NLP Project : Movie Sentiment Analysis Review

Antoine Rougier
ENSAE Paris
antoine.rougier@ensae.fr

Abstract

Dans ce projet, nous nous intéressons à l'analyse des sentiments des critiques de films. L'objectif de ce projet est de comprendre les travaux qui ont été fait sur le sujet et d'implémenter des modèles pour prédire si un film est bon ou mauvais. Pour cela, nous utiliserons comme base l'article suivant : `https://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf`. Vous pouvez retrouver le projet python ici `https://github.com/antoinerougier/NLP_Project`.

1 Introduction

Les techniques d'apprentissage non-supervisé permettent d'avoir une bonne approche pour représenter la sémantique d'un mot, mais elle a du mal à capturer la signification précise de celui-ci. Ainsi, pour résoudre ce problème, nous pouvons penser à une nouvelle approche qui combinerait une approche supervisée et non-supervisée. C'est ce que nous allons voir dans la suite de ce rapport. Notre objectif est de comprendre les travaux qui ont déjà été effectués sur le sujet, puis de l'appliqué sur un jeu de données (revue de film). Nous ferons une analyse des données, puis expliquerons notre méthode et les résultats que nous avons obtenu. L'objectif de ce rapport est d'examiner si la taille des embeddings a un impact significatif sur la précision des prédictions de sentiment concernant des films. Plus précisément, nous cherchons à déterminer si une taille d'embedding plus grande améliore la qualité des prédictions.

2 State-of-the-art

Pour commencer, il faut rappeler ce qui a déjà été proposé comme analyse dans ce domaine. Ainsi, nous nous appuyons sur l'article suivant : *Learning Word Vectors for Sentiment Analysis* de Andrew L. et al.

Cet article se base sur des travaux précédents et notamment du Latent Dirichlet Allocation (LDA) de Blei et al., qui est un modèle probabilitisque qui donne la probabilité conditionnel d'un mot w d'être dans un sujet T. Il évoque dans un second temps l'approche Latent Semantic Analysis (LSA) qui apprend la sémantique d'un mot grâce à sa représentation vectorielle en effectuant une décompostion SVD sur une matrice de réprésentation. Nous développerons cette méthode dans la suite de ce rapport. Nous revenons maintenant à la méthode de l'article. Ainsi, leur modèle fonctionne en deux temps, d'abord il capture la similarité sémantique des mots. Il s'agit d'un apprentisasage non-supervisé qui se base sur le modèle LDA. Puis dans un second temps, le modèle doit capturer le sentiment d'un mot. Cette fois-ci, il s'agit d'un apprentissage supervisé car il s'agit de retrouver la catégorie d'un texte (comme par exemple si le contenu est positif ou négatif). Ensuite, pour tester leur modèle, il apprenne sur les revues de 25000 films à prédire si celle-ci est bonne ou mauvaise. Il utilise un vector de taille 50. Puis ils utilisent un modèle linéaire SVM avec une cross-validation pour évaluer leur modèle. Les résultats de leurs modèles sont bien supérieurs aux anciennes approches.
Aujourd'hui, les modèles de langage de grande taille (LLM) sont capables d'effectuer des tâches de classification de sentiments avec de meilleurs résultats. Grâce à des architectures de réseaux de neurones sophistiquées, ces modèles peuvent analyser le contexte et la sémantique des textes pour

déterminer si une revue de film est positive ou négative. Cependant, bien que ces capacités soient impressionnantes, elles ne seront pas approfondies dans ce rapport. Nous passons maintenant à l'analyse de nos données.

3 Data Analyse

Maintenant, nous allons passé à l'analyse de données. Vous pouvez les retrouvés ici `https://ai.stanford.edu/~amaas/data/sentiment/`. Il s'agit de fichiers textes qui sont soit rangé dans un dossier positif, négatif ou neutre. Comme dans l'article, nous nous intéresserons seulement aux critiques positives et négatives. Il y a 25000 textes pour les données d'entraînement et de même pour les données de test. Nous gardons cette répartition pour pouvoir comparer notre approche avec celle de l'article. De plus, il y a autant de revues positives que négatives dans le jeu d'entraînements. Pour ne pas fausser notre analyse, nous effectuons nos statistiques descriptives uniquement sur le jeu d'entraînement. Les critiques des films sont en anglais. Nous avons utilisé la librairie langdetect pour vérifier cela (il détecte néanmoins une fois du néérlandais, mais il s'agit d'une erreur car cela n'est que le nom d'un personnage "Paul Verhoeven"). Dans un premier temps nous affichons les mots qui reviennent le plus souvent pour les critiques positives vs négatives :

