

Comparaison d'algorithmes pour classification de texte

Dans le cadre du cours de traitement automatique de la langue naturelle offert à polytechnique Montréal

Identification de l'équipe:

- Grover-Brando Tovar Oblitas
- Andy Chen
- Marco Mudenge

Description:

Dans ce premier TP, vous explorerez les bases du traitement automatique du langage naturel. Au cours de ce travail, vous appliquerez concrètement les concepts enseignés en classe pour résoudre une tâche de classification simple. De plus, le processus ressemblera à la manière dont vous aborderiez ce type de problème dans le monde réel. Tout au long de ce laboratoire, vous vous familiariserez avec des bibliothèques couramment utilisées en NLP ainsi qu'en science des données.

Dans ce laboratoire, vous travaillerez avec un jeu de données comprenant des évaluations de produits provenant d'Amazon. Pour chaque évaluation, le jeu de données contient trois informations : le titre fourni par l'utilisateur, le commentaire détaillé et le nombre d'étoiles attribué par l'utilisateur au produit.

L'objectif de cette tâche consistera à prédire le nombre d'étoiles attribué à une évaluation à partir du commentaire et du titre qui lui sont associés.

Le travail sera divisé en 3 parties:

- Chargement, prétraitement et visualisation des données: Dans cette première partie, vous allez charger et prétraiter les données afin qu'elles soient prêtes à être utilisées par les algorithmes lors de la deuxième partie.
- Classification: Cette partie consistera à explorer les différents algorithmes pouvant être appliqués à cette tâche. Vous ferez aussi une analyse des sorties du classificateur bayésien naïf.
- Amélioration de modèle: Cette dernière partie consistera à améliorer votre modèle de 2 façons différentes. D'abord, vous ferez une recherche d'hyper-paramètres avec de la validation croisée en utilisant un GridSearch. Ensuite, vous ferez de l'extraction d'attributs avec l'aide de ChatGPT afin de d'entraîner un nouveau modèle et de comparer ainsi une représentation de type "Bag of words" et une représentation avec attributs spécifiques.

Plan

1. Chargement, prétraitement et visualisation des données

- 1.1 Charger les données
- 1.1.1 Charger le jeu de données
- 1.1.2 Fusionner les colonnes title et text en une seule colonne
- 1.2 Prétraitement des données
- 1.3 Visualisation des données
 - 1.3.1 Afficher dans un graphique le nombre d'exemples présents dans le jeu de données pour chaque catégorie
 - 1.3.2 Afficher dans un graphique la quantité moyenne de jetons par exemple selon la catégorie
 - 1.3.3 Afficher en texte les top 10 jetons les plus fréquents par catégorie
 - 1.3.4 Afficher en texte les top 10 adjectifs les plus fréquents selon la catégorie
- 1.4 Diviser les données en ensembles d'entraînement et de test
- 1.5 Construction du vocabulaire
- 1.6 Vectorisation des données

2. Classification

- 2.1 Modèle aléatoire (Random baseline)
- 2.2 Analyse et compréhension d'un classificateur bayésien naïf (NB)
 - 2.2.1 Construction du modèle
 - 2.2.2 Matrice de confusion
 - 2.2.3 Visualisation des probabilités de NB
 - 2.2.4 Visualisation des erreurs commises
 - 2.2.5 Analyse d'erreurs commises
- 2.3 Régression logistique
- 2.4 MLP

3. Amélioration de modèle

- 3.1 Recherche d'hyper-paramètres et validation croisée
- 3.2 Extraction d'attributs (Feature extraction) avec ChatGPT
- 3.3 Amélioration du modèle en 3.2

1. Chargement, prétraitement et visualisation des données (30 points)

Dans cette première partie, vous allez charger et prétraiter les données afin qu'elles soient prêtes à être utilisées par les algorithmes lors de la deuxième partie.

1.1 Charger les données (2 points)

Ce numéro doit être résolu en utilisant la bibliothèque **pandas**.

1.1.1 Charger le jeu de données (1 point)

Chargez le jeu de données amazon_rating.csv. Affichez ensuite son contenu.

```
In [ ]: import pandas as pd
```

```
data = pd.read_csv('amazon_rating.csv')  
data
```

Out[]:

	title	text	rating
0	Five Stars	good as any name brand	5
1	Did The Job	Ordered on accident when I had searched for RE...	3
2	Great product	I was looking for something to read on and thi...	5
3	Leaking Acid EVERYWHERE!!	After 2nd recharge and use all but 3 are leaki...	1
4	One Star	They fail earlier than brand names. I assumed ...	1
...
2788	Three Stars	Weird but some didn't last long as they should	3
2789	Good for kids but SLOW	A good starter tablet, but very very slow. Don...	3
2790	good tablet to star	is a God tablet but the camera could be a litt...	3
2791	Just decent tablet	Not many apps. The first one was already retur...	3
2792	One Star	don't last long. Replace batteries in my clock...	1

2793 rows × 3 columns

1.1.2 Fusionner les colonnes title et text en une seule colonne (1 point)

Afin de faciliter la tâche pour le reste du TP, nous allons fusionner ces deux colonnes afin que nous n'ayons qu'un seul texte à considérer lors de la vectorisation.

Afin de s'assurer de l'intégrité des textes, fusionnez-les à l'aide d'un espace. Par exemple, une évaluation ayant le titre "Five Stars" et le commentaire "good as any name brand" aura comme texte final "Five Stars good as any name brand".

Stockez le résultat dans la colonne "text" et supprimez la colonne "title".

```
In [ ]: data['text'] = data['title'] + ' ' + data['text']  
data.drop(columns=['title'], inplace=True)  
data.head()
```

Out[]:

	text	rating
0	Five Stars good as any name brand	5
1	Did The Job Ordered on accident when I had sea...	3
2	Great product I was looking for something to r...	5
3	Leaking Acid EVERYWHERE!! After 2nd recharge ...	1
4	One Star They fail earlier than brand names. I...	1

1.2 Prétraitement des données (4 points)

En utilisant la librairie nltk, implémentez la fonction suivante qui :

- Enlève les majuscules.
- Enlève les caractères de ponctuation.
- Segmente la séquence en entrée en une liste de jetons (tokenization).
- Enlève les "stopwords"
- Effectue la racinisation.
- Retourne l'ensemble des jetons de la séquence

```
In [ ]: import nltk  
from nltk.corpus import stopwords  
from nltk.tokenize import word_tokenize  
import string  
import re  
  
nltk.download('stopwords')  
nltk.download('punkt')  
nltk.download('averaged_perceptron_tagger')  
stopwords = stopwords.words('english')
```

```

stemmer = nltk.stem.porter.PorterStemmer()
lemmer = nltk.stem.wordnet.WordNetLemmatizer()

def preprocess(sentence):
    """
    Fonction qui transforme une chaîne de caractère en liste de jetons.
    Les pré-traitements à implémenter sont:
    1. Enlever les majuscules
    2. Enlever les caractères de ponctuations
    3. Séparer la chaîne de caractères en une liste de jetons (tokenization)
    4. Enlever les stopwords
    5. Stemming (racinisation)

    :param sentence: une chaîne de caractère
    :return: la liste de jetons
    """

    # TODO: ENLEVER LES COMMENTAIRES APRÈS LE REVIEW
    # Enlever les majuscules
    sentence = sentence.lower()

    # Enlever les caractères de ponctuations
    sentence = re.sub(r'[^w\s]', ' ', sentence)
    #sentence = re.sub('[^A-Za-z]', ' ', sentence)                                # Diff Brando
    # Garde seulement les suites de lettres. Pas de chiffres. Don't = [don, t]
    # = 2000 mots en moins 52519 vs 50066
    # Pas de chiffre comme token

    # tokenization
    tokenized_words = word_tokenize(sentence)

    # Enlever les stopwords et stemming
    tokens = [stemmer.stem(token) for token in tokenized_words if token not in stopwords]

    return tokens

```

```

[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\mudma\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\mudma\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\mudma\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!

```

```

In [ ]: """
NE PAS MODIFIER

Le code suivant appliquera votre fonction sur tous les exemples. Il gardera aussi une version originale pour une analyse future.

"""

data["text_original"] = data["text"]
data["text"] = data["text"].apply(preprocess)

```

```

In [ ]: # TODO : ENLEVER CE CODE APRÈS REVIEW
# Pour comparer les fonctions preprocess
import numpy as np

# Count the total number of tokens
tokens_counts = data["text"].apply(lambda x: len(x)).sum()
tokens_as_set_count = data["text"].apply(lambda x: len(set(x))).sum()
unique_tokens_counts = data["text"].apply(lambda x: len(np.unique(x))).sum()
digital_tokens_counts = data["text"].apply(lambda x: len([token for token in x if token.isdigit()])).sum()
print("Total number of tokens: ", tokens_counts)
print("Total number of tokens as set: ", tokens_as_set_count)
print("Total number of unique tokens: ", unique_tokens_counts)
print("Total number of digital tokens: ", digital_tokens_counts)

```

```

Total number of tokens: 52519
Total number of tokens as set: 43948
Total number of unique tokens: 43948
Total number of digital tokens: 1312

```

1.3 Visualisation des données (15 points)

Utilisez la bibliothèque matplotlib pour les graphiques. Vous pouvez utiliser n'importe quelle classe de base de Python, par exemple collections.Counter, qui sera utile pour l'affichage des jetons.

La colonne "rating" contient le nombre d'étoiles associé à l'évaluation d'un utilisateur. Le nombre d'étoiles varie entre 1 et 5.

Afin de simplifier la tâche de classification, nous avons enlevé les commentaires ayant 2 et 4 étoiles du jeu de données. Cela signifie qu'il y a trois catégories de commentaires, c'est-à-dire ceux ayant 1, 3 ou 5 étoiles.

