Chinatsu KUROIWA
Mei GAN
Anaëlle PIERREDON

Rapport du projet de classification de tweets
Groupe CMA

Résumé

Notre projet a pour but de réaliser une classification sur un ensemble de tweets à propos du réchauffement climatique en trois catégories. Nous souhaitons réaliser un système dont le résultat d'évaluation atteint au moins 80% de f-mesure. Nous avons donc réalisé plusieurs scripts pythons qui permettent de classifier ces tweets à l'aide du machine learning. Nos meilleurs résultats sont 84% de f-mesure avec la méthode Multinomial Naive Bayes et 89% avec Word2Vec.

Introduction

La tâche réalisée est celle de la classification multiclasse de tweets : soit l'auteur du tweet pense que le réchauffement climatique existe, soit il pense que le réchauffement climatique n'existe pas, soit le tweet ne permet pas de déterminer son avis. Nous avons réalisé cette tâche sur un ensemble de 6090 tweets en anglais, à l'aide de plusieurs scripts pythons. Nous avons dans un premier temps testé trois méthodes de machine learning : Multinomial Naive Bayes, SVM et la Régression Logistique en utilisant la bibliothèque Sklearn. Ensuite, nous avons réalisé une autre version de notre script en utilisant word2vec. Nous avons testé plusieurs représentations et plusieurs méthodes afin de trouver les combinaisons qui marchaient le mieux. Nous verrons donc quelle est la meilleure méthode pour classifier des corpus ayant peu de données et étant constitués de documents courts comme les nôtres. Dans la suite de ce rapport nous présenterons les références utilisées, l'organisation de notre code, nos données et enfin, nos résultats.

Références et travaux similaires

Afin de mieux apprendre et comprendre le classifieur de la méthode de machine learning, nous avons principalement recherché ces trois informations : le principe et l'utilisation du modèle et de la méthode, les projets et les codes d'analyses de sentiments combinés avec le machine learning et les moyens d'optimisation du modèle.

Concernant le principe des méthodes, nous avons trouvé des sites Web avec des exemples simples, tels que l'introduction du Multinomial Naive Bayes[1], pour nous aider à comprendre la présentation de la Régression Logistique[2], la comparaison entre le SVM et la Régression Logistique[3], etc. Pour l'utilisation de la

[1] https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54
[2] https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148
[3] https://medium.com/axum-labs/logistic-regression-vs-support-vector-machines-svm-c335610a3d16

librairie sklearn pour ces méthodes, nous nous référons principalement à la page Web pertinente de sklearn pour les trois méthodes que l'on a testé : Multinomial Naive Bayes[4], le SVM[5] et la Régression Logistique[6]. Sur ce site, les utilisations de ces méthodes et de ces paramètres sont bien expliquées en détails avec des exemples faciles à comprendre.

Ensuite, nous avons recherché de nombreux articles et codes en ligne. Par exemple, nous avons trouvé un mémoire de master[7] qui utilise également le corpus du réchauffement climatique ainsi que trois méthodes pour effectuer des analyses et des recherches comparatives. Ce mémoire nous a été d'une grande aide. L'auteur utilisait une variété de combinaisons pour améliorer la précision. De même, nous avons amélioré la précision du modèle en combinant différents n-grammes. Nous avons également trouvé des classifications similaires à l'analyse des sentiments avec le machine learning en ligne, tels que celui-ci sur la classification des tweets[8] et la classification des questions et réponses[9]. L'article expliquait clairement word2vec, bien que l'auteur n'ait pas obtenu une précision élevée dans ses recherches. Word2Vec reste un bon choix car c'est une méthode efficace pour créer du word-embedding avec une représentation vectorielle de N dimensions. Ayant testé word2vec avec le SVM, nous avons constaté que les résultats n'étaient pas particulièrement bons. Nous avons donc eu l'idée d'essayer d'utiliser word2vec avec LSTM (Long Short-Term Memory). Nous avons recherché beaucoup de sites à propos des usages de LSTM avec word2vec, tels que celui-ci[10] qui analyse des

---

[4] l'utilisation de Multinomial NB avec sklearn
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[5] l'utilisation du SVM avec sklearn
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[6] l'utilisation de la Régression Logistique avec sklearn
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[7] Nithisha Mucha (2017), *Sentiment analysis of global warming using twitter data*, North Dakota State University (mémoire de master qui fait les analyses similaires)
https://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf

[8] la classification de tweets de global warming
https://medium.com/analytics-vidhya/comparative-study-of-sentiment-analysis-techniques-part-1-d2a6c6b49068 et
https://medium.com/analytics-vidhya/comparative-study-of-sentiment-analysis-techniques-part-2-710060d7ba63

[9] la classification des questions et des réponses du site Stack Overflow
https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568

[10] l'analyse des sentiments avec LSTM et word2vec
https://www.kaggle.com/guichristmann/lstm-classification-model-with-word2vec

sentiments avec un corpus de nouvelles d'un journal brésilien en utilisant le LSTM et le word2vec à l'aide de la bibliothèque Keras. Ou encore, ce tutoriel pour les débutants[11] ainsi que des articles qui comparent les méthodes traditionnelles de machine learning et le deep learning[12]. La raison pour laquelle le LSTM fonctionne si bien est qu'il est capable de stocker les informations importantes et de mettre de côté les informations qui ne le sont pas.

Par ailleurs, après avoir formé la structure de base et sélectionné la représentation, nous avons dû ajuster davantage les paramètres pour obtenir un modèle optimal. Les articles mentionnés ci-dessus et les recherches connexes nous ont fourni de nombreux exemples de paramètres utilisés. Par exemple, en modifiant le paramètre alpha de Multinomial Naive Bayes, nous avons pu optimiser notre modèle. Nous établissons la relation entre la précision et ce paramètre alpha pour sélectionner la valeur alpha la plus appropriée. De même, lorsque le modèle LSTM rencontre le problème de l'overfitting, nous pouvons essayer de réduire le nombre d'epochs et d'ajuster la valeur de « dropout » pour atténuer l'overfitting. On a également comparé la performance de word2vec qui a été entraîné par nous-même avec par les modèles pré-entraînés tels que le word2vec modèle de Google news[13], et le word2vec modèle de tweets généré par Frédéric Godin de Université de Ghent [14].

De plus, nous avons également recherché des codes similaires sur github, pour étudier leur structure, afin d'améliorer notre propre code et nous avons résolu pleins de problèmes grâce au site Stack Overflow[15].

Méthode et organisation du code

Nous avons séparé notre code en deux scripts : un pour le prétraitement et le nettoyage, et un pour la construction du modèle et son évaluation. Nous avons réalisé deux scripts finaux de construction de modèle : un pour Multinomial Naive Bayes et un pour Word2vec. Le code de prétraitement et de nettoyage est fonctionnel alors que les scripts d'évaluations sont des codes objets.

---

[11]un tutoriel pour les débutants
https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948

[12] Ayata, Deger & Saraclar, Murat & Ozgur, Arzucan. (2017). BUSEM at SemEval-2017 Task 4A *Sentiment Analysis with Word Embedding and Long Short Term Memory RNN Approaches*. 777-783. 10.18653/v1/S17-2131.https://www.researchgate.net/publication/322794715_BUSEM_at_SemEval-2017_Task_4A_Sentiment_Analysis_with_Word_Embedding_and_Long_Short_Term_Memory_RNN_Approaches

[13] le Word2vec modèle de Google new https://code.google.com/archive/p/word2vec/

[14] word2vec modèle Fréderic Godin de https://github.com/FredericGodin/TwitterEmbeddings

[15] le site de stack Overflow https://stackoverflow.com/

Dans le script pretraitement_nettoyage_final.py, notre fonction de nettoyage permet de retirer les lignes vides et les signes de ponctuations, de mettre les caractères en minuscules, de tokeniser, de supprimer les stopwords et de lemmatiser. La fonction de prétraitement permet quant à elle de supprimer les doublons, de garder uniquement les tweets pour lesquels l'indice de confiance est supérieur à 0,7 et d'avoir le même nombre de tweets dans chaque catégorie (456 tweets) pour équilibrer le corpus. La catégorie "non" est celle qui contient le moins de tweets, et lorsqu'on réduit le corpus aux tweets ayant plus de 0.7 de confiance, il reste 456 tweets dans cette catégorie. C'est la raison pour laquelle nous avons choisi de sélectionner uniquement 456 tweets par catégories. Ces tweets sont sélectionnés au hasard parmi ceux ayant plus de 0.7 de confiance. Le script réalise également la séparation en Train et en Test, avec 80% pour le Train et 20% pour le Test. Il utilise la bibliothèque pandas pour lire le fichier, les expressions régulières pour le nettoyage et NLTK pour le prétraitement.

