SatisPy Project

Rapport d'Évaluation

Saurons-nous modéliser correctement un corpus de commentaires clients pour en évaluer leur satisfaction ou mécontentement ?

Auteurs : Fred ZANGHI & Quan LIU
Promotion : DataScientest – Juillet 2022
Mentor de projet : Antoine Tardivon (DataScientest)

Rapport d'Évaluation - SatisPy Project

Préambule :

Le présent rapport est destiné à tout public et ne comporte aucune ligne de code.

Par souci de simplification, nous noterons tout vocabulaire qui concerne le code en couleur et les variables en *italique*.
Enfin, une notation d'exposant[x] nous renvoie en fin de rapport pour un complément d'information ou des références utiles.

Participants au projet :

Quan Liu : « Je suis ingénieur de sûreté de fonctionnement dans le domaine automobile et ferroviaire. Mais j'aime bien analyser des données et faire la programmation. Recommandé par mon amie, je suis la formation de DataScientest pour apprendre et approfondir mes connaissances en Data Science »

Fred Zanghi : « Je suis en complète reconversion professionnelle, débutant en informatique depuis quelques mois seulement, et me suis pris de passion pour l'intelligence artificielle grâce aux vidéos de DataScientest. Je suis en formation Data Scientist puis approfondissement Engineering.

Antoine Tardivon : mentor du projet et précieux soutien de chez DataScientest

Sommaire

Table de Figures

I. Contexte

Durant notre parcours de formation Data Scientist, nous avons choisi de réaliser notre projet d'étude sur la mesure de la « Satisfaction des Clients » de sites marchands.

Nous avons été motivés dans ce choix par plusieurs aspects :

- Réaliser une étude approfondie et des modélisations de Text Mining dont l'emploi est très prisé par la grande majorité des entreprises, afin d'acquérir une certaine expérience dans ce domaine.
- Mettre en pratique les modules dédiés à ce sujet de notre formation et trouver de nouvelles ressources à exploiter.

Deux jeux de données nous ont été remis, venant d'une part du site **Trusted shops** dont les avis sont vérifiés, c'est-à-dire qu'ils font suite à une commande client, et d'autre part du site **Trustpilot**, qui sont des avis d'internautes. Enfin, les avis concernent 2 sites marchands : **Showroomprive.com** et **VeePee.fr**

II. Objectifs

Les données transmises comprennent environs 20.000 commentaires en caractéristiques principales. Les avis proviennent de nombreuses langues différentes.

Après avoir détaillé les notions que ce projet allait introduire et regardé les différentes pratiques disponibles sur le net, nous avons fait le constat d'un manque apparent de traitements et modélisations de la langue Française.

L'analyse de ces données nous a ensuite dirigé vers 2 principaux objectifs :

- Réaliser notre étude sur les commentaires de la langue Française exclusivement.
- Pouvoir prédire à partir de ces commentaire la note de satisfaction de la clientèle.

Nous avons ainsi nommé naturellement notre étude « **SatisPy Project** » puisqu'il est écrit en code Python.

III. Datas explorations et analyses

Les parties III et IV sont à retrouver en code python sous format Jupyter NoteBook dans le fichier nommé : PP1_Exploitation_Visualisation_&_Préparation_données.ipynb

III.1. Présentation rapide des données

Le dataset transmis est le fichier reviews_trust.csv comportant 19.863 lignes et 11 colonnes dont voici un court extrait :

Index	Commentaire	star	date	client	reponse	source	company	ville	maj	date_commande	ecart
4754	Todo perfecto	5	2021-01-09	nan	Bonjour , Nous vous remercions de votre confiance et fidélité.Ayoub	TrustedShop	ShowRoom	nan	nan	2020-12-06	34
4755	Plus que satistaite model conforme au photo qualité top	5	2021-01-09	nan	Bonjour , Merci d'avoir partagé …	TrustedShop	ShowRoom	nan	nan	2020-12-07	33
4756	You do n't have a real size cha…	4	2021-01-09	nan	nan	TrustedShop	ShowRoom	nan	nan	2020-11-29	41
4757	Purtroppo camaieu non ha più ne…	5	2021-01-09	nan	Bonjour , Merci pour votre genti…	TrustedShop	ShowRoom	nan	nan	2020-12-26	14
4758	Super	5	2021-01-09	Celia M .	Bonjour , Un grand merci pour vo…	TrustedShop	ShowRoom	Bonnevi…	nan	2020-12-25	15
4759	Esperienza buona ma dispiaciuta…	4	2021-01-09	nan	nan	TrustedShop	ShowRoom	nan	nan	2020-09-30	101

Figure 1 Observation globale

Nous remarquons :

- la présence de différentes langues utilisées en *commentaire*
- la notation du client en *star* allant de 1 à 5 étoiles
- la *date* de commentaire, *date_commande* et *ecart* entre les ces dates
- une *reponse* éventuelle du site marchand
- la *source* et *compagny* d'où proviennent nos données

Une fonction desc_var crée nous décrit les caractéristiques de nos 11 variables :

