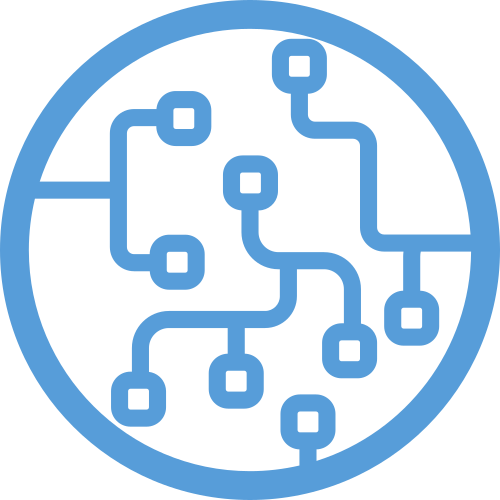
**Financial Sentiments**

****

***Projet P3***

***Capocasale Romain***

***INF3dlm-A***

***He-Arc, Suisse***

***2019***

***Abstract***

Table des matières

[1 Introduction 5](#_Toc29839739)

[2 Cahier des charges 5](#_Toc29839740)

[3 Planning 7](#_Toc29839741)

[3.1 Planning initial 7](#_Toc29839742)

[3.2 Planning final 8](#_Toc29839743)

[4 Analyse 9](#_Toc29839744)

[4.1 Etat de l’art 9](#_Toc29839745)

[4.2 Schéma du pipeline des processus 9](#_Toc29839746)

[5 Métrique d’évaluation des modèles 10](#_Toc29839747)

[5.1 Métrique de régression 10](#_Toc29839748)

[5.1.1 RMSE 10](#_Toc29839749)

[5.1.2 MAE 10](#_Toc29839750)

[5.2 Métrique de classification 10](#_Toc29839751)

[5.2.1 La précision 10](#_Toc29839752)

[5.2.2 Le rappel 11](#_Toc29839753)

[5.2.3 L’exactitude 11](#_Toc29839754)

[6 Jeu de données 11](#_Toc29839755)

[6.1 Titres d’articles financier + analyse des données 11](#_Toc29839756)

[6.1.1 Analyse des données 11](#_Toc29839757)

[6.2 Tweets financiers 11](#_Toc29839758)

[6.2.1 Analyse des données 12](#_Toc29839759)

[6.3 Tweets quelconques + analyse des données 12](#_Toc29839760)

[7 Analyseur de sentiment présent sur le web 12](#_Toc29839761)

[8 Solutions développées 13](#_Toc29839762)

[8.1 Prétraitement des données 13](#_Toc29839763)

[8.1.1 Traitement du texte 13](#_Toc29839764)

[8.1.1.1 Spacy 13](#_Toc29839765)

[8.1.1.2 Gensim 14](#_Toc29839766)

[8.2 Représentation d’une phrase 14](#_Toc29839767)

[8.2.1 TF-Idf 14](#_Toc29839768)

[8.2.2 Word2vec 15](#_Toc29839769)

[8.2.2.1 Gensim 15](#_Toc29839770)

[8.2.2.2 Word2vec financiers 16](#_Toc29839771)

[8.2.2.3 Glove 18](#_Toc29839772)

[8.2.3 Padding 18](#_Toc29839773)

[8.3 Modèle de machine learning 19](#_Toc29839774)

[8.3.1 Modèle de régression 19](#_Toc29839775)

[8.3.2 Modèle de classfication 19](#_Toc29839776)

[8.4 Modèle de deep learning 19](#_Toc29839777)

[8.5 Évaluation des modèles de machines learning 20](#_Toc29839778)

[8.5.1 Titres d’articles financiers 20](#_Toc29839779)

[8.5.1.1 Modèle avec le Tf-Idf 20](#_Toc29839780)

[8.5.1.2 Modèle avec Doc2vec de Gensim 21](#_Toc29839781)

[8.5.1.3 Modèle avec le Word2Vec financier 22](#_Toc29839782)

[8.5.2 Tweets financiers 23](#_Toc29839783)

[8.5.2.1 Modèles avec le Tf-Idf 23](#_Toc29839784)

[8.5.2.2 Modèle avec Doc2vec de Gensim 24](#_Toc29839785)

[8.5.2.3 Modèle avec le Word2Vec financier 24](#_Toc29839786)

[8.5.2.4 Classificateur avec Tf-Idf 24](#_Toc29839787)

[8.5.2.5 Calcul du score sur les titres d’articles financiers 25](#_Toc29839788)

[8.5.3 Tweets quelconques 26](#_Toc29839789)

[8.6 Évaluation des modèles de deep learning 26](#_Toc29839790)

[8.6.1 LSTM avec padding 26](#_Toc29839791)

[8.6.2 GRU avec padding 27](#_Toc29839792)

[8.6.3 CNN avec Word2Vec de Glove 27](#_Toc29839793)

[8.6.4 RNN avec Word2vec financier 28](#_Toc29839794)

[8.7 Conclusion des solutions développés 29](#_Toc29839795)

[9 Approfondissement possible 30](#_Toc29839796)

[9.1 Utilisation de Bert 30](#_Toc29839797)

[9.2 Agrandissement du jeu de données 30](#_Toc29839798)

[10 Conclusion 30](#_Toc29839799)

[11 Bibliographie 30](#_Toc29839800)

[11.1 Internet 30](#_Toc29839801)

[11.2 Livres 30](#_Toc29839802)

# Introduction

Dans le cadre du 5ème semestre de Bachelor dans la filière développement logiciel et multimédia, nous avons pour objectif de réaliser un projet individuel qui permet de nous préparer le mieux possible au travail de Bachelor. J’ai alors sélectionné un projet dans le domaine du machine learning.

Dans le cadre du machine learning, le domaine de l'analyse de sentiments dans les textes a pris beaucoup d'intérêt ces dernières années grâce au succès et aux bonnes performances des algorithmes de classification de texte. Dans un premier temps cette technologie a été utilisé pour analyser les sentiments dans des domaines tels que les commentaires de films, les hôtels, les restaurants, etc…

Le but du projet est d’appliquer cette technologie d’analyse de sentiment au domaine du marché financier dans le but de dire si un texte à plutôt un sentiment positif ou négatif. Ce sentiment pourrait alors être utilisé dans le domaine du FOREX (Foreign Exchange Market). Une information perçue comme positive pourrait indiquer une montée d’une devise par rapport à une autre tandis qu’une information négative pourrait en indiquer la chute.

L’objectif du projet est de créer un analyseur de sentiment pour le domaine de la finance. Le modèle sera entrainé avec des textes et vocabulaires propres au thème de la finance. Dans un premier temps, les solutions d’analyse de sentiment présente sur le web seront analysé pour résoudre ce problème. Dans un second temps, le but est de créé différents modèles de machine learning dans le but de prédire un sentiment par rapport à un texte financier.

# Cahier des charges

Le but du projet est de créer un modèle d'analyse de sentiment pour le domaine de la finance. Un analyseur de sentiment est un programme informatique qui en entrée demande une phrase ou un texte et en sortie indique un sentiment pour cette phrase sur une échelle de -1 à 1. Une valeur de -1 pour un sentiment négatif et 1 pour un sentiment positif. L’intervalle peut être discret ou continue.

Étant donné que le domaine de la finance dispose de son propre vocabulaire, il est intéressant de développer un modèle spécialisé dans ce secteur dans le but d'obtenir de meilleur résultat qu'un analyseur de sentiment global qui aurait été entraîné à faire des prédictions sur tout le vocabulaire de la langue anglaise.

Pour l'entraînement et le test du modèle, plusieurs sources de donnée différentes ont été fourni notamment, un ensemble de titres d'article de finance avec la valeur du sentiment correspondant. Les modèles seront réalisés en Python.

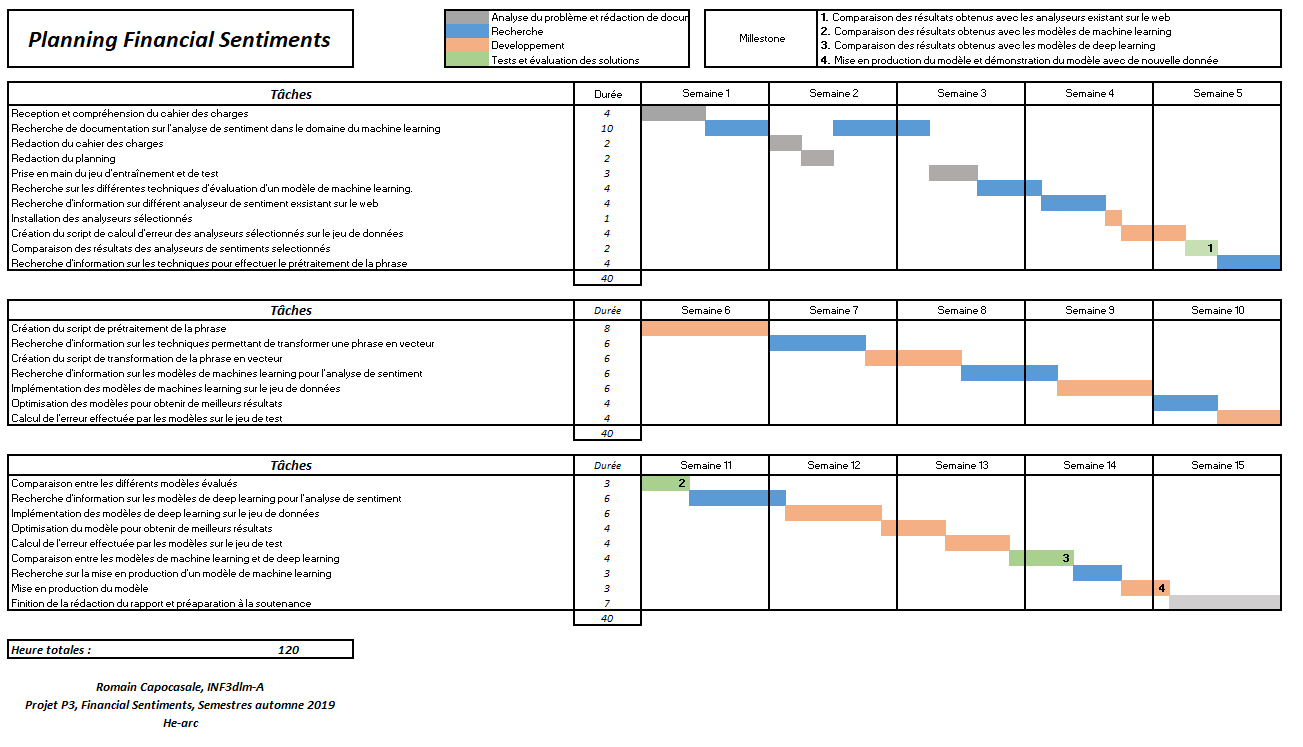
Pour ce projet, il faudra dans un premier temps, comparer le score des analyseurs de sentiment existant sur le web avec les jeux de donnée fourni. Par la suite, le but est de développer un modèle spécialisé dans l'analyse de sentiment pour la finance. La première étape consistera à prétraiter la phrase pour enlever la ponctuation, les déterminants, etc. Il faudra par la suite, transformer la phrase en vecteur dans le but d'être utilisé dans un modèle de machine learning. Différents modèles de machine learning devront être implémentés et testés à l'aide des jeux de donnée fourni. Par la suite, un modèle plus complexe utilisant des techniques de deep learning pourra être implémenté. Il faudra également comparer ces différents modèles entre eux pour sélectionner celui effectuant le moins d'erreurs de prédiction.

