NLP — Raport de projet

Groupe 7

20 mai 2024

Table des matières

1 Introduction

Ce rapport a pour objectif de présenter les travaux menés dans le cadre du projet de Traitement Automatique des Langues. Celui-ci s'appuie sur le jeu de données Wikipedia Movie Plots constitué de synopsis de films en tout genre extraits de Wikipédia.

2 Présentation du jeu de données

2.1 Présentation du jeu de données

But Ce corpus a été collecté par l'utilisateur Kaggle JustinR dans le but de promouvoir un jeu de données pouvant être utilisé dans diverses tâches de traitement automatique des langues comme

— la recommandation de films basée sur le contenu,
— la génération de synopsis de films,
— la récupération d'informations (rattacher un film à une description),
— ou encore la classification de texte (attribuer un genre à un synopsis).

La collecte de ce corpus est à l'initiative de l'utilisateur et n'a pas été financée.

Situation Les documents ont été rédigés de manière collaborative sous l'encyclopédie en ligne Wikipédia. De son ouverture en 2001 à aujourd'hui, l'ensemble des textes récoltés ont pu être édités par une grande diversité d'utilisateurs dans un but commun d'information et de synthèse des connaissances.

Variété linguistique Les documents récoltés sont rédigés en anglais.

Démographie des intervenants En raison de la nature collaborative de Wikipédia, il est difficile d'attribuer précisément les informations à des auteurs spécifiques. Le site accueille une diversité considérable parmi ses contributeurs, englobant ainsi une pluralité d'âges, de genres et de nationalités parmi ceux qui rédigent les articles.

Processus de collecte Le jeu de données constitue un corpus d'une taille de 81 mégaoctets. Celui-ci a été collecté librement et n'a pas été sous-échantillonné. Aucune information au sujet d'un éventuel pré-traitement n'étant fournit, la donnée est supposée brute. Il n'existe qu'une unique version de ce jeu datant d'il y a 6 ans (2018).

Processus d'annotation Chaque synopsis est accompagné d'une annotation indiquant le genre cinématographique correspondant. Cette donnée peut être extraite avec plus ou moins de facilité à partir des pages Wikipédia. En effet, constate que le genre d'un film n'est généralement mentionné que dans la phrase d'introduction, qui prend souvent la forme suivante :

> "`Titre du film` is a `Année de sortie` `Genres du film` film [...]"

Par exemple,

> "Combat Wombat is a 2020 Australian 3D computer-animated superhero film [...]"

Nous ne disposons pas d'informations supplémentaires concernant ce processus. (On suppose qu'il est automatisé par un programme.)

Distribution Ce jeu de données est exploitable sous la license CC BY-SA 4.0.

2.2 Présentation des tâches

Pour ce projet, nous allons nous concentrer sur les deux tâches suivantes :

1. Une tâche de classification multi-classe : à partir d'un synopsis donné, déterminer le genre auquel celui-ci appartient.
2. Une tâche de génération de texte : en l'occurrence générer un synopsis de film à partir d'un genre donné.

3 Pré-traitement des données

Sélection Tout d'abord, nous procédons à la sélection des données pertinentes en conservant uniquement les colonnes "Genre" et "Plot" du jeu de données, éliminant ainsi les autres colonnes telles que "Release Year", "Origin/Ethnicity", "Director", "Cast", etc. Cette décision découle de la nécessité de concentrer notre analyse sur les seules informations requises pour nos tâches de classification et de génération, à savoir le synopsis et le genre.

Filtrage Puis, lors de l'analyse de la répartition des classes, nous constatons un nombre significatif de documents étiquetés comme "unknown". Ces documents, du fait de leur absence d'annotations exploitables pour l'apprentissage de nos modèles, sont donc exclus à leur tour du jeu de données.

Normalisation Pour continuer, nous entreprenons un traitement des annotations visant à normaliser les caractères et à éliminer le bruit potentiel, tel que les espaces superflus en début ou en fin de texte, ainsi que les caractères spéciaux. Ainsi, le jeu de données obtenu présente des annotations composées exclusivement d'espaces et de lettres minuscules.

Similarités Suite à cette étape de nettoyage, nous identifions que certains genres sont représentés de manière similaire malgré des formulations différentes :

— "romantic" et "romance"
— "sci-fi" et "science fiction"
— "comedy musical" et "musical comedy" (différence dans l'ordre des mots)

Substitutions Nous décidons alors de procéder à des substitutions manuelles sur les occurrences identifiées dans les classes les plus significatives, telles que le remplacement de "romantic" par "romance" et de "science fiction" par "scifi". De plus, nous réorganisons les mots dans l'ordre alphabétique afin de regrouper les genres similaires, tels que "comedy musical" et "musical comedy", en une seule catégorie.

Dillution Enfin, nous observons que certains genres sont décrits de manière verbeuse, parfois sous forme de phrases complètes. Dans le souci de minimiser la perte de données, cruciales pour l'entraînement de nos modèles, nous développons un algorithme visant à identifier, au sein des annotations verbeuses, les occurrences correspondant aux genres les plus représentées afin de les y incorporer.

4 Analyse exploratoire des données

Le jeu de données initial prend une forme tabulaire constituée d'un total de **34 886** lignes et **8** colonnes détaillées ci-dessous.

