le cnam Paris

RCP216 – Ingénierie de la fouille et de la visualisation de données massives

Projet : Analyse de sentiment des revues de films

Geoffrey Receveur – Avril 2024

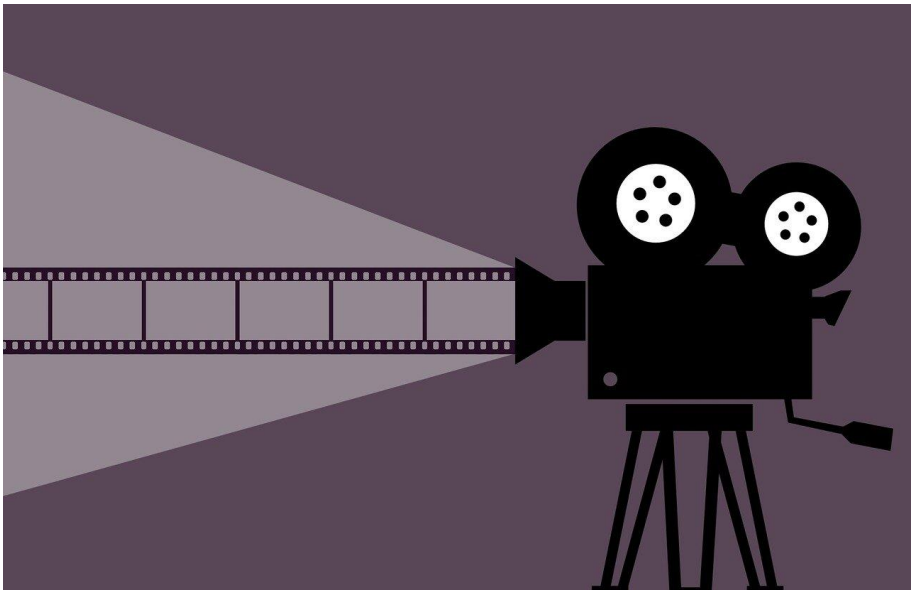


Table des matières

Résumé

L'objectif du projet est de prédire la note entre 1 à 10 des textes de revues de films issues de la base de données IMDB.
La problématique vise donc à affecter les critiques à des notes prédéfinies (1 à 10). Je me suis donc orienté vers la construction d'un prédicteur obtenus par apprentissage supervisé.
La démarche générale pour répondre au projet a été la suivante :

1. Compréhension de la problématique que l'on cherche à résoudre ;
2. Récupération des données textuelles et découverte de celles-ci ;
3. Pré-traitement des données ;
4. Représentations vectorielles des textes ;
5. Création du plusieurs prédicteurs et choix du plus performant ;
6. Conclusion et réflexions sur des pistes d'amélioration.

Présentation des données

Le répertoire avec le jeu de données a été récupéré sur le site Sentiment Analysis (stanford.edu). Ce jeu contient 50 000 critiques de film qui proviennent du site Internet Movie Database (IMDB). Il est découpé en sous-répertoires contenant les données pour l'entrainement et pour le test qui ont été découpées avec une distribution équilibrée entre les critiques positives et négatives.
On peut récapituler le contenu du répertoire avec le tableau suivant :

Nom du Répertoires	Données d'entrainement		Données de test	
Sous-répertoires (label)	Positives	Négatives	Positives	Négatives
Nombre de critique	12500	12500	12500	12500

Une critique est un fichier texte nommés avec cette convention [[id]_[rating]. txt] dont :