Affichez dans un graphique :

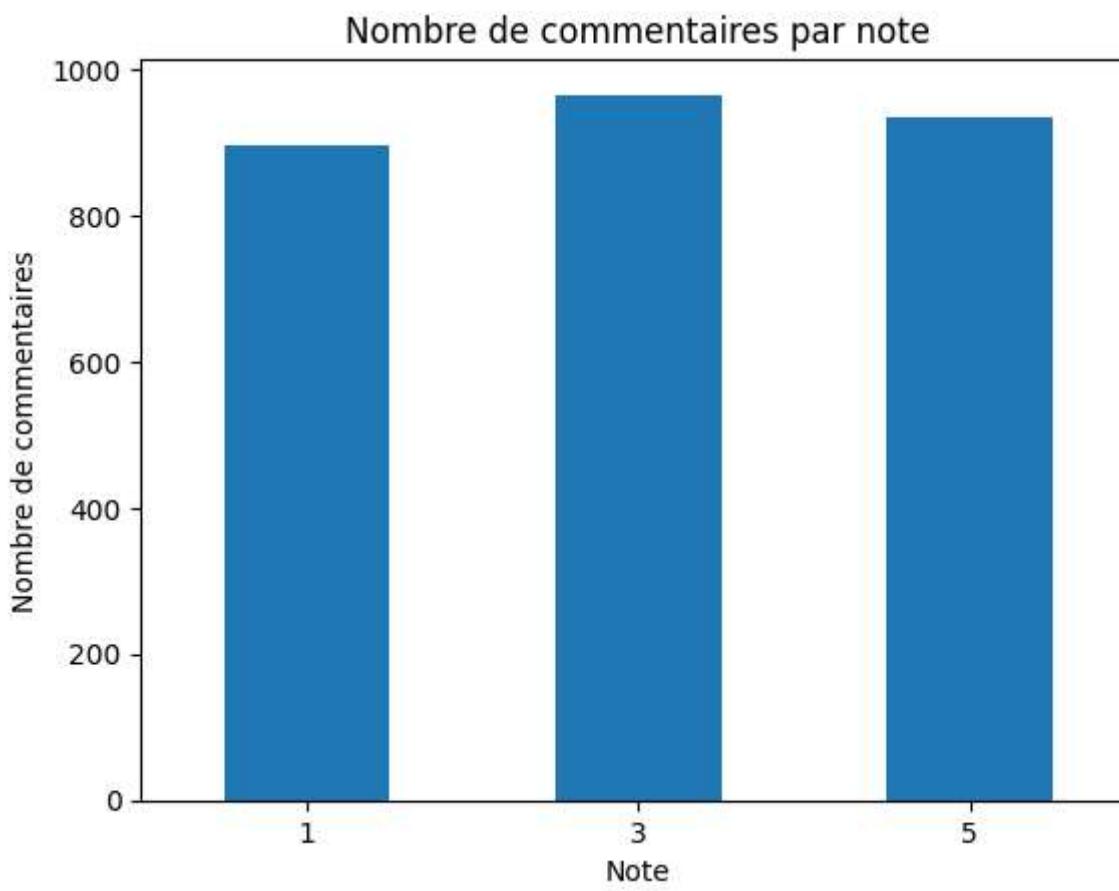
- Le nombre d'exemples présents dans le jeu de données par catégorie.
- La quantité moyenne de jetons par exemple selon la catégorie.

1.3.1 Afficher dans un graphique le nombre d'exemples présents dans le jeu de données pour chaque catégorie (3 points)

```
In [ ]: import matplotlib.pyplot as plt

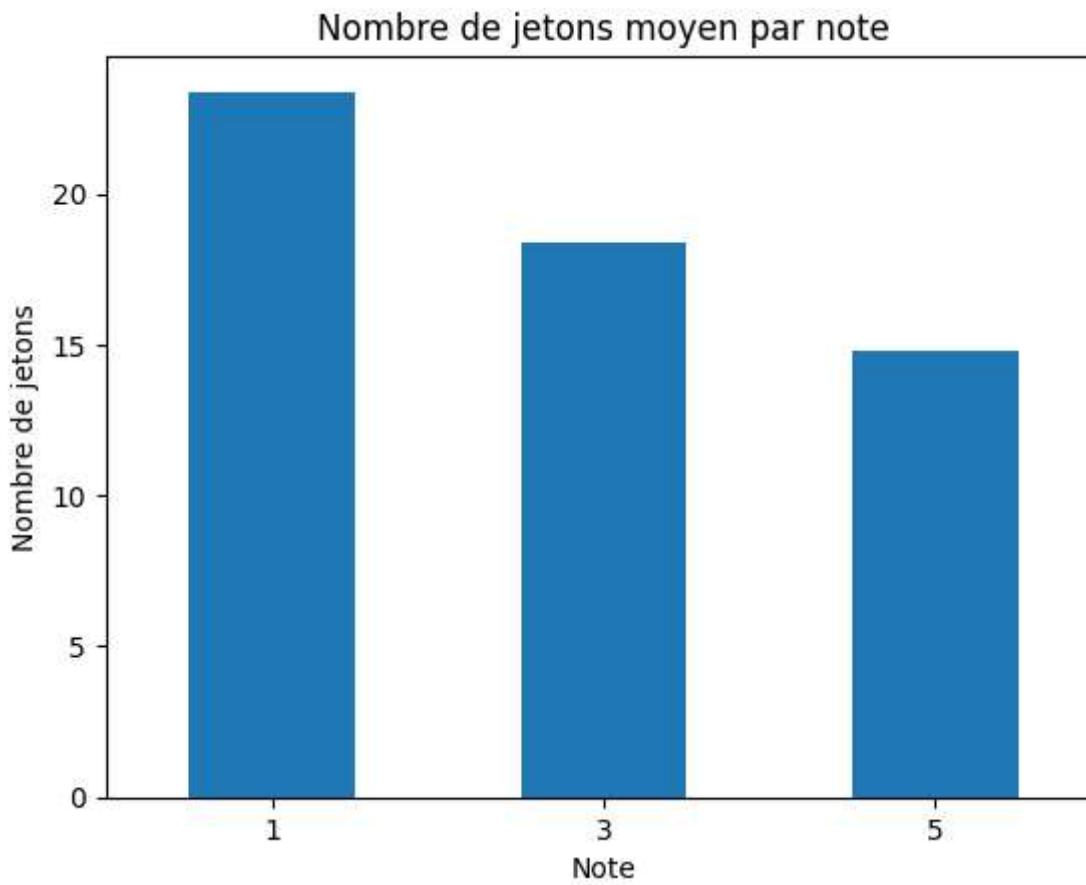
data.groupby("rating").count().plot(kind="bar", y="text", legend=False, rot=0)

plt.title('Nombre de commentaires par note')
plt.xlabel('Note')
plt.ylabel('Nombre de commentaires')
plt.show()
```



```
In [ ]: data["number_of_tokens"] = data["text"].apply(len)
data.groupby("rating").mean("number_of_tokens").plot(kind="bar", y="number_of_tokens", legend=False, rot=0)

plt.title('Nombre de jetons moyen par note')
plt.xlabel('Note')
plt.ylabel('Nombre de jetons')
plt.show()
```



1.3.3 Afficher en texte les top 10 des jetons les plus fréquents par catégorie (4 points)

Affichez en texte les 10 jetons les plus fréquents selon la catégorie.

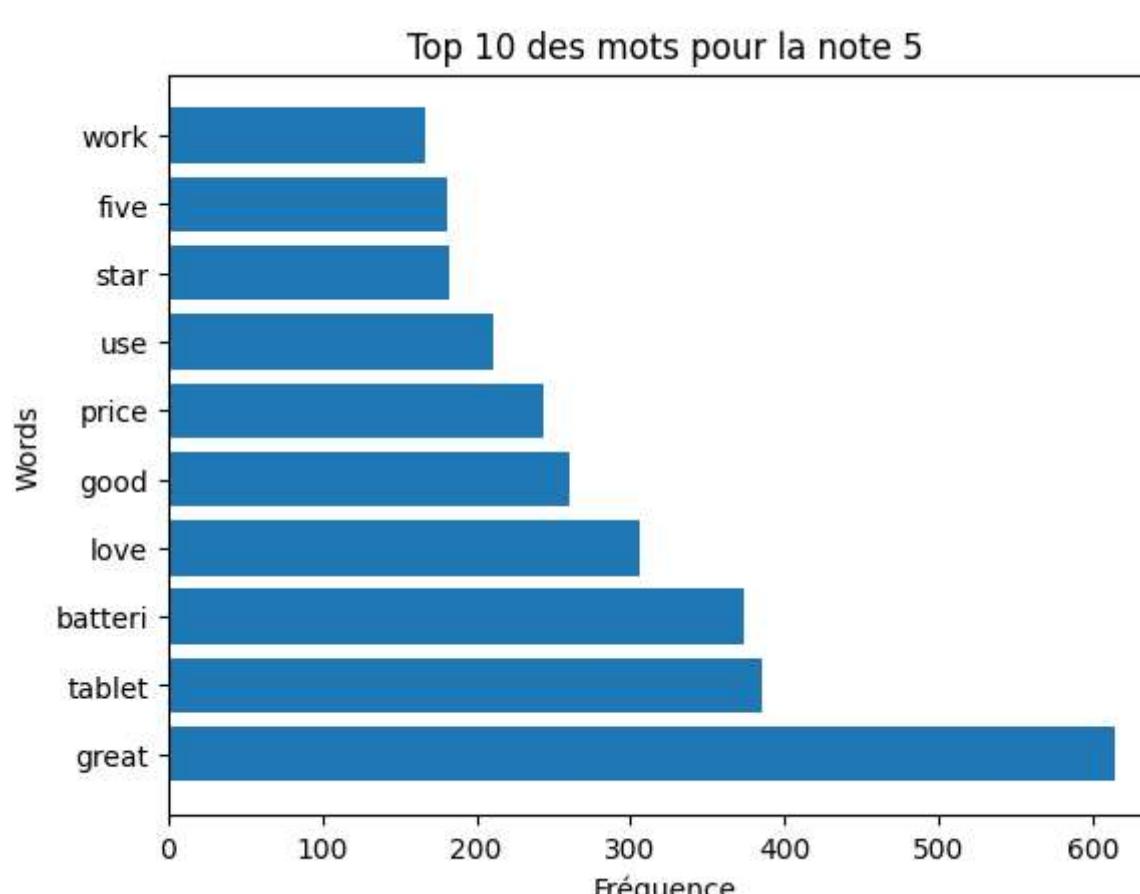
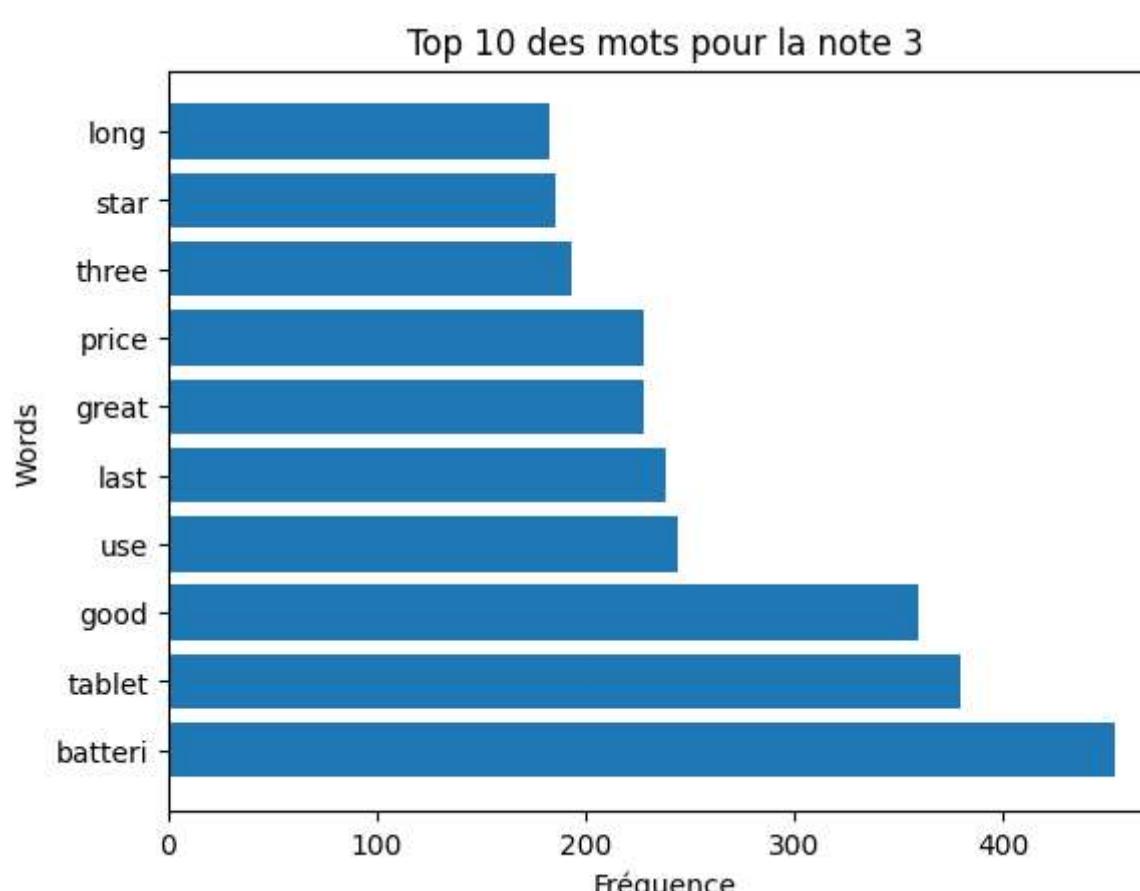
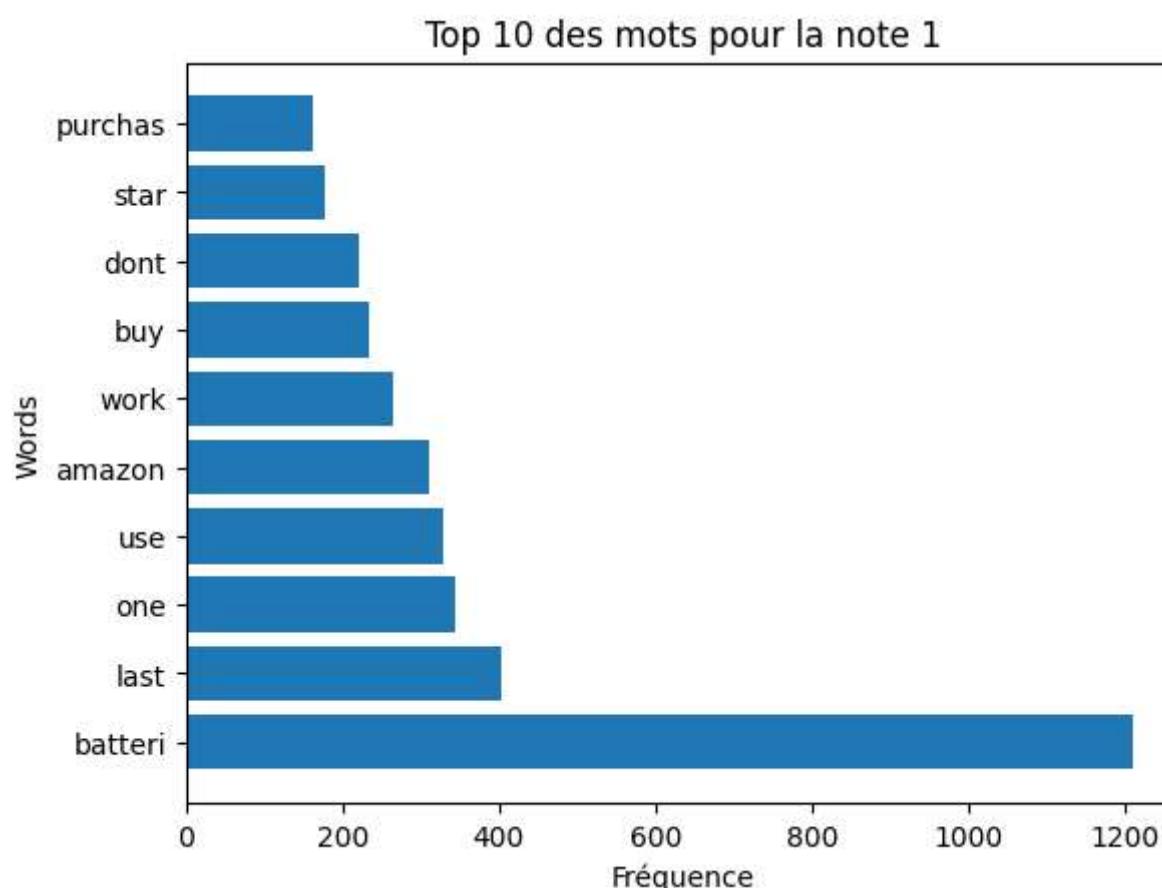
```
In [ ]: from collections import Counter

top_words_df = data[['text', 'rating']].groupby('rating').sum().reset_index()
top_words_df['top'] = top_words_df['text'].apply(lambda x: Counter(x).most_common(10))
top_words_df
```