Colonne	**Description**
Release Year	Année de sortie du film
Title	Titre du film
Origin/Ethnicity	Origine du film (American, Bollywood, Tamil, etc.)
Director	Réalisateur(s) et réalisatrice(s) du film
Cast	Acteurs et actrices principaux
Genre	Genre(s) du film
Wiki Page	Adresse de la page Wikipédia d'où le synopsis à été extrait
Plot	Synopsis du film

Une fois le jeu de données pré-traité, on retient alors **26 164** documents (soit une diminution de 25 % par rapport au jeu initial). Cette perte s'explique notamment par le volume important de documents dont le genre est étiqueté comme *unknown* représentant 17 % du corpus.

Ces documents retenus cumulent alors un un total de **11 499 934 tokens** répartis au sein de **501 008 phrases** identifiés respectivement par `word_tokenize` et `sent_tokenize` de la bibliothèque NLTK. Ces tokens forment un **vocabulaire** de taille **138 321** dont les fréquences sont illustrées sur la figure 1.

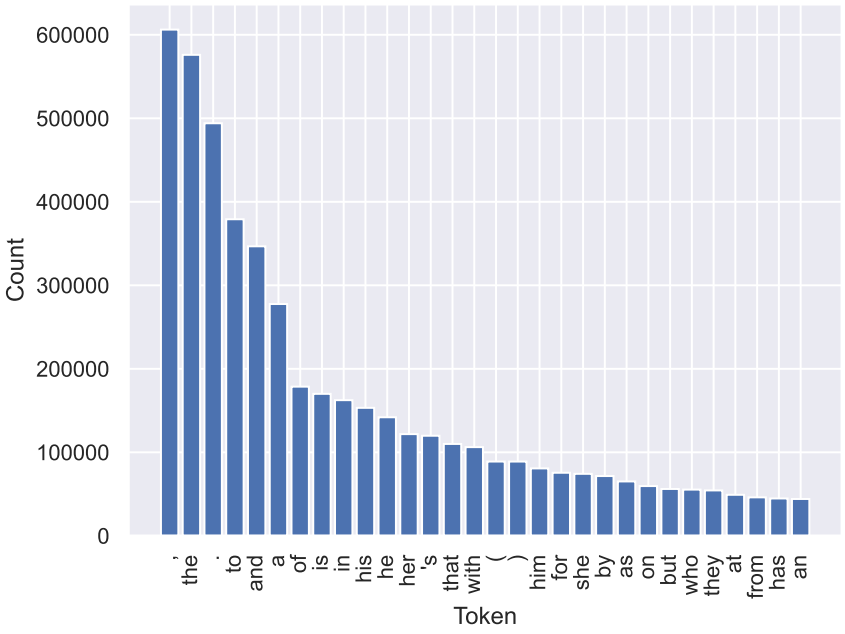
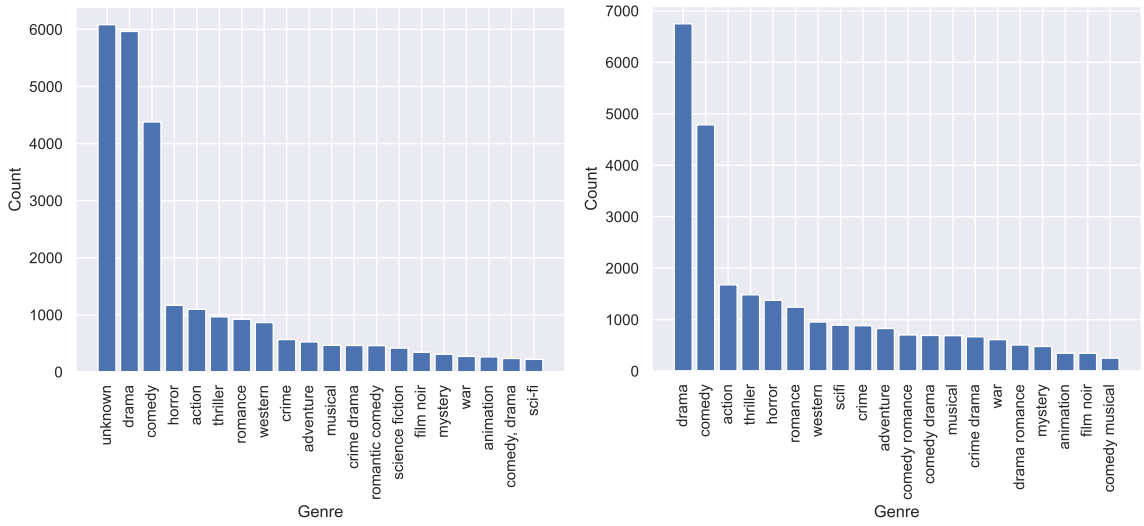


FIGURE 1 – Fréquence des tokens

Enfin, pour ce qui est de la distribution des classes à prédire, on observe parmi les 20 plus importantes des genres se démarquant fortement comme le dramatique ou la comédie, un biais

qu'il faudra prendre en considération lors de la conception et l'évaluation des performances de nos modèles.



(a) sans pré-traitement

(b) avec pré-traitement

FIGURE 2 – Distribution des classes

5 Entraînement des modèles

5.1 Régression logistique

L'un des premiers modèles de classification sur lesquels nous avons travaillé est un modèle de régression logistique. Ce modèle s'est développé au cours de trois itérations, chacune apportant des ajustements et des améliorations visant à mesurer leur impact sur les performances.

