Analyse de sentiments

Frédéric Wantiez – Pierre Vigier

15 juin 2016

1 Description du problème

1.1 Intérêt

L'analyse de sentiments, *sentiment analysis* ou *opinion mining* en anglais, est l'extraction d'information subjective dans une source textuelle. En d'autres mots, l'analyse de sentiments consiste à dire si un texte dégage un sentiment positif ou négatif. Ce procédé est déjà utilisé de manière industrielle. En marketing, il permet de déceler les promoteurs et les détracteurs d'un produit ou d'une marque. Une autre application importante est le minage des réseaux sociaux comme Twitter et des blogs afin de prédire des mouvements de marché ou la popularité d'un produit.

1.2 Formalisation

L'analyse de sentiments est un problème d'apprentissage supervisé. Notons V l'ensemble de tous les mots possibles et $V^* = \cup_{n\geq 0} V^n$ l'ensemble des textes sur ce vocabulaire. Soit $x_1, ..., x_N \in V^*$ des textes et $y_1, ..., y_N \in S$ le sentiment associé à chacun des textes. Ces sentiments peuvent être des valeurs dans $S = [0, 1]$ où 0 signifie que le texte est "très négatif" et 1, "très positif". Une variante plus simple est d'avoir les $(y_i)_{i\in 1,...,N}$ dans $S = \{0, 1\}$ où 0 signifie "négatif" et 1, "positif". Notre objectif est de déterminer une fonction f telle que $\forall i \in 1, ..., N, y_i \approx f(x_i)$ et qui devra, de plus, bien généraliser sur des textes jamais vus auparavant. Dans le cas où S est discret, il s'agit d'un problème de classification. Dans le cas continu, il s'agit d'un problème de régression.

Nous allons essentiellement nous concentrer sur le problème de classification. Plusieurs types d'entrée et plusieurs types de classifieurs seront essayés sur le problème. La mesure de performance choisie est la précision $A(y_1, ..., y_N, \hat{y}_1, ..., \hat{y}_N) = \frac{1}{N}\sum_{i=0}^{N} 1_{y_i=\hat{y}_i}$. Il s'agit du taux de textes bien classifiés. On remarquera que nous utilisons le terme "précision" pour traduire le terme "accuracy" anglais et qu'il ne s'agit pas de la précision comme définie en recherche d'information. L'objectif est de la maximiser. Elle nous permettra de comparer les performances des différents algorithmes. De plus, nous regarderons parfois des matrices des confusions et des courbes ROC pour avoir plus d'information.

2 Données

Il est assez facile de créer un ensemble de données pour entraîner nos algorithmes. En effet, il suffit de trouver un site où l'on peut commenter et mettre des notes sur des produits. La valeur numérique de la note correspond alors au sentiment dégagé par le texte. Cette configuration est présente sur les sites d'e-commerce comme Amazon ou sur les sites de critiques comme IMDB ou Rotten Potatoes.

Nous utilisons l'ensemble de données mis à disposition par Maas et al. [2]. Il s'agit d'un ensemble de 50 000 avis en anglais sur IMDB. Il est découpé en un ensemble d'apprentissage et un ensemble de test. À chaque avis est associé un label 0 ou 1 selon que l'avis est négatif ou positif. Le label 0 correspond à une note inférieure à 4 tandis que le label 1 correspond à une note supérieure à 7. Les notes de 5 et 6 sont exclues car représentant un avis neutre. Il y a autant d'avis positifs que d'avis négatifs dans l'ensemble d'apprentissage ainsi que dans l'ensemble de test. Dans la figure 1, on retrouve un avis négatif et un avis positif. On remarque que des fragments de code HTML et des libertés typographiques peuvent être présents.

2.1 Méthodologie

Afin de mener notre étude, nous avons choisi d'utiliser le langage Python (version 3.5) qui a l'avantage de présenter un écosystème de technologies liées à l'apprentissage automatique qui est développé et accessible. Nous utiliserons notamment les libraires suivantes :

- *pandas* pour manipuler et analyser des données ;
- *Numpy* pour l'algèbre linéaire ;
- *Matplotlib* pour les graphiques ;
- *scikit-learn* pour les algorithmes d'apprentissage automatique ;
- *TensorFlow* pour les réseaux de neurones ;
- *NLTK* pour manipuler le langage naturel.