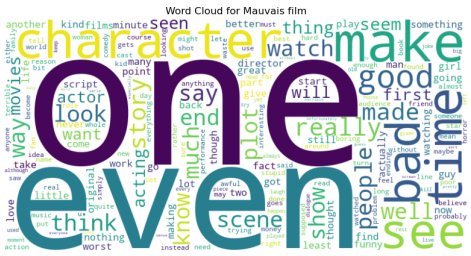

Figure 1: Mauvais Film

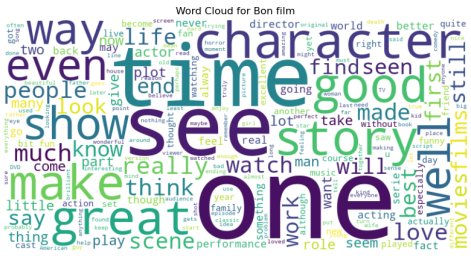

Figure 2: Bon film

On remarque alors que certains termes caractérises bien les bons et mauvais film comme par exemple : bad time, good, well, great...

Statistique	Bonnes revues	Mauvaises revues
Nombre d'échantillons (count)	12,500	12,500
Moyenne (mean)	236.71	230.87
Écart-type (std)	180.49	166.66
Valeur minimale (min)	12	10
25% (25%)	125	128
Médiane (50%)	174	174
75% (75%)	291	278
Valeur maximale (max)	2470	1522

Table 1: Comparaison des statistiques sur le nombre de mots des bonnes et mauvaises revues

Dans un second temps on s'intéresse aussi à la longueur des critiques (le nombre de mots), pour cela nous représentons la distribution du nombre de mots entre les bonnes et les mauvaises revues.

Nous pouvons observer que les mauvaises revues de films ont généralement moins de mots que les bonnes.

4 Nos modèles

Notre objectif est de comparer différents modèles pour trouver la meilleure approche pour prédire le sentiment d'une critique d'un film. Pour cela, nous allons comparer trois modèles :

- **Multinomial Naïve Bayes** (MultinomialNB), souvent utilisé pour la classification de texte.
- **Support Vector Machines** (SVM), qui maximise la séparation entre classes.

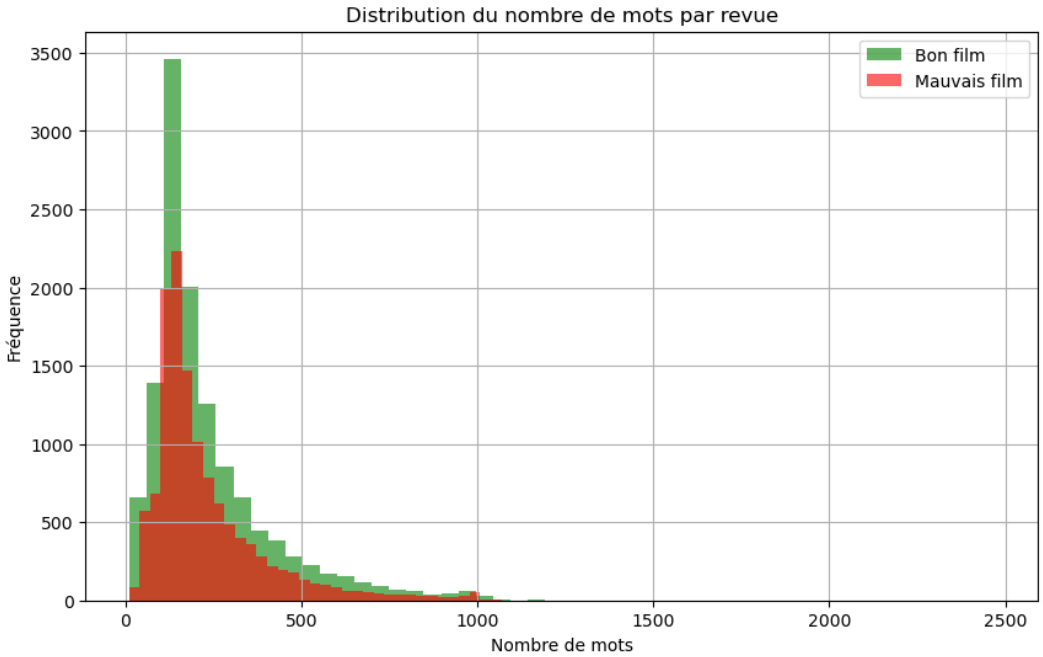

Figure 3: Distribution du nombre de mots

- **Régression Logistique**, un modèle probabiliste largement utilisé pour la classification binaire et multiclasse.

Ces trois modèles ont en commun de faire de la classification ce qui est notre cas ici. Nous voulons prédire la variable Y.

$$Y = \begin{cases} 1 & \text{si c'est un bon film} \\ 0 & \text{sinon, c'est un mauvais film} \end{cases}$$

Avant de passer à la partie de modélisation, nous faisons un rappel sur la création d'embedding de nos données. En effet, nous ne pouvons pas directement utiliser les données textes comme input dans les différents modèles, nous devons d'abord les transformer en vecteurs.