	rating	text	top
0	1	[leak, acid, everyewher, 2nd, recharg, use, 3...	[(batteri, 1211), (last, 403), (one, 345), (us...
1	3	[job, order, accid, search, recharg, batteri, ...	[(batteri, 454), (tablet, 380), (good, 360), (...
2	5	[five, star, good, name, brand, great, product...	[(great, 615), (tablet, 386), (batteri, 374), ...

```
In [ ]: def plot_top_10_words(row, type):
    """
        Fonction qui imprime un top 10 des mots les plus fréquents pour une note donnée.
    """
    words, frequencies = zip(*row['top']) # Unpack tuple into separate lists
    plt.barh(words, frequencies)
    plt.xlabel('Fréquence')
    plt.ylabel('Words')
    plt.title(f"Top 10 des {type} pour la note {row['rating']}"))
    plt.show()
```

```
In [ ]: top_words_df.apply(lambda row: plot_top_10_words(row, "mots"), axis=1)
```



```
Out[ ]: 0    None  
1    None  
2    None  
dtype: object
```

1.3.4 Afficher en texte les top 10 des adjectifs les plus fréquents selon la catégorie (4 points)

Pour cet exercice, vous devrez utiliser la fonction `nltk.pos_tag` et retenir les jetons identifiés comme JJ.

Pour obtenir de bons résultats, le tagger `nltk.pos_tag` doit être exécuté sur le texte original, incluant les stopwords. Vous devrez donc partir des évaluations originales. Pour vous simplifier la tâche, utilisez le tokeniser `word_tokenize` provenant de nltk.

Les adjectifs sont les jetons identifiés comme JJ.

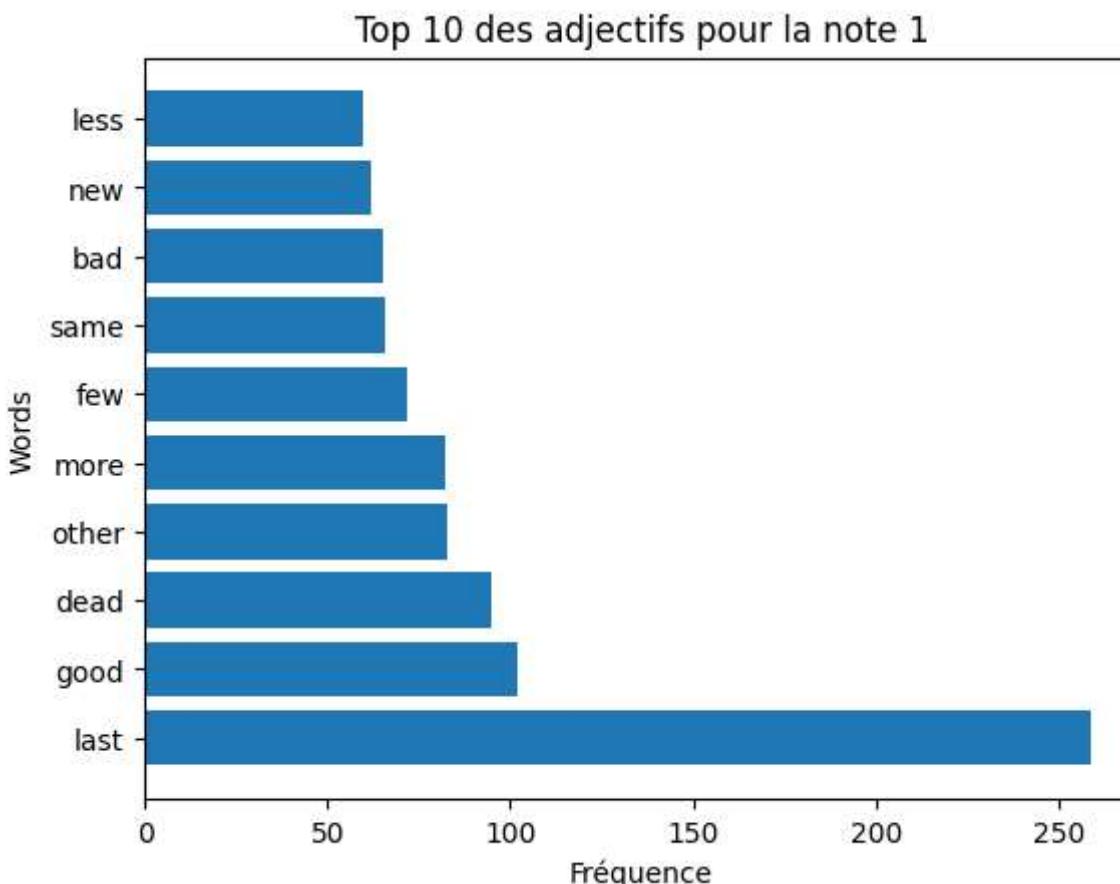
```
In [ ]: from nltk.tokenize import word_tokenize  
from nltk import pos_tag  
  
data['tag'] = data['text_original'].apply(word_tokenize).apply(pos_tag)  
  
# Les tag des adjectifs sont JJ, JJR, JJS  
data['adjectives'] = data['tag'].apply(lambda x: [item[0] for item in x if item[1].startswith('JJ')])  
  
data.head()
```

