Projet Analyse linguistique : prédiction de la note attribuée aux films par la presse, à partir de leur commentaire sur AlloCiné.

Timothée Templier et Paul Le Breton

1. Introduction

Ce projet vise à développer un algorithme de prédiction de la note attribuée à un film, en utilisant des techniques de traitement du langage naturel (NLP) et d'apprentissage automatique. L'objectif est d'explorer dans quelle mesure les critiques de la presse peuvent fournir des indices permettant d'estimer la note qu'ils ont attribué au film.
Pour ce faire, l'approche s'appuie sur des techniques NLP afin d'analyser et d'extraire des informations clés des commentaires, telles que les sentiments, les expressions récurrentes et les thématiques dominantes. Ces éléments textuels serviront ensuite à entraîner un modèle de machine learning capable d'établir une corrélation entre les caractéristiques des critiques et les notes attribuées.
L'algorithme ainsi créé devrait être en mesure de prédire la note à partir de la critique, offrant un aperçu du potentiel du NLP pour l'analyse de sentiment appliquée au domaine du cinéma.

2. Collecte des données

Pour la collecte des données, nous avons choisi de récupéré l'ensemble des films classés dans la catégorie France d'AlloCiné (ce sont les films sortis en salle en France).
Les films collectés sont donc ceux ayant reçu au moins une note et un commentaire de la part de la presse.
Ces informations sont ensuite organisées dans un DataFrame pour une vue d'ensemble et sont enregistrées dans un fichier pour pouvoir être utilisées plus facilement plus tard.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
from concurrent.futures import ThreadPoolExecutor, as_completed

def get_movies(page):
    url = f"https://www.allocine.fr/films/pays-5001/?page={page}"  # 
URL pour les films sortis en France
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) 
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 
Safari/537.36"
    }
    response = requests.get(url, headers=headers)
    soup = BeautifulSoup(response.text, "html.parser")
```

```
    movies = []
    # Récupération des films
    for film in soup.find_all("li", class_="mdl"):
        title_tag = film.find("a", class_="meta-title-link")
        title = title_tag.text.strip() if title_tag else "N/A"

        year_tag = film.find("span", class_="date")
        year = year_tag.text.strip().rsplit(' ', 1)[-1] if year_tag
else "N/A"

        rating_global_tag = film.find("div", class_="rating-holder")
        rating_global = None
        if rating_global_tag:
            star_eval_note_tag = rating_global_tag.find("span",
class_="stareval-note")
            rating_global =
float(star_eval_note_tag.text.strip().replace(',', '.')) if
star_eval_note_tag else None

        # Si le film n'a pas été noté par la presse sur AlloCiné, le
film ne sera pas récupéré
        if rating_global is None:
            continue

        # Récupérer les commentaires de la presse
        film_id = title_tag['href'].split('=')[-1].split('.')[0]
        comments =
get_press_comments(f"https://www.allocine.fr/film/fichefilm-
{film_id}/critiques/presse/")

        movies.append({
            "title": title,
            "year": year,
            "rating_global": rating_global,
            "comments": comments,
            "page": page  # Ajout du numéro de page
        })

    return movies

def get_press_comments(film_url):
    """Récupère les commentaires de la presse pour un film donné."""
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107
Safari/537.36"
    }
    response = requests.get(film_url, headers=headers)
    soup = BeautifulSoup(response.text, "html.parser")
```

```
    comments = []
    press_reviews = soup.find_all("div", class_="reviews-press-
comment")

    for review in press_reviews:
        items = review.find_all("div", class_="item")
        for item in items:
            publication_tag = item.find("h2").find("span")
            publication = publication_tag.text.strip() if
publication_tag else "N/A"

            comment_tag = item.find("p", class_="text")
            comment = comment_tag.text.strip() if comment_tag else
"N/A"

            rating_tag = item.find("div", class_="rating-mdl")
            rating = None
            if rating_tag and 'class' in rating_tag.attrs:
                rating_class = rating_tag['class']
                for cls in rating_class:
                    if cls.startswith('n'):
                        rating = int(cls[1:]) / 10
                        break

            comments.append({
                "publication": publication,
                "comment": comment,
                "rating": rating if rating is not None else "N/A"  #
Remplacer par "N/A" si rating est None
            })

    return comments

# Récupération des films pour un nombre maximum de pages
all_movies = []
max_pages = 1352  # 1352 correspond au nombre de pages, pour les films
sortis en France, au moment de la récupération des données
(29/10/2024)

# Utilisation de ThreadPoolExecutor pour le parallélisme
with ThreadPoolExecutor(max_workers=10) as executor:
    future_to_page = {executor.submit(get_movies, page): page for page
in range(1, max_pages + 1)}

    for future in as_completed(future_to_page):
        page = future_to_page[future]
        try:
            movies = future.result()
            if movies:  # Ajout des films récupérés
```

```
                all_movies.extend(movies)
        except Exception as e:
            print(f"Erreur lors de la récupération de la page {page}:
{e}")

# Création d'une liste pour stocker les données pour le DataFrame
data_for_df = []

# Remplissage de la liste avec les données
for movie in all_movies:
    for comment in movie['comments']:
        data_for_df.append({
            "Title": movie['title'],
            "Year": movie['year'],
            "Global Rating": movie['rating_global'],
            "Publication": comment['publication'],
            "Comment": comment['comment'],
            "Press Rating": comment['rating'],
            "Page": movie['page']  # Ajout du numéro de page
        })

# Création du DataFrame
df_movies = pd.DataFrame(data_for_df)

# Affichage du DataFrame
print(df_movies)

# Enregistrement du DataFrame
df_movies.to_csv("data_movies.csv", index=False, encoding='utf-8')
```

3 Exploration des données bruts

On obtient 6204 films et 98 305 commentaires distincts.

```
import pandas as pd
data_movies = pd.read_csv('data_movies.csv', encoding='UTF-8')

data_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98305 entries, 0 to 98304
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Title          98305 non-null  object
 1   Year           97942 non-null  float64
 2   Global Rating  98305 non-null  float64
 3   Publication    98305 non-null  object
 4   Comment        98303 non-null  object
```

```
 5   Press Rating   98305 non-null  float64
 6   Page           98305 non-null  int64
dtypes: float64(3), int64(1), object(3)
memory usage: 5.3+ MB
```

```
data_movies.shape[0]
```

```
98305
```

```
data_movies.head(10)
```

```
                Title    Year  Global Rating                      Publication  \
0  Iris et les hommes  2024.0            2.9                       20 Minutes
1  Iris et les hommes  2024.0            2.9  Dernières Nouvelles d'Alsace
2  Iris et les hommes  2024.0            2.9                  Femme Actuelle
3  Iris et les hommes  2024.0            2.9              Le Dauphiné Libéré
4  Iris et les hommes  2024.0            2.9                       Le Parisien
5  Iris et les hommes  2024.0            2.9                    Ouest France
6  Iris et les hommes  2024.0            2.9                      Sud Ouest
7  Iris et les hommes  2024.0            2.9              Franceinfo Culture
8  Iris et les hommes  2024.0            2.9                        La Croix
9  Iris et les hommes  2024.0            2.9                       Le Figaro

