Rapport de projet
présenté par :

Alexane Jouglar
Master 1
« Langue et informatique »
Méthodologie de la recherche en informatique
Enseignants :
Laurence Devillers,
Gaël Lejeune

Table des matières

Table des figures

Introduction

Les dernières décennies ont été marquées par le développement du numérique. Ce développement a entraîné dans la science du langage une modification profonde de son mode de fonctionnement. Mis à part quelques enquêtes linguistiques longues et fastidieuses s'étalant sur plusieurs mois voire plusieurs années, il était avant très rare de travailler sur une variété d'exemples attestés et nous parlions de "linguistique de fauteuil" pour désigner la linguistique théorique et fondée principalement sur l'introspection et le sentiment de la langue. Désormais nous avons la possibilité d'accéder à de grands jeux d'exemples attestés et sommes donc passés à une linguistique de corpus. Ce sont maintenant sur ces jeux de donnés que nombre de linguistes fondent leurs analyses. Dans ce contexte, le classement automatique des textes, et l'analyse des opinions et sentiments des personnes constituent un domaine de recherche très actif. Dans cette optique, et dans ce projet d'initialisation au traitement automatique des langues, nous avons nous-mêmes procédés au traitement automatique de textes. Nous avions la possibilité de traiter un corpus de tweets afin d'étudier leur polarité ; ceci dit, le choix s'est finalement porté sur l'analyse d'articles de journaux afin de les classer par pays concerné. Ceci dit, la polarité ne concerne pas seulement l'analyse des sentiments et c'est la raison pour laquelle nous travaillerons aussi sur un autre corpus rassemblant des courriers électroniques. Le but de ce travail est de tester des classifieurs et de connaître leur effectivité (ou leur non effectivité), de faire varier les paramètres de classification, et de comparer les résultats obtenus dans l'un et dans l'autre des corpus afin de savoir quelles méthodes sont les plus à même de donner des résultats satisfaisants. Pour ce faire, ce rapport se divise en quatre parties. En premier lieu un état de l'art, tout ce qu'il y a de plus classique, permettra d'évoquer les précédents de ce type d'expérience et leurs conclusions. Ce n'est qu'alors que nous détaillerons les jeux de données utilisés, puis les méthodes utilisés. Et dernier lieu, nous présenterons les résultats obtenus.

Chapitre 1

Etat de l'art

Nous avons indiqué dans l'introduction avoir eu la possibilité de procéder à l'étude de tweets. Il y a des précédents dans ce domaine. Ainsi l'article de vulgarisation scientifique publié sur Data Scientest par Gary B. [B, 2020] non seulement s'intéresse à l'étude des sentiments révélés par les tweets mais il combine cette étude avec une approche temporelle. Le jeu de données saisi est en effet différent. Alors que celui qui sera présenté en annexe indique uniquement le sentiment du tweet, le jeu de données utilisé pour l'article indique également sa date de publication. En quelques manipulations sur Python, le journaliste parvient à déterminer que la proportion de tweets négatifs est accrue en juin. Si le corpus de tweets a été téléchargé tel quel sur la plateforme Kaggle, il peut être construit de toute pièce comme expliqué par Stéphane Tufféry dans son ouvrage [Tufféry, 2007]. Dans ce même ouvrage, l'auteur évoque la détection de spams à l'aide d'un "réseau de neurones récurrent LSTM" où il divise son corpus avec une proportion de 80-20 puis entraîne le modèle avec l'algorithme d'optimisation ADAM. L'analyse du sentiment ou *opinion mining* est également un sujet de cet ouvrage. On peut détecter le sentiment d'un texte donné selon deux approches distinctes. La première consiste en l'attribution d'un score à chaque mot en fonction de sa polarité. La seconde, plus exacte, repose sur le *machine learning* (ou *apprentissage statistique* en français). Les méthodes de *machine learning* indiquées dans le chapitre en question s'implémentent avec le langage R et non Python comme c'est le cas pour le travail présent. Là où Python est un langage multi-tâche, R est spécifiquement destiné au traitement des données.