Le but final (hors projet), serait d'utiliser ce modèle d'analyse de sentiment dans le but de suivre la tendance haussière et baissière dans les pairs de devises (par exemple USD/CHF et EUR/USD). Des sources textuelles seraient récoltées depuis le site de Reuters et d'autres canaux financiers spécialisés dans le but d'en prédire un sentiment (entre -1 et 1) d'une période de trading (une heure, une demi-journée, journée, semaine ...) et relatif à un sujet (par exemple Brexit, échanges commerciaux avec la Chine, ....).

Ce classificateur avec d'autres en cours de développement, sera intégré dans un outil de prédiction du taux change.

# Planning

## Planning initial



## Planning final

# Analyse

## Etat de la technologie

Il existe de nombreux analyseur de sentiment sur le web comme par exemple : Textblob, VADER (nltk), etc. Ces analyseurs sont très simples à utiliser, il suffit d’indiquer la phrase et la librairie va indiquer la polarité de la phrase entre -1 et 1. Ces analyseurs ont cependant été entrainé sur tout le vocabulaire de langue anglaise et non sur le vocabulaire de la finance spécifiquement. Ils auront alors tendance à commettre plus d’erreur qu’un classificateur entrainé sur un vocabulaire spécifique. Un modèle entrainé avec un vocabulaire spécifique donnera une pondération plus importante aux mots qu’il considère plus important dans le contexte donné.

Si on regarde dans le domaine de la finance spécifiquement, d’après mes recherches ils n’existent pas de libraire ou d’analyseur de sentiment spécifique à la finance. On trouve sinon pas mal d’article qui explique l’utilité du NLP (Natural Language Processing) appliqué au domaine de la finance.

Avec ces différentes recherches, on constate donc qu’il serait intéressant de créer un modèle de Machine Learning entrainé avec un vocabulaire spécifique à la finance.

## Schéma du pipeline des processus

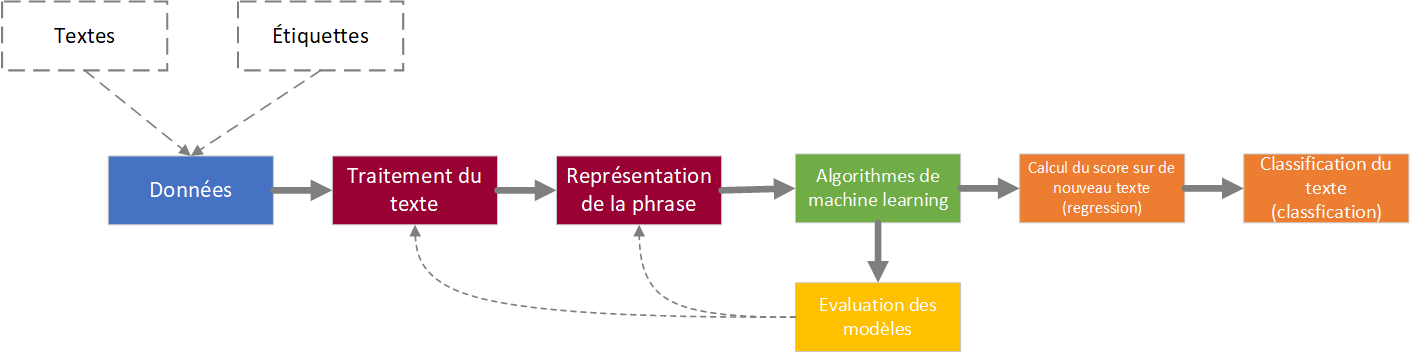


Figure - Machine Learning pipeline

Ce schéma montre les différentes étapes qui seront effectués dans ce projet dans le but d’obtenir un modèle de machine learning.

Dans un premier temps, les données provenant de différentes sources devront être recueillis puis séparé entre les textes (phrase à analyser) et les étiquettes (sentiment pour une phrase). C’est également dans cette étape que les données devront être analysé dans le but d’en ressortir les tendances, subtilité, etc.

Dans un second temps, les données doivent être prétraités, dans le but de pouvoir être introduit dans un algorithme de machine learning. Le texte doit d’abords être traité dans le but de retiré le bruit. C’est-à-dire enlever la ponctuation, enlever certains mots, etc.

Après avoir traité le texte, il faut maintenant trouver une manière de représenter les phrases sous forme de nombre. En effet, les algorithmes de machine learning ne peuvent travailler qu’avec des nombres et non du texte. Différentes techniques comme le word2vec et le tfidf pourront être utilisé pour cette étape.

A présent, différents algorithmes de machines learning doivent être étudié et implémenté pour résoudre le problème. Les données seront alors séparées en données d’entrainement et de test. Le modèle sera alors conçu à partir du jeu d’entrainement.

Une fois le modèle conçu, il faut l’évaluer avec les données de test et calculer différents scores qui permettront de se faire une idée de la fiabilité ou non du modèle. Selon les scores obtenus, il peut être intéressant de modifier certains paramètres des étapes vu précédemment dans le but d’améliorer les modèles. Certains paramètres du modèle (hyper-paramètres) peuvent aussi être adapté et modifié.

Une fois qu’un modèle satisfaisant a été obtenu, il est maintenant possible d’effectuer de nouvelle prédiction sur de nouveau texte. Le modèle prédira un score entre -1 et 1 sur un intervalle continue pour ce texte (regression) ou sur un intervalle discret (classification).

# Métrique d’évaluation des modèles

Avant de commencer à parler des solutions développées, il est important de définir les métriques qui nous permettront d’évaluer notre modèle. C’est grâce à cela, qu’on pourra dire si un modèle est viable ou non.

## Métrique de régression

### RMSE

* La RMSE (Root Mean Square Error) est la racine carrée de l’erreur quadratique moyenne, elle permet de donner une idée de l’importance des erreurs de prédictions commise par le système.

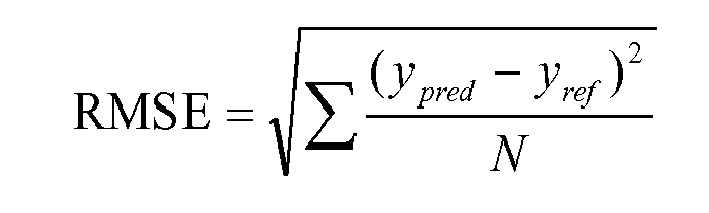


Figure - RMSE formule

### MAE

* La MAE (Mean Absolute Error) est l’erreur absolue moyenne, elle permet de donner moins d’importance au valeurs extrêmes.

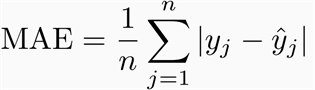


Figure - MAE formule

## Métrique de classification

Pour les métriques de classification il faut d’abords définir ce que sont :

* Les vrais négatifs (TN) : C’est le nombre de phrases négatives qui sont classé correctement dans la classe négative.
* Les faux positifs (FP) : C’est le nombre de phrases négative qui sont classé incorrectement c’est-à-dire dans la classe positive.
* Les faux négatifs (FN) : C’est le nombre de phrases positives qui sont classé incorrectement c’est-à-dire dans la classe négative.
* Les vrais positifs (TP) : C’est le nombre de phrases positives qui sont classé correctement dans la classe positive.

### La précision

La formule de la précision est :

Figure - Précision formule

### Le rappel

La formule du rappel est :

Figure - Rappel formule

Pour ce projet il est important d’avoir un bon compromis entre la précision et le rappel. C’est-à-dire que l’on souhaite que la précision et le rappel ai un peu près la même valeur. Il faut un bon compromis car il est autant important de repérer les phrases négatives que positives.

### L’exactitude

L’exactitude en anglais « accuracy » corresponds au nombre de prédiction correcte divisé par le nombre total d’observation.

# Jeu de données

## Titres d’articles financier

Deux sources différentes de données m’ont été fourni[[1]](#footnote-1) [[2]](#footnote-2). C’est le premier jeu de données que j’ai eu ma disposition pour ce projet. Les fichiers était au format JSON. Les 2 jeux de données contenaient chacun une colonne avec la phrase et le sentiment correspondant. Cependant, chaque jeu de données avait des colonnes supplémentaires que l’autre jeu de donnée n’avait pas avec d’autres informations. Un contenait par exemple l’entreprise qui était le sujet du texte et dans l’autre jeu de données une colonne contenait la provenance des textes. Même si ces informations auraient pu être intéressant à inclure dans le modèle, j’ai préféré ne garder que les colonnes avec la phrase et le sentiment correspondant dans le but d’avoir exactement les mêmes colonnes pour les deux jeux de données. Cela me permet d’accroitre la taille du jeu de données final. De plus, si d’autre variables avaient été inclus dans le modèle, il aurait fallu les renseigner lors de future prédiction ce qui peut être contraignant.

J’ai alors créé un script python (présent sous dataset/headlines\_dataset/rebuild\_data.py) permettant de récupérer les deux jeux de donnés et d’en ressortir un unique avec seulement les informations sélectionnées précédemment. Les données retournées sont sous forme d’un fichier au format JSON.

### Analyse des données

Une fois les données récupérées, il est important d’effectuer une analyse des données dans le but de voir différentes statistiques sur les données, des tendances ou des subtilités. Ceci dans le but de bien comprends comment elles sont structurées, afin de construire un modèle efficace. Il est important de noter que les statistiques ci-dessous concerne les données non prétraitées.

On remarque alors différentes statistiques sur les données, voici les plus importantes (pour plus de statistiques, consulter le notebook) :

* Il y a en tout 2866 lignes (2866 phrases et 2866 sentiment correspondant). On remarque qu’il y a assez peu de données.
* Il y a 1756 phrases positives, 1045 phrases négatives et 65 phrases neutres. On remarque alors qu’il y a plus de phrase positive que négative.
* Il y a environ 8 mots par phrase. Ce qui est une longueur courte pour une phrase.