Nous allons devoir faire face à plusieurs difficultés. Premièrement, nous devons prétraiter les avis en enlevant le code HTML. Nous devrons aussi réussir à modéliser les avis, qui sont des chaînes de caractères de taille variable, par des caractéristiques permettant d'utiliser des algorithmes d'apprentissage. La plupart d'entre eux nécessitant des entrées de taille fixe et numériques. Finalement, nous verrons que réussir à modéliser l'ordre des mots sera une nécessité pour améliorer la précision de la classification.

Notre démarche est assez simple, nous allons procéder par complexité croissante. Nous allons commencer par utiliser les modèles et algorithmes les plus "naïfs" puis analyser les faiblesses de chaque méthode afin d'avancer et proposer des solutions pour améliorer nos résultats.

id	sentiment	review
0_3	0	Story of a man who has unnatural feelings for a pig. Starts out with a opening scene that is a terrific example of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it's singers. Unfortunately it stays absurd the WHOLE time with no general narrative eventually making it just too off putting. Even those from the era should be turned off. The cryptic dialogue would make Shakespeare seem easy to a third grader. On a technical level it's better than you might think with some good cinematography by future great Vilmos Zsigmond. Future stars Sally Kirkland and Frederic Forrest can be seen briefly.
9999_8	1	The plot had some wretched, unbelievable twists. However, the chemistry between Mel Brooks and Leslie Ann Warren was excellent. The insight that she comes to, ""There are just moments,"" provides a philosophical handle by which anyone could pick up, and embrace, life. That was one of several moments that were wonderfully memorable.

FIGURE 1 – Exemple d'un commentaire négatif et d'un commentaire positif extrait de l'ensemble d'apprentissage.

3 Premières tentatives

Nous allons commencer par utiliser deux représentations pour nos textes, la première étant les sacs de mots et la seconde se basant sur les vecteurs de mot. Dans chaque cas, nous allons nous concentrer sur la création des caractéristiques. Nous utiliserons pour commencer des algorithmes d'apprentissage relativement simples.

3.1 Sac de mots

3.1.1 Description

Les sacs de mots [5] est une représentation très simple. Numérotons les éléments de V, on a alors $V = \{w_1, ..., w_D\}$. Prenons un texte $t \in V^*$ et notons

$tf_{i,t} = card(\{j, t_j = w_i\})$ le nombre d'occurrences du mot w_i dans le texte t. Le sac de mots d'un texte t est le vecteur b de $\mathbb{R}^D$ tel que $\forall i \in 1, ..., D, b_i = tf_{i,t}$. Autrement dit, la i^e coordonnée du sac de mots de t est le nombre d'occurrences du mot w_i dans t. Par exemple, si l'on considère les deux phrases suivantes :

$$S_1 = \text{"Bob aime les films d'action."}$$
$$S_2 = \text{"Alice préfère les films d'amour."}$$

Le vocabulaire commun aux deux phrases est :

$$V = \{\text{Bob}, \text{aime}, \text{les}, \text{films}, \text{d}, \text{action}, \text{Alice}, \text{préfère}, \text{amour}\}$$

En numérotant les mots dans l'ordre affiché ci-dessus, on a alors que les sacs de mots associés aux phrases S_1 et S_2 sont :

$$b_1 = (1, 1, 1, 1, 1, 1, 0, 0, 0)^T$$
$$b_2 = (0, 0, 1, 1, 1, 0, 1, 1, 1)^T$$

Afin d'obtenir des sacs de mots, nous avons nettoyé les critiques en enlevant le code HTML et en remplaçant la ponctuation par des espaces. Ensuite, nous avons utilisé les espaces pour découper les mots. Puis nous avons limité notre vocabulaire aux 5000 mots les plus courants afin d'avoir des vecteurs de taille raisonnable.

3.1.2 Apprentissage

Nous avons choisi de commencer nos tests en utilisant la régression logistique et les forêts aléatoires. Ces deux algorithmes ont l'avantage d'être facile à mettre en place et sont relativement rapides. De plus, ils sont relativement différents dans leur fonctionnement. La régression logistique modélise seulement une interaction linéaire entre les composantes du vecteur d'entrée contrairement aux forêts aléatoires. Cependant, on peut interpréter les coefficients de la régression logistique ce qui n'est pas possible pour les forêts aléatoires. On peut retrouver les différents résultats sur la figure 2.