5.1.1 Première itération

La première itération de ce modèle utilise une *pipeline* relativement simple : chaque document est mis en lettres minuscules et tokenisé avec `word_tokenize` de NLTK, puis vectorisé en une représentation par sac de mots.

Cette représentation est ensuite utilisée dans un modèle de régression logistique sans pénalité, avec un nombre maximal d'itérations fixé à 1 000.

De plus, en raison du déséquilibre des classes observé lors de l'analyse exploratoire des données, une contrainte de poids est ajoutée au modèle pour équilibrer l'apprentissage sur le nombre d'échantillons par classe.

5.1.2 Deuxième itération

Nous nous intéressons ici aux limites de notre représentation par sac de mots. Certains termes comme "le", "on", "pour", "vous", etc., n'apportent pas de sens particulier mais sont tout de même inclus dans la représentation. Ces mots sont appelés des *stop words*.

Dans le but d'améliorer les performances du modèle, nous choisissons d'éliminer ces termes non informatifs afin que le modèle puisse se concentrer uniquement sur les termes pertinents du texte. Pour cela, nous utilisons la liste de *stop words* anglais de NLTK.

De plus, pour réduire la variation entre les mots de même racine (par exemple : "run" et "running"), nous ajoutons une étape de lemmatisation (avec `WordNet` de NLTK) à notre tokenizer après `word_tokenize`.

Nous prévoyons que la concentration des termes via la lemmatisation ainsi que la suppression des *stop words* permettront au modèle d'apprendre plus rapidement et efficacement les poids pour les termes les plus distinctifs de chaque catégorie.

5.1.3 Troisième itération

Enfin, pour comparer nos performances à une représentation plus avancée, nous remplaçons la représentation par sac de mots par une représentation TF-IDF. En effet, l'élimination des *stop words* ne garantit pas à elle seule que notre modèle accorde plus de poids aux termes pertinents pour les genres donnés. En prenant en compte la fréquence à laquelle un terme apparaît dans le corpus, nous prévoyons une amélioration de la précision du modèle.

5.2 Bayésien naïf

Un autre modèle de classification que nous avons étudié est le classifieur bayésien naïf. Ce modèle à été étudié sous six itérations : trois basées sur le modèle `MultinomialNB` de scikit-learn, et trois sur un modèle qui a été implémenté de zéro. Tous les modèles utilisent le tokenizer `word_tokenize` de la bibliothèque NLTK.

Itération	Modèle	Minuscules	Lemmatisation	Stop words
1	`MultinomialNB`	Oui	Non	Non
2	`MultinomialNB`	Oui	`WordNetLemmatizer`	Non
3	`MultinomialNB`	Oui	`WordNetLemmatizer`	Oui
4	Fait main	Non	Non	Non
5	Fait main	Oui	Non	Non
6	Fait main	Oui	`WordNetLemmatizer`	Non

Les modèles `MultinomialNB` ont tous un paramètre de lissage, qui a été sélectionné suite à différents tests sur le F1-score de la première itération de ce modèle. La valeur finale de ce paramètre de lissage est de 0.1.

5.3 N-gram

Deux approches ont été prises pour les n-grams. La première, implémentée à la main, en trois itérations, produit à la fin les n-grams les plus fréquents pour les différents genres de films. La seconde approche, implémentée avec NLTK, sert à produire des phrases grâce à des probabilités tirées d'un corpus d'entraînement.

TABLE 1 – Itérations des tri-grams à la main.

Itération	Tokenization	Stop words	Ponctuation
1	Sans	Sans	Avec
2	Avec	Sans	Avec
3	Avec	Sans	Sans

5.4 Word2vec et tf-idf

Le premier usage de Word2vec a été de créer les vecteurs pour trouver les mots proches les uns des autres. Pour cela, on tokenize le corpus contenant tous les scénarios des films avec `word_tokenize` puis on entraîne un modèle de `gensim word2vec` dessus.

La deuxième utilisation à été l'entraînement de différents modèles pour classifier les films grâce à word2vec et une régression logistique. Trois entraînements ont été réalisés, deux avec le tokenizer `word_tokenize` et un avec `tiktoken gpt-4`. Les stops words ont aussi été retirés du dernier entraînement afin d'essayer d'améliorer les résultats.

5.5 GPT-2 : Tokenizer et model

5.5.1 Le modèle

Dans le but de trouver plusieurs modèle pour la comparaison, nous avons essayé d'utiliser le modèle pré-entraîné de GPT-2 ainsi que son tokenizer. C'est un modèle connu et qui permet de faire de la génération de texte. C'est un modèle de langage avancé qui a la capacité de généré des textes cohérents. De plus, il a été facilement adaptable de manière à utiliser notre jeu de données pour qu'il puisse donner un résultat qui nous intéresse. Il est aussi fortement paramétrable. Cependant, avec nos utilisation personnelles, nous nous sommes rendu compte qu'il avait une forte tendance à tomber dans l'overfitting.