Avant de créer le script NB_model.py, nous avions écrit le script train_evaluation_v2.py afin de tester les différentes méthodes et de déterminer la méthode et la représentation à conserver pour le script final. Le script train_evaluation_v2.py permettait de créer une représentation du texte en sac de mots et en vecteurs de TF-IDF avec des n-grammes. Selon nos besoins, nous pouvions choisir de créer des unigrammes uniquement, des unigrammes et des bigrammes, seulement des bigrammes… Nous avions ensuite écrit trois fonctions permettant chacune de réaliser une des méthodes. Nous avons appliqué ces méthodes aux deux représentations testées et nous avons ainsi réalisé six tests dont vous pourrez observer les résultats dans la suite de ce rapport. Les résultats des fonctions s'affichent sous la forme de graphiques grâce à matplotlib et d'un tableau récapitulatif des différentes valeurs calculées (rappel, précision et f-mesure). Ce script utilise les bibliothèques sklearn pour la représentation en vecteur de TF-IDF et pour le traitement avec nos méthodes de machine learning et pandas pour lire le fichier.

Le script NB_model.py est en code objet et a été créé à partir du script précédent en conservant la combinaison de représentation du texte et la méthode de machine learning ayant donné les meilleurs résultats : vecteur de TF-IDF et Multinomial Naive Bayes. Ce script donne aussi la possibilité d'analyser une phrase directement donnée par l'utilisateur, il contient donc des fonctions de nettoyage et de prétraitement.

Après avoir écrit la version du script avec Multinomial Naive Bayes et le TF-IDF, nous voulions améliorer la performance de notre modèle. Nous avons d'abord représenté les données dans word2vec, puis testé avec nos trois méthodes. Mais les résultats obtenus n'étaient pas bons, car lorsque nous utilisions word2vec,

nous ne pouvions pas utiliser de n-grammes. La taille de nos données était donc considérablement réduite. Nous avons ensuite essayé une autre méthode plus adaptée au modèle word2vec et nous avons choisi LSTM car il convient mieux à la classification de séquences que les autres modèles. Le LSTM est explicitement conçu pour éviter le problème de dépendance à long terme et il peut se souvenir des informations pendant de longues périodes. Tous les réseaux de neurones récurrents ont la forme d'une chaîne de modules répétitifs de réseaux de neurones. Le LSTM a également cette structure en chaîne, mais ce module répétitif a une structure différente. Au lieu d'avoir une seule couche de réseau neuronal, ce modèle en a quatre, qui interagissent de manière très spéciale.

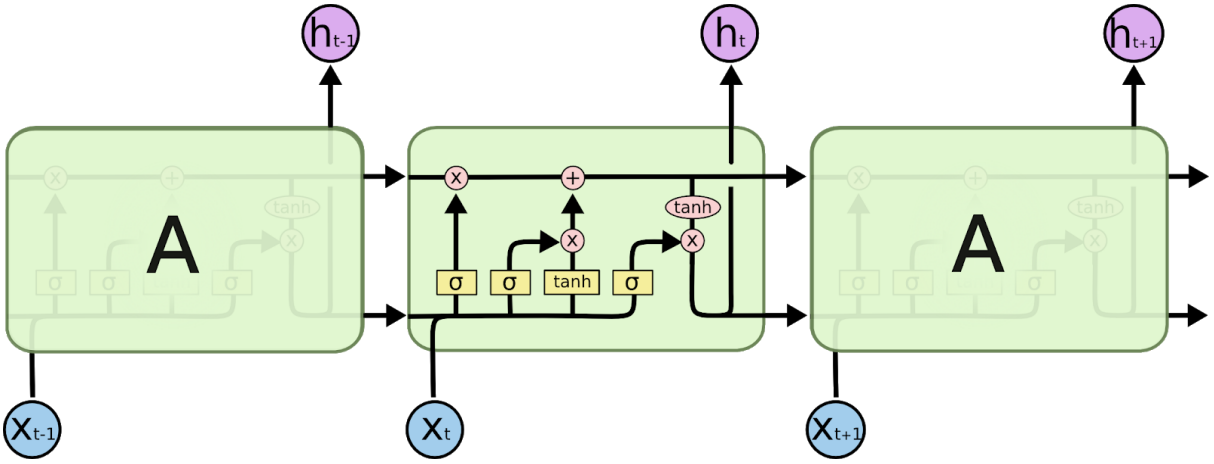


Image représentant le modèle de LSTM et ses quatre couches[16]