Forêt aléatoire à 100 estimateurs + BOW	0.84356
Forêt aléatoire à 100 estimateurs + BOW + TF-IDF	0.83952
Régression logistique + BOW	0.84748
Régression logistique + BOW + TF-IDF	0.88308

FIGURE 2 – Précisions des sacs de mots avec différents algorithmes (BOW : sacs de mots, TF-IDF : sacs de mots pondérés par IDF)

Nous avons testé une variante des sacs de mots qui améliore significativement les résultats avec la régression logistique. Les composantes des sacs de mots sont

maintenant pondérées par une mesure de la rareté d'un mot, IDF [5] que l'on définit comme suit :

$$\forall i \in \{1, ..., D\}, idf_i = log(\frac{N}{N_i})$$

où $N_i = card(\{k, tf_{i,x_k} > 0\})$ est le nombre de critiques qui contiennent le mot w_i. La i_e composante du sac de mot d'un texte t est alors $b_i = tf_{i,t} \times idf_i$.

On peut retrouver sur la figure 3, les matrices de confusion pour la régression logistique et les forêts aléatoires. On remarque qu'il y a environ autant de faux positif que de faux négatifs.

		Sentiment prédit	
		0	1
Sentiment réel	0	10674	1826
	1	1987	10513

		Sentiment prédit	
		0	1
Sentiment réel	0	10692	1808
	1	2103	10397

FIGURE 3 – Matrice de confusion de la régression logistique et des forêts aléatoires pour des sacs de mots.

La régression logistique a comme avantage qu'il est possible d'interpréter le modèle après apprentissage. Ainsi dans la régression logistique, à chaque mot est associé un poids. Et pour émettre une prédiction, le classifieur fait une combinaison linéaire des ces poids comme sur la figure 4. On peut alors interpréter le poids associé à chaque mot comme le sentiment que porte ce mot, seul. Dans les figures 5 et 6, on retrouve les mots jugés les plus négatifs et les plus positifs par la régression logistique.

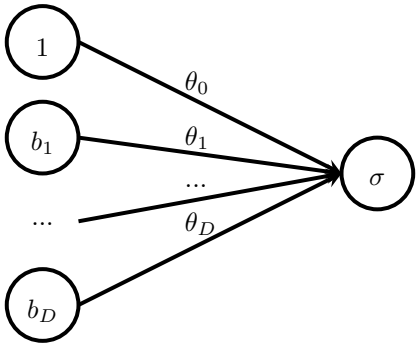

FIGURE 4 – Représentation d'une régression logistique sous forme de réseau.

```
NEGATIVE
1. worst (-9.207327806842622)
2. bad (-7.259958398881469)
3. awful (-6.415897979257145)
4. waste (-6.330645081158189)
5. boring (-5.972322478351235)
6. poor (-5.290181559864669)
7. terrible (-4.680185463018839)
8. worse (-4.432747319290786)
9. nothing (-4.411951085231576)
10. dull (-4.377273111550886)
```

FIGURE 5 – Les 10 mots ayant les poids associés les plus négatifs

```
POSITIVE
1. great (6.731768881148247)
2. excellent (6.010829696751981)
3. perfect (4.944364766157447)
4. best (4.721911978651118)
5. wonderful (4.516264655915644)
6. amazing (4.08891536519038)
7. today (3.6921102148583946)
8. favorite (3.658223120278884)
9. loved (3.589046275536792)
10. well (3.5422155660495744)
```

FIGURE 6 – Les 10 mots ayant les poids associés les plus positifs

3.2 Vecteurs de mots

3.2.1 Description

Une autre approche est d'essayer de représenter les mots avec une représentation continue. Le premier avantage de cette méthode est de fournir une représentation compacte des mots. En effet, avec les sacs de mots, les entrées sont des vecteurs de taille $card(V)$ ce qui induit un grand nombre de paramètres dans les algorithmes d'apprentissage ensuite. Au contraire, avec les représentations continus, nous pouvons fixer arbitrairement la taille des vecteurs qui vont représenter les mots du vocabulaire. Plus le vecteur aura de coordonnées, plus la représentation sera fine. Nous avons pas exemple choisi des vecteurs de $\mathbb{R}^{300}$. Nous noterons dans la suite v_i le vecteur de $\mathbb{R}^{300}$ associé au mot w_i de V.