Ce qui ressort de ces différentes études, et du traitement du langage en général, est la difficulté pour les algorithmes actuels de reconnaître toutes les subtilités que possède le langage humain et en particulier :

— l'ironie ("C'est génial" est une expression à double tranchant, elle peut

être prise dans son sens littéral ou bien signifier son contraire) ;
— les figures de style comme les métaphores (« Midi, dans la mer Rouge. De la lumière, de la lumière, tant de lumière que l'on admire et l'on s'étonne, comme si, au sortir d'une espèce de demi-nuit, les yeux s'ouvraient davantage, voyaient plus clair, toujours plus clair. » [Loti, 2008] : il ne s'agit pas là de la lumière du soleil mais de l'accès à la clairvoyance) ;

et bien d'autres phénomènes relevant de la langue naturelle, autrement dit tout ce qui n'est pas directement accessible et qui demande une capacité d'abstraction. D'ailleurs un exemple d'étude sur la présence de ces phénomènes dans les tweets a été fourni dans un article de 2017 [Farah Benamara, 2017]. Cet article est d'autant plus pertinent pour ce travail qu'il fait parti d'un atelier appelé DEFT (Défi Fouille de Texte) et qui tous les ans "[confronte], sur un même corpus, des méthodes et logiciels d'équipes différentes" [def, 2005]. Le but est toujours d'obtenir le meilleur résultat possible et c'est ce que nous avons tenté de faire dans le présent travail, à savoir : tester plusieurs méthodes de classification pour connaître la meilleure.

A quelle fin voudrait-on faire du *text mining* (traduit en français par "traitement automatique de texte") ? La réponse a été donnée en 1957 par Hans Peter Luhn dans une étude disponible sur le site d'IBM et dont le titre est "The Automatic Creation of Literature Abstracts" [Luhn, 1957]. On peut y lire notamment dans l'introduction : "The purpose of abstracts in technical literature is to facilitate quick and accurate identification of the topic of published papers". Ce type de traitement permet à la fois de soulager les humains du travail fastidieux qu'il représente et d'éliminer les biais de traitement qui peuvent se présenter.

Chapitre 2

Jeu de données

Nous introduisons dans ce rapport l'étude de deux jeux de données : le premier est constitué de courriers électroniques (5728 instances au total), le second est constitué d'articles. Ces deux jeux préalablement construits par leurs auteurs respectifs Balakishan Molankula [Molankula, 2018] et David D. Lewis [Lewis, 1997] sont fournis en deux formats différents. Le premier est au format csv et les données sont rangées en deux catégories : 1 ou 0. Le 1 indique que le mail est un spam, le 0 qu'il ne l'est pas. Le deuxième jeu de données est fourni sous la forme de 22 fichiers SGML (l'un des fichiers n'a pas pu être utilisé car non reconnu par le programme de lecture utilisé) ; la DTD y est incluse. Dans chacun de ces fichiers il y a environ mille articles et chacun de ces articles est issu de l'agence Reuters, fondée en 1851 à Londres. Ces articles sont entourés de certaines informations (date, parfois le thème, le pays dont parle l'article). Voici des exemples issus du jeu de mails :

— "Subject : security alert - confirm your national credit union information $\Longrightarrow$ " (spam)

— "Subject : re : requests for help thanks vince ." (ham)

Des remarques peuvent être faites ici : on remarque que les spams peuvent contenir des caractères spéciaux susceptibles de les catégoriser comme tels ("$\Longrightarrow$" par exemple), là où les autres mails possèdent peu de caractères autres que la virgule et le point. La marque "re :" indique une réponse à un mail précédent et est donc peu susceptible de se retrouver dans un spam. Et voici des exemples issus du jeu de données Reuters :

— "Key Tronic corp said it has received contracts to provide seven original equipment manufacturers with which it has not done business recently with over 300,000 computer keyboards for delivery within the next 12 months. The company said "The new contracts represent an annual increase of approximately 25 pct in unit volume over last year." (titre : "KEY TRONIC & lt ;KTCC GETS NEW BUSINESS" ; pays :

Etats-Unis)

— "President Jose Sarney today declared "a war without quarter" on inflation and said the government would watch every cent of public expenditure. Sarney, addressing his cabinet live on television, also reiterated that he intended to remain in power for five years, until 1990. There has been a long-running political debate about how long his mandate should be. Brazil is currently suffering from the worst inflation of its history. In April monthly inflation reached 21 pct. ("BRAZIL'S SARNEY RENEWS CALL FOR WAR ON INFLATION" ; Brazil)

Pour les deux jeux de données, la langue utilisée est l'anglais.

