est la tranche la plus basse du corpus avec le nombre maximum de jeux (10, les tranches en dessous en ont moins)

**90-100**, c'est la tranche la plus haute du corpus.

**Global**, l'intégralité du corpus

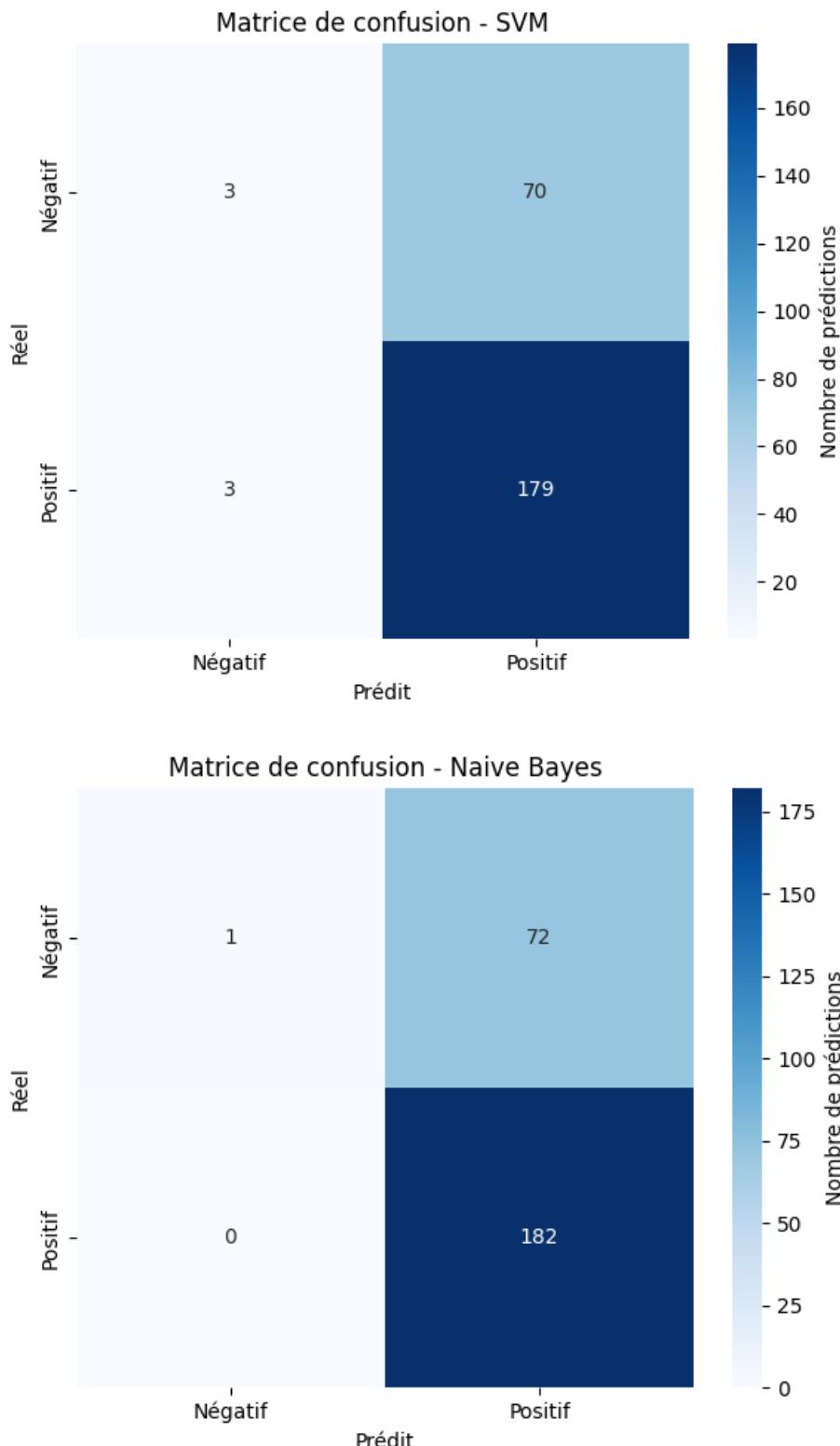
**Global (sans oversampling)**, simplement pour mettre en avant la différence entre le corpus sans modification et sa version oversamplé.