Il existe de nombreuses méthodes pour générer de telles représentations, nous avons choisi le modèle Skip-gram proposé par Mikolov et al. [3] [4]. Le modèle consiste en un réseau de neurone, voir figure 7, où à partir d'un mot, nous essayons de prédire son contexte i.e. les mots qui l'entourent.

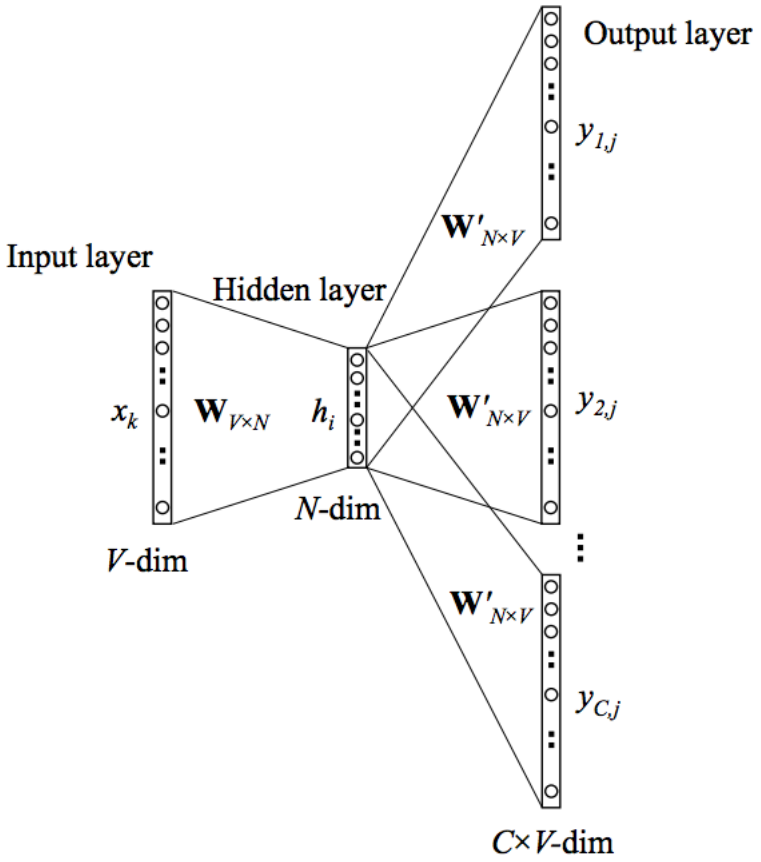

FIGURE 7 – Réseau de neurones du modèle Skip-gram (source : Wikimedia).

Ce modèle a l'avantage de fournir de l'information syntaxique et sémantique sur les mots. En effet, des mots qui se retrouvent souvent dans des contextes similaires auront des vecteurs proches. Et s'ils se retrouvent dans des contextes similaires c'est qu'ils ont des fonctions grammaticales et des sens proches. Pour s'en apercevoir, nous avons projeté nos vecteurs dans $\mathbb{R}^2$ pour les visualiser en utilisant l'algorithme t-SNE [6] qui préserve les voisinages (figure 8).

En particulier, on peut remarquer la présence de clusters de mots. Nous avons mis en évidence dans la figure 9 un cluster de mots liés au cinéma 9 et un autre contenant des mots-outils.

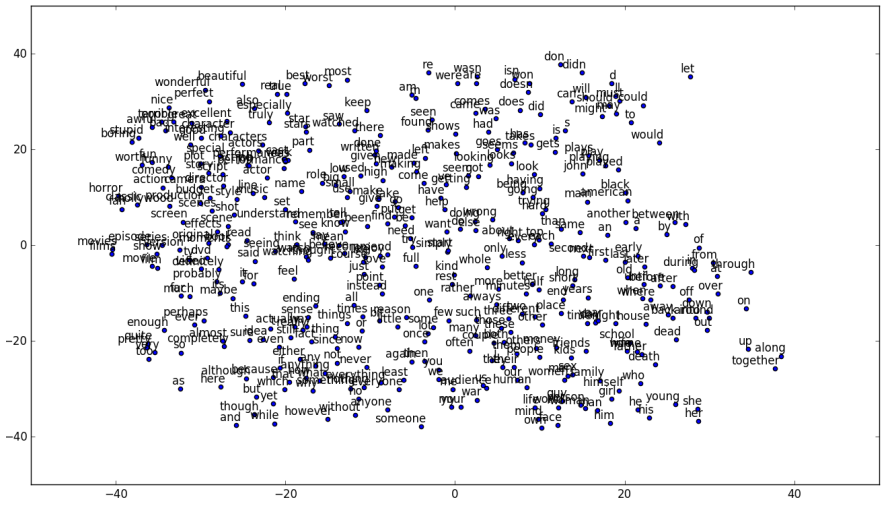

FIGURE 8 – Projection des vecteurs de mots en 2D en utilisant l'algorithme t-SNE.