Nous cherchons à donner à voir ces exemples à la machine afin de lui enseigner la manière dont sont classées les données, et cela afin qu'elle puisse le faire ensuite sans assistance. Il s'agit donc de séparer les données en deux ensembles : un *train* et un *test*. C'est à dire qu'un extrait des données est utilisé pour indiquer à la machine comment les classer et les données restantes permettent de voir si la machine a bien appris. Toutes les expériences sont menées sur Python. Dans le premier exemple, on associe respectivement 0 et 1 aux catégories X et y. Dans le deuxième exemple, on associe chaque article à un pays (il y a 101 pays en tout) ; il aurait été opportun de pouvoir associer chaque article à un thème mais tous les articles n'en ont pas, en revanche la grande majorité des articles sont rattachés à un pays.

Pour finir, quelques données chiffrées sur le corpus de mails :

— le jeu comporte 5728 mails.
— on compte 1878460 tokens en incluant la ponctuation, 1722321 sans l'inclure ;
— on compte 111600 phrases dans l'ensemble du corpus ;
— il y a 1368 spams au total et 4360 hams.
— La taille du *test* représente également 30% du corpus, soit 1718 mails ; ce qui fait que le *train* représente 4009 mails.

Quelques données chiffrées sur le corpus d'articles :

— le jeu comporte 21578 articles.
— on compte 2852163 tokens en incluant la ponctuation, 2725273 sans l'inclure ;
— on compte 123448 phrases dans l'ensemble du corpus ;
— si l'on prend en compte titres et corps ensemble, il y a 4016941 tokens en incluant la ponctuation., 3761395 tokens sans l'inclure et 173110 phrases dans l'ensemble du corpus.
— La taille du *test* représente également 30% du corpus, soit environ

6473.4 articles. Donc environ 15104.6 articles sont utilisés pour le *train*.

Chapitre 3

Méthode

Le travail se divise globalement en quatre grands mouvements : lecture, vectorisation, classification, evaluation. Vectorisation et classification forment le coeur du projet. Et prenons en considération que le passage de l'une à l'autre peut se répéter plusieurs fois, étant donné que nous avons à certains moments modifié le texte de départ (notamment lors de la suppression des *stopwords* et de la ponctuation) pour vectoriser à nouveau le corpus. De la même manière, une évaluation a été réalisée après chaque expérience.

3.1 Lecture

Il s'agit avant tout de lire le jeu de données et de le rendre accessible à Python dans le notebook : nous avons utilisé un fichier csv et des fichiers SGML. Pour importer le fichier csv, on fait appel à la librairie Panda [pandas development team, 2020]). Pour les fichiers SGML, nous nous sommes aussi servis d'un programme écrit par un ingénieur en informatique nommé Bachir et qui a publié son code sur le net [Bachir, 2018] ainsi que du module *re* qui permet de faire des opérations sur des expressions régulières. Il faut par ailleurs calculer le nombre d'instances dans les corpus, que ce soit en phrases, tokens, caractères, classes. Pour ce faire, nous avons utilisé la librairie nltk [Bird and Klein, 2009].

3.2 Vectorisation

La deuxième étape réside dans la vectorisation et la séparation du corpus en deux groupes (le *train* et le *test*). Cette étape a été réalisée une première fois en début de travail, et juste après la lecture des données évoquée précédemment, mais également plus tard lorsque l'on a voulu utiliser d'autres

caractéristiques ou rafiner la classification : c'est le cas de la stylométrie par exemple. C'est aussi le cas lorsque l'on décide de supprimer tous les *stopwords* ("mot vide" en français) ou la ponctuation. La logique voudrait que nous ayons de meilleurs résultats avec la suppression des *stopwords* par exemple ; la qualité de la classification sera évoquée dans la prochaine partie de ce rapport.

3.3 Classification

Dans la partie classification, il s'agit de tester différents classifieurs. On tente dans le notebook une première évaluation, la même pour les deux corpus, avec le module *perceptron* (algorithme inventé en 1957 par Franck Rosenblatt [Bangwa, 2019]) de la librairie *Python Scikit Learn* [Pedregosa et al., 2011]. Et au total, nous avons utilisé 6 classifieurs différents :

— *Perceptron* (évoqué ci-dessus)
— *DecisionTreeClassifier*
— *SVC*
— *KNeighborsClassifier*
— *LogisticRegression*
— *RandomForestClassifier*