Tous les résultats de ces analyses sont disponibles dans le dossier « **Analyse de Classification** »

Ici, on va simplement mettre en évidence le Global et le Global sans oversampling.

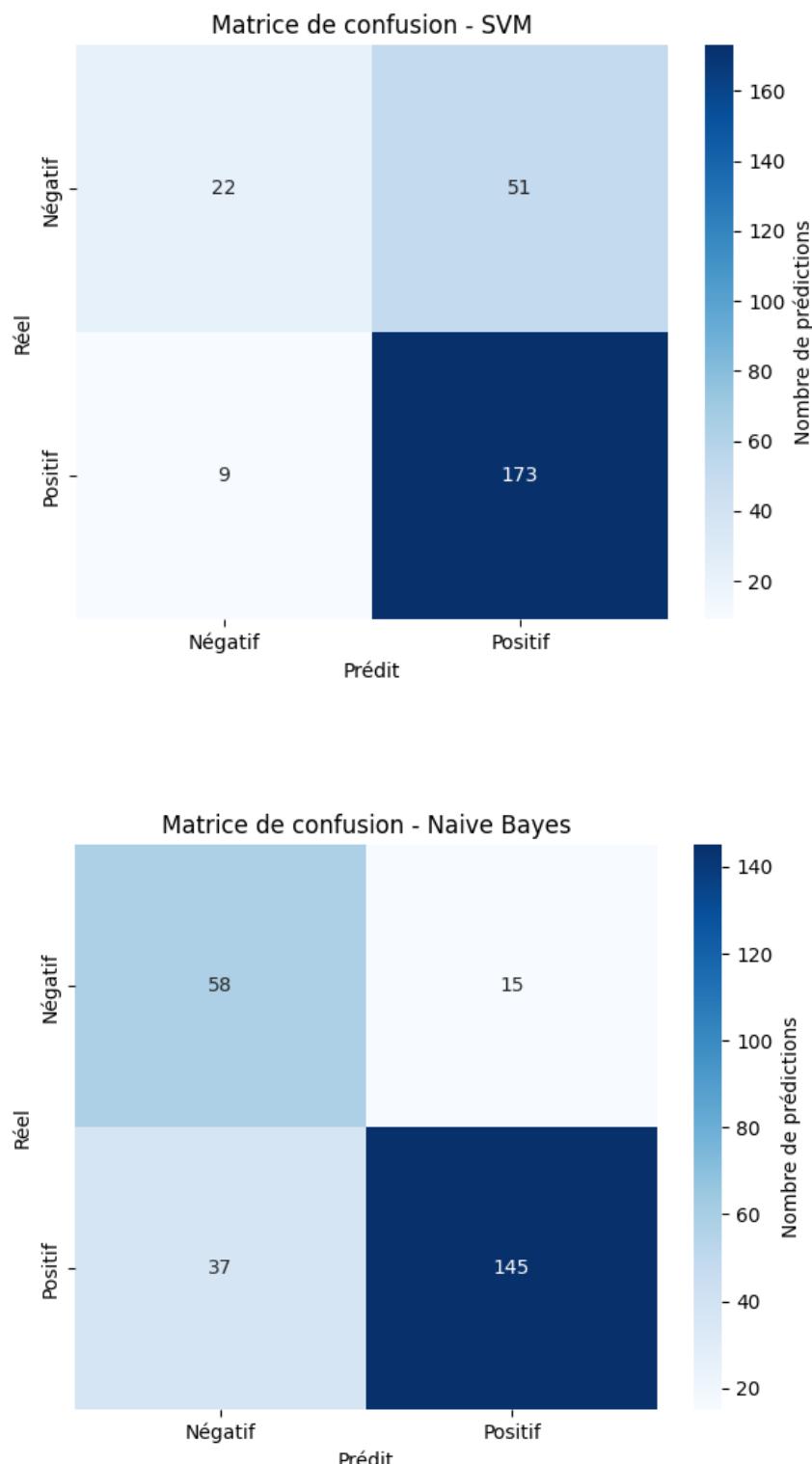
**Rappel de la séparation des données : 1275 reviews (100%) / 1020 reviews (80%) pour l'entraînement / 255 reviews pour le test (20%)**

### Matrice de confusion pour SVM et Naïve Bayes (global non oversamplié) :



Comme prévu, les avis négatifs (TN et FN – True Negative et False Negative) sont presque inexistant et les avis positifs (particulièrement le TP – True Positive) est exagérément haut.