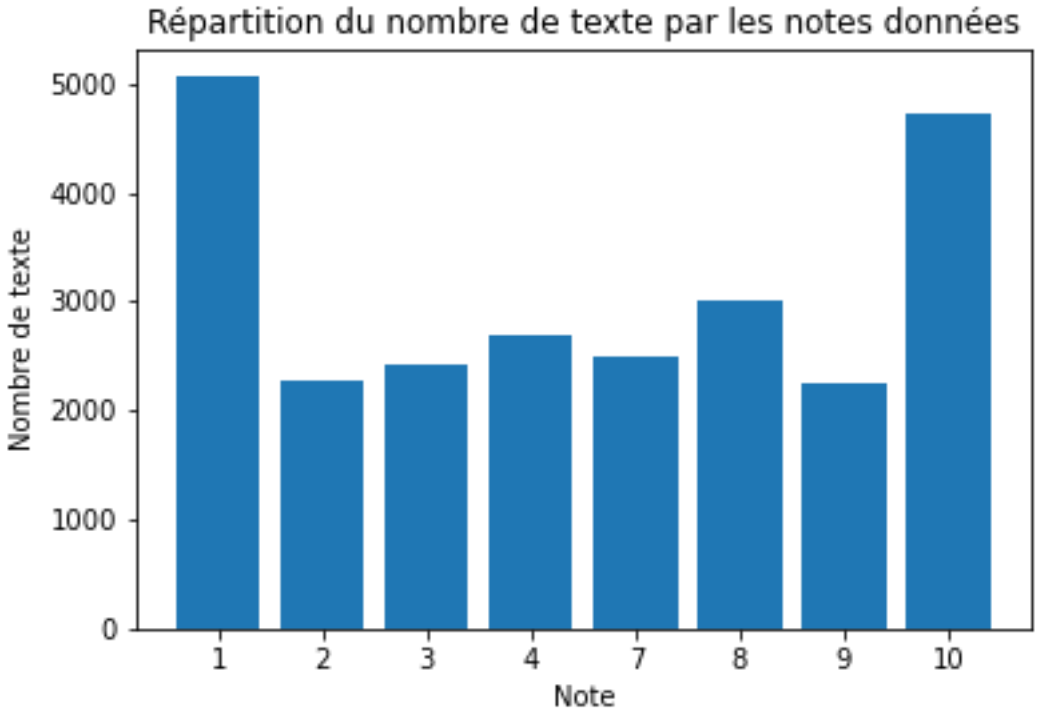
- id est l'identifiant unique de la critique ;
- rating la note attribuée sur une échelle de 1 à 10.

Une critique négative a une note <= 4 tandis qu'une critique positive a une note >= 7.
Les critiques neutres n'ont pas été incluses dans le jeu de données, ce qui veut dire qu'il n'y pas de critique avec une note de 5 ou de 6.