3.4 Evaluation

Pour réaliser l'évaluation, nous avons utilisé la librairie *Python Scikit Learn* dont il était question dans la section précédente. Nous nous sommes servis de trois modules d'évaluation. Le premier se nomme *Classification report* (littéralement rapport de classification) et se présente sous la forme d'un tableau. On y trouve toutes les mesures pour évaluer une classification. On prendra en compte, la précision (proportion des items pertinents parmi l'ensemble des items proposés), le rappel (proportion des items pertinents proposés parmi l'ensemble des items pertinents), le support (nombre d'instances concernées), la micro f-mesure (qui permet d'avoir des classes qui comptent en fonction de leur taille), et la macro f-mesure (ne prend pas en compte la taille des classes). Le deuxième se nomme *Confusion matrix* (ou "matrice de confusion") ; plus concis, il donne sous la forme d'un tableau à deux entrées les résultats bruts obtenus. Le troisième module se nomme *precision_recall_fscore_support* et présente les mêmes résultats que le *Classification report* mais sous une autre forme. Enfin, au-delà de ces deux méthodes d'évaluation, nous avons également fait apparaître dans tous les cas la quan-

tité de bons résultats, et la quantité d'erreurs ainsi que le taux d'exactitude qui permet en un coup d'oeil de savoir si une classification est bonne ou mauvaise.

Chapitre 4

Résultats

Nous exposerons séparément les résultats obtenus avec l'un et l'autre des jeux de données avant de les mettre en relation.

4.1 Avec le jeu de données de courriels

On obtient le meilleur résultat avec le module *Logistic Regression*, et plus encore lorsque l'on procède au tri de la ponctuation et des *stopwords*. En effet, sans ce traitement le nombre d'erreurs est de 22 alors qu'avec le traitement on tombe à 20 et le taux de précision de est de 0.99. Cela est surprenant puisque j'avais pensé qu'avec les mails, la suppression de la ponctuation aurait un impact négatif sur la reconnaissance (les spams présentant a piori plus de caractères spéciaux). Le Perceptron obtient la deuxième place en terme de qualité de classification (98% d'exactitude avec les valeurs par défaut). On en voit toutes les données dans la figure 4.1. On notera dans cette figure les 1719 données qui constituent le t*test* : 1314 de *ham* et 405 de *spam*.

DecisionTreeClassifier : les scores sont moins bons et cela même en faisant varier les paramètres du module. En regardant ceux-ci sur une large échelle (de 1 à 99), on remarque que les scores ne s'améliorent pas infiniment et la quantité d'erreurs augmente après avoir baissé sans jamais atteindre un palier satisfaisant. De même avec *min_samples_split*, textitmin_samples_leaf et textitmax_features. De manière générale, pour ces trois paramètres on ne passe jamais sous la barre des 85 erreurs. De la même manière avec une forêt aléatoire, on remarque une amélioration des résultats jusqu'à $max\_depth = 95$, puis une augmentation du nombre d'erreurs et les résultats sont bons qu'avec les paramètres par défaut.

	precision	recall	f1-score	support
ham	0.99	0.98	0.99	1314
spam	0.95	0.97	0.96	405
accuracy			0.98	1719
macro avg	0.97	0.98	0.97	1719
weighted avg	0.98	0.98	0.98	1719

FIGURE 4.1 – Rapport de classification pour le Perceptron avec traitements par défaut

4.2 Avec le jeu de données Reuters

A première vue, l'ajout des titres aux corps ne semble pas concluant. Là où l'on otenait 5350 résultats corrects et 1124 erreurs avec le *Perceptron* dans le premier *notebook* (soit 83% d'exactitude), on obtient 4285 résultats corrects et 2189 erreurs dans le deuxième (66%). L'arbre de décision connaît une régression moins importante mais une régression tout de même (de 77 à 75%) et la variation des paramètres n'y change rien (les résultats restent moins bons dans le second cas).

En revanche, si l'on tient compte des n-grammes, *RandomForestClassifier* semble un module plus prometteur si l'on prend en compte les titres et les articles ensemble. On passe de 68% à 70% seulement avec l'augmentation du grammage sans avoir pris en compte les titres mais de 70% d'exactitude à 77% d'exactitude avec les titres. L'exactitude des autres classifieurs augmentent également avec l'augmentation du grammage mais les taux restent malgré tout plus bas qu'avec la forêt aléatoire. Si l'on ne prend pas en compte les titres, *LogisticRegression* obtient jusqu'à 77% d'exactitude. Les deux modules sont donc très proches.

