UNIVERSITÉ DE SAAD DAHLEB – 1 –

FACULTÉ DES SCIENCES

DEPARTEMENT D'INFORMATIQUE

SPÉCIALITÉ : INGÉNERIE DES LOGICIELS

MEMOIRE DE MASTER

Analyse sémantique des publications dans les réseaux sociaux par apprentissage profond

---

Réalisé par

Nouas Sihem

Devant le jury composé de :

Mme.HADJ HENNI Malika	Université de Blida	*Président*
Mme.BOUTOUMI Bachira	Université de Blida	*Examinateur*
Mme.OUKID Lamia	Université de Blida	*Promoteur*
M.CHEMCHEM Med Amine	Université de Blida	*Co-Promoteur*

Soutenu le 17/07/2019, Blida

UNIVERSITÉ DE SAAD DAHLEB – 1 –

FACULTÉ DES SCIENCES

DEPARTEMENT D'INFORMATIQUE

SPÉCIALITÉ : INGÉNERIE DES LOGICIELS

MEMOIRE DE MASTER

Analyse sémantique des publications dans les réseaux sociaux par apprentissage profond

---

Réalisé par

Nouas Sihem

Devant le jury composé de :

Mme.HADJ HENNI Malika	Université de Blida	*Président*
Mme.BOUTOUMI Bachira	Université de Blida	*Examinateur*
Mme.OUKID Lamia	Université de Blida	*Promoteur*
M.CHEMCHEM Med Amine	Université de Blida	*Co-Promoteur*

Soutenu le 17/07/2019, Blida

RESUME

Avec la croissance explosive des données sur les réseaux sociaux, la fouille sémantique des statuts est devenue inéluctable pour la prise de decision. Dans ce projet, notre objectif est d'analyser les publications sur les réseaux sociaux en s'inspirant des techniques de classification automatique de données textuelles. Ceci, afin de pouvoir filtrer les messages comme ceux à propos racistes et violents, qui consiste souvent une minorité. De ce fait, cette analyse doit notamment prendre en considération les corpus déséquilibrés.

Pour atteindre notre objectif, tout d'abord, différents algorithmes de classification sont testés afin de comparer leurs performances. Nous proposons une analyse des tweets à travers une combinaison de deux algorithmes d'apprentissage automatique CNN et LSTM. De plus pour répondre au problème des corpus déséquilibrés, nous proposons une technique de sur-échantillonnage appelée « *Semantic-Oversampling* ». Les expériences menées sur les solutions proposées ont achevés des résultats satisfaisants.

Mots clés : Classification supervisé, Classification non supervisé, Corpus de données déséquilibré, Oversampling, Réseaux sociaux, Réseaux de neurones profond, LSTM, Twitter.

ABSTRACT

With the explosive growth of data on social networks, the semantic analysis of statutes has become unavoidable for decision-making. In this project, our goal is to analyze publications on social networks by using techniques of automatic classification of textual data. And this, in order to be able to filter messages like those about racism and violence, which is often a minority. As a result, this analysis must take into account the unbalanced corpora.

To achieve our goal, firstly, different classification algorithms are tested to compare their performance. We propose an analysis of tweets through a combination of two machine learning algorithms CNN and LSTM. Moreover, to answer the problem of unbalanced corpora, we propose a technique of oversampling called "Semantic-Oversampling". The experiments conducted on the proposed solutions have achieved satisfactory results.

Keywords: Supervised Classification, Unsupervised Classification, Unbalanced Data Corpus, Oversampling, Social Networks, Deep neural networks, LSTM, Twitter.

ملخص

مع النمو الهائل للبيانات على الشبكات الاجتماعية ، أصبح الحفر الدلالي للقوانين أمرًا لا مفر منه لاتخاذ القرارات. في هذا المشروع ، هدفنا هو تحليل المنشورات على الشبكات الاجتماعية من خلال الاعتماد على تقنيات التصنيف التلقائي للبيانات النصية. هذا ، من أجل أن تكون قادرة على تصفية رسائل مثل تلك المتعلقة بالعنصرية والعنيفة ، التي غالبا ما تكون أقلية ، نتيجة لذلك ، يجب أن يأخذ هذا التحليل في الاعتبار البيانات غير المتوازنة

لتحقيق هدفنا ، أولاً ، يتم اختبار خوارزميات تصنيف مختلفة لمقارنة أدائها. نقترح تحليلًا للتغريدات من خلال مجموعة من خوارزميتي تعلم الآلة CNN و LSTM. علاوة على ذلك ، للإجابة على مشكلة الشركات غير المتوازنة ، فإننا نقترح أسلوبًا للإفراط في الاختراق يطلق عليه "الدلالة الفوقية". حققت التجارب التي أجريت على الحلول المقترحة نتائج مرضية.

الكلمات الرئيسية التصنيف الخاضع للإشراف، التصنيف غير الخاضع للإشراف، مجموعة البيانات غير المتوازنة، الإفراط، الشبكات الاجتماعية، الشبكات العصبية العميقة، ل س ت م، تويتر.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES FORMULES

INTRODUCTION GENERALE

Depuis les deux dernières décennies, les réseaux sociaux prennent de plus en plus de place dans la vie habituelle de beaucoup d'individus. En effet, l'utilisation quotidienne des réseaux comme Facebook, Twitter ou encore Instagram ne cesse d'augmenter. Tous ces réseaux sociaux amassent de très nombreuses données, des messages et des images, sont soigneusement enregistrées. C'est ainsi que se pose le problème de la fouille et l'exploitation de cette masse d'informations. Parmi les réseaux sociaux les plus célèbre on retrouve Twitter, un réseau social de microblogage géré par l'entreprise Twitter Inc, crée en mars 2006 avec un slogan initial « *What are you doing?* » (Qu'est-ce que vous faites ?) , et remplacé par « *Compose new Tweet…* » (Composer un nouveau tweet) dans la dernière version de septembre 2011. Ce service permet aux internautes d'envoyer gratuitement de brefs messages « tweets » limités à 280 caractères. Twitter compte 330 millions d'utilisateurs actifs chaque mois, chaque seconde environ 5 900 tweets sont expédiés, cela représente 504 millions de tweets par jour ou 184 milliards par an dans différents langues. Cette masse d'information vient alimenter le flot d'informations « Big data». Twitter est donc devenu une cible de recherche et de collecte d'informations pour beaucoup d'entreprises, et notamment pour des chercheurs qui visent à étudier le comportement humain, les dernières tendances, l'analyse des opinions, l'analyse du sentiments et surtout le filtrage des propos violents, terroriste, racistes, …etc.

Ces dernières années, les approches d'apprentissage profond commencent a démontré leur efficacité d'analyse sur différents corpus de données, notamment pour l'analyse des publications dans les réseaux sociaux visant à extraire des connaissances. Lorsque nous traitons des données réelles, dans la plus part du temps, les données sont très déséquilibrées, ce qui rend l'apprentissage des algorithmes très biaisé vers la classe majoritaire, et c'est le cas dans plusieurs domaines comme la bio-informatique, détection de fraud, classification du texte …etc. Ce problème est notamment present dans Twitter, les tweets contenant des propos racistes ou violents représentent généralement une minorité, et souvent, les algorithmes d'apprentissage n'arrivent pas à les détectés.

Notre objectif dans ce travail est de proposer des techniques d'analyse sémantique des publications dans les réseaux sociaux par apprentissage profond, et ceci en se basant sur les méthodes de classification supervisées et non supervisés. Dans ce cas de travail, nous utilisons des corpus de tweets de langue anglaise. Nous exposons une étude comparative entre les différentes méthodes existantes pour la fouille des données. A travers cette étude nous avons proposé une combinaison entre les deux couches de réseaux de neurones CNN et LSTM. Nous avons proposé notamment une méthode de sur-échantillonnage appelé *Semantic-Oversampling*. Nous avons mené une étude expérimentale pour comparer nos approches avec les solutions existantes selon plusieurs mesures d'évaluation.

Dans un premier temps, le premier chapitre présentera les travaux connexes à la problématique. Puis dans un deuxième chapitre, des méthodes de prétraitement et de représentation de texte sont introduites, ainsi que des algorithmes détaillés de classification supervisée (les réseaux de neurones) et de classification non supervisé (de Topic Modeling/Clustering). Un troisième chapitre est introduit pour présenter les techniques de *sampling* pour le traitement des corpus déséquilibrées et les mesures d'évaluations utilisées. Puis un quatrième chapitre exposera nos contributions algorithmiques qui concernent l'apprentissage supervisé et le sur-échantillonnage des données textuelles. Enfin, dans le cinquième chapitre, une étude comparative des différentes approches proposées est réalisée.

CHAPITRE 1 : ETAT DE L'ART

INTRODUCTION

Dans ce chapitre, les différents acquis et travaux en cours sur la problématique sont introduits. Dans un premier temps, on s'intéresse aux différentes techniques de prétraitement sur les données textuelles proposées dans la littérature. Ainsi qu'aux méthodes de représentation du texte et de mesures de similarité sémantique. Par la suite on présente les travaux liés à la classification supervisée et non supervisée utilisées pour l'analyse du texte. On présente notamment les solutions existantes pour l'échantillonnage des corpus déséquilibrés et les mesures d'évaluations qui sont généralement utilisées.

1.1 Prétraitement du texte

Pour tout type de données, avant de procéder à une analyse quelconque, il est nécessaire de faire un prétraitement ou encore une normalisation. Dans ce cas d'étude on doit procéder à un filtrage de tweets pour ne laisser que ceux qui présentent des informations pertinentes. Des techniques intuitives sont évoquées dans plusieurs travaux de classification des tweets comme dans [Gao et al., 2014 ; Lee et al., 2011] ou il s'agit de supprimer les liens, les signes, les nombres, les noms d'utilisateur ainsi que les stop words appelés aussi mot vides. Seulement Gao prévient que tout ce nettoyage résulte en une diminution de 45.85% du vecteur caractéristique d'un tweet et qu'il faut procéder avec attention. Des techniques pour la réduction du bruit et de normalisation des vecteurs caractéristiques ont été proposées afin d'améliorer les performances de classification automatique des nouvelles extraites de Twitter dans [Beverungen & Kalita, 2011].

Si l'unité linguistique choisie est le mot, il faut remarquer que plusieurs mots ont des sens communs ou forment simplement une autre forme de conjugaison, de ce fait, une technique appelé *Stemming* doit être appliqué pour l'analyse du texte [Kantrowitz et al., 2000]. Al-Khafaji explique que c'est une étape importante qui permet de joindre les mots d'une même racine et ainsi garder le même contexte [Al-Khafaji, 2017].

1.2 Représentation du texte

Les données usuellement exploitées en data mining se présentent sous forme de tableaux attributs-valeurs numériques. Par contre les données dans le text-mining se présentent sous leur forme brute c'est à dire du texte. Afin de faire de l'analyse, un premier challenge serait de trouver une représentation numérique de façon à ce que l'on puisse effectuer un traitement analytique sans perdre trop d'information. Chollet, (2018) explique que le texte peut être compris comme une séquence de caractères ou bien comme une séquence de mots, mais c'est plus commun de donner des représentations vectorielles au niveau des mots.

Dans sa version la plus simple, un document particulier est représenté par l'histogramme des occurrences des mots le composant. Une première référence est trouvée dans les travaux de Zellig [Zellig, 1954].

Une méthode basée sur l'occurrence des mots dans un document tout en prenant compte des autres documents est TF-IDF *Term Frequency-Inverse Document Frequency* [Salton & McGill, 1983]. Le poids TF augmente proportionnellement au nombre d'occurrences du mot dans le document, et varie aussi en fonction de la fréquence du mot dans le corpus IDF. La représentation TF-IDF a déjà été appliquée sur des tweets dans [Lee et al., 2011].

Le *Word Embedding* « plongement de mots en français » est une méthode d'apprentissage d'une représentation de mots utilisée notamment en traitement automatique des langues et rendu possible grâce à l'hypothèse distributionnelle *distributional hypothesis*. Cette représentation permet aux mots apparaissant dans des contextes similaires de posséder des vecteurs relativement proches (en termes de distance). Une implémentation populaire du calcul des plongements de mots serait Word2vec [Mikolov et al., 2013] ou encore Glove *Global Vectors for Word Representation* [Pennington et al., 2014].

L'utilisation du *Word Embedding* est notamment appliquée sur des tweets avec la méthode word2vec dans [Joshua et al., 2017], et avec la méthode Glove (*Global Vectors for Word Representation*) dans [Dasgupta et al., 2016].

LSI (*Latent Semantic Indexing*) permet de représenter les termes par des vecteurs qui sont une indication du profil d'importance du terme dans les documents. Cette propriété peut donc être utilisée pour établir une notion de similarité entre termes, ou représenter un document comme la moyenne des vecteurs représentant les termes qu'il contient. LSI a été utilisé dans plusieurs travaux de recherche comme : [Hull ,1994], [Schutze, 1998], [Weigend, Wiener, & Pedersen, 1999] et [Steinberger & Jezek, 2004].

1.3 Mesures de similarités

Une première approche serait de transformer la phrase en vecteur grâce à un outil comme word2vec et ensuite calculer la distance entre les deux vecteurs comme proposé dans [Kusner et al. 2015].

Afin de mesurer la similarité sémantique entre deux phrases, Sravanthi et Srinivasu proposent une méthode qui consiste d'abord à comparer les deux phrases à un niveau syntaxique, ensuite à un niveau de *synset* (synonymes), après à un niveau sémantique où on considère la définition des mots [Sravanthi & Srinivasu, 2017].

Dans [Lesk, 1986], l'auteur propose un algorithme qui est basé sur le fait que les mots dans un même voisinage vont partager le même concept. Cet algorithme peut être appliqué dans *Wordnet*, une base de données lexicale qui répertorie du contenu sémantique et lexical de la langue anglaise. Une autre version de Lesk a été adapté [Banerjee & Pederson, 2002]. La lacune de cet algorithme est justement le voisinage, si on a un voisinage riche ça serait plus facile de déterminer le concept en d'autres termes l'absence de certain mots peut radicalement changer le résultat.

1.4 Approches supervisées

Bien que les réseaux neuronaux convolutifs (CNN : *Convolutional Neural Network)* aient beaucoup de succès dans le traitement d'images ou la reconnaissance faciale, les CNNs sont

maintenant utilisés pour le traitement de langage. Comme dans [Kalchbrenner et al., 2014], les auteurs adaptent un modèle basé sur une architecture CNN pour la classification du texte court.

Dans [Yoon, 2014], l'auteur utilise un réseau neuronal convolutif (CNN) multicanal pour la classification des phrases, l'auteur utilise des vecteurs pré-entrainés de mots avec word2vec, ces vecteurs ont été entraînés sur 100 billions de mots grâce à Google News, l'idée est de capturer des caractéristiques génériques. Plusieurs variantes du model sont testées : CNN-rand ou l'initialisation est aléatoire ensuite modifiée durant l'apprentissage, CNN-static ou les vecteurs pré-entrainé word2vec sont utilisés sans modification dans l'entrainement, CNN_nonstatic ou la modification est activé et enfin un CNN_ multicanal ou les mots sont représentés grâce à deux vecteurs l'un étant statique et l'autre dynamique. Ces modèles sont régularisés aussi avec un drop-out de 2% à 4% (une technique pour éviter le sur-apprentissage). Les résultats mettent en évidence qu'une représentation word2vec pré-entrainé donne de meilleurs résultats.

Les réseaux de neurones récurrents (RNN) sont aussi appliqués pour des taches supervisées comme dans [Lai et al., 2015] pour la classification du texte. Du et Huang proposent un RNN bidirectionnel et l'ont appliqué sur des dataset de langue chinoise et anglaise. Le model a surpassé les algorithmes classiques comme LDA mais aussi d'autres types de RNN, le LSTM (*Long short-term memory)* et le GRU (*gated recurrent units)*.

Dans [Iyyer et al., 2015], les auteurs ont introduit un CNN qui prend la moyenne des Embeddings des mots par plusieurs couches avant d'effectuer la classification du texte. Dans [Zhou & al., 2016], les auteurs proposent d'intégrer un LSTM bi directionnel, leurs deux modèles BLSTM-2DPo et BLSTMCNN peuvent retenir des dépendances dans le temps mais aussi capturer des informations sur les caractéristiques des vecteurs.

Pour la classification des tweets, dans [Liu et al., 2018], les auteurs utilisent LSTM afin d'extraire des informations relatives à l'influenza, le F-score de LSTM surpasse celui de SVM (*Support Vector Machine)* et SVM avec SMOTE (*Synthetic Minority Oversampling)*.

Beaucoup de recherches pour la classification des tweets ont été réalisées comme dans [Liu, 2012] pour l'analyse du sentiment, [Al-Ayyoub et al., 2018] pour l'analyse des tweets en arabe ou encore dans [Sriram et al., 2010], ou les auteurs proposent une classification du texte court en utilisant le model bag of word BOW avec Naïve Bayes et decision trees.

Reed et Marks, (1999) adressent la question du choix du nombre de couches dans les réseaux de neurones, et expliquent qu'une seule couche est parfois suffisante pour approximer la plupart des fonctions, seulement, il existe des cas pour lesquelles une solution à une seule couche cachée est très inefficace par rapport à des solutions comportant plusieurs couches [Reed & Marks, 1999, p. 37].

Dans [Goodfellow et al., 2016, p. 201], les auteurs déclarent qu'empiriquement, les réseaux de neurones profonds (avec plusieurs couches cachées) semblent aboutir à de meilleurs résultats, et ce, pour différentes taches. Cela suggère que l'utilisation d'architectures profondes exprime bien un préalable utile sur l'espace des fonctions apprises par les modèles.

1.5 Classification non supervisée (Clustering)

K-means [MacQueen, 1967] a été adapté dans différents corpus qui incluent les données numériques, catégoriques et binaires. Dans le contexte de twitter on trouve le travail de Friedemann, (2015) qui utilise K-means avec une mesure de similarité pour faire un clustering des clients d'une compagnie. Le nombre de topic k est choisi en se basant sur le calcul du coefficient de la silhouette maximal. Par la suite chaque cluster a été manuellement étiqueté. Cette technique a donnée des résultats satisfaisants.

Dans [Zhao, 2011], l'auteur utilise K-means et K-Medoids pour le clustering des tweets avec une distance euclidienne et la distance Manhattan sur un dataset étiqueté. L'auteur initialise le nombre de K selon le nombre des classes pour examiner l'homogénéité des clusters, en prenant les tops mots fréquents dans chaque cluster. Les mots étaient bien distincts ainsi l'auteur déduit que les clusters obtenus sont bien harmonieux. D'un autre coté dans [Antenucci et al, 2011], les auteurs présentent un algorithme pour apprendre les relations entre le contenu d'un tweet et les types de hashtags qui pourraient décrire avec précision ce contenu. Chaque tweet est représenté par l'occurrence de ses mots (*bag of words*), et ses catégories sont les hashtags utilisés dans le

tweet. Les auteurs utilisent des algorithmes de clustering et de réduction de dimensionnalité pour effectuer une classification supervisée.

Dans [Popovici et al,.2014], les auteurs proposent une méthode de clustering des tweets en ligne, pour identifier un cluster, le tweet avec le meilleur degré de pertinence sémantique c'est à dire avec une similarité maximale dans un cluster est sélectionné comme topic.

Pour trouver le nombre de topic K dans K-means, grâce à LDA *Latent Dirichlet Allocation*, Guan cherche à trouver les topics typiques en calculant le support du document et le comparant à un seuil, le résultat donne des topics avec un sens sémantique clair sur le dataset 20 Newsgroups data. [Guan, 2016].

Dans le but de détecter des évènements en se basant sur les hashtags, dans [Boom et al., 2015], les auteurs utilisent l'algorithme DBSCAN (Density-Based-Special Clustering of Application with noise).

Pour l'amélioration de prédiction du sentiment de tweets via un model « cluster then predict », les auteurs dans [Soni & Mathai, 2015], proposent d'appliquer K-means ensuite deux autres algorithmes, arbre de decision et SVM *Support Vector Machine*, le model proposé a pu améliorer les résultats.

1.6 Topic modeling

Dans [Alvarez-Melis & Saveski, 2016], les auteurs utilisent LDA (*Latent Dirichlet Allocation)* pour l'analyse des tweets et proposent de faire une agrégation de tweets par conversation, et ceci, afin de gagner dans la longueur, un document serait donc tous les tweets réponses d'un tweet. Une autre méthode serait de combiner les tweets avec le même hashtag ou le nom d'utilisateur [Mehrotra et al., 2013], mais Alvarez-Melis et Savesk expliquent que les documents résultants ne seront pas homogènes, assumer que les topics seront cohérent pour chaque utilisateur et que les hashtags vont permettre de regrouper les tweets qui parlent du même sujet n'est pas toujours vérifié, car souvent, les utilisateurs ne respectent pas l'utilisation des hashtags.

La grande contribution dans [Nguyen et al.,2015], fut d'introduire deux nouveaux « *latent feature topic models* » nommé (LF-LDA et LF-DMM) qui intègrent LDA et DMM (*Dirichlet Multinomial Mixture*) un modèle qui suppose non pas plusieurs topics par document comme dans un LDA de base, mais un seul. Les deux algorithmes ont été comparés aux valeurs de base de LDA, et ont réalisé des performances supérieures.

Dans [Shi et al., 2018], les auteurs proposent un SeaNMF *Semantics-assisted Non-negative Matrix Factorization* pour le topic modeling du texte court, en utilisant une matrice de factorisation non négative sémantique, le model retient la relation entre les mots et les concepts grâce au model *skip-gram* de Word2vec, les sujets découverts par le modèle SeaNMF ont moins de mots bruyants et les mots clés les plus utilisés sont plus corrélés. Les résultats d'analyse sémantique ont démontrés que le modèle SeaNMF peut découvrir des sujets pertinents et cohérents pour des textes courts.

Afin de détecter l'intensité de criminalité dans des lieux et emprunter un chemin sûr, dans [Sharma et al., 2015], les auteurs proposent un modèle basé sur LDA et la classification naïve bayésienne. D'un autre coté dans [Morchid et al., 2015], les auteurs utilisent LDA afin de faire un clustering des tweets pour analyser l'évènement "JeSuisCharlie" dans différents pays. L'approche cherche à mapper un tweet dans un environnement de haut niveau de représentation,

et ceci, en utilisant le modèle *Author-Topic* (AT) qui prend en compte toutes les informations contenues dans un tweet: le contenu lui-même (mots), l'étiquette (pays) et la relation entre la répartition des mots dans le tweet et son emplacement, considérés comme une relation latente. Une représentation de haut niveau a permis d'obtenir des résultats très prometteurs lors de l'identification du pays.

Un autre travail réalisé dans [Wang et al., 2016], ou les auteurs utilisent LDA sur le corpus PubMed pour identifier la dépression et les substances alcooliques que les adolescents utilisent. LDA à découvert d'autres sujets pertinents, tels que les facteurs de risque de la consommation de substances par les adolescents, et ont démontré l'utilité de la modélisation par sujet en tant qu'outil de recherche pour structurer les collections de documents. Ou encore dans [Daneshvar & Inkpen., 2018], ou les auteurs utilisent LSA (*Latent Semantic Analysis)* et SVM (*Support Vector Machine)* pour identifier le genre des personnes dans Twitter en considérant notamment les N-grams (une sous-séquence de n éléments).

1.7 Technique de sampling et d'évaluation

Diffèrent mesures existe pour évaluer différents caractéristiques du ML (*Machine Learning algorithm*). Dans [Sokolova et al., 2006], les auteurs expliquent que ça dépendra du cas d'utilisation. Parfois c'est utile d'avoir un taux de réussite général, mais dans certaines situations, on s'intéresse plutôt à un taux de réussite spécifique.

Ceci est généralement le cas quand il faut détecter des classes minoritaires. Dans [Somasundaram & Reddy, 2016], les auteurs rappellent que dans la détection d'anomalie, classification d'image, des gènes et des documents, les patterns qui sont intéressants sont souvent inferieur a 1 % du data complet. Dans cette étude pour la détection de credit/débit card fraud, des techniques de sampling sont d'abord utilisées, pour l'évaluation de l'algorithme foret d'arbres décisionnels (*Random Forest Classifier)* les auteurs ont utilisé : l'accuracy, true negative rate (TNR), negative predictive value (NPV) et le F-score proposée par Rijsbergen, (1979).

Le problème du dataset déséquilibré est notamment adressé dans [Kaur & Gosain, 2018], les auteurs expliquent que dans plusieurs scenarios sérieux comme la détection de tumeurs, les algorithmes de machine learning échouent. Les auteurs comparent deux approches, *l'oversampling* avec SMOTE (*Synthetic Minority Oversampling*) [Bowyer et al., 2002], dont le but est de créer des données synthétiques similaires aux k-plus proches voisins d'un ensemble de données aléatoire d'une classe minoritaire et *l'undersampling* avec RUS (*Random Undersampling*). Les auteurs utilisent un dataset qui contient 70% de bruit et observent que *l'oversampling* donne de meilleurs résultats. De la même manière dans [Hacibeyoglu & Ibrahim, 2018], afin de prédire des patients malades, les auteurs trouvent que grâce à un *oversampling* avec SMOTE l'algorithme de classification donne de meilleurs résultats qu'un *undersampling*.

Dans [Ah-Pine & Soriano-Morales, 2016], les auteurs s'intéressent au problème du dataset déséquilibré, et proposent d'effectuer un *oversampling* synthétique SMOTE sur des vecteurs TF-IDF pour la classification binaire du sentiment des tweets en utilisant l'apprentissage par arbre de décision ainsi que la régression logistique. Les auteurs utilisent trois dataset publique « Obama-McCain Debate », « Health Care Reform» et « Imagiweb», pour l'évaluation le F-score, l'accuracy global et l'accuracy moyenne sont utilisés. Toutefois les auteurs s'aperçoivent qu'après un oversmapling, l'accuracy global diminue dans tous les tests, et mettent en évidence que l'accuracy global favorise la classe majoritaire par contre le F-score augmente après un oversampling car l'algorithme a pu détecter la class minoritaire, l'accuracy moyenne a augmenté dans un seul cas.

L*'oversampling* avec SMOTE est notamment proposé dans [Chen et al., 2014] pour l'analyse du sentiment d'un corpus déséquilibré, ou SMOTE est appliqué sur des vecteurs du Word Embedding, et obtient les meilleures performances.

Derczynski, (2016) adresse le problème des classes minoritaires dans le traitement de langage et indique que faire des comparaisons qu'avec l*'accuracy* ne serait pas très intéressant vu que les résultats seront très biaisés, et que le recall, la précision et le F-score sont plus convenables. Pour son étude l'auteur propose un F-score dit complémentaire.

Dans [Río et al., 2014], les auteurs trouvent que la procédure Undersampling serait plus approprié pour les grandes corpus, vu qu'il y aura toujours assez de data pour l'apprentissage du ML. Les méthodes d'undersampling sont beaucoup plus utilisées pour le Big Data comme dans [Imran & Srinivasa, 2018], ou dans [Triguero et al., 2017].

1.8 Synthèse des travaux antérieurs

D'après les articles cités dans ce chapitre :

- Pour la représentation vectorielle du texte les auteurs utilisent généralement une représentation aux niveaux des mots (ou *tokens*). La représentation vectorielle avec Tf-idf est très utilisée pour la modélisation d'un document, et il existe notamment des techniques plus récentes comme le *word embedding,* ce dernier permet de capturer des relations sémantiques entre les mots. Plusieurs auteurs utilisent un *word embedding* pré-entrainé pour apprendre des caractéristiques génériques de la langue, cependant, ceci pourrait ne pas convenir aux statuts sociaux, qui souvent, contiennent des fautes d'orthographes, typologiques, …etc.

- Pour la classification supervisée du texte, les réseaux de neurones sont largement utilisés. Généralement les auteurs utilisent les réseaux de neurones à convolution pour le traitement d'image, cependant récemment, les CNNs sont notamment utilisés pour la classification du texte, et ceci est dû à leurs habilités a détecté des caractéristiques locales importantes. Les auteurs utilisent notamment les réseaux de neurones récurrents comme les LSTMs et les GRUs. Ces réseaux permettent de retenir des dépendances sémantiques temporelles, ce qui convient parfaitement aux données séquentielles comme les données textuelles.

- Les méthodes de *Topic-Modeling* les plus utilisées pour la segmentation des documents sont LSA et LDA. Certains auteurs pensent que ces méthodes ne sont pas appropriées pour le texte court et introduisent d'abord des méthodes de combinaisons (par hashtag, discussion ou auteurs …). D'autres utilisent le *word embedding* pour enrichir ces modèles sémantiquement. Pour le *clustering* du texte les auteurs utilisent plus souvent K-means ou bien DBSCAN.

- Pour l'échantillonnage, les auteurs utilisent beaucoup plus le sur-échantillonnage et plus particulièrement SMOTE. Les auteurs préfèrent le sous-échantillonnage dans le cas où le corpus est très grand, car le nombre des données après un sous-échantillonnage est toujours suffisant pour l'apprentissage du ML.

- Pour l'évaluation du ML, les auteurs utilisent plusieurs métriques comme le rappel, la précision ou le F-score, et pas seulement l'*accuracy*, jugé trop biaisée dans le cas des corpus déséquilibrés.

CONCLUSION

Dans ce chapitre nous avons parcouru les différents travaux qui traitent sur l'analyse des données textuelles, plus particulièrement sur les statuts dans les réseaux sociaux. Nous avons d'abord présenté les différentes solutions proposées dans la littérature pour le prétraitement, le calcul de similarités, ainsi que la représentation du texte. Nous avons présenté les travaux et les solutions liés à la classification supervisée utilisés récemment comme les réseaux de neurones à convolution CNNs ainsi que les réseaux de neurones recurrent RNNs. Notamment pour la classification non supervisée, nous avons passé en revue les solutions existantes pour le clustering du texte, comme le *Topic-Modeling* qui est établi généralement avec deux approches LDA et LSA. Nous avons de même, présenté les solutions existantes pour la problématique liée aux corpus de données déséquilibrées, qui consistaient à établir un échantillonnage de données, et ce, avec un sur-échantillonnage *Oversampling* ou bien un sous-échantillonnage *Undersampling*, ainsi que les mesures d'évaluations appropriées pour la classification avec des corpus déséquilibrés.

Dans le chapitre suivant, des techniques pour la représentation du texte, ainsi que des algorithmes de classification seront exposés en détail.

CHAPITRE 2 : ALGORITHMES DE CLASSIFICATION AUTOMATIQUE

INTRODUCTION

Ce chapitre présente en détail les algorithmes qui vont être utilisés pour la représentation, ainsi que la classification supervisée et non supervisé des tweets.