                                             Comment  Press Rating  Page
0  Laure Calamy est plus pétillante que jamais da...           4.0     4
1  Il faut un solide travail d’actrice et l’exubé...           4.0     4
2  Le ton malicieux, les surprises du récit donne...           4.0     4
3  Souriante et décomplexée, sans cynisme ni crua...           4.0     4
4  Trois ans après « Antoinette dans les Cévennes...           4.0     4
5            Une comédie gonflée portée avec malice.           4.0     4
6  Une comédie de mœurs joyeuse et inventive sur ...           4.0     4
```

```
7  Si le film ne tient pas ses promesses de bout ...           3.0
4
8  Mais la fantaisie d’Antoinette court toujours ...           3.0
4
9  Laure Calamy retrouve Caroline Vignal, réalisa...           3.0
4
```

Nous obtenons les colonnes suivantes :

- Title : Titre du film.
- Year : Année de sortie du film.
- Global Rating : Note globale attribuée au film par la presse, sur une échelle de 0 à 5 (c'est la moyenne de toutes les notes de la presse).
- Publication : Nom du média ou de la publication ayant publié un avis sur le film.
- Comment : Critique publié par le média concernant le film.
- Press Rating : Note attribuée par chaque média, avec des valeurs possibles par paliers de 0,5 allant de 0 à 5.
- Page : Numéro de la page d'AlloCiné d'où les informations ont été extraites.

```
data_movies.describe()
```

```
                Year  Global Rating  Press Rating          Page
count  97942.000000   98305.000000  98305.000000  98305.000000
mean    2012.694779       3.247146      3.241595    280.569401
std        7.852530       0.578670      1.056976    217.143546
min     1928.000000       1.000000      0.500000      1.000000
25%     2007.000000       2.900000      3.000000     98.000000
50%     2014.000000       3.300000      3.000000    232.000000
75%     2019.000000       3.600000      4.000000    427.000000
max     2024.000000       5.000000      5.000000   1303.000000
```

```
data_movies.columns
```

```
Index(['Title', 'Year', 'Global Rating', 'Publication', 'Comment',
       'Press Rating', 'Page'],
      dtype='object')
```

3.1 Analyse des notes

Notre objectif est de prédire la note attribuée par l'auteur d'un commentaire, en se basant uniquement sur le contenu de ce commentaire. Commençons donc par analyser l'ensemble des notes données par les commentateurs pour mieux comprendre leur évaluation.

```
data_movies['Press Rating'].describe()
```

```
count    98305.000000
mean         3.241595
std          1.056976
min          0.500000
```

```
25%           3.000000
50%           3.000000
75%           4.000000
max           5.000000
Name: Press Rating, dtype: float64
```

```
data_movies.groupby('Press Rating').agg({'Comment' : 'count'})
```

Press Rating	Comment
0.5	8
1.0	6537
1.5	21
2.0	17464
2.5	71
3.0	28485
3.5	200
4.0	36755
4.5	12
5.0	8750

Press Rating est une variable qualitative à 10 classes, dont les valeurs vont de 0.5 à 5. En regroupant les notes selon les valeurs de Press Rating, on constate que les valeurs avec des décimales (comme 0.5, 1.5, etc.) sont peu fréquentes. Utiliser un algorithme avec ces données déséquilibrées ne serait pas optimal pour identifier ces valeurs. C'est pourquoi nous avons décidé de regrouper ces 10 classes en 5 classes, allant de 1 à 5, selon le schéma suivant :

- 0.5 et 1 → 1
- 1.5 et 2 → 2
- 2.5 et 3 → 3
- 3.5 et 4 → 4
- 4.5 et 5 → 5

Cela permet de réduire le déséquilibre tout en maintenant la signification des valeurs.

3.2 Anayse des commentaires

Nous allons ici explorer le texte, pour se faire une idée des pré-traitements à effectuer.

```
data_movies['Comment'].values[:15]
```

```
array(['Laure Calamy est plus pétillante que jamais dans cette
fantaisie au ton très libre.',
       'Il faut un solide travail d’actrice et l’exubérance
naturellement expressive de la lumineuse Laure Calamy, mélange
d’émotion brute et de vulnérabilité, pour échapper à la banalité et à
la futilité. C’est ça le miracle toujours renouvelé de cette grande
actrice, qui exalte l’ordinaire avec un charme sûr de lui,
irrésistible.',
```

```
       "Le ton malicieux, les surprises du récit donnent un charme
ingénu à cette histoire dans l'ère du temps.",
       'Souriante et décomplexée, sans cynisme ni cruauté, jamais
inconfortable ni embarrassante, cette sympathique peinture de mœurs
féministe ne s'encombre d'aucune complexité, délibérément simpliste et
limpide.',
       'Trois ans après « Antoinette dans les Cévennes », la
réalisatrice Caroline Vignal embauche à nouveau Laure Calamy, mais
pour une histoire beaucoup plus urbaine. La comédienne, déchaînée,
offre un nouveau numéro très drôle, savoureux et mélancolique face à
une dizaine d'acteurs masculins très différents.',
       'Une comédie gonflée portée avec malice.',
       'Une comédie de mœurs joyeuse et inventive sur le désir au sein
du couple et le rôle des sites de rencontres.',
       "Si le film ne tient pas ses promesses de bout en bout, avec
quelques baisses de régime et l'impression par moments, de tourner en
rond, la réalisatrice tire néanmoins parti à plein de son sujet.",
       'Mais la fantaisie d'Antoinette court toujours dans les
péripéties d'Iris et Laure Calamy fait merveille (...).',
       'Laure Calamy retrouve Caroline Vignal, réalisatrice
d'Antoinette dans les Cévennes, dans une comédie sympathique quoiqu'un
peu vaine.',
       'La réussite de la comédie doit beaucoup au talent clownesque
de Laure Calamy, drôle et touchante à la fois.',
       "Moins efficace qu'Antoinette dans les Cévennes, le film n'en
est pas moins un divertissement enjoué et fantasque.",
       'Avec "Iris et les hommes", Caroline Vignal peine à retrouver
l'élan comique de son "Antoinette…" mais réussit le portrait de son
héroïne et sa nouvelle étude de l'amour et du désir.',
       "Si elle n'est pas toujours crédible, cette comédie de mœurs
est ponctuée de scènes jubilatoires.",
       "Si elle n'est pas toujours crédible, cette comédie de mœurs
est ponctuée de quelques séquences jubilatoires (...)."],
      dtype=object)
```

4 Pré-Traitement des données textuelles

4.1 Traitement

```
import pandas as pd
import re
import spacy
from unidecode import unidecode
from nltk.corpus import stopwords
from nltk.stem.snowball import FrenchStemmer
import nltk

# Télécharger les stopwords si nécessaire
# nltk.download('stopwords')
```

```
# Importation des mots à supprimer
with open('mots_a_supprimer.txt', encoding='utf-8') as fichier:
    mots_a_supprimer = [mot.strip().lower() for mot in
fichier.readlines()]

# Chargement du modèle de la langue française de spaCy
nlp = spacy.load("fr_core_news_sm")

# Initialiser le stemmer français
stemmer = FrenchStemmer()

def clean_text(data):
    # Définir les stop words
    stop_words = set(stopwords.words('french'))