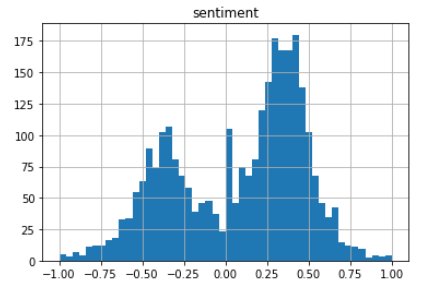
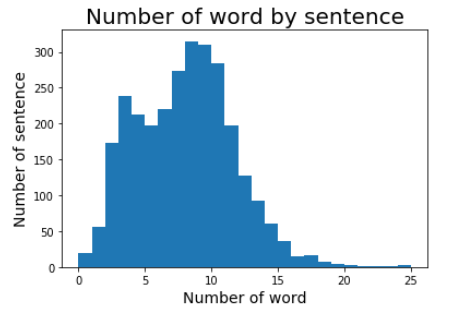


Figure – Nombre de phrase par polarité (titres articles financiers)

Figure - Nombre de mot par phrase (titres articles financiers)

Sur la figure 7 un histogramme avec sur l’axe X, la polarité d’une phrase et sur l’axe Y la quantité de phrase. Sur la figure 6 un graphe du nombre de mot par phrase.

## Tweets financiers

Mon second jeu de données provient du repository GitHub suivant : <https://github.com/AndMu/Market-Wisdom>. J’ai reçu ce jeu de données à la 12ème semaine du projet. Ces données ont été récolté par un robot sur twitter. Le robot avait pour but d’analyser les tweets financiers et de compter le nombre d’adjectif dans les catégories suivantes :

* Anger -> négatif
* Anticipation -> neutre
* Disgust -> négatif
* Fear -> négatif
* Joy -> positif
* Sadness -> négatif
* Surprise -> neutre
* Trust -> positif

Ces différentes données proviennent de plusieurs fichiers csv différents. Il a d’abords fallu remettre toutes ces différentes données ensemble puis faire une moyenne pondérer entre avec les différents adjectifs. Les adjectifs positifs avec une pondération de 1 et -1 pour les négatifs. Par exemple, si une phrase contient 2 adjectifs de type Trust, 1 de type Anger et 3 de type Sadness, la polarité du tweet correspondant sera de (2 + (-4)) / 6 = -0.33.

Ces différentes opérations se trouvent dans le notebook : *MarketDataPreProcess.ipynb.*

### Analyse des données

Ce jeu de données est de taille importante. Il contient plus de 1,2 million de tweets. Voici différentes statistiques sur ces données :

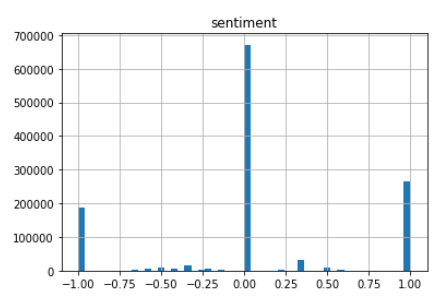
* Il y a 315'852 phrases positive, 238'591 phrases positives et 671'275 phrases neutres.
  + On remarque qu’il y a énormément de phrase neutre. Les phrases neutres n’ont pas beaucoup d’utilité pour la construction d’un modèle. La moitié des phrases du jeu de données n’ont donc pas beaucoup d’utilité. De plus, il a de grande chance pour que quand la phrase à une polarité de 0, il s’agit juste du robot qui a scanné les tweets qui n’as pas réussit à déterminer les sentiments du tweet.
  + On constate de nouveau qu’il y a plus de phrase négative que positive
* Il y a moyenne 13 mots par phrases (plus du double par rapport au premier jeu de données).

Figure - Nombre de phrase par polarité (Tweets financiers)

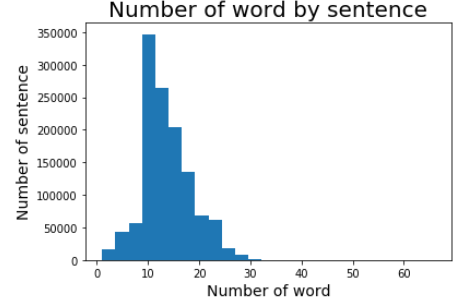


Figure - Nombre de mot par phrase (Tweets financiers)

Sur la figure 8 un histogramme avec sur l’axe X, la polarité d’une phrase et sur l’axe Y la quantité de phrase. Sur la figure 9 un graphe du nombre de mot par phrase. On constate qu’il y a comme indiqué précédemment beaucoup de phrases avec une polarité de 0 dans le jeu de données. On remarque sinon que le sentiment est sois -1 sois 1. Il y a très peu de valeur intermédiaire.

Pour pouvoir utiliser ce jeu de données de manière efficace il faudra donc supprimer une grande quantité de phrase qui ont une polarité de 0. J’ai décider d’en supprimer les 3/4. Comme il y a énormément de donnée, je n’en ai utilisé qu’une petite partie sélectionnée aléatoirement (30'000) pour que le temps d’entrainement des modèles ne soit pas trop long.

## Tweets quelconques

Ce jeu jeu de donné provient de <https://lionbridge.ai/datasets/the-best-25-datasets-for-natural-language-processing/> . Il m’a été fourni à la semaine 7 du projet. Il contenait 160'000 tweets annotés dans n’importe quelle catégorie. C’est-à-dire pas forcements financiers. De plus, ces tweets était annoté à partir des smileys présents dans le texte. Ce qui ne permet pas de définir la polarité de manière précise et sure.

Une liste de différents topics financiers m’a été fourni dans le but d’extraire les tweets contenant au moins un de ces mots. Après extraction, il ne restait que 9'000 tweets. Après analyse des tweets restants, j’ai pu constater que les qualités de ce ses phrases était médiocre. En effet, elle contenait certes des mots financiers mais le contexte avait dans la grande majorité rien à voir avec la finance.

Voici certains tweets obtenus après extraction :

* Oracle buying Sun for $7.4 billion..not a happy day
* I missed out on the $3 off Huggies coupons!
* Another day another dollar... i don't want to work.
* I want a subscription to New Scientist magazine. Too bad it costs $72 a year.
* I have headache =$ I still have to go to school tough.

On constate donc que la plupart des phrases n’ont aucun rapport avec la finance sauf quelques exceptions comme la première phrase (il m’a quand même fallu pas mal de temps pour la trouver dans l’ensemble des tweets). On voit que les phrases trouvées sont loin de titres d’articles financiers. C’est pour cette raison que cette solution a été abandonné.

Le script de prétraitement des tweets peut être trouvé dans le dossier : *dataset/tweets\_datatset/rebuild\_data.ipynb.* Un notebook contenant différentes étapes pour la création de modèles à partir de ce jeu de données se nomme *TweetsMachineLearning.ipynb*.

Ces modèles donnant de mauvais résultats en grande partie du a la qualité des données en entrée, ces modèles ont été abandonnés et ne seront pas détaillé dans ce rapport. Cependant la marche à suivre peut se trouver dans le notebook cité ci-dessus.

## Titres d’articles financiers annotés par des experts

Ces données proviennent de l’entreprise qui a mandaté le projet. Elles ont été annotées à la main par des experts financiers. Ce sont donc ces données qui devront être utilisé comme tests pour valider les différents modèles. Ces données m’ont été fournit à la semaine 14 du projet. Le problème avec ces données est qu’elles n’ont pas été annoté à partir des mots présents dans la phrase mais à partir de la montée du dollar par rapport à l’euro et inversement. C’est-à-dire qu’une phrase peut être rempli de mot négatif mais qu’elle sera annotée positivement parce que dans le contexte de la phrase l’euro augmente.

Voici par exemples 4 phrases du jeu de données de test

* « *berlin - uk foreign secretary hunt says we all want to work with urgency to avoid brexit going wrong* » annoté positivement
* « *snb's zurbruegg says situation on forex market remains fragile* » annoté positivement
* « *offshore chinese yuan vs dollar rises above 6.8 for the first time since july 2017 <cnh=ebs>*» annoté négativement
* « *draghi says rate guidance very effective as reflected by surveys and markets* » annoté avec une polarité neutre

On remarque par exemple dans la première phrase que les mots « urgency », « avoid » et « wrong » ainsi que la phrase ont un contexte négatif pourtant la phrase est annoté positivement.

Pour la seconde phrase le contexte de la phrase et le mot « fragile » pourrait présager que la phrase est négative pourtant elle est annotée positivement.

Le contexte de la troisième phrase ainsi que le mot « rises » ont un contexte positif mais la phrase est annotée de manière négative.

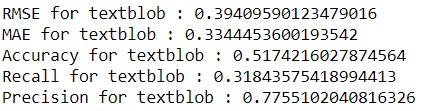
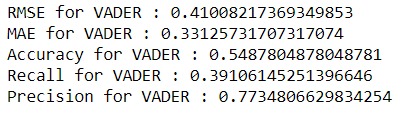
Pour la dernière phrase, on constate que le contexte de la phrase ainsi que le mot « very » et « effective » ont un contexte positif mais la phrase est annotée avec une polarité neutre.

On remarque donc que ce jeu de données est annoté de manière différente du premier jeu de données qui m’as été fournis.

# Analyseur de sentiment présent sur le web

Pour le projet, il m’a été demandé de comparer les scores d’analyseur de sentiment présent sur le web. Les scores obtenus ici seront utilisés comme score de référence pour l'évaluation des futurs modèles. Le but des futurs modèles est de battre ces scores. Sur la base des recherches effectués, deux modèles d’analyse de sentiment ont été retenu : la librairie Textblob et VADER un sous package de la libraire NLTK. J’ai calculé ces scores avec le jeu de données des titres d’articles financiers.

Voici les scores qu’obtiennent ces deux analyseurs :

* Pour textblob :
* Pour VADER :

On constate que les résultats obtenus sont plutôt mauvais. Même si la RMSE et la MAE ne sont pas si mauvaise, la précision est quant à elle très mauvaises. Les 2 modèles obtiennent une précision d’a peine plus de 50%, ce qui corresponds à un choix presque aléatoire de la classe. J’ai également constaté que ces 2 analyseurs ont souvent tendances à prédire un sentiment de 0 pour les phrases.

Au vu des scores, on constate qu’il est très intéressant de construire un modèle de machine learning avec pour but de battre les scores ci-dessus.

# Solutions développées

Voici les différentes solutions développées ainsi que l’argumentation sur le choix de ces solutions sur la base des recherches effectué en amont sur le domaine.

## Prétraitement des données

Ici, le but est de prétraités les données dans le but de pouvoir les introduire dans un algorithme de machine Learning ou de deep learning.