2.1 Representation du texte

Le texte brut ne peut pas être compris par un ML, de ce fait, il est essentiel de trouver une représentation numérique qui peut être traitée.

Il existe plusieurs représentations, comme le vecteur d'occurrence de mot (*Count Vector)*, le vecteur Tf-idf (*Term Frequency-Inverse Document Frequency)*, un encodage one-hot ou encore le *word embedding*.

Récemment il existe même des représentations au niveau des caractères, mais dans cette étude, la représentation est au niveau des tokens, le vecteur d'occurrence, Tf-idf et la méthode *word embedding* sont utilisés.

2.1.1 Vecteur d'occurrence

La représentation par sac de mots (ou *bag of words* en anglais) est la description de document (texte, image...) la plus intuitive mais aussi la plus utilisée en recherche d'information.

Une représentation des textes par sacs de mots aboutit à une représentation vectorielle de grande taille et donc très creuse, du fait qu'un document donné ne contient qu'une petite partie des mots possibles dans le dictionnaire, ou un document est représenté par l'histogramme des occurrences des mots le composant, pour un document donné, chaque mot se voit attribué le nombre de fois qu'il apparaît.

2.1.2 TF-IDF

La TF-IDF (de l'anglais *Term Frequency-Inverse Document Frequency)* est une méthode de pondération très utilisée pour le traitement du langage naturel. Cette mesure statistique permet d'estimer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids augmente corrélativement au nombre d'occurrences du mot dans le document. Il change également en fonction de la fréquence du mot dans le corpus.

La fréquence « brute » d'un terme TF (*Term Frequency*), est simplement le nombre d'occurrences de ce dernier dans le document considéré.

La fréquence inverse de document IDF (*Inverse Document Frequency*) mesure l'importance du terme dans l'ensemble du corpus, et vise à donner un poid important aux termes moins fréquent dans l'ensemble du corpus.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \quad 2.1$$

$$idf(t,D) = log \frac{N}{|\{d \in D | t \in d\}|} \quad 2.2$$

$$tf - idf = tf(t,d) \times idf(t,D) \quad 2.3$$

Source : [Salton & McGill, 1983].

- t : le terme present dans le document.
- D : l'ensemble des documents.
- d : le document ou t est present.
- t' : les termes present dans le document.

2.1.3 Word Embedding

Le *word embedding* (« plongement de mots » ou « plongement lexical » en français) est une technique d'apprentissage d'une représentation sémantique, qui vise à représenter chaque mot par un vecteur de nombre réel. Les *word embeddings* constituent une projection des mots du

vocabulaire dans un espace de faible dimension de manière à préserver les similarités sémantiques et syntaxiques. Par conséquent, si les vecteurs de mots sont proches les uns des autres en terme de distance, ces derniers seront considères comme sémantiquement proches. Il existe deux méthodes populaires pour le *word embedding* : Word2vec et Glove : (*Global Vectors for Word Representation*).

Dans ce travail, la méthode Word2vec et un modèle de langage neuronal (une couche embedding) sont utilisées.

2.1.3.1 Word2vec

Word2vec est essentiellement un réseau de neurone qui permet d'apprendre des représentations de mots, c'est-à-dire des vecteurs, il existe deux modèles, le premier vise à prédire le contexte en fonction des mots *Skip-gram* et à l'inverse le deuxième vise à prédire les mots en fonction du contexte (*Continuous Bag Of Words*, CBOW).

Dans ce travail, le model CBOW est utilisée.

2.1.3.1.1 CBOW

Cette architecture vise à prédire un mot en fonction des mots qui l'entourent, c'est-à-dire le contexte, et ceci en fixant la taille d'une fenêtre des mots qui le précèdent et qui le suivent (les mots voisins).

L'apprentissage des Word Embeddings est réalisé en calculant la somme des word Embeddings du contexte, puis en appliquant sur le vecteur produit un classifieur log-linéaire pour prédire le mot cible. Enfin, le modèle compare sa prédiction avec la réalité et corrige la représentation vectorielle du mot par rétro-propagation du gradient. Formellement, le modèle a pour objectif de maximiser la fonction suivante :

$$S = \frac{1}{I}\sum_{i=1}^{I} \log p\left(m_i | m_{i-n}, \dots, m_{i-1}, \dots, m_{i+n}\right) \qquad 2.4$$

Source : [Chaubard, Mundra, & Socher, 2016].

- I : nombre de mots dans le corpus
- n : taille de la fenêtre des mots.

- p : fonction Softmax

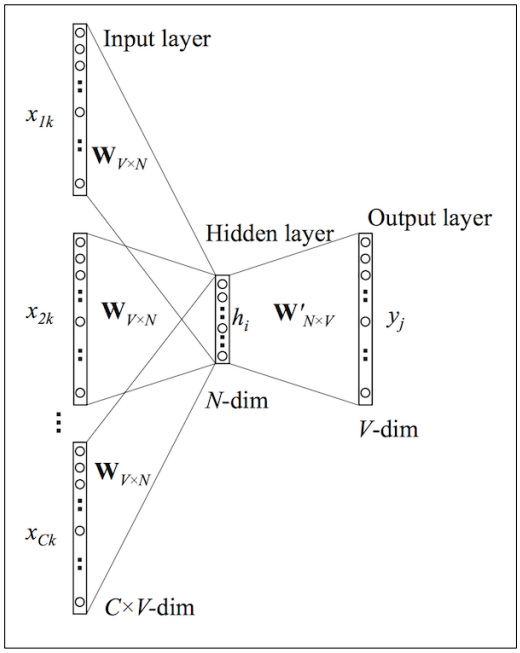

Figure 2. 1 - Model Continuous Bag of words
Source : [Meyer, 2016]

2.1.3.1.2 Skip-Gram

L'architecture skip-gram est l'inverse de CBOW. Le mot cible m_i est utilisé comme entrée, et les mots voisins (le contexte) sont utilisés en sortie. De cette manière, cette méthode vise à prédire, pour un mot donné, le contexte dont il est issu.

La couche cachée va donc prendre le mot cible pour produire un vecteur dans la couche de sortie. Par la suite, ce vecteur est comparé à chaque mot voisin du mot cible, ainsi le model corrige la représentation par rétro-propagation du gradient, afin que la représentation vectorielle du mot cible soit proche de celles des mots du même contexte. Formellement, le modèle a pour objectif, de maximiser la fonction suivante :

$$S = \frac{1}{I}\sum_{i=1}^{I}\sum_{-n\leq j\leq n,\neq 0} \log p\left(m_i \middle| m_{i+j}\right) \qquad 2.5$$

Source : [Chaubard, Mundra, & Socher, 2016].

- I : nombre de mots dans le corpus
- n : taille de la fenêtre des mots.
- p : fonction Softmax

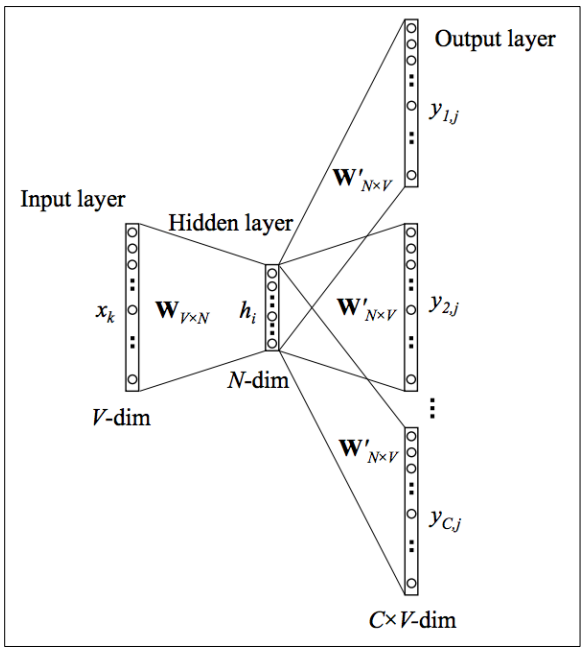

Figure 2. 2 - Model Skip-gram
Source : [Meyer, 2016]

2.1.3.2 Model de langage neuronal

Chollet explique qu'il existe deux façons d'obtenir des *word embeddings*, la première c'est avec des modèles pré-entrainées comme avec word2vec ou Glove, et la deuxième c'est d'apprendre les *embeddings* en parallèle avec la tâche principale du réseau de neurones (comme la classification des documents ou la prédiction du sentiment), dans ce cas les vecteurs des mots sont initialisés aléatoirement, et sont appris au fur et à mesure de l'entrainement du réseau de neurones [Chollet, 2018].

Cependant il est a noté que la nature des *embeddings* dans ce cas diffère du cas précèdent, vu que ce model ne constitue qu'une couche du réseau de neurone complet, les *embeddings* vont apprendre des caractéristiques plus général selon la tâche du réseau de neurone.

La couche *embedding* peut être considérer comme un dictionnaire, qui associe chaque indice (pour chaque mot spécifique dans le corpus) a un vecteur dense de nombre réel, qui prend des entiers en entrée, et retourne les vecteurs associés.

Index du mot ⟶ couche Embedding ⟶ vecteur correspondant.

2.2 Approche non supervisé

Dans l'approche non supervisé, il s'agit de trouver des structures sous-jacentes, qui présentent des caractéristiques communes, et cela à partir de données non étiquetées[1].

Il existe plusieurs algorithmes d'apprentissage non supervisé, comme les algorithmes de *clustering*, les réseaux de neurones ainsi que les algorithmes de *Topic-Modeling*.

Pour la classification non supervisé, on s'intéresse aux algorithmes de *Topic-Modeling* ainsi qu'au *clustering*.

2.2.1 Topic Modeling

Dans la classification automatique et dans le traitement automatique du langage naturel, un topic model (modèle thématique ou « modèle de sujet ») est un modèle probabiliste qui permet de déterminer des sujets ou thèmes abstraits dans un document.

Les topics model sont construit avec l'idée que les documents sont gouvernés par une variable caché. Le but est de trouver ces variables.

Parmi les techniques du topic modeling :

- Analyse sémantique latente (LSA)
- Allocation de Dirichlet latente (LDA)
- Analyse sémantique latente probabiliste (PLSA)

Dans cette étude on s'intéresse à LSA et LDA.

[1] https://fr.wikipedia.org/wiki/Apprentissage_non_supervis%C3%A9

2.2.1.1 Allocation de Dirichlet latente

Dans le domaine du traitement automatique des langues, l'allocation latente de Dirichlet (de l'anglais *Latent Dirichlet Allocation*) ou LDA est une technique générative probabiliste qui crée des probabilités au niveau des mots et des documents.

L'idée consiste à considérer les documents comme des mélanges aléatoires sur des topics sous-jacents (caché), où chaque topic est caractérisé par une distribution sur les mots.

Ainsi, si on fixe le nombre de topic a 3, un document va être considéré comme un mélange de 3 sujets (par exemple : 10% du topic 1, 20% topic du 2, 70% topic du 3).

Formellement le processus génératif est décrit par la fonction suivante :

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{k} p(\beta_i) \prod_{d=1}^{D} (p\theta_d) \prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:k}, z_{d,n}) \quad 2.6$$

Source : [Clark, 2013]

- $\beta_{1:K}$: les topics ou chaque topic β_K est une distribution sur le vocabulaire.
- θ_D : les proportions de topic pour le document d.
- $\theta_{d,k}$: les proportions de topic pour le topic k dans le document d.
- z_d : les assignations de topic pour le document d.
- $z_{d,n}$: les assignations de topic pour un mot n dans un document d.
- $w_{d,n}$: les mots pour un document d.

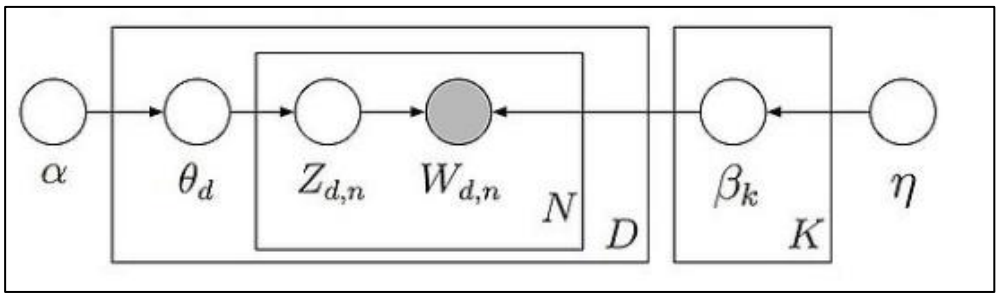

Figure 2. 3 – Model Graphique LDA
Source : [Clark, 2013]

- α and η sont des paramètres de la distribution Dirichlet, tel que, plus α est grande, plus le document sera assigner a plusieurs topics et vise-vers-ça. D'un autre côté, plus η est grande, plus le topic va contenir beaucoup de mots.

➢ **Pseudo algorithme**

1. Choisir le nombre de topic K
2. Attribuer un thème a chaque document selon une distribution une distribution de Dirichlet sur l'ensemble de K thèmes.

$\theta_i \sim Dir(\alpha), avec\ i\ \in \{1 \dots M\}\ et\ Dir(\alpha)$ est une distribution de Dirichlet avec un paramètre symétrique α creux (α <1).

On cherche à améliorer le topic model généré aléatoirement en initialisation. Pour cela, dans chaque document, on prend chaque mot et on met à jour le thème auquel il est lié. Ce nouveau thème est celui qui aurait la plus forte probabilité de le générer dans ce document. On fait donc l'hypothèse que tous les thèmes sont corrects, sauf pour le mot en question.

Pour chaque mot (w) de chaque document (d), on calcule deux choses pour chaque thème (t) :

- p (thème t | document d): la probabilité que le document d soit assigné au thème t.
- p (mot w | thème t): la probabilité que le thème t dans le corpus soit assigné au mot w.

On choisit alors le nouveau thème t avec la probabilité p (thème t | document d) * p (mot w | thème t). Ceci correspond à la probabilité que le thème t génère le mot w dans le document d.

Les étapes précédentes sont répété plusieurs de fois, jusqu'à ce que les assignations se stabilisent.

2.2.1.2 Analyse sémantique latente

L'analyse sémantique latente (LSA, de l'anglais : *Latent semantic Analysis*) 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.

LSA utilise une matrice d'occurrence (*The count matrix*), dans cette matrice les mots représentent les lignes et les documents représentent les colonnes, chaque cellule contient le nombre de fois que le mot apparait dans le document. La matrice résultante est une grande matrice creuse, le problème de la grande dimension est résolu on effectuant une décomposition en valeur singulière (en anglais *Singular Value Decomposition* : SVD) sur la matrice, qui va réduire le nombre de ligne tout en préservant la structure de similarité entre les colonnes. La méthode LSA repose donc entièrement sur SVD.

- **Principe SVD**

  Toute matrice de nombre réel peut être décomposée en 3 autres matrices.

$$A = USV^T \qquad 2.7$$

  Source : [Klema & Laub, 1980]

  - A : matrice d'occurrence m × n
  - U : matrice orthogonal de m x n
  - S : matrice diagonale de n x n
  - V^T : transposée de la matrice V, une matrice orthogonal n x n

- **Pseudo algorithme LSA**

1. Choisir K, le nombre de topics.
2. Construction de la matrice d'occurrence.
3. Corrélation :
   - Le produit matriciel AA^T contient tous les produits scalaires entre les vecteurs « termes », donnant la corrélation entre deux termes du corpus.

- le produit matriciel $A^T A$ contient tous les produits scalaires entre les vecteurs « documents », donnant leurs corrélations sur l'ensemble du lexique.

4. Decomposition en valeur singulières SVD

$$A = USV^T$$

5. Espace de concepts

   Les K plus grandes valeurs singuliers sont sélectionnés avec les vecteurs U et V correspondants. Les vecteurs « termes » et « documents » sont alors traduits dans l'espace des « concepts ».

   Chaque terme possède alors K composantes qui reflètent son importance dans le concept, de même, pour chaque document. L'approximation est écrite de cette manière : $A_k = U_k S_k V_k^T$.

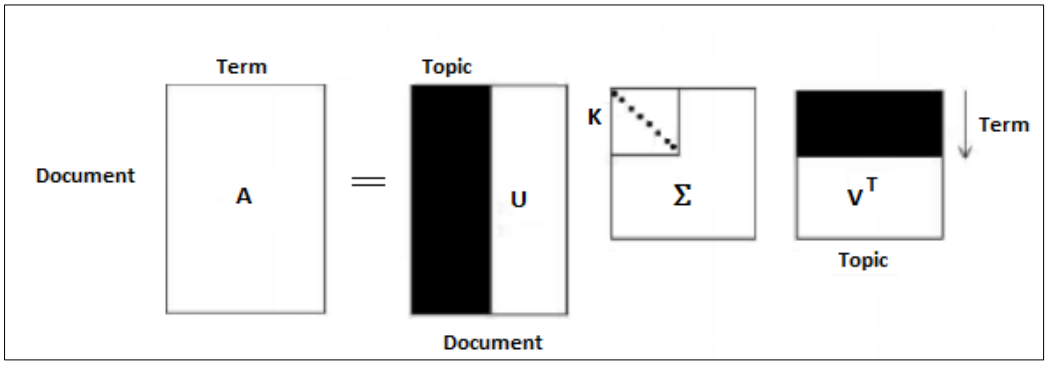

Figure 2. 4 - Decomposition en valeur singulière LSA
Source : [Steinberger& Ježek, 2004]

2.2.2 Clustering

Le partitionnement de données (ou *data clustering* en anglais) est une des méthodes d'analyse des données. Elle vise à diviser un ensemble de données en différents « groupes » homogènes. Comme par exemple : l'algorithme K-means, DBSCAN, regroupement hiérarchique etc.

Dans cette étude on s'intéresse à l'algorithme K-means.

2.2.3 K-means

Le partitionnement en k_moyennes (ou *K-means* en anglais) est une méthode de partitionnement de données et un problème d'optimisation combinatoire.

L'algorithme vise à diviser des points en K groupes appelés *clusters*, et ceci en minimisant l'inertie ou somme des carrés intra-classe.

$$I_{intra} = \sum_{k=1}^{k} \sum_{i \in C_k} d^2(x_i, \mu_k) \qquad 2.8$$

Source : [Chavent, 2013]

Ou :

- d : une mesure de distance caractérisant les proximités entre les individus.
- μ_k : Le centroide des points dans C_k .

➢ **Pseudo algorithme**

1. Choisir K (le nombre de cluster).
2. Choisir K objets arbitraire du training-set comme des centres de cluster initiaux

Répéter :

- Chaque donnée est assignée à la classe du centre dont elle est la plus proche.

  $C_l \longleftarrow x_i$ tel que $l = \min d(x_i, \mu_k)$ ou d est la distance Euclidienne :

$$d(x_{i,}\mu_k) = \sqrt{\sum_{j=1}^{n} (x_{i_j} - \mu_{k_j})^2} \qquad 2.9$$

  Source : [Chevalier & Bellac, 2013].

- Le centre de chaque cluster k est recalculé comme étant la moyenne arithmétique de toutes les données appartement à cette classe

$$\mu_k = \frac{1}{N_k} \sum_{i \in C_k} x_i \text{ avec } N_k = card(C_k)$$

Jusqu'à stabilisation des clusters.

2.3 Approches supervisées

L'apprentissage supervisé est une tâche qui consiste à apprendre une fonction qui permettra de regrouper des objets en se basant sur un ensemble d'apprentissage étiqueté de paires (entré-sortie).

On peut citer plusieurs algorithmes d'apprentissage supervisé comme :

- Méthode des k plus proches voisins.
- Réseau de neurones.
- Arbre de décision.
- Classification naïve bayésienne.

Dans cette étude on s'intéresse aux réseaux de neurones artificiels ainsi qu'à la méthode des k plus proches voisins.

2.3.1 Réseau de neurones artificiel

Un réseau de neurones artificiel est un modèle inspiré du fonctionnement des neurones biologiques, dont l'origine est d'essayer de résoudre des problèmes comme le fait un cerveau d'un être humain, et qui par la suite s'est rapproché des méthodes statistiques. Maintenant les réseaux de neurones sont utilisés pour apprendre différentes taches dans plusieurs domaines, comme la classification, la prédiction, la traduction, …, etc.

- **Perceptron**

Le perceptron peut être considéré comme le type de réseau de neurones le plus simple. Le perceptron prend des entrés booléennes plus généralement des nombres réels et produit une seule sortie booléenne.

- **Principe du perceptron**

La Figure 2. 5 montre un perceptron a trois input (x1,x2,x3) avec des poids « *weights* » (w1,w2,w3), des nombres réels qui exprime l'importance de l'input par rapport à

l'output, l'output y étant 0 ou 1 est déterminé par l'application de la fonction de 2.10 au potentiel post-synaptique ou θ est un seuil (*bias*).

$$\boldsymbol{Z = \sum_{i=1}^{n} w_i . x_i} \qquad 2.10$$

$$\boldsymbol{y = f}(z) = \begin{cases} 0 \; si \; z \leq \theta \\ 1 \; si \; z > \theta \end{cases} \qquad 2.11$$

Source : [Petitjean, 2006]

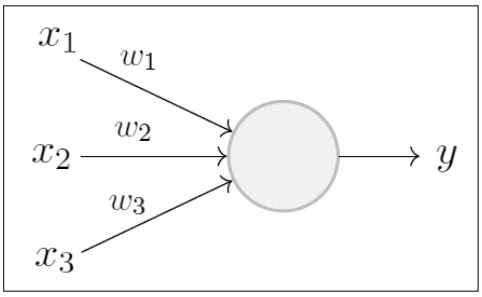

Figure 2. 5 - Schéma d'un perceptron
Source : [Minsky-Papert, *1969*]

- **Perceptron multicouche**

Le perceptron multicouche est un réseau à propagation directe, contenant plusieurs couches, dont chacune prend en entré ses entrées sur les sorties de la précédente.

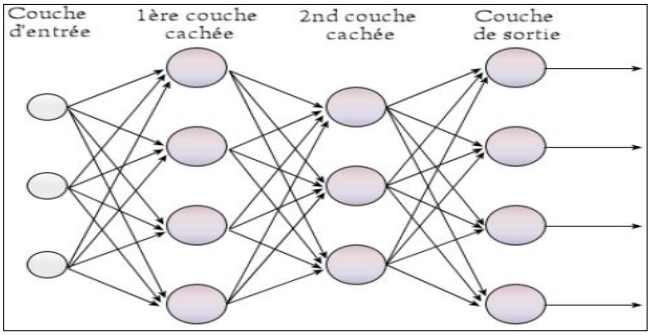

Figure 2. 6 - Perceptron multicouche.
Source : [HRcommons, 2009]

- Chaque couche (i) est composée de Ni neurones, prenant leurs entrées sur les Ni-1 neurones de la couche précédente.
- À chaque synapse est associé un poids synaptique, de sorte que les Ni-1 sont multipliés par ce poids, puis additionnés par les neurones de niveau i, ce qui est équivalent à multiplier le vecteur d'entrée par une matrice de transformation.
- Le résultat est transmis à une fonction d'activation qui opère une transformation sur la combinaison (c'est-à-dire le produit scalaire entre le vecteur des entrées et le vecteur des poids synaptiques) et renvoie la sortie.

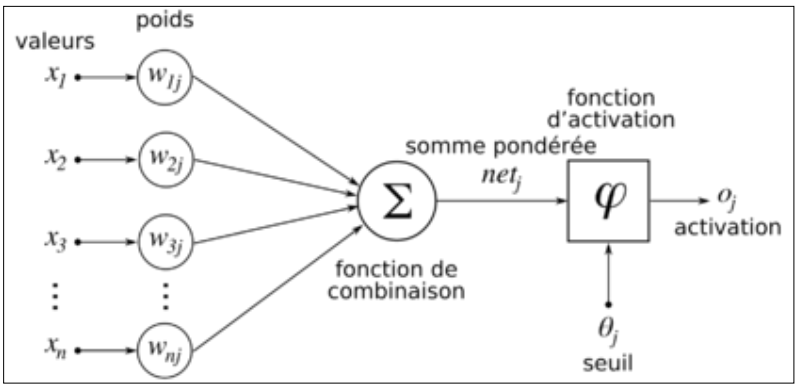

Figure 2. 7 - Structure d'un neurone artificiel.
Source : [Chrislb, 2005]

2.3.1.1 Réseau neuronal recurrent

Un réseau de neurones récurrents (en anglais RNN pour Recurrent Neural Network) est un réseau de neurones artificiel, adapté pour le traitement des séquences temporelles en conservant un état pour chaque élément de la séquence, cet état contient des informations relatives sur les éléments précédents.

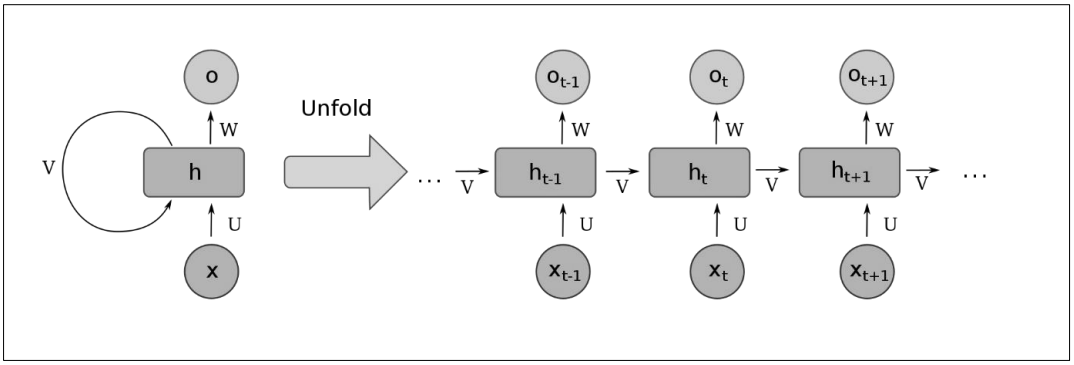

Figure 2. 8 - Schéma d'une unité récurrente
Source : [Deloche, 2017a]

- La sortie des étapes précédentes est redonnée en entrée à l'étape courante, l'information contextuelle parcourt le réseau dans sa totalité, affectant ainsi toutes les couches du réseau.
- Contrairement aux réseaux neurones traditionnelles qui utilisent différents paramètres dans chaque couche, RNN partage les mêmes paramètres dans toutes les étapes.
- Dans la figure 2.8, on a des sorties dans chaque étape, mais ceci ni pas nécessaire, par exemple, si on devait prédire le sentiment d'une phrase, on se souciera que de la sortie final (pas de la sortie qui concerne chaque mot).

Les réseaux de neurones récurrents classiques sont exposés au problème de disparition du gradient (la disparition du gradient rend la minimisation de la perte difficile) pratiquement, un RNN n'est pas vraiment capable de capturer toutes les informations précédentes. En d'autres termes plus une séquence est longue plus le RNN devient vulnérable. Pour répondre à cette problématique, d'autres types de réseaux de neurones recurrent sont introduits : *Long short-term memory* LSTM et *Gated recurrent unit* GRU.

Dans cette étude on s'intéresse à LSTM.

2.3.1.1.1 Réseau récurrent à mémoire court et long terme

Comme les RNN, les LSTM *Long Short-Term Memory* ont une structure chainée, seulement le module qui se répète a une structure différente.

- **Principe LSTM**

Les LSTMs ont des mécanismes internes appelés portes (*Gates*), ces portes servent à régulariser le flux d'information en apprenant quelles informations sont utiles à garder ou à oublier dans le temps. Ce faisant, le réseau pourra transmettre des données pertinentes, et ceci, grâce à l'état de la cellule, agissant comme une route de transport qui transfère les informations relatives tout au long de la chaîne de séquences.
Au fur et à mesure que l'état se poursuit, des informations sont ajoutées ou supprimées grâce à trois portes principales.
Dans un premier temps, en entrée il y a la sortie prédite précédemment h_{t-1} ainsi que l'entrée actuelle X_t. Ces données vont être décomposées en trois flux. L'objectif étant de mettre à jour l'état de la cellule de C_{t-1} à C_t.

- **Porte d'entrée :**

La porte d'entrée *Input Gate*, met à jour l'état de la cellule. La fonction d'activation est Sigmoïde. Pour chaque input, la porte fournira en sortie une valeur entre 0 et 1, et décide quelle valeur mettre à jour (0 signifie pas important et 1 signifie important), Ce résultat est ensuite multiplié par l'état actuel.

$$f_t = \sigma(W_t[h_{t-1}, X_t] + b_t) \qquad 2.12$$

$$C_t = f_t \times C_{t-1} \qquad 2.13$$

- **Porte d'oubli :**

La porte d'oubli *Forget Gate*, décide quelles informations doivent être jetées ou conservées. Les informations de l'état précèdent sont passés à la fonction Tanh donnant des valeurs entre -1 et 1. Ensuite, afin de sélectionner les caractéristiques importantes, une couche sigmoïde va décider quelle valeur va être mise à jour.

$$f_c = \tanh(W_c[h_{t-1}] + b_c) \quad 2.14$$

$$I_t = \sigma(W_i[h_{t-1}, X_t] + b_i) \quad 2.15$$

Ces deux résultats seront multipliés et ajouté à l'état, a ce stade l'état de la cellule est :

$$C_t = f_t \times C_{t-1} + f_c \times I_t \quad 2.16$$

- **Porte de sortie :**

La porte de sortie *Output Gate*, passera l'état caché précédent et l'entrée actuelle dans une fonction sigmoïde.

$$o_t = \sigma(W_o[h_{t-1}, X_t] + b_o) \quad 2.17$$

Le nouvel état caché est donc :

$$h_t = o_t \times \tanh(C_t) \quad 2.18$$

Source : [Donadio, 2018].

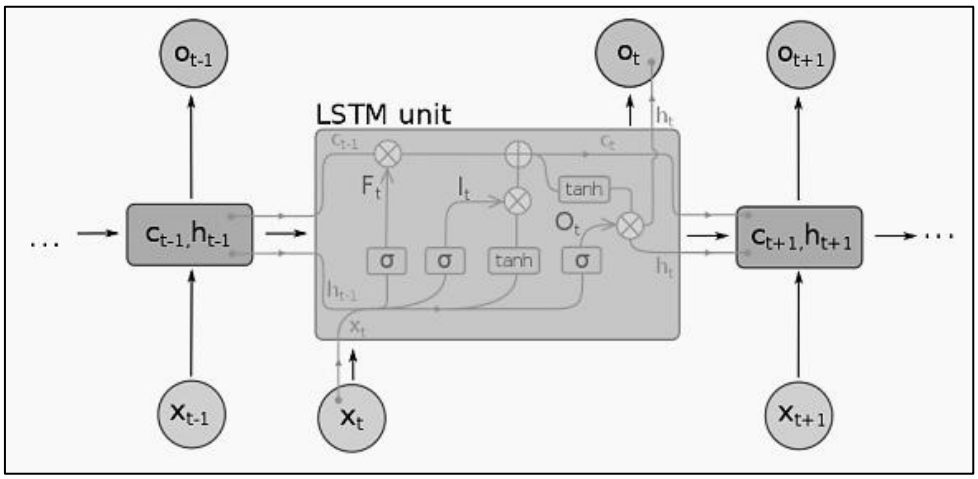

Figure 2. 9 - Schéma d'une unité LSTM
Source : [Deloche, 2017b]

- **Architecture réseau de neurone avec une couche LSTM**

  La première couche c'est la couche *embedding*, suivi d'un dropout pour éviter l'overfitting ensuite une couche LSTM, et enfin, une dernière couche de sortie (dense layer).

