CNAM Paris

RCP209

Apprentissage Statistique 2

Projet Individuel: Critiques IMDB

Peggi ABREU

22 septembre 2024

Table des Matières

1. Introduction

1.1. Contexte

Ce projet s'inscrit dans le cadre du cours CNAM RCP209 Apprentissage statistique 2: : modélisation décisionnelle et apprentissage profond.

L'objectif du projet est d'explorer des modèles d'apprentissage statistique (Machine Learning) permettant de classer des critiques de films comme étant positives ou négatives, en analysant le texte de la critique.

Il s'agit d'un exemple d'analyse de sentiments des textes dont le résultat est une classification binaire (1 ou 0) qui indique l'appartenance à la classe "positive" ou pas (donc négative sinon).

Le domaine de l'analyse des sentiments, ainsi que les classifications sont des problèmes d'apprentissage automatique largement répandus et avec des nombreuses applications.

Définition de l'Analyse des sentiments

"En informatique, l'opinion mining (aussi appelé sentiment analysis) est l'analyse des sentiments à partir de sources textuelles dématérialisées sur de grandes quantités de données (big data)…

L'objectif de l'opinion mining est d'analyser une grande quantité de données afin d'en déduire les différents sentiments qui y sont exprimés. Les sentiments extraits peuvent ensuite faire l'objet de statistiques sur le ressenti général d'une communauté."

https://fr.wikipedia.org/wiki/Opinion_mining

1.2. Jeu de Données

Nous allons utiliser l'ensemble de données IMDB "Large Movie Review Database". Le lien vers le dataset :

https://ai.stanford.edu/~amaas/data/sentiment/

Les détails sur les données seront dans la section Analyse des données.

1.3. Approches

Pour accomplir la tâche de classification demandée, nous allons explorer deux algorithmes de Machine Learning traditionnel et trois architectures Deep Learning.

Dans la section méthodologie, nous aborderons l'implémentation et évaluation de ces différents modèles, ainsi que quelques définitions générales.

1.3.1. Machine Learning Traditionnel

Dans la catégorie Machine Learning traditionnel, deux méthodes considérées comme étant efficaces pour les tâches de classification binaire, comme celle qui nous concerne (critique positive ou négative), seront explorées :

- ✓ Un modèle de Régression Logistique
- ✓ Un modèle basé sur des Forêts Aléatoires.

1.3.2. Deep Learning: Réseaux de Neurones Récurrents

Dans la catégorie Deep learning, nous allons d'abord tester deux exemples de Réseaux de Neurones Récurrents (RNNs) avec des types de cellules améliorées:

- ✓ Une architecture basée sur une couche Gated Récurrent Unit (GRU)
- ✓ Une architecture avec une couche Long Short-Term Memory (LSTM)

1.3.3.Deep Learning: Transfer Learning (module pré-entrainé)

Et pour finir, nous regarderons un exemple de "Transfer Learning" qui réutilise un module pré-entrainé d'Embedding (vectorisation) des mots, disponible de la librairie TFHub de Tensorflow.

2. Analyse des données

Le jeu de données contient un ensemble de critiques de films très polarisées (25.000 pour l'entraînement et 25.000 pour le test).