### Traitement du texte

#### Spacy

Voici les différents processus qui ont été effectué pour prétraiter la phrase :

* **Tokenisation** : Cette étape consiste à découper la phrase en mot.
* **Suppression des stop word** : Cette étape consiste à supprimer des mots qui ont peu d’importance dans la phrase comme « the », « are », etc.
* **Tagging et suppression de certains mots tagués** : Le tagging consiste à détecter à quelles classes appartient un mot comme par exemple nom, adjectif, verbe, etc. Par la suite, il faut supprimer le type de mot que l'on ne veut pas garder dans la phrase. Ceci peut être utile pour enlever par exemple les déterminants.
* **Suppression des mots en dessous d’une certaine taille** : Ce processus consiste à ne garder que les mots d'une certaine taille dans la phrase.
* **Suppression de la ponctuation** : Cette étape consiste à supprimer les caractère comme !, ?, etc.
* **Lemmatisation** : Ce processus consiste à ne garder que la racine des mots. Par exemple, les mots comme « runs », « running » seront transformer en « run ». Ceci permet de réduire le nombre de mot dans le dictionnaire final.

Il est a noté que les différents processus ci-dessus ne sont pas tous effectué systématiquement lors du traitement de la phrase. Le but est de regarder quels processus fait améliorer ou non le score du modèle.

Le traitement du texte est effectué avec la librairie Spacy[[3]](#footnote-3). Pour effectuer ceci, j’ai créé une classe nommée « TextProcessor » héritant de « BaseEstimator » et de « TransformerMixin ». Le fait d'hériter de ces 2 classes permettra d'inclure cette classe dans le pipeline sklearn. Les différents processus vus ci-dessus peuvent être ajouté ou non au traitement via le constructeur de la classe. Le principale avantage de cette méthode est qu'elle se combine avec l'utilisation de la classe GridSearchCV() ou RandomizedSearchCV() qui permet de chercher les meilleurs hyper-parametrès pour un modèle.

#### Gensim

La bibliothèque python Gensim possède une fonction permettant d’effectuer le prétraitement d’une phrase[[4]](#footnote-4). Cette fonction met les mots en minuscule, les tokenize, enlève les accents et par défaut enlève les mots plus petits que 2 caractères et plus grand que 15 caractères. Cette méthode est moins complète que celle-ci-dessus mais à l’avantage d’être extrêmement rapide.

## Représentation d’une phrase

### TF-Idf

Comme un modèle de machine Learning fonctionne avec des nombres et non du texte, il faut trouver un moyen de représenter les phrases sous formes de nombres. Le TF-idf permet d’effectuer cette opération en représentant un mot sous forme de nombre.

Le TF-idf se compose de 2 parties, le TF (Term Frequency) qui est le nombre de fois qu’apparait le mot dans la phrase. Cette fréquence peut être normalisé à l’aide d’un logarithme.

Où f\_t,d représente la fréquence du mot dans la pharse.

L’idf (Inverse Document Frequency) est le nombre total de document dans le corpus divisé par le nombre de document où le terme apparait.

Où D est le nombre total de document dans le corpus et dj le nombre de document où le terme apparait.

Pour finir le score TF et idf est multiplié. Ce score est calculé pour chaque phrase et représenté dans une matrice sparse[[5]](#footnote-5) de scipy. C’est cette matrice qui est passé au différent modèle.

Cette technique à été utilisé dans mon projet pour les modèles de machines learning.

### Word2vec

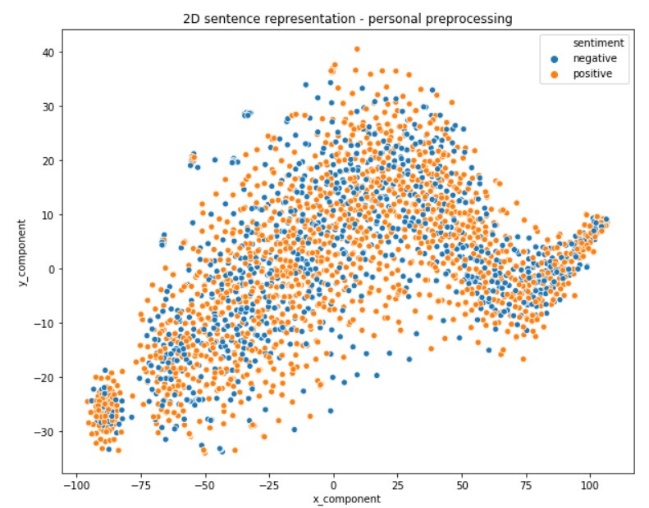
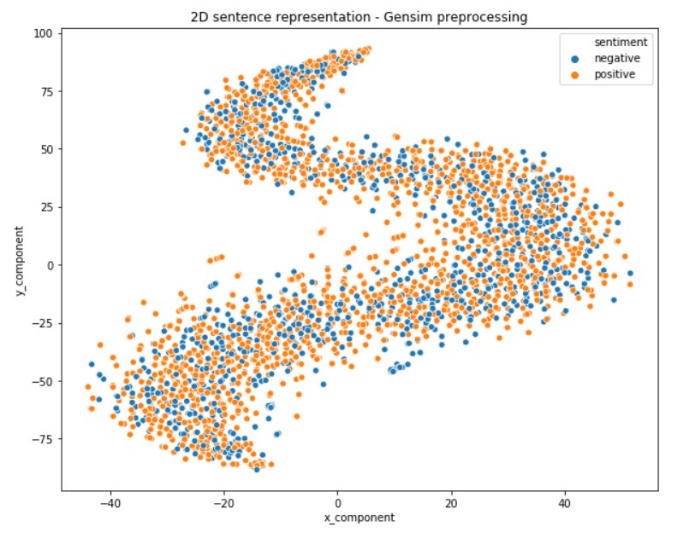
Trois modèles de Word2vec ont été utilisé pour représenter une phrase dans l’espace. En premier le word2vec de la librairie Gensim[[6]](#footnote-6). En deuxième, un word2vec financier qui m’as été fourni par un assistant de l’équipe de recherche. En troisième, le word2vec de Glove[[7]](#footnote-7).

#### Gensim

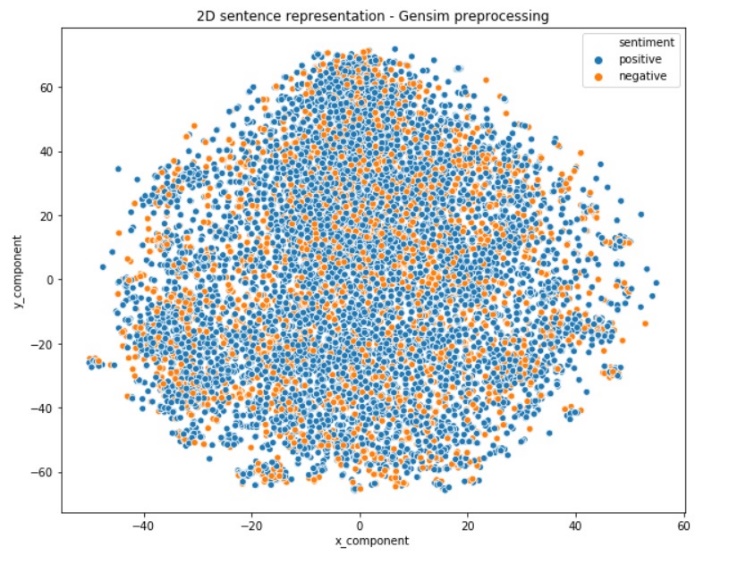
La méthode utilisée via Gensim pour représenter la phrase dans l’espace n’est pas un word2vec mais un doc2vec[[8]](#footnote-8). Un doc2vec permet de représenter une phrase entière dans l’espace contrairement au word2vec qui permet de représenter seulement un mot. La classe doc2vec utilisé est un wrapper sklearn pour Gensim. Ce qui signifie que le doc2vec peut donc être directement utilisé avec sklearn et par exemple être placé dans un pipeline.

Le doc2vec représente une phrase avec 100 composantes. Pour envoyer les phrases représentées dans l’espace au modèle de machine Learning avec sklearn, il suffit de le positionner dans un pipeline.

Voici une représentation T-sne[[9]](#footnote-9) du jeu des données des HeadLines financière. Une représentation T-sne permet de réduire le nombre de dimension d’un vecteur donné. Il est utile pour visualiser les données en 2 dimensions. Comme les vecteurs ont ici deux dimensions, il peut être utile de les représenter en 2 dimensions pour par exemple voir si les phrases sont bien réparti dans l’espace, si les phrases positives sont proches des positives et inversement, etc …



On peut voir le jeu de données des headlines financières représenté dans l’espace. À gauche, avec le prétraitement effectué avec Gensim et à droite avec le prétraitement créé par moi-même. On peut constater que dans les 2 cas les phrases positives ne sont pas proches des phrases positives et inversement. Avec Gensim, on peut voir que les phrases sont moins éparpillées dans l’espace qu’avec l’autre méthode, même si on peut voir un amas de phrase dans le coin en bas à gauche de la figure gauche. Le fait que les phrases soit bien dispersé dans l’espace peut montrer que le modèle n’aura pas tendances à surévaluer mais se généralisera bien.



On peut voir ici une dernière représentation T-sne mais cette fois si avec le modèle des tweets financiers. Seul 10'000 phrases ont été représenté sur 1'000'000 (la fonction T-sne demande beaucoup de temps de calcul). On peut constater que les données sont plutôt bien réparties en vue de leur grand nombre.

Cette technique de représentation des phrases à été utilisé dans mon projet pour des modèles de machines learning.

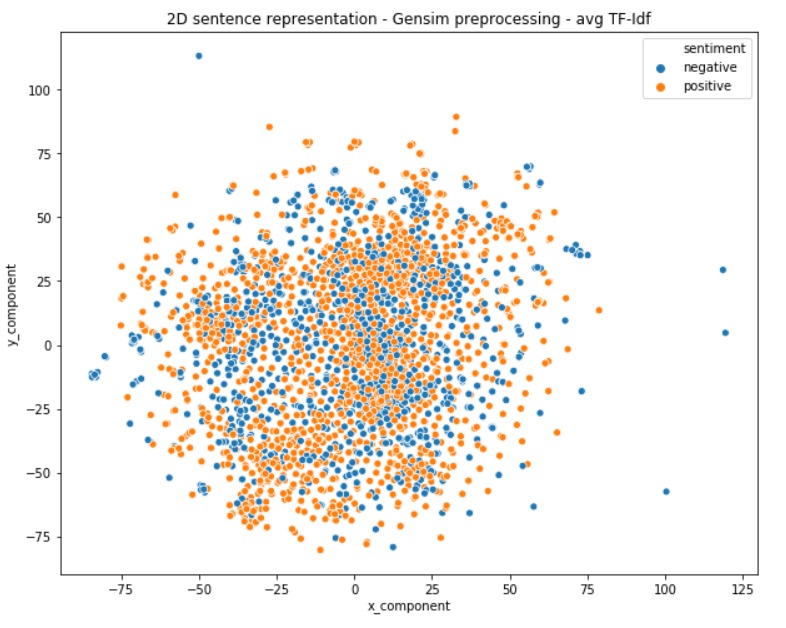
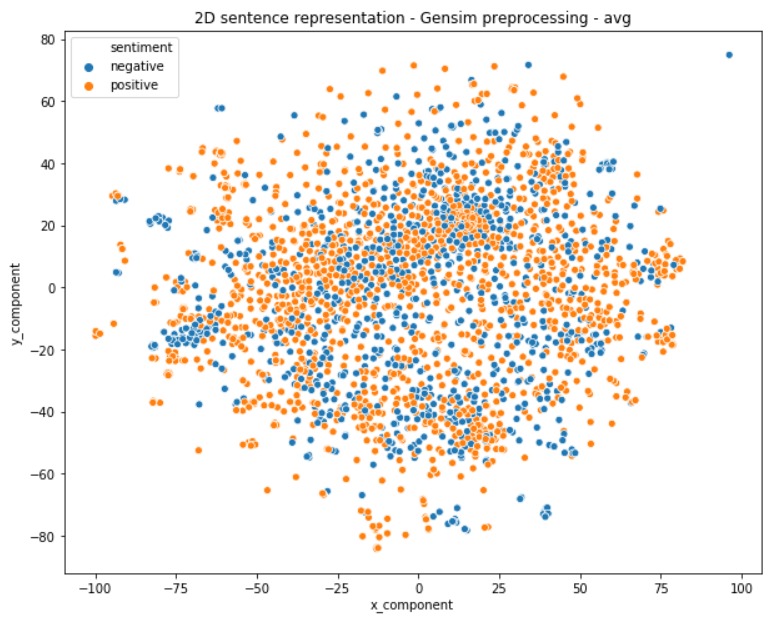
#### Word2vec financiers