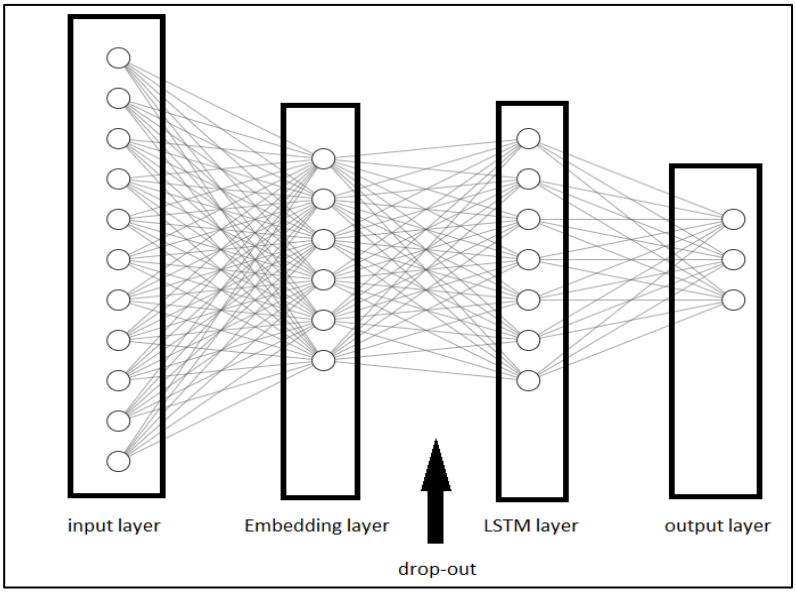

Figure 2. 10 - Architecture Réseau de neurones LSTM

Le dropout permet d'éviter le sur-apprentissage (*Overfitting*).

Cette couche est utilisée pendant l'apprentissage. Elle permet de désactiver aléatoirement des neurones durant les différentes itérations de l'apprentissage. Chaque neurone étant possiblement inactif pendant une itération d'apprentissage, cela force chaque unité à « bien apprendre » indépendamment des autres et évite ainsi la « coadaptation ».

Le sur-apprentissage (*Overfitting*), s'interprète comme un apprentissage « par cœur » des données, un genre de « mémorisation». C'est le cas quand la perte d'entrainement est très inferieur a la perte de validation, par exemple, si le modèle réussit mieux sur le *set* d'entraînement que sur le *set* de test, il est probable que ce soit à cause d'un sur-apprentissage. D'un autre coté il y a aussi le sous-apprentissage (*Underfitting*) ou la perte d'entrainement est très supérieure à la perte de validation c'est généralement le cas quand le model est très simple ou qu'il nécessite plus d'entrainement.

2.3.1.2 Les réseaux de neurones à convolutions (CNN)

Les réseaux de neurones à convolution (CNN : *Convolutional Neural Networks*) sont inspirés du cortex visuel. Deux types de cellules existent. Les cellules dites simples pouvant détecter des caractéristiques liées à la forme. Et d'autres dites complexes, plus sensibles à la forme entière.

Un avantage majeur des réseaux convolutifs est l'utilisation d'un poids unique associé aux signaux entrant dans tous les neurones d'un même noyau de convolution. Cette technique réduit l'utilisation de la mémoire, la complexité du modèle et améliore les performances. Par contre dans un réseau de perceptron multicouche, chaque neurone est indépendant, de ce fait, des poids différents seront affecter à chaque signal.

Dans ce travail, le CNN de Yoon pour la classification du texte court va être testé sur des tweets. Mais d'abord, le principe du CNN est présenté :

- **Principe CNN**

Dans un CNN, chaque neurone est relié à une petite zone (un sous-ensemble) de l'output de la couche précédente. On peut considérer chaque neurone comme une unité de détection d'une caractéristique locale, qui effectue une opération de filtrage (convolution) et dont le principe, est de faire glisser une fenêtre qui parcourt les données.

$$h_{ij}^{k} = g((W^{k} * x)_{ij} + b_{k}) \qquad 2.19$$

Chaque liaison entre les neurones est équilibré par une matrice des poids W et un vecteur de biais b qui contrôle le potentiel d'activation des neurones. La sortie est obtenue par l'utilisation d'une fonction d'activation g.

Un autre concept important des CNNs est le *pooling*, une forme de sous-échantillonnage appliqué sur la fenêtre glissante. Cette couche est souvent placée entre deux couches de convolution, dont le but est de réduire la taille des données toute en préservant les caractéristiques importantes.

- Couche de Max-Pooling :

Le *Max-Pooling* a pour effet de ne prendre que la valeur maximale de la sortie de la fonction d'activation g appliquée sur différentes régions, qui ne se chevauchent pas, du signal global. Cela a pour effet de retenir que certaines informations.

- Couche de Mean –Pooling :

Le *Mean–Pooling* (*Average-Pooling*) prend en compte toute les caractéristiques et ne rejette aucune information, et ceci en calculant la moyenne des valeurs du patch d'entrée.

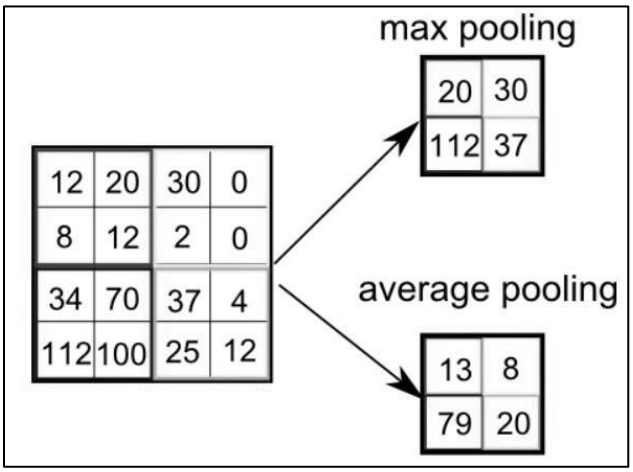

Figure 2. 11- Schéma du fonctionnement du Max-Pooling et Mean-Pooling
Source : [Saha, 2018]

Le *pooling* permet d'éviter des scénarios comme le sur-apprentissage, et rapporte aussi de gros gains en puissance de calcul. Cependant, en raison de la réduction brutal de la taille de la représentation (et donc de la perte d'information associée), la tendance actuelle est d'utiliser de petits filtres (2 x 2) ou (3 x 3).

Souvent, dans CNN, la fonction d'activation Relu est utilisée, Cette fonction, appelée aussi « fonction d'activation non saturante », augmente les propriétés non linéaires sans affecter les champs récepteurs de la couche de convolution.

Dans un réseau de neurone à plusieurs couches CNN, La couche *Fully-Connected* constituera la dernière couche, Les neurones dans une couche entièrement connectée ont des connexions vers toutes les sorties de la couche précédente (comme on le voit régulièrement dans les réseaux réguliers de neurones).

- **Architecture CNN-Yoon**

Dans l'architecture du réseau de neurones de Yoon pour la classification binaire du texte court. La première couche c'est la couche *embedding*. La couche suivante va effectuer des convolutions avec plusieurs filtres, sur 3,4 et 5 mots, par la suite, un Max-Pooling est effectué. Un drop-out est notamment ajouté pour éviter *l'overfitting*. Le résultat final est donné par une dernière couche de sortie (appelé aussi une couche dense).

Yoon a essayé plusieurs variantes du model. Dans ce travail, le CNN-rand est utilisé, ou le model initialise les *embeddings* aléatoirement et les ajustes au fur et à mesure dans l'apprentissage.
Plus précisément, une implémentation inspirée du travail de Britz pour le CNN-rand va être testée [Britz, 2015]. Britz a laissé le même prétraitement que Yoon (similaire au prétraitement présenté dans le troisième chapitre, seulement sans Stemming), et a effectué quelques changements :

- La norme L2 ne va pas être utilisée sur les vecteurs, une étude [Zhang & Wallace, 2016], a prouvé que ça avait un effet négatif sur le résultat final. Cette norme (Régularisation L2) a pour but de minimiser la perte mais aussi la complexité du model.
- L'optimiseur Adam est utilisé au lieu d'Adadelta.

Enfin une dernière modification est accomplie pour prendre en considération la classification multiple.

Dans ce travail, les mêmes paramètres du travail original sont utilisés (voir le cinquième chapitre).

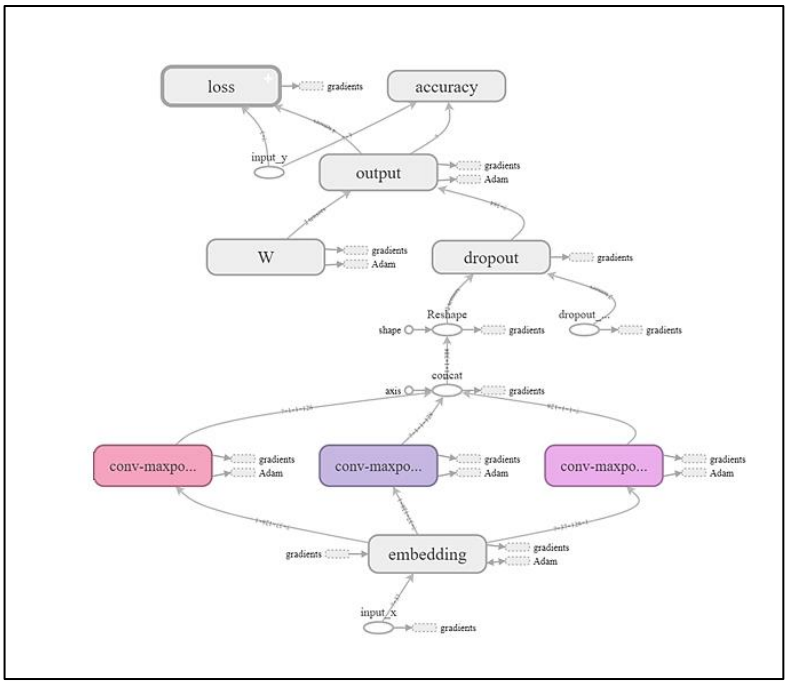

Figure 2. 12 - Architecture du CNN de Britz obtenu avec Tensorboard

2.3.1.3 Fonctions d'activations

Une fonction d'activation est une fonction mathématique appliquée à une combinaison des entrées x_i, et introduit donc, le principe de non linéarité, qui permet de s'adapter et de différencier les résultats. Le choix de la fonction d'activation dépend de l'application, s'il faut avoir des sorties binaires ou non, dans ce travail les fonctions Sigmoïde, Softmax, Tanh ainsi que ReLu sont utilisées.

- **Principe Sigmoïde**

  La fonction sigmoïde, est définie par :

$$f(x) = \frac{1}{1+e^{-x}} \quad ou\ x \in R \qquad 2.20$$

  Le but étant de convertir la valeur d'entrée en une probabilité de 1 si c'est un très grand nombre positif, et à l'inverse, à 0 si l'entrée est un très grand nombre negative.

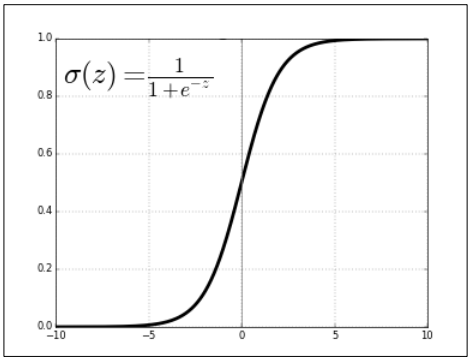

Figure 2. 13 – Graphe de la fonction Sigmoïde

- **Principe Softmax**

  La fonction Softmax, ou fonction exponentielle normalisée, est définit par :

$$f(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; pour\ tout\ j\ \in \{1, \dots, K\}, x = \{x_1, \dots, x_k\}\ ou\ k \in R \qquad 2.21$$

  La fonction donne en sortie un vecteur de K nombres réels strictement positifs et de somme 1.

- **Principe Tanh**

  La fonction tangente hyperbolique, notée Tanh, est définit par :

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad 2.22$$

  La sortit dans Tanh sera rangé dans le champ [-1,1].

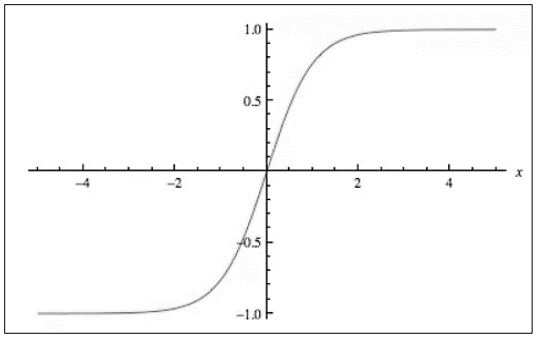

Figure 2. 14 – Graphe de la fonction Tanh

- **Principe ReLu**

  La fonction d'unité de rectification linéaire ReLu est la fonction la plus simple, définit par :

$$f(x) = \begin{cases} 0 \; pour \; x < 0 \\ x \; pour \; x \geq 0 \end{cases} \qquad 2.23$$

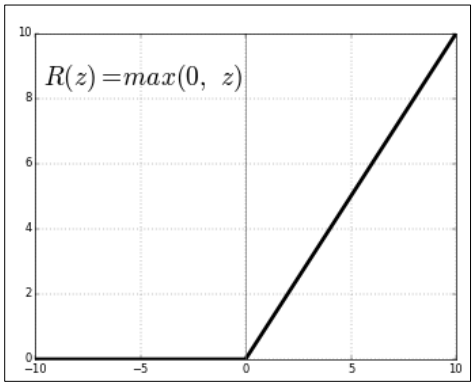

Figure 2. 15 – Graphe de la fonction ReLu

La sortie vaut donc 0 si l'entré est inferieur a 0, sinon, elle sera égal à l'entrée.

2.3.1.4 Fonctions d'optimisation

Généralement, les fonctions d'optimisation dans les réseaux de neurones, ont pour objective de calculer le gradient pour trouver des valeurs (poids) optimisées, et ceci en modifiant les poids dans la direction opposée, ce cycle est répété jusqu'à la minimisation de la fonction objective.

Dans ce travail différentes fonctions d'optimisations sont testés : Adam, Adamax, Adadelta et SGD.

- **Principe SGD**

  Le mot «stochastique» désigne un système ou un processus lié à une probabilité aléatoire. Par conséquent, dans la descente de gradient stochastique quelques échantillons sont sélectionnés de manière aléatoire au lieu de l'ensemble des données de chaque itération

SGD (*Stochastic Gradient Descent*), met à jour les paramètres pour chaque exemple de l'ensemble de données x_i et étiquette (*label*) y_i.

- Initialiser avec x0 (au hasard)
- Répéter :

$$x_t + 1 = x_t - \eta \times \nabla f(x_t)$$

Jusqu'à convergence

- η : permet de moduler la correction (η trop faible, lenteur de convergence ; η trop élevé, oscillation)
- $\nabla f(x_t)$: Le gradient au point x_t montre la direction et l'importance de la pente au voisinage de x_t.

Cette méthode est généralement plus rapide, cependant, en raison des mises à jour fréquentes, la convergence devient plus difficile (trouver le minimum de la fonction objective).

- **Principe Adam**

Adam *Adaptive Moment optimization,* est l'un des algorithmes les plus récents et les plus efficaces pour l'optimisation par descente de gradient.
Adam calcul la moyenne exponentielle du gradient ainsi que les carrés du gradient pour chaque paramètre, le taux d'apprentissage est ensuite multiplié par la moyenne du gradient et en le divisant par la racine carré de la moyenne exponentielle des gradients. Ensuite la mise à jour est ajoutée.

$$v_t = \beta_1 \times v_{t-1} - (1 - \beta_1) \times g_t \quad (2.24)$$

$$s_t = \beta_2 \times s_{t-1} - (1 - \beta_2) \times g_t^2 \quad (2.25)$$

$$\Delta w_t = -\eta \frac{v_t}{\sqrt{s_t + \epsilon}} \times g_t \quad (2.26)$$

$$w_{t+1} = w_t + \Delta w_t \quad (2.27)$$

Source : [Kingma & Ba, 2015]

- η : learning rate initial.
- g_t : le gradient dans un temps t.
- v_t : la moyenne exponentielle des gradients.
- s_t : la moyenne exponentielle des carrées des gradients.
- $\beta_1, \beta_2, \epsilon$: des hyper paramètres initialisés généralement a 0.90, 0.99 et 1e-10.

- **Principe Adamax**

  Adamax est une généralisation d'Adam pour obtenir une convergence plus stable, ou au lieu de prendre :

$$s_t = \beta_2 \times s_{t-1} - (1 - \beta_2) \times g_t^2$$

  s_t devient :

$$s_t = \beta_2^p \times s_{t-1} - (1 - \beta_2^p) \times g_t^p \qquad 2.28$$

  Ou p = ∞.

  Source : [Kingma & Ba, 2015]

- **Principe Adadelta**

  Adadelta propose une adaptation du gradient dépendante de son moment de second ordre et de celui de l'état de la couche du réseau corrigé, et utilise une taille fixe w pour la prise en compte de la somme des carrés de gradients accumulés dans le passé.

2.3.1.5 Fonctions de pertes

Une fonction de perte représente une certaine mesure de la différence entre les valeurs observées des données et les valeurs calculées. C'est la fonction qui est minimisée dans la procédure d'optimisation d'un modèle.

Il existe plusieurs fonctions, et le choix dépend du cas d'utilisation, pour une régression, une classification binaire ou une classification multiple. Dans cette étude on s'intéresse à la classification binaire et multiple.

- **Fonction de perte pour un problème de classification**

  Cette catégorie regroupe plusieurs méthodes, parmi ces méthodes :

  - Negative Log Likelihood.
  - Margin Classifier.
  - Cross Entropy.
  - Poisson.
  - Hinge.

  Dans cette étude, la perte Cross Entropy est utilisée.

- **Principe Cross Entropy**

  Dans une classification binaire, ou le nombre de classes à prédire est égale à deux, la perte Cross Entropy est calculée comme suit :

$$-(ylog(p) + (1-y)\log(1-p)) \qquad 2.29$$

  Si le nombre de classes > 2 (classification multiple) la perte distincte est calculée pour chaque étiquette de classe par observation et additionnement du résultat :

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c}) \qquad 2.30$$

  Source : [Fortuner, Viana, & kowshik, 2018]

  - M : nombre de classes
  - y : indicateur binaire (0 ou 1) si l'étiquette de la classe c est la classification correcte pour l'observation o.
  - p : la probabilité que l'observation o appartient à la classe c

2.3.2 Méthode des k plus proches voisins

La méthode des k plus proches voisins (KNN : *K-Nearest Neighbors*), vise à classifier une instance en fonction de ces K plus proches voisins, et ceci, en calculant une distance quelconque.

- **Pseudo algorithme**

1. Choisir le nombre K : nombre des voisins.
2. Pour chaque instance test :
   - Trouver la distance (Cosinus, Euclidienne …) avec toutes instances d'apprentissage.
   - Trié les distances dans une liste.
   - Choisir les K premier classes (étiquettes) relative au K premières distances.
   - Assigner une classe à l'instance test en se basant sur la majorité des classes des K premier point.

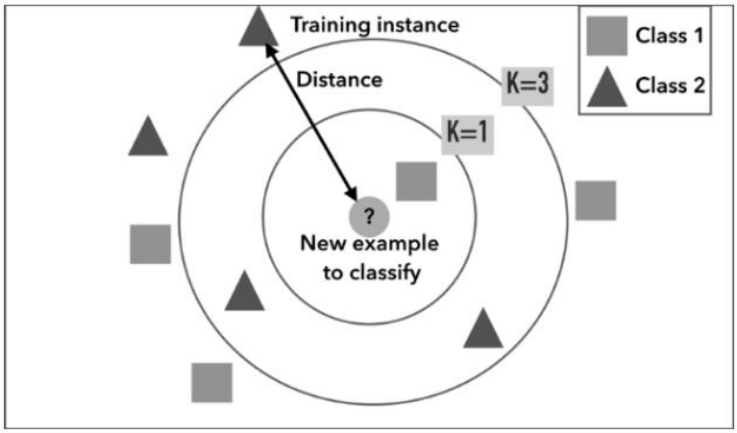

Figure 2. 16 - Example d'une classification K-NN.
Source : [Arnaud, 2018].

CONCLUSION

Dans ce chapitre, nous avons exposé des techniques de représentation du texte comme le *word embedding*, ainsi que des algorithmes de classification supervisée. Essentiellement nous avons présenté la classification supervisée avec deux types de réseaux de neurones. Le premier est un réseau de neurones à convolution CNN, nous avons donné comme exemple le model de Yoon pour la classification du texte. Le deuxième, étant les réseaux de neurones récurrents RNNs de type LSTM. Nous avons notamment présenté des algorithmes de *clustering* ainsi que de *Topic-Modeling* pour la classification non supervisé.

Dans le chapitre suivant, des techniques d'échantillonnage (*Sampling*) ainsi que des mesures d'évaluations seront exposées.

CHAPITRE 3 : TECHNIQUES D'ECHANTILLONNAGE ET MESURES D'EVALUATIONS

INTRODUCTION

Dans ce chapitre, nous nous intéressons aux techniques adaptées pour le traitement de corpus de données déséquilibrées, et plus précisément les techniques de sur-échantillonnage (*Oversampling)* et de sous-échantillonnage (*Undersampling)*. Par la suite, des mesures d'évaluations des algorithmes de classification sont présentées.

3.1 Techniques d'échantillonnage

Plusieurs scenarios pourraient influencer les systèmes d'apprentissage existants, un corpus déséquilibré dans lequel le nombre des objets d'une classe dépasse fortement le nombre d'objet d'une autre classe est sans doute parmi les causes d'une mauvaise prédiction. Dans ce cas, les algorithmes d'apprentissage seront incapables d'identifier des objets qui appartiennent aux classes minoritaires.

Les techniques de *sampling* sont utilisées pour ajuster la distribution des classes d'un corpus. On retrouve deux catégories principales : le sur-échantillonnage et sous-échantillonnage (*l'oversampling* et *l'undersampling*). Cependant il existe aussi des hybridations entre le sur-échantillonnage et le sous-échantillonnage. Elles consistent à établir d'une manière simultanée l'augmentation des *samples* de la classe minoritaire par les approches de sur-échantillonnage et la diminution des *samples* de la classe majoritaire par les approches de sous-échantillonnage pour équilibrer la répartition des classes.

Dans ce travail, on s'intéresse à *l'oversampling* ainsi qu'à *l'undersampling*.

3.1.1 Oversampling

L'*oversampling* (sur-échantillonnage) est une technique qui rajoute de nouvelles données dans la classe minoritaire. Parmi les techniques de sur-échantillonnage on retrouve un Random Oversampling, SMOTE (*Synthetic Minority Oversampling Technique*) ou encore ADASYN (*Adaptive Synthetic Sampling Approach*).

Dans cette étude on s'intéresse à l'oversampling avec SMOTE.

3.1.1.1 SMOTE

SMOTE (*Synthetic Minority Oversampling Technique)*, est une technique de sur-échantillonnage qui repose sur l'algorithme des K-plus proches voisins K-NN (Voir le deuxième chapitre). SMOTE permet de générer des individus artificiels dans la classe minoritaire. Pour chaque individu de la classe minoritaire, ses k plus proches voisins sont calculés, puis des individus parmi les voisins sont sélectionnés aléatoirement, des individus artificiels (synthétiques) sont ensuite disséminés aléatoirement le long de la ligne entre l'individu de la classe minoritaire et ses voisins sélectionnés.

- **Pseudo algorithme**

Pour chaque instance de la class minoritaire :

1. Choisir les K plus proches voisins avec K-NN.
2. Choisir un voisin parmi ces voisins.
3. Calculer la différence entre le vecteur voisin et le vecteur du point actuel.
4. Multiplier le résultat par un nombre entre 0 et 1.
5. Le nouveau point synthétique est le résultat de l'addition du résultat de l'étape 4 avec le vecteur du point actuel.

Répéter jusqu'à ce que le dataset soit équilibré.

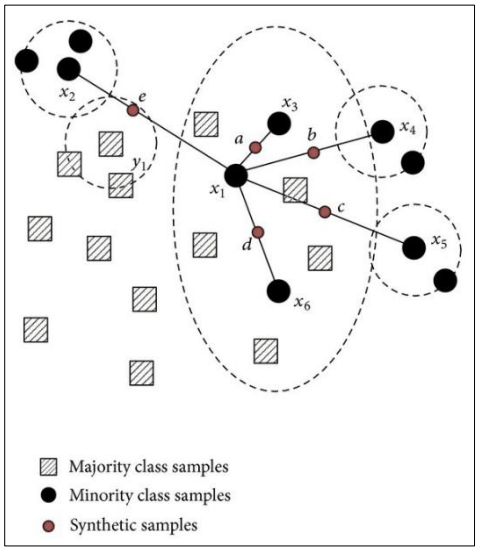

Figure 3. 1- Schéma création d'une donnée synthétique dans SMOTE

[Hu & Li, 2013]

3.1.2 Undersampling

L'undersampling (sous-échantillonnage) est une technique qui supprime des donnés de la classe majoritaire. Parmi les techniques de sous-échantillonnage on retrouve le Random Undersampling et Tomek-links. Dans cette étude on réduit la classe majoritaire jusqu'à-ce-que le nombre des samples dans chaque classe soit équilibré.

3.2 Métriques d'évaluations

Il existe plusieurs métriques pour l'évaluation des algorithmes d'apprentissage, et le choix dépend entièrement du cas d'utilisation. L'évaluation consiste à utiliser des *benchmark*s, afin de mesurer la différence entre un résultat réel et un résultat obtenu par le ML.

Dans cette étude, l'*accuracy*, le rappel, la précision et le F-score sont utilisés.

3.2.1 Accuracy

C'est la métrique le plus utilisés et la plus intuitive pour l'évaluation des algorithmes d'apprentissage.

$$Accuracy = \frac{nombre\ total\ de\ documents\ correctement\ classifiés}{nombre\ total\ de\ documents} \qquad 3.1$$

Source : [valuations, 2015].

Parfois, cette métrique est très biaisée. Lorsque 90 % des instances appartiennent à une classe A et 10% des instances à une classe B, il est très probable que le ML (*Machine Learning algorithm)* prédira que toute les instances appartiendrons a la classe A, ainsi l'*accuracy* sera de 90%.

3.2.2 Le Rappel

Le rappel ou sensibilité (*Recall* en anglais*),* est défini par le nombre de documents pertinents correctement prédits d'une classe i, au regard du nombre de documents total de la classe i.

Un rappel élevé, signifie que le ML a pu prédire correctement un bon nombre de documents pour chaque classe.

$$recall = \frac{nombre\ de\ documents\ correctement\ attribués\ a\ la\ classe\ i}{nombre\ de\ document\ appartenant\ a\ la\ classe\ i} \qquad 3.2$$

Source : [valuations, 2015].

3.2.3 La précision

La précision est le nombre de documents pertinents retrouvés rapporté au nombre de documents total proposé par le ML. Si elle est élevée, cela signifie que la plupart des documents d'une classe i sont correctement classifiés, ainsi ce dernier peut être considéré comme « précis ».

$$precision = \frac{nombre\ de\ documents\ correctement\ attribués\ a\ la\ classe\ i}{nombre\ de\ document\ attribués\ a\ la\ classe\ i} \qquad 3.3$$

Source : [valuations, 2015].

3.2.4 F-score

L'apprentissage à partir de données déséquilibrées revient souvent à effectuer un arbitrage entre le rappel et la précision du modèle. Avoir une haute précision signifie que la plupart des documents attribués à une classe i appartiennent à cette classe. Par contre, avoir un taux de rappel élevé signifie qu'on peut identifier la plupart des individus pour chaque classe du corpus (de cette manière on est capable de détecter des individus d'une classe minoritaire).

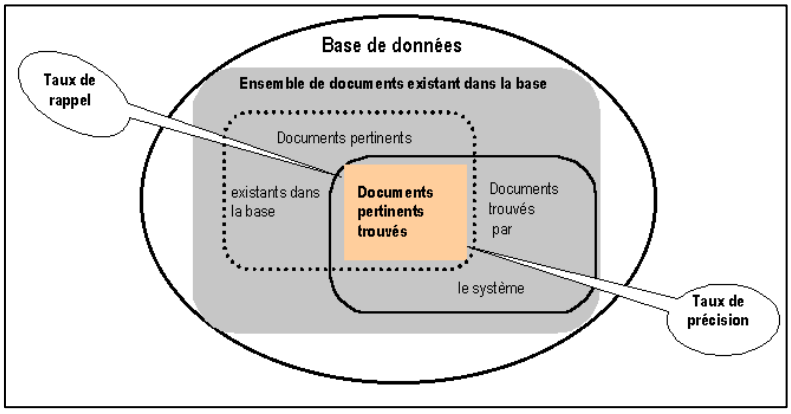

Figure 3. 2 - Taux de rappel et taux de précision dans un système d'information.

Source : [ROUISSI, 2001].

Idéalement, on voudrait réduire le nombre d'individus mal attribués à une classe i. La différence entre le rappel et la précision se situe surtout dans la façon de considérer l'objectif :

- Soit l'établissement des classes les plus justes possibles.
- Soit l'établissement d'une classification de textes la plus juste possible.

Les deux visions sont donc plus complémentaires qu'antagonistes.

Le F-score (la F-mesure ou indice de Dice) correspond à un compromis de la précision et du rappel, donnant la performance du système. Ce compromis est donné de manière simple par la moyenne harmonique de la précision et du rappel.

$$F - score = 2.\frac{precision.recall}{precision + recall} \quad 3.4$$

Source : [valuations, 2015].

3.2.5 Matrice de confusion

La matrice de confusion est une matrice qui va permettre de mesurer la qualité d'un système de classification. Chaque ligne correspond à une classe réelle, chaque colonne correspond à une classe estimée.