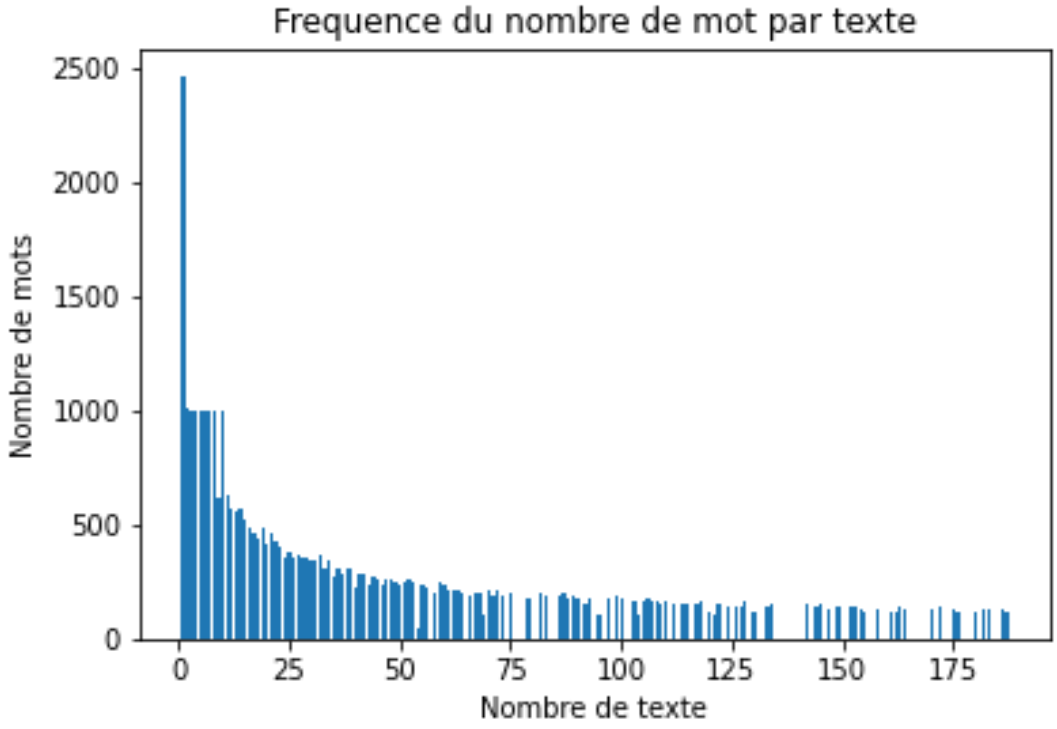
Descriptions élémentaires

La première approche est de comprendre les données afin d'orienter les méthodes à utiliser (pré-traitement, méthodes de vectorisation, algorithme d'apprentissage à utiliser).
Les données ci-dessous sont des informations élémentaires sur les données d'entrainement du répertoire afin d'orienter les prétraitements et traitements futurs, plusieurs visuels ont été générés afin de mieux comprendre la distribution de celles-ci.

Un diagramme en barres de la fréquence des notes permet de voir qu'il y a une hétérogénéité et une surreprésentation de certaines notes (notamment la note 1 et la note 10).

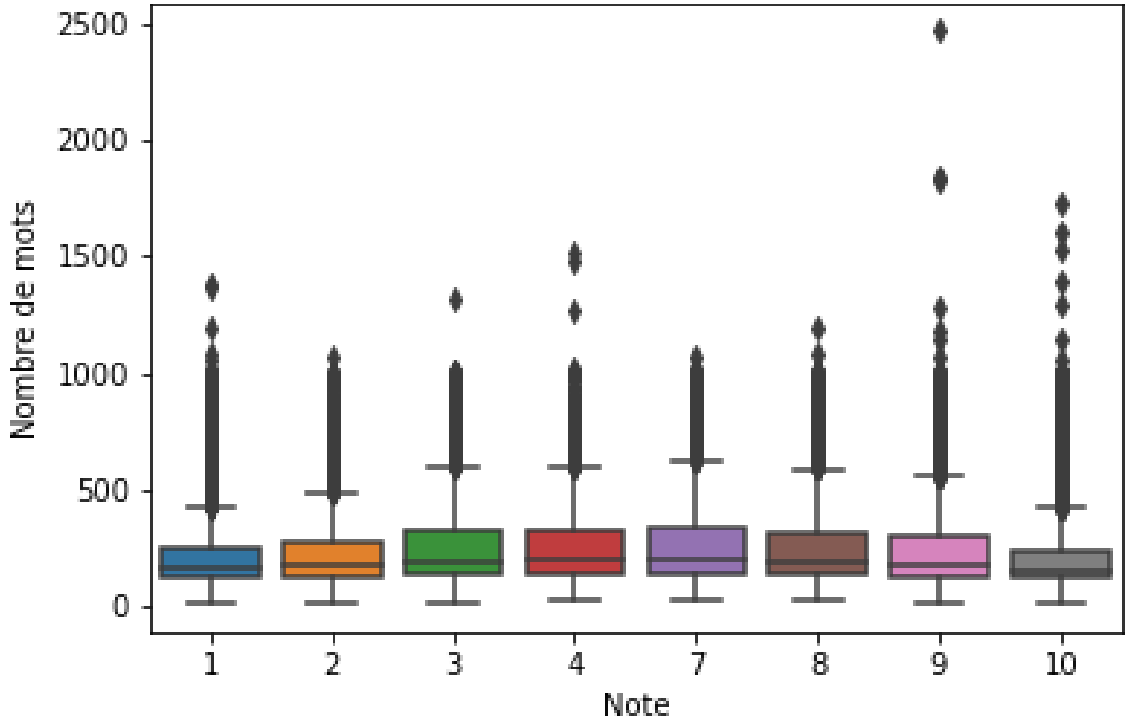


L'histogramme ci-dessous permet de voir le nombre de mots et le nombre de texte associé à ce nombre, on remarque que la majorité des textes ont bien moins de 500 mots, en revanche, il existe des textes moins nombreux mais avec bien plus de mots (environ 2500 mots pour la valeur la plus élevée)



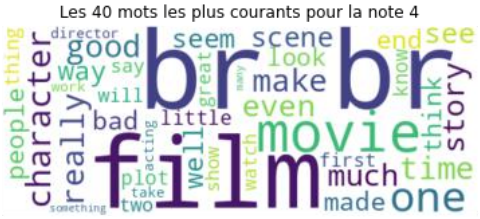
Les diagrammes en boite ci-dessous permettent de voir certains indicateurs de tendances centrale et de dispersion du nombre de mots en fonction de la note donnée.