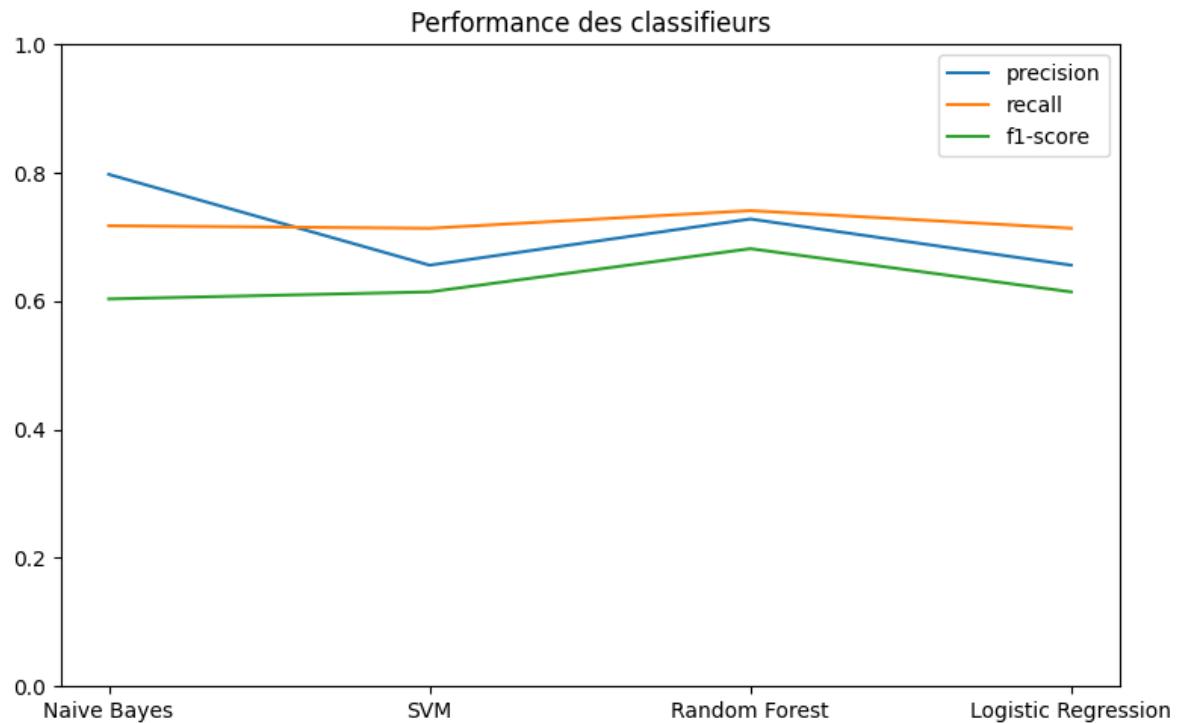
### Matrice de confusion pour SVM et Naïve Bayes (global oversamplié) :