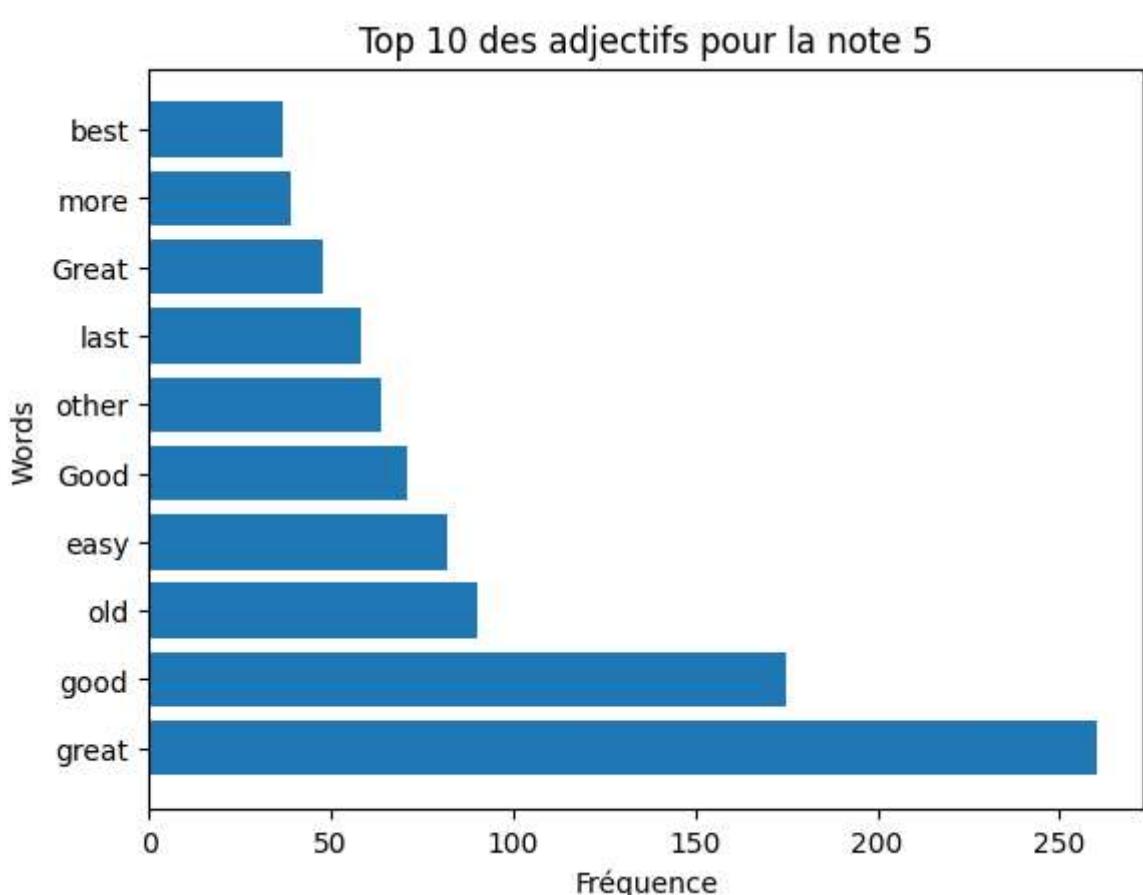
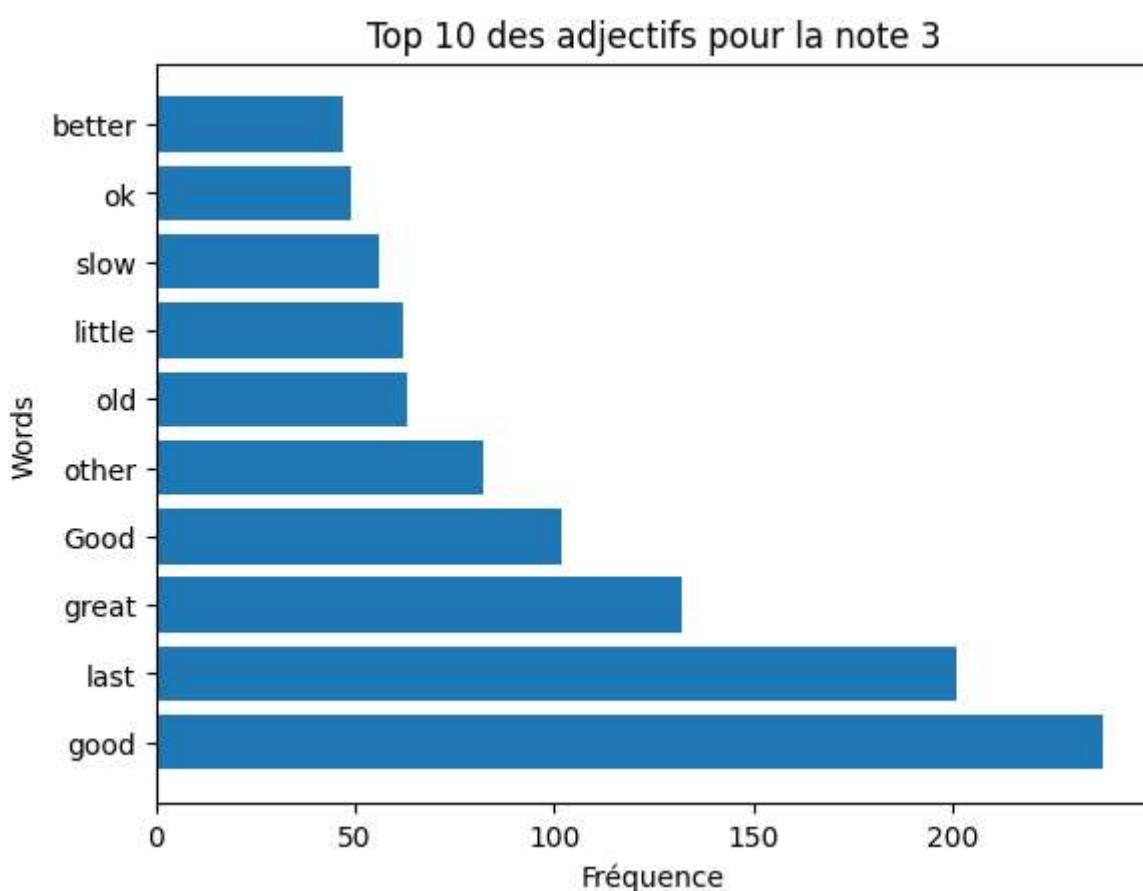
	text	rating	text_original	number_of_tokens	tag	adjectives
0	[five, star, good, name, brand]	5	Five Stars good as any name brand	5	[(Five, CD), (Stars, NNS), (good, JJ), (as, IN...]	[good]
1	[job, order, accid, search, recharg, batteri, ...	3	Did The Job Ordered on accident when I had sea...	15	[(Did, NNP), (The, DT), (Job, NNP), (Ordered, ...	[happy, free]
2	[great, product, look, someth, read, fit, bill...	5	Great product I was looking for something to r...	25	[(Great, NNP), (product, NN), (I, PRP), (was, ...	[right, screen, good, great, nice, Big]
3	[leak, acid, everyewher, 2nd, recharg, use, 3...	1	Leaking Acid EVERYWHERE!! After 2nd recharge ...	12	[(Leaking, VBG), (Acid, NNP), (EVERYWHERE, NN...]	[corrosive]
4	[one, star, fail, earlier, brand, name, assum,...	1	One Star They fail earlier than brand names. I...	14	[(One, CD), (Star, NNP), (They, PRP), (fail, V...]	[earlier]

```
In [ ]: top_adjectives_df = data[['adjectives', 'rating']].groupby("rating").sum().reset_index()  
top_adjectives_df["top"] = top_adjectives_df ["adjectives"].apply(lambda x: Counter(x).most_common(10))  
top_adjectives_df
```

	rating	adjectives	top
0	1	[corrosive, earlier, regular, bad, less, last, ...	[(last, 259), (good, 102), (dead, 95), (other, ...]
1	3	[happy, free, okay, smaller, lighter, original, ...	[(good, 238), (last, 201), (great, 132), (Good, ...]
2	5	[good, right, screen, good, great, nice, Big, ...	[(great, 260), (good, 175), (old, 90), (easy, ...]

```
In [ ]: top_adjectives_df.apply(lambda row: plot_top_10_words(row, "adjectifs"), axis=1)
```





```
Out[ ]: 0    None
        1    None
        2    None
       dtype: object
```

1.4 Diviser les données en ensembles d'entraînement et de test (1 point)

À l'aide de la fonction `train_test_split` de SKlearn, séparez les données en ensembles d'entraînement (67% des données) et de test (33% des données). Gardez les deux ensembles dans 2 variables.

```
In [ ]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['rating'], test_size=0.33, random_state=0)
```

1.5 Construction du vocabulaire (4 points)

Dans un modèle Bag-of-Words (BoW), un vocabulaire est prédéterminé à partir de l'ensemble d'entraînement. Seuls les mots faisant partie de ce vocabulaire seront considérés pour la suite.

Complétez la fonction `build_voc` qui retourne une liste de jetons qui sont présents au moins n fois (threshold passé en paramètre) dans la liste d'exemples (également passée en paramètre). Vous pouvez utiliser la classe Counter.

Ensuite, appelez cette fonction pour construire votre vocabulaire.

```
In [ ]: def build_voc(documents, threshold):
    documents = documents.sum()
    print('Taille du document (nombre de mots initial): {}'.format(len(documents)))
    # Count the number of occurrences of each word
    word_counts = Counter(documents)
    print('Taille du vocabulaire avant le filtrage (nombre de mots uniques): {}'.format(len(word_counts)))
    # Keep only the words that appear at Least threshold times
    vocabulary = [word for word, count in word_counts.items() if count >= threshold]
    print('Taille du vocabulaire après le filtrage: {}'.format(len(vocabulary)))
```

```

    return vocabulary

voc = build_voc(X_train, 10)

Taille du document (nombre de mots initial): 34633
Taille du vocabulaire avant le filtrage (nombre de mots uniques): 3139
Taille du vocabulaire après le filtrage: 595

```

1.6 Vectorisation des données (4 points)

À l'aide de la classe `TfidfVectorizer` de Sklearn, transformez l'ensemble de jetons en matrice de co-occurrence utilisant TF-IDF.

Utilisez le vocabulaire construit au numéro précédent dans votre matrice de co-occurrence (voir le paramètre `vocabulary` de `TfidfVectorizer`).

Faites attention: Il ne faut pas entraîner (`fit`) la vectorisation sur l'ensemble de test

```

In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(vocabulary=voc)

X_train_matrix = vectorizer.fit_transform(X_train.apply(lambda x: ' '.join(x))) # X_train contains tokens. We need to join them to
X_test_matrix = vectorizer.transform(X_test.apply(lambda x: ' '.join(x)))

In [ ]: print('Reviews in X_train: {}'.format(len(X_train)))
print('Count of features: {}'.format(len(vectorizer.get_feature_names_out())))
print('Shape of co-occurrence matrix: {} \n'.format(X_train_matrix.shape))
print('Original comment: {}'.format(data['text_original'].iloc[X_train.index[0]]))
print('First row X_train: {}'.format(X_train.iloc[0]))
print('Non-zeros features in the first row of matrix: {}'.format(vectorizer.get_feature_names_out()[X_train_matrix[0].nonzero()[1]]))

Reviews in X_train: 1871
Count of features: 595
Shape of co-occurrence matrix: (1871, 595)

Original comment: Three Stars They ran out of energy very quick but it's a deal
First row X_train: ['three', 'star', 'ran', 'energi', 'quick', 'deal']
Non-zeros features in the first row of matrix: ['deal' 'quick' 'ran' 'star' 'three']

```

2. Classification (35 points)

Maintenant que les données sont prêtes à être utilisées par nos modèles, nous allons entraîner et tester différents types de modèles sur le jeu de données afin d'en faire la comparaison.