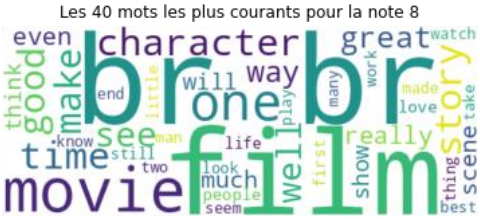
On remarque que la médiane est généralement à l'alentour de 200 mots et que la majorité des textes ont au maximum aux alentours de 500-600 mots et confirme l'histogramme précédent. Ces diagrammes écartent une première supposition de ma part au début de la fouille qui était qu'il y avait peut-être un lien entre le nombre de mots d'une critique et la note qui lui est associée, or, ce n'est pas réellement significatif ici.



Pour finir, on calcul la fréquence des 40 mots les plus courants pour chaque note. Plus un mot est fréquent et plus sa taille sera grande dans les nuages de mots ci-dessous.
A noter que la plupart des mots vides ont été supprimés afin de faire ressortir au mieux ceux avec un réel sens sémantique.

On peut apercevoir dans les nuages qui suivent que certains mots qui auraient pu être une séparation entre une critique avec une note faible et une critique avec une note forte ne le sont pas directement. Par exemple, le mot 'good' se retrouve aussi bien dans les notes mauvaises (1, 2, 3, 4) que les notes élevées (7,8,9 et 10). Ces mots sont peut-être utilisés dans un contexte spécifique. Ce qui pousse à essayer d'obtenir le contexte dans lequel les mots ont été utilisés et pas uniquement de traiter ceux-ci de manière strictement individuelle.





Représentation vectorielle des textes

À la suite des observations effectuées sur le jeu de données, deux méthodes ont été appliquées pour vectoriser les textes :

- L'utilisation d'un modèle de langage qui permet comprendre le contexte d'utilisation des mots tel que vu dans certains TP du cours.
  Les textes étant plus au moins long (de 200 à 2500 mots en fonction des textes), BERT sera utilisé.
- La vectorisation du texte avec pondération TF-IDF des mots qui permet de donner des poids aux mots.

2 pipelines ont donc été configurées, cependant, préalablement à la construction de ces pipelines, certains pré-traitements ont été réalisés comme la suppression des doublons.

Le pipeline numéro 1 basé sur la pondération TF-IDF effectue :

- Un découpage en jeton ;
- La normalisation des mots ;
- La suppression des mots vides (stopwords) ;
- La lemmatisation des mots ;
- Le calcul TF-IDF de chacun des mots.

```
#Transformation qui convertit les textes au format chaînes de caractères dans le format attendu par SparkNLP :
documentAssembler = DocumentAssembler() \
                    .setInputCol("text") \
                    .setOutputCol("document")

sentence = SentenceDetector() \
                .setInputCols(["document"]) \
                .setOutputCol("sentence")

#Création des jetons :
tokenizer = Tokenizer() \
                .setInputCols(["sentence"]) \
                .setOutputCol("token")

#Normalisation des données:
normalizer= Normalizer()\
    .setInputCols(["token"])\
    .setOutputCol("normalized")\
    .setLowercase(True)\
    .setCleanupPatterns(["[^\w\d\s]"])


#Suppression des stopwords:
stopword_cleaner = StopWordsCleaner().setInputCols(["normalized"]).setOutputCol("cleaned_tokens").setCaseSensitive(True)
#setsenitive à false
#Lemmatisation
lemmatizer = Lemmatizer().setInputCols(["cleaned_tokens"]).setOutputCol("lemmas").setDictionary("AntBNC_lemmas_ver_001.txt",value_delimiter="\t", key_delimiter="->")

finisher = Finisher().setInputCols(["lemmas"]).setOutputCols(["token_features"]).setOutputAsArray(True)

#Calcul de TF-IDF
hashTF = HashingTF(inputCol="token_features", outputCol="raw_features", numFeatures=5000)
idf= IDF(inputCol="raw_features", outputCol="features", minDocFreq=100)
```