5.5.2 Les paramètres

Pour ce qui est des paramètres, voici ceux que nous avons utilisé pour ce résultat :

Paramètres	**Valeur**	**Description**
Epochs	3	Nombre de passes sur les données d'entrée
Batch size	128	Taille des batches
Learning Rate	0.001	Taux d'apprentissage
Tokenizer	GPT2-medium Tokenizer	Fonction pour tokeniser les documents
Entrée	Genre de film	Genre du film géréné
Sortie	Plot	Résultat du modèle
Temps d'entraînement	8h30	avec 2 GPU

D'autres paramètres ont été essayés mais les résultats étaient soit des phrases qui ne veulent rien dire (avec trop peu d'entraînement) soit que des "," qui étaient peut-être le résultat d'une erreur lors de l'entraînement

5.6 Réseaux de neurones feedforwards et récurrent

Des réseaux de neurones feedforwards et récurrents sont aussi d'autres méthodes possible pour faire de la classification. Pour ces modèles, nous avons fait 4 itérations différentes :

5.6.1 Première itération

Pour la première itération, nous allons tokeniser les synopsis. Nous allons mettre en lettre minuscule les synopsis des films, puis enlever tout les caractères non alphabétiques. Dans un second temps, on tokenise cela avec `Tokenizer` de Keras, pour ensuite créer un vocabulaire avec `fit_on_text`, qu'on convertit en séquences de nombres.

Quant aux genres, nous les encodons en valeurs numériques. Nous commençons cette méthode de classification par un petit réseau de neurones pour expérimenter rapidement et simplement :

```
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=50))
model.add(GlobalAveragePooling1D())
model.add(Dense(50, activation='relu'))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

FIGURE 3 – Le premier modèle et ses spécificités

5.6.2 Deuxième itération

Pour cette deuxième itération, nous avons gardé le même pré-traitement et tokenisation que la première itération. Ce qui va changer ici est le réseau de neurones. Il est plus complexe, contient plus de dimensions dans chaque couches et donc possède de plus grandes capacités d'apprentissage.

```
model2 = Sequential()
model2.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=100))
model2.add(Bidirectional(LSTM(128, return_sequences=True)))
model2.add(GlobalAveragePooling1D())
model2.add(Dense(256, activation='relu'))
model2.add(Dropout(0.4))
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.4))
model2.add(Dense(len(label_encoder.classes_), activation='softmax'))
```

```
model2.compile(optimizer=Adam(learning_rate=1e-4), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model2.fit(X_train, y_train, epochs=3, batch_size=32, validation_data=(X_test, y_test))
```

FIGURE 4 – Le deuxième modèle et ses spécificités

5.6.3 Troisième itération

Quant à la troisième itération, nous avons complètement changé le tokenizer. À la place d'utiliser celui que nous avons fait, nous utiliserons directement le tokenizer de GPT-2, qui est déjà très

complet. On utilisera aussi un `attention_mask` qui permettra de s'assurer que les parties non pertinentes comme les paddings n'influent pas les calculs. Le modèle est le même que celui de la deuxième itération, ce qui permettra de savoir si une bonne tokenisation influe sur les résultats.

5.6.4 Quatrième itération

Pour cette 4ème itération, nous utilisons aussi le tokenizer GPT2, mais le réseau de neurones est quant à lui un réseaux de neurones récurrents. Il est particulièrement adapté pour les tâches de classification, on peut donc espérer une amélioration des résultats et prédictions de ce modèle par rapport aux précédents.

```
input_ids4 = Input(shape=(X4.shape[1],), dtype=tf.int32, name='input_ids')
attention_mask4 = Input(shape=(X4.shape[1],), dtype=tf.int32, name='attention_mask')

embedding_layer4 = Embedding(input_dim=tokenizer.vocab_size + 1, output_dim=128)(input_ids4)
masked_layer4 = Masking(mask_value=0)(embedding_layer4)
rnn_layer4 = SimpleRNN(128, return_sequences=False)(masked_layer4)
dropout_layer4 = Dropout(0.5)(rnn_layer4)
batch_norm_layer4 = BatchNormalization()(dropout_layer4)
dense_layer4 = Dense(64, activation='relu')(batch_norm_layer4)
dropout_layer42 = Dropout(0.5)(dense_layer4)
batch_norm_layer42 = BatchNormalization()(dropout_layer42)
output_layer4 = Dense(len(label_encoder.classes_), activation='softmax')(batch_norm_layer42)