    # Remplacer les valeurs NaN par des chaînes vides
    data['Comment'] = data['Comment'].fillna('')

    # **Lemmatisation** - Effectuée en premier sur le texte brut
    lemmatized_comments = []
    for comment in data['Comment']:
        if comment.strip() == '':
            lemmatized_comments.append('')
            continue

        doc = nlp(comment)
        lemmatized_text = ' '.join([token.lemma_ for token in doc if
not token.is_punct and not token.is_space])
        lemmatized_comments.append(lemmatized_text)

    data['Comment'] = lemmatized_comments

    # Nettoyage supplémentaire
    data['Comment'] = data['Comment'].apply(lambda x: x.replace("'",
"e ").replace("’", "e ").replace("’", "e "))

    # Supprimer les URLs
    data['Comment'] = data['Comment'].apply(lambda x: re.sub(r'\b\
w*(www|http|https|\.fr|\.com)\w*\b', '', str(x)))

    # Supprimer les chaînes "(...)"
    data['Comment'] = data['Comment'].str.replace(r'\(\.\.\.\)', '',
regex=True)

    # Supprimer les accents
    data['Comment'] = data['Comment'].apply(unidecode)

    # Supprimer les caractères spéciaux et les nombres
    data['Comment'] = data['Comment'].apply(lambda x: re.sub(r'[^a-zA-
```

```
Z\s]', '', x))

    # Mise en minuscule
    data['Comment'] = data['Comment'].str.lower()

    # Supprimer les mots vides (stopwords)
    data['Comment'] = data['Comment'].apply(lambda x: ' '.join(word
for word in x.split() if word not in stop_words))

    # Supprimer les mots de taille 1
    data['Comment'] = data['Comment'].apply(lambda x: ' '.join([word
for word in x.split() if len(word) > 1]))

    # Supprimer les mots qui sont dans la liste des mots à supprimer
    data['Comment'] = data['Comment'].apply(lambda x: ' '.join([word
for word in x.split() if word not in mots_a_supprimer]))

    # Supprimer les lignes où le commentaire est vide
    data = data[data['Comment'] != ''].copy()

    # Racinisation des commentaires nettoyés
    data['Comment_stem'] = data['Comment'].apply(lambda x: '
'.join([stemmer.stem(word) for word in x.split()]))

    return data

# Charger les données et appliquer le nettoyage
data_movies = pd.read_csv('data_movies.csv')
data_movies_clean = clean_text(data_movies)

# Enregistrer le DataFrame nettoyé dans un nouveau fichier CSV
data_movies_clean.to_csv('data_movies_clean.csv', index=False,
encoding='utf-8')
```

La fonction `clean_text` nettoie et prépare les commentaires textuels dans un DataFrame pour une analyse de texte. Elle suite les étapes suivantes :

- Lemmatisation : utilisation du module `spaCy` pour réduire les mots à leur forme de base.
- Nettoyage : suppression des apostrophes, des URLs, des caractères spéciaux, des accents et des caractères numériques.
- Filtrage : suppression des mots vides à l'aide du module `nltk`, ainsi que les mots présents dans une liste des mots à exclure (film, scene, scenes).
- Racinisation : application d'un stemming pour réduire les mots à leur racine.

Enfin, les commentaires nettoyés sont enregistrés dans un nouveau fichier .csv. On obtient 2 colonnes représentants les commentaires :

- `Comment` : qui sont les commentaires nettoyés et lemmatisés.
- `Comment_stem` : qui sont les commentaires nettoyés et racinisés.

```
import pandas as pd
data_movies_clean = pd.read_csv('data_movies_clean.csv')

data_movies_clean.head(15)
```

```
                  Title    Year  Global Rating                    
Publication  \
0   Iris et les hommes  2024.0            2.9                     20
Minutes
1   Iris et les hommes  2024.0            2.9  Dernières Nouvelles
d'Alsace
2   Iris et les hommes  2024.0            2.9                Femme
Actuelle
3   Iris et les hommes  2024.0            2.9            Le Dauphiné
Libéré
4   Iris et les hommes  2024.0            2.9                   Le
Parisien
5   Iris et les hommes  2024.0            2.9                  Ouest
France
6   Iris et les hommes  2024.0            2.9                    Sud
Ouest
7   Iris et les hommes  2024.0            2.9            Franceinfo
Culture
8   Iris et les hommes  2024.0            2.9                     La
Croix
9   Iris et les hommes  2024.0            2.9                    Le
Figaro
10  Iris et les hommes  2024.0            2.9        Le Journal du
Dimanche
11  Iris et les hommes  2024.0            2.9          Les Fiches du
Cinéma
12  Iris et les hommes  2024.0            2.9            Les
Inrockuptibles
13  Iris et les hommes  2024.0            2.9               Télé 2
semaines
14  Iris et les hommes  2024.0            2.9                  Télé
Loisirs

                                                 Comment  Press Rating
Page  \
0   laure calamy etre plus petillant jamais fantai...           4.0
4
1   falloir solide travail acteur exuberance natur...           4.0
4
2   malicieux surprise recit donner charme ingenu ...           4.0
4
3   souriante decomplexer sans cynisme ni cruauter...           4.0
4
4   trois an apres antoinette cevenne realisateur ...           4.0
4
```

```
5                        comedie gonfler porter malice           4.0
4
6   comedie moeur joyeux inventif desir sein coupl...           4.0
4
7   si tenir promesse bout bout quelque baisse reg...           3.0
4
8   fantaisie antoinette court toujours peripetie ...           3.0
4
9   laure calamy retrouver caroline vignal realisa...           3.0
4
10  reussite comedie devoir beaucoup talent clowne...           3.0
4
11  moins efficace antoinette cevenne etre moins d...           3.0
4
12  iris homme caroline vignal peine retrouver ela...           3.0
4
13  si etre toujours credible comedie moeur etre p...           3.0
4
14  si etre toujours credible comedie moeur etre p...           3.0
4
```

```
                                        Comment_stem
0   laur calamy etre plus petill jam fantais tre libr
1   falloir solid travail acteur exuber naturel ex...
2   malici surpris rec don charm ingenu histoir er...
3   souri decomplex san cynism ni cruaut jam incon...
4   trois an apre antoinet ceven realis carolin vi...
5                              comed gonfl port malic
6   comed moeur joyeux invent des sein coupl rol s...
7   si ten promess bout bout quelqu baiss regim im...
8   fantais antoinet court toujour peripet iris la...
9   laur calamy retrouv carolin vignal realis anto...
10  reussit comed devoir beaucoup talent clownesqu...
11  moin efficac antoinet ceven etre moin divert e...
12  iris homm carolin vignal pein retrouv elan com...
13  si etre toujour credibl comed moeur etre ponct...
14  si etre toujour credibl comed moeur etre ponct...
```