3.2.2 Apprentissage

Nous avons donc une représentation des mots cependant, nous avons besoin de représenter des textes. Une solution simple est de faire une combinaison linéaire des vecteurs représentants les mots du texte. Comme pour les sacs de mots, plusieurs variantes s'offrent à nous. Nous pouvons supposer que tous les mots ont un poids égal, l'application associant à un texte t son vecteur sera alors définie par :

$$h_{vec}(t) = \frac{\sum_{w_i \in t} tf_{i,t} v_i}{\sum_{w_i \in t} tf_{i,t}}$$

Cependant, on peut aussi considérer que les mots ont des poids différents selon leur rareté, et donc comme pour les sacs de mots, nous allons pondérer par l'IDF :

$$h_{vec+idf}(t) = \frac{\sum_{w_i \in t} tf_{i,t} idf_i v_i}{\sum_{w_i \in t} tf_{i,t} idf_i}$$

De même que pour les sacs de mots, nous avons mesuré la précision de la régression logistique et des forêts aléatoires avec comme entrées des vecteurs de mot. On retrouvera ces résultats dans la figure 10.

Ici contrairement aux sacs de mots, la pondération des mots par l'IDF n'offre aucune amélioration. Cela peut s'expliquer par le fait que des mots rares mais ne portant aucun sens utile pour l'analyse de sentiments vont diminuer l'importance d'autres mots importants pour l'analyse de sentiments mais plus courants.

3.3 Conclusion

Afin d'y voir plus clair et avoir des pistes pour améliorer nos modèles, nous avons tracé des courbes d'apprentissages que l'on peut retrouver sur la figure 11.

On remarque clairement, en particulier pour les sacs de mots, que le modèle n'apprend plus tôt dans l'apprentissage. Les modèles ont un grand biais. Pour améliorer notre précision rajouter des données est donc inutile, il nous faut choisir des modèles avec plus de variance i.e. avec plus de paramètres.

Une source d'erreur plutôt intuitive est qu'on ne prend pas en compte l'ordre des mots. Par conséquent, nos modèles ne peuvent pas prendre en compte des structures grammaticales complexes. Par exemple, si nous classifions la phrase "It was not amazing.", qui est négative, avec le modèle de la régression logistique, nous obtenu le label 1 qui signifie que la phrase est positive. C'est compréhensible car le modèle considère les mots indépendamment les uns des autres et le mot "amazing" a un poids positif et élevé. Une piste pour rajouter des paramètres à notre modèle est donc de prendre en compte l'ordre des mots.