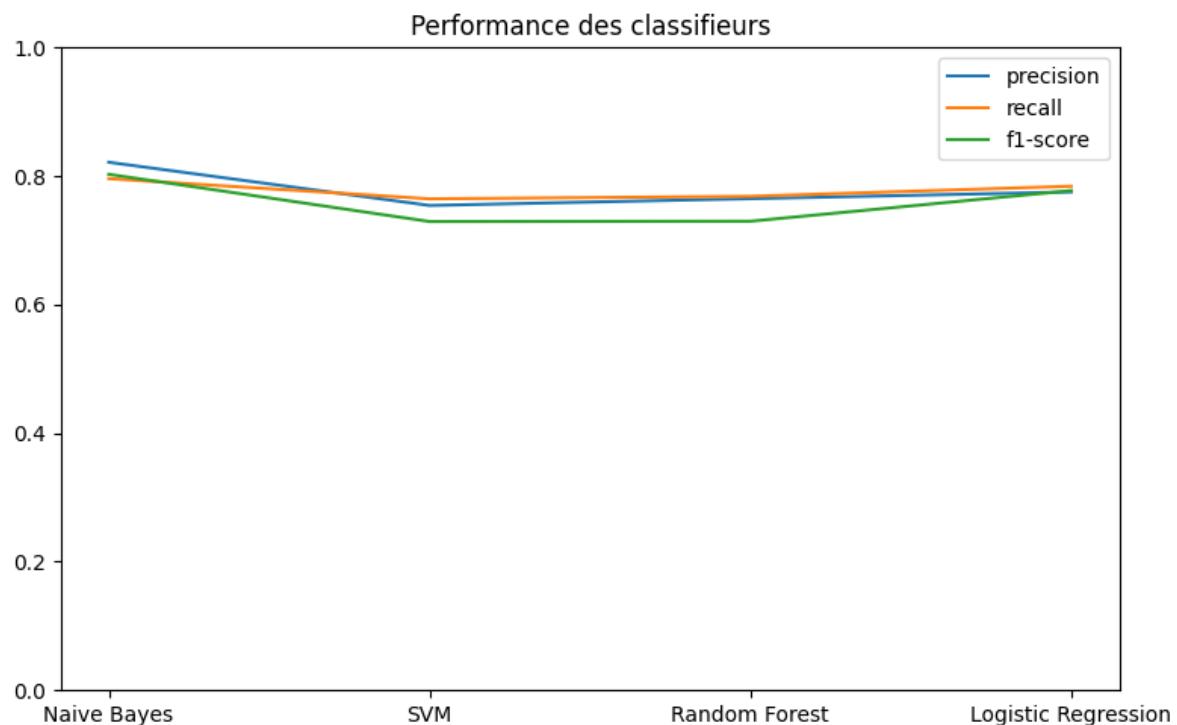
Ici, on a une meilleure prédiction des avis négatifs mais toujours des erreurs de classifications en général. On avance.

**Comparaison entre les classifieurs et différences entre la tranche globale oversamplé et la tranche globale classique.**