Dans le rectangle vert du milieu, la clé des LSTM est la ligne horizontale passant par le haut du diagramme. Et la couche sigmoïde σ génère des nombres compris entre 0 et 1, décrivant la quantité de chaque composant qui doit être laissée passer. Une valeur de 0 signifie « ne laissez rien passer », tandis qu'une valeur de 1 signifie « laissez tout passer! ». Le modèle peut augmenter ou diminuer les informations grâce à un tel traitement.

En pratique, nous avons utilisé la bibliothèque Keras pour la méthode LSTM et la bibliothèque Word2Vec pour le word embedding. Le paramètre «epochs» nous permet d'entraîner les données plusieurs fois. Par rapport aux trois méthodes mentionnées ci-dessus qui ne s'entraînent qu'une seule fois, les résultats obtenus par la formation en boucle multiples seront meilleurs.

```
self.history_lstm = self.lstm_model.fit(Train_X,Train_Y,
                                        batch_size = batch_size,
                                        class_weight=class_weights,
                                        epochs = 13,
                                        verbose=0,
                                        validation_split=0.25)
```

[16] Understanding LSTM Networks
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Image présentant la modification du paramètre epochs

Ensuite en nous référençant aux documents et articles mentionnés au-dessus, nous avons produit un modèle word2vec avec notre corpus sélectionné, puis divisé en 80% de train et 20% de test. Nous avons essayé de traiter la représentation pour word2vec avec différentes moyens. Par exemple, nous avions calculé la moyenne des vecteurs de mots par phrases, et avons dû ajouter un embedding couche. Toutefois, cela n'a pas augmenté la f-mesure. Nous avons donc abandonné ce traitement.

```
model_lstm.add(Embedding(input_dim=vocab_size,
                         output_dim=embedding_size,
                         weights=[w2v_weights],
                         input_length=num_features,
                         mask_zero=True))
                          trainable=False))
```

Image représentant l'utilisation de l'embedding couche

Après avoir cherché les informations sur Internet, nous avons décidé d'essayer d'utiliser un modèle word2vec pré-entraîné. Nous avons choisi le modèle word2vec[17] de tweets généré par Frédéric Godin de Université de Ghent, qui contient 400 millions de tweets et dont la taille des vecteurs est de 400. Avec ce modèle et la modification des paramètres, nous avons réussi à augmenter la performance et nous sommes arrivés à une f-mesure de 89%.

Le script LSTM_word2Vec.py est en code objet et les fichiers train et test doivent être séparés et prétraités avant de passer par ce script. Il donne également la possibilité de charger un modèle déjà entraîné et d'analyser une phrase directement donnée par l'utilisateur.

Description de nos données

Notre corpus est un ensemble de tweets en anglais sur le réchauffement climatique. Il est composé de 6090 tweets comportant au total environ 95 000 mots avant le nettoyage. C'est fichier csv avec un tweet par ligne.

Il est séparé en 3 colonnes : le tweet, la catégorie et l'indice de confiance. Il y a 3 catégories possibles : oui (3111 tweets), non (1114 tweets) et on ne sait pas (1862 tweets). Dans l'ensemble de données, la catégorie "on ne sait pas" est présentée de différentes manières : soit "N/A", soit "NA". Avant de réaliser les différents prétraitements, nous avons donc unifié ces catégories : "Y" pour la catégorie "oui", "N" pour la catégorie "non" et "A" pour la catégorie "on ne sait pas" afin de pouvoir faciliter les traitements de données.

[17] la présentation du word2vec modèle https://fredericgodin.com/research/twitter-word-embeddings/

L'indice de confiance permet de connaître la fiabilité du classement de chaque tweet dans une catégorie. Si l'indice de confiance est égal à 1 alors la fiabilité est élevée et on considère que le tweet est correctement classé. En revanche, la fiabilité baisse lorsqu'on s'approche de 0 et il y a donc plus de chance qu'il y ait des erreurs sur le choix de la catégorie de ces tweets. Les indices de confiance se trouvent principalement entre 0,6 et 0,7.