```
# Convertir Press Rating en variable catégorielle
data_movies_clean['Press Rating'] = pd.cut(data_movies_clean['Press
Rating'],bins=[0,1,2,3,4,5],
                          labels=[1,2,3,4,5])

data_movies_clean['Press Rating'].unique()
```

```
[4, 3, 2, 1, 5]
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

4.2 Analyse exploratoire

4.2.1 Nuage de mots des adjectifs

```
from wordcloud import WordCloud
from matplotlib import pyplot as plt
import spacy

nlp = spacy.load("fr_core_news_sm")

dico_reccurence = {}

# Extraire les adjectifs
for comment in data_movies_clean['Comment']:
    doc = nlp(comment)
    for token in doc:
        # Vérifier si le token est un adjectif
        if token.pos_ == 'ADJ':
            if token.text in dico_reccurence:
                dico_reccurence[token.text] += 1  # Utiliser la forme
originale pour la récurrence
            else:
                dico_reccurence[token.text] = 1

# Générer le nuage de mots pour les adjectifs
wordcloud = WordCloud(width=800,
                      height=400,
                      background_color='white',

max_words=200).generate_from_frequencies(dico_reccurence)

# Afficher le nuage de mots
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Ne pas afficher les axes
plt.title('Nuage de mots des adjectifs les plus fréquents')
plt.show()
```

4.2.2 Histogrammes de la taille des avis selon la note

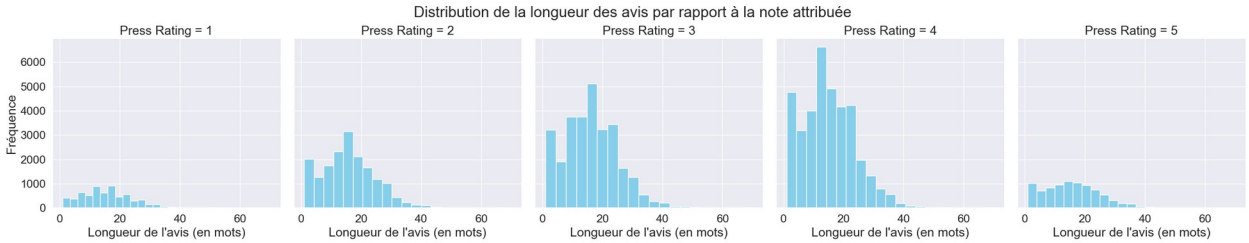
```
import seaborn as sns
import matplotlib.pyplot as plt

# Filtrer pour ne garder que les notes de 1 à 5 et créer une nouvelle
colonne 'comment_length' en comptant les mots
data_filtered = data_movies_clean[data_movies_clean['Press
Rating'].isin([1, 2, 3, 4, 5])].assign(comment_length=lambda x:
x['Comment'].str.split().str.len())

# Ajuster le style et la taille de la police pour une meilleure
visibilité
sns.set_theme(font_scale=1.5)

# Créer un graphique avec un histogramme de la longueur des avis,
réparti par note de presse (toutes les notes sur une seule ligne)
g = sns.FacetGrid(data_filtered, col='Press Rating', height=5,
col_wrap=5)
g.map(plt.hist, 'comment_length', bins=20, color='skyblue')

# Ajuster les axes et afficher le graphique
g.set_axis_labels("Longueur de l'avis (en mots)", "Fréquence")
g.figure.subplots_adjust(top=0.85)
g.figure.suptitle("Distribution de la longueur des avis par rapport à
la note attribuée")
plt.show()
```



La longueur des critiques ne semble pas être liée à l'appréciation et à la note attribuée au film de la part de la presse.

4.2.3 Notes moyennes attribuées, en fonction de la presse

```
tab = data_movies_clean
tab['Press Rating'] = tab['Press Rating'].astype(float)

comment_counts = tab['Publication'].value_counts()

publications_filtered = comment_counts[comment_counts > 3000].index

tab[tab['Publication'].isin(publications_filtered)].groupby('Publicati
on')['Press Rating'].agg(['mean']).round({'mean': 1})
```

Publication	mean
Le Journal du Dimanche	3.1
Le Monde	2.9
Le Parisien	3.5
Les Inrockuptibles	3.1
Libération	3.3
Positif	3.3
Première	2.9
Télérama	3.1

Ce tableau nous montre que Le Parisien semble être le plus "généreux", avec une note moyenne de 3.5, alors que Le Monde et Première sont les médias qui mettent les notes les plus basses en moyenne.

5. Tests des différents algorithmes de prédictions

5.1 Sur les commentaires lemmatisés

Séparation des données

Les données en apprentissage automatique sont généralement séparées en trois jeux :

- **Entraînement** : données destinées à l'apprentissage du modèle.
- **Validation** : données destinées à une évaluation intermédiaire du modèle pour permettre l'ajustement de ses hyperparamètres. Une fois les hyperparamètres du

modèle arrêtés, on peut le ré-entraîner sur l'ensemble des données (entraînement + validation) avant de le tester sur le jeu de test.

- **Test** : données destinées exclusivement à l'évaluation finale (à réaliser une fois uniquement) du modèle choisi retenu. Elles ne doivent sous aucune forme servir à la conception du modèle.

```
from sklearn.model_selection import train_test_split
import numpy as np

# Discrétiser les notes en classes (par exemple de 1 à 5)
bins = np.arange(1, 6,1)  # Définir les bacs (0 à 5)
data_movies_clean['Press Rating'] =
np.digitize(data_movies_clean['Press Rating'], bins) # Convertir en
classes

# print(data_movies_clean['Press Rating'].unique())

X_Entrainement, X_test, y_Entrainement, y_test =
train_test_split(data_movies_clean['Comment'],

data_movies_clean['Press Rating'],
                                                    train_size=0.75,
                                                    random_state=5)

X_train , X_validity , y_train , y_validity =
train_test_split(X_Entrainement,
                                                    y_Entrainement,
                                                    train_size=0.60,
                                                    random_state=5)


print(f"Shape X_train : {X_train.shape},  Shape y_train :
{y_train.shape}")
print(f"Shape X_validity : {X_validity.shape},  Shape y_validity :
{y_validity.shape}")
print(f"Shape X_test : {X_test.shape},  Shape y_test :
{X_test.shape}")
```

```
Shape X_train : (44197,),  Shape y_train : (44197,)
Shape X_validity : (29465,),  Shape y_validity : (29465,)
Shape X_test : (24555,),  Shape y_test : (24555,)
```

5.1.2 Calcul des valeurs descripteurs

Étant donné la taille de notre corpus, avec plus de 98 000 commentaires, nous allons tester deux types de vectorisations :

- **1) TF-IDF :** cette méthode mesure l'importance relative des termes dans un document par rapport à leur fréquence dans l'ensemble du corpus. Elle est simple à implémenter et aide à mettre en valeur les mots spécifiques à chaque document.
- **2) Comptage de bi-grammes :** cette méthode va nous permettre de capturer les relations entre mots adjacents, ce qui va être particulièrement pertinent pour notre jeu de données constitué de phrases courtes. Cette vectorisation aide à identifier des contextes immédiats entre mots.