Cette section sera divisée en cinq modèles:

- Modèle aléatoire (Random baseline)
- Classificateur bayésien naïf
- Régression Logistique
- Multi-Layer Perceptron (MLP)

2.1 Modèle aléatoire (Random baseline) (5 points)

Un seuil (baseline) est un modèle servant de référence et dont les performances représentent un seuil à dépasser.

a) Générez ce seuil en effectuant des prédictions aléatoires parmi les valeurs 1, 3 et 5. Ensuite, affichez les mesures de performance : précision, rappel (recall) et F1. Utilisez la classe `classification_report` de SKlearn et affichez 4 chiffres après la virgule. (3.5 points)

```

In [ ]: import numpy as np

classes = np.unique(y_train)
y_random = np.random.choice(classes, size=len(y_test))

```

```

In [ ]: from sklearn.metrics import classification_report

print(classification_report(y_test, y_random, digits=4))

```

	precision	recall	f1-score	support
1	0.3426	0.3183	0.3300	311
3	0.3107	0.3387	0.3241	310
5	0.3119	0.3056	0.3087	301
accuracy			0.3210	922
macro avg	0.3217	0.3209	0.3209	922
weighted avg	0.3218	0.3210	0.3211	922

b) Comment pouvez-vous expliquer le F1-score obtenu? (1.5 points)

Le F1-score est faible (proche de 0.34 en moyenne), ce qui signifie que le taux de précision et de rappel ne sont pas élevés. Les résultats sont générés aléatoirement sans tenir compte des variables explicatives.

En choisissant une classe au hasard parmi N classes, on a théoriquement 1/N chances de prédire une bonne classe.

Le graphique #1.3.1 montre que les classes sont équitablement représentées dans le jeu de données. Ainsi, on s'attend à avoir une

précision sur l'ensemble de validation d'environ 33,3%.

La précision pour chaque classe sera donnée par $\text{TP} / (\text{TP} + \text{FP})$. Pour chaque classe TP sera environ égal à $0.33^2 / (0.33^2 + 0.33 - 0.33^2) = 0.33$.

Ceci s'explique par le fait que la classe en question est présente dans un ratio de 0.33 et lorsqu'on fait la prédiction de celle-ci, il y a 0.33 de probabilité d'avoir choisi la bonne classe donc TP est 0.33^2 .

FP peut se calculer avec $0.33 - 0.33^2$ puisque on va choisir la classe en question avec 0.33 de probabilité et que 0.33^2 de ces fois seront des TP.

Ensuite le Recall peut être calculé d'une façon similaire et nous obtenons 0.33 aussi. Le f1 score théorique sera donc $2 * 0.33 * 0.33 / (0.66) = 0.33$.

2.2 Analyse et compréhension d'un classificateur bayésien naïf (NB) (22 points)

Naive Bayes (NB) est un algorithme très simple pouvant servir de bon point de départ (baseline) pour les tâches de classification. Ce numéro portera sur l'analyse de ce modèle afin de bien comprendre son comportement.

2.2.1 Construction du modèle (4 points)

Commencez d'abord par construire le modèle à l'aide de la classe MultinomialNB de SKlearn. Utilisez les données vectorisées produites en 1.6.

Affichez les performances de votre classificateur (précision, recall, F1-score).

```
In [ ]: from sklearn.naive_bayes import MultinomialNB
```

```
clf_nb = MultinomialNB()
clf_nb.fit(X_train_matrix, y_train)
```

```
y_pred = pd.Series(clf_nb.predict(X_test_matrix))
```

```
In [ ]: print(classification_report(y_test, y_pred, digits=4))
```

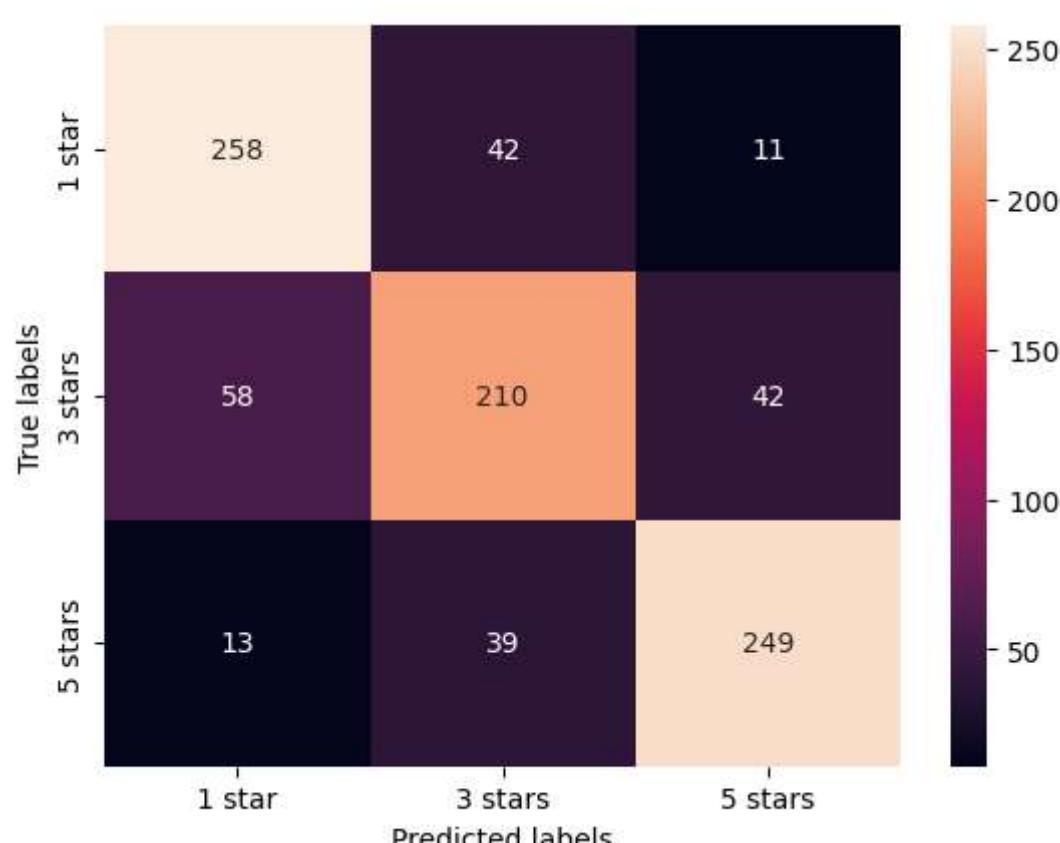
	precision	recall	f1-score	support
1	0.7842	0.8296	0.8063	311
3	0.7216	0.6774	0.6988	310
5	0.8245	0.8272	0.8259	301
accuracy			0.7777	922
macro avg	0.7768	0.7781	0.7770	922
weighted avg	0.7763	0.7777	0.7765	922

2.2.2 Matrice de confusion (3 points)

Visualisez la matrice de confusion de votre modèle en utilisant la fonction `heatmap` de seaborn. Celle-ci peut prendre en entrée une matrice de confusion comme celle fournie par la fonction `confusion_matrix` dans SKLearn.

```
In [ ]: from sklearn.metrics import confusion_matrix
from seaborn import heatmap
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
ax = heatmap(conf_matrix, annot=True, fmt='d')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.xaxis.set_ticklabels(['1 star', '3 stars', '5 stars'])
ax.yaxis.set_ticklabels(['1 star', '3 stars', '5 stars'])
plt.show()
```



2.2.3 Visualisation des probabilités de NB (5 points)

Naive Bayes est un classificateur suivant une approche générative. Durant son entraînement, il apprend les probabilités $P(x_i|y)$. En utilisant le théorème de Bayes, on peut exprimer la probabilité d'une classe donnée y étant donné un ensemble de caractéristiques x_1, x_2, \dots, x_n comme suit :

$$P(y|x_1, x_2, \dots, x_n) = P(y) * P(x_1|y) * P(x_2|y) * \dots * P(x_n|y)$$

Ainsi, étant donné un exemple ayant le jeton x_i , plus la probabilité $P(x_i|y)$ est élevée pour une classe, plus la probabilité que l'exemple provienne de cette classe augmente.

Écrivez du code permettant de visualiser les jetons ayant les plus grandes probabilités selon la classe dans un graphique de type `barh`. Consultez la documentation de `MultiNomialNB` afin de trouver les probabilités $P(x_i|y)$. Le graphique produit devrait montrer, sur l'axe des Y, les 10 jetons associés au $P(x_i|y)$ le plus grand selon y . L'axe des X devrait représenter la valeur des probabilités.