4.1 Embedding

Pour créer l'embedding de notre corpus nous utilisons la librairie de scikit-learn TfidfVectorizer. Comme notre jeu de données est en anglais nous utilisons le stop-word english et nous mettons max-feature à 5000.

La matrice résultante X est une matrice TF-IDF où chaque élément x_{ij} est calculé comme suit :

$$x_{ij} = \text{TF}(i,j) \times \text{IDF}(j)$$

où nous avons :
Fréquence du terme j dans le document i.

$$\text{TF}(i,j) = \frac{\text{nombre d'occurrences du mot } j \text{ dans le document } i}{\text{nombre total de mots dans le document } i}$$

Importance inverse du terme j dans le corpus.

$$\text{IDF}(j) = \log\left(\frac{N}{\text{nombre de documents contenant le mot } j}\right)$$

où N est le nombre total de documents.

A la fin, nous avons donc pour chacun des texts d'entraînement, un embedding de taille 5000 associé.

4.2 Latent Semantic Analysis (LSA)

Pour bien caractériser notre corpus de textes nous effectuons un embedding de 5000. Mais cette taille est bien trop grande pour que un modèle comme SVM puisse apprendre correctement. Ce que nous faisons alors, c'est une réduction de dimensions comme dans l'article en utilisant la technique LSA. Cette méthode permet d'établir des relations entre un ensemble de documents et les termes qu'ils contiennent, en construisant des « concepts » liés aux documents et aux termes. Nous détaillons ici la méthide pour réduire la dimension des embeddings.

On construit une matrice M de taille $m \times n$, où :

- m est le nombre de mots dans le vocabulaire,
- n est le nombre de documents,
- Chaque entrée M_{ij} représente le poids du mot i dans le document j (ex: fréquence TF, pondération TF-IDF).

On applique la décomposition SVD à la matrice M :

$$M = U\Sigma V^T \tag{1}$$

- $U \in \mathbb{R}^{m\times m}$: matrice des vecteurs propres des termes,
- $\Sigma \in \mathbb{R}^{m\times n}$: matrice diagonale des valeurs singulières,
- $V \in \mathbb{R}^{n\times n}$: matrice des vecteurs propres des documents.

On ne conserve que les k plus grandes valeurs singulières et leurs vecteurs associés :

$$M_k = U_k\Sigma_k V_k^T \tag{2}$$

- $U_k \in \mathbb{R}^{m\times k}$: représente les mots dans l'espace réduit,
- $\Sigma_k \in \mathbb{R}^{k\times k}$: contient les k plus grandes valeurs singulières,
- $V_k \in \mathbb{R}^{n\times k}$: représente les documents dans l'espace réduit.

- Deux mots ayant un sens proche auront des vecteurs similaires dans U_k.
- Deux documents traitant de sujets similaires auront des vecteurs proches dans V_k.
- LSA capture ainsi les relations sémantiques latentes entre mots et documents.

Dans notre cas, nous allons utiliser deux LSA différents, un premier avec k = 50, puis un second avec k = 100. L'objectif est de comparer les modèles sur ces deux embeddings pour voir si la taille de la réduction impact significativement la prédiction du modèle.

4.3 Le choix des modèles

Le modèle **Naïve Bayes** repose sur le théorème de Bayes et l'hypothèse d'indépendance conditionnelle des caractéristiques.

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)} \tag{3}$$

où :

- $P(C_k|X)$ est la probabilité que l'exemple X appartienne à la classe C_k.
- $P(X|C_k)$ est la probabilité d'observer X sachant la classe C_k.
- $P(C_k)$ est la probabilité a priori de la classe C_k.

- $P(X)$ est la probabilité totale d'observer X.

Dans le cas d'un **Naïve Bayes multinomial**, utilisé pour le traitement du texte, on suppose que les caractéristiques suivent une distribution multinomiale :

$$P(X|C_k) = \prod_{i=1}^{n} P(x_i|C_k)^{x_i} \tag{4}$$

où x_i est la fréquence du mot i dans le document X.

Le SVM cherche à maximiser la marge entre les classes en trouvant un hyperplan optimal qui sépare les données.

$\max_{w,b} \frac{1}{||w||}$ sous contrainte $y_i(w \cdot x_i + b) \geq 1, \forall i$(5)

où :

- w est le vecteur des poids,
- b est le biais,
- y_i est la classe de l'exemple x_i.

Si les données ne sont pas séparables linéairement, on utilise des fonctions noyaux $K(x_i, x_j)$ pour projeter les données dans un espace de plus haute dimension.