Décompte bi-grammes

```
data_movies_clean.groupby('Title').agg({'Comment':'count'}).mean()
```

```
Comment    15.831238
dtype: float64
```

En moyenne, on a ≃16 commentaires par film. Pour éviter un vocabulaire dédié à un seul film, nous allons ignorer les termes présents dans moins de 15 documents du corpus.

```
from sklearn.feature_extraction.text import CountVectorizer

vect_count_bigrams = CountVectorizer(min_df=16,
ngram_range=(1,2),strip_accents='unicode',max_df=0.8).fit(X_train)
X_train_vectorized_count_bigrams =
vect_count_bigrams.transform(X_train)
X_valid_vectorized_count_bigrams =
vect_count_bigrams.transform(X_validity)
X_test_vectorized_count_bigrams = vect_count_bigrams.transform(X_test)

len(vect_count_bigrams.get_feature_names_out())
```

```
7416
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

vect_tfidf =
TfidfVectorizer(min_df=15,strip_accents='unicode',max_df=0.8).fit(X_tr
ain)
len(vect_tfidf.get_feature_names_out()),
len(vect_tfidf.get_feature_names_out())
```

```
(5708, 5708)
```

```
X_train_vectorized_tfidf = vect_tfidf.transform(X_train)
X_valid_vectorized_tfidf = vect_tfidf.transform(X_validity)
X_test_vectorized_tfidf = vect_tfidf.transform(X_test)
```

5.1.2 Test des modèles

5.1.2.1 Modèles de réferences faibles

Le modèle DummyClassifier est un modèle de base simple qui sert souvent de référence pour évaluer la performance d'autres modèles plus complexes. Il est utilisé principalement pour établir un point de comparaison et vérifier si un modèle plus sophistiqué offre réellement une amélioration significative.

Le prédicteur respecte la distribution des classes dans les données d'entraînement.

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, classification_report, confusion_matrix

from sklearn.dummy import DummyClassifier

print("**********************************************************")
print("Pour vecteur compteur bi-grammes :")

random_prop_class =
DummyClassifier(strategy='stratified').fit(X_train_vectorized_count_bi
grams,

y_train)
predictions_valid =
random_prop_class.predict(X_valid_vectorized_count_bigrams)

conf_mat = confusion_matrix(y_validity, predictions_valid)
# print(f'conf_mat : {conf_mat}')
print(f"Accuracy : {accuracy_score(y_validity, predictions_valid)}")

print("**********************************************************")
print("Pour vecteur TF-IDF :")
random_prop_class =
DummyClassifier(strategy='stratified').fit(X_train_vectorized_tfidf,

y_train)
predictions_valid =
random_prop_class.predict(X_valid_vectorized_tfidf)

conf_mat = confusion_matrix(y_validity, predictions_valid)
# print(f'conf_mat : {conf_mat}')
print(f"Accuracy : {accuracy_score(y_validity, predictions_valid)}")
```

```
**********************************************************
Pour vecteur compteur bi-grammes :
Accuracy : 0.2704225352112676
**********************************************************
Pour vecteur TF-IDF :
Accuracy : 0.26706261666383846
```

5.1.2.2 RandomForest

Cas TF-IDF :

```
from sklearn.ensemble import RandomForestClassifier

# Modèle de classification
model = RandomForestClassifier()
model.fit(X_train_vectorized_tfidf, y_train)

# Prédictions
y_pred = model.predict(X_valid_vectorized_tfidf)

# Évaluation
print(classification_report(y_validity, y_pred))
print("Accuracy:", accuracy_score(y_validity, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.47      0.08      0.13      1904
           2       0.46      0.23      0.30      5313
           3       0.41      0.53      0.46      8556
           4       0.52      0.72      0.60     11088
           5       0.51      0.03      0.05      2604

    accuracy                           0.47     29465
   macro avg       0.47      0.31      0.31     29465
weighted avg       0.47      0.47      0.43     29465

Accuracy: 0.4712031223485491
```

```
confusion_matrix(y_validity, y_pred)
```

```
array([[ 148,  480,  596,  675,    5],
       [ 112, 1200, 2629, 1370,    2],
       [  29,  641, 4494, 3382,   10],
       [  22,  248, 2797, 7971,   50],
       [   4,   37,  479, 2013,   71]], dtype=int64)
```

En examinant la matrice de confusion, on observe que le modèle a des performances de prédiction très faibles.
Par exemple, pour un commentaire noté à 1 étoile, le modèle prédit 4 étoiles dans 675 cas, ce qui est incohérent.
Un modèle performant devrait prédire des notes plus proches de la note réelle.

Essayons d'optimiser les hyper-paramètres du modèle pour regarder si cela améliore le modèle :

Remarque : Très long, à ne pas lancer.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [100, 200, 300],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 5, 10],
    'max_features': ['log2', 'sqrt'],
    'class_weight': ['balanced', None]
}

grid_search =
GridSearchCV(estimator=RandomForestClassifier(random_state=42),
param_grid=param_grid, cv=3)
grid_search.fit(X_train_vectorized_tfidf, y_train)
print("Meilleurs paramètres :", grid_search.best_params_)
```

```
Meilleurs paramètres : {'class_weight': 'balanced', 'max_features':
'log2', 'min_samples_leaf': 1, 'min_samples_split': 10,
'n_estimators': 300}
```

```python
# Récupérer les résultats et s'assurer que 'mean_test_score' est bien
un type numérique
results = pd.DataFrame(grid_search.cv_results_)
results['mean_test_score'] = pd.to_numeric(results['mean_test_score'],
errors='coerce')

# Trier par 'mean_test_score' en ordre décroissant
results = results.sort_values(by='mean_test_score', ascending=False)

# Afficher les 15 meilleures configurations de paramètres
for i in range(15):
    print(f"Paramètre : {results['params'].iloc[i]}, Score :
{results['mean_test_score'].iloc[i]}")
```

```
Paramètre : {'class_weight': 'balanced', 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300},
Score : 0.5006931036189911
Paramètre : {'class_weight': 'balanced', 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200},
Score : 0.4977969206396356
Paramètre : {'class_weight': 'balanced', 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100},
Score : 0.49269765829991585
Paramètre : {'class_weight': None, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200},
Score : 0.49269765829991585
Paramètre : {'class_weight': None, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300},
Score : 0.4925738897965246
```

```
Paramètre : {'class_weight': None, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300},
Score : 0.49220258428635083
Paramètre : {'class_weight': None, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200},
Score : 0.4900242586266647
Paramètre : {'class_weight': None, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100},
Score : 0.4897767216198821
Paramètre : {'class_weight': None, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100},
Score : 0.4868557849398485
Paramètre : {'class_weight': 'balanced', 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300},
Score : 0.48678152383781376
Paramètre : {'class_weight': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300},
Score : 0.4858656369127184
Paramètre : {'class_weight': 'balanced', 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200},
Score : 0.48542007030051
Paramètre : {'class_weight': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300},
Score : 0.48527154809644046
Paramètre : {'class_weight': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200},
Score : 0.4848754888855884
Paramètre : {'class_weight': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200},
Score : 0.48467745928016237
```

```python
# Modèle de classification
model = RandomForestClassifier(class_weight = 'balanced',
max_features='log2',min_samples_leaf = 1, min_samples_split= 10,
n_estimators= 300 )
model.fit(X_train_vectorized_count_bigrams, y_train)

# Prédictions
y_pred = model.predict(X_valid_vectorized_count_bigrams)

# Évaluation
print(classification_report(y_validity, y_pred))
print("Accuracy:", accuracy_score(y_validity, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.30      0.31      0.30      1904
           2       0.40      0.40      0.40      5313
           3       0.49      0.37      0.42      8556
           4       0.54      0.71      0.61     11088
```

```
           5       0.37      0.15      0.22      2604

    accuracy                           0.48     29465
   macro avg       0.42      0.39      0.39     29465
weighted avg       0.47      0.48      0.46     29465

Accuracy: 0.47985745800101814

confusion_matrix(y_validity, y_pred)

array([[ 598,  577,  257,  443,   29],
       [ 671, 2144, 1344, 1111,   43],
       [ 368, 1523, 3145, 3412,  108],
       [ 316,  884, 1527, 7850,  511],
       [  69,  213,  157, 1763,  402]], dtype=int64)
```

Le modèle a été amélioré, mais il y a encore de nombreuses incohérences.