Ce code devra être sous forme d'une fonction où on passe la classe y en paramètre.

```
In [ ]: import matplotlib.pyplot as plt

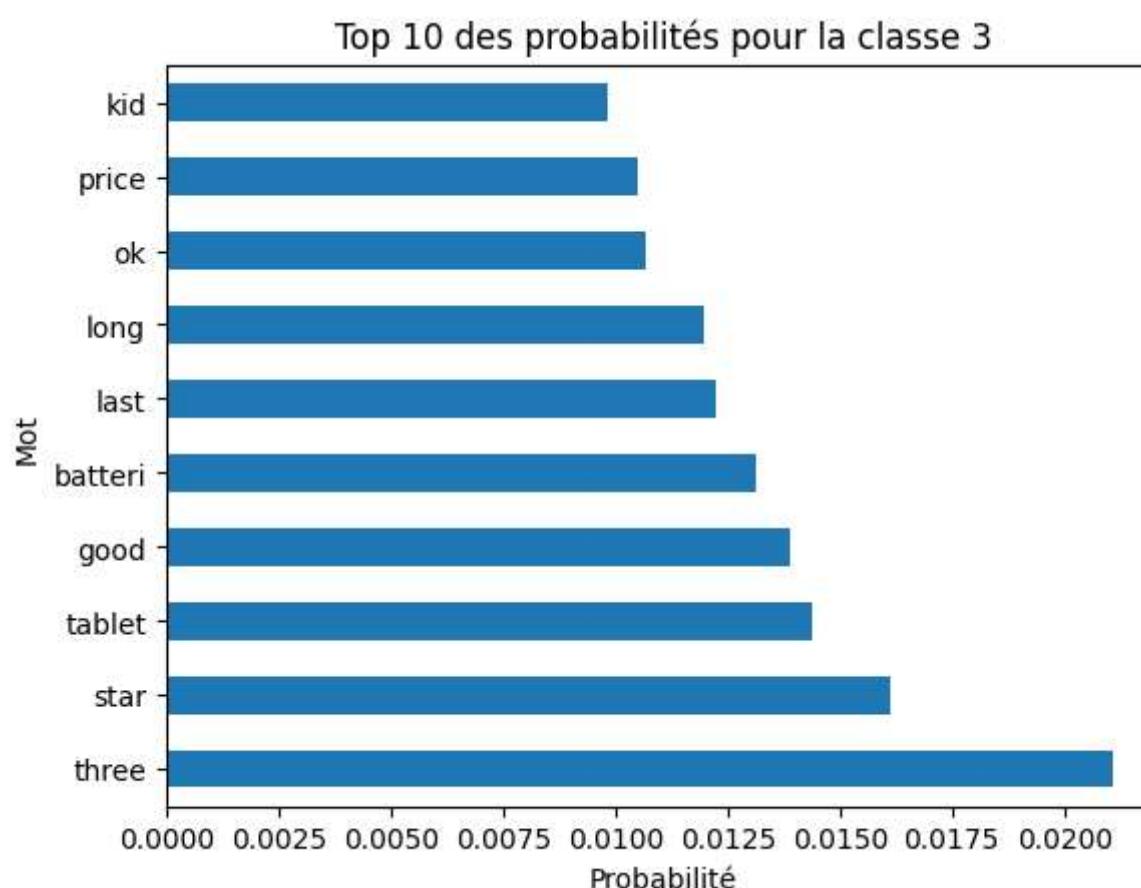
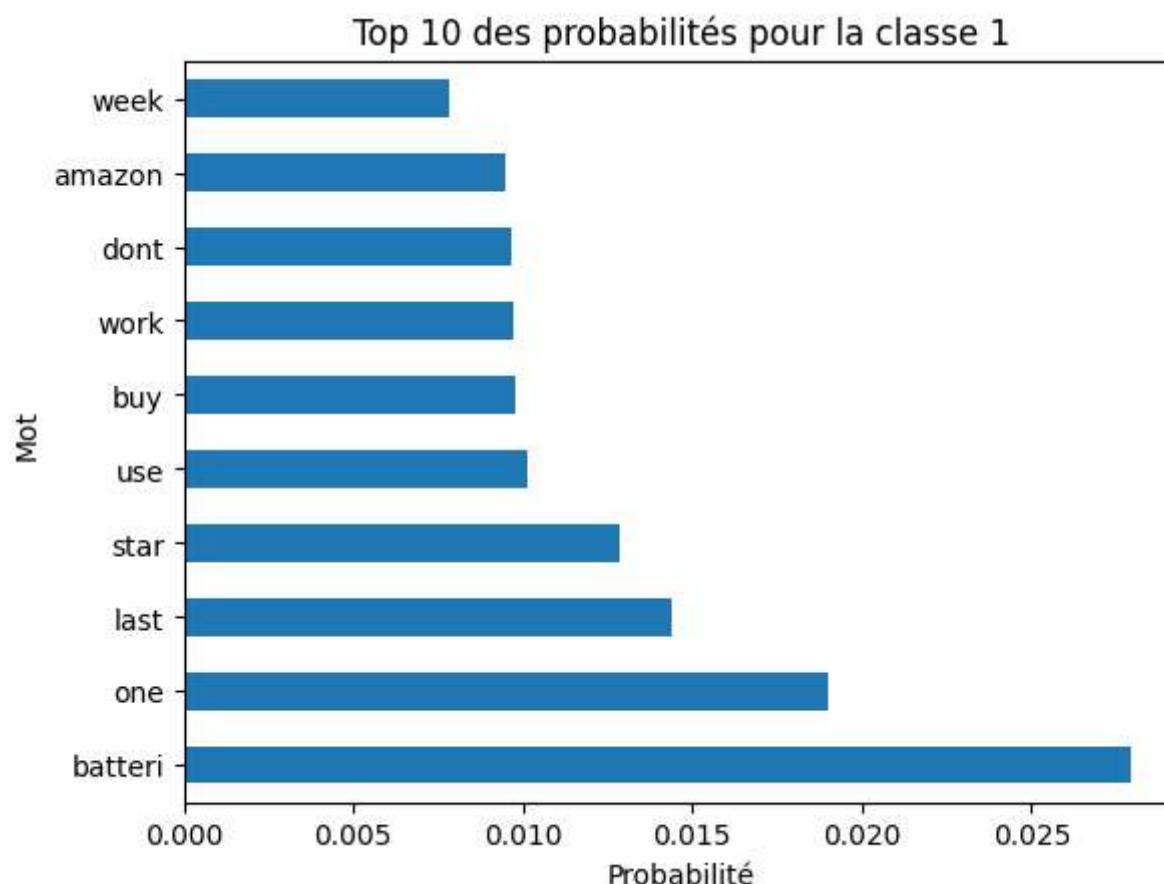
classes = clf_nb.classes_
features_prob = pd.DataFrame(np.exp(clf_nb.feature_log_prob_), index=clf_nb.classes_, columns=vectorizer.get_feature_names_out())

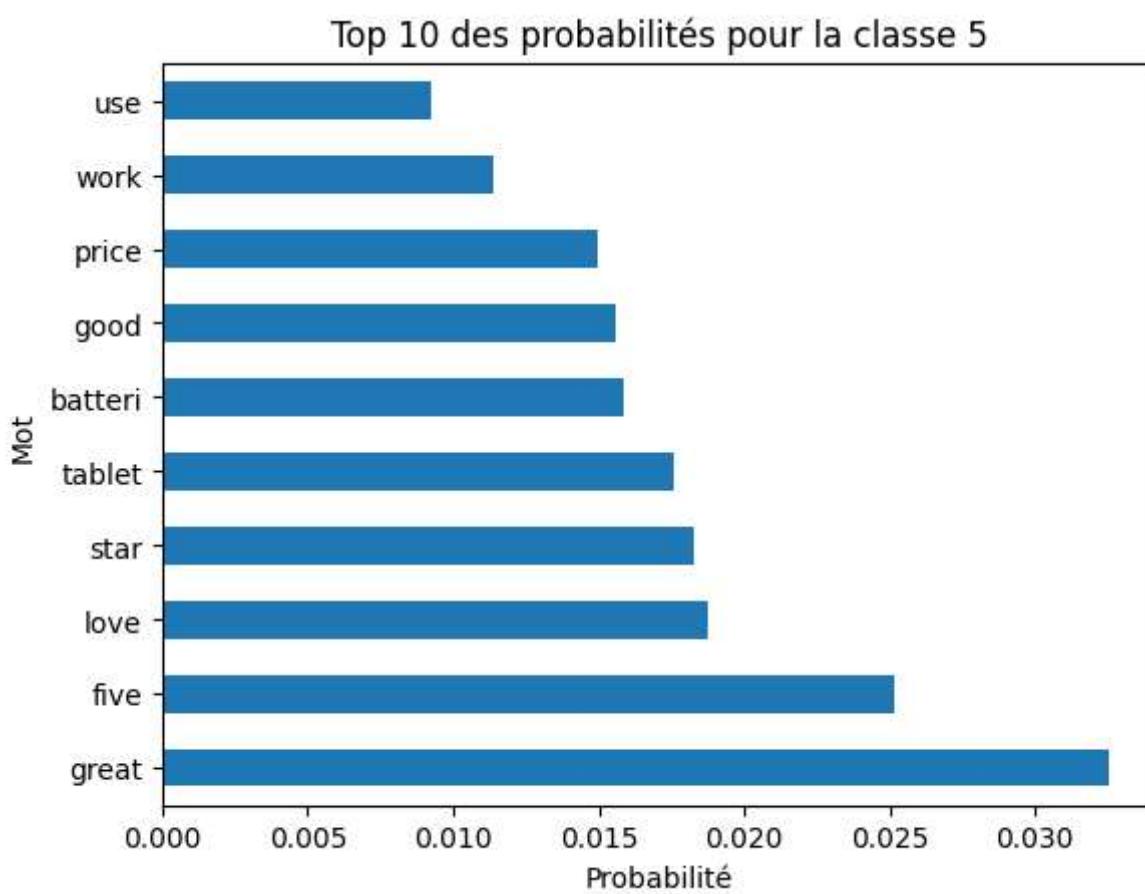
features_prob.iloc[0].sort_values(ascending=False).head(10).reset_index().plot(kind='barh', x='index', legend=False)
plt.xlabel('Probabilité')
plt.ylabel('Mot')
plt.title('Top 10 des probabilités pour la classe {}'.format(classes[0]))

features_prob.iloc[1].sort_values(ascending=False).head(10).reset_index().plot(kind='barh', x='index', legend=False)
plt.xlabel('Probabilité')
plt.ylabel('Mot')
plt.title('Top 10 des probabilités pour la classe {}'.format(classes[1]))

features_prob.iloc[2].sort_values(ascending=False).head(10).reset_index().plot(kind='barh', x='index', legend=False)
plt.xlabel('Probabilité')
plt.ylabel('Mot')
plt.title('Top 10 des probabilités pour la classe {}'.format(classes[2]))
```

Out[]: Text(0.5, 1.0, 'Top 10 des probabilités pour la classe 5')





Que pouvez-vous remarquer à propos des jetons affichés dans le graphique?

Plusieurs des jetons apparaissent aussi dans le top 10 des mots les plus fréquents.

On remarque aussi que beaucoup de commentaire contiennent le nombre d'étoiles données au produit.

Les mots 'one', 'three' et 'five' sont de bons indicateurs pour catégoriser le commentaires dans la bonne classes.

2.2.4 Visualisation des erreurs commises (3 points)

Trouvez toutes les phrases dont la vraie valeur est 5 mais la valeur prédictive est de 1.

Affichez ces exemples d'une manière lisible.

```
In [ ]: #%%pip install ipython
from IPython.display import display, HTML

# Indexes of 5-values in the test set
y_test_5_indexes = y_test[y_test == 5].index

# We must make them match
y_pred.index = y_test.index
predicted_1_indexes = y_pred[y_pred == 1].index

comments_5_1_index = [index for index in predicted_1_indexes if index in y_test_5_indexes]

# Showing the comments
html_str = f'

### {len(comments_5_1_index)} commentaires 5-étoiles sont prédits en 1-étoile (y_test: 5, y_pred: 1)



'
for comment in data.loc[comments_5_1_index]['text_original']:
    html_str += f'- {comment}
'
html_str += '
'

display(HTML(html_str))
```

13 commentaires 5-étoiles sont prédits en 1-étoile (y_test: 5, y_pred: 1)

- Just as good as any other batteries, better for the price Absolute best value for standard batteries (especially when using subscribe and save). If you aren't seeking rechargeable, I haven't ever been able to distinguish any real difference in quality or longevity between these and other top brands of batteries (Duracell, Energizer, etc.). Between my kids toys and my Xbox controllers, we go through about all of these batteries in about 6 months, so subscribe and save is the way to go.
- My remote now works! Only needed batteries. Best way to have electronics work is having them be charged or have batteries! Buying this pack will fix the needing batteries part!
- Saved Me From Bear Attack So... One day I was camping in the Wild West on the Rocky Mountains in Colorado... I was carrying my trusty Power Rangers limited edition collectors sword. Out of nowhere... A bear attacked me. I grabbed my trusty sword, but it was out of batteries so it did not make the swoosh sounds that adds to the effect of my swing. So I hurriedly went to my trusty AmazonBasics AA batteries, and switched out the ones in my Power Rangers sword. My swooshing sounds were amazing, and I escaped with my life (not my leg but that's okay). Thank you AmazonBasics AA Batteries for saving a life.
- AmazonBasics AAA Performance Alkaline Batteries are AAA Performance Alkaline Batteries. WOW! These AmazonBasics AAA Performance Alkaline Batteries are actually AmazonBasics AAA Performance Alkaline Batteries and work as such. Unbelievable.
- great product this is my second one and must have one of these at all times
- Amazon Basic products rule!! Not the first time I have bought Amazon Basic products and won't be the last. From my experience these products do just as well as any name brand product but with less experience. Love the packaging these batteries came in and so far from what I can tell work like a charm. Put them in a couple of my gadgets and worked perfectly. The way we go through batteries the cost savings is well worth it.
- Worth the money Function as promised!
- For what they are, great buy! It's not a high discharge battery, but for typical uses such wireless mouse and flashlights, these are great.
- I have had these only for a week I cannot ... I have had these only for a week I cannot tell you anymore
- These batteries are great, I used Duracell before for my remotes etc These batteries are great, I used Duracell before for my remotes etc, but i am impressed with these, I am getting much longer life from them. I am updating this review, this will be the only battery I use, they last a long time, and are a good price!
- Battery Performance You know, I really like the Amazon line of batteries. At first I was hesitant on using anything other than Duracell, I tried these and especially for the price you simply cannot go wrong....
- Poor power Terrible life. These are poorest quality batteries I have ever purchased both AA and AAA
- Amazon Batteries I always try to buy Amazon rechargeable batteries. Good price and lasts a long time between charges. Saves a bit of the environment compared to disposable ones.