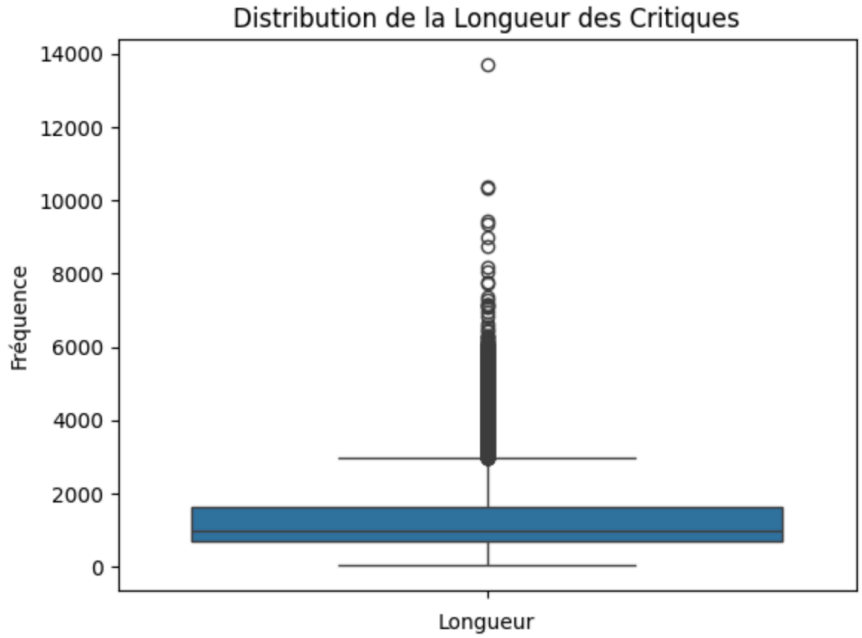
Chaque échantillon est constitué d'une phrase représentant une critique de film, et d'une étiquette (sentiment) qui indique si la critique est positive ou négative.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   critique   25000 non-null  object
 1   sentiment  25000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 390.8+ KB
```

	sentiment
count	25000.00000
mean	0.50000
std	0.50001
min	0.00000
25%	0.00000
50%	0.50000
75%	1.00000
max	1.00000

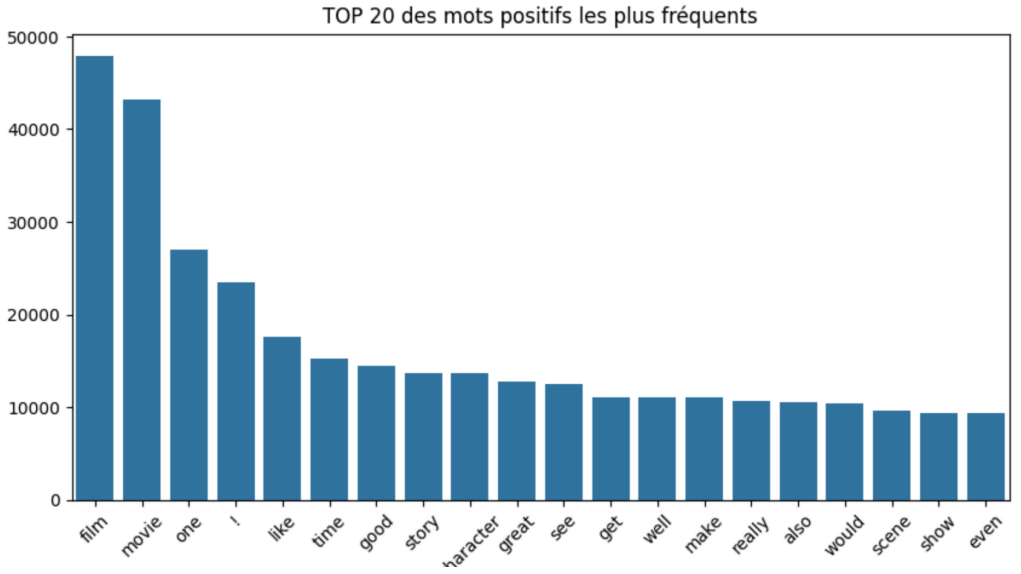
Les ensembles d'entrainement et de test sont équilibrés, avec un nombre égal de critiques positives et négatives.
Dans les données imdb, la phrase n'a subi aucun pré-traitement. L'étiquette est une valeur entière de 0 ou 1, où 0 correspond à une critique négative et 1 à une critique positive.

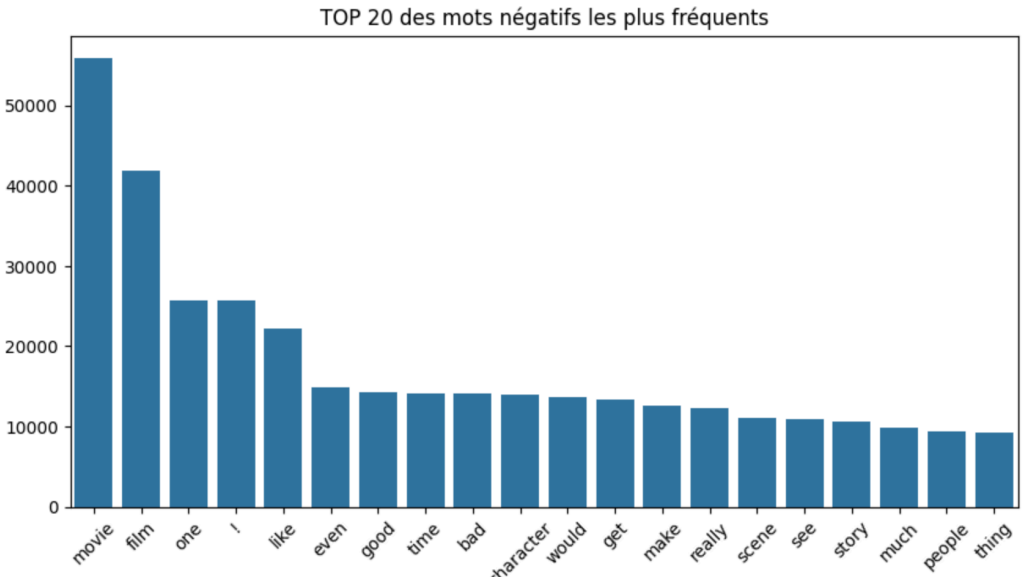
La moitié des critiques ont une longueur autour de 980 caractères. Un quart des critiques ont une longueur inférieure à 700 caractères. Il y a des critiques très longues pouvant aller jusqu'à environ 13700 caractères.



Un grand nombre de mots dans le TOP 20 des mots les plus fréquents se retrouvent à la fois dans les critiques positives et négatives. Ce qui veut dire que les modèles doivent aller au-delà de la simple fréquence des mots, grâce aux différentes techniques utilisées.

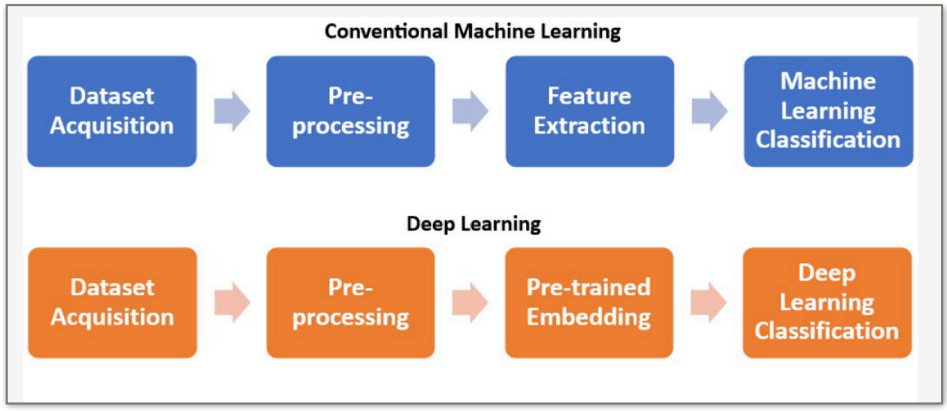
Par exemple le TF-IDF (Term Frequency - Inverse Document Frequency) qui sera utilisé dans les différentes approches pour vectoriser les données d'entrée, pondère l'importance des mots en fonction de leur fréquence d'apparition dans l'ensemble des données. Cela permet de réduire l'impact des mots très fréquents (qui apparaissent dans de nombreuses critiques positives et négatives) et de donner plus de poids aux mots spécifiques à chaque classe (positive ou négative).





3. Méthodologie

Les principales étapes à implémenter pour tester chaque approche sont représentées dans la figure ci-dessous, extraite de l'article [2] (voir section Références):



3.1.Régression Logistique

Ce modèle est facile à implémenter et rapide à entraîner. Voici les étapes envisagées:

Pré-processing : Il est souvent conseillé ou nécessaire de pré-traiter les données avec des techniques NLP classiques : tokenization, suppression des stop-words, ponctuation, etc.

Feature Extraction : Ce modèle peut être utilisé avec des représentations simples des textes comme TF-IDF (Term Frequency-Inverse Document Frequency) ou countVectorizer.

Ici le choix se porte sur TF-IDF car il associe un poids plus important au mots les plus significatifs par rapport à l'ensemble des mots, ce qui semble intéressant pour l'analyse des sentiments.

Classification : Modèle linéaire de Régression logistique. Ce modèle étant sujet à du surapprentissage, il est aussi souvent nécessaire d'ajouter une régularisation.

3.2.Forêt Aléatoire

C'est un autre algorithme d'apprentissage simple basé sur le principe d'avoir plusieurs avis afin de prendre la meilleure décision.

> *"Cette technique fait partie des méthodes d'apprentissage ensembliste, c'est-à-dire des méthodes qui utilisent la sagesse des foules".* - Wikipedia

L'algorithme se base sur plusieurs arbres de décision individuels (les estimateurs) entraînés sur des sous-ensembles du jeu de données légèrement différents.