Un des avantages de la matrice de confusion, est le fait de montrer rapidement si un système de classification parvient à classifier correctement.

Tableau 3. 1 - Exemple d'une matrice de confusion.

	GoldLabel_A	**GoldLabel_B**	**GoldLabel_C**	
Predicted_A	30	20	10	TotalPredicted_A=60
Predicted_B	50	60	10	TotalPredicted_B=120
Predicted_C	20	20	80	TotalPredicted_C=120
	TotalGlodLabel_A=100	TotalGlodLabel_B=100	TotalGlodLabel_C=100	

Dans le tableau on a :

- A, B et C sont les classes.
- La diagonal contient les documents correctement classifiés.
- La somme d'une colonne est le nombre total des instances d'une classe x.
- La somme d'une ligne est le nombre des instances attribués à une classe x.

Ainsi, pour plusieurs classes n, le rappel et la précision sont calculés comme suit :

$$recall = \frac{\sum_{i=1}^{n} recall_i}{n} \quad 3.5$$

$$precision = \frac{\sum_{i=1}^{n} precision_i}{n} \quad 3.6$$

CONCLUSION

Dans ce chapitre, nous avons présenté les techniques d'échantillonnages des corpus déséquilibrés. Ce déséquilibre se présente aux niveaux des classes. Nous avons exposé les deux principales méthodes d'échantillonnage, le sur-échantillonnage (*oversampling*) ainsi que le sous-échantillonnage (*undersampling)*. Nous avons aussi présenté les mesures d'évaluation qui sont plus appropriées aux algorithmes de classification à partir de données déséquilibrées.

Dans le chapitre suivant, les approches proposées pour la classification des tweets seront présentées en détail.

CHAPITRE 4 : APPROCHES PROPOSEES POUR LA CLASSIFICATION DES TWEETS

INTRODUCTION

Afin de choisir les meilleures approches pour notre contribution, nous avons d'abord effectué une étude comparative entre les différents algorithmes vus en littérature, et ce, de manière expérimentale. Les tests ont été effectués sur deux collections de tweets, qui consistaient à faire une classification multiple avec Airline et une classification binaire avec Trolls (voir chapitre 4).

En se basant sur ces résultats, on propose deux approches :

- Une combinaison entre CNN et LSTM.
- Une méthode d'augmentation de données appelé *Semantic-Oversampling*.

La Figure 4.1 ci-dessous montre l'architecture générale de notre système.

4.1 Architecture général

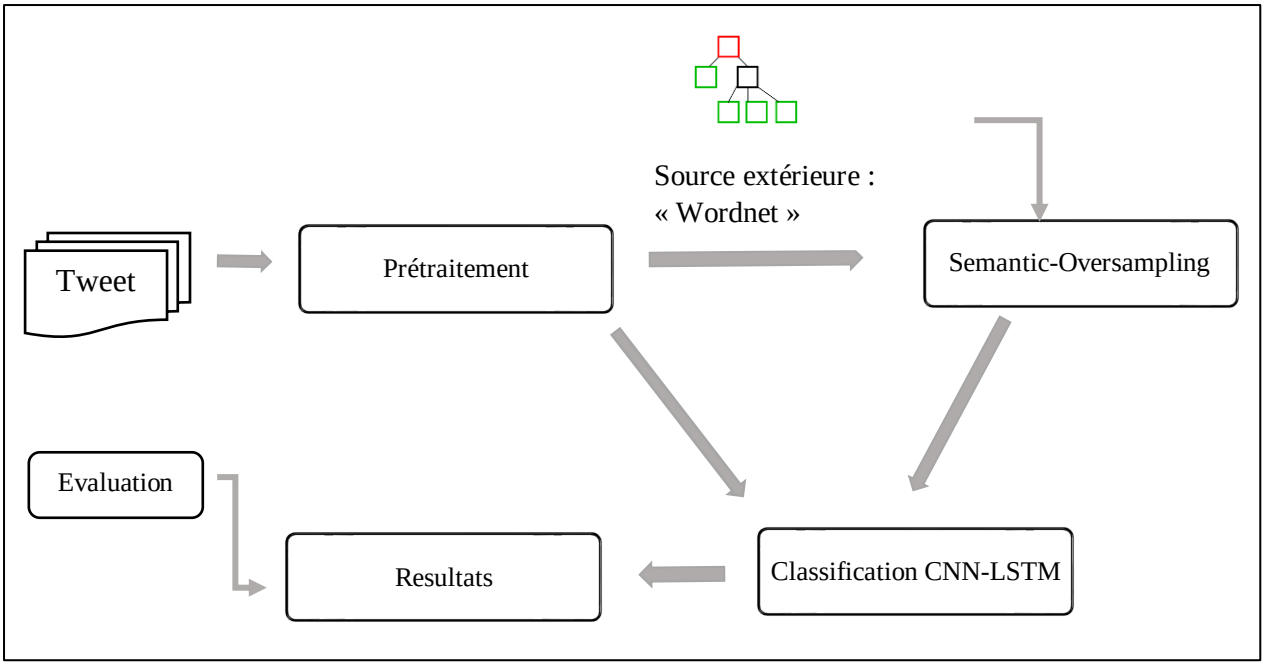

Figure 4. 1 - Architecture générale du système

4.1.1 Prétraitement

Avant de procéder à une représentation vectorielle, il est indispensable de réaliser un prétraitement, et ce, pour réduire les informations indésirables.

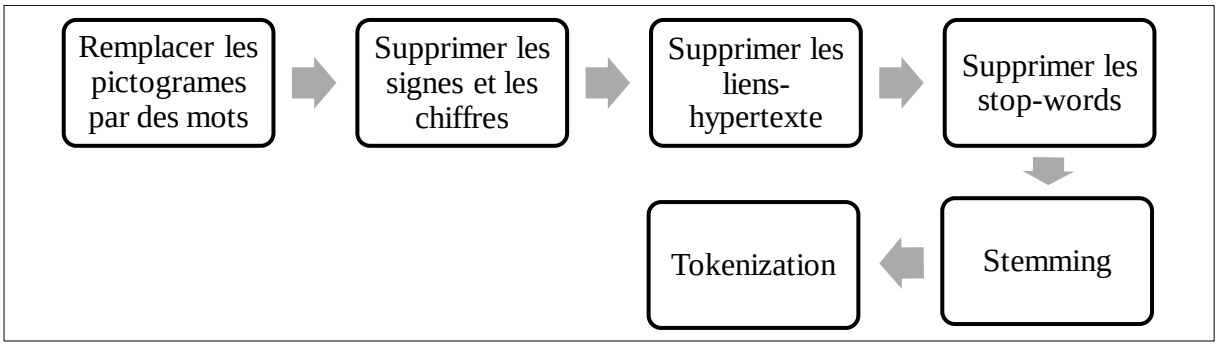

Figure 4. 2 – Schéma de la procédure pour le prétraitement des tweets

- **Procédure de prétraitement**

  - Certain pictogrammes populaires [D'Urso, 2018] sont remplacés par des mots qui les expriment :

Tableau 4. 1 - Remplacement des pictogrammes.

:)	><	xD	\m/	:(	: D	:' (	;)	^^	:o
'happy'	'happy'	'happy'	'rock'	'sad'	'laugh'	'cry'	'funny'	'happy'	'surprise'

  - Les chiffres, signes comme les hashtags (# friends => friends) ainsi que les identifiants (@username) sont supprimés.
  - Les liens-hypertexte sont supprimés car dans ce travail un lien n'apporte aucun contexte.
  - Les stops words (and, a, to, etc…) appelés aussi mots vides, n'apportent aucun sens sémantique, il est donc inutile de les conservés.
  - Un *stemming*, un procédé de transformation des flexions en leur radical (ou racine) est accompli.

    Une autre méthode appelé Lemmatisation existe, la différence est que la lemmatisation consiste à ramener un terme, quels que soient ses accords, a sa

forme la plus simple en utilisant un vocabulaire et une analyse morphologique des mots, dans le but habituel de supprimer les fins d'inflexion et de renvoyer la forme de base ou de dictionnaire d'un mot, appelé lemme.
Seulement la lemmatisation ne peut pas gérer des mots inconnus (pas present dans le dictionnaire) par exemple un Stemmer transformera « iphone et iphones » a iphon, Un *lemmatizer* laissera ces deux mots tel qu'ils sont, de ce fait, dans ce travail un Stemming est adapté.

- Tous les tweets sont notamment convertis en minuscule et ensuite découper en token (Tokenization).
- Les tweets vides (après prétraitement) sont supprimés.

4.1.2 Semantic-Oversampling

L'oversampling est une méthode d'augmentation de données. SMOTE permet de créer des données synthétiques dont les valeurs sont proches des données appartenant à la classe minoritaire, ne prenant pas en considération l'aspect sémantique des données, l'*oversampling* avec SMOTE ne serait pas tout à fait approprié pour le traitement sémantique des données textuelles.

Nous proposons une méthode appelée *Semantic-Oversampling*. L'intuition derrière cette technique est de créer des tweets sémantiquement similaire aux tweets qui appartiennent à la classe minoritaire, tout en préservant les *tokens* importants dans la classe.

Afin d'éviter le Sur-apprentissage et au lieu de dupliquer les tweets, pour générer de nouveaux tweets, nous allons utiliser des synonymes, et ceci grâce à *Wordnet*.

- **Wordnet**

  Wordnet[2] est une base de données lexicale dont le but est d'archiver, classifier et mettre en connexion le contenu sémantique et lexical de la langue anglaise.

[2] https://wordnet.princeton.edu/ [Consulté le 15 juin 2019]

Des versions de *Wordnet* pour d'autres langues existent, mais la version anglaise est cependant la plus complète à ce jour.

Wordnet fournit des *synsets* (un set de synonymes). C'est un groupe d'éléments de données considérés comme sémantiquement équivalents, ces synsets sont reliés entre eux par des relations lexicales ou taxonomiques. De cette manière pour un mot donné, il est possible de lister plusieurs synonymes, antonymes …etc.

Tableau 4. 2 - Nombre de mots, synsets et sens[3] dans *Wordnet*.

Pos	Unique Strings	Synsets	Total Word-sense pairs
Noun	117798	82115	146312
Verb	11529	13767	25047
Adjective	21479	18156	30002
Adverb	4481	3621	5580
Totals	155287	117659	206941

La composante atomique sur laquelle repose le système entier est donc le synset, un groupe de mots interchangeables, désignant un sens ou un usage particulier. Par exemple le mot « *car* en anglais » peut être définit comme suit :

- *car, auto, automobile, machine, motorcar* → *(4-wheeled motor vehicle; usually propelled by an internal combustion engine; he needs a car to get to work*)
- *car, elevator car* → *(where passengers ride up and down; the car was on the top floor*)

Chaque synset dénote donc une acception différente du mot *car*, décrite par une courte définition.

En moyenne, un mot dans *Wordnet* aurait au moins 2.15 synonymes.[4]

Exemple synonymes dans *Wordnet* :

[3] https://wordnet.princeton.edu/documentation/wnstats7wn
[4] https://gist.github.com/jeroenl/1126520

Mot : Small

Synonymes : little, minor, modest, small-scale, pocket-size, humble, low, belittled, diminished.

- **Pseudo-algorithme Semantic-Oversampling**

1) Calculer les poids Tf-idf

   Les poids Tf-idf sont calculés pour chaque tweet dans tout le corpus.

2) Choisir N (dans ce travail N est initialisé a 200)

   Pour chaque class minoritaire i :

   Parcourir les tweets et prendre les n tokens avec le plus haut poid Tf-idf.

   Top_tokens_i = tokens

3) Tant que taille (class_minoritaire) < taille (class_majoritaire)

   Pour chaque tweet de chaque class minoritaire :

   Intersection = tokens du tweet ∩ Top_tokens

   Tweet = Tweet - Intersection

   Syn_tokens= [ ]

   Si Intersection n'est pas vide :

   Pour chaque token du tweet :

   Si Token n'est pas dans Intersection :

   Tweet (Token_index) = synonym (Token)

   Tweet_syntetique.ajouter(Tweet)

- **Exemple :**

1) On calcul les poids Tf-idf des tweets suivants (On considère ici que les tweets minoritaires) :

   Tweet 1 : like very much lol.

   Tweet 2 : thanks very happy indeed.

   . . .

2) Dans cet exemple, on va considérer que les tops tokens avec un poids TF-IDF élevé sont les tokens suivants :

   Ou N = 4

   Top_tokens = {like, happy, joy, lol}.

3) On considère que taille (class_majoritaire) =5 et que taille (class_minoritaire) =2.

   Pour chaque tweet de la classe minoritaire on crée un tweet synthétique :

   Intersection = {like, happy, joy, lol} ∩ {tweet1}.

   Intersection = {like, happy, joy, lol} ∩ {like, very, much, lol}.

   Intersection = {like, lol}.

   Pour chaque *token* qui n'appartient pas à l'intersection, on choisit un synonyme aléatoire.

   very → a-lot.

   much → amount.

   Nouveau_tweet1 = <u>like</u> a-lot amount <u>lol</u>.

   Intersection = {like, happy, joy, lol} ∩ {tweet2}.

   Intersection = {like, happy, joy, lol} ∩ {thanks, very, happy, indeed}.

   Intersection = {happy}.

   Pour chaque *token* qui n'appartient pas à l'intersection, on choisit un synonyme aléatoire.

   Thanks → thank-you.

   Very → high-degree.

   Indeed → truly.

   Nouveau_tweet2 = thank-you high-degree <u>happy</u> truly.

   Intersection = {like, happy, joy, lol} ∩ {tweet1}.

   Intersection = {like, happy, joy, lol} ∩ {like, very, much, lol}.

   Intersection = {like}.

   Pour chaque *token* qui n'appartient pas à l'intersection, on choisit un synonyme aléatoire.

   very → extremely.

   much → amount.

   Nouveau_tweet3 = <u>like</u> extremely amount <u>lol</u>.

   taille (class_majoritaire) = taille (class_minoritaire)=5.

   Voici les tweets synthétiques obtenu à la fin :

Nouveau_tweet1 = like lot amount lol.

Nouveau_tweet2 = thank-you high-degree happy truly.

Nouveau_tweet3 = like extremely amount lol.

4.1.3 Classification CNN+LSTM

Une combinaison entre CNN et LSTM est proposée. La motivation derrière ce travail, serait de bénéficier des propriétés des deux couches CNN et LSTM. Comme il a été expliqué précédemment, les CNNs peuvent extraire des caractéristiques locales importantes, et les LSTMs soutiennent la prédiction de séquence dans le temps.

- **Architecture CNN-LSTM**

  La figure 4.3 ci-dessous montre l'architecture de notre combinaison CNN-LSTM.

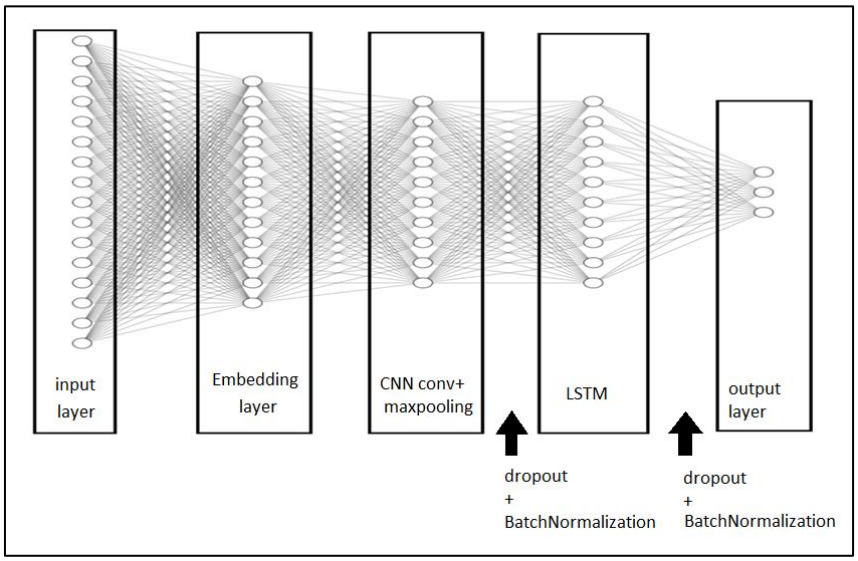

Figure 4. 3 – Architecture de la combinaison CNN-LSTM

- Une couche *embedding* est d'abord introduite. Elle va servir à améliorer la capacité des réseaux à apprendre à partir de données textuelles, en représentant ces données sous forme de vecteurs de dimension inférieure.

- Une couche de convolution CNN suivi d'un *max-pooling* est ajoutée. Le *max-pooling* est utilisé en raison du fait que ça considère que les caractéristiques les plus importantes, contrairement à l'*average-pooling* qui considère la totalité de l'input.

- Une couche LSTM est ajouté dont le pseudo-algorithme (d'une unité) est le suivant :

  - **Pseudo algorithme :**

    Def lstmcell (prev_ct, prev_ht, input) :

    1. Concaténation de l'état caché précèdent et l'input actuel

    combine = prev_ht + input

    2. dans cette couche on supprime les informations dont on n'a pas besoin.

    ft = forget_layer (combine)

    3. une couche de candidats est créée, contient les valeurs possibles que peut contenir l'état de notre cellule

    candidate = condidate_layer (combine)

    4. de même, combine passe dans la couche input. cette couche va décider quelles infos ajouter dans le nouvel état.

    it = input_layer (combine)

    5. après avoir calculé *forget layer*, candidate layer et *input layer*, l'état est calculé avec ces vecteurs et l'état précèdent.

    ct = prev_ct * ft+ candidate * it

    6. On calcul l'output

    ot = output_layer (combine)

    7. le nouvel état caché est calculé avec notre output multiplié par le nouvel état caché.

    ht = ot * tanh (ct)

    ht, ct sont retournées

- Dans ce modèle, La couche de normalisation par lots (*Batch-Normalization*) introduite initialement dans [Ioffe & Szegedy, 2015] est ajoutée. C'est une couche qui permet. d'améliorer la vitesse, les performances et la stabilité des réseaux de neurones artificiels, et ceci, en normalisant l'output de la fonction d'activation. Durant l'apprentissage, les couches de *Batch-Normalization* apprennent des paramètres (un facteur d'échelle et un biais) permettant d'ajuster cette normalisation. Ces couches sont plus utilisées dans les réseaux de neurones profonds.

- Un Dropout est ajouté pour éviter le sur-apprentissage.

Il a été jugé plus intuitive d'introduire une couche CNN avant la couche LSTM, et ce, pour extraire des caractéristiques importantes ensuite de « s'en souvenir » grâce à LSTM.

4.1.4 Evaluation des résultats

Pour l'évaluation de nos approches, on va utiliser l'*accuracy*, la précision, le rappel (*recall*) et le F-score.

CONCLUSION

Dans ce chapitre, nous avons proposé une nouvelle architecture de réseaux de neurones pour la classification sémantique des tweets. La méthode consiste à faire une combinaison entre deux couches de réseaux de neurones CNN et LSTM. Nous avons notamment proposé une technique de sur-échantillonnage appelé *Semantic-Oversampling*.

Dans le prochain chapitre, différents tests et résultats seront exposés en détail.

CHAPITRE 5 : EXPERIMENTATION ET COMPARAISON DES RESULTATS

INTRODUCTION

Dans ce chapitre, les résultats de différents tests établis en utilisant les techniques présentées dans les chapitres précédents sont présentés.

5.1 Jeu de données

Dans ce travail, deux dataset sont utilisés pour comparer nos approches pour la classification des tweets.

- **Trolls Detection**

  Trolls Detection[5] contient 20 001 tweets étiquetés, avec deux classes, non agressive : 12179 tweets et agressive : 7822 tweets.

- **Airline**

  Airline[6] contient 14 640 tweets étiquetés, et possède trois classes, « negative » avec 9178 tweets, « neutral » avec 3099 et « positive » avec 2363 tweets.

5.2 Protocol expérimental

Les algorithmes sont implémentés en utilisant Python 3.6.

Les librairies (modules) utilisées pour la réalisation de ce projet :

- NLTK (Natural Language Toolkit).
- Gensim (pour l'implémentation des *Topic-Model* et word2vec).
- Keras (pour l'implémentation des réseaux de neurones).
- Sklearn (pour l'implémentation de K-means).

[5] https://www.kaggle.com/dataturks/dataset-for-detection-of-cybertrolls [consulté le 15 Avril 2019]
[6] https://www.kaggle.com/crowdflower/twitter-airline-sentiment#Tweets.csv [consulté le 10 Février 2019]

5.2.1 Prétraitement

Afin de nettoyer les tweets pour ne laisser que les informations jugés importantes, nous utilisons les techniques suivantes :

- Remplacer les pictogrammes par des mots (représentant l'émotion) (librairie String).
- Remplacer les chiffres et les signes … (librairie String).
- Remplacer les liens liens-hypertextes … (librairie String).
- Supprimer les stopwords en utilisant la librairie NLTK qui fournit la liste des stopwords dans la langue anglaise.
- Faire un Stemming en utilisant le Stemmer disponible dans la librairie NLTK.
- Convertir les tweets en minuscule (librairie String).
- Faire une Tokenisation (librairie String).

5.2.2 Implémentation et tests réalisés

Différents tests ont été effectués sur les deux corpus Trolls et Airline et ceci pour :

- Le dataset original (déséquilibré).
- Le dataset après Undersampling.
- Le dataset après Oversampling.

Afin d'évaluer ces approches, l'*accuracy*, la précision, le recall ainsi que le F-score sont utilisés.

5.2.2.1 Etude comparative entre algorithmes de classification

Nous avons testé différents algorithmes pour la classification des tweets afin de choisir celui avec les meilleurs résultats pour notre approche.

Des approches classiques de classification comme K-means, LDA ainsi que LSA son implémentées et testés sur les deux corpus Trolls et Airline.

Deux réseaux de neurones sont notamment implémentés :

- Réseau de neurones a convolution de Yoon.

- Réseaux de neurones récurrents en utilisant LSTM.

❖ **Liste des paramètres et notations**

Avant d'exposer les résultats, cette partie explique les différentes notations et paramètres utilisées pour les tests.

➢ **LDA**

- Nombre de topic k (a été initialisé).
- Chunksize : 2000 (laissé par défaut).
- Passes : modifié dans chaque test.
  Exprime combien de fois l'algorithme devrait passer sur tout le corpus.
  Exemple : si le nombre de tweets =50 000, Chunksize =1000 et Passes =2 :

Passe 1 :	Passe 2 :
#1 documents 0-9,999	#6 documents 0-9,999
#2 documents 10,000-19,999	#7 documents 10,000-19,999
#3 documents 20,000-29,999	#8 documents 20,000-29,999
#4 documents 30,000-39,999	#9 documents 30,000-39,999
#5 documents 40,000-49,999	#10 documents 40,000-49,999

- Itération : modifié dans chaque test.

➢ **LSA**

- Nombre de topic k (a été initialisé).
- Chunksize : 2000 (laissé par défaut).

➢ **K-means**

- Nombre de clusters k (a été initialisé).
- Itération : modifié dans chaque test.
- random_state : None
  L'initialisation des centroides est Random.
- Distance : Euclidienne.

- Word2vec :
    - Window-size : Selon la taille maximale (du tweet) dans le corpus.
    - Epoch : 10
    - Batchsize : 100 (par défaut)
    - Model : CBow

➢ **Réseau de neurones avec LSTM**

- Lstm-output : nombre d'unité (modifié dans chaque test).
- Activation : tanh (fonction d'activation par défaut).
- recurrent_dropout : 0.2 (fixé).
  Le recurrent_dropout est un dropout entre chaque unité.
- Dropout : modifié dans chaque test.
- Optimizer : Adam (mentionné si modifié).

➢ **CNN de Yoon**

- embedding_dim :128 nombre d'unité (laissé par défaut).
- num_filters : 128 (laissé par défaut).
- Kernel-size : [3, 4,5] (laissé par défaut)
  La taille de la fenêtre 3x3, 4x4, 5x5.
- Activation (couche CNN) : Relu (laissé par défaut).
- Batch-size : modifié dans chaque test.
- Optimizer : Adam (laissé par défaut).
- Dropout : 0.5 (laissé par défaut).
- Activation (dernière couche) : Softmax (laissé par défaut).
- Epoch : modifié dans chaque test.

➢ **Oversampling SMOTE**

- Nombre des K plus proches voisins : 6.
- Distance : Euclidienne.

➢ **Division du dataset**

- Le dataset a été divisé en 15% pour les tests et 85% pour l'apprentissage (de manière aléatoire).
- Random_state (de division) =50.
- Random_state (de validation) =50.
- Pour les réseaux de neurones, le dataset a été divisé en 15% pour les tests. 33% de l'ensemble d'apprentissage est utilisé pour l'ensemble de validation pour les réseaux de neurones.

5.2.2.2 Evaluation de notre approche pour l'augmentation des données

Nous avons testé SMOTE avec des vecteurs TF-IDF (en plus des vecteurs Word2vec pour K-means), un *undersampling* (ou la classe majoritaire a été réduite jusqu'à ce que les classes deviennent équilibrées) sur les deux dataset Trolls et Airline.

Nous avons notamment testé notre approche *Semantic-Oversampling* sur les deux dataset Trolls et Airline et comparer avec SMOTE.

❖ **Liste des paramètres et notations**

➢ **Semantic-Oversampling**

N = 200 (Les N premiers tokens).
Min-count = 0.01 (Dans la fonction TF-IDF : ignorer les termes qui apparaissent dans moins de 1% des documents, mentionné si appliqué).

5.2.2.3 Evaluation de notre approche pour la classification des tweets (CNN+LSTM)

La combinaison CNN+LSTM est implémenté et testé sur les deux corpus Trolls et Airline.

Pour évaluer notre approche CNN+LSTM, nous l'avons comparé avec toutes les autres approches cités en haut.

❖ **Liste des paramètres et notations**

➢ **CNN+LSTM**

Les mêmes paramètres de LSTM en modifiant/ajoutant :

- Kernel-size : 4x4. (mentionné si modifié)
- Pool-size : 2. (mentionné si modifié)
- Num_filters (Cnn_dim) : 1024.
- Dropout 1 : modifié dans chaque test.
- Dropout 2 : modifié dans chaque test.
- BatchNormalization1 : fixé.
- BatchNormalization2 : fixé.
- Optimizer : Adam (mentionné si modifier).

5.3 Résultat

Nous exposons dans ce qui suit les résultats de l'étude comparative des algorithmes de classification supervisée / non supervisée vu en littérature. Par la suite nous exposons les résultats de l'approche proposée CNN-LSTM.

5.3.1 Résultats algorithmes vu en littérature

Nous exposons d'abord une récapitulation des résultats des tests sur les algorithmes vus en littérature.

Tableau 5. 1 – Récapitulation meilleurs résultats avec corpus Trolls

	F-Score			Accuracy		
	Original	Undersamp	SMOTE	Original	Undersamp	SMOTE
LDA	37.41	57.62	**58.59**	**60.71**	56.06	54.82
LSA	37.62	**51.44**	47.82	**59.91**	50.09	30.43
K-means	37.67	55.35	**54,52**	**60.44**	52.59	55.11
CNN-Yoon	86.33	83.17	**89.21**	86.6	81.19	**89.1**
LSTM	81.68	81.45	**87.86**	82.01	82.28	**87.27**

Tableau 5. 2 – Récapitulation meilleurs résultats avec corpus Airline

	F-Score			Accuracy		
	Original	Undersamp	SMOTE	Original	Undersamp	SMOTE
LDA	25.70	40.83	**42.81**	**62.79**	58.86	37.43
LSA	**46.06**	44.33	41.99	**67.85**	48.13	33.21

K-means	25.70	39.34	**42.03**	62.74	58.09	**65.11**
CNN-Yoon	69.02	**69.59**	**70.69**	**77.04**	71.91	76.01
LSTM	**72.01**	70.27	71.32	**78.64**	75.47	76.38

- **Discussion et commentaires :**

  - Pour les deux dataset, Airline et Trolls, les meilleurs résultats sont obtenus avec les réseaux de neurones.

  - Afin d'évaluer les performances des algorithmes en considérant le rappel et la précision, le F-score est utilisé. Dans les tableaux, le F-score connait une grande amélioration après un *Over*/*Undersampling*, car les algorithmes ont pu détecter des classes minoritaires.

  - Dans ce cas d'étude, les réseaux de neurones renvoient un F-score élevé même dans le cas des dataset déséquilibrés, ceci est potentiellement dû au fait que l'apprentissage soit supervisé. Même pour les réseaux de neurones, le F-score peut être amélioré d'avantage avec des méthodes de *sampling*.

  - Dans cette étude, SMOTE a surpassé le sous-échantillonnage *undersampling* dans la plupart des tests (4 algorithmes /5 pour Trolls) et (3 algorithmes /5 pour Airline).

5.3.2 Résultats approche CNN-LSTM

Nous exposons dans ce suit les résultats de l'approche proposée CNN-LSTM.

5.3.2.1 Dataset Trolls

5.3.2.1.1 Résultat CNN-LSTM (Dataset Original)

Tableau 5. 3 – Résultat CNN-LSTM sur dataset Trolls Original.