2.2.5 Analyse d'erreurs commises (7 points)

Complétez la fonction plot_example qui:

- Prend en entrée une liste de jetons provenant d'un exemple.
- Produit un graphique qui pour chaque jeton, affiche la valeur $P(x_i|y=5)$ et $P(x_i|y=1)$

Pour vous faciliter le travail, utiliser barh de pandas et non de matplotlib:

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.barh.html>

a) Exécutez votre fonction avec une phrase au choix dont la vraie valeur est 5 mais la valeur prédictée est de 1. (4 points)

```
In [ ]: import numpy as np

def plot_example(tokens):
    words, prob_5, prob_1 = [], [], []
    for token in tokens:
        # Probabilities are for words in the vocabulary.
        # Not all words in X_train or y_train are in the vocabulary because of the threshold
        # TODO: Confirm that it's ok to omit words that are not in the vocabulary
        if token in features_prob.columns:
            words.append(token)
            prob_5.append(features_prob.loc[5][token])
            prob_1.append(features_prob.loc[1][token])

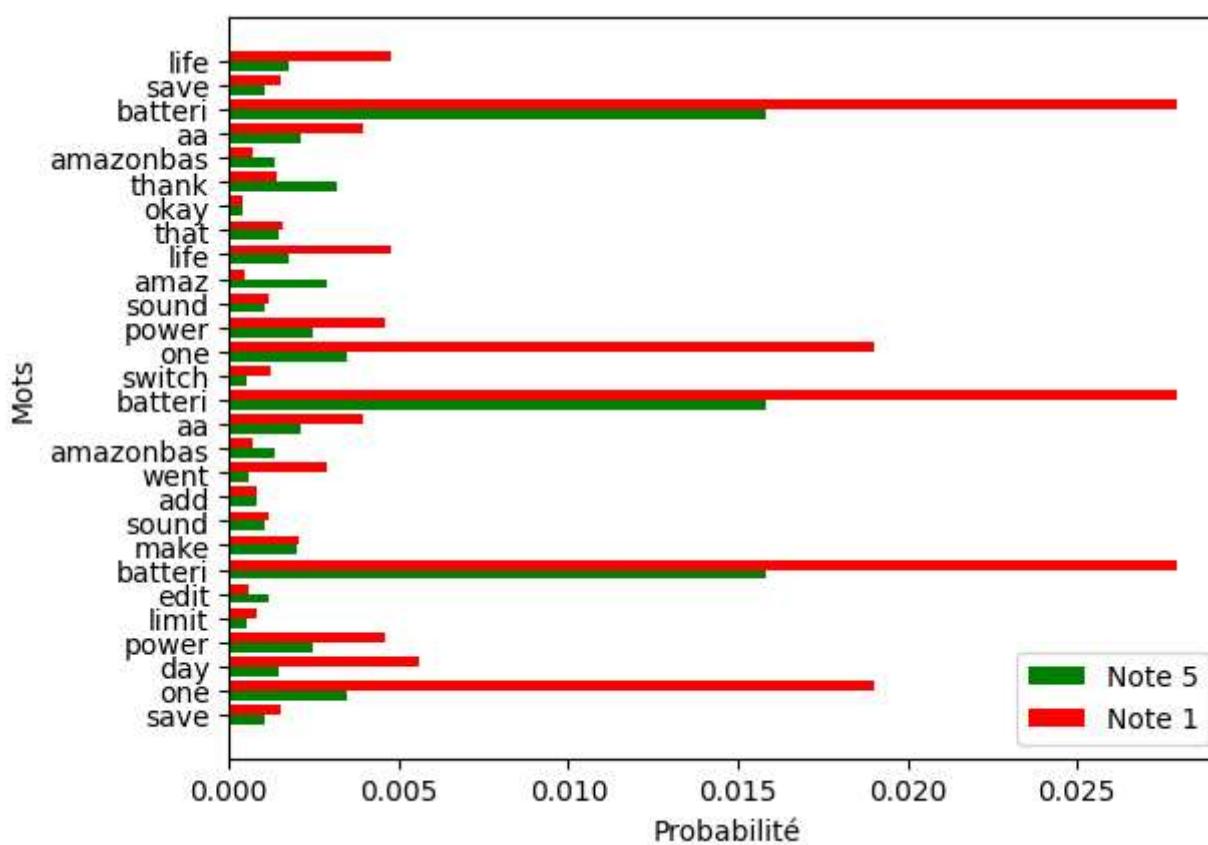
    fig, ax = plt.subplots()
    ax.barh(y=np.arange(len(words)) - 0.2, width=prob_5, height=0.4, color='green', label='Note 5')
    ax.barh(y=np.arange(len(words)) + 0.2, width=prob_1, height=0.4, color='red', label='Note 1')
    ax.set_yticks(np.arange(len(words)), words)
    ax.legend()
    plt.xlabel('Probabilité')
    plt.ylabel('Mots')
    plt.show()

# print original comment
print('Commentaire original: \n{}\n'.format(data.iloc[comments_5_1_index[2]]['text_original']))
print('Tokens: {}'.format(X_test[X_test.index == comments_5_1_index[2]].values[0]))
plot_example(X_test[X_test.index == comments_5_1_index[2]].values[0])
```

Commentaire original:

Saved Me From Bear Attack So... One day I was camping in the Wild West on the Rocky Mountains in Colorado... I was carrying my trusty Power Rangers limited edition collectors sword. Out of nowhere... A bear attacked me. I grabbed my trusty sword, but it was out of batteries so it did not make the swoosh sounds that adds to the effect of my swing. So I hurriedly went to my trusty AmazonBasics AA batteries, and switched out the ones in my Power Rangers sword. My swooshing sounds were amazing, and I escaped with my life (not my leg but that's okay). Thank you AmazonBasics AA Batteries for saving a life.

Tokens: ['save', 'bear', 'attack', 'one', 'day', 'camp', 'wild', 'west', 'rocki', 'mountain', 'colorado', 'carri', 'trusti', 'power', 'ranger', 'limit', 'edit', 'collector', 'sword', 'nowher', 'bear', 'attack', 'grab', 'trusti', 'sword', 'batteri', 'make', 'swoosh', 'sound', 'add', 'effect', 'swing', 'hurriedli', 'went', 'trusti', 'amazonbas', 'aa', 'batteri', 'switch', 'one', 'power', 'ranger', 'sword', 'swoosh', 'sound', 'amaz', 'escap', 'life', 'leg', 'that', 'okay', 'thank', 'amazonbas', 'aa', 'batteri', 'save', 'life']



b) Suite à cette analyse, pouvez-vous voir une tendance dans les exemples qui sont prédisits comme faisant partie de la classe 1 mais faisant réellement partie de la classe 5 ? (3 points)

Dans l'exemple ci-haut, on note que plusieurs mots se rapportent à la note 1.

Le classifieur MultinomialNB n'effectue pas de vote majoritaire. Il effectue plutôt un calcul interne qui prend en considération les probabilité de chaque mot de la phrase.

Ce calcul peut être influencé lorsqu'on se retrouve avec plusieurs mots attribués à une classe fausse.

Certains jetons ont tendance à avoir beaucoup plus de poids dans la prédiction que d'autres. Le jeton batteri soit principalement responsable de la confusion du model puisque ce jeton a la plus grande probabilité $P(x_{ij}|y=1)$. Cela signifie que notre modèle n'est pas bien entraîné pour détecter le contexte des mots autour d'une phrase (ce qui est normal puisque nous avons utilisé un bag of words)

2.3 Régression logistique (4 points)

Entraînez un modèle de [régression logistique](#) à l'aide de SKLearn en utilisant les données produites en 1.6 et affichez sa performance avec les mêmes métriques que précédemment.

```
In [ ]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, make_scorer

clf_lr = LogisticRegression()
clf_lr.fit(X_train_matrix, y_train)
```

```
Out[ ]: LogisticRegression
LogisticRegression()
```

```
In [ ]: y_pred_LR = pd.Series(clf_lr.predict(X_test_matrix))
print(classification_report(y_test, y_pred_LR, digits=4))
print(f"Weighted F1: {f1_score(y_test, y_pred_LR, average='weighted')}\n")
```

	precision	recall	f1-score	support
1	0.8066	0.8585	0.8318	311
3	0.7516	0.7419	0.7468	310
5	0.8561	0.8106	0.8328	301
accuracy			0.8037	922
macro avg	0.8048	0.8037	0.8038	922
weighted avg	0.8043	0.8037	0.8035	922

Weighted F1: 0.8035117847343994

2.4 MLP (4 points)

Entraînez un modèle neuronal de type [Multi-layer Perceptron classifier](#) à l'aide de SKLearn en utilisant les données produites en 1.6. Affichez sa performance avec les mêmes métriques que précédemment.

```
In [ ]: from sklearn.neural_network import MLPClassifier

clf_mlp = MLPClassifier(max_iter=1000)
clf_mlp.fit(X_train_matrix, y_train)
```

```
Out[ ]: 
  ▾ MLPClassifier
    MLPClassifier(max_iter=1000)
```

```
In [ ]: y_pred_MLP = pd.Series(clf_mlp.predict(X_test_matrix))
print(classification_report(y_test, y_pred_MLP, digits=4))
print(f"Weighted F1: {f1_score(y_test, y_pred_MLP, average='weighted')}\n")

      precision    recall   f1-score   support

      1       0.8291    0.8424    0.8357      311
      3       0.7121    0.7419    0.7267      310
      5       0.8339    0.7841    0.8082      301

  accuracy                           0.7896      922
macro avg       0.7917    0.7895    0.7902      922
weighted avg    0.7913    0.7896    0.7901      922

Weighted F1: 0.7900879823249253
```

3. Amélioration de modèle (30 points)

Cette dernière partie consistera à améliorer votre modèle de deux façons différentes.

Tout d'abord, vous effectuerez une recherche d'hyper-paramètres avec une validation croisée en utilisant une grille de recherche (GridSearch). Ensuite, vous réaliserez de l'extraction d'attributs (feature extraction) afin d'entraîner un nouveau modèle.