model4 = Model(inputs=[input_ids4, attention_mask4], outputs=output_layer4)
```

```
model4.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
model4.fit([X_train4, train_attention_masks4], y_train4, epochs=10, batch_size=32, validation_data=([X_test4, test_attention_masks4], y_test4)
```

FIGURE 5 – Le quatrième modèle et ses spécificités

6 Évaluation des modèles, comparaison des performances

6.1 Régression logistique

On présente ici les F1-scores obtenus afin de comparer et mesurer l'impact des performances sur les trois itérations de modèles de régression logistiques développés.

	iter 1	iter 2	iter 3
action	0.40	0.40	0.37
adventure	0.34	0.35	0.38
animation	0.46	0.51	0.54
comedy	0.50	0.49	0.53
comedy drama	0.08	0.09	0.12
comedy musical	0.08	0.08	0.11
comedy romance	0.15	0.18	0.16
crime	0.21	0.22	0.25
crime drama	0.16	0.14	0.15
drama	0.52	0.51	0.53
drama romance	0.14	0.10	0.12
film noir	0.13	0.19	0.19
horror	0.52	0.52	0.59
musical	0.25	0.26	0.25
mystery	0.29	0.31	0.34
romance	0.23	0.24	0.27
scifi	0.62	0.60	0.65
thriller	0.21	0.28	0.28
war	0.52	0.53	0.50
western	0.77	0.70	0.81
accuracy	0.42	0.42	0.45
macro avg	0.33	0.33	0.36
weighted avg	0.41	0.41	0.44

On observe que la lemmatisation et l'élimination des *stop words* entre l'itération 1 et l'itération 2 n'a eu que très peu d'impact dans l'amélioration des performances de la représentation par sac de mots. En revanche, on remarque une augmentation globale de 3 % de précision dans le F1-score du modèle employant la représentation TF-IDF.

De plus, on observe que les synopsis étiquetés "western" ont généralement une bien meilleure précision que les autres genres. Cela peut s'expliquer par le fait que les synopsis de ce genre emploient un vocabulaire bien spécifique qui lui permet d'être facilement discriminé par le modèle.

Enfin, on peut lire sur la matrice de confusion (figure 6) que les genres "drama" et "comedy" sont souvent confondus avec tous les autres. Cela peut s'expliquer par le fait que le modèle soit statistiquement biaisé par le nombre important d'échantillons que ces deux genres représentent (en l'occurrence, "drama" et "comedy" représentent ensemble 44 % du corpus), et ce malgré l'équilibrage du poids de chaque classe.

6.2 Bayésien naïf