Les spécificités de ce corpus sont que les textes sont très courts (280 caractères maximum), et qu'ils contiennent des hashtags et des retweets. Nos tweets contiennent entre 6 et 161 caractères. Les mots les plus courants sont "warming" (2012 occurrences), "change" (1878 occurrences) et "climate" (1876 occurrences).

La méthode Multinomial Naive Bayes est optimisée pour nos données car nous avons pu lire dans des travaux similaires qu'elle était efficace sur des documents courts et avec peu de données d'apprentissage. Dans le mémoire de master que l'on a utilisé comme référence car il traite de la classification de données twitter à propos du réchauffement climatique, Multinomial Naive Bayes et SVM ont également été utilisées.

Description des résultats

Nous avons réalisé 10 combinaisons différentes de méthodes et de prétraitements. Nous avons divisé le corpus après avoir réalisé ces prétraitements en 80% pour le corpus train et 20% pour le corpus test.

Description des trois prétraitements différents :

A : Transformation en minuscule, tokenization, stopwords, lemmatisation, suppression des caractères inutiles et des signes de ponctuations

B : Transformation en minuscule, tokenization, stopwords, lemmatisation, suppression des caractères inutiles et des signes de ponctuations. Sélection des tweets ayant plus de 0.7 d'indice de confiance, équilibrage de la proportion entre les différentes catégories (Yes, No et on ne sait pas) à 456 tweets.

C : Suppression des signes de ponctuation

		A	B	C
1	Multinomial Naive Bayes + BOW	62%	81%	X

2	SVM + BOW	60%	82%	X
3	Régression Logistique +BOW	62%	82%	X
4	Multinomial Naive Bayes + TF-IDF + Unigramme	62%	81%	X
5	SVM+ TF-IDF + Unigramme	61%	82%	X
6	Régression Logistique + TF-IDF + Unigramme	62%	83%	X
7	Multinomial Naive Bayes + TF-IDF + Unigramme + Bigramme	62%	**84%**	X
8	SVM + TF-IDF + Unigramme + Bigramme	**63%**	83%	X
9	Régression Logistique + TF-IDF + Unigramme + Bigramme	**63%**	82%	X
10	word2vec + LSTM	X	X	**89%**

Tableau représentant la f-mesure obtenue en fonction des différents traitements

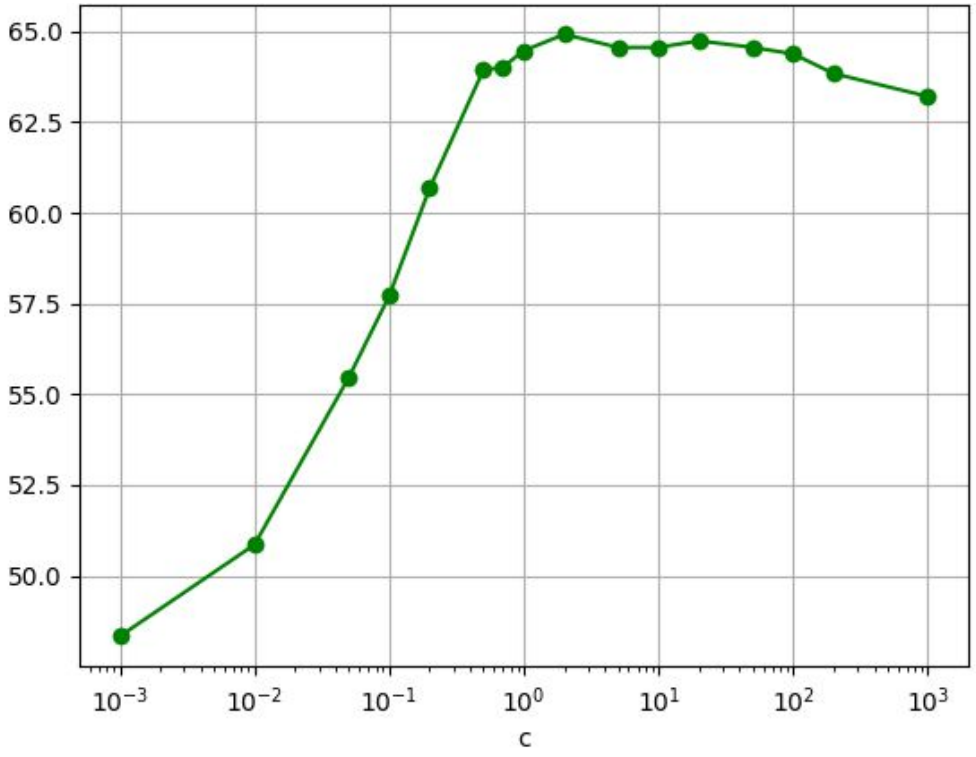
Pour calculer nos résultats nous avons utilisé la bibliothèque Sklearn qui nous permet de calculer la précision, le rappel et la f-mesure. La valeur que nous avons utilisé comme référence pour évaluer notre modèle est celle de la f-mesure.