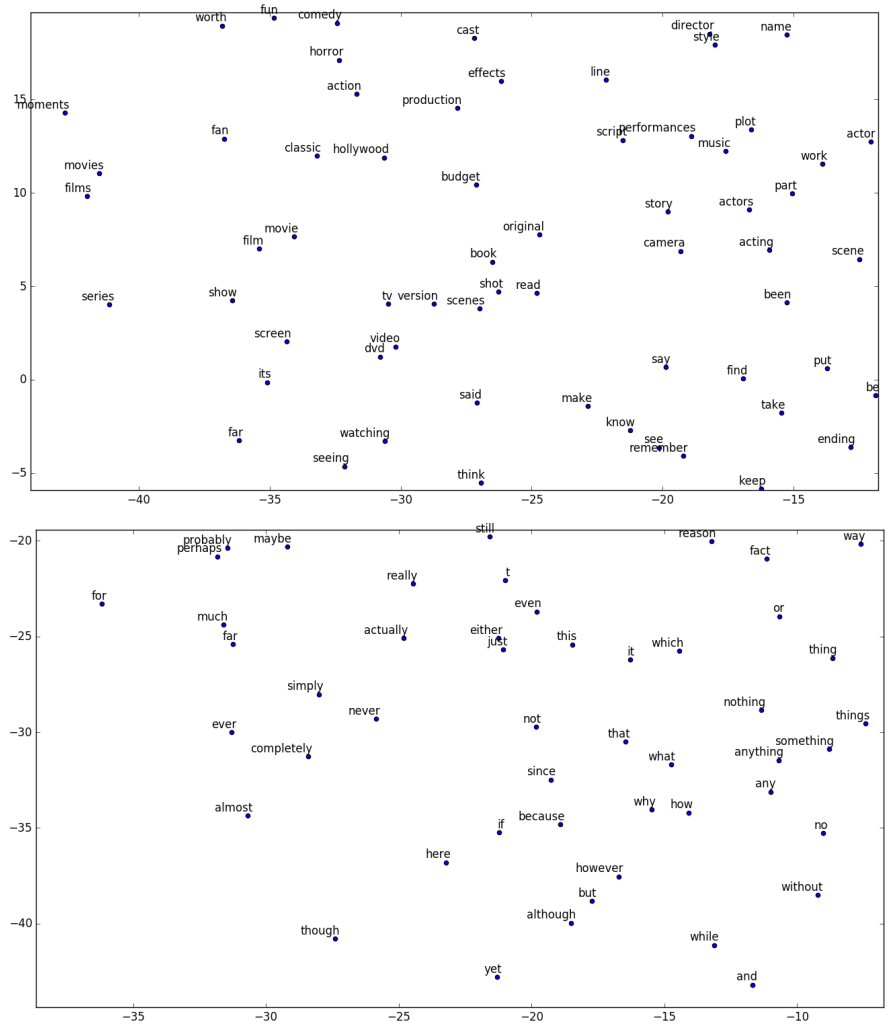

FIGURE 9 – Un cluster contenant des mots liés au mot du cinéma et un autre contenant des mots-outils.

Forêt aléatoire à 100 estimateurs + Vec	0.7954
Forêt aléatoire à 100 estimateurs + Vec + TF-IDF	0.79416
Régression logistique + Vec	0.81652
Régression logistique + Vec + TF-IDF	0.81688

FIGURE 10 – Précisions des vecteurs de mot avec différents algorithmes (Vec : vecteurs de mots, TF-IDF : vecteurs de mot pondérés par IDF)

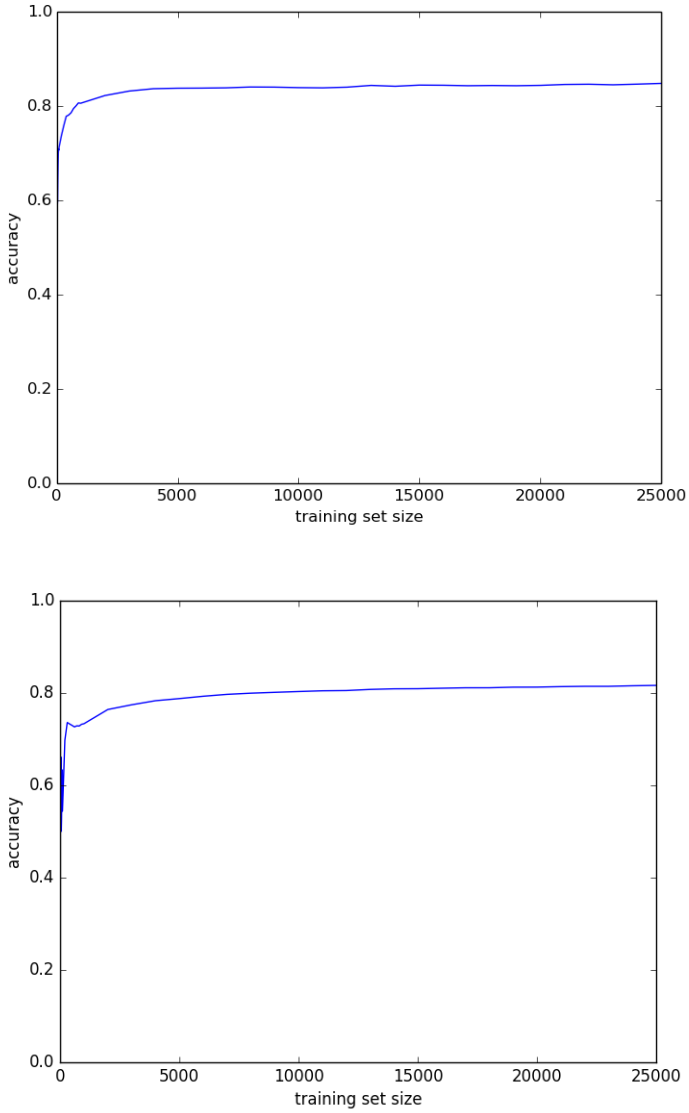

FIGURE 11 – Évolution de la précision en fonction de la taille de l'ensemble d'apprentissage avec des sacs de mots en entrée (en haut) et des vecteurs de mots (en bas).