La décision finale, c'est à dire la classification d'une nouvelle critique en positive ou négative, est prise par vote majoritaire, en assemblant les résultats des différents arbres. C'est ce procédé qui permet de rendre a prédiction extrêmement performante.

Voici les étapes envisagées:

Pré-processing : similaire à celui de la Régression logistique.

Feature Extraction : comme pour la régression logistique, TF-IDF sera utilisé pour la transformation des données d'entrée (les critiques) vecteurs compréhensibles par le modèle.

Classification : modèle à base d'arbres de décision (Random Forest). Pour éviter le surapprentissage sur ce genre de modèle, on peut jouer sur les hyperparametrès tels que le nombre d'arbres (nombre d'estimateurs) et la profondeur des arbres, entre autres. Seuls quelques paramètres usuels seront explorés afin de limiter le nombre de combinaisons et le temps de calcul nécessaire pour l'optimisation.

3.3. Réseaux de neurones récurrents (RNN)

Les réseaux de neurones récurrents permettent de modéliser des séquences temporelles, ce qui très utile pour les textes, car les mots sont considérés dans leur contexte.

Les cellules GRU et LSTM sont des versions améliorées des cellules RNN conçues pour résoudre une des limitations principales des RNNs qui est leur

mémoire trop courte! Les GRU et LSTM permettent de gérer des séquences plus longues que les RNN simples.

Les versions améliorées GRU et LSTM, avec une mémoire étendue, sont donc particulièrement adaptées pour modéliser les données séquentielles comme les textes, car elles peuvent capturer le contexte des mots dans une séquence (phrase) plus longue, ce qui est essentiel pour comprendre les sentiments dans les critiques de films.

3.3.1.Exemple de RNN avec couche GRU (Gated Récurrent Unit)

Cet exemple est inspiré du chapitre 16 du livre [3] (voir section Références).

Les principales étapes sont :

Pré-processing : on ajoute une couche de vectorisation des critiques appelée "TextVectorization" qui découpe le texte en mots. Elle utilise les espaces pour identifier les limites des mots, ce qui est à priori suffisant pour des le type de texte de critiques en anglais du jeu de données IMDB dont on dispose.

> *Note : Cette méthode ne fonctionne pas bien dans certaines langues. En effet :*
> *"L'écriture chinoise n'utilise pas d'espaces entre les mots, le vietnamien utilise des espaces même à l'intérieur des mots, et l'allemand attache souvent plusieurs mots ensemble, sans espaces)." [3]*

La taille du vocabulaire, c'est-à-dire le nombre maximum de mots les plus importants qui seront considérés par le modèle, est à adapter. Plus il y a des mots dans le vocabulaire, plus le modèle doit apprendre de paramètres, ce qui rend l'apprentissage plus long.

Embedding : une couche d'embedding convertit les identifiants de mots en "embeddings". La matrice d'embeddings doit avoir une ligne par mot du vocabulaire.

Classification : vient ensuite une couche GRU suivie d'une couche Dense (entièrement connectée) avec un seul neurone et la fonction d'activation sigmoïde. Puisqu'il s'agit d'une tâche de classification binaire, grâce à la fonction sigmoïde, la sortie du modèle sera la probabilité estimée que la critique exprime un sentiment positif à l'égard du film.

3.3.2. Exemple de RNN avec couche Long Short-Term Memory (LSTM)

Les différences de cette version de RNN avec LSTN, par rapport à la version GRU, se trouvent dans les étapes de pré-traitement et la première couche de classification. Voici les étapes:

Pré-traitement : on utilise un tokeniser (à la place de TextVectorization) de la librairie Keras/Tensorflow qui permet de transformer les critiques en séquences de longueur identique traitables par le modèle. Le texte est découpé, et les séquences trop courtes sont complétées avec un padding.

Embedding : comme dans l'exemple précédent une couche d'embedding permet de vectoriser les entrées.

Classification :

La couche LSTM permet de capturer les dépendances temporelles dans les séquences. Ensuite :

✓ Une couche de dropout aide à éviter le sur apprentissage en éteignant aléatoirement des neurones pendant l'entraînement.

✓ La couche finale dense avec une activation sigmoïde, comme dans l'exemple précédent, car elle produit une probabilité d'appartenance à une seule classe (critique positive ou négative).

3.3.3. Optimisation des RNNs

Pour l'optimisation de ce type d'architecture, on cherche à ajuster certains des paramètres. Parmi les plus courants, les suivants seront regardés:

✓ Le nombre d'Epochs (nombre de fois que le modèle passe en revue les données d'entrainement)

✓ Nombre d'unités : Le nombre de neurones dans la couche GRU/LSTM. Des valeurs courantes incluent 32, 64, 128, voire 256 ou plus.

✓ Taux de Dropout pour les LSTM : Le taux de suppression aléatoire des neurones pendant l'entraînement pour éviter le surapprentissage. Des valeurs typiques sont 0.2, 0.3, ou 0.5.

3.4. Transfer Learning - module pré-entraîné

Cet exemple reprend et adapte le code du tutoriel sur le site tensorflow [5] (voir la section Références).

Dans cette approche avec du transfert learning, la vectorisation est mise en ouvre en réutilisant un module pré-entraîné d'embedding des mots comme première couche. Donc, aucun pré-traitement supplémentaire du texte n'est nécessaire.

Les couches empilées pour construire notre classifieur sont les suivantes:

Embedding : C'est la première couche TensorFlow Hub (le modèle pré-entrainé). Le modèle pré-entrainé utilisé dans cet exemple est le google/nnlm-en-dim50/2. Il divise la phrase en tokens, associe un embedding à chaque token, puis combine les embeddings.

Classification :

✓ Une couche entièrement connectée (Dense) avec 16 unités cachées, et reçoit le vecteur de longueur fixe qui sort de la première couche.

✓ La dernière couche (dense), comme pour les exemples précédents, est connectée à un seul nœud de sortie, avec une activation sigmoïde pour la classification binaire (0 pour les critiques négative, 1 pour les critiques positives).

3.5. Méthode d'évaluation des modèles

Nous allons utiliser l'exactitude (accuracy) en entrainement et en test pour évaluer et comparer les différents modèles de classification. Dans notre cas, le jeu de données IMDB utilisé contient deux classes qui sont équilibrées (25000 critiques positives, 25000 critiques négatives). Donc c'est une métrique adaptée pour comparer les différents modèles.

L'Accuracy (exactitude) mesure la proportion de critiques correctement classées.

4. Résultats et conclusions

4.1. Machine Learning : Régression Logistique

Tableau récapitulatif des différents essais :

#	Cas testé	Accuracy Entraînement	Accuracy Test
RL01	Pré-traitement Manuel + TF-IDF	0.91292	0.87748
RL02	Pré-traitement Manuel +TD-IDF + régularisation L2 (C=1.5)	0.87972	0.87668
RL03	TF-IDF sans options	0.91372	0.88252
RL04	TF-IDF sans options + L2 (C=3)	0.88624	0.88232
RL05	Options de pré-traitement intégrées dans TF-IDF	0.91588	0.87832