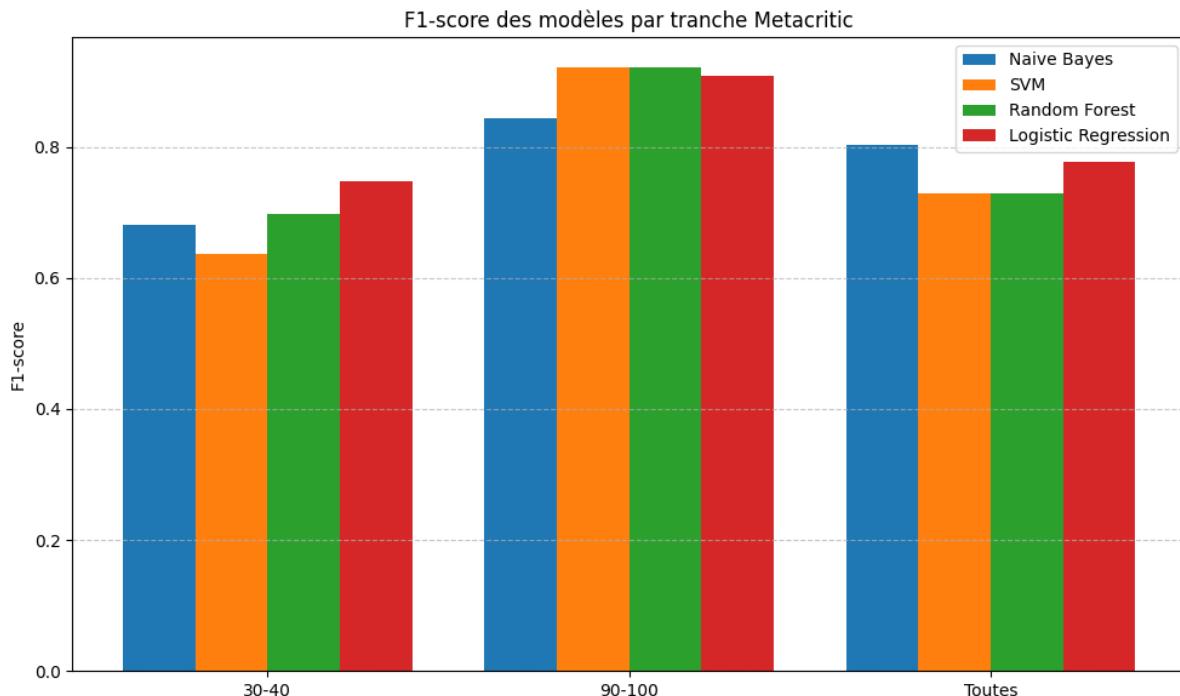
**Sans oversampling :**



**Avec oversampling :**



Sans oversampling, les valeurs sont beaucoup plus contrastées, on remarque plus de variations entre les modèles. Alors qu'avec l'oversampling, c'est beaucoup plus rapproché et globalement plus haut. Maintenant, est-ce que ça veut pour autant dire qu'ils sont plus efficaces ? Difficile à conclure. Cependant, on peut essayer de déterminer quel est le modèle le plus performant du lot. Pour une dernière fois, on utilise un script (**f\_score\_graph.py**), beaucoup plus simple que les autres (**matplotlib et numpy**), sans interface graphique, il nous permet de créer **une visualisation des f-mesure des classifieurs évalués sur leurs tranches** (excepté, la tranche Global sans oversampling).



En regardant ce graph, le modèle semblant être le plus performant sur les différentes tranches (en étant oversmplé) se révèle être **l'algorithme de Logistic Regression**.

Valeurs supérieures (f-score, précision, rappel) et globalement plus stable que les autres, meilleures valeurs que sa contrepartie sans l'oversampling et également supérieur au calcul des mesures d'évaluations des sentiments.

Alors, peut-on dire que le modèle de Logistic Regression est le plus efficace et adapté aux données ? Oui et non. Oui car c'est effectivement ce que nous annonceraient les résultats. Non, car nos données ne sont pas les plus équilibrées possibles et montrent sûrement **un biais de performance sur les modèles**, qui n'est pas du tout aidé par l'oversampling. Cependant, avec tout cela, l'hypothèse que nous avions émise à notre début se confirme une fois de plus (pas de la meilleure des façons).

### Améliorations possibles – Classification et Analyse :

Premièrement, on l'a pensé que durant la rédaction du rapport mais on aurait pu **rajouter la possibilité de choisir sur l'interface graphique les paramètres du modèle** comme la **taille/pourcentage de données dédiés à l'entraînement ou encore au test**. On pouvait également créer une option pour **laisser durant l'analyse la ligne concernant la « Note » dans l'entraînement du classifieur pour y évaluer son impact**. Notre évaluation se repose entièrement sur le calcul de la précision, rappel et f-mesure. Etant donné notre situation de déséquilibre, on aurait pu implémenter **une validation croisée afin d'affiner nos résultats d'évaluations**. Ajouter la possibilité **de récupérer ou encore lister les fichiers qui ont été mal classés pour les examiner**. On peut également faire le **calcul de la variance sur les scores par tranches afin de vérifier l'efficacité et la robustesse des modèles** selon les tranches. Enfin il y a deux améliorations particulièrement évidentes que l'on aurait pu faire mais l'idée ne nous est venue à l'esprit qu'à la fin. La première, c'est **le fait de ne pas avoir mis la création du graphique du f-score des classifieurs évalués dans le script dédié à l'analyse des classifieurs**. Le second et c'est le plus pénalisant vu qu'on y avait pensé pour le script de l'analyse des sentiments mais complètement oublié ici, c'est le fait que **la classification, l'entraînement, le test et l'analyse soit faite sans prendre en compte la langue des reviews**. Si notre corpus était intégralement en Français ou en Anglais, il n'y aurait pas de problème. Ici, on a **un corpus multilingue** et donc, on aurait du prendre cela en compte ici, pour la vectorisation (avec la gestion des stopwords, lemmatisation ou stemming) et pour l'apprentissage des modèles, qui s'en retrouve impacté encore plus qu'elle était déjà.