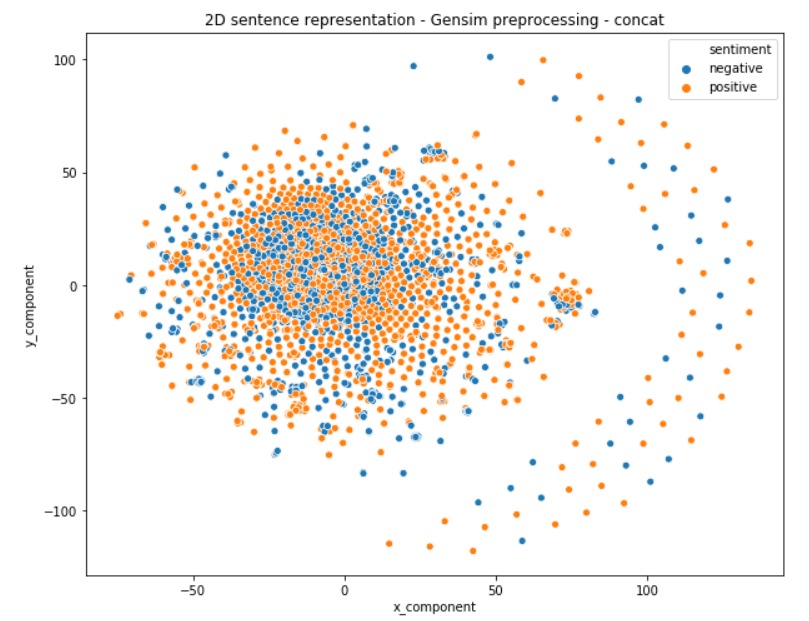
Le word2vec financiers fourni par un assistant de l’équipe de recherche à été entrainé sur un million de mot financiers. Un mot est représenté avec 100 composantes. Comme le word2vec fournit la représentation d’un mot et non d’une phrase dans l’espace, il faut utiliser un système de padding (expliqué plus bas) avec la phrase la plus longue du jeu de données. En effet, il faudra donner des vecteurs de même taille au modèle sklearn. De plus, les mots non présents dans le jeu de donné seront représenté par un vecteur contenant des 0.

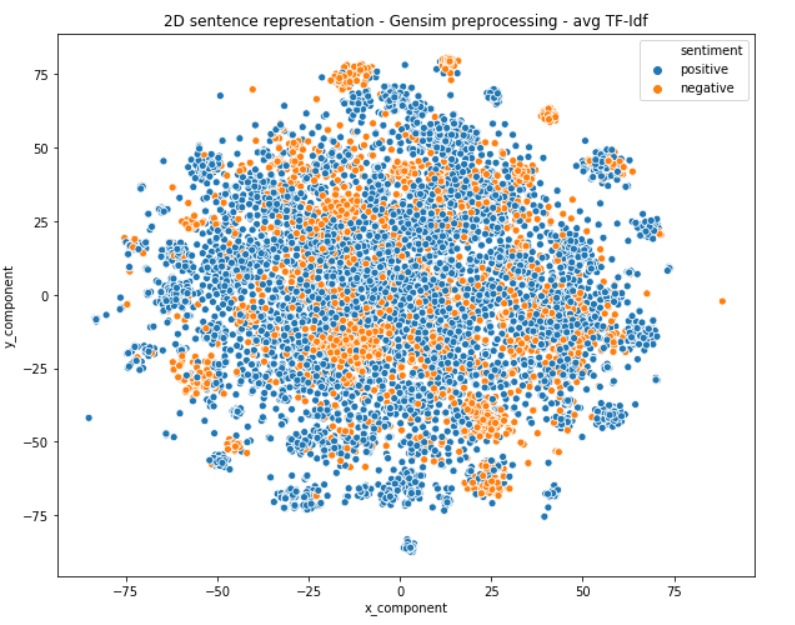
Différentes méthodes de représentation dans la phrase ont été développé :

* Concaténation des vecteurs : Les vecteurs représentants les mots sont concaténé les uns à la suite des autres. Si la phrase la plus longue du jeu de données à une taille de 25, et que la phrase à représenter à une taille de 14, il faudra concaténer 11 vecteurs rempli de 0 à la fin du vecteur (c’est le principe du padding). Si les vecteurs représentant un mot ont 100 dimensions, la taille finale d’un vecteur pour une phrase sera de 25'000 composantes.
* Concaténation des vecteurs multiplié par le score TF-idf du mot : C’est le même principe que la méthode précédente sauf qu’avant de concaténer le mot, on va multiplier chaque composante du vecteur par son score TF-idf. Ceci permet également de prendre en compte la fréquence du mot dans le corpus et dans un document.
* Somme des vecteurs : Les composantes des vecteurs sont sommées avec le vecteur précédent. Peu importe la taille de la phrase, le vecteur aura toujours 100 composantes.
* Somme des vecteurs multiplié par le score Tf-idf : C’est le même principe que l’étape précédent sauf qu’avant de sommer les composantes, on multiplie par le score TF-Idf du mot.
* Moyenne des vecteurs : Les composantes des vecteurs sont sommes et divisé par la longueur de la phrase.
* Moyenne des vecteurs multiplié par le score TF-Idf : C’est le même principe que l’étape précédent sauf qu’avant de sommer les composantes, on multiplie par le score TF-Idf du mot. Les composantes sont également divisées par la taille de la phrase.

Une fois que tous les vecteurs représentant une phrase font la même taille, ils peuvent être envoyé au modèle de machine learning.







On retrouve ci-dessus 4 représentations T-sne. Les 3 graphes avec les moins de points sont les vecteurs appartenant au jeu de données des Headlines financières. Le graphe en bas à droite appartient aux tweets financiers. Différentes techniques ont été utilisé ici notamment :

* La moyenne : On constate que la moyenne est la représentation qui disperse le mieux les données. On peut de plus voir que les phrases positives ont plus tendances à se trouver proche des phrases positives et inversement.
* La moyenne multipliée par le TF-Idf : On peut voir que pour les deux jeux de données cette méthode à tendances plus faire des paquets de phrases. La encore une fois les phrases positives ont tendances à être positionné vers les positives et inversement.
* La concaténation : On constate que cette méthode regroupe beaucoup les données et forme des pattern bizarre dans le bord droit.

Il est bien sur possible de créer plein d’autre exemple de représentation T-sne. Notamment, en modifiant le jeu de données utilisé, la méthode de prétraitement du texte et la représentation en vecteurs. Ci-dessus, ne sont affichés que 4 exemples.

Cette technique a été utilisé dans mon projet pour des modèles de machines learning et de deep learning.

#### Glove

Glove est un algorithme de machine learning non supervisé permettant de représenter les phrases dans l’espace. La distance entre les mots est liée à leur similitude sémantique. Cet algorithme est Open-source er a été développé par l’université de Stanford[[10]](#footnote-10).

Sur le site de Glove[[11]](#footnote-11), il est possible de récupérer un fichier ou un vecteur représentant un mot à été précalculé à l’avance. Ces vecteurs ont été précalculé sur une très grande quantité de donné. Le fichier que j’ai choisi représente un mot avec un vecteur à 100 composantes. Le fichier se présente sous la forme d’un fichier csv ou la première colonne indique le mot et les 100 suivantes indiquent les composants du vecteur.

Pour représenter un mot dans l’espace, il suffit alors de regarder si ce mot est présent dans le dictionnaire. Si le mot est présent on récupère le vecteur sinon on lui attribut un vecteur avec 100 composantes à 0.

Cette méthode a été utilisé dans mon projet pour les modèles de deep learning.

### Padding

Pour la technique du padding, il suffit d’attribuer à chacun des mots du jeu de données un indice. Chaque mot est alors représenté dans une phrase par son indice. Comme il faut que toutes les données envoyées au modèle soient de la même taille, il faut calculer la taille de la plus longue des phrases du jeu de données. Toutes les phrases envoyées au modèle devront faire cette taille. Les mots manquants sont comblés pas l’indice 0.

Par exemple, si la phrase : [Ceci(34) est(2) une(4) phrase(7)] où les chiffres en parenthèse représente l’indice du mot dans le jeu de données et que la taille de la plus grande phrase du jeu de données est de 8, cette phrase deviendra alors : [34, 2, 4, 7, 0, 0, 0, 0]. C’est sous cette forme que la phrase sera envoyée au modèle.

Cette technique à été utilisé dans mon projet pour les modèles de deep learning.

## Modèle de machine learning

Une multitude de modèle de machine Learning ont été testé pour ce projet. En effet, dans sklearn, le changement d’un modèle pour un autre est facile.

### Modèle de régression

* La régression linéaire (LinearRegression)
* La machine à vecteur de support (SVR)
* La machine à vecteur de support linéaire (LinearSVR)
* La forêt aléatoire (RandomForestRegressor)

La régression linéaire a été choisi car c’est un modèle de machine learning basique pour les tâches de régression. La régression linéaire est surtout utilisée pour s’assurer que les autres modèles ont de meilleurs résultats qu’un modèle basique. Les 2 autres modèles ont été sélectionné car ce sont 2 modèles puissant pour les tâches de régression. La machine à vecteur de support linéaire est utilisée dans le cadre de jeu de données de grande taille comme par exemple celui des tweets financiers. En effet le SVR classique à une complexité algorithmique qui croit fortement pour plus de 10'000 observations[[12]](#footnote-12).