RL06	Options de pré-traitement intégrées dans TF-IDF + L2 (C=2.5)	0.881	0.8748

Observations :

✓ RL01: Avec un TF-IDF vectorizer par défaut pour convertir la variable d'entrée (le texte des critiques) en vecteur, on obtient des bonnes performances et seulement 3% de différence entre les scores de train et de test, donc un léger surapprentissage. La taille du vocabulaire (max_features) est de 5000 mots.

✓ RL02 et 04: La régularisation L2, conseillée dans le cas des application NLP (Natural Language Processing), permet bien de réduire l'écart entre test et apprentissage. Donc on n'a plus de sur apprentissage, même s'il n'est pas très prononcé dans les différents cas envisagés.

✓ Un pré-traitement manuel (lemmatizing,suppression stop words et ponctuation ne semble pas améliorer l'accuracy. RL01 et 02, comparés à RL03 et 04

✓ L'utilisation des options intégrées dans TF-IDF pour supprimer les stopwords et les majuscules à la place du pré-traitement manuel ne semble

pas avoir non plus une influence significative sur les performances du modèle. RL05 et 06 comparés à RL03 et 04.

La meilleure combinaison pour ce modèle (RL04) est d'utiliser TF-IDF par défaut, sans options stopwords/lower, mais avec une régularisation L2.

En effet avec un C= 3 (trouvé avec GridSearchCV), il n'y a pratiquement plus de différence entre le score en entrainement (0.88624) et le score en test (0.88232). Et c'es la meilleure accuracy trouvée.

Conclusions pour ce modèle de Régression Logistique :

✓ Dans le cadre de ces essais, aucune amélioration vraiment notable n'a été mise en évidence avec des techniques classiques de lemmatization , élimination des stopwords. TF-IDF parait déjà déjà suffisant.

✓ La régularisation L2 corrige le léger sur apprentissage observé.

✓ Le fait qu'il s'agisse d'un jeu de données qui est très grand, qui est modéré par IMDB, et dont les critiques sont très polarisées explique peut-être les résultats obtenus.

4.2. Machine Learning : Forêts Aléatoires

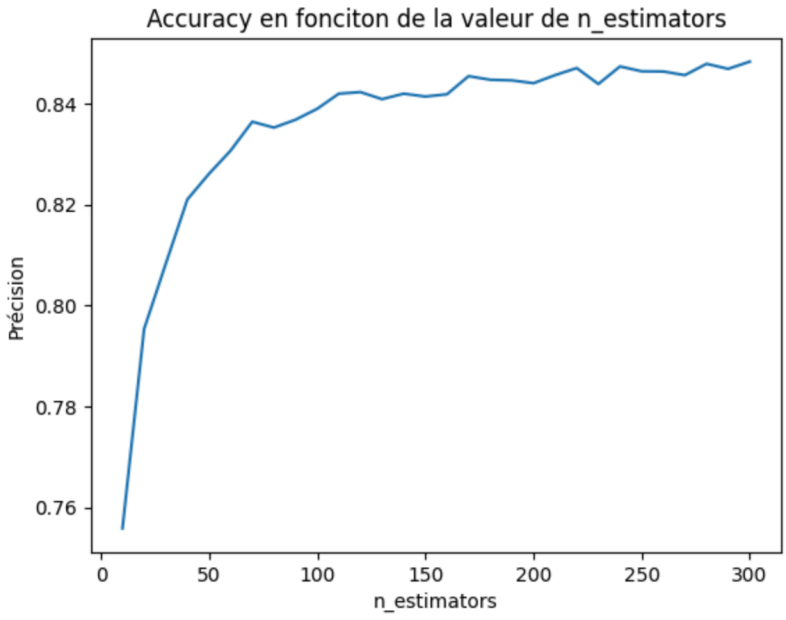
Tableau récapitulatif des différents essais :

#	Cas testé	Accuracy Entraînement	Accuracy Test
FA01	Sans optimisation, n_estimators= 200	1.	0.84556
FA02	GridSearchCV Best estimator (max_depth=9, max_features='log2', max_leaf_nodes=9,n_estimators=150)	0.83908	0.82304
FA03	RandomSearchCV Best Estimator (max_depth=9, max_features='sqrt', max_leaf_nodes=9,n_estimators=100)	0.81952	0.80824

Observations :

✓ FA01: Le classifieur RandomForest avec 200 estimateurs, avant optimisation, atteint une accuracy en test autour de 0.83, mais on est face à du surapprentissage (accuracy en apprentissage = 1).

✓ Une optimisation par validation croisée semble nécessaire. Par cela, la courbe de l'accuracy en fonction du nombre d'estimateurs nous permet de cibler l'intervalle du paramètre n_estimators.



La courbe montre qu'à partir d'environ 100 estimateurs la croissance de l'accuracy (Accuracy) ralenti et augmente très doucement. Elle se stabilise autour de 0.84 au delà de 110 estimators.

Optimisation avec recherche GridSearchCV (FA02) :

La meilleure combinaison de paramètres trouvée est: (max_depth=9, max_features='log2', max_leaf_nodes=9, n_estimators=150). Le temps de calcul est très long pour les 81 combinaisons.

Model grid:		precision	recall	f1-score	support
	0	0.81	0.83	0.82	12084
	1	0.84	0.81	0.83	12916
accuracy				0.82	25000
macro avg		0.82	0.82	0.82	25000
weighted avg		0.82	0.82	0.82	25000