Configuration du pipeline 1 qui vectorise le texte avec pondération TF-IDF

Le pipeline numéro 2 basé sur le modèle de langage BERT effectue :

- Un découpage en jeton ;
- L'encodages des textes à partir d'encodages BERT. Différents modèles pré-entrainé BERT ont été testés. Si les modèles larges de BERT obtiennent des résultats légèrement plus intéressants, ils ont des inconvénients non négligeables (taille des modèles très élevé et temps de calcul qui augmente très rapidement en fonction des modèles utilisés)
  Le choix a donc été fait de rester sur des modèles plus simples (BERT L8 768), un peu moins performant mais plus rapide à appliquer au regard des ressources machines disponibles.

```
#Transformation qui convertit les textes au format chaînes de caractères dans le format attendu par SparkNLP
documentAssembler = DocumentAssembler() \
                    .setCleanupMode("inplace") \
                    .setInputCol("text") \
                    .setOutputCol("document")
#Création des jetons :
tokenizer = Tokenizer() \
                .setInputCols(["document"]) \
                .setOutputCol("token")

embeddings = BertEmbeddings.pretrained("small_bert_L8_768") \
                           .setInputCols("token", "document") \
                           .setOutputCol("bert_embeddings")

embeddingsSentence = SentenceEmbeddings() \
                           .setInputCols(["document", "bert_embeddings"]) \
                           .setOutputCol("sentence_embeddings_bert") \
                           .setPoolingStrategy("AVERAGE")

embeddingsFinisher = EmbeddingsFinisher() \
                           .setInputCols("sentence_embeddings_bert") \
                           .setOutputCols("finished_embeddings_bert") \
                           .setOutputAsVector(True) \
                           .setCleanAnnotations(False)
```

Configuration du pipeline 2 d'encodages de textes avec BERT

Après la vectorisation des textes par le pipeline numéro 1 ou le pipeline 2, d'autres opérations ont été effectuée avant de créer les prédicteurs :

- Un échantillonnage stratifié des données en fonction de la variable cible qui permet d'obtenir une homogénéité des notes avant l'entrainement.

- Un test de réduction des dimensions via l'analyse en composante principale a également été réalisé afin de diminuer la taille des vecteurs créées après la vectorisation des textes, cependant, l'essai réalisé n'a pas été concluant et a été écarté par manque de temps. Nous reviendrons sur ce point dans la conclusion.

Apprentissage supervisé

Pour l'apprentissage supervisé, plusieurs algorithmes de classification ont été testés, à noter que des algorithmes de régression auraient pu également être utilisés mais j'ai préféré m'orienter sur de la classification plutôt que de la régression dans le cadre du projet.

Avant de discrétiser la variable note, j'ai appliqué les méthodes de classification pour voir les performances obtenues

3 méthodes d'apprentissage supervisé ont été testées :

- Le perceptron multicouche,
- Les forêts aléatoires,
- La régression logistique multinomiale.

Ces méthodes ont été testées avec différentes grilles multidimensionnelle telle que ci-dessous afin de garder la meilleure combinaison de paramètre :

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

lr= LogisticRegression(labelCol='label',featuresCol='features',family='multinomial')
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1,0.05,0.02,0.005,0.001]) \
                              .addGrid(lr.elasticNetParam, [0.1,0.2,0.5,0.7]) \
                              .addGrid(lr.maxIter, [5]) \
                              .build()