CAS Bi-grammes :

```
# Modèle de classification
model = RandomForestClassifier(class_weight = 'balanced',
max_features='log2',min_samples_leaf = 1, min_samples_split= 10,
n_estimators= 300 )
model.fit(X_train_vectorized_tfidf, y_train)

# Prédictions
y_pred = model.predict(X_valid_vectorized_tfidf)

# Évaluation
print(classification_report(y_validity, y_pred))
print("Accuracy:", accuracy_score(y_validity, y_pred))

              precision    recall  f1-score   support

           1       0.32      0.26      0.29      1904
           2       0.39      0.35      0.37      5313
           3       0.47      0.39      0.43      8556
           4       0.53      0.73      0.61     11088
           5       0.38      0.09      0.15      2604

    accuracy                           0.48     29465
   macro avg       0.42      0.36      0.37     29465
weighted avg       0.46      0.48      0.45     29465

Accuracy: 0.47659935516714746

confusion_matrix(y_validity, y_pred)

array([[ 490,  545,  316,  537,   16],
       [ 456, 1836, 1688, 1310,   23],
```

```
       [ 270, 1316, 3332, 3571,   67],
       [ 254,  813, 1592, 8145,  284],
       [  54,  198,  183, 1929,  240]], dtype=int64)
```

5.1.2.3 LogisticRegression

CAS Bi-grammes :

```
from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression(solver='lbfgs',
max_iter=1000).fit(X_train_vectorized_count_bigrams, y_train)
predictions_valid = model_lr.predict(X_valid_vectorized_count_bigrams)
accuracy_score(y_validity, predictions_valid)

0.46886136093670455

confusion_matrix(y_validity, predictions_valid)

array([[ 478,  680,  412,  305,   29],
       [ 538, 1883, 2046,  797,   49],
       [ 180, 1184, 4143, 2869,  180],
       [ 117,  406, 2955, 6844,  766],
       [  21,   69,  477, 1579,  458]], dtype=int64)
```

CAS TF-IDF :

```
from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression(solver='lbfgs',
max_iter=1000).fit(X_train_vectorized_tfidf, y_train)
predictions_valid = model_lr.predict(X_valid_vectorized_tfidf)
accuracy_score(y_validity, predictions_valid)

0.49845579501103005

confusion_matrix(y_validity, predictions_valid)

array([[ 310,  722,  447,  414,   11],
       [ 222, 1810, 2270,  999,   12],
       [  52,  879, 4368, 3222,   35],
       [  30,  252, 2634, 7961,  211],
       [   4,   36,  405, 1917,  242]], dtype=int64)
```

Pour la LogisticRegression, la vectorisation TF-IDF a de meilleurs résultats. On a moins d'erreurs abérantes que pour un modèle de RandomForest.

5.1.2.4 LSTM :

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Prétraitement des données
X = data_movies_clean['Comment'].values
y = data_movies_clean['Press Rating'].values

# Convertir les notes en classes
classes = np.unique(y)
y_class = np.array([np.where(classes == label)[0][0] for label in y])
num_classes = len(classes)
y_categorical = to_categorical(y_class, num_classes=num_classes)

# Tokenisation et padding
max_words = 2000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
sequences = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(sequences, maxlen=max_len)

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_pad,
y_categorical, test_size=0.2, random_state=42)

# Création du modèle
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=100,
input_length=max_len))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(64))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))

# Compiler le modèle
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Entraîner le modèle avec Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
model.fit(X_train, y_train, epochs=30, batch_size=200,
validation_split=0.2, callbacks=[early_stopping])

# Évaluation du modèle
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Précision du modèle : {accuracy * 100:.2f}%")
```

```
Epoch 1/30

c:\Users\timot\anaconda3\Lib\site-packages\keras\src\layers\core\
embedding.py:90: UserWarning: Argument `input_length` is deprecated.
Just remove it.
  warnings.warn(

315/315 ━━━━━━━━━━━━━━━━━━━━ 178s 551ms/step - accuracy: 0.4025 -
loss: 1.3800 - val_accuracy: 0.4868 - val_loss: 1.1939
Epoch 2/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 161s 510ms/step - accuracy: 0.5020 -
loss: 1.1584 - val_accuracy: 0.4917 - val_loss: 1.1736
Epoch 3/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 169s 536ms/step - accuracy: 0.5133 -
loss: 1.1327 - val_accuracy: 0.4950 - val_loss: 1.1726
Epoch 4/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 166s 526ms/step - accuracy: 0.5226 -
loss: 1.1089 - val_accuracy: 0.4954 - val_loss: 1.1692
Epoch 5/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 166s 528ms/step - accuracy: 0.5344 -
loss: 1.0762 - val_accuracy: 0.4938 - val_loss: 1.1772
Epoch 6/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 159s 505ms/step - accuracy: 0.5429 -
loss: 1.0569 - val_accuracy: 0.4928 - val_loss: 1.1855
Epoch 7/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 157s 499ms/step - accuracy: 0.5571 -
loss: 1.0377 - val_accuracy: 0.4899 - val_loss: 1.1981
614/614 ━━━━━━━━━━━━━━━━━━━━ 27s 44ms/step - accuracy: 0.5027 - loss:
1.1568
Précision du modèle : 50.23%
```

```
# Prédictions sur l'ensemble de test
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Obtenir les classes
prédites
y_test_classes = np.argmax(y_test, axis=1)  # Obtenir les classes
réelles

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)

print("Matrice de confusion :\n", conf_matrix)
```

```
614/614 ━━━━━━━━━━━━━━━━━━━━ 30s 49ms/step
Matrice de confusion :
 [[ 127  663  186  314    4]
 [  91 1432 1015  883    5]
 [  24  755 2236 2704   13]
```

```
 [  38  277 1104 5972   64]
 [   5   49  120 1462  101]]
```

Comme tout à l'heure, la matrice de confusion nous présente des incohérences, le modèle ne prédit pas assez bien les bonnes classes et les erreurs s'éloignent trop de la note réelle.

5.2 Sur les commentaires racinisés

Séparation des données

```
from sklearn.model_selection import train_test_split
import numpy as np


# Discrétiser les notes en classes (par exemple de 1 à 5)
bins = np.arange(1, 6,1)  # Définir les bacs (0 à 5)
data_movies_clean['Press Rating'] =
np.digitize(data_movies_clean['Press Rating'], bins) # Convertir en
classes

# print(data_movies_clean['Press Rating'].unique())

X_Entrainement, X_test, y_Entrainement, y_test =
train_test_split(data_movies_clean['Comment_stem'],

data_movies_clean['Press Rating'],
                                                    train_size=0.75,
                                                    random_state=5)