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	49.20%	50.85%	70.05%	58.92%

epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	90.04%	91.12%	88.30%	89.69%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	86.37%	85.94%	85.61%	85.78%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256% Dropout1 = 0.6 Dropout2 = 0.6	**90.67%**	**92.72%**	**88.53%**	**90.58%**
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	89.62%	90.72	87.74%	89.21%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = none Dropout2 = 0.6	87.90	88.14%	86.44%	87.28%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	84.94%	84.16%	84.77%	84.46%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	89.47%	90.11%	87.89%	88.99%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adamax /adam/Adadelta	20 epoch : 60.31% -- -- -- -- -- 20 epoch : 88.80% -- -- -- -- -- 20 epoch : 88.10% -- -- -- -- -- 20 epoch : 86.27%	30.15% -- -- -- -- 89.48% -- -- -- -- 88.82% -- -- -- -- 85.77%	50.00% -- -- -- -- -- 87.12% -- -- -- -- -- 86.31% -- -- -- -- -- 85.43%	37.62% -- -- -- -- 88.28% -- -- -- -- 87.55% -- -- -- -- 85.60%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	81.71	81.11%	82.24%	81.67%

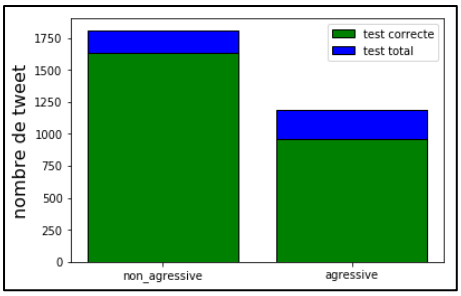

Figure 5. 1 - Test correct par classe CNN-LSTM (Trolls Original)

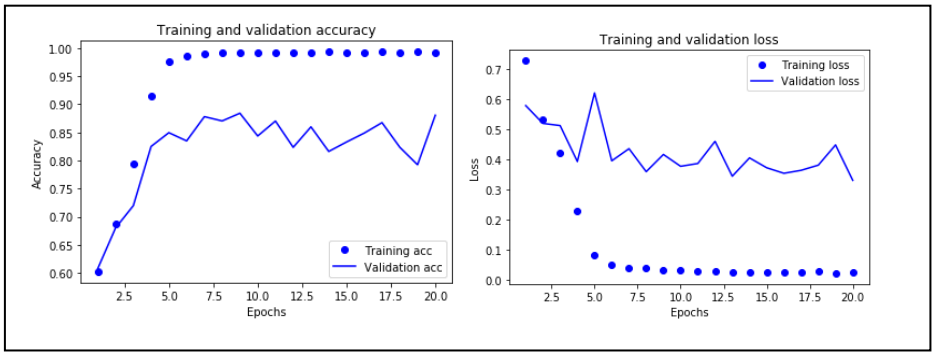

Figure 5. 2 - L'accuracy et perte de validation (avec Adamax)

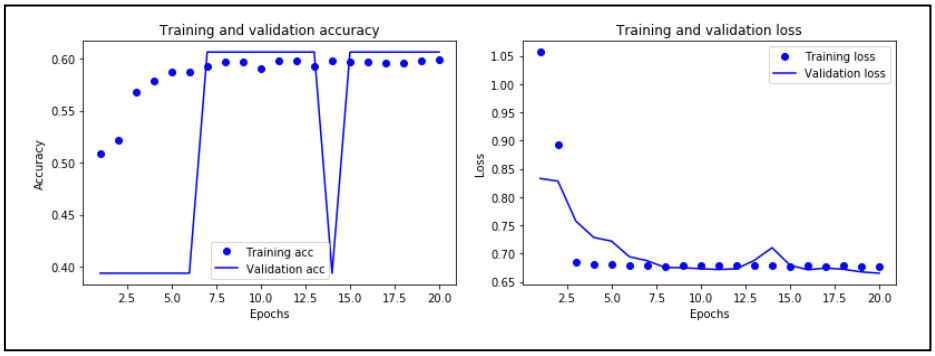

Figure 5. 3 - L'accuracy et perte de validation (avec SGD)

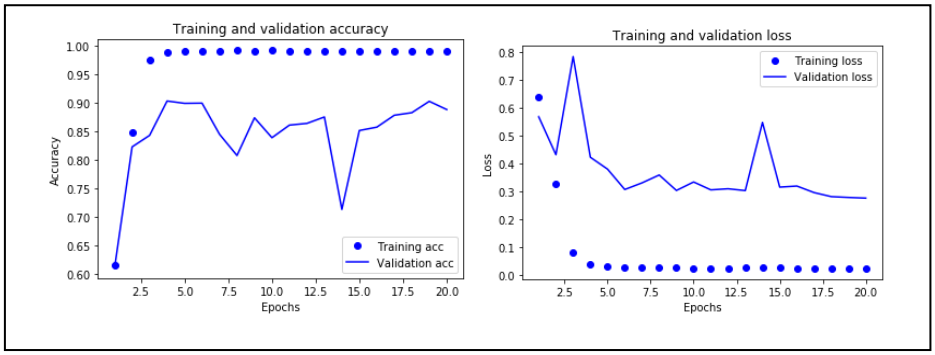

Figure 5. 4 - L'accuracy et perte de validation (avec Adam)

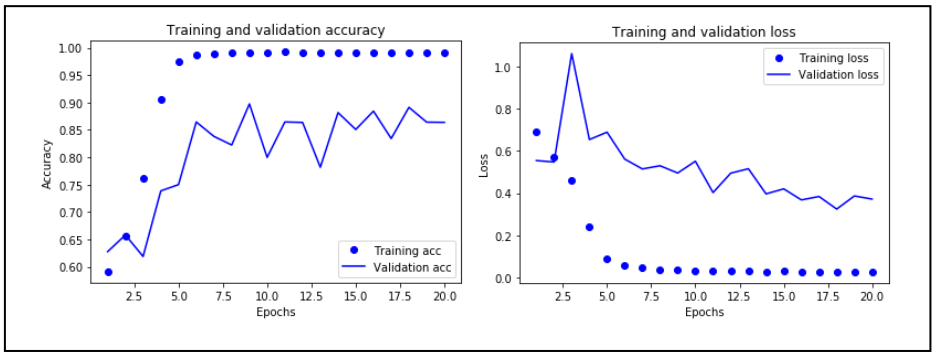

Figure 5. 5 - L'accuracy et perte de validation (avec Adadelta)

5.3.2.1.2 Résultat CNN-LSTM (Dataset après Undersampling)

Tableau 5. 4 - Résultat CNN-LSTM sur dataset Trolls après Undersampling.

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	60.26%	53.20%	50.56%	51.85%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	76.53%	78.29%	78.29%	78.73%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	74.85%	73.67%	73.11%	73.39%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	**82.43%**	81.54%	82.35%	81.94%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	80.89%	81.03%	82.50%	81.76%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 128	80.75%	80.79	82.26%	81.52%

lstm_output = 256 Dropout1 = none Dropout2 = 0.6				
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	80.62%	80.70%	82.16%	81.42%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	77.36%	78.82%	79.85%	79.34%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adamax adam/Adadelta	30 epoch : 59.99% -- -- -- -- -- 30 epoch : 78.03% -- -- -- -- -- 30 epoch : 82.03% -- -- -- -- -- 30 epoch : 81.89%	57.69% -- -- -- -- -- 79.13% -- -- -- -- -- 81.81% -- -- -- -- -- **81.84%**	57.47% -- -- -- -- -- 80.30% -- -- -- -- -- 83.31% -- -- -- -- -- **83.36%**	57.58% -- -- -- -- 79.71% -- -- -- -- 82.55% -- -- -- -- **82.60%**
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	30 epoch : 79.86%	80.31%	81.71%	81.00%

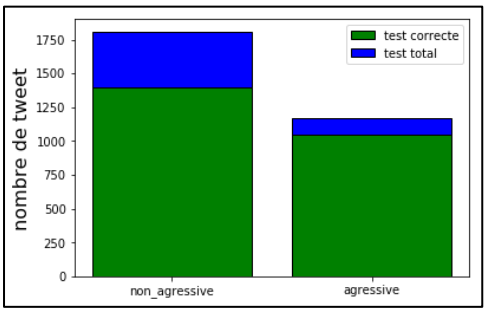

Figure 5. 6 - Test correct par classe CNN-LSTM (Trolls Undersampling)

5.3.2.1.3 Résultat CNN-LSTM (Dataset après Oversampling)

5.3.2.1.3.1 Oversampling avec SMOTE

Tableau 5. 5 - Résultat CNN-LSTM sur dataset Trolls Apres Oversampling (SMOTE).

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**

epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	63.78%	63.23%	55.68%	59.22%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	73.51%	78.86%	77.89%	78.37%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	68.98%	82.53%	60.39%	69.75%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	86.67%	86.38%	88.15%	87.25%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	88.94%	88.30%	89.93%	89.11%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = none Dropout2 = 0.6	**90.40%**	**89.68%**	**91.12%**	**90.39%**
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	87.24%	86.87%	88.65%	87.75%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	87.20%	86.87%	88.66%	87.76%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adamax /adam/Adadelta	20 epoch : 56.58% -- -- -- -- -- 20 epoch : 87.30% -- -- -- -- -- 20 epoch : 89.00% -- -- -- -- -- 20 epoch : 89.84%	58.96% -- -- -- -- 86.92% -- -- -- -- 88.41% -- -- -- -- 89.14%	59.07% -- -- -- -- 88.69% -- -- -- -- 90.11% -- -- -- -- 90.70%	59.01% -- -- -- -- 87.79% -- -- -- -- 89.25% -- -- -- -- 89.92%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256	89.84%	89.14%	90.70%	89.92%

lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde				

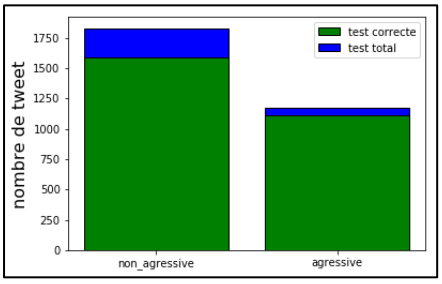

Figure 5. 7 - Test correct par classe CNN-LSTM (Trolls SMOTE)

5.3.2.1.3.2 Semantic-Oversampling

Tableau 5. 6- Résultat CNN-LSTM sur dataset Trolls après Oversampling (Synonymes).

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	60.97%	57.71%	55.16%	56.41%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	83.79%	82.93%	83.62%	83.27%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	83.52%	82.50%	83.33%	82.91%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	81.67%	81.20%	82.60%	81.89%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5 -- -- -- Min_count = 0.01	**86.03%** -- -- -- -- -- 81.82%	**85.27%** -- -- -- -- -- 8130%	**85.55%** -- -- -- -- -- 82.62%	**85.41%** -- -- -- -- -- 81.98%
epoch : 10 batch= 512	83.82%	83.09%	84.30%	83.69%

Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = none Dropout2 = 0.6				
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	80.65%	80.54%	81.99%	81.26%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	84.19%	83.45%	83.31%	83.38%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adamax /adam/Adadelta	30 epoch : 59.46% -- -- -- -- -- 30 epoch : 83.77% -- -- -- -- -- 30 epoch : 84.51% -- -- -- -- -- 30 epoch .: 85.95%	58.06% -- -- -- -- 83.04% -- -- -- -- 83.68% -- -- -- -- 85.21%	58.26% -- -- -- -- -- 84.29% -- -- -- -- -- 84.69% -- -- -- -- -- 85.45%	58.16% -- -- -- -- 83.66% -- -- -- -- 84.19% -- -- -- -- 85.33%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	82.23%	81.70%	83.10%	82.39%

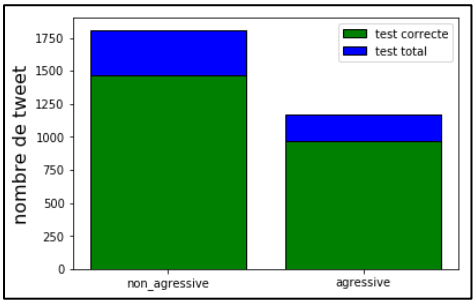

Figure 5. 8 - Test correct par classe CNN-LSTM (Trolls Semantic-Oversampling)

5.3.2.2 Dataset Airline

5.3.2.2.1 Résultats CNN-LSTM (Dataset Original)

Tableau 5. 7 - Résultat CNN-LSTM sur dataset Airline Original.

paramètres	Accuracy (%)	Precision (%)	Recall(%)	F-score (%)

epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	69.95%	67.46%	50.78%	57.94%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	77.46%	72.57%	68.64%	70.55%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	76.87%	71.52%	68.43%	69.94%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	76.37%	70.93%	70.23%	70.58%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Kernel : 3x3 Pooling : 4	**77.87%**	73.43%	68.67%	**70.97%**
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = none Dropout2 = 0.6 Kernel : 3x3 Pooling : 4	77.14%	71.19%	69.60%	70.39%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	76.64%	70.33%	67.11%	68.68%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	76.46%	70.18%	68.61%	69.39%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adamax/ adam/Adadelta	20 epoch : 70.58% -- -- -- -- -- 20 epoch : 75.36% 40 epoch : 73.32% 80 epoch : 76.05% -- -- -- -- -- 20 epoch : 75.96% -- -- -- -- --	68.13% -- -- -- -- 68.50% 70.98% 69.92% -- -- -- -- 69.38% -- -- -- --	51.96% -- -- -- -- 68.29% 66.43% 68.10% -- -- -- -- 71.04% -- -- -- --	58.96% -- -- -- -- 68.40% 68.63% 69.00% -- -- -- -- 70.20% -- -- -- --

	20 epoch : 74.45%	66.92%	67.55%	67.23%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	73.72%	66.59%	64.28%	65.41%

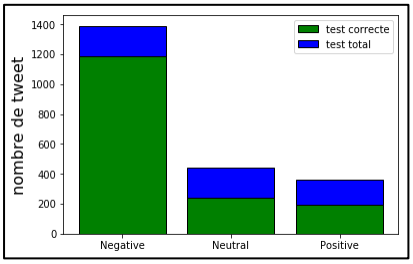

Figure 5. 9 - Test correct par classe CNN-LSTM (Airline Original)

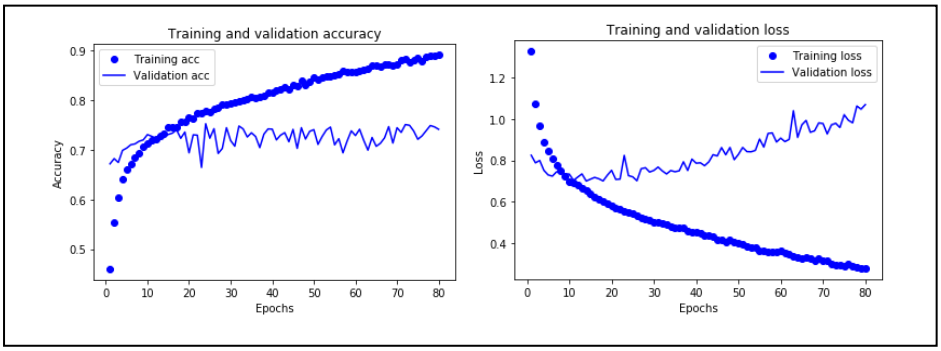

Figure 5. 10 - L'accuracy et la perte training et validation (avec dropout)

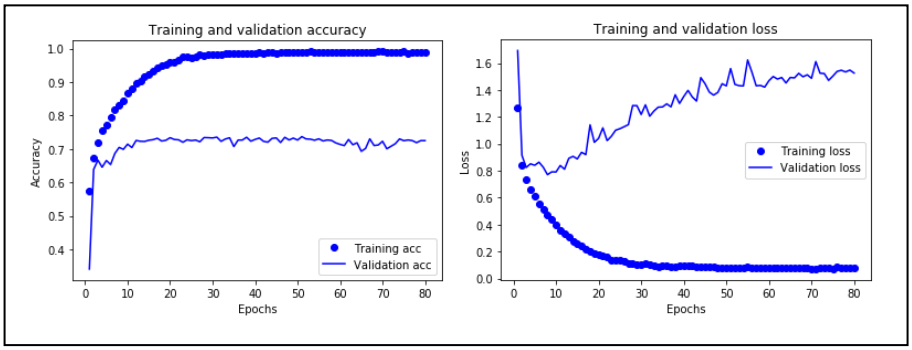

Figure 5. 11 - L'accuracy et la perte training et validation (sans dropout)

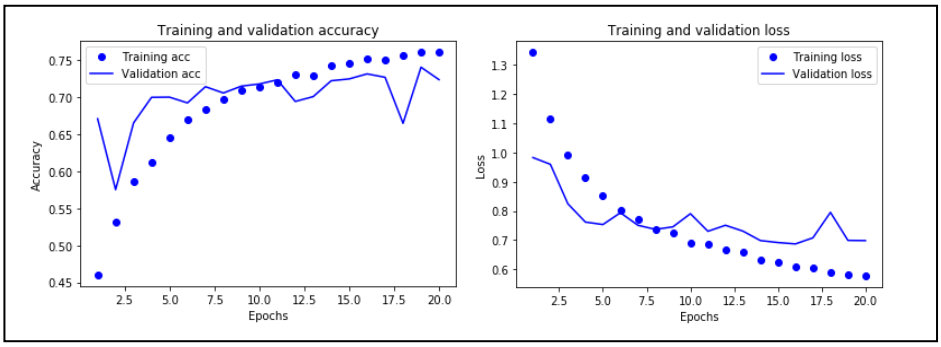

Figure 5. 12 - L'accuracy et la perte training et validation (Adadelta)

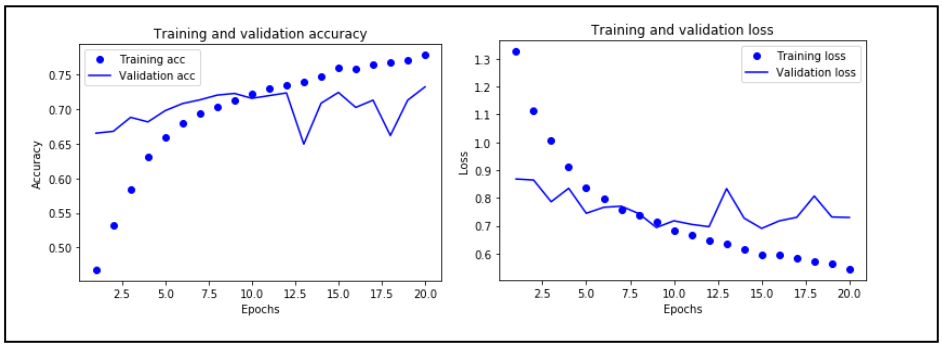

Figure 5. 13 - L'accuracy et la perte training et validation (Adam)

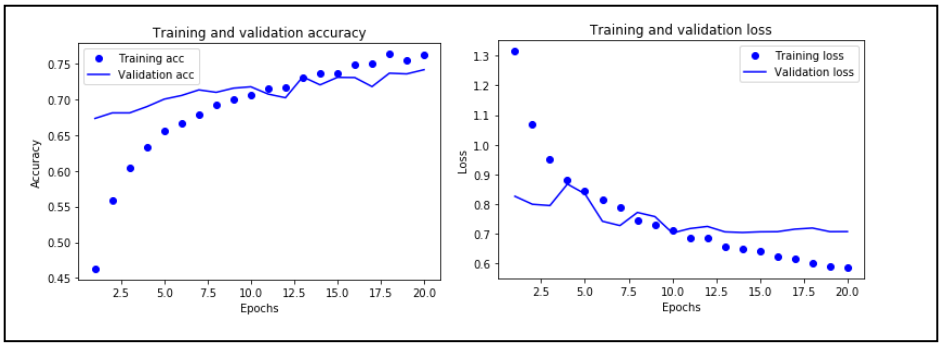

Figure 5. 14 - L'accuracy et la perte training et validation (Adamax)

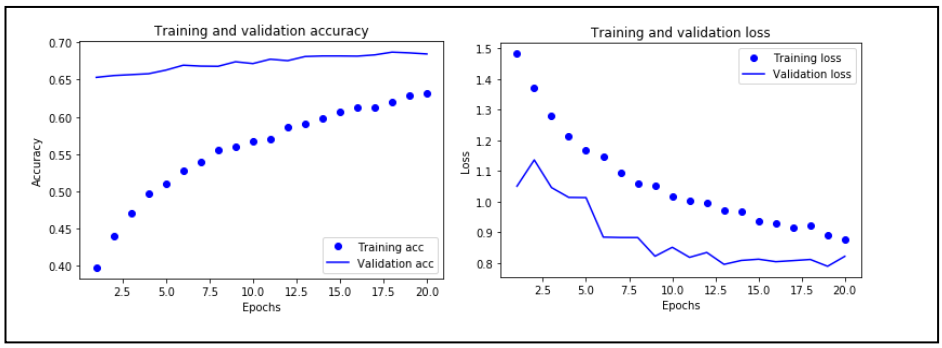

Figure 5. 15 - L'accuracy et la perte training et validation (SGD)

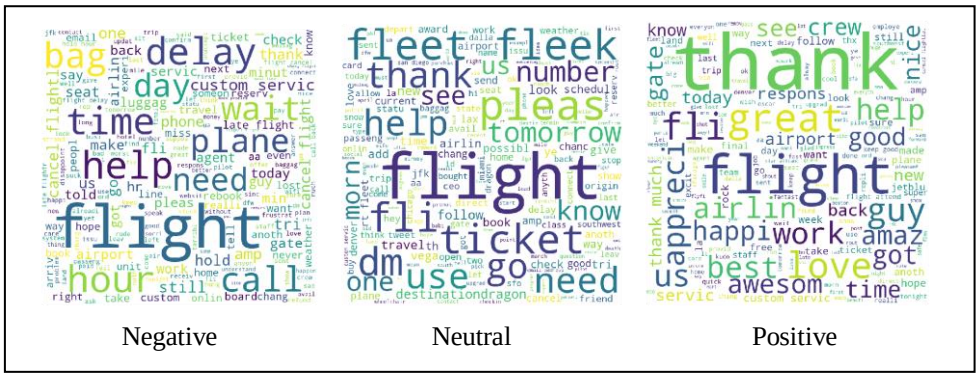

Figure 5. 16 - Nuages des mots les plus fréquents dans chaque classe CNN-LSTM (Dataset Airline Original).

5.3.2.2.2 Résultats CNN-LSTM (Dataset après Undersampling)

Tableau 5. 8– Résultat CNN-LSTM sur dataset Airline après Undersampling.

paramètres	Accuracy (%)	Precision(%)	Recall(%)	F-score (%)
epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	59.96%	55.16%	57.20%	56.16%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	73.10%	66.60%	70.06%	68.29%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	**75.70%**	**69.74%**	72.45%	**71.07%**
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	71.77%	66.69%	72.59%	69.52%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	71.45%	66.53%	**72.91%**	69.58%

epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = none Dropout2 = 0.6	72.28%	67.97%	70.95%	69.43%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	71.50%	66.36%	70.68%	68.45%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	70.73%	65.27%	70.15%	67.62%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adamax/ adam/Adadelta	30 epoch : 70.27% -- -- -- -- -- 30 epoch : 71.04% -- -- -- -- -- 30 epoch : 70.27% -- -- -- -- -- 30 epoch : 72.18%	65.38% -- -- -- -- -- 64.65% -- -- -- -- -- 65.36% -- -- -- -- -- 65.60%	63.75% -- -- -- -- -- 68.85% -- -- -- -- -- 70.02% -- -- -- -- -- 70.38%	64.55% -- -- -- -- -- 66.68% -- -- -- -- -- 67.61% -- -- -- -- -- 67.91%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	71.87%	65.37%	69.43%	67.34%

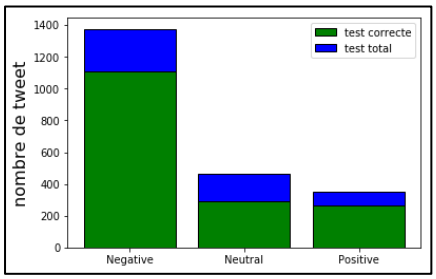

Figure 5. 17 - Test correct par classe CNN-LSTM (Airline Undersampling)

5.3.2.2.3 Résultats CNN-LSTM (Dataset Apres Oversampling)

5.3.2.2.3.1 Oversampling avec SMOTE

Tableau 5. 9 – Résultat CNN-LSTM sur dataset Airline après Oversampling (SMOTE).

paramètres	Accuracy (%)	Precision(%)	Recall(%)	F-score (%)

epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	68.54%	62.63%	65.22%	**63.90%**
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	69.86%	61.51%	61.56%	61.54%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	63.98%	58.81%	61.20%	59.98%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	70.41%	61.27%	59.87%	60.56%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	70.36%	61.85%	61.75%	61.80%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = none Dropout2 = 0.6	69.13%	60.26%	59.35%	59.80%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	**70.36%**	63.65%	58.32%	60.87%
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	69.95%	60.63%	59.08%	59.84%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Optim:SGD/Adama x/adam/Adadelta	20 epoch : 66.44% -- -- -- -- -- 20 epoch : 68.63 % -- -- -- -- -- 20 epoch : 68.67% -- -- -- -- -- 20 epoch : 70.09%	62.42% -- -- -- -- -- 59.71% -- -- -- -- -- 59.52% -- -- -- -- -- 61.42%	61.03% -- -- -- -- 60.48% -- -- -- -- 60.55% -- -- -- -- 61.84%	61.71 % -- -- -- -- 60.09% -- -- -- -- 60.03% -- -- -- -- 61.63%

batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	69.72 %	69.72%	60.95%	61.30%

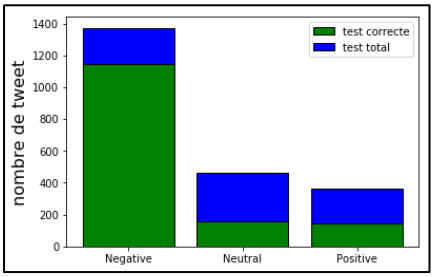

Figure 5. 18 - Test correct par classe CNN-LSTM (Airline SMOTE)

5.3.2.2.3.2 Semantic-Oversampling

Tableau 5. 10 – Résultat CNN-LSTM sur dataset Airline après Oversampling (Synonymes).

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	76.61%	70.04%	69.52%	69.78%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	76.93%	71.13%	69.81%	70.46%
epoch : 5 batch= 512 Cnn_dim = 1024 embed_dim = 128 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5	75.38%	68.52%	**70.28%**	69.39%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6	75.38%	68.34%	69.23%	68.79%
epoch : 5 batch= 60 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 -- -- -- --	**77.61%** -- -- -- --	**72.98%** -- -- -- --	68.18% -- -- -- --	**70.50%** -- -- -- --

Min-count = 0.01	76.50%	70.90%	67.10%	68.94%
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = none Dropout2 = 0.6	77.47%	72.76	68.03	70.31
epoch : 10 batch= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = none Dropout2 = none	77.11%	71.13	68.66	69.87
epoch : 15 batch= 512 Cnn_dim = 1024 embed_dim = 1024 lstm_output = 256 Dropout1 = 0.6 Dropout2 = none	76.42%	70.97	67.11	68.99
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.5 Dropout2 = 0.5 Optim:SGD/Adamax /adam/Adadelta	30 epoch : 74.42% -- -- -- -- -- 30 epoch : 75.51% -- -- -- -- -- 30 epoch : 76.88% -- -- -- -- -- 30 epoch : 75.97%	68.32% -- -- -- -- 68.97% -- -- -- -- 71.20% -- -- -- -- 69.56%	66.39% -- -- -- -- -- 67.75% -- -- -- -- -- 68.72% -- -- -- -- -- 69.33%	67.34% -- -- -- -- 68.35% -- -- -- -- 69.94% -- -- -- -- 69.45%
batchsize= 512 Cnn_dim = 1024 embed_dim = 256 lstm_output = 256 Dropout1 = 0.6 Dropout2 = 0.6 Activation : sigmoïde	74.69%	68.78%	67.21%	67.98%

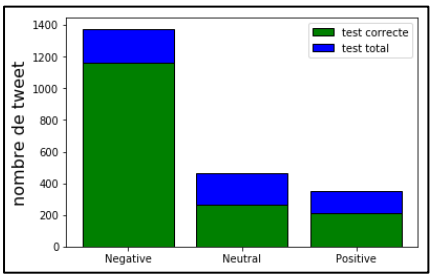

Figure 5. 19 - Test correct par classe CNN-LSTM (Airline Semantic-Oversampling)

5.3.3 Comparaison de nos approches avec les algorithmes de l'état de l'art

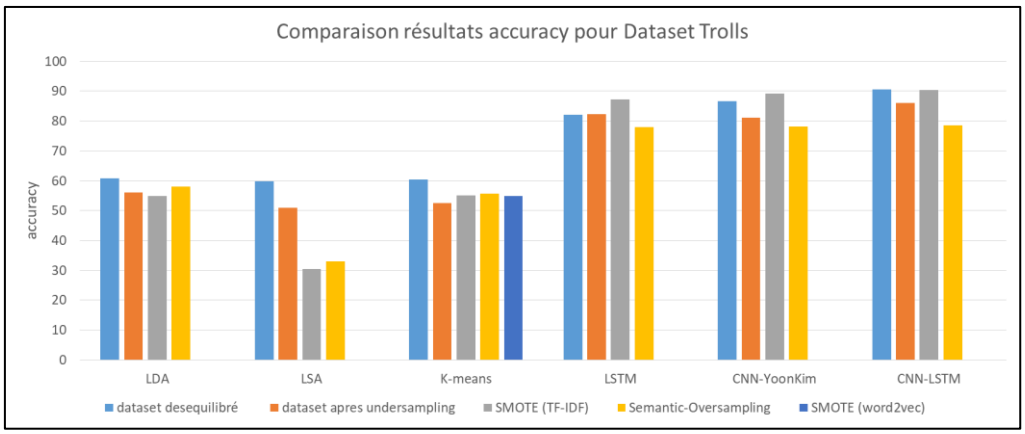

Figure 5. 20 - Comparaison de l'accuracy pour les différentes approches (Trolls)

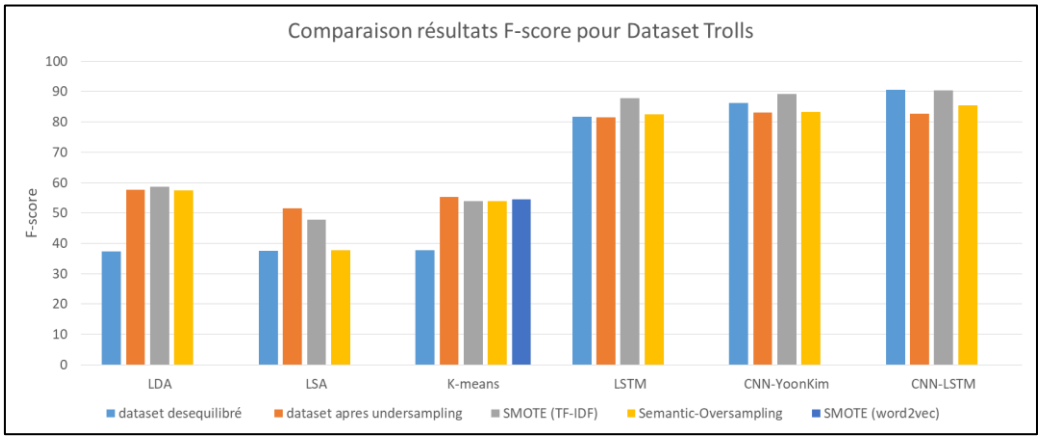

Figure 5. 21 - Comparaison du F-Score pour les différentes approches (Trolls)

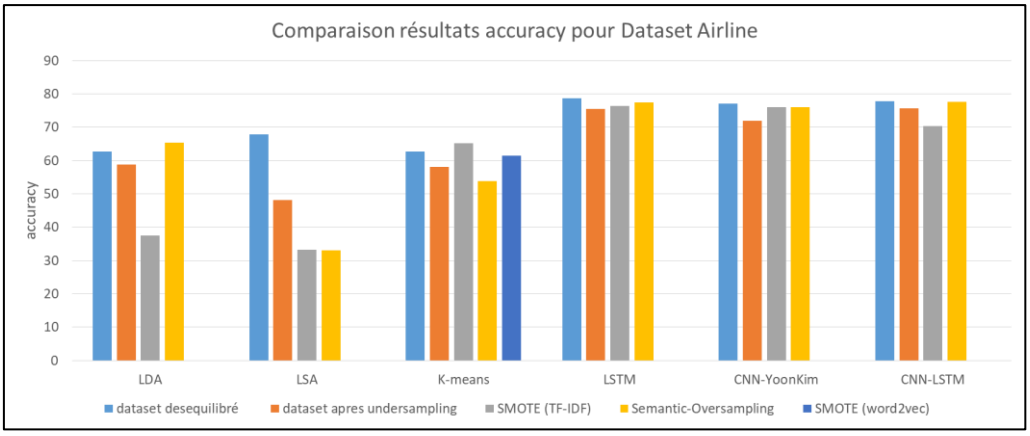

Figure 5. 22 – Comparaison de l'accuracy pour les différentes approches (Airline).