cvSvclr = CrossValidator().setEstimator(lr) \
                          .setEstimatorParamMaps(paramGrid) \
                          .setNumFolds(5) \
                          .setEvaluator(evaluator)
```

Exemple d'une grille de paramètres et de validation croisée de la régression logistique multinomiale

Les grilles disponibles dans le fichier source (Projet RCP216 versions du 8 avril.ipynb) ont été testées plusieurs fois avec de nombreuses combinaisons de valeurs.
Afin de diminuer les temps de calcul, le fichier actuel ne contient que les grilles avec les valeurs qui étaient les plus intéressantes au 8 avril 2024 .

Résultats des prédicteurs

Prédicteurs à partir de la vectorisation du texte et pondération TF-IDF

Dans le cadre de la pondération TF-IDF, 5000 mots ont été gardés.
À la suite de l'échantillonnage stratifié, environ 2300 textes par note sont utilisés pour l'entrainement, soit environ 18000 textes.
L'entrainement s'effectue sur 70% des données et le test est effectué sur les 30% restants.

Résultats pour la prédiction des notes de 1 à 10

Méthodes utilisées	Meilleurs paramètres	Précision
Régression logistique multinomiale	maxIter: 5 regParam: 0.005 elasticNetParam: 0.5	~35%
Forêts aléatoires	maxDepth : 10 numTrees : 160	~31%
Perceptron multicouche	Layers : 5000 – 512- 32 – 8 blockSize : 256 maxIter: 100	~ 30%

La précision ne donnant seulement qu'un aperçu global des performances des modèles créés, les résultats ont également été visualisés plus finement à l'aide de matrices de confusion.

prediction_label	1	10	2	3	4	7	8	9
1.0	416	34	216	123	92	24	19	17
10.0	41	389	41	59	60	125	186	229
2.0	124	17	199	128	91	25	24	15
3.0	53	11	100	121	104	16	17	15
4.0	39	18	77	131	185	65	32	21
7.0	12	64	35	63	92	217	159	110
8.0	11	55	16	25	43	122	168	99
9.0	6	91	19	23	40	73	111	168

Matrice de confusion de la régression logistique multinomial sur jeu de test

La matrice de confusion ci-dessus permet de voir les erreurs effectuées par notre prédicteur, on remarque que la précision est un peu plus élevée sur certaines notes comme la note 1 (59%) ou la note 10 (57%), en revanche, dans la plupart des cas, le prédicteur confond souvent des notes proches. Par exemple, la note 2 est souvent confondue avec la note 1, la note 10 avec la note 9, etc… On peut en déduire que la sémantique des textes qui ont des notes proches sont très similaires et le prédicteur a du mal à les distinguer.

Prédicteurs à partir d'encodages BERT

Comme la vectorisation avec pondération TF-IDF, environ 2300 textes par note sont utilisés pour l'entrainement, soit environ 18000 textes.
Pour rappel, le modèle simple BERT L8 768 a été utilisé ici principalement pour des raisons de ressources.

Résultats pour la prédiction des notes de 1 à 10 :

Méthodes utilisées	Paramètres	Précision
Régression logistique multinomiale	maxIter: 5 regParam: 0.05 elasticNetParam: 0.1	~30%
Perceptron multicouche	Layers : 768 – 256 – 8 blockSize : 256 maxIter: 100	~ 28%
Forêts aléatoires	maxDepth : 10 numTrees : 120	~27%

prediction_label	1	10	2	3	4	7	8	9
1.0	489	60	267	192	149	45	48	45
10.0	32	360	43	37	42	128	171	251
2.0	89	24	138	125	102	47	28	23
3.0	19	7	43	39	41	12	9	8
4.0	60	26	116	156	184	96	69	33
7.0	12	58	38	72	126	192	186	121
8.0	6	64	23	27	48	106	123	112
9.0	12	68	9	20	17	50	83	95

Matrice de confusion de la régression logistique multinomiale avec BERT sur le jeu de test

Nous avons parfois une meilleure précision comme sur la note 1 (~69%), en revanche, si certaines notes proches sont toujours confondues, d'autres notes ont beaucoup de mal à être identifiées par notre prédicteur comme la note 3 et la note 9.

Tant pour l'encodages de textes avec BERT que la vectorisation avec pondération TF-IDF, il existe une certaine confusion entre les notes proches. Afin d'augmenter les performances des prédicteurs, une discrétisation a été opérée.

Discrétisation de la variable note et nouveaux résultats

La variable note a été discrétisée de deux façons :

En quatre catégories :

- Catégorie 1 : (1,2),
- Catégorie 2 : (3,4),
- Catégorie 3 : (7,8),
- Catégorie 4 : (9,10).

Puis en deux catégories :

- Catégorie 0 : (1,2,3,4)
- Catégorie 1 : (7,8,9 et 10)

Pour l'entrainement, seulement la régression logistique multinomiale a été réutilisée après la discrétisation car cette méthode a obtenu les meilleures performances jusqu'à présent.

Résultats sur les quatre catégories :

Méthodes utilisées	Paramètres	Précision	Remarques
Régression logistique multinomiale avec vectorisation et pondération TF-IDF	maxIter: 5 regParam: 0.05 elasticNetParam: 0.5	~59%	La précision atteint plus de 70% sur certaines catégories
Régression logistique multinomiale avec BERT	maxIter: 5 regParam: 0.05 elasticNetParam: 0.5	~54%	Globalement le prédicteur reste moins efficace