La régression logistique est un modèle de classification probabiliste qui estime la probabilité qu'un exemple appartienne à une classe donnée :

$$P(y = 1|X) = \sigma(w \cdot X + b) \tag{6}$$

où $\sigma(x)$ est la fonction sigmoïde :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

L'entraînement se fait en maximisant la vraisemblance via la descente de gradient :

$$w := w - \alpha \nabla L(w) \tag{8}$$

où α est le taux d'apprentissage et $L(w)$ est la fonction de perte logistique.

4.4 Entraînements des différents modèles

Pour entraîner nos modèles, nous effectuons des CVGridsearch pour trouver les meilleurs hyperparamètres de chaque modèle. Pour chaque entraînement, nous effectuons une cross-validation de taille 5.

Pour le modèle Naïve Bayes, nous gardons l'embedding initial de taille 5000 et nous testons différentes valeurs de alpha.

Pour les modèles SVM et de régression logistique, nous allons effectuons l'entraînement sur des embeddings de taille 50 et 100 après une réduction avec la technique LSA sur les embeddings de taille 5000. L'objectif est de voir si nous parvenons à trouver plus d'informations dans les embeddings de taille 100 que dans ceux de taille 50. Comme notre jeu de données est équilibrée, nous utilisons comme score l'accuracy. De plus, il s'agit de la métrique utilisée dans l'article ce qui nous permettra de comparer objectivement nos résultats avec ceux de l'article.

On s'intéressera aussi à d'autres métriques comme par exemple le score gini, ou encore les courbes AUC-ROC.

$$\text{Gini} = 2 \times \text{AUC} - 1$$

où :

- AUC : Aire sous la courbe ROC (Receiver Operating Characteristic).
- Gini : Indicateur de la qualité des modèles de classification.

Ainsi, un modèle parfait aura un AUC de 1 et un score de Gini de 1, tandis qu'un modèle aléatoire aura un AUC de 0.5 et un score de Gini de 0.

5 Résultat

Une fois que nous avons entraîner nos différents modèles, nous pouvons les valider ou non sur le jeu de test. Nous affichons les résultats des scores gini dans le tableau suivant :

Modèle	**Score Gini**
Naive Bayes	0.85
SVM (50)	0.84
Logistic Regression (50)	0.84
SVM (100)	0.87
Logistic Regression (100)	0.87

Table 2: Scores Gini pour différents modèles

Ainsi, nous voyons ici que les meilleurs modèles sont le modèle SVM et celui de régression logistique avec une réduction de l'embedding à 100. Ainsi, nous pouvons dire que cette taille supérieur permet de mieux caractériser l'embedding de départ. Nous pouvons relever que même les réductions à taille 50 permettent de bien caractériser les embedding initiaux, en effet ces deux modèles ont un score très proche du Naïve Bayes. Nous pouvons aussi soulever le point que les embedding de taille 100 n'apporte pas une bien plus grande connaissance que ceux de taille 50. En effet, la différence entre les modèles n'est que de 0.03, et aucun des modèles n'a atteint un score supérieur à 0.90.

Nous affichons ici les résultats pour des courbes AUC-ROC. Nous voyons ici que les modèles ont quasiment tous les mêmes résultats. Ainsi, nous pouvons conclure qu'un embedding de taille 50 suffit pour bien caractériser notre corpus de textes.

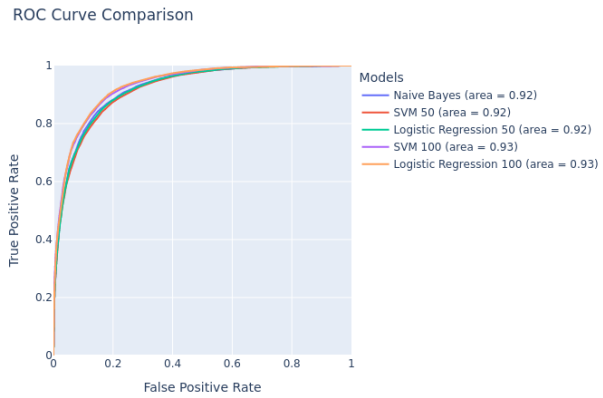

Figure 4: Courbes AUC-ROC des 5 modèles

6 Conclusion

A travers ce projet, nous nous sommes concentrés sur l'analyse de revue de film grâce à différentes approches. Notre objectif était de créer un embedding de la revue puis d'entraîner différents modèles pour voir lesquels étaient les meilleurs. Nous avons réussi à avoir des résultats très satisfaisant en utilisant des validations croisées.

Dans cette conclusion, j'aimerai relevé le fait que aujourd'hui, il existe des nouveaux outils pour faire ce type de classification grâce à l'aide des LLM. Nous n'avons pas abordé plus en détail dans le rapport, mais il y a dans le repository github un notebook qui vient finetuner un modèle déjà entraîner pour lui apprendre à classer les revues. Nous arrivons à une accuracy de 89 %