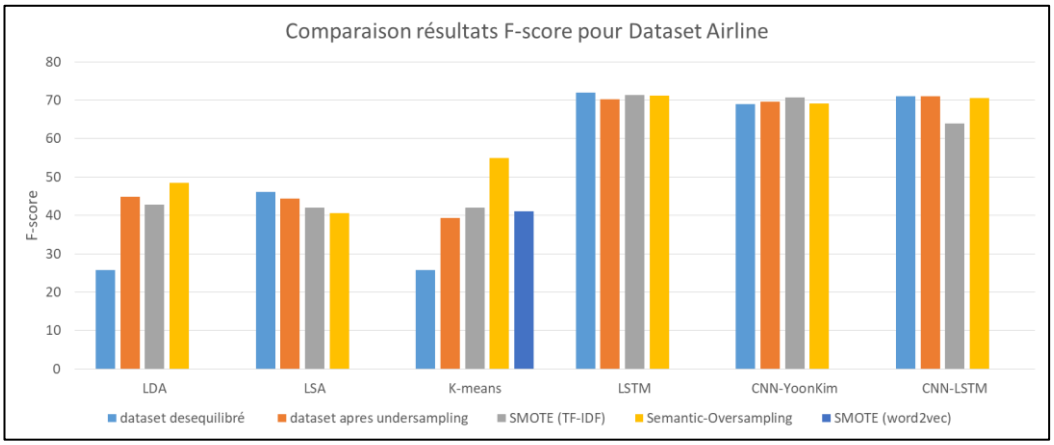

Figure 5. 23 - Comparaison du F-Score pour les différentes approches (Airline)

5.4 Analyse des résultats

D'après les tableaux et les graphes précédents :

- Pour Trolls, le meilleur score d'*accuracy* dans LSTM et CNN-Yoon est obtenu avec un Oversampling (SMOTE avec TF-IDF), pour CNN-LSTM l'accuracy dans le cas du dataset déséquilibré dépasse légèrement celle avec un *Oversampling* (90.67% et 90.4%). Par contre pour Airline le meilleur score d'*accuracy* est obtenu avec le dataset déséquilibré. Le *sampling* pourrait améliorer l'accuracy et le F-score en même temps mais c'était plus difficile dans le cas d'une classification multiple.

- L'approche proposée «*Semantic-Oversampling* » a pu améliorer le F-score de LDA (F-score : 42.50 et 48.50) et K-means (F-score : 42.03 et 55.02). pour le corpus Airline La représentation utilisé dans K-means était les vecteurs Word2vec, le fait d'ajouter des synonymes autour des tokens avec un poids TF-IDF élevé dans chaque classe a semble-t-il ajouté plus de contexte pour ces tokens, et donc une meilleure représentation Word2vec. Pour LDA, l'intuition c'est que les documents sont représentés par une distribution de topics et chaque topic par une distribution de mots, le fait d'ajouter des tweets synthétiques avec

les tokens qui ont un poids TF-IDF élevé pour chaque classe minoritaire, a possiblement aider à avoir une meilleure distribution de mots pour chaque topic.

Pour le dataset Trolls, l'oversampling avec SMOTE a donné des résultats assez proches avec notre approche *Semantic-Oversampling.*

La combinaison CNN-LSTM avec *Semantic-Oversampling* a dépassé SMOTE pour Airline. Pour Trolls le meilleur F-score est obtenu avec CNN-LSTM avec SMOTE.

- La combinaison CNN-LSTM proposée, présente de meilleurs résultats pour le Dataset Trolls et dépasse les résultats obtenu avec CNN-Yoon et LSTM. Cependant pour
  Airline, les meilleurs résultats sont obtenus avec LSTM. CNN-LSTM a probablement échoué en raison de la taille réduite du dataset, car les réseaux de neurones profonds nécessite beaucoup de data. Ainsi une architecture profonde ne conviendrait pas toujours à tous les corpus.

- Dans la plupart des tests, le dropout s'est avéré utile pour éviter l'*overfitting* et a présenté une amélioration d'accuracy pour la plupart des tests.

- Quatre optimizers ont été testé pour LSTM et CNN+LSTM d'après les résultats, les optimizers qui ont convergé le plus vite sont Adam et Adamax suivi d'Adadelta. Cependant il n'y a pas eu un optimizer meilleur qu'un autre.
  Par exemple les plus mauvais résultats sont obtenus avec SGD, seulement il se pourrait qu'avec plus d'epoch (plus d'entrainement) que les résultats avec SGD s'améliorent, comme avec Adadelta dans LSTM (Airline), qui commence a convergé vers l'epoch 60 contrairement à Adam vers l'epoch 20.

CONCLUSION

Dans ce chapitre, plusieurs tests expérimentaux ont été effectués sur deux dataset Airline et Trolls. Ces tests ont été réalisés avant *sampling*, et après *Under*/*Oversampling*.

La combinaison CNN-LSTM à améliorer les résultats pour le corpus Trolls. Pour Airline elle a donné des résultats assez proches de LSTM. Une architecture profonde pourrait nécessiter plus de data.

Après un *sampling*, les algorithmes de classification ont pu détecter des classes minoritaires, ainsi, ils ont présentés une amélioration du F-Score. L'approche proposée pour le sur-échantillonnage des tweets *Semantic-Oversampling* a pu surpasser l'oversampling avec SMOTE, et c'est surtout le cas pour la classification non supervisés avec les algorithmes K-means et LDA, et ceci, pour le corpus Airline.

En outre, nous constatons que les techniques d'échantillonnage présentent une grande amélioration dans la détection des classes minoritaires pour la classification des corpus déséquilibrés.

CONCLUSION GENERALE ET FUTUR PERSPECTIVES

Dans cette étude nous nous sommes intéressées à l'analyse sémantique des statuts dans les réseaux sociaux, nous avons réussi à mettre en œuvre un système d'apprentissage automatique pour la classification des tweets. Dans un premier temps, une phase de prétraitement a été appliquée afin de nettoyer les tweets des stops words et des *emojis* pour ne laisser que les mots pertinents. En second lieu, une représentation vectorielle a été implémentée pour modéliser les tweets. Et enfin une nouvelle approche basé sur une hybridation CNN-LSTM a été proposé et comparée à d'autres approches de la littérature.

Le problème du corpus déséquilibré a été traité, de ce fait, nous avons proposé également une technique de sur-échantillonnage appelé « *Semantic-Oversampling* » pour rééquilibrer les classes, et ainsi, reconnaitre des classes minoritaires.

Nous avons élaboré une étude comparative entre différents algorithmes de classification. Les résultats obtenus avec les réseaux de neurones dépassent ceux obtenus avec des algorithmes classiques comme K-means, LDA et LSA.

Les performances de la combinaison CNN-LSTM proposée ont pu dépasser celles des algorithmes LSTM et le CNN de Yoon, et ceci pour le dataset Trolls (Original). Pour le dataset Airline (Original), CNN-LSTM a dépassé le CNN de Yoon, cependant LSTM a légèrement surpasser notre modèle. Il semblerait qu'une architecture profonde pourrait nécessiter plus de données, et que dans ce cas, une architecture avec seulement deux couches serait meilleure.

Le F-score a connu une grande amélioration après l'échantillonnage (*Sampling*), et ceci est surtout le cas des algorithmes de classification non supervisées. Le F-score avec « *Semantic-Oversampling* » a dépassé celui obtenu avec SMOTE, ceci est le cas pour LDA et K-means dans le corpus Airline. Pour Trolls les résultats obtenus avec « *Semantic-Oversampling* » sont assez proches de celles obtenus avec SMOTE.

Les résultats d'analyse des tweets avec les réseaux de neurones pourraient éventuellement être améliorés en utilisant un *Automatic Machine Learning*, le but serait d'apprendre au model, à régler les paramètres automatiquement ainsi obtenir le meilleure score possible.

En dernier lieu, une hybridation entre un embedding à un niveau de *tokens* et un niveau de caractère serait une perspective forte intéressante, en effet, une représentation au niveau de caractères pourrait être plus appropriée pour la modélisation des statuts courts, qui contiennent

souvent beaucoup de bruits, des fautes de grammaire, des abréviations et des erreurs typographiques.

Annexe : Résultats des algorithmes de classification

1. Dataset Trolls

1.1. Résultats LDA (dataset original)

Tableau 6. 1 – Résultat LDA sur dataset Trolls Original.

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Passes = 1 Itération =1	**60.49%**	30.33%	50.00%	**37.75%**
Passes =1 Itération =5	60.49%	30.33%	50.00%	37.75%
Passes = 2 Itération = 10	60.49%	30.33%	50.00%	37.75%
Passes = 2 Itération = 20	60.49%	30.33%	50.00%	37.75%
Passes = 3 Itération = 50	60.49%	30.33%	50.00%	37.75%
Passes = 3 Itération =100	60.49%	30.33%	50.00%	37.75%
Passes = 4 Itération = 300	60.49%	30.33%	50.00%	37.75%
Passes = 4 Itération = 500	60.49%	30.33%	50.00%	37.75%
Passes = 5 Itération = 600	60.49%	30.33%	50.00%	37.75%
Passes = 6 Itération = 900	60.49%	30.33%	50.00%	37.75%
Passes = 7 Itération = 1000	60.49%	30.33%	50.00%	37.75%
Passes = 10 Itération = 1500	60.49%	30.33%	50.00%	37.75%
Passes = 15 Itération = 2000	60.49%	30.33%	50.00%	37.75%
Passes =20 Itération = 3000	60.49%	30.33%	50.00%	37.75%

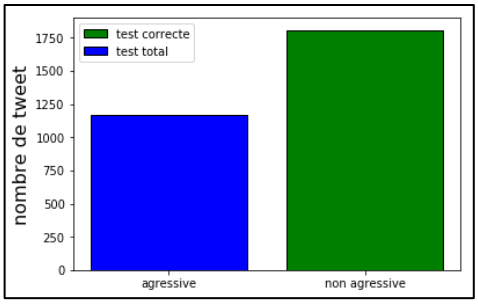

Figure 6. 1 - Test correct par classe LDA (Trolls Original)

1.2. Résultats LDA (dataset après undersampling)

Tableau 6. 2 - Résultat LDA sur dataset Trolls après Undersampling.

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Passes = 1 Itération =1	53.08%	51.09%	51.15%	51.12%
Passes =1 Itération =5	53.25%	51.97%	52.05%	52.01%
Passes = 2 Itération = 10	58.01%	56.39%	56.56%	56.47%
Passes = 2 Itération = 20	56.77%	55.29%	55.47%	55.38%
Passes = 3 Itération = 50	57.64%	55.75%	55.86%	55.80%
Passes = 3 Itération =100	58.18%	56.29%	56.40%	56.35%
Passes = 4 Itération = 300	58.31%	56.32%	56.41%	56.37%

Passes = 4 Itération = 500	58.21%	56.20%	56.28%	56.24%
Passes = 5 Itération = 600	59.18%	57.09%	57.16%	57.13%
Passes = 6 Itération = 900	59.25%	57.18%	57.24%	57.21%
Passes = 7 Itération = 1000	**59.62%**	**57.33%**	**57.42%**	**57.37%**
Passes = 10 Itération = 1500	55.29%	53.92%	54.07%	53.99%
Passes = 15 Itération = 2000	55.86%	54.31%	54.47%	54.39%
Passes =20 Itération = 3000	56.13%	54.45%	54.60%	54.53%

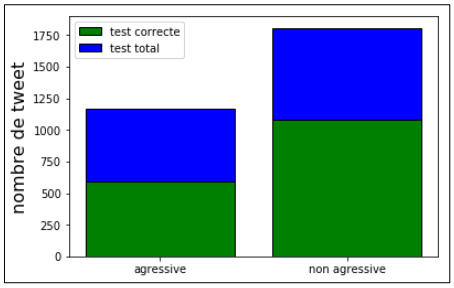

Figure 6. 2 - Test correcte par classe LDA (Trolls après Undersampling)

1.3. Résultats LDA (dataset après oversampling)

1.2.1 Résultats SMOTE

Tableau 6. 3 - Résultat LDA sur dataset Trolls après Oversampling (SMOTE).

paramètres	**Accuracy (%)**	**Recall (%)**	**Precision (%)**	**F-score (%)**
Passes = 1 Itération =1	48.55%	47.52%	47.40%	47.46%
Passes =1 Itération =5	56.50%	55.48%	55.71%	55.60%
Passes = 2 Itération = 10	57.81%	56.42%	56.64%	56.53%
Passes = 2 Itération = 20	58.28%	56.82%	57.05%	56.93%
Passes = 3 Itération = 50	57.04%	55.71%	55.92%	55.81%
Passes = 3 Itération =100	57.04%	55.73%	55.95%	55.84%
Passes = 4 Itération = 300	57.61%	56.24%	56.46%	56.35%
Passes = 4 Itération = 500	57.61%	56.24%	56.46%	56.35%
Passes = 5 Itération = 600	57.91%	56.44%	56.65%	56.54%
Passes = 6 Itération = 900	58.11%	56.65%	56.88%	56.77%
Passes = 7 Itération = 1000	58.11%	56.65%	56.88%	56.77%
Passes = 10 Itération = 1500	58.55%	56.99%	57.21%	57.10%
Passes = 15 Itération = 2000	59.42%	57.80%	58.02%	57.91%
Passes =20 Itération = 3000	**60.09%**	**58.47%**	**58.70%**	**58.59%**

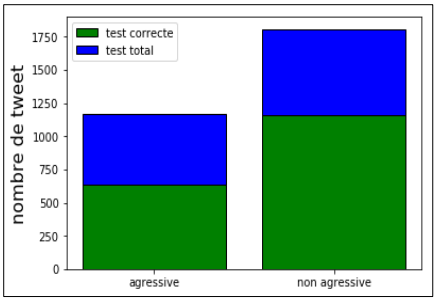

Figure 6. 3 - Test correct par classe LDA (Trolls SMOTE)

1.2.2 Semantic-Oversampling

Tableau 6. 4 - Résultat LDA sur dataset Trolls après Oversampling (Semantic-Oversampling).

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-score (%)**
Passes = 1 Itération =1	51.67%	50.02%	50.02%	50.02%
Passes =1 Itération =5	51.07%	49.99%	49.99%	49.99%
Passes = 2 Itération = 10	53.85%	50.23%	50.21%	50.22%
Passes = 10 Itération = 20	52.34%	54.30%	54.40%	54.35%
Passes = 3 Itération = 50	57.47%	51.73%	51.14%	51.43%
Passes = 3 Itération =100	57.47%	51.46%	50.93%	51.20%
Passes = 4 Itération = 300	57.44%	50.15%	50.08%	50.11%
Passes = 4 Itération = 500	57.41%	50.10%	50.05%	50.07%
Passes = 5 Itération = 600	57.14%	48.75%	49.40%	49.08%
Passes = 6 Itération = 900	57.74%	49.47%	49.77%	49.62%
Passes = 7 Itération = 1000	57.91%	49.33%	49.73%	49.53%
Passes = 10 Itération = 1500	57.91%	49.33%	49.73%	49.53%
Passes = 15 Itération = 2000	58.48%	49.75%	49.91%	49.83%
Passes =20 Itération = 3000 -- -- -- -- -- -- -- TF-IDF min-count : 0.01	58.51% -- -- -- - **57.98%**	49.55% -- -- -- - **57.35%**	49.85% -- -- -- - **57.65%**	49.70% -- -- -- - **57.50%**

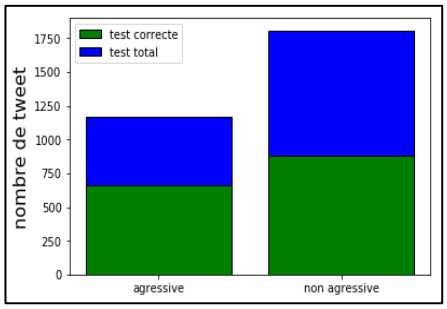

Figure 6. 4 - Test correct par classe LDA (Semantic-Oversampling)

1.3 Résultats LSA (dataset original)

Tableau 6. 5 - Résultat LSA sur dataset Trolls Original.

Accuracy (%)	**Precision(%)**	**Recall(%)**	**F-score (%)**

59.91%	30.15%	50.00%	37.62%

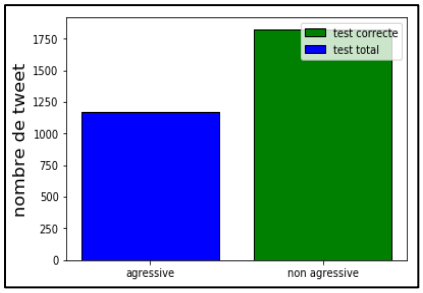

Figure 6. 5 - Test correct par classe LSA (Original)

1.4 Résultats LSA (dataset après undersampling)

Tableau 6. 6 - Résultat LSA sur dataset Trolls après Undersampling.

Accuracy (%)	**Precision(%)**	**Recall(%)**	**F-score (%)**
59.49%	52.24%	50.66%	51.44%

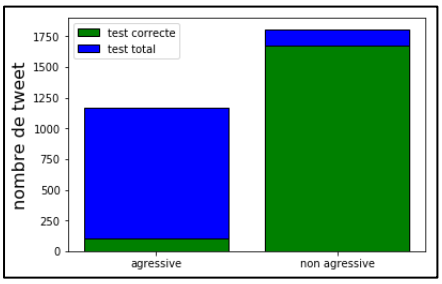

Figure 6. 6 - Test correcte par classe LSA (Trolls Undersampling)

1.5 Résultats LSA (dataset après oversampling)

1.5.1 Résultats SMOTE

Tableau 6. 7 - Résultat LSA sur dataset Trolls après Oversampling (SMOTE).

Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
59,12%	45.83%	50,00%	47.82%

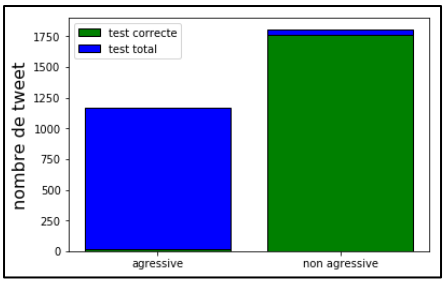

Figure 6. 7 - Test correcte par classe LSA (Trolls Undersampling).

1.5.2 Semantic-Oversampling

Tableau 6. 8- Résultat LSA sur dataset Trolls après Oversampling (Synonymes).

Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
39.50% -- -- -- -- Min-df : 0..01 :39.50%	30.33% -- -- -- -- 30.33%	50.00% -- -- -- -- 50.00%	37.75% -- -- -- -- -- 37.75%

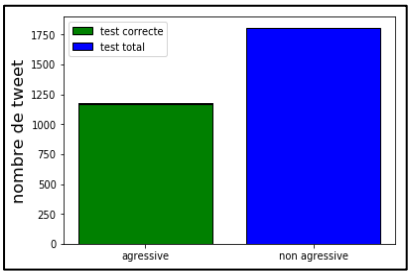

Figure 6. 8 - Test correct par classe LSA (Semantic-Oversampling).

1.6 Résultats K-means (dataset original)

Tableau 6. 9 - Résultat K-means sur dataset Trolls Original.

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Window size : 129 Itération =1	**60.66%**	30.33%	50.00%	**37.75%**
Window size : 129 Itération =5	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 10	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 20	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 50	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération =100	60.66%	30.33%	50.00%	37.75%

Window size : 129 Itération = 300	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 500	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 600	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 900	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 1000	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 1500	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 2000	60.66%	30.33%	50.00%	37.75%
Window size : 129 Itération = 3000	60.66%	30.33%	50.00%	37.75%

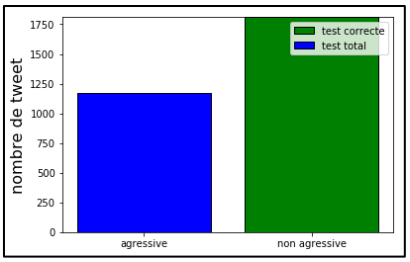

Figure 6. 9 - Test correct par classe K-means (Trolls Original).

1.7 Résultats K-means (dataset après undersampling)

Tableau 6. 10 - Résultat K-means sur dataset Trolls après Undersampling.

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall(%)**	**F-score (%)**
Window size : 129 Itération =1	46.57%	44.98%	44.85%	44.91%
Window size : 129 Itération =5	38.93%	42.94%	46.25%	44.53%
Window size : 129 Itération = 10	38.93%	42.94%	46.25%	44.53%
Window size : 129 Itération = 20	38.96%	43.00%	46.29%	44.59%
Window size : 129 Itération = 50	**61.06%**	**57.05%**	53.74%	**55.35%**
Window size : 129 Itération =100	38.93%	42.94%	46.25%	44.53%
Window size : 129 Itération = 300	61.06%	57.05%	53.74%	55.35%
Window size : 129 Itération = 500	61.06%	57.05%	53.74%	55.35%
Window size : 129 Itération = 600	38.93%	42.94%	46.25%	44.53%
Window size : 129 Itération = 900	38.93%	42.94%	46.25%	44.53%
Window size : 129 Itération = 1000	55.53%	54.21%	**54.34%**	54.28%
Window size : 129 Itération = 1500	38.96%	43.00%	46.29%	44.59%
Window size : 129 Itération = 2000	61.03%	56.99%	53.70%	55.29%
Window size : 129 Itération = 3000	46.57%	44.98%	44.85%	44.91%

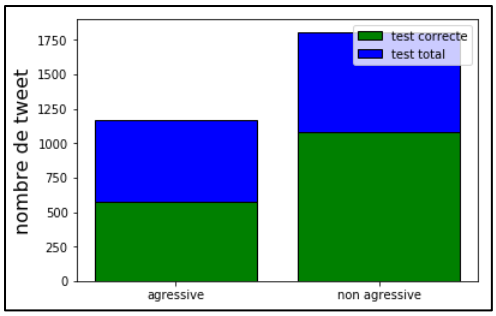

Figure 6. 10- Test correcte par classe K-means (Trolls Undersampling

1.8 Résultats K-means (dataset après oversampling)

1.8.1 Résultats SMOTE

Tableau 6. 11 - Résultat K-means après Oversampling (SMOTE).

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
Window size : 129 Itération =1	**55.11%**	**53.88%**	**53.76%**	**53.82%**
Window size : 129 Itération =5	44.88%	45.99%	46.13%	46.06%
Window size : 129 Itération = 10	55.04%	53.64%	53.54%	53.59%
Window size : 129 Itération = 20	54.91%	53.54%	53.45%	53.49%
Window size : 129 Itération = 50	55.04%	53.75%	53.63%	53.69%
Window size : 129 Itération =100	55.04%	53.75%	53.63%	53.69%
Window size : 129 Itération = 300	44.95%	46.45%	46.35%	46.40%
Window size : 129 Itération = 500	55.04%	53.64%	53.54%	53.59%
Window size : 129 Itération = 600	55.04%	53.64%	53.54%	53.59%
Window size : 129 Itération = 900	44.95%	46.45%	46.35%	46.40%
Window size : 129 Itération = 1000	55.01%	53.61%	53.51%	53.56%
Window size : 129 Itération = 1500	55.04%	53.54%	53.64%	53.59%
Window size : 129 Itération = 2000	55.04%	53.64%	53.54%	53.59%
Window size : 129 Itération = 3000	55.04%	53.64%	53.54%	53.59%

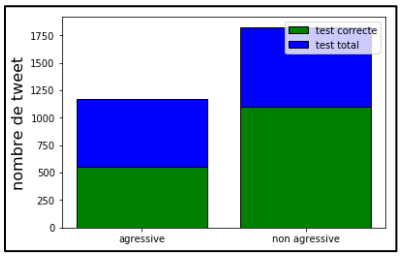

Figure 6. 11 - Test correcte par classe K-means (Trolls SMOTE).

1.8.2 SMOTE (word2vec)

Tableau 6. 12 - Résultat K-means après Oversampling (SMOTE avec Word2vec).

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Window size : 129 Itération =1	43.66%	45.96%	46.06	46.01%
Window size : 129 Itération =5	43.89%	45.47%	45.41%	45.44%
Window size : 129 Itération = 10	**56.03%**	**54.49%**	**54.55%**	**54.52%**
Window size : 129 Itération = 20	43.96%	45.50%	45.44%	45.47%
Window size : 129 Itération = 50	56.03%	54.49%	54.55%	54.52%
Window size : 129 Itération =100	43.96%	45.50%	45.44%	45.47%
Window size : 129 Itération = 300	43.96%	45.50%	45.44%	45.47%
Window size : 129 Itération = 500	56.03%	54.49%	54.55%	54.52%
Window size : 129 Itération = 600	43.89%	45.47%	45.41%	45.44%
Window size : 129 Itération = 900	56.03%	54.49%	54.55%	54.52%
Window size : 129 Itération = 1000	56.03%	54.49%	54.55%	54.52%
Window size : 129 Itération = 1500	43.89%	45.47%	45.41%	45.44%
Window size : 129 Itération = 2000	56.03%	54.49%	54.55%	54.52%
Window size : 129 Itération = 3000	43.66%	45.96%	46.06	46.01%

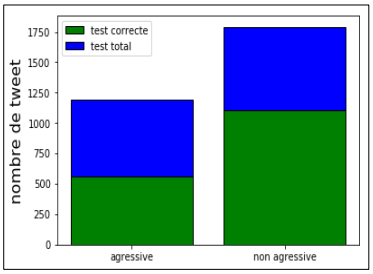

Figure 6. 12 - Test correcte par classe K-means (Trolls SMOTE).

1.8.3 Semantic-Oversampling

Tableau 6. 13 - Résultat K-means après Oversampling (Semantic-Oversampling).

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Itération =1 Window size=129	44.33%	46.05%	45.99%	46.02%
Itération =5 Window size=129	44.36%	46.08%	46.02%	46.05%

Itération = 10 Window size=129	48.33%	36.19%	47.70%	41.15%
Itération = 20 Window size=129	**55.66%**	**53.94%**	**54.00%**	**53.97%**
Itération = 50 Window size=129	55.66%	53.94%	54.00%	53.97%
Itération =100 Window size=129	44.33%	46.05%	45.99%	46.02%
Itération = 300 Window size=129	44.33%	46.05%	45.99%	46.02%
Itération = 500 Window size=129	55.66%	53.94%	54.00%	53.97%
Itération = 600 Window size=129	55.66%	53.94%	54.00%	53.97%
Itération = 900 Window size=129	44.36%	46.08%	46.02%	46.05%
Itération = 1000 Window size=129	55.66%	53.94%	54.00%	53.97%
Itération = 1500 Window size=129	44.36%	46.08%	46.02%	46.05%
Itération = 2000 Window size=129	44.33%	46.05%	45.99%	46.02%
Itération = 3000 Window size=129 -- -- -- -- -- -- -- --	44.36% -- -- -- --	46.08% -- -- -- --	46.02% -- -- -- --	46.05% -- -- -- --
TF-IDF : Min-count : 0.01	39.33%	19.66%	50.00%	28.23%

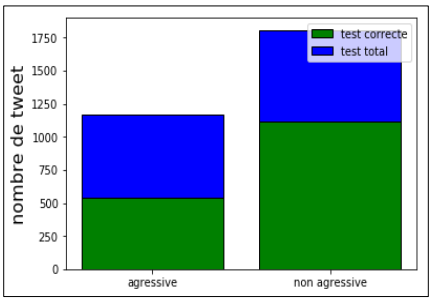

Figure 6. 13 - Test correcte par classe K-means (Semantic-Oversampling).

1.9 Résultats CNN-Yoon (dataset Orignal)

Tableau 6. 14 - Résultat CNN-Yoon sur dataset Trolls Original.

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-score (%)**
epoch : 1 batchsize= 512	62.15%	50.00%	31.07%	38.32%
epoch : 5 batchsize= 512	82.64%	82.75%	84.52%	83.63%
epoch : 10 batchsize= 512	82.64%	50.86%	50.90%	50.89%
epoch : 15 batchsize= 512	84.57%	83.66%	85.09%	84.37%
epoch : 20 batchsize= 512	79.67%	82.22%	83.01%	82.61%
epoch : 25 batchsize= 512	82.21%	83.53%	84.81%	84.16%

epoch : 30 batchsize= 512	**86.60%**	85.86%	86.81%	**86.33%**

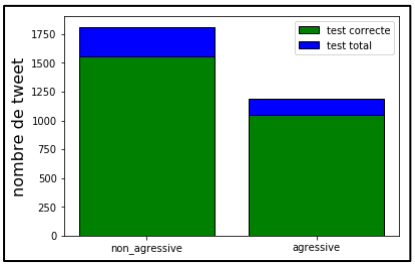

Figure 6. 14 - Test correcte par classe K-means (Semantic-Oversampling).

1.10 Résultats CNN-Yoon (dataset après undersampling)

Tableau 6. 15 - Résultat CNN-Yoon sur dataset Trolls après Undersampling.

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-score (%)**
epoch : 1 batchsize= 512	65.73%	71.75%	76.71%	74.15%
epoch : 5 batchsize= 512	79.91%	81.85%	80.47%	81.15%
epoch : 10 batchsize= 512	79.68%	81.58%	80.20%	80.89%
epoch : 15 batchsize= 512	**83.37%**	**83.75%**	**82.59%**	**83.17%**
epoch : 20 batchsize= 512	82.60%	83.58%	82.14%	82.85%
epoch : 25 batchsize= 512	75.32%	78.75%	78.59%	78.67%
epoch : 30 batchsize= 512	75.32%	78.75%	78.59%	78.67%

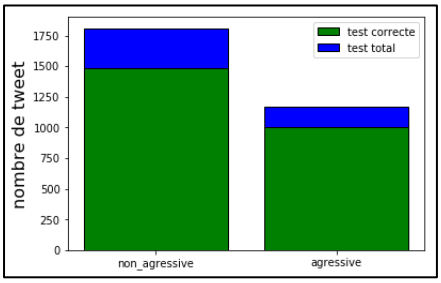

Figure 6. 15 - Test correcte par classe après Undersampling.