4 Prise en compte de l'ordre des mots

4.1 N-grammes

Les n-grammes, $n \in \mathbb{N}$, d'un texte t sont les n-uplets de mots consécutifs de t. Considérons la phrase t = "Bob aime les films d'action.", les bigrammes ou 2-grammes de t sont $\{(\text{Bob}, \text{aime}), (\text{aime}, \text{les}), (\text{les}, \text{films}), (\text{films}, \text{d}), (\text{d}, \text{action})\}$. Les n-grammes permettent de rendre compte de structures syntaxique comme "not like". Plus n est grand, plus nous pourrons modéliser des structures complexes. Cependant, le nombre de n-uplets croient rapidement avec n, en $O(card(V)^n)$. Pour n grand, il y aura peu d'occurrences des n-uplets, ce ne seront donc pas des caractéristiques pertinents pour la classification. Nous nous limiterons donc à $n = 3$.

Maintenant, nous pouvons faire des sacs de n-grammes à la place des sacs de mots, lesquels étaient des 1-grammes. Dans la pratique, nous n'allons pas considérer soit que les bi-grammes ou que les tri-grammes. Nous allons prendre les k-grammes pour $k \in \{1, 2, ..., n\}$ les plus courants. On retrouvera dans la figure 12 les performances des n-grammes en entrée avec la régression logistique pour différents n, différentes valeurs du nombre de caractéristiques C, et en utilisant ou non TF-IDF.

C	n	TF-IDF	Précision
5000	1	Non	0.85164
10000	1	Non	0.85432
15000	1	Non	0.85664
5000	2	Non	0.8582
10000	2	Non	0.86844
15000	2	Non	0.87544
5000	1	Oui	0.88308
10000	1	Oui	0.88364
15000	1	Oui	0.88412
5000	2	Oui	0.88848
10000	2	Oui	0.89312
15000	2	Oui	0.89432
20000	2	Oui	0.8954
30000	2	Oui	0.89564

FIGURE 12 – Précisions des sacs de n-grammes en entrée d'une régression logistique.

On remarque que si on prend que les 1-grammes, augmenter le nombre de caractéristiques n'améliore pas les performances. Par contre rajouter des bigrammes fait presque gagner 0.01 de précision. Nous avons essayé avec des trigrammes mais ça ne donne pas de meilleurs résultats.

4.2 Vecteurs de paragraphe

Pour représenter nos textes entièrement, nous avons utilisé l'approche de Mikolov et al. [1] qui consiste à étendre le cadre de Word2vec à des morceaux entiers de textes comme des paragraphes ou des phrases. La démarche reste la même, le réseau essaie de prédire pour un contexte donné, le mot suivant. La différence réside néanmoins dans la présence d'une Paragraph Matrix, voir figure 13, qui stocke l'information contenue dans le paragraphe. Cette représentation offre l'interêt de nous fournir des représentations de taille fixe de nos textes. Nous obtenons ainsi des vecteurs de $\mathbb{R}^n$ avec $n = 300$ que nous pouvons fournir à nos algorithmes d'apprentissage.

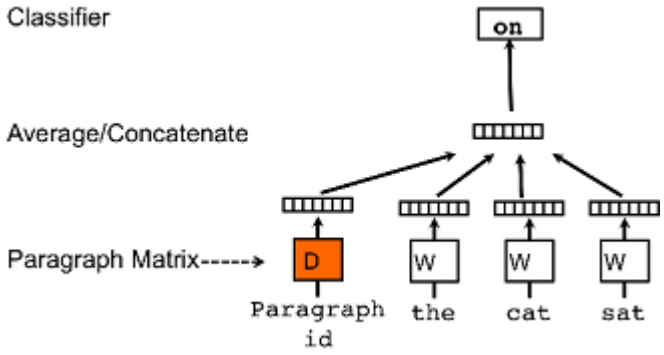

FIGURE 13 – Framework d'apprentissage des paragraph vectors (source : Mikolov et al. [1]).