Discussion

Pour notre projet, nous avons testé quatre méthodes de machine learning : Naive Bayes, SVM, Régression logistique et word2vec. Nous présenterons ci-dessous les étapes d'amélioration de notre système de classification.

Premièrement, nous avons préparé les corpus de test et de train qui ont été prétraités avec la transformation en minuscule, la tokenization, les stopwords, la

lemmatisation et la suppression des caractères inutiles et des signes de ponctuations (correspondant à la colonne A dans la partie précédente). Ils ont été divisés en 80% et 20%. Nous avons calculé la F-mesure avec les méthodes présentées dans les lignes 1 à 6 du tableau ci-dessus. Cependant nous n'avons pas eu de résultats satisfaisants. Nous avons donc ajouté les bigrammes à chaque méthode car nous avons lu que certaines combinaisons avec des unigrammes et des bigrammes donnaient de meilleurs résultats que les unigrammes seuls. Les résultats sont dans le tableau ci-dessus et correspondent aux lignes 7 à 9. Néanmoins, ils n'ont pas été améliorés comme prévu.



Graphique représentant la précision en fonction de la valeur du paramètre de régularisation pour le modèle ayant utilisé la Régression Logistique avec un vecteur TF-IDF unigramme et bigramme.

Nous avons ensuite commencé à réfléchir à comment nous pourrions améliorer les corpus de test et de train avec un prétraitement différent. Dans notre corpus de tweets, il y a des chiffres qui montrent l'indice de confiance du choix de catégorisation (Yes/No/Je ne sais pas). Nous avons remarqué que cela pourrait nous servir à améliorer notre corpus de test et de train. Nous avons donc calculé les moyennes de chacun des trois choix. Nous avons obtenu les résultats suivants :

```
Yes : 0.8419560143626571 et N/A : 0.7682225026852841 et No : 0.8503360655737706
```

Résultat des moyennes de l'indice de confiance des trois catégories

Chinatsu KUROIWA
Mei GAN
Anaëlle PIERREDON

Nous avons donc fait l'hypothèse suivante après avoir obtenu ces résultats :

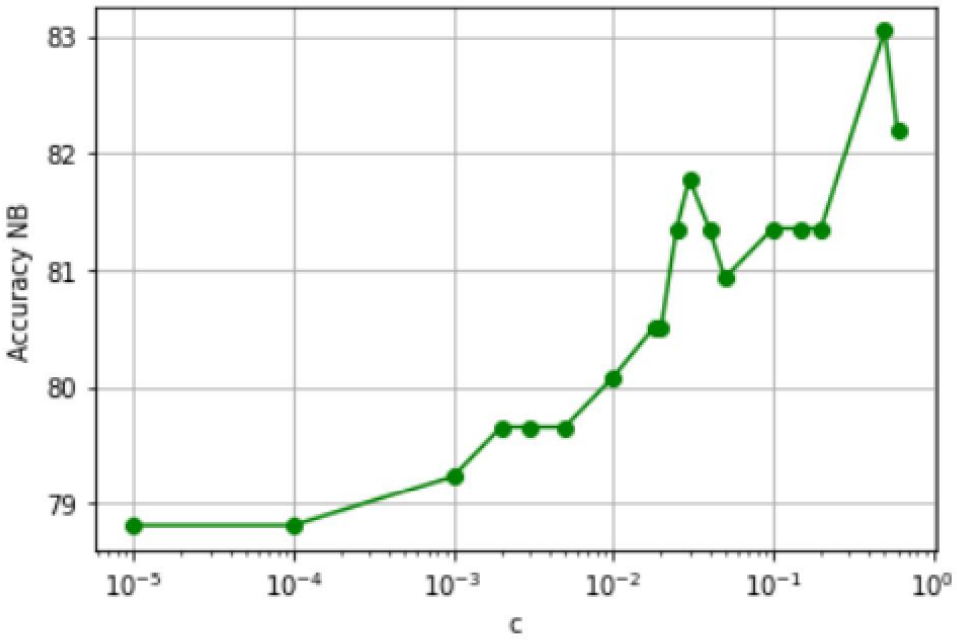
Modèle / Corpus	Bon modèle	Mauvais modèle
Bon corpus > 0.7 fidélité	**f-mesure ↑** **tout va bien**	**f-mesure ↓** Notre modèle est mauvais
Mauvais corpus tout le score de la fidélité	**f-mesure ↓** Le corpus annoté par l'humain n'est pas bien.	**f-mesure ↑** Ce n'est pas bien car le modèle fait la même classification sur le corpus n'est pas bien annoté. C'est-à-dire notre modèle est mauvais. **f-mesure ↓** On ne peut pas dire 100% que notre modèle ne marche pas car le corpus n'est pas bien. Il est possible que f-mesure baisse car le corpus n'est pas bien annoté. Ou on peut dire aussi c'est parce que notre modèle est mauvais. Mais on ne sait pas lequel.