1.11 Résultats CNN-Yoon (dataset après oversampling)

1.11.1 Résultats SMOTE

Tableau 6. 16 - Résultat CNN-Yoon sur dataset Trolls après Oversampling (SMOTE).

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
epoch : 1 batchsize= 512	72.64%	76.34%	76.29%	76.32%
epoch : 2 batchsize= 512	81.74%	81.15%	82.56%	81.85%
epoch : 3 batchsize= 512	85.80%	85.13%	86.62%	85.87%
epoch : 5 batchsize= 512	87.10%	86.66%	88.40%	87.52%
epoch : 10 batchsize= 512	86.77%	86.53%	88.32%	87.42%
epoch : 15 batchsize= 512	**89.10%**	**88.43%**	**90.01%**	**89.21%**
epoch : 20 batchsize= 512	88.37%	87.83%	89.55%	88.68%

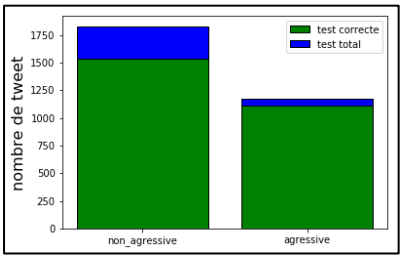

Figure 6. 16 - Test correcte par classe dataset Trolls après Oversampling (SMOTE).

1.11.2 Semantic-Oversampling

Tableau 6. 17 - Résultat CNN-Yoon sur dataset Trolls après Oversampling (Semantic-Oversampling).

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
epoch : 1 batchsize= 512	70.95%	69.21%	69.51%	69.36%
epoch : 5 batchsize= 512	81.62%	80.89%	80.71%	80.80%
epoch : 10 batchsize= 512	83.09%	82.90%	82.21%	82.56%
epoch : 15 batchsize= 512	83.53%	83.56%	82.68%	83.12%
epoch : 20 batchsize= 512	83.26%	83.61%	82.48%	83.04%
epoch : 25 batchsize= 512	**83.53%** -- -- -- -- Min_count : 0.01 81.99%	**83.74%** -- -- -- 81.35%	**82.71%** -- -- -- 82.65%	**83.22%** -- -- -- 82.00%

epoch : 30 batchsize= 512	83.10%	83.65%	82.39%	83.02%

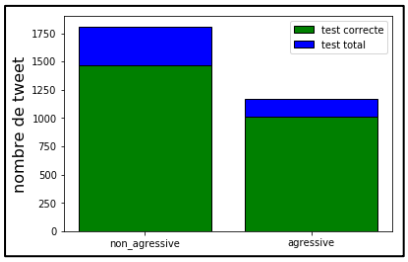

Figure 6. 17 - Test correct CNN-Yoon par classe (dataset Trolls SMOTE).

1.12 Résultats LSTM (dataset original)

Tableau 6. 18 - Résultat LSTM sur dataset Trolls (Original)

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	60.31%	50.00%	30.15%	37.62%
epoch : 5/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	76.61%	74.34%	75.96	75.14
epoch:10/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	79.81%	78.57%	78.99%	78.78%
epoch:10/batchsize= 512 embed_dim = 64 lstm_output = 196 Dropout = 0.5	80.37%	79.50%	80.21%	79.85%
epoch:10/batchsize= 512 embed_dim = 128 lstm_output = 196 Dropout = 0.5	81.17%	80.32%	80.33%	80.33%
epoch : 10/batchsize= 512 embed_dim = 256 lstm_output = 196 Dropout = 0.5	81.77%	82.00%	81.01%	81.50%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_output = 256 Dropout = 0.5	81.24%	79.95%	80.56%	80.25%
epoch : 20/batchsize= 512 Embed_dim =128 lstm_output = 192 Dropout = none	81.71%	82.09	81.00%	81.54%

epoch : 20/batchsize= 1000 Embed_dim =128 lstm_output = 192 Dropout = 0.2	80.21%	78.81%	79.49%	79.14%
batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	30 epoch : 60.31% -- -- -- -- -- 30 epoch : 81.91% -- -- -- -- -- 30 epoch : **82.01%** -- -- -- -- -- 30 epoch : 70.74%	50.00% -- -- -- -- -- 82.20% -- -- -- -- -- 81.15% -- -- -- -- -- 67.08%	30.15% -- -- -- -- -- 81.16% -- -- -- -- -- 81.72% -- -- -- -- -- 69.96%	37.62% -- -- -- -- -- **81.68%** -- -- -- -- -- 81.38% -- -- -- -- -- 68.49%
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0,5 Activation : sigmoïde	76.76%	74.73%	75.96%	75.34%

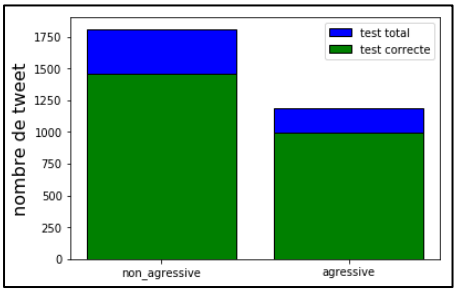

Figure 6. 18 - Test correct par classe LSTM (dataset Trolls Original).

1.13 Résultats LSTM (dataset après undersampling)

Tableau 6. 19 - Résultat LSTM sur dataset Trolls après Undersampling.

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	54.69%	56.51%	56.34%	56.43%
epoch : 5/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	73.78%	73.13%	74.11%	73.62%
epoch:10/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	77.67%	77.65%	78.96%	78.30%
epoch:10/batchsize= 512 embed_dim = 64 lstm_output = 196 Dropout = 0.5	78.00%	79.23%	77.90%	78.56%
epoch:10/batchsize= 512 embed_dim = 128 lstm_output = 196 Dropout = 0.5	79.04%	80.41%	79.03%	79.71%

epoch : 10/batchsize= 512 embed_dim = 256 lstm_output = 196 Dropout = 0.5	79.24%	79.60%	80.98%	80.29%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_output = 256 Dropout = 0.5	**80.45%**	80.55%	8201%	81.27%
epoch : 20/batchsize= 512 Embed_dim =128 lstm_output = 192 Dropout = none	78.17%	78.06%	79.39%	78.72%
epoch : 20/batchsize= 1000 Embed_dim =128 lstm_output = 192 Dropout = 0.2	30 epoch : 43.13% -- -- -- -- -- 30 epoch : 80.95% -- -- -- -- -- 30 epoch : 78.71% -- -- -- -- -- 30 epoch : 65.83% 60 epoch:73.69% 120 epoch:71.91% 200 epoch : 71.55%	46.03% -- -- -- -- 80.73% -- -- -- -- 80.03% -- -- -- -- 61.09% 69.77% 71.99% 69.67%	46.86% -- -- -- -- **82.18%** -- -- -- -- 78.67% -- -- -- -- - 63.95% 65.94% 67.09% 65.55%	45.95% -- -- -- -- -- **81.45%** -- -- -- -- -- 79.34% -- -- -- -- -- 62.49% 67.80% 69.46% 67.55%
batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	79.51%	**81.37%**	79.99%	80.67%
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0,5 Activation : sigmoïde	79.24%	79.60%	80.98%	80.29%

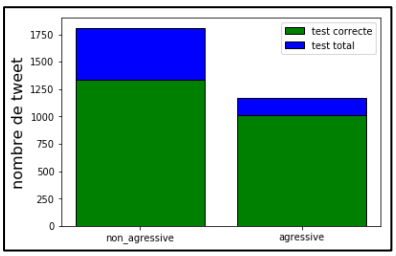

Figure 6. 19 - Test correct par classe LSTM (dataset Trolls undersampling)

1.14 Résultats LSTM (dataset après oversampling)

1.14.1 Résultats SMOTE

Tableau 6. 20 - Résultat LSTM sur dataset Trolls après Oversampling (SMOTE).

paramètres	Accuracy (%)	Precision(%)	Recall(%)	F-score (%)
epoch : 1/batchsize= 512 embed_dim = 32 lstm_output = 196	53.98%	62.95%	60.02%	61.45%

Dropout = 0.5				
epoch : 5/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	80.94%	81.23%	82.76%	81.98%
epoch:10/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	83.91%	83.57%	85.19%	84.37%
epoch:10/batchsize= 512 embed_dim = 64 lstm_output = 196 Dropout = 0.5	84.61%	84.31%	85.98%	85.14%
epoch:10/batchsize= 512 embed_dim = 128 lstm_output = 196 Dropout = 0.5	85.54%	85.17%	86.87%	86.01%
epoch : 10/batchsize= 512 embed_dim = 256 lstm_output = 196 Dropout = 0.5	86.27%	85.90%	87.62%	86.75%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_output = 256 Dropout = 0.5	**87.27%**	86.96%	88.76%	**87.86%**
epoch : 20/batchsize= 512 Embed_dim =128 lstm_output = 192 Dropout = none	86.30%	86.18%	87.97%	87.06%
epoch : 20/batchsize= 1000 Embed_dim =128 lstm_output = 192 Dropout = 0.2	86.94%	86.59%	88.35%	87.46%
batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	20 epoch : 39.12% -- -- -- -- -- 20 epoch : 84.84% -- -- -- -- -- 20 epoch : 84.47% -- -- -- -- -- 20 epoch : 67.68%	19.56% -- -- -- -- - 84.93% -- -- -- -- - 84.16% -- -- -- -- - 72.54%	50.00% -- -- -- -- 86.66% -- -- -- -- 85.83% -- -- -- -- 71.77%	28.11% -- -- -- -- -- 85.78% -- -- -- -- -- 84.99% -- -- -- -- -- 72.15%
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0,5 Activation : sigmoïde	85.84%	85.56%	87.30%	86.42%

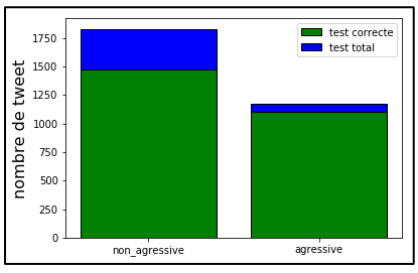

Figure 6. 20 - Test correct par classe LSTM (dataset Trolls SMOTE).

2.1.1 Semantic-Oversampling

Tableau 6. 21 - Résultat LSTM sur dataset Trolls après Oversampling (Semantic-Oversampling).

paramètres	**Accuracy (%)**	**Recall (%)**	**Precision (%)**	**F-score (%)**
epoch : 1/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	58.01%	58.08%	57.72%	57.90%
epoch : 5/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	75.86%	75.30%	73.05%	74.16%
epoch:10/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	79.58%	79.03%	78.58%	78.80%
epoch:10/batchsize= 512 embed_dim = 64 lstm_dim = 196 Dropout = 0.5	80.11%	79.51%	79.13%	79.32%
epoch:10/batchsize= 512 embed_dim = 128 lstm_dim = 196 Dropout = 0.5	81.96%	81.74%	81.05%	81.39%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_dim = 256 Dropout = 0.5	81.49%	80.80%	80.57%	80.68%
epoch : 20/batchsize= 512 Embed_dim =128 lstm_dim = 192 Dropout = none	82.16%	81.46%	**82.71%**	82.08%
epoch : 20/batchsize= 1000 Embed_dim =128 lstm_dim = 192 Dropout = 0.2	**82.73%**	**83.11%**	81.96%	**82.53%**
batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	30 epoch : 55.47% -- -- -- -- -- 30 epoch : 81.62% Min_df : 0.01 : 81.09% -- -- -- -- -- 30 epoch : 81.35% -- -- -- -- -- 30 epoch :60.60%	53.64% -- -- -- -- -- 81.49% 80.47% -- -- -- -- -- 80.49 -- -- -- -- 64.97%	53.60% -- -- -- -- -- 80.72% 81.75 % -- -- -- -- 81.45% -- -- -- -- - 66.24%	53.62% -- -- -- -- -- 81.10% 81.11% -- -- -- -- -- 80.97% -- -- -- -- -- 65.60%
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0,5 Activation : sigmoïde	81.69%	81.85%	.8085%	81.34%

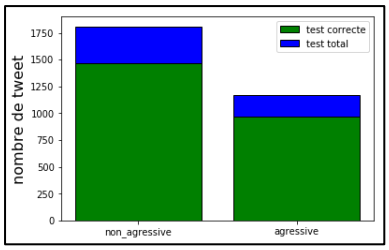

Figure 6. 21- Test correct par classe LSTM (dataset Trolls Semantic-Oversampling)

2. Dataset Airline

2.1. Résultats LDA (dataset original)

Tableau 6. 22 - Résultat LDA sur dataset Airline (Original)

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
Passes = 1 Itération =1	**62.79%**	33.33%	20.91%	**25.70%**
Passes =1 Itération =5	62.79%	33.33%	20.91%	25.70%
Passes = 2 Itération = 10	62.79%	33.33%	20.91%	25.70%
Passes = 2 Itération = 20	62.79%	33.33%	20.91%	25.70%
Passes = 3 Itération = 50	62.79%	33.33%	20.91%	25.70%
Passes = 3 Itération =100	62.79%	33.33%	20.91%	25.70%
Passes = 4 Itération = 300	62.79%	33.33%	20.91%	25.70%
Passes = 4 Itération = 500	62.79%	33.33%	20.91%	25.70%
Passes = 5 Itération = 600	62.79%	33.33%	20.91%	25.70%
Passes = 6 Itération = 900	62.79%	33.33%	20.91%	25.70%
Passes = 7 Itération = 1000	62.79%	33.33%	20.91%	25.70%
Passes = 10 Itération = 1500	62.79%	33.33%	20.91%	25.70%
Passes = 15 Itération = 2000	62.79%	33.33%	20.91%	25.70%
Passes =20 Itération = 3000	62.79%	33.33%	20.91%	25.70%

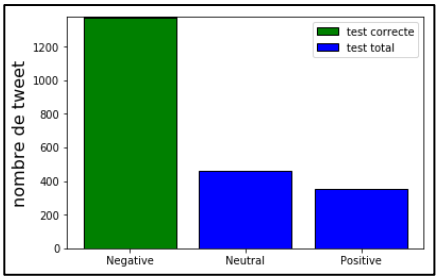

Figure 6. 22 - Test correct par classe LDA (dataset Airline Original)

2.2. Résultats LDA (dataset après undersampling)

Tableau 6. 23 - Résultat LDA sur dataset Airline (après undersampling)

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Passes = 1 Itération =1	50.25%	28.57%	34.26%	31.16%

Passes =1 Itération =5	44.68%	40.74%	42.18%	41.45%
Passes = 2 Itération = 10	46.69%	41.75%	42.90%	42.32%
Passes = 2 Itération = 20	45.37%	41.70%	42.20%	41.95%
Passes = 3 Itération = 50	**58.86%**	31.16%	36.71%	33.71%
Passes = 3 Itération =100	38.39%	38.90%	38.43%	38.67%
Passes = 4 Itération = 300	57.40%	29.98%	35.61%	32.55%
Passes = 4 Itération = 500	44.96%	32.64%	38.62%	35.38%
Passes = 5 Itération = 600	46.23%	42.29%	42.66%	42.47%
Passes = 6 Itération = 900	46.96%	42.70%	42.91%	42.81%
Passes = 7 Itération = 1000	47.10%	42.26%	42.37%	42.31%
Passes = 10 Itération = 1500	49.84%	43.57%	43.84%	43.70%
Passes = 15 Itération = 2000	52.16%	44.37%	44.63%	44.50%
Passes =20 Itération = 3000	53.03%	**44.60%**	**45.07%**	**44.83%**

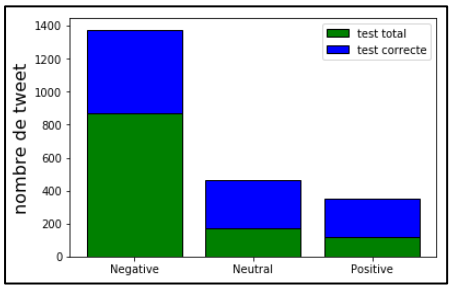

Figure 6. 23 - Test correct par classe LDA (dataset Airline Original).

2.3. Résultats LDA (dataset après oversampling)

2.3.1 Résultats SMOTE

Tableau 6. 24 - Résultat LDA sur dataset Airline (après SMOTE)

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
Passes = 1 Itération =1	47.74%	25.56%	32.16%	28.48%
Passes =1 Itération =5	33.33%	32.20%	32.21%	32.20%
Passes = 2 Itération = 10	36.43%	32.36%	32.24%	32.30%
Passes = 2 Itération = 20	35.38%	26.14%	33.14%	29.23%
Passes = 3 Itération = 50	39.80%	33.40%	33.28%	33.34%
Passes = 3 Itération =100	38.39%	38.90%	38.43%	38.67%
Passes = 4 Itération = 300	54.12%	26.77%	33.94%	29.93%
Passes = 4 Itération = 500	42.08%	33.75%	33.75%	33.75%
Passes = 5 Itération = 600	**51.48%**	26.22%	33.23%	29.31%
Passes = 6 Itération = 900	46.96%	**42.70%**	**42.91%**	**42.81%**
Passes = 7 Itération = 1000	41.26%	33.59%	33.59%	33.59%

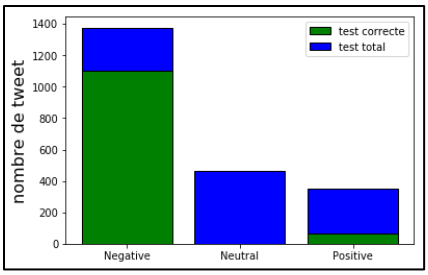

Figure 6. 24 - Test correct par classe LDA (dataset Airline SMOTE).

Passes = 10 Itération = 1500	40.08%	26.44%	33.81%	29.68%
Passes = 15 Itération = 2000	40.62%	26.22%	33.19%	29.30%
Passes =20 Itération = 3000	33.49%	26.37%	33.49%	29.50%

2.3.2 Semantic-Oversampling

Tableau 6. 25 - Résultat LDA sur dataset Airline (Semantic-Oversampling)

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Passes = 1 Itération =1	50.56%	28.38%	36.83%	32.05%
Passes =1 Itération =5	43.77%	38.25%	39.68%	38.95%
Passes = 2 Itération = 10	50.88%	40.91%	42.18%	41.53%
Passes = 2 Itération = 20	42.27%	37.46%	38.91%	38.17%
Passes = 3 Itération = 50	48.24%	38.71%	39.51%	39.11%
Passes = 3 Itération =100	49.11%	38.81%	39.43%	39.12%
Passes = 4 Itération = 300	51.52%	40.46%	41.10%	40.78%
Passes = 4 Itération = 500	49.79%	39.35%	39.99%	39.67%
Passes = 5 Itération = 600	53.48%	41.30%	41.76%	41.53%
Passes = 6 Itération = 900	56.26%	43.00%	43.18%	43.09%
Passes = 7 Itération = 1000	58.45%	45.15%	44.86%	45.00%
Passes = 10 Itération = 1500	58.45%	45.15%	**44.86%**	45.00%
Passes = 15 Itération = 2000	65.75%	53.31%	44.49%	**48.50%**
Passes =20 Itération = 3000	**65.98%**	**53.73%**	43.17%	47.88%
-- -- -- -- --	-- -- --	-- -- -- --	-- -- -- --	-- -- -- --
Min-df = 0.01	14.27%	9.46%	24.56%	13.66%

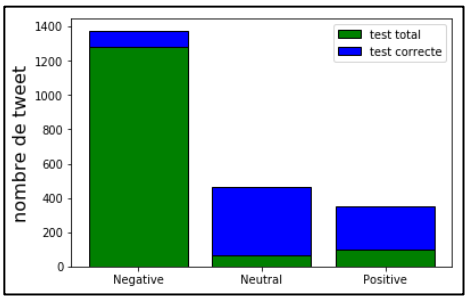

Figure 6. 25 - Test correct par classe LDA (dataset Airline Semantic-Oversampling).

2.4 Résultats LSA (dataset original)

Tableau 6. 26 - Résultat LSA sur dataset Airline (Original)

Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
67.85%	47.21%	44.97%	46.06%

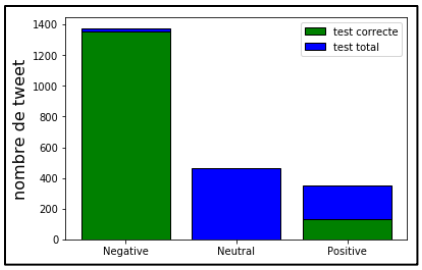

Figure 6. 26 - Test correct par classe LSA (dataset Airline Original).

2.5 Résultats LSA (dataset après undersampling)

Tableau 6. 27 - Résultat LSA sur dataset Airline (après undersampling)

Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
67.25%	40.39%	49.13%	44.33%

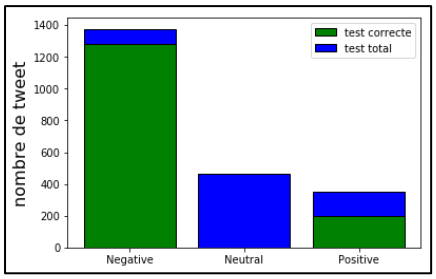

Figure 6. 27- Test correct par classe LSA (dataset Airline undersampling).

2.6 Résultats LSA (dataset après oversampling)

2.6.1 Résultats SMOTE

Tableau 6. 28 - Résultat LSA sur dataset Airline (après SMOTE)

Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
37.43%	45.83%	38.74%	41.99%

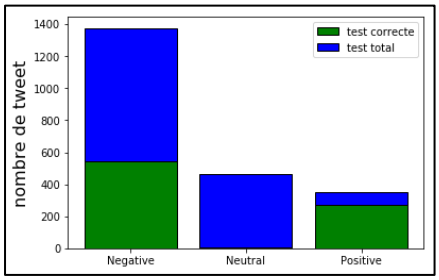

Figure 6. 28 - Test correct par classe LSA (dataset Airline SMOTE).

2.6.2 Semantic-Oversampling

Tableau 6. 29 - Résultat LSA sur dataset Airline (Semantic-Oversampling)

Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
65.43% Min-df =0.01 62.79%	38.35% 20.91%	43.03% 33.33%	40.56% 25.70%

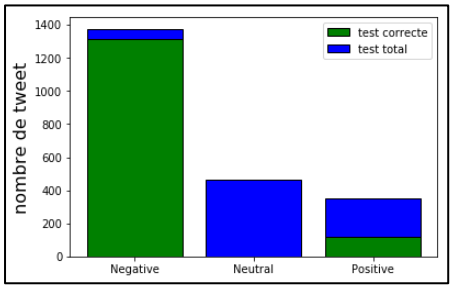

Figure 6. 29 - Test correct par classe LSA (dataset Airline Semantic-Oversampling).

2.7 Résultats K-means (dataset original)

Tableau 6. 30 - Résultat K-means sur dataset Airline (Original)

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
Window size : 22 Itération =1	**62.74%**	**33.33%**	**20.91%**	**25.70%**
Window size : 22 Itération =5	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 10	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 20	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 50	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération =100	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 300	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 500	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 600	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 900	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 1000	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 1500	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 2000	62.74%	33.33%	21.91%	26.70%
Window size : 22 Itération = 3000	62.74%	33.33%	21.91%	26.70%

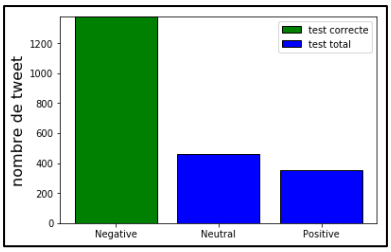

Figure 6. 30 - Test correct par classe K-means (dataset Airline Original).

2.8 Résultats K-means (dataset après undersampling)

Tableau 6. 31 - Résultat K-means sur dataset Airline (après undersampling).

paramètres	Accuracy (%)	Precision (%)	Recall (%)	F-score (%)
Window size : 22 Itération =1	48.79%	**36.17%**	**43.11%**	**39.34%**
Window size : 22 Itération =5	55.99%	35.06%	42.78%	38.54%
Window size : 22 Itération = 10	44.55%	26.14%	30.97%	28.35%
Window size : 22 Itération = 50	43.91%	21.73%	25.95%	23.66%
Window size : 22 Itération =100	44.27%	26.13%	30.92%	28.32%
Window size : 22 Itération = 300	**58.09%**	35.08%	42.85%	38.57%
Window size : 22 Itération = 500	44.55%	26.33%	31.21%	28.56%
Window size : 22 Itération = 600	43.91%	26.33%	31.21%	28.56%
Window size : 22 Itération = 900	57.91%	35.18%	42.99%	38.70%
Window size : 22 Itération = 1000	57.91%	35.15%	42.94%	38.66%
Window size : 22 Itération = 1500	44.36%	21.70%	26.05%	23.68%
Window size : 22 Itération = 2000	57.95%	35.14%	42.92%	38.64%
Window size : 22 Itération = 3000	44.87%	26.34%	31.24%	28.58%

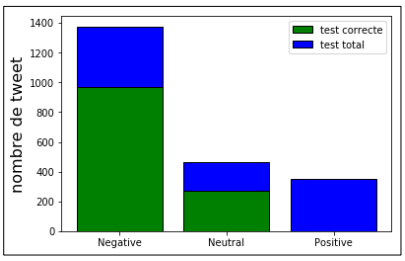

Figure 6. 31 - Test correct par classe K-means (dataset Airline undersampling).

2.9 Résultats K-means (dataset après oversampling)

2.9.1 Résultats SMOTE

Tableau 6. 32 - Résultat K-means sur dataset Airline (après SMOTE).

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)

Itération =1 Window size=25	43.27%	31.30%	39.49%	34.92%
Itération =5 Window size=25	47.46%	36.09%	**47.24%**	40.92%
Itération = 10 Window size=	**65.11%**	**43.81%**	40.39%	**42.03%**
Itération = 20 Window size=25	48.33%	36.19%	47.70	41.15%
Itération = 50 Window size=25	32.74%	17.87%	19.70%	18.74%
Itération =100 Window size=25	33.37%	18.05%	20.04%	18.99%
Itération = 300 Window size=25	44.09%	32.09%	40.07%	35.64%
Itération = 500 Window size=25	33.42%	18.02%	19.99%	18.95%
Itération = 600 Window size=25	65.11%	43.81%	40.39%	42.03%
Itération = 900 Window size=25	43.27%	31.99%	39.91%	35.52%
Itération = 1000 Window size=25	34.01%	18.50%	20.79%	19.58%
Itération = 1500 Window size=25	42.77%	31.77%	39.36%	31.77%
Itération = 2000 Window size=25	33.10%	17.98%	19.89%	18.89%
Itération = 3000 Window size=25	43.27%	31.30%	39.49%	34.92%

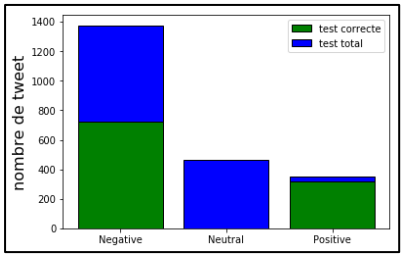

Figure 6. 32 - Test correct par classe K-means (dataset Airline oversampling).

2.9.2 SMOTE (word2vec)

Tableau 6. 33 - Résultat K-means sur dataset Airline (SMOTE avec word2vec).

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-score (%)**
Window size : 22 Itération =1	60.51%	27.52%	33.82%	30.35%
Window size : 22 Itération =5	30.64%	24.84%	30.68%	27.45%
Window size : 22 Itération = 10	46.37%	27.32%	35.26%	30.79%
Window size : 22 Itération = 20	55.35%	27.74%	35.00%	30.92%
Window size : 22 Itération = 50	44.09%	36.51%	46.36%	40.85%
Window size : 22 Itération =100	43.86%	**36.49%**	46.24%	40.79%

Window size : 22 Itération = 300	**61.55%**	21.90%	32.77%	26.25%
Window size : 22 Itération = 500	50.66%	20.68%	27.68%	23.67%
Window size : 22 Itération = 600	47.37%	36.23%	47.05%	40.94%
Window size : 22 Itération = 900	49.20%	36.12%	47.67%	41.10%
Window size : 22 Itération = 1000	43.82%	36.48%	46.21%	40.77%
Window size : 22 Itération = 1500	49.15%	35.97%	47.58%	40.97%
Window size : 22 Itération = 2000	49.29%	36.01%	**47.65%**	**41.02%**
Window size : 22 Itération = 3000	60.51%	27.52%	33.82%	30.35%

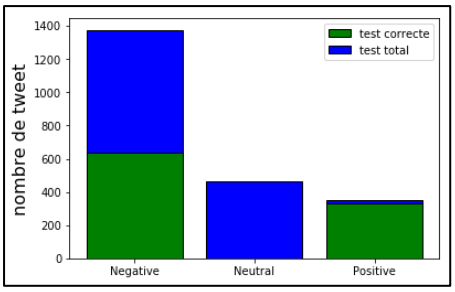

Figure 6. 33 - Test correct par classe K-means (dataset Airline SMOTE-Word2vec).

2.9.3 Semantic-Oversampling

Tableau 6. 34 - Résultat K-means sur dataset Airline (Semantic-Oversampling).

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-score (%)**
Itération =1 Window size=25	19.61%	31.45%	44.70%	2416 %
Itération =5 Window size=25 -- -- -- -- Count-df=0 .01	35.65% -- -- -- -- 36.11%	42.91% -- -- -- -- 21.35%	29.47 -- -- -- -- 22.00%	54.76% -- -- -- -- 21.67%
Itération = 10 Window size=	53.30%	**59.67%**	51.03%	55.01%
Itération = 20 Window size=25	31.32%	22.25%	20.73%	21.46%
Itération = 50 Window size=25	36.70%	43.21%	29.93%	35.37%
Itération =100 Window size=25	43.36%	36.65%	38.87%	37.73%
Itération = 300 Window size=25	31.32	22.25%	20.73%	2146%
Itération = 500 Window size=25	20.24%	20.62%	31.85%	25.04%
Itération = 600 Window size=25	35.75%	43.08%	29.62%	35.10%
Itération = 900 Window size=25	35.75%	43.08%	29.62%	35.10%
Itération = 1000 Window size=25	44.00%	36.28%	38.51%	37.37%
Itération = 1500 Window size=25	44.03%	36.28%	38.51%	37.37%
Itération = 2000 Window size=25	35.65%	43.07%	29.62%	35.10%
Itération = 3000 Window size=25	**53.76%**	59.56%	**51.13%**	**55.02%**

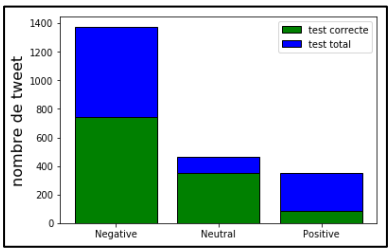

Figure 6. 34 - Test correct par classe K-means (dataset Airline Semantic-Oversampling).