Enfin, à la différence du notebook SpamHam, on ne connaît pas les meilleurs résultats avec le retrait de la ponctuation est des *stopwords*. En effet, on avait obtenu jusqu'à 85% d'exactitude avec le *Logistic Regression* (une partie du rapport de classification est donnée à la page suivante ; on y notera également les 6474 entités du *test*). Cette fois, ni ce module, ni *perceptron* ne parviennent à faire mieux. Avec ce dernier, on parvient à aller jusqu'à 80% mais pas au-delà, et moins encore avec l'ajout des titres (77% pour le *perceptron*, 67% pour *Logistic Regression*).

accuracy			0.85	6474
macro avg	0.49	0.33	0.38	6474
weighted avg	0.84	0.85	0.84	6474

FIGURE 4.2 – Extrait de rapport de classification pour *LogisticRegression* avec traitements par défaut

4.3 Mise en relation des deux jeux de données

Dans les deux cas, la variation des caractéristiques ne semble pas apporter à l'exactitude des résultats. Ils sont soit moins bons, soit similaires (par exemple, pour le cas des articles de presse, on tombe sous la barre des 50% avec la stylométrie seule ce qui est moins bien que ce que le hasard serait en mesure de produire, et on obtient le même score qu'avec les valeurs par défaut en combinant sac-de-mot et stylométrie soit 76%).

Egalement, il saute aux yeux que les résultats sont biens meilleurs avec les mails qu'avec les articles de journaux. Cela n'a rien d'étonnant : il y a beaucoup plus de classes dans le deuxième cas que dans le premier (c'est d'ailleurs pour cette raison qu'on ne peut montrer dans la figure 4.1 l'entièreté du rapport de classification, il prendrait beaucoup trop d'espace) et proportionnellement il n'y a pas autant de données étudiées. Il serait intéressant de mener les mêmes expériences pour les articles de presse mais avec une quantité de données plus importantes pour voir si les réflexions faites au point suivant sont réalisées.

Malgré tout, on peut quand même remarquer que le succès des modules est le même dans l'un et l'autre des cas. En effet : *LogisticRegression* est le module qui obtient le meilleur score avec les mails (99 % d'exactitude), comme avec les articles de presse (85 % d'exactitude).

Conclusion

Les expériences menées se sont étalées sur plusieurs semaines et sur une même langue : l'anglais. On trouvera en index un exemple des mêmes expériences avec des données de langue française. Le classifieur qui semble le plus convaincant est le *LogisticRegression* même si la variation des paramètres peut lui faire perdre de sa superbe. Le bon sens dirait que la suppression des *stopwords* et de la ponctuation augmenterait les scores mais ce n'est pas systématiquement le cas ; par ailleurs, l'ajout des titres aux articles de journaux laisserait penser également que l'on peut obtenir de meilleurs résultats mais l'expérience montre que ce n'est pas le cas puisque dans l'ensemble les résultats étaient bien moins bons dans le deuxième notebook. Il serait bon dans le futur de pouvoir à la fois augmenter la quantité d'article pour voir si l'on peut augmenter la qualité de classification ; il serait également judicieux de tester davatange le traitement pré-vectorisation des textes pour voir s'ils peuvent avoir un aspect plus bénéfique qu'on ne l'a pu observer dans nos expériences.

Annexes

Faire tourner le code des deux notebooks

Le code pour l'analyse des courriels entre en un seul notebook. Il est facile à faire tourner et a été réalisé dans le même sens que la présentation des méthodes utilisées.

Le code pour l'analyse des articles de Reuters est divisé en deux notebooks :

1. le premier notebook reproduit l'analyse faite avec les spams mais a été adaptée pour la lecture des données Reuters.
2. le deuxième notebook ajoute dans l'analyse les titres aux textes pour voir si le taux de reconnaissance est meilleurs

On ajoutera à ces deux notebooks un notebook d'initialisation qui montre comment le travail a commencé pour le traitement de ces données.

Enfin, le notebook SpamHam est facile à faire tourner ; en revanche, je ne recommande pas de réexécuter les cellules du notebook Reuters (les premières cellules prennent peu de temps ; en revanche, les cellules contenant les classifieurs sont plus chronophages).

Analyse de tweets

Les résultats avec le jeu de tweets se sont révélés extrêmement longs à recevoir. Cela est dû à la quantité de données fournis au programme (plus d'1,5 millions comme indiqué dans précédemment). C'est pour ça que nous nous n'avons traité ce jeu de données qu'en plus des deux autres (les résultats ont mis beaucoup de temps à être traités et toutes les cellules n'ont pas pu être exécutées dans le temps imparti).