Le second avantage de cette représentation est qu'elle permet de prendre en compte l'ordre des mots de nos textes tout en gardant la sémantique des mots présents. Pour s'en convaincre, on peut analyser les représentations des mots obtenues avec ce modèle, que nous avons inclus dans la figure 14.

Grâce à cette représentation de nos textes, nous pouvons entraîner nos algorithmes de classification. L'article de Mikolov et al. présente plusieurs modèles, nous avons utilisé le PV-DM (Paragraph Vector with Distributed Memory) dans deux de ses variantes, l'une qui concatène les vecteurs de contextes et l'autre qui considère leur moyenne. Les résultats obtenus sont présentés dans le tableau 15.

On peut aussi regarder les courbes ROC pour comparer les deux variantes. Les courbes sont données figure 16 et figure 17

```
In[1]: model.most_similar('interesting')
Out[1]:
[('enjoyable', 0.549770712852478),
 ('entertaining', 0.5360784530639648),
 ('important', 0.5295416712760925),
 ('intriguing', 0.4986931085586548),
 ('amazing', 0.49587947130203247),
 ('exciting', 0.4942770004272461),
 ('excellent', 0.4942253828048706),
 ('awesome', 0.46036389470100403),
 ('amusing', 0.4536857306957245),
 ('impressive', 0.448122501373291)]
```

FIGURE 14 – Les 10 mots les plus semblables à 'interesting' pour un modèle PV-DM avec moyenne.

Régression logistique + PV-DM (concat.)	0.883
Régression logistique + PV-DM (mean)	0.823

FIGURE 15 – Précision des paragraph vectors avec les deux variantes utilisées pour des vecteurs de dimension 300

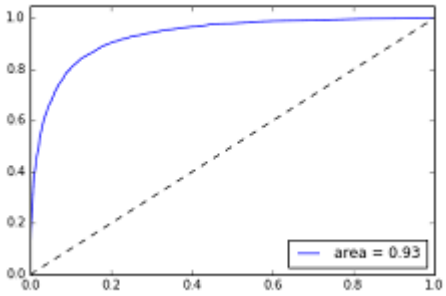

FIGURE 16 – Courbe ROC pour le modèle PC-DM avec concaténation.

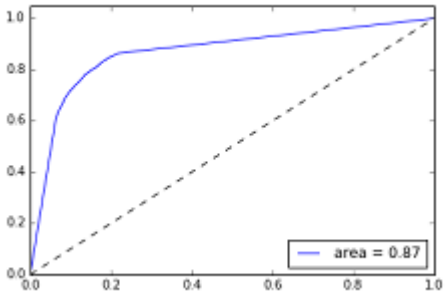

FIGURE 17 – Courbe ROC pour le modèle PC-DM avec moyenne.

4.3 Derniers essais

Finalement, nous avons concaténé les sacs de n-grammes et les vecteurs de paragraphes. En prenant des sacs de n-grammes de longueur 10000 et des vecteurs de paragraphe de longueur 300 puis en les concaténant et en entraînant une régression logistique, nous obtenons une précision de 0.90116, il s'agit de notre meilleur résultat.

5 Conclusion

Nous avons proposé différents modèles de complexité croissante pour résoudre le modèle. Nous avons vu que des méthodes très simples comme les sacs de mots donnent de très bons résultats. Puis qu'il est possible d'obtenir d'encore meilleurs résultats en prenant en compte l'ordre des mots.

Une autre piste pour obtenir de meilleurs résultats que nous n'avons pas eu le temps de mettre en place est d'utiliser des réseaux de neurones récurrents comme les LSTM. L'avantage de ces réseaux est qu'ils prennent des entrées de taille variable, il n'y a donc pas besoin de créer des caractéristiques pour représenter une phrase ou un paragraphe.

Au travers de ce projet, nous avons découvert une large palette des techniques utilisées en traitement automatique du langage naturel, ce qui a été très enrichissant. Nous nous sommes de plus familiarisés avec l'écosystème de l'apprentissage automatique sous Python.

Références

[1] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv :1405.4053*, 2014.

[2] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv :1301.3781*, 2013.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[5] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[6] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605) :85, 2008.