3.1 Recherche d'hyper-paramètres et validation croisée (5 points)

La classe `GridSearchCV` permet d'explorer toutes les combinaisons possibles d'hyper-paramètres que l'on spécifie afin de trouver la configuration optimale. De plus, il est tout à fait possible de fusionner les paramètres du pré-traitement et ceux du classificateur en utilisant la classe `Pipeline`. Pour la rédaction de votre code, vous avez la possibilité de vous référer au tutoriel du cours.

a) Dans cette phase, l'objectif est de découvrir une configuration optimale pour le modèle `LogisticRegression` en conjonction avec la technique de vectorisation TF-IDF. Cette recherche devra être guidée par la métrique du F1-score pondéré (weighted F1). Vous devrez aussi effectuer une exploration de paramètres sur au moins deux attributs liés à TF-IDF et deux paramètres de la régression logistique. Affichez ensuite la performance finale du modèle optimal ainsi que ses paramètres. (3 points)

```
In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import f1_score, make_scorer
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(vocabulary=voc)),
    ('clf_lr', LogisticRegression()),
])

parameters = {
    'tfidf_use_idf': (True, False),
    'tfidf_max_features': (500, 600),
    'clf_lr_C': [0.1, 1, 10],
    'clf_lr_max_iter': [500, 1000],
    'clf_lr_solver': ['lbfgs', 'saga']
}

clf_grid_search = GridSearchCV(pipeline, parameters, cv=5, n_jobs=-1, scoring=make_scorer(f1_score, average='weighted'))
clf_grid_search.fit(X_train.apply(lambda x: ' '.join(x)), y_train)
```

```
Out[ ]: 
  ▾ GridSearchCV
    ▾ estimator: Pipeline
      ▾ TfidfVectorizer
      ▾ LogisticRegression
```

```
In [ ]: y_pred_gs = clf_grid_search.predict(X_test.apply(lambda x: ' '.join(x)))

print("Meilleurs paramètres trouvés:\n", clf_grid_search.best_params_, "\n")
print("Meilleur score atteint: ", clf_grid_search.best_score_, "\n")
print(f"Weighted F1: {f1_score(y_test, y_pred_gs, average='weighted')}\n")
print(classification_report(y_test, y_pred_gs, digits=4))
```

```

Meilleurs paramètres trouvés:
{'clf_lr_C': 10, 'clf_lr_max_iter': 1000, 'clf_lr_solver': 'saga', 'tfidf_max_features': 500, 'tfidf_use_idf': False}

Meilleur score atteint: 0.8159059745543458

Weighted F1: 0.8068889517282264

precision    recall   f1-score   support
1      0.8338   0.8714   0.8522      311
3      0.7429   0.7548   0.7488      310
5      0.8475   0.7940   0.8199      301

accuracy          0.8069      922
macro avg       0.8081   0.8067   0.8070      922
weighted avg     0.8077   0.8069   0.8069      922

```

b) Quels sont les attributs que vous avez choisis et quels sont leurs valeurs optimales? (2 points)

Les attributs choisi sont :

'tfidf_use_idf': (True, False), 'tfidf_max_features': (500, 600), 'clf_lr_C': [0.1, 1, 10], 'clf_lr_max_iter': [500, 1000], 'clf_lr_solver': ['lbgfs', 'saga']

Les valeurs idéales pour trouvés sont:

{'clf_lr_C': 10, 'clf_lr_max_iter': 1000, 'clf_lr_solver': 'saga', 'tfidf_max_features': 500, 'tfidf_use_idf': True}

3.2 Extraction d'attributs (Feature extraction) avec ChatGPT (15 points)

ChatGPT peut être très utile pour donner des idées ou donner du squelette de code (lorsque c'est permis! :). Cette partie vous fait explorer l'utilisation de ChatGPT pour générer du code permettant d'extraire des attributs (feature extraction) à partir du texte des évaluations.

En utilisant ChatGPT ainsi que votre recherche personnelle, essayez de déterminer un ensemble d'attributs que vous pourriez utiliser pour représenter chaque évaluation. A vous de voir comment vous pouvez obtenir une réponse satisfaisante de ChatGPT.

a) Indiquez dans la cellule ci-dessous les descriptions d'attributs suggérées par ChatGPT ainsi que les vôtres. Différenciez clairement vos attributs - s'il y en a - de ceux de ChatGPT. (4 points)

Text-based Techniques

Bag-of-Words (BoW): A simple representation that counts the frequency of each word in each document. This approach doesn't consider the order of words but is fast and often effective.

Word Embeddings: Word2Vec, GloVe, and FastText are popular methods to convert words into dense vectors which capture semantic meanings based on the context in which words appear. You can average or sum these vectors to get a single vector for each comment.

Count Vectorizer: Similar to BoW but instead of term frequency, it uses the raw counts of words. It could be useful if the frequency of certain words correlates with the rating.

n-grams: Instead of considering single words, you could consider sequences of words or characters. For example, in a bigram model, "very good" and "not good" would be considered as distinct two-word sequences.

Sentiment Analysis Scores: You can use pre-built sentiment analysis models to give each comment a sentiment score and use it as a feature.

Topic Modeling: Techniques like Latent Dirichlet Allocation (LDA) can be used to extract the topics present in the comments. Then, you can use the topic distribution as features for each comment.

Statistical and Structural Features

Document Length: The number of words in the comment could be indicative of its rating.

Punctuation Count: The number of exclamation marks or question marks may correlate with the sentiment of a review.

Capitalization: Use of uppercase words could indicate strong sentiment.

Readability Scores: Metrics like Flesch-Kincaid or Gunning-Fog index could potentially correlate with the complexity or quality of a comment.

Hybrid Features

Feature Engineering: Combine existing features to create new ones, for example, the ratio of positive to negative words in a comment.

Feature Selection: Use techniques like Recursive Feature Elimination, SelectKBest, or model-specific feature importances to retain only the most informative features.

Ensemble Methods: Use different feature sets for different models and then ensemble the models for final prediction. For example, one model could use BoW features and another could use Word Embeddings.

Remember that the usefulness of each feature can depend on your specific dataset and problem, and it's often necessary to experiment with various combinations to achieve the best performance.

b) Indiquez ci-dessous le code généré par ChatGPT que vous avez décidé de conserver pour représenter chaque évaluation. (2 points)

```
In [ ]: import numpy as np
from transformers import BertTokenizer, BertModel
import torch
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")

def extract_bert_features(texts, model, tokenizer):
    features = []
    for text in texts:
        inputs = tokenizer(text, padding=True, truncation=True, max_length=512, return_tensors="pt")
        with torch.no_grad():
            outputs = model(**inputs)
        features.append(outputs.last_hidden_state.mean(dim=1).squeeze().numpy())
    return np.array(features)
```

c) Il se peut que le code généré ait besoin d'être adapté à notre jeu de données. Si c'est le cas, corrigez le code et montrez le résultat après vos correction dans la cellule ci-dessous. Le code final devrait être une fonction qui vous retourne, pour un document, un dictionnaire d'attributs et leurs valeurs. N'oubliez pas d'indiquer votre propre code s'il y en a. (5 points)

```
In [ ]: from sklearn.model_selection import train_test_split
X_train_original, X_test_original, y_train_original, y_test_original = train_test_split(data["text_original"], data["rating"], test_size=0.2, random_state=42)

X_train_bert = extract_bert_features(X_train_original, model, tokenizer)
X_test_bert = extract_bert_features(X_test_original, model, tokenizer)
```

d) Utilisez le code corrigé ci-dessus pour entraîner un modèle MLP avec votre nouvelle représentation des évaluations. Affichez sa performance. (4 points)

```
In [ ]: from sklearn.metrics import f1_score, make_scorer
mlp = MLPClassifier(max_iter=1000)

mlp.fit(X_train_bert, y_train_original)
y_pred_bert = mlp.predict(X_test_bert)

print(f1_score(y_test_original, y_pred_bert, average='weighted'))
print(classification_report(y_test_original, y_pred_bert))
```

```
0.8368453988944957
          precision    recall  f1-score   support
1         0.87      0.92      0.89      311
3         0.78      0.79      0.78      310
5         0.87      0.80      0.83      301

accuracy                           0.84      922
macro avg       0.84      0.84      0.84      922
weighted avg    0.84      0.84      0.84      922
```

```
In [ ]: from sklearn.linear_model import LogisticRegression

clf_lr = LogisticRegression(max_iter=1000)
clf_lr.fit(X_train_bert, y_train_original)
y_pred_bert = clf_lr.predict(X_test_bert)
print(f1_score(y_test_original, y_pred_bert, average='weighted'))
print(classification_report(y_test_original, y_pred_bert))
```

```
0.8135479706405389
          precision    recall  f1-score   support
1         0.84      0.90      0.87      311
3         0.77      0.74      0.75      310
5         0.84      0.81      0.82      301

accuracy                           0.81      922
macro avg       0.81      0.81      0.81      922
weighted avg    0.81      0.81      0.81      922
```

3.3 Amélioration du modèle en 3.2 (10 points)

Il est possible que les résultats obtenus au numéro précédent ne soient pas très élevés.

a) Trouvez une manière d'utiliser ces attributs avec d'autres éléments afin d'**au moins** obtenir une meilleure performance que n'importe quel score obtenu au numéro 2.x , sans faire de recherche d'hyper-paramètres. Essayez d'obtenir la meilleure performance possible. Vous êtes libres d'utiliser n'importe quel algorithme de ce laboratoire. Affichez le code et les performances de votre modèle. (8 points)

```
In [ ]: X_train_combined = np.hstack((X_train_bert, X_train_matrix.toarray()))
X_test_combined = np.hstack((X_test_bert, X_test_matrix.toarray()))
```

```

mlp = MLPClassifier()

mlp.fit(X_train_combined, y_train_original)

y_pred = mlp.predict(X_test_combined)

print(f1_score(y_test_original, y_pred, average='weighted'))
print(classification_report(y_test_original, y_pred))

```

0.8589614775427304

	precision	recall	f1-score	support
1	0.90	0.92	0.91	311
3	0.80	0.82	0.81	310
5	0.88	0.84	0.86	301
accuracy			0.86	922
macro avg	0.86	0.86	0.86	922
weighted avg	0.86	0.86	0.86	922

In []: `from sklearn.linear_model import LogisticRegression`

```

clf_lr = LogisticRegression(max_iter=1000)
clf_lr.fit(X_train_combined, y_train_original)

y_pred = pd.Series(clf_lr.predict(X_test_combined))

print(f1_score(y_test_original, y_pred, average='weighted'))
print(classification_report(y_test_original, y_pred, digits=4))

```

0.8342733858213167

	precision	recall	f1-score	support
1	0.8614	0.9196	0.8896	311
3	0.7881	0.7677	0.7778	310
5	0.8542	0.8173	0.8353	301
accuracy			0.8351	922
macro avg	0.8346	0.8349	0.8342	922
weighted avg	0.8344	0.8351	0.8343	922

b) Quelles sont vos conclusions concernant l'utilisation de ChatGPT et les représentations possibles des documents ? (2 points)

Il y a plusieurs façons de représenter des documents et plusieurs techniques d'extraction d'attribut de ceux-ci. ChatGPT est très bon pour nous donner des techniques à haut niveau telles que les différentes méthodes d'extraction possibles. Il peut aussi facilement et efficacement nous donner le code correspondant à la technique. Il faut aussi être très spécifique et descriptif afin d'obtenir un bon résultat avec ChatGPT. Ce n'est pas nécessairement très bon pour optimiser une réponse fonctionnelle. Par exemple lorsque je lui ai demandé comment faire pour optimiser le score obtenu avec Bert il a tout simplement réécrit des méthodes connues d'optimisation de modèle tel que le GridSearch.