Dans ce jeu de donnée, le 1 signifie que le tweet reflète un sentiment ou une émotion positifs, le 0 reflète la négativité. Pour le second d'entre eux, lConcernant le code pour l'analyse des tweets, le code est diviser en deux notebooks. On aura dans le premier, nommé *tweets*, lecture du jeu de

données, vectorisation, classification, évaluation mais l'on s'arrête à l'utilisation du *DecisionTreeClassifier*. Il faut alors se rendre sur le deuxième notebook, *Tweets(2)* dans lequel toutes les autres expériences sont menées. A noter que, bien que divisé en deux notebooks, le travail sur ce jeu de données reprend le même ordre que celui utilisé pour les courriels. Je ne recommande pas d'exécuter ces deux derniers notebooks mais d'observer simplement les résultats obtenus parce qu'ils sont extrêmement longs à faire tourner.

Dans les tweets récoltés, tout ce qui ne se rapportait pas directement à un texte n'a pas été pris en compte ; cela comprend les emoji, les hashtags, les images. Les doublons également ont été évités. Voici des exemples d'instances pour le jeu de tweets :

— "Un de mes amis m'a appelé et a demandé de la rencontrer à la mivallée aujourd'hui ... mais je n'ai pas le temps * soupir *" (tweet négatif) ;
— "Regarder lady gaga remonter au sommet ! ! ! Félicitations, gaga, vous rock ! ?" (tweet positif)

On peut d'ores-et-déjà faire certaines remarques. Les tweets sont souvent mal traduits et n'ont donc pas beaucoup de sens pour un locuteur natif du français. De ce point de vue, l'étude de ce jeu de données peut se révéler biaisée voire inutile. Il faudrait à ce titre pouvoir la comparer avec celle d'un jeu de tweets uniquement issus d'utilisateurs francophones. On remarque également que les hashtags ont été elevés mais pas les descriptions de réactions corporelles ou mentales telles "* soupir *".

Quelques données supplémentaires sur le corpus de tweets :

— on y compte 1526724 tweets,
— 25979719 tokens en incluant la ponctuation,
— 23145150 tokens si on ne l'inclut pas,
— 2580261 phrases en tout,
— 771605 tweets négatifs et 755119 tweets positifs ;
— la taille du test représente 30% du corpus soit 458017 items.

Il serait utile comme évoqué dans la deuxième partie de relancer la même expérience sur un jeu de données issues de publications francophones (et non de traductions) pour voir si les résultats sont les mêmes. On trouvera à titre indicatif ci-dessous une figure indiquant les résultats pour le *perceptron*, ce sont les meilleurs résultats ayant pu être obtenus sachant que tous les codes n'ont pu être exécutés. On trouvera dans le dossier du rendu les notebooks de ces expériences.

	precision	recall	f1-score	support
Positif	0.77	0.72	0.75	231214
Négatif	0.73	0.78	0.76	226804
accuracy			0.75	458018
macro avg	0.75	0.75	0.75	458018
weighted avg	0.75	0.75	0.75	458018

FIGURE 4.3 – Résultats obtenus pour le *perceptron* avec les paramètres par défaut

Remerciements

Je remercie l'équipe enseignante d'avoir, malgré les circonstances, continué à assurer les cours.

Bibliographie

[def, 2005] (2005). Deft (défi fouille de texte).

[B, 2020] B, G. (2020). NLP Twitter – Analyse de Sentiment. In *DataScientest*.

[Bachir, 2018] Bachir (2018). Parsing xml into pandas dataframe.

[Bangwa, 2019] Bangwa, N. (2019). The perceptron with python. *Root Admin*.

[Bird and Klein, 2009] Bird, Steven, E. L. and Klein, E. (2009). Natural language processing with python. O'Reilly Media Inc.

[Farah Benamara, 2017] Farah Benamara, Cyril Grouin, J. K. V. M. I. R. (2017). Analyse d'opinion et langage figuratif dans des tweets : présentation et résultats du Défi Fouille de Textes DEFT2017. In *DÉfi Fouille de Texte 2017 – DEFT2017*, Orléans, France.

[Lewis, 1997] Lewis, D. D. (1997). Reuters-21578 text categorization test collection.

[Loti, 2008] Loti, P. (1ère publication : 1903, édition utilisée : 2008). L'inde (sans les anglais).

[Luhn, 1957] Luhn, H. P. (1957). The automatic creation of literature abstracts.

[Molankula, 2018] Molankula, B. (2018). Spam-Email-Classifier.

[pandas development team, 2020] pandas development team, T. (2020). pandas-dev/pandas : Pandas.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830.

[Tufféry, 2007] Tufféry, S. (2007). Le traitement du langage naturel, l'analyse des réseaux sociaux. In *Big Data, Machine Learning et Apprentissage Profond*, pages 367–476.