### Conclusion :

En ayant des conditions d'analyse assez spécifiques, on voulait analyser les reviews Steam des jeux. En allant des jeux les mieux notés sur Metacritic, site référence agglomérant la notation de la presse (et des joueurs mais c'est plus anecdotique que Steam sur ce point et ici on a capturé seulement les avis de la presse) au pire du pire, nous avons récolté un certain nombre d'avis pour un grand nombre de jeux différents. En utilisant en premier lieu les informations gravitant autour des reviews (une analyse statistique), nous avons pu établir une analyse entre les avis des joueurs et la notation de la presse et notre hypothèse que les joueurs et l'avis presse sont majoritairement en accord. Pour confirmer cela, on a également établi une analyse de sentiment (NLP) utilisant les reviews directement afin d'en soutirer le sentiment de ces derniers, faire des calculs et obtenir la deuxième confirmation que les avis de la presse et des joueurs sont en harmonie. Pour éviter de conclure l'analyse trop vite, nous avons voulu évaluer le modèle d'analyse de sentiment afin de savoir si nos analyses peuvent être prise en compte. Ce qui nous a révélé le fait que les analyses pouvaient être utilisés mais qu'elles ne sont pas entièrement fiables, des erreurs sont à prévoir. Après cela, nous avons donc décidé d'utiliser plusieurs classifieurs différents, de les entraîner afin qu'ils puissent classés les avis précédemment collectés dans le but de vérifier notre hypothèse et

trouver lequel est le classifieur le plus efficace dans notre cas. Non pas sans difficultés, erreurs et oublis, nous avons obtenu un résultat qui peut être pris en compte mais qui est loin d'être optimale. Cela reste une recherche intéressante mais clairement améliorable.

## Q&A :

### **Pourquoi avoir créé une case « Controverses » dans la base de données principale ?**

- Nous avons décidé d'ajouter cette colonne et mention car notre premier jeu à analyser (Disco Elysium) a eu une controverse ayant un impact sur la notation Steam du jeu. A partir de là, nous nous sommes rappelés de cette possibilité et du review-bombing donc nous avons ajouté cette colonne, cependant en remplissant notre base de données, nous nous sommes rendus compte que très peu de jeux de notre base de données est concerné par cette case (un seul).

### **Comment avez-vous choisi les jeux de votre base de données ?**

- Pour le choix des jeux, comme on l'a expliqué précédemment, on a utilisé Metacritic, uniquement les jeux PC disponibles sur Steam, puis l'on a sélectionné un par un les jeux possédant assez de reviews (si possible) pour être étudié. Pour les jeux ayant le même Metacritic, c'était souvent le choix de la personne.

### **Pourquoi ce sujet ?**

- Alors, ici, c'est Jean-Charles qu'il faut pointer du doigt. (Seif)
- J'ai choisi ce sujet car je ne vais pas le cacher, je suis un joueur assez prononcé. J'hésitais entre un sujet sur la musique (un autre point que j'affectionne beaucoup) et les jeux-vidéo, puis j'ai entendu dans la salle un autre groupe avoir une idée traitant de la musique, j'ai donc changé de sujet. (Jean-Charles)

### **Pourquoi avoir autant de scripts différents ?**

- Il est vrai que ce n'était pas nécessaire d'avoir autant de scripts. Cependant, on voulait séparer notre travail, donc faire plusieurs scripts pour nos travaux était plus facile même si l'on reconnaît, particulièrement sur le script de l'analyse de sentiment et celui des mesures d'évaluations, que l'on pouvait les fusionner. Après, étant donné tout ce qu'on a fait, il est probable que ce soit plus clair ainsi plutôt que tout fusionné en un seul script.

### **Pourquoi avoir utilisé PySide6 ?**

- J'apprenais déjà à l'utiliser pour des petits projets personnels. Ce projet a été un moyen d'apprendre des nouvelles fonctionnalités. Ce n'est sûrement pas le plus optimal mais ça « fonctionne ». (Jean-Charles)