```
Forêt aléatoire grid best train : 0.83908
Forêt aléatoire grid best test : 0.82304
```

L'accuracy est meilleure: 0.823 en test et 0.839 en entrainement. Et, il n'y a pas de surapprentissage.

Optimisation avec RandomizedSearchCV (FA03) :

Avec les résultat d'une recherche RandomizedSearchCV, la meilleure combinaison de paramètres trouvée est : Max depth : 9, Max features : sqrt, Max leaf nodes : 9, N estimators : 100

Le temps de calcul était raisonnable mais l'accuracy du modèle optimisé baisse légèrement en test pour se situer à 0.8195. Il n'y a pratiquement pas de surapprentissage (accuracy en entrainement: 0.8082)

	precision	recall	f1-score	support
0	0.77	0.83	0.80	11666
1	0.84	0.79	0.81	13334
accuracy			0.81	25000
macro avg	0.81	0.81	0.81	25000
weighted avg	0.81	0.81	0.81	25000

```
Forêt aléatoire random best train : 0.81952
Forêt aléatoire random best test : 0.80824
```

Conclusions pour ce modèle de Forêt Aléatoire :

✓ Même avec des classes équilibrées de notre jeu de données (autant de critiques positives que négatives) le modèle est en surapprentissage avant optimisation.

✓ La meilleure combinaison d'hyper paramètres est trouvé à l'aide de GridSearch CV (82% en test) contre 80% en test pour RandomSearchCV, et il n'y a plus de surentraînement.

✓ L'accuracy 82% est un peu moins bonne que le modèle de Régression Logistique précédent (88%), toujours en utilisant TF-IDF sans pré-traitement supplémentaire, et une taille de vocabulaire (max_features) identique dans les deux cas (5000 mots).

4.3. Deep Learning : RNN (GRU)

La taille du vocabulaire est fixée à 5.000 mots, comme pour les exemples précédents.

Tableau récapitulatif des différents essais:

#	Cas testé	Accuracy Entraînement	Accuracy Test
GRU01	Avant optimisation (2 epochs)	accuracy: 0.8579 val_accuracy: 0.8654	0.87188
GRU02	Optimisation avec Keras Tuner (3 epochs) Best Hyperparameters: {'embedding_dim': 64, 'gru_units': 64, 'learning_rate': 0.00021669207674046597}	accuracy: 0.9136 val_accuracy: 0.8708	0.86048
GRU02	Optimisation avec Keras Tuner (2 epochs) Best Hyperparameters: idem	accuracy: 0.8775 val_accuracy: 0.8758	0.87756

Observations:

✓ Avec un faible nombre d'Epochs (2), et sans optimisation particulière, l'accuracy est élevée (0.87188)

✓ L'optimisation avec Keras Tuner prend beaucoup de temps, des intervalles très réduits permettent d'explorer quelques valeurs.

✓ Apres optimisation des paramètres, si on augmente le nombre d'epochs à 3, l'accuracy final en test diminue, probablement du à un léger surapprentissage

✓ Si on reste à 2 epochs, alors l'accuracy s'améliore très légèrement (0.87756). C'est probablement la meilleure combinaison de celles testées.

Conclusions pour ce modèle de RNN GRU : le modèle optimisé présente un accuracy de 0.87756, presqu'aussi bonne que celle du modèle de Régression Logistique (0.88232). Quoique plus long, l'entrainement de ce modèle requiert de moins d'epochs (moins de passage sur le jeu de données d'entrainement).

4.4.Deep Learning : RNN (LSTM)

Tableau récapitulatif des différents essais :

#	Cas testé	Accuracy Entraînement	Accuracy Test
LSTM01	Avant optimisation (10 epochs)	accuracy: 0.9804 val_accuracy: 0.8124	0.7871
LSTM02	Avant optimisation (3 epochs)	accuracy: 0.8328 val_accuracy: 0.8188	0.7952
LSTM03	Après optimisation Keras Tuner (5 Epochs). Meilleurs paramètres : {'embedding_dim': 128, 'lstm_units': 32, 'dense_units': 80, 'dropout': 0.2}	accuracy: 0.9142 val_accuracy: 0.8046	0.78744
LSTM04	Après optimisation Keras Tuner (3 Epochs). Parmètres idem.	accuracy: 0.8816 val_accuracy: 0.8176	0.79471

Observations :

✓ LSTM01: Avant optimisation le modèle est en surapprentissage et l'accuracy en test est de 0.7871, donc pas particulièrement élevée.

✓ LSTM02: L'accuracy de validation durant l'entrainement commence à décroitre à partir de 3 epochs. Après ajustement du nombre d'epochs à 3, on a une légère augmentation de l'accuracy (0.7952) et surtout, pratiquement plus de surapprentissage.

✓ La tentative d'optimisation avec Keras Tuner donne à peu près les mêmes résultats, donc aucune amélioration observée (LSTM03 et 04)