X_train , X_validity , y_train , y_validity =
train_test_split(X_Entrainement,
                                                    y_Entrainement,
                                                    train_size=0.60,
                                                    random_state=5)


print(f"Shape X_train : {X_train.shape},  Shape y_train :
{y_train.shape}")
print(f"Shape X_validity : {X_validity.shape},  Shape y_validity :
{y_validity.shape}")
print(f"Shape X_test : {X_test.shape},  Shape y_test :
{X_test.shape}")

Shape X_train : (44197,),  Shape y_train : (44197,)
Shape X_validity : (29465,),  Shape y_validity : (29465,)
Shape X_test : (24555,),  Shape y_test : (24555,)
```

5.2.1 Calcul des valeurs descripteurs

Décompte bi-grammes

```
from sklearn.feature_extraction.text import CountVectorizer

vect_count_bigrams = CountVectorizer(min_df=16,
ngram_range=(1,2),strip_accents='unicode',max_df=0.8).fit(X_train)
X_train_vectorized_count_bigrams =
vect_count_bigrams.transform(X_train)
X_valid_vectorized_count_bigrams =
vect_count_bigrams.transform(X_validity)
X_test_vectorized_count_bigrams = vect_count_bigrams.transform(X_test)
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

vect_tfidf =
TfidfVectorizer(min_df=15,strip_accents='unicode',max_df=0.8).fit(X_tr
ain)
X_train_vectorized_tfidf = vect_tfidf.transform(X_train)
X_valid_vectorized_tfidf = vect_tfidf.transform(X_validity)
X_test_vectorized_tfidf = vect_tfidf.transform(X_test)
```

5.2.2 Test de modèles

5.2.2.1 Modèle de références faibles

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, classification_report, confusion_matrix

from sklearn.dummy import DummyClassifier


print("*********************************************************")
print("Pour vecteur compteur bi-grammes :")

random_prop_class =
DummyClassifier(strategy='stratified').fit(X_train_vectorized_count_bi
grams,

y_train)
predictions_valid =
random_prop_class.predict(X_valid_vectorized_count_bigrams)

conf_mat = confusion_matrix(y_validity, predictions_valid)
# print(f'conf_mat : {conf_mat}')
print(f"Accuracy : {accuracy_score(y_validity, predictions_valid)}")

print("*********************************************************")
print("Pour vecteur TF-IDF :")
random_prop_class =
DummyClassifier(strategy='stratified').fit(X_train_vectorized_tfidf,
```

```
y_train)
predictions_valid =
random_prop_class.predict(X_valid_vectorized_tfidf)

conf_mat = confusion_matrix(y_validity, predictions_valid)
# print(f'conf_mat : {conf_mat}')
print(f"Accuracy : {accuracy_score(y_validity, predictions_valid)}")
```

```
**********************************************************
Pour vecteur compteur bi-grammes :
Accuracy : 0.26981164092991683
**********************************************************
Pour vecteur TF-IDF :
Accuracy : 0.2701170880705922
```

5.2.2.2 RandomForest

Cas TF-IDF :

```
from sklearn.ensemble import RandomForestClassifier

# Modèle de classification
model = RandomForestClassifier(class_weight = 'balanced',
max_features='log2',min_samples_leaf = 1, min_samples_split= 10,
n_estimators= 300 )
model.fit(X_train_vectorized_count_bigrams, y_train)

# Prédictions
y_pred = model.predict(X_valid_vectorized_count_bigrams)

# Évaluation
print(classification_report(y_validity, y_pred))
print("Accuracy:", accuracy_score(y_validity, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.32      0.32      0.32      1904
           2       0.40      0.40      0.40      5313
           3       0.49      0.36      0.42      8556
           4       0.54      0.73      0.62     11088
           5       0.35      0.13      0.19      2604

    accuracy                           0.48     29465
   macro avg       0.42      0.39      0.39     29465
weighted avg       0.47      0.48      0.46     29465

Accuracy: 0.4839300865433565
```

```
confusion_matrix(y_validity, y_pred)
```

```
array([[ 602,  565,  283,  428,   26],
       [ 652, 2120, 1385, 1119,   37],
       [ 326, 1514, 3121, 3484,  111],
       [ 257,  856, 1443, 8067,  465],
       [  58,  200,  144, 1853,  349]], dtype=int64)
```

CAS Bi-grammes :

```
# Modèle de classification
model = RandomForestClassifier(class_weight = 'balanced',
max_features='log2',min_samples_leaf = 1, min_samples_split= 10,
n_estimators= 300 )
model.fit(X_train_vectorized_tfidf, y_train)

# Prédictions
y_pred = model.predict(X_valid_vectorized_tfidf)

# Évaluation
print(classification_report(y_validity, y_pred))
print("Accuracy:", accuracy_score(y_validity, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.34      0.25      0.29      1904
           2       0.39      0.35      0.37      5313
           3       0.47      0.39      0.43      8556
           4       0.53      0.74      0.62     11088
           5       0.37      0.08      0.13      2604

    accuracy                           0.48     29465
   macro avg       0.42      0.36      0.37     29465
weighted avg       0.46      0.48      0.45     29465

Accuracy: 0.47945019514678433
```

```
confusion_matrix(y_validity, y_pred)
```

```
array([[ 475,  551,  332,  533,   13],
       [ 437, 1847, 1674, 1334,   21],
       [ 234, 1331, 3342, 3587,   62],
       [ 198,  818, 1563, 8252,  257],
       [  49,  193,  166, 1985,  211]], dtype=int64)
```

On obtient ici des résultats équivalents pour les deux types de vectorisation.

5.2.2.3 LogisticRegression

CAS Bi-grammes :

```
from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression(solver='lbfgs',
max_iter=1000).fit(X_train_vectorized_count_bigrams, y_train)
predictions_valid = model_lr.predict(X_valid_vectorized_count_bigrams)
accuracy_score(y_validity, predictions_valid)
```

```
0.46886136093670455
```

```
confusion_matrix(y_validity, predictions_valid)
```

```
array([[ 492,  674,  423,  288,   27],
       [ 561, 1881, 2075,  748,   48],
       [ 174, 1152, 4176, 2889,  165],
       [ 120,  441, 2908, 6888,  731],
       [  20,   64,  433, 1638,  449]], dtype=int64)
```

CAS TF-IDF :

```
from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression(solver='lbfgs',
max_iter=1000).fit(X_train_vectorized_tfidf, y_train)
predictions_valid = model_lr.predict(X_valid_vectorized_tfidf)
accuracy_score(y_validity, predictions_valid)
```

```
0.49845579501103005
```

```
confusion_matrix(y_validity, predictions_valid)
```

```
array([[ 326,  729,  410,  427,   12],
       [ 255, 1857, 1827, 1359,   15],
       [  61,  905, 3750, 3803,   37],
       [  34,  262, 2049, 8535,  208],
       [   4,   37,  216, 2108,  239]], dtype=int64)
```

La vectorisation en bi-grammes prédit mieux.
On obtient cette fois-ci une meilleure pédiction que pour le modèle de RandomForest.