2.10 Résultats CNN-Yoon (dataset Orignal)

Tableau 6. 35 - Résultat CNN-Yoon sur dataset Airline (Original).

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
epoch : 1 batchsize= 512 batchsize =100	64.70% -- --- --- -- 69.71%	41.42% -- --- --- -- 52.23%	52.77% -- --- --- -- 64.27%	46.41% -- --- --- -- 57.63%
epoch : 5 batchsize= 512	72.35%	54.20%	72.73%	62.12%
epoch : 10 batchsize= 512	74.72%	60.32%	71.66%	65.50%
epoch : 15 batchsize= 512	74.95%	58.64%	**74.68%**	65.70%
epoch : 20 batchsize= 512	75.63%	62.84%	71.71%	66.99%
epoch : 25 batchsize= 512	**77.04%**	64.44%	74.07%	68.92%
epoch : 30 batchsize= 512	77.04%	**64.78%**	73.86%	**69.02%**

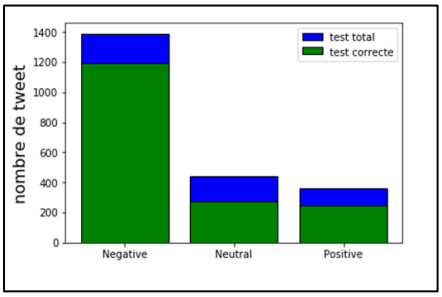

Figure 6. 35 - Test correct par classe K-means (dataset Airline Original).

2.11 Résultats CNN-Yoon (dataset après undersampling)

Tableau 6. 36 - Résultat CNN-Yoon sur dataset Airline (après undersampling).

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
epoch : 1 batchsize= 512	64.39%	40.29%	**69.12%**	50.90%
epoch : 5 batchsize= 512	**71.91%**	**71.66%**	67.65%	**69.59%**
epoch : 10 batchsize= 512	71.73%	69.98%	65.96%	67.91%
epoch : 15 batchsize= 512	70.27%	68.59%	64.19%	66.32%

epoch : 20 batchsize= 512	70.63%	68.40%	64.03%	66.15%
epoch : 25 batchsize= 512	70.00%	68.50%	64.13%	66.24%
epoch : 30 batchsize= 512	68.95%	67.35%	62.89%	65.04%

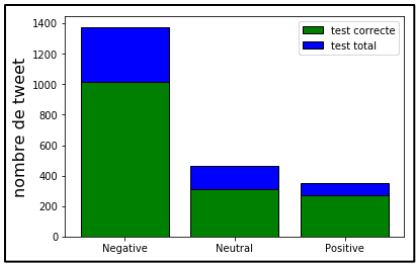

Figure 6. 36 - Test correct par classe K-means (dataset Airline undersampling).

2.12 Résultats CNN-Yoon (dataset après oversampling)

2.12.1 Résultats SMOTE

Tableau 6. 37 - Résultat CNN-Yoon sur dataset Airline (SMOTE).

paramètres	**Accuracy (%)**	**Precision(%)**	**Recall(%)**	**F-score (%)**
epoch : 1 batchsize= 512	71.82%	66.33%	69.69%	67.97%
epoch : 2 batchsize= 512	75.38%	70.56%	**71.51%**	71.03%
epoch : 3 batchsize= 512	**76.01%**	**70.95%**	70.43%	**70.69%**
epoch : 5 batchsize= 512	74.97%	69.43%	66.46%	67.91%
epoch : 10 batchsize= 512	73.55%	67.71%	65.40%	66.53%
epoch : 15 batchsize= 512	72.46%	66.19%	64.49%	65.33%
epoch : 20 batchsize= 512	73.01%	67.00%	65.21%	66.09%
epoch : 25 batchsize= 512	73.05%	66.51%	64.08%	65.27%
epoch : 30 batchsize= 512	72.05%	66.51%	63.24%	64.36%

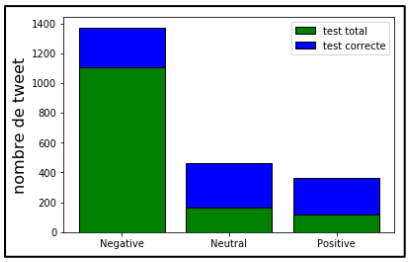

Figure 6. 37 - Test correct par classe CNN-Yoon (dataset Airline SMOTE).

2.12.2 Semantic-Oversampling

Tableau 6. 38 - Résultat CNN-Yoon sur dataset Airline (Semantic-Oversampling).

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
epoch : 1 batchsize= 512	72.78%	61.49%	66.99%	64.12%
epoch : 5 batchsize= 512	**76.06%** -- -- -- Min-df : 0.01: 73.83%	**69.97%** -- -- -- 67.20%	**68.35%** -- -- -- 67.87%	**69.15%** -- -- -- 67.53%
epoch : 10 batchsize= 512	75.28%	69.13%	66.98%	68.03%
epoch : 15 batchsize= 512	73.96%	67.36%	65.94%	66.64%
epoch : 20 batchsize= 512	73.37%	65.71%	66.40%	66.05%
epoch : 25 batchsize= 512	66.20%	65.13%	66.20%	65.66%
epoch : 30 batchsize= 512	73.14%	65.73%	65.96%	65.84%

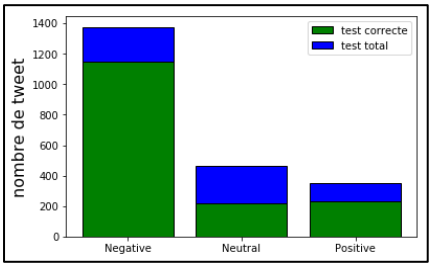

Figure 6. 38 - Test correct par classe CNN-Yoon (dataset Airline Semantic-Oversampling).

2.13 Résultats LSTM (dataset original)

Tableau 6. 39 - Résultat LSTM sur dataset Airline (Original).

paramètres	Accuracy (%)	Recall (%)	Precision (%)	F-score (%)
epoch : 1/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	63.30%	33,33%	21,09%	25,83%
epoch : 5/batchsize= 512 embed_dim = 32 lstm_dim = 196	69.89%	47.12%	71.79%	56.90%

Dropout = 0.5				
epoch:10/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	78.36%	71.03%	72.30%	71.66%
epoch:10/batchsize= 512 embed_dim = 64 lstm_dim = 196 Dropout = 0.5	77.91%	67.69%	72.18%	69.87%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_dim = 196 Dropout = 0.5	78.41%	70.58%	72.90%	71.72%
epoch : 10/batchsize= 512 embed_dim = 256 lstm_dim = 196 Dropout = 0.5	77.77%	71.08%	71.56%	71.32%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_dim = 256 Dropout = 0.5	**78.64%**	70.95%	73.10%	**72.01%**
epoch : 20/batchsize= 512 Embed_dim =128 lstm_dim = 192 Dropout = none	74.95%	68.62%	68.79%	68.70%
epoch :20/batchsize= 1000 Embed_dim =128 lstm_dim = 192 Dropout = 0.2	77.78%	70.94%	72.48%	71.70%
batchsize= 2000 embed_dim = 128 lstm_dim = 192 Dropout = 0.5 Optim : SGD/adam Adamax/Adadelta	30 epoch : 63.30% -- -- -- -- -- 30 epoch : 78.00% -- -- -- -- -- 30 epoch : 78.42% -- -- -- -- -- 30 epoch : 71.85% 60 epoch : 76.63% 120 epoch: 78.36%	21.09% -- -- -- -- 70.33% -- -- -- -- 70.49% -- -- -- -- 52.38% 66.56% 71.35%	33.33% -- -- -- -- 72.71% -- -- -- -- 72.91% -- -- -- -- 66.03% 70.71% 72.08%	25.84% -- -- -- -- -- 71.50% -- -- -- -- -- 71.68% -- -- -- -- -- 58.42% 68.57% 71.71%
epoch :30/batchsize= 2000 embed_dim = 128 lstm_dim = 192 Dropout = 0,5 Activation : sigmoïde	78.19%	69.10%	73.50%	71.23%

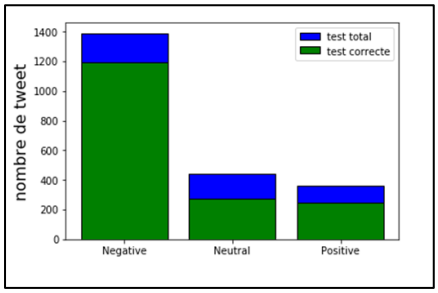

Figure 6. 39 - Test correct par classe LSTM (dataset Airline Original).

2.14 Résultats LSTM (dataset après undersampling)

Tableau 6. 40- Résultat LSTM sur dataset Airline (après undersampling).

paramètres	**Accuracy (%)**	**Recall (%)**	**Precision (%)**	**F-score (%)**
epoch : 1/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	36.89%	39.56%	35.03%	37.16%
epoch : 5/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	59.83%	38.59%	47.59%	42.62%
epoch: 10/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	71.00%	68.95%	**69.33%**	69.14%
epoch: 10/batchsize= 512 embed_dim = 64 lstm_dim = 196 Dropout = 0.5	**75.47%**	71.64%	69.00%	70.29%
epoch: 10/batchsize= 512 embed_dim = 128 lstm_dim = 196 Dropout = 0.5	74.69%	71.38%	69.20%	**70.27%**
epoch : 10/batchsize= 512 embed_dim = 256 lstm_dim = 196 Dropout = 0.5	72.96%	71.97%	66.46%	69.11%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_dim = 256 Dropout = 0.5	71.50%	72.38%	66.95%	69.56%
epoch : 20/batchsize= 512 Embed_dim =128 lstm_dim = 192 Dropout = none	70.04%	69.30%	64.33%	66.72%
epoch : 20/batchsize= 1000 Embed_dim =128 lstm_dim = 192 Dropout = 0.2	71.87%	71.41%	66.06%	68.63%
batchsize= 2000 embed_dim = 128 lstm_dim = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	30 epoch : 55.77% -- -- -- -- -- 30 epoch : 72.00% -- -- -- -- -- 30 epoch : 72.59% -- -- -- -- -- 30 epoch : 66.39% 60 epoch:73.69% 120 epoch:71.91% 200 epoch : 71.55%	41.71% -- -- -- -- 72.04% -- -- -- -- **72.45%** -- -- -- -- 61.58% 69.77% 71.99% 69.67%	33.97% -- -- -- -- - 67.14% -- -- -- -- - 67.85% -- -- -- -- - 65.38% 65.94% 67.09% 65.55%	37.44% -- -- -- -- -- 69.50% -- -- -- -- -- 70.08% -- -- -- -- -- 63.42% 67.80% 69.46% 67.55%
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_dim = 192 Dropout = 0,5 Activation : sigmoïde	72.96%	71.78%	68.35%	70.03%

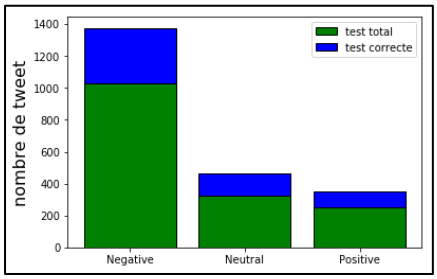

Figure 6. 40 - Test correct par classe LSTM (dataset Airline Undersampling).

2.15 Résultats LSTM (dataset après oversampling)

2.15.1 Résultats SMOTE

Tableau 6. 41 - Résultat LSTM sur dataset Airline (SMOTE).

paramètres	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-score (%)**
epoch : 1/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	55.51%	70.35%	45.83%	55.51%
epoch : 5/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	73.64%	70.47%	72.01 %	71.23%
epoch: 10/batchsize= 512 embed_dim = 32 lstm_output = 196 Dropout = 0.5	**76.38%**	**71.24%**	**71.40%**	**71.32%**
epoch: 10/batchsize= 512 embed_dim = 64 lstm_output = 196 Dropout = 0.5	75.56%	69.44%	68.31%	68.87%
epoch: 10/batchsize= 512 embed_dim = 128 lstm_output = 196 Dropout = 0.5	76.11%	70.92%	67.99%	69.42%
epoch : 10/batchsize= 512 embed_dim = 256 lstm_output = 196 Dropout = 0.5	75.33%	69.29%	67.38%	68.33%
epoch : 10/batchsize= 512 embed_dim = 128 lstm_output = 256 Dropout = 0.5	75.15%	68.43%	68.75%	68.59%
epoch : 20/batchsize= 512 Embed_dim =128 lstm_output = 192 Dropout = none	75.28%	6956%	6706%	68.29%
epoch : 20/batchsize= 1000 Embed_dim =128 lstm_output = 192 Dropout = 0.2	73.92%	68.33%	67.65%	67.99%

batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	30 epoch : 41.04% -- -- -- -- -- 30 epoch : 73.87% -- -- -- -- -- 30 epoch : 75.06% -- -- -- -- -- 30 epoch : 69.31%	40.58% -- -- -- -- 68.07% -- -- -- -- 69.71% -- -- -- -- 67.94%	41.29% -- -- -- -- 66.50% -- -- -- -- 69.37% -- -- -- -- 68.85%	40.93% -- -- -- -- 67.27% -- -- -- -- 69.54 % -- -- -- -- 68.39 %
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_output = 192 Dropout = 0,5 Activation : sigmoïde	67.17%	63.26%	64.53%	63.89%

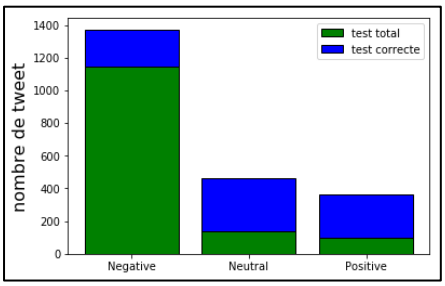

Figure 6. 41 - Test correct par classe LSTM (dataset Airline SMOTE).

2.15.2 Semantic-Oversampling

Tableau 6. 42 - Résultat LSTM sur dataset Airline (Semantic-Oversampling).

paramètres	**Accuracy (%)**	**Recall (%)**	**Precision (%)**	**F-score (%)**
epoch : 1/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	67.26%	51.40%	58.37%	54.66 %
epoch : 5/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	77.20%	71.87%	69.46%	70.65%
epoch: 10/batchsize= 512 embed_dim = 32 lstm_dim = 196 Dropout = 0.5	77.70 %	69.88%	**72.30 %**	71.07%
epoch: 10/batchsize= 512 embed_dim = 64 lstm_dim = 196 Dropout = 0.5	**77.84%** -- -- -- -- Min-df : 0.01 : 75.15%	70.35% -- -- -- -- 68.40%	72.11 % -- -- -- -- 69.04%	70.61% -- -- -- -- -- 68.72%
epoch: 10/batchsize= 512 embed_dim = 128 lstm_dim = 196 Dropout = 0.5	76.97%	70.49%	70.47%	70.48%
epoch : 10/batchsize= 512 embed_dim = 256 lstm_dim = 196 Dropout = 0.5	77.34 %	69.75%	71.11%	70.42%

epoch : 10/batchsize= 512 embed_dim = 128 lstm_dim = 256 Dropout = 0.5	76.29%	67.17%	70.38%	68.13%
epoch : 20/batchsize= 512 Embed_dim =128 lstm_dim = 192 Dropout = none	75.28%	66.96%	68.58 %	67.76%
epoch : 20/batchsize= 1000 Embed_dim =128 lstm_dim = 192 Dropout = 0.2	75.60%	68.04%	69.26%	68.64 %
batchsize= 2000 embed_dim = 128 lstm_dim = 192 Dropout = 0.5 Optim:SGD/adam Adamax/Adadelta	30 epoch : 27.18% -- -- -- -- 30 epoch : 76.15% -- -- -- -- 30 epoch : 77.52% -- -- -- -- 30 epoch : 74.42% 60 epoch : 77.06%	38.83% -- -- -- -- 68.02% -- -- -- -- 70.85% -- -- -- -- 68.94% **71.58%**	40.35% -- -- -- -- 69.74% -- -- -- -- 71.48% -- -- -- -- 71.08% 70.94%	39.57% -- -- -- -- 68.87% -- -- -- -- 71.16 % -- -- -- -- 69.06% **71.26%**
epoch : 30/batchsize= 2000 embed_dim = 128 lstm_dim = 192 Dropout = 0,5 Activation : sigmoïde	76.42%	68.86%	69.97%	69.41%

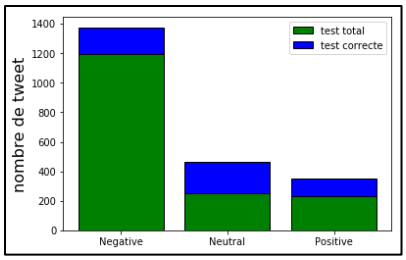

Figure 6. 42 - Test correct par classe LSTM (dataset Airline Semantic-Oversampling).

BIBLIOGRAPHIE

Acosta, J.M., Lamaute, N., Luo, M., Finkelstein, E., & Cotoranu, A. (2017). Sentiment Analysis of Twitter Messages Using Word 2 Vec. Récupéré 10 Avril, 2019, de https://www.semanticscholar.org/paper/Sentiment-Analysis-of-Twitter-Messages-Using-Word-2-Acosta-Lamaute/784d1b2aebda3b80567bf8244e89499c31cf42a9

Al-Ayyoub, M., Khamaiseh, A. A., Jararweh, Y., & Al-Kabi, M. N. (2018). A comprehensive survey of arabic sentiment analysis. *Information Processing & Management*, *56*(2), 320–342. DOI : 10.1016/j.ipm.2018.07.006.

Al-Khafaji, H., Habeeb, A. (2017). Efficient Algorithms for Preprocessing and Stemming of Tweets in a Sentiment Analysis System, Journal of Computer Engineering, 19(3), 44-50, DOI: 10.9790/0661-1903024450

Alvarez-Melis, D., & Saveski, M. (2016). Topic Modeling in Twitter: Aggregating Tweets by Conversations. ICWSM.

Ah-Pine, J., & Soriano-Morales, E. (2016). A Study of Synthetic Oversampling for Twitter Imbalanced Sentiment Analysis. DMNLP@PKDD/ECML.

Antenucci, A., Handy, G.,Modi, A, Tinkerhess, M. (2011), classification of tweets via clustering of hashtags, EECS 545 final project.

Arnaud. 2018, https://outweb.eu/du-machine-learning-dans-mes-cocktails/ [Consulté le 30 juin 2019].

Banerjee, S., & Pedersen, T. (2002). An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. CICLing.

Beverungen G., Kalita J. (2011). « Evaluating Methods for Summarizing Twitter Posts », Proceedings of the 5th AAAI ICWSM.

Fortuner, B., Viana, M., kowshik, B. (2018). Loss Functions. https://github.com/bfortuner/ml-cheatsheet/blob/master/docs/loss_functions.rst [Consulté le 22 Mai 2019].

Boom, C.D., Canneyt, S.V., & Dhoedt, B. (2015). Semantics-driven Event Clustering in Twitter Feeds. #MSM.

Britz, D. (2015) https://github.com/dennybritz/cnn-text-classification-tf (consulté le 10 Mars 2019)

Chaubard, F., Mundra, R., & Socher, R. (2016). CS 224D: Deep Learning for NLP1.

Chavent, M. (2013).Classification automatique. Université de Bordeaux.

Chen, S., Santoso, A., Lee, Y., & Wang, J. (2015). Latent dirichlet allocation based blog analysis for criminal intention detection system. *2015 International Carnahan Conference on Security Technology (ICCST)*, 73-76.

Chen, T., Lu, Q., Xu, R., Liu, B., & Xu, J. (2014). WEMOTE - Word Embedding based Minority Oversampling Technique for Imbalanced Emotion and Sentiment Classification.

Chevalier, F., Bellac, J. (2013). La classification. https://perso.univ-rennes1.fr/valerie.monbet/ExposesM2/2013/Classification2.pdf.

Chollet, F. (2018). Deep learning with python, united state,New York Manning.

Chrislb (2005). Structure d'un neurone artificiel. https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels#/media/Fichier:ArtificialNeuronModel_francais.png (consulté le 20 mai 2019).

Clark, S. (2013). Topic Modelling and Latent Dirichlet Allocation. https://www.cl.cam.ac.uk/teaching/1213/L101/clark_lectures/lect7.pdf (consulté le 29 Mai 2019).

Daneshvar, S., & Inkpen, D. (2018). Gender Identification in Twitter using N-grams and LSA: Notebook for PAN at CLEF 2018. CLEF.

Dasgupta, S., Kumar, A., Das, D., Naskar, S.K., & Bandyopadhyay, S. (2016). Word Embeddings for Information Extraction from Tweets. FIRE.

Deloche, F. (2017). A diagram for a one-unit recurrent neural network (RNN). From bottom to top : input state, hidden state, output state. U, V, W are the weights of the network. Compressed diagram on the left and the unfold version of it on the right. https://upload.wikimedia.org/wikipedia/commons/b/b5/Recurrent_neural_network_unfold.svg .(consulté le 20 Mai 2019).

Deloche, F. (2017). diagram for a one-unit Long Short-Term Memory (LSTM). From bottom to top : input state, hidden state and cell state, output state. Gates are sigmoïds or hyperbolic tangents. Other operators : element-wise plus and multiplication. Weights are not displayed. https://fr.wikipedia.org/wiki/Fichier:Long_Short-Term_Memory.svg.(consulté le 20 Mai 2019).

Derczynski, L. (2016). Complementarity, F-score, and NLP Evaluation. LREC.

Donadio, J. (2018). Intelligence artificielle RAPPORT THEORIQUE ET PRATIQUE.

[Du & Huang, 2018] Du, C., & Huang, L. (2018). Classification Research with Attention-based Recurrent Neural Networks, journal of computers communications & control, 13(1), 50-61, DOI: 10.15837/ijccc.2018.1.3142

D'Urso, J. (2018). What's the least popular emoji on Twitter?. Tiré de https://www.bbc.com/news/blogs-trending-44952140 [Consulté le 1 Février 2019].

valuations, e. (2015). a review on evaluation metrics for data classification evaluations.

Imran, M., & Srinivasa, V. (2018). A Novel Technique on Class Imbalance Big Data using Analogous under Sampling Approach. *International Journal of Computer Applications*, *179*(33), DOI : 10.5120/ijca2018916743

Friedemann, V. (2015). Clustering a Customer Base Using Twitter Data.

Gao, D., Li, W., Cai, X., Zhang, R., and Ouyang, Y. (2014). *Sequential summarization : A full view of twitter trending topics*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning (Adaptive Computation and Machine Learning series). MIT Press.

Guan, P. (2016). K-means Document Clustering Based on Latent Dirichlet Allocation.

Hacibeyoglu, M., Ibrahim,M. H. (2018) *The Effect of Over-sampling and UndersamplingTechniques in Medical Datasets,* International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18), May 11-13, 2018 Safranbolu, Turkey.

HRcommons. (2009) https://fr.wikipedia.org/wiki/Perceptron_multicouche#/media/Fichier:Perceptron_4layers.png (consulté le 20 Mai 2019).

Hu, F., & Li, H. (2013). A Novel Boundary Oversampling Algorithm Based on Neighborhood Rough Set Model: NRSBoundary-SMOTE. *Mathematical Problems in Engineering*, *2013*, 1–10. DOI : 10.1155/2013/694809.

Hull, D. (1994). Improving Text Retrieval for the Routing Problem using Latent Semantic Indexing. SIGIR '94, 282–291. https://doi.org/10.1007/978-1-4471-2099-5_29

Iyyer, M., Manjunatha, V., Boyd-Graber, J.L., & Daumé, H. (2015). Deep Unordered Composition Rivals Syntactic Methods for Text Classification. ACL.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. ACL.

Kantrowitz, M., Mohit, B., and Mittal, V. (2000) "Stemming and Its Effects on TFIDF Ranking",In Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 357–359, Athens, Greece.

Kaur, P., & Gosain, A. (2017). Comparing the Behavior of Oversampling and Undersampling Approach of Class Imbalance Learning by Combining Class Imbalance Problem with Noise. Advances in Intelligent Systems and Computing, 23–30. DOI : 10.1007/978-981-10-6602-3_3.

Kingma, D.P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. CoRR,

abs/1412.6980.

Klema, V., & Laub, A.J. (1980). The singular value decomposition: Its computation and some applications.

Ioffe, S., Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift.*ICML*

Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). Recurrent Convolutional Neural Networks for Text Classification. AAAI.

Lee, K., Palsetia, D., Narayanan, R., Patwary, Md. M. A., Agrawal, A., & Choudhary, A. (2011). Twitter Trending Topic Classification. 2011 IEEE 11th International Conference on Data Mining Workshops. DOI : 10.1109/ICDMW.2011.171.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In SIGDOC '86: Proceedings of the 5th annual international conference on Systems documentation, pages 24-26, New York, NY, USA. ACM.

Liu, B. (2012). Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies. DOI:10.2200/S00416ED1V01Y201204HLT016.

Liu, L., Han, M., Zhou, Y., & Wang, Y. (2018). LSTM Recurrent Neural Networks for Influenza Trends Prediction. *Bioinformatics Research and Applications*, 259–264. DOI: 10.1007/978-3-319-94968-0_25

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations.

Mehrotra, R., Sanner, S., Buntine, W., & Xie, L. (2013). Improving LDA topic models for microblogs via tweet pooling and automatic labeling. *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '13*. DOI:10.1145/2484028.2484166

Meyer, D. (2016). How exactly does word2vec work? https://pdfs.semanticscholar.org/49ed/be35390224dc0c19aefe4eb28312e70b7e79.pdf (consulté le 20 Mai 2019).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J. 2013. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS 2013.

Minsky-Papert. (*1969*). https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d (consulté le 20 Mai)

Morchid, M., Josselin, D., Portilla, Y., Dufour, R., Altman, E., & Linarès, G. (2015). A TOPIC MODELING BASED REPRESENTATION TO DETECT TWEET LOCATIONS. EXAMPLE OF THE EVENT "JE SUIS CHARLIE". ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XL-3/W3, 629–634. DOI: 10.5194/isprsarchives-XL-3-W3-629-2015.

Nguyen, D.Q., Billingsley, R., Du, L., & Johnson, M. (2015). Improving Topic Models with Latent Feature Word Representations. Transactions of the Association for Computational Linguistics, 3, 299-313.

Pennington, J., Socher, R., & Manning, C.D. (2014). Glove: Global Vectors for Word Representation. EMNLP.

Petitjean, G. (2006). INTRODUCTION AUX RESEAUX DE NEURONES. https://www.lrde.epita.fr/~sigoure/cours_ReseauxNeurones.pdf.[consulté le 16 juin 2019].

Reed, R., & Marks, R. J. (1999). Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. MIT Press.

Río, S.D., López, V., Benítez, J.M., & Herrera, F. (2014). On the use of MapReduce for imbalanced big data using Random Forest. Inf. Sci., 285, 112-137.

Rouissi, J. (2001). Les effets de réseau en bibliothèques : pour une meilleure prise en compte des coûts et avantages qualitatifs de la coopération. Tiré de http://theses.univ-lyon2.fr/documents/getpart.php?id=lyon2.2001.rouissi_j&part=178747 [Consulté le 30 Mai 2019]

Saha, S. (2018). Types of Pooling, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (consulté le 20 Mai 2019).

Salton, G., McGill, M. J. (1983) Introduction to Modern Information Retrieval.

Schütze, H. (1998). Automatic Word Sense Discrimination. Computational Linguistics, 24, 97-123.

Sharma, V., Kulshreshtha, R., Singh, P., Agrawal, N., & Kumar, A. (2015). *Analyzing Newspaper Crime Reports For Identification Of Safe Transit Paths*. HLTNAACL.

Shi, T., Kang, K., Choo, J., & Reddy, C.K. (2018). Short- TextTopic Modeling via Non-negative Matrix Factorization Enriched with Local Word-ContextCorrelations. WWW.

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. Australian Conference on Artificial Intelligence., DOI:10.1007/11941439_114.

Somasundaram, A., Reddy, S. (2016). Data Imbalance: Effects and Solutions for Classification of Large and Highly Imbalanced Data. Proc. of 1st International Conference on Research in Engineering.

Soni, R., & Mathai, K.J. (2015). Improved Twitter Sentiment Prediction through Cluster-then-Predict Model. CoRR, abs/1509.02437.

Sravanthi, P.A., & SRINIVASU, D.B. (2017). SEMANTIC SIMILARITY BETWEEN SENTENCES. International Research Journal of Engineering and Technology.

Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., & Demirbas, M. (2010). Short text classification in twitter to improve information filtering. *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '10*. DOI : 10.1145/1835449.1835643.

Steinberger, J., Ježek, K. (2004). Using Latent Semantic Analysis in Text Summarization and Summary Evaluation. https://www.researchgate.net/publication/220017752_Using_Latent_Semantic_Analysis_in_Text_Summarization_and_Summary_Evaluation (Consulté le 20 Mai 2019).

Triguero, I., Galar, M., Bustince, H., & Herrera, F. (2017). A first attempt on global evolutionary undersampling for imbalanced big data. *2017 IEEE Congress on Evolutionary Computation (CEC)*. DOI : 10.1109/CEC.2017.7969553.

Wang, S.-H., Ding, Y., Zhao, W., Huang, Y.-H., Perkins, R., Zou, W., & Chen, J. J. (2016). Text mining for identifying topics in the literatures about adolescent substance use and depression. *BMC Public Health*, *16*(1). DOI : 10.1186/s12889-016-2932-1.

Weigend, A.S., Wiener, E.D., & Pedersen, J.O. (1999). Exploiting Hierarchy in Text Categorization. Information Retrieval, 1, 193-216.

Yoon, K. (2014). (2014). Convolutional Neural Networks for Sentence Classification. *EMNLP*.

Zhang, Y., & Wallace, B.C. (2017). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. IJCNLP.

Zellig, H. (1954). Distributional Structure, WORD, 10:2-3, 146-162, DOI:10.1080/00437956.1954.11659520.

Zhao, Y. (2011). R and Data Mining: Examples and Case Studies. Academic Press.

Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., & Xu, B. (2016). Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. ACL.