```
+----------------------------+----+---+---+----+ +----------------------------+----+---+---+---+
|prediction_label_discrete_1| 0.0|1.0|2.0| 3.0| |prediction_label_discrete_1| 0.0|1.0|2.0|3.0|
+----------------------------+----+---+---+----+ +----------------------------+----+---+---+---+
|                         0.0|1110|480| 99|  99| |                         0.0|1054|495|153|150|
|                         1.0| 283|692|172|  47| |                         1.0| 305|653|288|107|
|                         2.0|  45|181|698| 273| |                         2.0|  79|251|647|329|
|                         3.0| 107|155|536|1083| |                         3.0|  81|125|455|888|
+----------------------------+----+---+---+----+ +----------------------------+----+---+---+---+
```

A gauche la régression logistique avec TF-IDF et à droite avec BERT

Résultats sur les deux catégories :

Méthodes utilisées	Paramètres	Précision	Remarques
Régression logistique multinomiale avec vectorisation et pondération TF-IDF	maxIter: 5 regParam: 0.02 elasticNetParam: 0.1	~81%	
Régression logistique multinomiale avec vectorisation BERT	maxIter: 5 regParam: 0.05 elasticNetParam: 0.5	~81%	Le prédicteur avec BERT commence à devenir plus performant qu'avec la pondération TF-IDF

prediction_label_discrete_2	0.0	1.0
0.0	3028	706
1.0	752	3037

prediction_label_discrete_2	0.0	1.0
0.0	3017	742
1.0	700	3064

A gauche la régression logistique multinomiale avec TF-IDF et à droite avec BERT

En conclusion

Plusieurs représentations vectorielles de textes ont été utilisées (avec pondération TF-IDF, BERT) avant d'appliquer plusieurs méthodes de classification supervisée (régression logistique multinomial, forêts aléatoires, perceptron multicouche).

En résumé :

- Les prédicteurs issus de la régression logistique multinomial sont plus performants et cela peu importe le type de représentation vectorielle utilisé.
- Le prédicteur créé à partir de la vectorisation avec pondération TF-IDF obtient des résultats intéressants, cependant, les données nécessitent d'être discrétisées avec en contrepartie une perte d'information. En effet, si la distinction des notes négatives (1, 2 ,3, 4) des notes positives (7, 8, 9, 10) obtient une précision supérieure à 80% sur les derniers essais réalisés, il est en revanche bien plus difficile d'obtenir une précision correcte pour chacune de ces notes.
- Le prédicteur créé à partir de la vectorisation via le modèle de langage BERT obtient des résultats légèrement en deca mais ceux-ci sont à relativiser car au cours du projet c'est un modèle de langage simple qui a été utilisé (BERT L8 768).

Plusieurs axes d'améliorations restent possibles et sont à tester pour tenter d'améliorer les performances :

- Une réduction de la dimension des données avec une analyse en composante principale avant d'entrainer les prédicteurs.
- L'augmentation du jeu de données. Dans le cadre du projet, seulement 25 000 textes ont été utilisés jusqu'à présent alors que 50 000 textes sont disponibles.
- Enfin, pour l'encodage de textes avec Bert, il serait intéressant d'utiliser des modèles pré-entrainés plus complexes (les modèles larges).