Le modèle de machine learning est ajouté à la suite du pipeline. Les différents modèles sont alors entrainés sur le jeu d’entrainement.

### Modèle de classfication

* La classification naïve bayésienne (MultinominalNB)
* La régression logistique (LogisticRegression)
* La machine à vecteur de support (SVC)
* La machine à vecteur de support linéaire (LinearSVC)
* La forêt d’arbre de régression (DecisonTreeClassifier)

La classification naïve bayésienne et la régression logistique sont comme pour les tâches de régression des modèles basiques qui permet de s’assure que les autres modèles apprennent bien à partir des données. La machine à vecteur de support et la forêt d’arbre de décision sont comme pour les tâches de régression, de puissant modèle. Le LinearSVC permet une nouvel fois comme pour la régression de pouvoir traiter des jeux de données de grande taille.

Il est également possible avec des modèles de régression d’obtenir la précision des données précites comme nous le donnerai un classificateur. Pour effectuer cette étape, les prédictions du modèle devront être placé dans la classe positive ou négative selon un certain seuil. Cette manière de faire peut donner de mauvais résultat si beaucoup de prédiction ont une polarité proche de 0.

## Modèle de deep learning

Voici les différents types de réseau de neurones que j’ai testé pour mon projet. Les architectures sont détaillé dans la section qui parle des résultats de ces modèles.

* LSTM
* Recurrent Neural Network
* GRU
* Convolutional Neural Network

## Évaluation des modèles de machines learning

Les scores dont je parle dans les chapitres suivants peuvent changer de quelques pourcents entre l’exécution du notebook et le rapport. Ceci vient du fait qu’il y a tellement de petits paramètres pour un modèle de machine learning qu’un petit changement peut engendrer un score légèrement différent. Cependant l’ordre d’erreur reste la même.

Les captures suivantes montrent à chaque fois le modèle ayant le meilleur score pour cette étape.

### Titres d’articles financiers

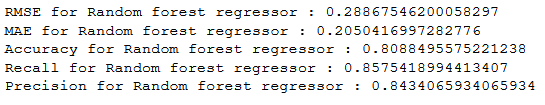
Pour chacune des étapes, j’ai prétraité les phrases à l’aide de Spacy. J’ai en effet remarqué que le modèle donnait de meilleur score qu’avec le prétraitement de Gensim. Les étapes de prétraitement effectué sont :

* Tokenisation
* Suppression des « stop-word »
* Suppression de la ponctuation
* Lemmatisation du mot

Voici les différents scores que j’obtiens avec ce jeu de données. Pour chacune des étapes c’est le modèle avec le meilleur score qui a été conservé.

#### Modèle avec le Tf-Idf

Voici le score du modèle utilisant le Tf-Idf et le prétraitement effectu avec Spacy. On voit que par rapport à VADER et Textblob, le résultat est bien meilleur. On remarque que l’on atteint une exactitude de 80%. Le modèle qui obtient les meilleurs résultats est une forêt aléatoire.

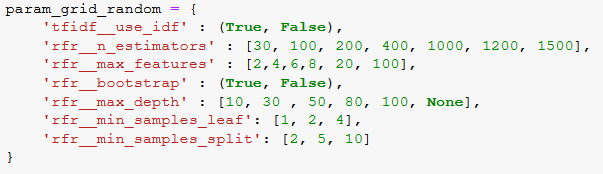


##### Recherche par quadrillage

Pour prouver que le modèle ne surévalue pas trop et pour optimiser les hyperparamètres j’ai entrainé le modèle avec une recherche aléatoire (RandomizedSearchCV dans sklearn). Cette fois si le modèle est entrainé en 15 blocs (fold) différents. Le jeu d’entrainement est découpé aléatoirement en 15 blocs distincts, puis l’entrainement et l’évaluation s’effectue en 15 passes successives. Un bloc est à chaque phase gardé pour l’évaluation et les 14 autres pour l’entrainement[[13]](#footnote-13). Ce processus permet de s’assurer que le modèle ne surévalue pas trop.

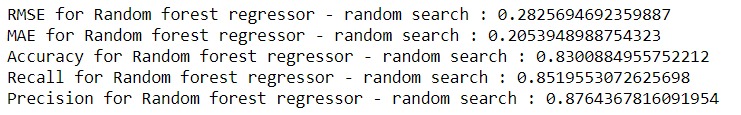
Cette étape est alors réalisée plusieurs fois en changeant les hyperparamètres du modèle (pour ce modèle j’ai choisi 50 itérations). J’ai choisi d’effectuer la recherche aléatoire avec la forêt aléatoire car c’est ce modèle qui donnait les meilleurs résultats à l’étape précédente.

Voici les hyperparamètres utilisés :

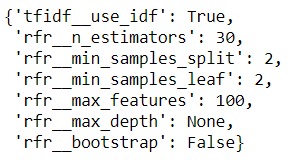
La clé du dictionnaire correspond au hyperparamètres à tester et la valeur du dictionnaire correspond aux valeurs à tester pour le modèle.

La recherche aléatoire va entrainer le modèle avec des paramètres aléatoire pour chaque itération contrairement à la recherche par quadrillage qui va tester toutes les combinaisons possible d’hyperparamètres.

Après avoir entrainé le modèle avec la recherche aléatoire, on constate que les résultats se sont améliorés. Le modèle atteint une exactitude de 83 %. De plus comme le modèle à été entrainé par validation croisé, cela nous assure qu’il ne surévalue pas.

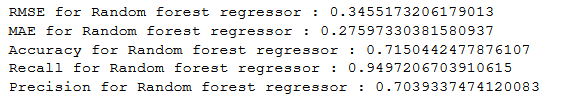


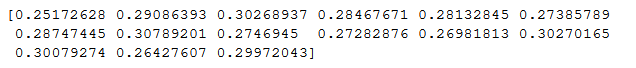
Voici les différents hyperparamètres de ce modèle :



#### Modèle avec Doc2vec de Gensim

Pour ce modèle, j’ai utilisé la classe Doc2Vec de Gensim pour représenter mes données dans l’espace. Le prétraitement du texte a été effectué avec Spacy. Le modèle qui obtient les meilleurs résultats est de nouveau la forêt aléatoire. Cependant, on voit cette fois ci que le modèle obtient de moins bons résultats que précédemment. L’exactitude est moins bonne d’environ 10% et le compromis précison/rappel est moins bon. De plus la RMSE et la MAE ont augmentée.

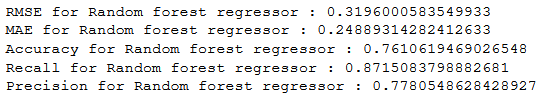


J’ai également entrainé la forêt aléatoire toujours avec les mêmes données en utilisant une validation croisée en 15 phases pour prouver que le modèle ne surévalue pas. 

Voici les scores de la validation croisée. Chaque élément de la liste correspond au score d’une des phases de la validation croisée. On constate que la MAE au même ordre de grandeur que celle au point précédent. Cela prouve que le modèle ne surévalue pas. De plus, le graphe TSNE présenté lors de l’étape « prétraitement avec Gensim » montre que les données sont assez dispersées dans l’espace. Ceci est une autre preuve que le modèle ne surévalue pas.

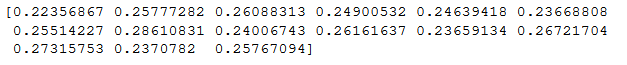
#### Modèle avec le Word2Vec financier

Pour ce modèle j’ai utilisé le prétraitement du texte de Spacy. Pour la représentation de la phrase dans l’espace j’ai utilisé la moyenne des vecteurs de mots. J’ai utilisé cette représentation, car c’est celle qui disperse le mieux les données dans l’espace tout en gardant un bon score par rapport à l’autre représentation spatiale présenté au chapitre « Word2Vec financiers ». Le meilleur modèle est encore une fois une forêt aléatoire.



Les autres modèles comme le SVC donne aussi de bon résultat mais à peine moins bon que la forêt aléatoire. C’est pour cette raison que je garde uniquement la forêt aléatoire. Pour ce cas, par exemple, le SVC donnais une précision de 75,9% contre 76,1 pour la forêt aléatoire.

Après avoir entrainé ce modèle avec la validation croisé de 15 phases j’obtiens les résultats suivants :



On constate que la MAE n’a pas augmenté par rapport au résultat précédent. Cela indique que le modèle ne surévalue pas trop. Cependant, on peut constate sur le graphe T-sne que les données sont un peu moins dispersées dans l’espace qu’avec Gensim.

##### Modèles avec différentes représentation vectorielle

Voici les résultats obtenus si l’on entraine une forêt aléatoire mais en utilisant une autre représentation dans l’espace des données :

* Pour la concaténation des vecteurs :
  + Exactitude : 75,4 %
  + MAE : 0.25
* Pour la somme multipliée par le score Tf-Idf :
  + Exactitude : 77,1 %
  + MAE : 0.24

On constate que ces 2 modèles ont des scores semblables à celui obtenu avec la moyenne des vecteurs. Cependant ces 2 modèles ont tendances à représenter les phrases plus proches l’une de l’autre dans l’espace. Cela augmente alors la chance du modèle de surévaluer. De plus, pour la concaténation de vecteur le vecteur en entrée contient dans mon cas 2500 composantes. Plus il y a de composantes plus le modèle est complexe et a de chance de surévaluer.

C’est pour ces différentes raison que mon choix c’est plutôt porter sur la moyenne des vecteurs de mot.

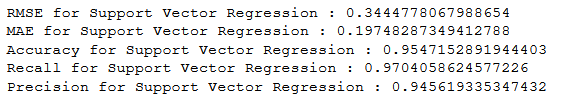
### Tweets financiers

Voici les différents scores que j’obtiens avec ce jeu de données. Pour chacune des étapes c’est le modèle avec le meilleur score qui a été conservé.

Étant donné, le grand nombre de donnée à ma disposition plus de …, il m’a fallu réduire la taille du jeu de donnée. Les scores suivants ont été obtenus avec 30'000 données. Pour augmenter légèrement les scores et diminuer le risque que le modèle surévalue il est possible d’augmenter le nombre de données utilisées. Cependant le temps d’exécution des cellules prendra beaucoup plus de temps.

#### Modèles avec le Tf-Idf

Pour ces modèles j’ai utilisé le prétraitement de Gensim et la représentation Tf-Idf pour les phrases. Le modèle présenté ci-dessous est un SVR. On constate que le modèle à de très bon résultats.



Le modèle obtient une exactitude de 95% et un bon taux rappel/précision. La MAE et la RMSE ont également de bons résultats.