Voici les scores F1 obtenus sur toutes les itérations des classifieurs bayésiens naïfs :

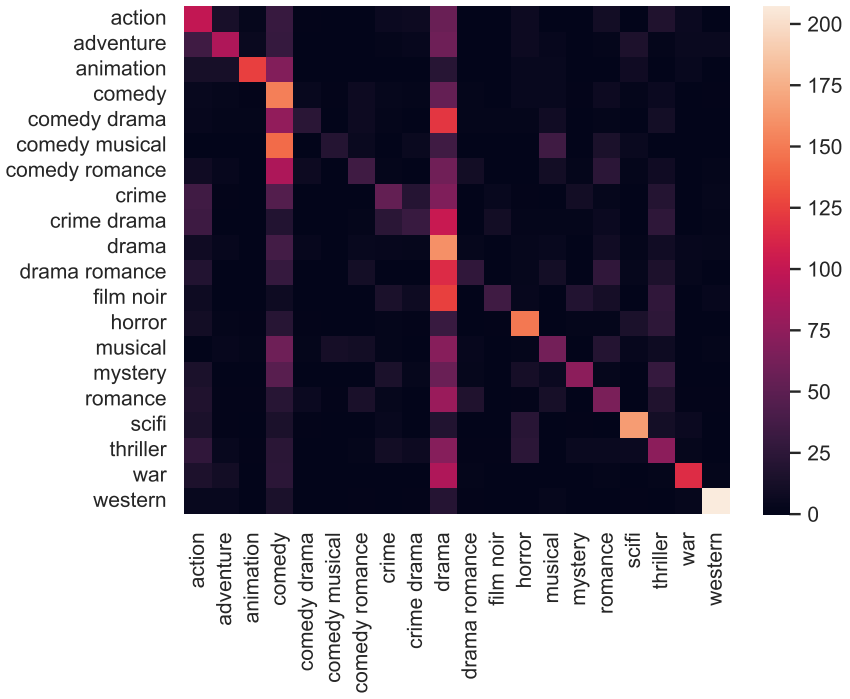


FIGURE 6 – Matrice de confusion du modèle utilisant TF-IDF

	iter 1	iter 2	iter 3	iter 4	iter 5	iter 6
action	0.40	0.40	0.41	0.00	0.00	0.00
adventure	0.45	0.47	0.46	0.00	0.00	0.00
animation	0.52	0.51	0.52	0.00	0.00	0.00
comedy	0.53	0.53	0.53	0.00	0.00	0.00
comedy drama	0.07	0.09	0.10	0.00	0.00	0.00
comedy musical	0.06	0.06	0.06	0.00	0.00	0.00
comedy romance	0.14	0.15	0.12	0.06	0.06	0.06
crime	0.24	0.24	0.24	0.00	0.00	0.00
crime drama	0.15	0.16	0.15	0.00	0.00	0.00
drama	0.52	0.52	0.52	0.14	0.14	0.14
drama romance	0.10	0.10	0.10	0.00	0.00	0.00
film noir	0.16	0.17	0.16	0.00	0.00	0.00
horror	0.60	0.60	0.59	0.00	0.00	0.00
musical	0.24	0.26	0.23	0.00	0.00	0.00
mystery	0.39	0.37	0.36	0.00	0.00	0.00
romance	0.34	0.33	0.33	0.00	0.00	0.00
scifi	0.66	0.64	0.64	0.00	0.00	0.00
thriller	0.29	0.27	0.27	0.00	0.00	0.00
war	0.62	0.62	0.61	0.00	0.00	0.00
western	0.83	0.84	0.83	0.00	0.00	0.00
accuracy	0.46	0.46	0.45	0.05	0.05	0.05
macro avg	0.37	0.37	0.36	0.01	0.01	0.01
weighted avg	0.44	0.45	0.44	0.04	0.04	0.04

On peut remarquer que les trois modèles faits à la main, et qui n'ont pas de lissage, ont de très mauvais résultats comparés aux modèles `MultinomialNB`. En effet, ces modèles sélectionnent majoritairement la catégorie "drama", qui est la catégorie ayant le plus de représentants, et n'arrivent à distinguer qu'un seul autre genre de film.

Les modèles `MultinomialNB`, de leur côté, ont une meilleure capacité à distinguer une majorité de catégories. Cependant, les sous-catégories comme "comedy drama", "comedy musical" et "drama romance" ont plus de mal à être distinguées.

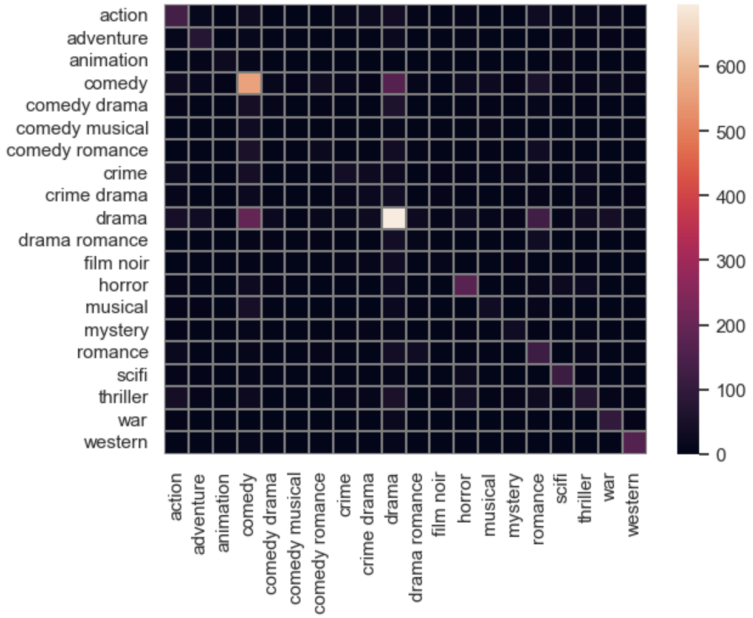


FIGURE 7 – Matrice de confusion de l'itération 2 du modèle MultinomialNB

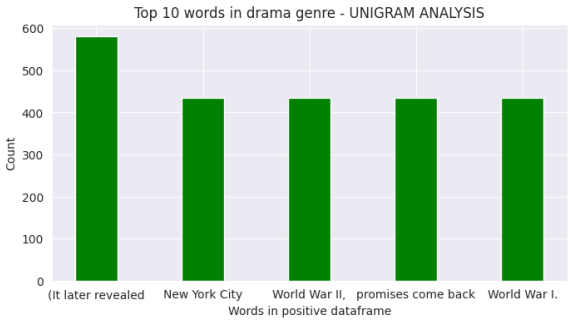
La matrice de confusion (figure 7) montre que le plus grand nombre de confusions se déroulaient entre les catégories "comedy", "drama" and "romance", et que les sous-catégories de drama et comédies ont tendance à être classifiées comme leurs catégories générales.

6.3 N-gram

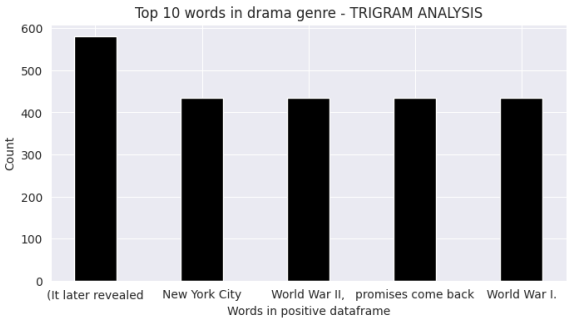
Voici des exemples de phrases générées grâce au modèle nltk :