Tableau de nos hypothèses selon le choix de l'indice de confiance par catégories

De plus, nous avons remarqué que la proportion de chaque tweet n'est pas équilibrée : oui (3111 tweets), non (1114 tweets) ou ne sait pas (1862 tweets). Nous avons supposé que cela était un biais de la sélection de données. Nous avons donc calculé le nombre de tweets ayant plus de 0,7 d'indice de confiance pour chaque catégorie. Suite à cela, nous avons décidé de ne prendre que 456 tweets car c'était le chiffre plus bas entre les trois.

Nous avons ajouté les conditions suivantes dans le script de prétraitement :

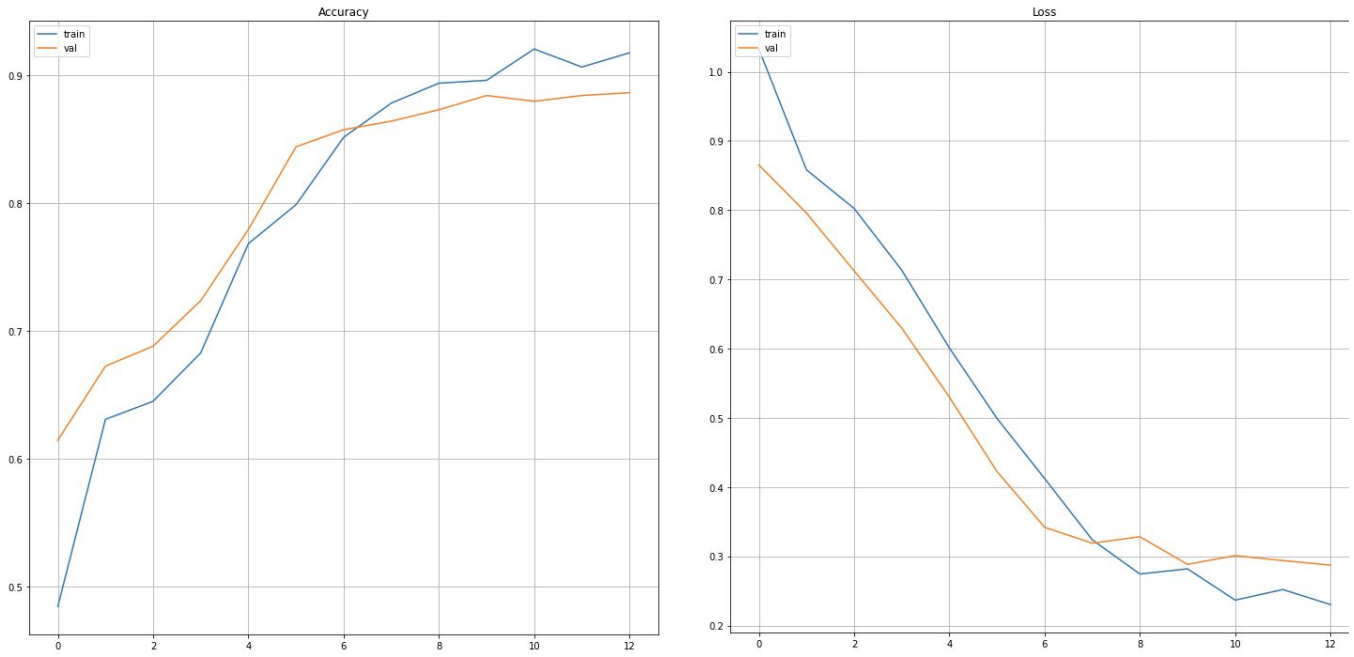
1. Nous n'allons prendre que les tweets qui ont plus de 0,7 d'indice de confiance.
2. Nous n'allons prendre que 456 tweets par catégories. Pour les catégories qui ont plus de 456 tweets, nous allons sélectionner les tweets à utiliser au hasard.