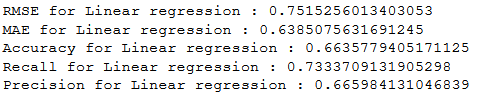
Si l’on entraine le modèle avec la validation croisée, on remarque que la MAE est semblable au point précédent. Le modèle n’a donc pas l’air de surajuster. De plus comme le jeu de données contient pas mal de donnée, les chances de surajuster diminue.



Pour conclure, on peut dire que ce modèle semble intéressant.

#### Modèle avec Doc2vec de Gensim

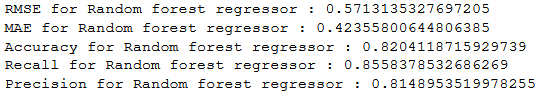
Les phrases de ce modèle ont été prétraité avec Gensim et elles sont représenté dans l’espace à l’aide de la classe Doc2Vec de Gensim. Le meilleur modèle pour cette étape est une régression linéaire.



On constate avec ce modèle que les performances du modèle se sont considérablement dégradés. Quand on regarde la RMSE et la MAE on constate que les prédictions son très éloignés de la réalité. Le modèle à de la peine à généralisé. Les autres modèles développés pour cette étape ont aussi de mauvaises performances.

#### Modèle avec le Word2Vec financier

Les phrases de ce modèle ont été prétraité avec Gensim et la représentation dans l’espace s’effectue en faisant la moyenne des vecteurs de mot. J’ai choisi cette représentation dans l’espace pour les mêmes raisons que celle évoqué avec le jeu de données des titres d’articles financiers. Voici les résultats obtenus :



On constate que le modèle à une bon exactitude, rappel et précisons. Mais, la RMSE et MAE sont quand même élevé. Cela veut dire que le modèle fait pas mal d’erreur mais arrive quand même à trouver la bonne classe pour les phrases.

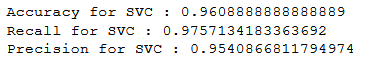
Si on entraine le modèle avec une validation croisée de 10 phases, on constate que la MAE est semblable au point précédent donc le modèle n’a pas l’air de surajuster.



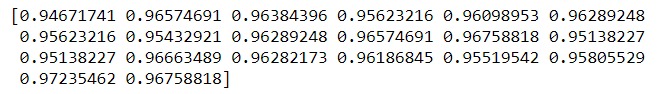
Cependant quand on regarde le graphe T-sne on constate que les phrases sont assez proches dans l’espace et se regroupe en petit tas. Cela est surement du au fait que beaucoup de phrases sont représenté sur le graphe.

#### Classificateur avec Tf-Idf

Pour cette étape. J’ai enlevé du jeu de données les phrases qui avait un sentiment entre -0.3 et 0.3. J’ai également converti le sentiment de la phrase en des valeurs discrètes 0 ou 1. Pour cette étape les phrases ont été prétraité avec Gensim. Le meilleur modèle est ici un SVC. Les autres modèles testés ont également de très bon résultat.



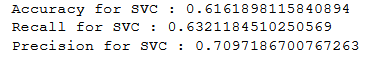
On constate que l’exactitude atteint les 96% et que le compromis précision rappel est très bon. Si l’on entraine le modèle avec une validation croisée en 20 phases on constate que le modèle n’a pas l’air de surajuster. L’exactitude ne change pas par rapport à avant.



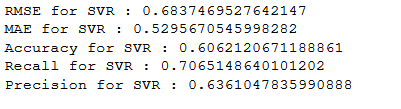
#### Calcul du score sur les titres d’articles financiers

Ce modèle a été entrainé uniquement avec des tweets. Cependant ce qui nous intéresse dans ce projet ce sont les titres d’articles financiers. Il faut donc tester les modèles précédemment crée sur le jeu de données financiers. J’ai alors utilisé tous les titres d’articles financiers que j’ai à ma disposition. J’ai alors sélectionné les modèles suivants pour effectuer des prédictions sur les titres d’articles :

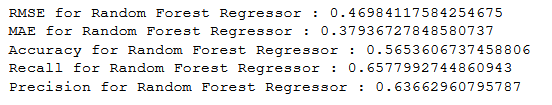
* SVC avec prétraitement Gensim et Tf-Idf



* SVR avec prétraitement Gensim et Tf-Idf



* Foret aléatoire avec prétraitement Gensim et Moyenne des vecteurs de mot



On constate dans les trois cas que les performances se sont dégradé considérablement. La précision est à peine en dessus de 50%. Le compromis rappel et précision et bas. La MAE et RMSE sont également élevé.

On remarque donc que les modèles entrainés avec les tweets sont très bons sur les tweets mais mauvais sur les titres d’articles financiers. On peut donc dire que les modèles surajuste considérablement.

Cette solution n’est donc pas bonne et est à abandonner.

### Tweets quelconques

Étant donné que les tweets financier extrait de l’ensemble des tweets à ma disposition n’est pas de qualité je ne développe pas les résultats obtenus dans le rapport. J’ai tout de même créé un notebook jupyter du nom de « TweetsMachineLearningModel.ipynb » qui retrace les différentes des étapes pour la création de modèle avec ce jeu de données. Les étapes du notebook et les modèles développés sont semblables aux étapes expliqué précédemment pour les autres jeux de données. Les résultats obtenus ne sont pas et ne dépasse pas les 65 %. De plus comme les tweets sont très différents des titres d’articles financiers, le modèle aurait donner encore de plus mauvais résultat pour des prédiction à partir de tweet financier.

## Évaluation des modèles de deep learning

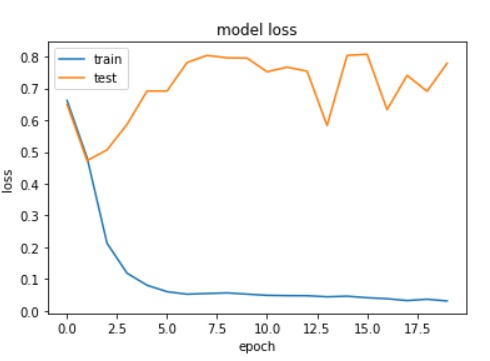
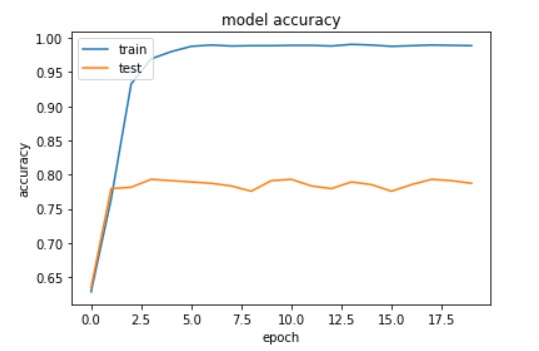
Les modèles de deep learning ont été développé à partir des titres d’articles financiers.

### LSTM avec padding

Pour ce modèle j’ai représenté les phrases à l’aide de la technique du padding.

Ce modèle contient comme première couche, une couche Embedding qui prends en paramètre une dimension d’entrée qui corresponds à la taille du vocabulaire et la dimension de sortie que j’ai fixé à 32. La seconde couche est une couche LSTM[[14]](#footnote-14) de taille 32. La dernière couche du modèle est une couche de type Dense avec une sortie de taille 1 et une fonction d’activation sigmoid.

J’ai entrainé ce modèle sur 20 epoch avec une taille d’échantillon de 64. Voici les résultats de ce modèle avec un graphe de l’exactitude en fonction des epochs et un graphe de la perte en fonction des epochs.



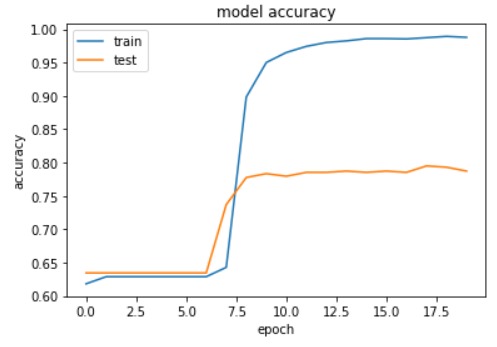
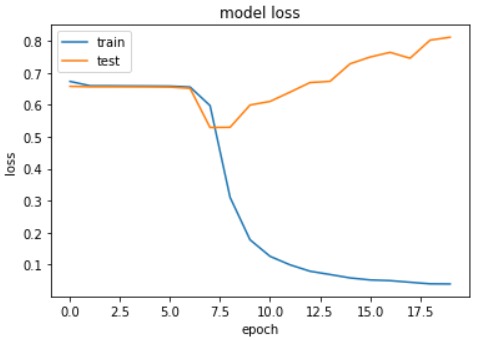
On constate que les performances à l’entrainement sont excellentes mais dès qu’on passe sur le jeu de tests les performances deviennent médiocre. En regardant le graphe de droite, on constate que la perte ne diminue pas voir pire qu’elle augmente. Le modèle n’arrive pas à apprendre à partir du jeu d’entrainement. Autrement dit, le modèle surajuste beaucoup. Même en diminuant la complexité du modèle en réduisant la dimension de sortie de la couche Emedding et la dimension de la couche LSTM le modèle surajuste toujours autant et les performances sont toujours médiocres.

Il y a trop peu de données pour entrainer un réseau de neurone et les phrases à ma disposition sont trop courte. Ce type de réseau de neurone ne donne donc pas de bonne performance.

### GRU avec padding

Ce modèle utilise également la technique du padding pour représenter une phrase. La configuration de ce modèle est la même que celle du précédent à l’exception que la dimension de sortie de la couche Embedding de 16, la couche LSTM est remplacé par une couche GRU de taille 16.

J’ai de nouveau entrainé ce modèle avec 20 epoch et une taille d’échantillon de 64. Voici les résultats et les graphes d’exactitude et de perte.



On constate que comme le modèle précédent, ce modèle surajuste beaucoup. La perte augmente au lieu de diminuer et l’exactitude n’augmente.

Comme pour le cas d’avant il faudrait plus de phrases à disposition.

### CNN avec Word2Vec de Glove

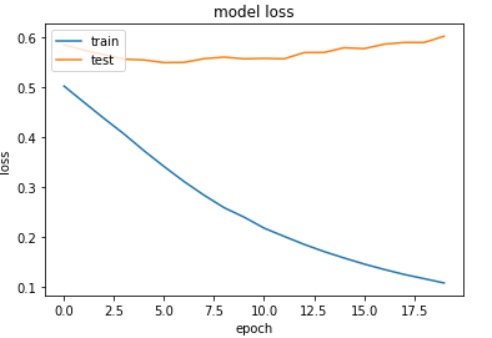
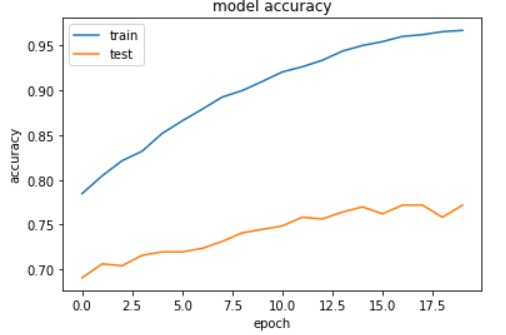
Pour ce modèle j’ai utilisé le Word2Vec fournis par Glove. J’ai alors créé un dictionnaire de la taille du jeu de données. La clé de ce dictionnaire est l’index du mot dans le vocabulaire et la valeur est le vecteur correspondant au mot. Ce vecteur provient du fichier fournit par Glove. Si un mot ne se trouve pas dans fichier il est remplacé par un vecteur avec toutes ses composantes à 0.