```
['the police for a six-month tour of south america and returns',
 'the brooding landscape : the gesticulating passengers in the adirondacks .',
 'the north , one of these rich people would take a',
 'the old man , made easier by the local butterflies ,',
 'the forgiveness of jack on his return , he lies unidentified']
```

Pour les itérations des modèles "à la main", voici les différentes fréquences de tri-grams obtenues pour le genre "drama" (figure 9)



(a) Itération 1



(b) Itération 2

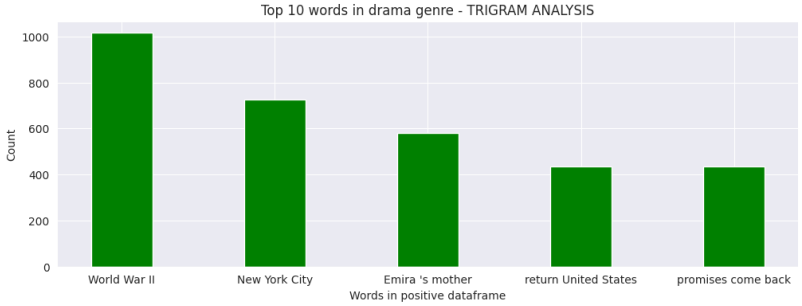


FIGURE 9 – Itération 3

On remarque que les deux premières générations de tri-grams produisent un résultat similaire et que donc, la tokenization appliquée n'avait que peu d'effet. Les tri-grams les plus pertinents sont cependant obtenus après tokenization et en enlevant la ponctuation du corpus qui a servis à la génération des tri-grams.

6.4 GPT-2 : Modèle

6.4.1 Résultats

La figure 10 montre l'évolution de la "loss" au fil des étapes de l'entraînement sur notre jeu de donnée.

Nous pouvons remarquer que la fonction de perte décroît bien au fil du temps indiquant que le modèle est bien en train d'apprendre. Des pertes significatives sont observées à deux reprises du au changement d'epoch. On peut imaginer que avec un nombre d'epoch supérieur, la tendance de la Loss continuerai.

Résultats par genre :

Comédie The film opens with a young man named Michael (Michael Rennie) who is a student at a local college. He is a bit of a loner, and is always in trouble with his friends. One day, he meets a girl named Sarah (Sarah Silverman), who is a student at the same college. They both fall in love

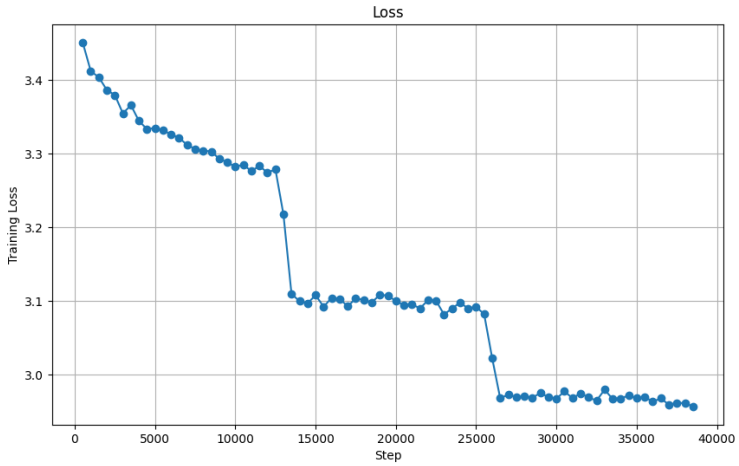


FIGURE 10 – Loss durant l'entraînement du modèle GPT-2

with each other, and they decide to get married. Michael's father, a lawyer, is a very strict man, and he is very strict with his son.
Michael's father is very strict with his son, and he is very strict with his father.
Michael's father is very strict with his son, and he is very strict with his father.
Michael's father is very strict with his son, and he is very strict with his father.
Michael's father is very strict with his son, and he is very strict with his father.
Michael's father is very strict with his son,

Action The film opens with a young man named Raju (Rajesh Khanna) who is a student of the same college as his father. Raju is a typical student who is always in trouble with the law. He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who is always in trouble with the law.
He is a typical student who

Nous pouvons remarquer que la génération de texte fini par boucler. Nous avons choisi de garder ce modèle malgré le fait qu'il ne soit pas "parfait" pour pouvoir comparer et mettre en évidence des problèmes qui sont récurrent dans ce genre de cas.

6.5 Word2vec

Les différents modèles n'ont pas réussi à converger malgré 2000 itérations lors de l'entraînement. Le modèle avec le tokenizer `word_tokenize` a été cependant plus efficace.

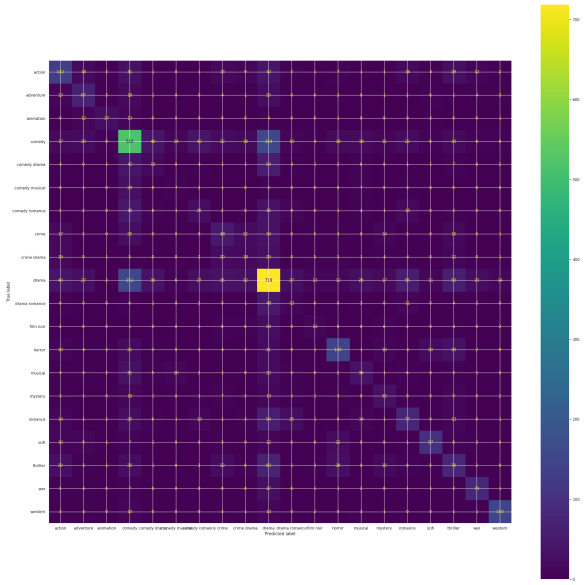


FIGURE 11 – Matrice de confusion du meilleur modèle

On remarque que les genres que le modèle classifie le mieux sont les genres les plus présents dans le corpus (comedy avec le carré vert et drama avec le carré jaune), malgré la présence des poids pour balancer ce problème lors de l'entraînement.

	Precision	Recall	F1-score	Support
action	0.40	0.38	0.39	350
comedy	0.49	0.56	0.52	943
comedy drama	0.06	0.05	0.06	139
drama	0.53	0.54	0.53	1366
horror	0.58	0.60	0.59	255
musical	0.29	0.25	0.27	148
scifi	0.63	0.57	0.60	174
thriller	0.28	0.26	0.27	293
war	0.53	0.53	0.53	127
western	0.81	0.81	0.81	205
accuracy			0.44	5233
macro avg	0.36	0.35	0.35	5233
weighted avg	0.43	0.44	0.44	5233

Pour finir sur word2vec, le modèle qui permet de donner les mots proches produisaient en revanche des résultats satisfaisants. Avec le mot "drama", les mots proches retournés étaient dans l'ordre de

similarité : "comedy, thriller, narrative, documentary", ce qui correspond à d'autres thèmes de films, proche du mot donné.

6.6 Réseaux de neurones feedforwards et récurrents

	iter 1	iter 2	iter 3	iter 4
action	0	0	0.35	0
adventure	0	0	0.14	0
animation	0	0	0.43	0
comedy	0.14	0	0.48	0
comedy drama	0	0	0	0
comedy musical	0	0	0	0
comedy romance	0	0	0.05	0
crime	0	0	0.02	0
crime drama	0	0	0.16	0
drama	0.45	0.41	0.41	0.41
drama romance	0	0	0	0
film noir	0	0	0.03	0
horror	0.19	0	0.47	0
musical	0	0	0.03	0
mystery	0	0	0	0
romance	0	0	0.23	0
scifi	0.32	0	0.49	0
thriller	0	0	0.18	0
war	0	0	0.46	0
western	0	0	0.50	0
accuracy	0.27	0.26	0.37	0.26
macro avg	0.05	0.02	0.22	0.02
weighted avg	0.16	0.10	0.33	0.10