Conclusions pour ce modèle de RNN LSTM :

Les paramètres par défaut semblent déjà adaptés, et seul un suivi de l'accuracy de validation pendant l'entraînement pour ajuster le nombre d'epochs a permit d'améliorer légèrement l'accuracy et réduire le surentraînement.

4.5. Deep Learning : Transfer Learning (Embedding pré-entrainé)

L'apprentissage par transfert consiste à prendre un modèle pré-entraîné sur une tâche initiale et à l'adapter à une autre tâche similaire, en tirant parti des connaissances acquises.

Dans cet exemple, nous utilisons un module TensorFlow Hub comme couche d'embedding des mots. Cette couche est un modèle pré-entraîné qui a déjà appris des représentations des mots sur la base de vastes quantités de données textuelles. Puis, on l'applique à notre tâche d'analyse des sentiments sur les critiques de films.

Tableau récapitulatif:

#	Cas testé	Accuracy Entraînement	Accuracy Test
TFL01	Split 60% train-40% validation (10 Epochs)	accuracy: 0.9908 val_accuracy: 0.8696	0.853
TFL02	Split 80% train-20% validation (10 Epochs)	accuracy: 0.9941 val_accuracy: 0.8720	0.8546
TFL03	Split 80% train-20% validation (4 Epochs)	accuracy: 0.9133 val_accuracy: 0.8622	0.852
TFL04	Split 80% train-20% validation (3 Epochs)	accuracy: 0.8558 val_accuracy: 0.8410	0.8234

Observations:

✓ A noter que dans le tutoriel de base, le partitionnement des données d'entrainement (train/validation) est de 60% - 40% (TFL01). Alors que dans tous les exemples deep learning étudiés auparavant on avait plutôt 80-20%.

✓ En changent les proportions à 80% train - 20% validation, on obtient à peu près les mêmes résultats (TFL01 Vs TFL02).

✓ En diminuant le nombre d'epochs on réduit légèrement les écarts de accuracy entre validation et test. Le meilleur compromis est peut-être de rester autour de 4 epochs (TFL03)

✓ Il n'a pas été possible de mettre en ouvre l'optimisation avec Keras Tuner :(

4.6. Tableau Récapitulatif des résultats

En première position on retrouve le modèle Machine Learning Régression Logistique (88.23%). Suivi de l'architecture Deep Learning RNN-GRU (87.76%).

Les autres sont loin derrière :)

Algo #cas de test	Cas testé	Accuracy Entraînement	Accuracy Test
Régression Logistique RL04	TF-IDF sans options + L2 (C=3)	0.88624	0.88232
Forêt Aléatoire FA02	GridSearchCV Best estimator (max_depth=9, max_features='log2', max_leaf_nodes=9,n_estimators=150)	0.83908	0.82304
RNN-GRU GRU02	Optimisation avec Keras Tuner (2 epochs) Best Hyperparameters: idem	accuracy: 0.8775 val_accuracy: 0.8758	0.87756
RNN-LSTM LSTM02	Sans optimisation (3 epochs)	accuracy: 0.8328 val_accuracy: 0.8188	0.7952
Transfer Learning TFL03	Split 80% train-20% validation (4 Epochs)	accuracy: 0.9133 val_accuracy: 0.8622	0.852

5. Conclusions

Avec un modèle simple et rapide à entrainer – comme la régression Logistique –, on obtient des résultats très bons, avec une accuracy proche de 85%. Même si la recherche d'optimisation des paramètres peut-être longue, cela reste une option très intéressante, accessible, qui nécessite à priori peu d'expérience dans le domaine de l'apprentissage automatique. L'accuracy trouvée est suffisante pour des problèmes non critiques, où les erreurs n'ont pas de conséquences majeures.

Les réseaux de neurones avec des architectures plus élaborées s'avèrent plus difficiles à mettre au point. Dans les modèles étudiés, surtout l'architecture RNN-GRU montre une accuracy aussi élevée que le modèle de Régression Logistique, même sans optimisation, mais avec des temps d'entraînement beaucoup plus longs.

Les valeurs de accuracy obtenues avec tous les modèles étudiés restent raisonnablement proches de beaucoup d'exemples cités dans la littérature (voir l'article [2] datant de 2023 qui récapitule l'ensemble des méthodes et les différentes études effectuées autour de l'analyse des sentiments).

6. Difficultés rencontrées

Les principales difficultés rencontrées sur ce projet:

1. Le Manque de repères pour l'optimisation des hyperparamètres: les recherches systématiques de meilleures combinaison de paramètres à l'aide de la validation croisée (GridSearchCV, RandomizedSearch, Keras tuner) sont très pratiques, encore faut-il pouvoir définir des intervalles à tester qui soient les plus adaptés selon le type de modèle et le jeu de données, de façon à ne pas rater les valeurs les plus appropriées tout en restant dans des temps de calcul raisonnables.
2. Les temps d'exécution difficiles à prévoir: Pour l'optimisation des hyperparamètres à l'aide de GridSearchCV ou RandomizedSearchCV, des temps trop longs limitent le nombre de tentatives d'ajustement des intervalles. Difficile de savoir à l'avance… que ce soit sur Google Colab ou en local avec Jupiter las (Anaconda).
3. Le manque de connaissances et d'expérience dans l'utilisation de Keras et Tensorflow a entrainé un effort supplémentaire d'investigation, et quelques blocages, dont certains dus à un incompatibilité de versions Keras et Tensorflow keras. Exemples:
    a. Modèle vide après empilement des couches dans l'approche RNN
    b. Des difficultés pour trouver le moyen de faire marcher le tutoriel pourtant très simple utilisé dans l'approche Transfer Learning, issu du site tensorflow; le tutoriel ne semble pas à jour avec les dernières versions des librairies utilisées.
    c. Même soucis de compatibilité et de connaissance insuffisante du tuner keras pour l'optimisation. Il n'a pas été possible de chercher à optimiser le dernier exemple avec Tensorflow Hub.

4. Perspectives

Après avoir testé et investigué ces différents exemples, il resterait à approfondir l'étape d'optimisation pour les modèles de la catégorie deep learning. Ainsi qu'éventuellement explorer des architectures basés sur des CNN.

Ensuite, l'axe qui se dégage naturellement afin de poursuivre cette exploration de l'analyse de sentiments est celui des méthodes ensemblistes (Ensemble Learning) où plusieurs modèles sont combinés:

"L'apprentissage d'ensemble est une approche prometteuse de l'analyse des sentiments, qui permet de tirer parti des forces de plusieurs modèles pour obtenir de meilleures performances. Plusieurs études sur l'analyse des sentiments ont été menées en utilisant des modèles d'ensemble avec différents algorithmes d'apprentissage automatique, notamment Naive Bayes, machine à vecteur de support (SVM), forêt aléatoire, régression logistique et XGBoost. Le système de vote majoritaire était la méthode la plus couramment utilisée pour combiner les résultats de plusieurs modèles dans un ensemble. L'accuracy des modèles d'ensemble utilisés dans l'analyse des sentiments varie de 80 % à 98,99 %." [2]

Références

1. **TensorFlow Datasets Documentation**. *IMDB Reviews Dataset*. https://www.tensorflow.org/datasets/catalog/imdb_reviews

2. Tan, K.L.; Lee, C.P.; Lim, K.M. "*A Survey of Sentiment Analysis: Approaches, Datasets, and Future Research*". Appl. Sci. 2023, 13, 4550. https://doi.org/10.3390/app13074550

3. **Aurélien Géron**, "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow_ Concepts, Tools, and Techniques to Build Intelligent Systems*" (2022, O'Reilly Media)

4. *"Deep learning : réseaux neuronaux RNN et CNN quelles différences ?*". https://www.lemagit.fr/conseil/Reseaux-de-neurones-RNN-et-CNN-comment-les-differencier

5. **TensorFlow Core/Tutoriels**. *"Classification de texte avec TensorFlow Hub : critiques de films"*. https://www.tensorflow.org/tutorials/keras/text_classification_with_hub?hl=fr

6. https://stackoverflow.com/questions/78530756/error-only-instances-of-keras-layer-can-be-added-to-a-sequential-model

7. https://github.com/tkeldenich/NLP_Classification_Binaire_DeepLearning

8. https://github.com/GuglielmoCerri/IMDb-Sentiment-Analysis/blob/main/Sentiment_analysis.ipynb

9. https://github.com/JS12540/IMDb-Sentiment-Analysis-with-TensorFlow/blob/main/RNN_Imdb_Sentiment.ipynb

10. https://cedric.cnam.fr/vertigo/cours/ml2/tpForetsAleatoires.html

11. https://www.geeksforgeeks.org/random-forest-hyperparameter-tuning-in-python/

12. https://france.devoteam.com/paroles-dexperts/algorithme-n2-comprendre-comment-fonctionne-un-random-forest-en-5-min/