La taille finale du dictionnaire est la taille totale du vocabulaire et j’ai fixer la taille des vecteurs à 32.

Les phrases sont comme pour les 2 étapes précédentes représenté à l’aide de la technique du padding.

L’architecture du modèle contient une première couche de type Embedding. Je dois donner en paramètre à cette couche le dictionnaire de vecteur créé précédemment. C’est ce qui permettra au modèle de lier les indices des mots à leurs vecteurs. La seconde couche est de type GlobalMaxPool1d. La suivante est une couche de type Dense avec une dimension 10 de sortie et une fonction d’activation relu. La dernière couche est une couche Dense avec une dimension de sortie de 1 et une fonction d’activation Sigmoid.

J’ai entrainé ce modèle sur 20 epochs avec une taille d’échantillon de 64. Voici les résultats et graphes de ce modèles.



On constate encore une fois que le modèle surajute beaucoup trop et que la perte diminue au lieu d’augmenter. Le modèle n’arrive rien à apprendre du jeu d’entrainement.

### RNN avec Word2vec financier

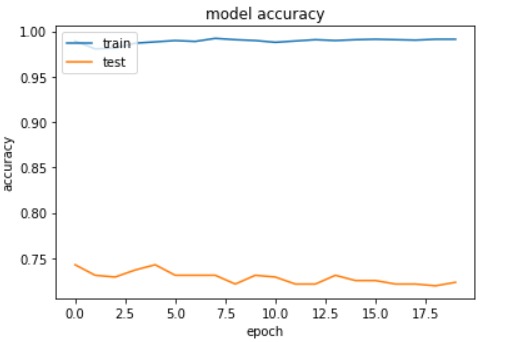
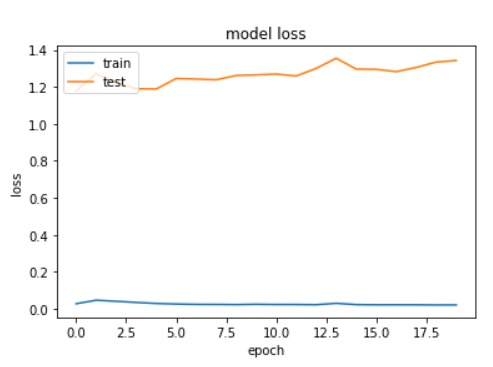
Pour ce modèle j’ai représenté comme pour les précédents les phrases avec la technique du padding.

J’ai ensuite utilisé le Word2Vec financiers entrainé sur plusieurs millions de mots dans le domaine de la finance.

J’ai alors créé une matrice de la taille totale du vocabulaire multiplié par la taille des vecteurs que j’ai fixé à 100. Le numéro de la colonne de la matrice représente l’index du mot dans le vocabulaire. Une ligne de la matrice représente les composantes du vecteur pour le mot donné. Si le mot n’existe pas dans le Word2Vec il est remplacé par un vecteur avec toutes ses composantes à 0.

L’architecture de ce modèle est composée d’une première couche Embedding qui à une taille d’entrée de la dimension du vocabulaire et une taille de sortie de la taille d’un vecteur représentant un mot c’est-à-dire 100. Il faut également passer en paramètre à cette couche la matrice créée précédemment. La seconde couche est une couche SimpleRNN de taille 16. La dernière couche est une couche Dense avec une taille de sortie de 1 et une fonction d’activation Sigmoid.

Voici les résultats et les graphes obtenus avec ce modèle.



Calcul de sentiment sur de nouveau texte

On constate que le fait que les vecteurs de mot ont 100 composantes fait que le modèle surajuste énormément. La perte est très élevée et l’exactitude de monte pas.

Ce modèle comme tous les autres modèles de deep learning présenté précédemment ne sont pas utilisable cat ils surajustent beaucoup trop. Les réseaux de neurone ont besoin de beaucoup de donnée pour donner de bon résultat. Le jeu de données mis à ma disposition ne contient clairement pas assez de phrase pour qu’un réseau de neurone donne des résultats satisfaisants.

## Conclusion des solutions développés

Pour ce qui est des solutions de deep learning on constate que les résultats ne sont vraiment pas bons. Les performances sur le jeu d’entrainement sont excellentes mais celles sur le jeu de test sont très mauvaises. Les différents modèles surajute beaucoup trop, la perte ne diminue pas d’une epoch à l’autre (au contraire elle monte) et l’exactitude n’augmente pas. Le problème vient du fait que les modèle de deep learning ont besoin d’une grande quantité de donnée pour donner de bons résultats. Les 2800 titres d’articles financiers à ma disposition ne sont clairement pas assez suffisants pour donner de bons résultats.

Pour ce qui est des tweets financiers on constate que les modèles donnent de très bons résultats quand les données de tests sont également des tweets financiers. Dès que l’on teste le modèle avec les titres d’articles financiers ont vois que les performances diminuent beaucoup. On constate donc que la structure des titres d’articles financiers et des tweets financiers est trop différente l’une de l’autre. Le modèle n’arrive pas bien à généralisé et donne de mauvais scores.

Les modèles de machines learning développé à partir des titres d’articles financiers donnent des résultats plus intéressants. Les 2 modèles ayant les meilleurs résultats pour ce projet sont :

* La forêt aléatoire, prétraitement avec Spacy, utilisation du Tf-Idf, entrainement avec recherche d’hyper-paramètres aléatoire.
* La forêt aléatoire, prétraitement avec Spacy, moyenne des vecteurs de mot, entrainement par validation croisée.

Ces deux modèles comportent plusieurs avantages. Le premier est qu’ils sont développés à partir d’une fôret aléatoire. L’utilisation de la forêt aléatoire augmente la chance que le modèle ne surajuste pas. De plus, ces 2 modèles ont été entrainés à l’aide de la validation croisée. Comme le montre les scores obtenus à l’aide de ce procédé les modèles n’ont pas l’air de surajuster. Pour le modèle utilisant la moyenne des vecteurs de mots, le graphe T-sne montre que les données sont assez bien réparties dans l’espace ce qui veux dire que le modèle apprend à partir de phrase assez différente l’une de l’autre. Voici ci-dessus plusieurs preuves qui montre que le modèle ne surajuste pas.

J’ai choisi ces deux modèles car, d’une part le score du modèle avec le Tf-Idf est le meilleur score que je puisse obtenir sur ces données sans surajustement. Pour le modèle de Word2vec.

Pour obtenir de meilleurs résultats il faudrait avoir plus de données à disposition. Les scores des modèles de machine learning et deep learning serait alors amélioré.

# Approfondissement possible

## Utilisation de Bert

Bert[[15]](#footnote-15) est un modèle spécialisé dans les tâches d’analyse de language (NLP). C’est un modèle préentrainé qui a été développé par Google. Ce modèle donne de très bon résultat pour l’analyse de sentiment. Il serait incessant de développer un modèle en utilisant cette technologie pour visualiser les scores qu’il est possible d’atteindre.

## Agrandissement du jeu de données

EDA[[16]](#footnote-16) (Easy Data Augmentation) est un ensemble de technique permettant d’augmenter la taille d’un jeu de données textuelle. Les techniques proposé ici sont :

* Remplacement d’un nombre aléatoire de mot dans la phrase par leur synonyme.
* Insertion d’un nombre aléatoire de synonyme dans la phrase à partir des mots de cette phrase.
* Inversion de deux mots dans la phrase plusieurs fois.
* Suppression de mot aléatoire dans la phrase.

Cette technique permet de créer de nouvelle phrase et donc d’augmenter la taille du jeu de données. Cette solution donne apparemment de bons résultats sur les petits jeux de données comme celui que j’ai à disposition.

Selon la quantité de phrase généré, il pourrait être intéressé d’exécuter les modèles de machine learning et de deep learning sur ces nouvelles données pour voir si les scores des modèles augmentent.

# Conclusion

# Bibliographie

## Internet

* <https://medium.com/>
* <https://www.wikipedia.org/>
* <https://stackoverflow.com/>
* <https://scikit-learn.org/stable/user_guide.html>
* <https://keras.io/>
* <https://www.kaggle.com/>
* <https://towardsdatascience.com/>
* <https://stackabuse.com/>
* <https://spacy.io/api/doc>
* <https://radimrehurek.com/gensim/>
* <https://realpython.com/>

## Livres

* Machine learning avec Scikit-Learn, Aurélien, Géron, Paris, Dunod
* Deep Learning avec TensorFlow, Aurélien, Géron, Paris, Dunod

1. <https://bitbucket.org/ssix-project/semeval-2017-task-5-subtask-1/src/master/> [↑](#footnote-ref-1)
2. <https://bitbucket.org/ssix-project/semeval-2017-task-5-subtask-2/src/master/> [↑](#footnote-ref-2)
3. <https://spacy.io/> [↑](#footnote-ref-3)
4. <https://tedboy.github.io/nlps/generated/generated/gensim.utils.simple_preprocess.html> [↑](#footnote-ref-4)
5. <https://docs.scipy.org/doc/scipy/reference/sparse.html> [↑](#footnote-ref-5)
6. <https://radimrehurek.com/gensim/> [↑](#footnote-ref-6)
7. <https://nlp.stanford.edu/projects/glove/> [↑](#footnote-ref-7)
8. <https://radimrehurek.com/gensim/sklearn_api/d2vmodel.html> [↑](#footnote-ref-8)
9. <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html> [↑](#footnote-ref-9)
10. <https://en.wikipedia.org/wiki/GloVe_(machine_learning)> [↑](#footnote-ref-10)
11. <https://nlp.stanford.edu/projects/glove/> [↑](#footnote-ref-11)
12. <https://stackoverflow.com/questions/31681373/making-svm-run-faster-in-python> [↑](#footnote-ref-12)
13. Explication de la validation croisée dans le livre « Machine learning avec Scikit-Learn, Aurélien, Géron, Paris, Dunod », page 68. [↑](#footnote-ref-13)
14. <https://keras.io/layers/recurrent/#lstm> [↑](#footnote-ref-14)
15. <https://en.wikipedia.org/wiki/BERT_(language_model)> [↑](#footnote-ref-15)
16. <https://github.com/jasonwei20/eda_nlp> [↑](#footnote-ref-16)