Avec ce tableau, on constate que les 2 premiers modèles sont totalement inutilisables. Un tokeniser basique va influer sur les résultats des prédictions de nos réseaux de neurones. Pour le troisième modèle, il y a une nette amélioration des prédictions, et surtout réparties sur presque tout les genres. Une bonne tokenisation et l'ajout d'un masque d'intention (qui améliore l'efficacité et la performance) ont un rôle crucial sur l'amélioration des prédictions des modèles. Mais le résultat global est quand même bien moins performant que la régression logistique par exemple.

7 Difficultés rencontrées, limitations et pistes d'améliorations

7.1 Régression logistique

Pour être considérée comme consistante par scikit-learn, une liste de *stop words* donnée doit produire la même liste après l'application du tokenizer sur chaque élément. Ainsi, lors de l'itération 2, nous avons du trouver un moyen de rendre consistante la liste de *stop words* bruts fournie par NLTK avec notre tokeniser. Pour cela nous avons simplement généré une nouvelle liste issue de la tokenization des *stop words* originaux.

La principale difficulté rencontrée lors du développement de ces modèles concerne leur explicabilité. En effet, nous avons étudié l'utilisation de bibliothèques comme `eli5` qui permettent d'identifier les tokens du vocabulaire ayant une influence significative sur les décisions du modèle. Cependant, nous n'avons pas pu utiliser cette bibliothèque en raison d'un problème de compatibilité de versions entre `scikit-learn` et cette dernière. (Ce problème est dû au fait que `eli5` utilise une fonction dépréciée qui n'est plus supportée dans les versions récentes de `scikit-learn`.)

Enfin, bien que le modèle reste limité par la complexité d'associer des tokens à un genre spécifique, une piste d'amélioration possible serait de tester davantages de configurations d'hyperparamètres, notamment au niveau de la régularisation et de la tokenization.

7.2 GPT-2

Pour ce qui est de la mise en place du modèle de GPT-2 adapté pour notre jeu de données, nous avons eu quelques problèmes.

Le temps d'entraînement En effet, même avec un nombre d'epoch assez faible (3), un haut learning rate, et un grand batch size, le temps d'entraînement est assez long plus de 8h avec 2 GPU dédiés à cela. N'ayant pas accès physiquement à ce genre d'outils, nous sommes passé par un système en ligne : Kaggle. C'est un outil puissant qui permet de réaliser des tâches variées de data science et machine learning, mais toutes les 1h30/2h, Kaggle demande une vérification pour savoir si vous n'avez pas oublié de fermer votre session. Ce qui est un problème parce que lorsqu'il demande il stoppe la machine qui vous était dédiée, ce qui annule tout l'entraînement que vous auriez pu faire. Pour simplifier l'entraînement, nous avons réussi à faire des sauvegardes qui nous ont permit de reprendre l'entraînement malgré un arrêt, mais la solution a été découverte tard, et ce n'est pas forcément totalement stable.

En essayant de réduire le temps d'entraînement, nous avons essayé de faire varier certains paramètres mais sans succès pour le résultat (il était majoritairement composé de suite de ",").

Exportation Kaggle Nous avons également eu des problèmes avec l'exportation depuis Kaggle des éléments de sortie. Pour ce qui est du modèle sauvegardé, pour les quelques fois où nous avons essayé, l'exportation a échoué ou alors le fichier était corrompu lors de l'utilisation sur nos ordinateurs personnels. C'est pour cette raison que dans le dossier GPT-2 sur le GitHub, seul le notebook est présent. Les résultats sont visible à l'intérieur mais l'exécuter reviendrait à refaire l'entraînement (+ 300h). De plus, les fichiers générés dépassent la taille maximale pour un push Git.

7.3 Réseaux de neurones feedforwards et récurrent

Durant les différentes expérimentations, nous avons rencontré plusieurs difficultés :

Les réseaux de neurones stagnent. Avec les différents réseaux que nous vous avons montré, manipuler et changer le learning rate, le batch size, le nombre d'époques,... ne modifie presque pas l'apprentissage et le résultat des réseaux de neurones. L'accuracy reste entre 20 et 30 pourcents.

Même avec plusieurs méthodes, que ce soit en utilisant LSTM (qui est utile pour les réseaux de neurones de classification, qui permet de traiter des séquences à longueur variable, capture les dépendances long termes,...), ou encore en utilisant `class_weight` qui permet de gérer le déséquilibre dans les classes des données d'entraînement, ou alors un Recurrent Neural Network,... rien n'améliore les résultats des modèles.

En plus de cela, par exemple, LSTM ou RNN entraîne aussi une nette augmentation du temps d'apprentissage. On passe de 2 min par epoch à plus d'une heure. Son temps d'apprentissage a fait

que l'on ne pouvait pas se permettre de tester des plages de valeurs d'hyperparamètres (learning Rate, batch size, etc.).

Comme piste d'améliorations, on pourrait essayer de creuser plus loins en regardant des documents de recherche ou scientifiques plus poussés qui nous permettrait d'améliorer nos réseaux feedforwards ou récurrents.