5.2.2.4 LSTM :

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# Prétraitement des données
X = data_movies_clean['Comment'].values
```

```
y = data_movies_clean['Press Rating'].values

# Convertir les notes en classes
classes = np.unique(y)
y_class = np.array([np.where(classes == label)[0][0] for label in y])
num_classes = len(classes)
y_categorical = to_categorical(y_class, num_classes=num_classes)

# Tokenisation et padding
max_words = 2000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)
sequences = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(sequences, maxlen=max_len)

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_pad,
y_categorical, test_size=0.2, random_state=42)

# Création du modèle
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=100,
input_length=max_len))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(64))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))

# Compiler le modèle
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Entraîner le modèle avec Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
model.fit(X_train, y_train, epochs=30, batch_size=200,
validation_split=0.2, callbacks=[early_stopping])

# Évaluation du modèle
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Précision du modèle : {accuracy * 100:.2f}%")
```

```
c:\Users\timot\anaconda3\Lib\site-packages\keras\src\layers\core\
embedding.py:90: UserWarning: Argument `input_length` is deprecated.
Just remove it.
  warnings.warn(
```

```
Epoch 1/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 163s 504ms/step - accuracy: 0.4037 -
loss: 1.3900 - val_accuracy: 0.4819 - val_loss: 1.1939
Epoch 2/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 150s 477ms/step - accuracy: 0.5035 -
loss: 1.1600 - val_accuracy: 0.4945 - val_loss: 1.1693
Epoch 3/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 168s 533ms/step - accuracy: 0.5102 -
loss: 1.1314 - val_accuracy: 0.4928 - val_loss: 1.1746
Epoch 4/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 160s 507ms/step - accuracy: 0.5210 -
loss: 1.1096 - val_accuracy: 0.4948 - val_loss: 1.1656
Epoch 5/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 157s 499ms/step - accuracy: 0.5375 -
loss: 1.0739 - val_accuracy: 0.4955 - val_loss: 1.1727
Epoch 6/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 168s 532ms/step - accuracy: 0.5453 -
loss: 1.0575 - val_accuracy: 0.4910 - val_loss: 1.1853
Epoch 7/30
315/315 ━━━━━━━━━━━━━━━━━━━━ 181s 576ms/step - accuracy: 0.5512 -
loss: 1.0371 - val_accuracy: 0.4880 - val_loss: 1.1960
614/614 ━━━━━━━━━━━━━━━━━━━━ 31s 51ms/step - accuracy: 0.4974 - loss:
1.1580
Précision du modèle : 49.74%
```

```python
# Prédictions sur l'ensemble de test
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Obtenir les classes
prédites
y_test_classes = np.argmax(y_test, axis=1)  # Obtenir les classes
réelles

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)

print("Matrice de confusion :\n", conf_matrix)
```

```
614/614 ━━━━━━━━━━━━━━━━━━━━ 27s 44ms/step
Matrice de confusion :
 [[ 205  545  289  252    3]
 [ 156 1151 1565  549    5]
 [  31  586 2910 2196    9]
 [  47  264 1660 5426   58]
 [   7   44  222 1386   78]]
```

On obtient ici un meilleur modèle, cependant, on peut voir qu'il a du mal à prédire les classes extrêmes 1 et 5.

6. Conclusion et choix du modèle final

Après avoir comparé les résultats des différents modèles testés, nous avons opté pour le modèle **LogisticRegression**, qui utilise la vectorisation par **comptage de bi-grammes**, appliquée aux **commentaires racinisés**.

Ce choix repose sur la bonne performance du modèle ainsi que sa capacité à maintenir une cohérence dans les classes prédictes. L'analyse de la matrice de confusion montre que, dans les cas où le modèle fait des erreurs, la note prédite reste généralement proche de la valeur réelle, ce qui suggère une bonne robustesse du modèle et une capacité à faire des prédictions raisonnablement précises.

Entraînement du modèle final :

rappel :

X_Entrainement, X_test, y_Entrainement, y_test = train_test_split(data_movies_clean['Comment_stem'], data_movies_clean['Press Rating'], train_size=0.75, random_state=5)

```
X_Entrainement, X_test, y_Entrainement, y_test =
train_test_split(data_movies_clean['Comment_stem'],

data_movies_clean['Press Rating'],
                                                    train_size=0.75,
                                                    random_state=5)

from sklearn.feature_extraction.text import CountVectorizer

vect_count_bigrams = CountVectorizer(min_df=16,
ngram_range=(1,2),strip_accents='unicode',max_df=0.8).fit(X_Entraineme
nt)
X_train_vectorized_count_bigrams =
vect_count_bigrams.transform(X_Entrainement)
X_test_vectorized_count_bigrams = vect_count_bigrams.transform(X_test)

from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression(solver='lbfgs',
max_iter=1000).fit(X_train_vectorized_count_bigrams, y_Entrainement)
predictions = model_lr.predict(X_test_vectorized_count_bigrams)
accuracy_score(y_test, predictions)
```

```
0.48250865404194665
```

```
confusion_matrix(y_test, predictions)
```

```
array([[ 422,  649,  331,  242,   13],
       [ 470, 1694, 1630,  573,   36],
       [ 141,  985, 3349, 2477,  145],
```

```
       [  71,  320, 2263, 5958,  606],
       [  23,   54,  331, 1345,  427]], dtype=int64)
```

```
feature_names = vect_count_bigrams.get_feature_names_out()
coefficients = model_lr.coef_
classes = model_lr.classes_

for idx, class_label in enumerate(classes):
    if idx == 0:
        continue

    print(f"Classe : {class_label}")

    coef_with_features = zip(coefficients[idx], feature_names)

    sorted_features = sorted(coef_with_features, key=lambda x: x[0],
reverse=True)

    print("Top 5 n-grams influents :")
    for coef, feature in sorted_features[:5]:
        print(f"{feature}: {round(coef, 2)}")
    print("-" * 30)
```

```
Classe : 1

Top 5 n-grams influents :
ratag: 2.01
soporif: 2.01
nanar: 1.9
ringardis: 1.89
grand import: 1.8
------------------------------
Classe : 2
Top 5 n-grams influents :
hel: 1.63
dommag: 1.59
etiol: 1.57
echou: 1.55
san grand: 1.54
------------------------------
Classe : 3
Top 5 n-grams influents :
mondain: 1.51
hauteur ambit: 1.42
etre sublim: 1.41
demun: 1.39
balasko: 1.38
------------------------------
Classe : 4
Top 5 n-grams influents :
```

```
premi sequenc: 1.67
si souvent: 1.51
san faut: 1.43
lunet: 1.41
quelqu longueur: 1.36
------------------------------
Classe : 5
Top 5 n-grams influents :
magistral: 2.27
bijou: 2.14
magnif: 1.94
laguion: 1.82
boulevers: 1.75
------------------------------
```

Les résultats montrent que le modèle identifie des n-grams reflétant bien les tonalités des critiques pour chaque note. Les classes faibles (1-2) incluent des termes négatifs comme "ratag", "dommag" ou "echou" tandis que les classes élevées (4-5) utilisent des termes élogieux comme "magistral" ou "bijou". Cela indique que le modèle capture efficacement les différences de sentiment dans les commentaires en lien avec les notes.