Graphique représentant la précision en fonction du paramètre de régularisation pour le modèle ayant utilisé la méthode Multinomial Naive Bayes avec un vecteur TF-IDF unigramme et bigramme

Grâce à cela nous avons réussi à dépasser les 80% de f-mesure avec toutes les combinaisons possibles de nos méthodes. Comme nous avons choisi 456 tweets au hasard, nous avons testé plusieurs fois et nous avons gardé les modèles de test et de train qui ont donné les meilleurs résultats pour chaque méthode. Les résultats sont dans le tableau au-dessus dans la partie “Description des résultats” à la colonne B.

Enfin, Nous avons implémenté le word2vec qui est un système plus évolué capable de mieux apprendre le vocabulaire propre au domaine avec LSTM afin d'améliorer notre système. En effet, nous avons d’abord implémenté le word2vec avec Multinomial Naive Bayes et aussi SVM qui ont donné les meilleurs résultats avec les corpus A et B. Néanmoins, ils ont donné des résultats inférieurs à 80% de f-mesure. Nous avons finalement décidé de l'implémenter avec LSTM qui est présenté dans une référence que nous avons utilisé. Pour le prétraitement, nous avons simplement supprimé les signes de ponctuations et nous n’avons sélectionné que les tweets qui ont plus de 0,8 d’indice de confiance. Nous ainsi avons obtenu le meilleur résultat parmi toutes les méthodes implémentés (89% de f-mesure). Le résultat est présenté dans le tableau de la partie “Description des résultats” à la colonne C.



Graphiques représentants la précision et la perte en fonction du nombre d'epochs pour le modèle ayant utilisé Word2vec et LSTM.

Conclusion

La méthode qui a donné le meilleur résultat dans l'ensemble de notre projet est le word2vec avec LSTM. La F-mesure est 89%. C'est un résultat qui était attendu mais en même temps c'était assez curieux qu'il ait donné des résultats moins bons que les autres méthodes selon les combinaisons testées.

Concernant l'hypothèse selon laquelle nous pensions avoir le meilleur résultat avec la méthode Naive Bayes Multinomial, nous avons pu le démonter avec les résultats du corpus B. Il a eu 84% de f-mesure (Ligne 7 de la colonne B sur le tableau de la partie de description des résultats.)

Dans l'ensemble du projet, le point qui était le plus difficile pour nous était de réussir à aller au delà des 80% de f-mesure. Nous sommes restées assez longtemps entre 65% et 78% en testant les différentes combinaisons de méthodes. Enfin, nous avons constaté que c'était à cause des corpus test et train qui n'étaient pas de bonne qualité. Pendant ces essais, nous avons pu comprendre que l'équilibre des données et les choix de combinaisons de méthodes ont des impacts importants pour ce travail.

Nous avons pu créer des modèle qui donnent 89% f-mesure sur notre corpus test et train. En revanche, nous pensons également qu'il est possible que notre modèle ne marche pas sur des tweets inconnus qui seraient ambiguës mêmes pour les humains, car nous n'avons sélectionné que les tweets qui ont plus de 0,7 en

indice de confiance. Cependant, avec les données de tweets que nous avons choisi, nous étions sûres que c'était le meilleur choix de prendre uniquement des tweets ayant un indice de confiance élevé. Si nous sommes amenées à refaire ce type de tâche, nous essayerons de collecter des données plus stables et d'en récupérer une plus grande quantité afin d'avoir plus de données variées pour le corpus de test.