	Commentaire	star	date	client	reponse	source	company	ville	maj	date_commande	ecart
type	object	int64	object	object	object	object	object	object	object	object	float64
nb_nan	29.000	0.000	375.000	9,648.000	11,386.000	0.000	0.000	15,003.000	19,858.000	13,177.000	13,177.000
%_nan	0.146	0.000	1.888	48.573	57.323	0.000	0.000	75.532	99.975	66.339	66.339
count	19834	19,863.000	19488	10215	8477	19863	19863	4860	5	6686	6,686.000
unique	17174	NaN	1880	7569	2033	2	2	2457	5	385	NaN
top	Parfait	NaN	2020-06-12	Client	Bonjour , Merci d'avoir partagé	TrustedShop	ShowRoom	Paris	2021-04-30	2020-08-19	NaN
freq	284	NaN	257	31	1162	14503	16823	110	1	73	NaN
mean	NaN	3.408	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	14.286
std	NaN	1.651	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	10.373
min	NaN	1.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000
25%	NaN	1.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9.000
50%	NaN	4.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12.000
75%	NaN	5.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.000
max	NaN	5.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	304.000

Figure 2 Description statistique des données

Nous remarquons :

- la présence importante de valeurs manquantes NaN (Not a Number) dans *client, reponse, ville, maj, date_commande* et *ecart*
- une moyenne de notation *star* au-dessus de 3 indiquant des clients en moyenne satisfaits

III.2. Analyse graphique préliminaire des données avec nos remarques et décisions éventuelles

III.2.i. Distribution *star* avec filtre *réponse*

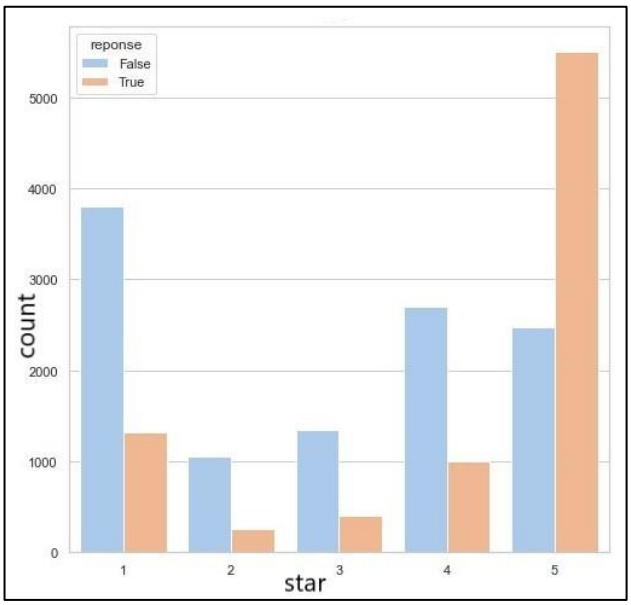

Figure 3 Distribution star avec filtre réponse

- les réponses du site se font majoritairement sur les clients très satisfaits
- le site a peu répondu aux clients très mécontents
- à l'exception des clients 'très satisfaits', le site répond peu aux notations
- dans tous les cas, il y a beaucoup de notations sans réponse du site

Etoile (*star*)	Sentiment du client
1	client très mécontent
2	client mécontent
3	client moyennement satisfait
4	client satisfait
5	client très satisfait

Tableau 1 Décision de classement des notes par étoile

III.2.ii. Distribution *source* avec filtre *star*

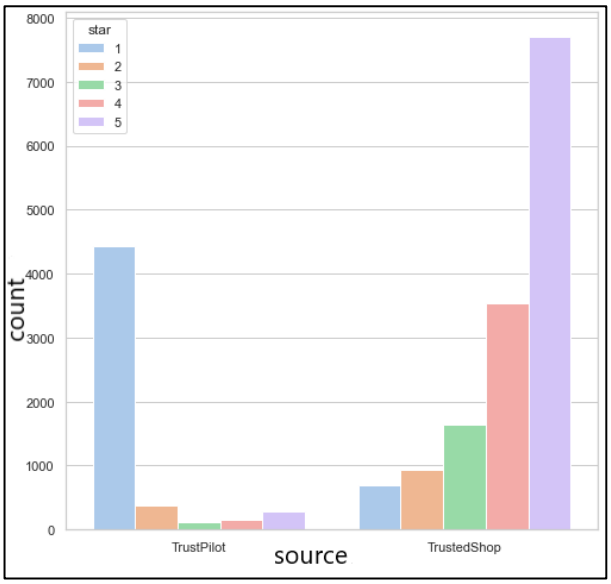

Figure 4 Distribution source avec filtre star

Il y a deux entreprises sources de nos données TrustPilot et TrustedShop

<u>Variation des données entre ces 2 entreprises :</u>

- TrustedShop a beaucoup + de données 'star' récoltées que son concurrent
- TrustPilot a essentiellement des notes de clients très mécontents
- TrustedShop a une grande majorité de clients très satisfaits, jusqu'aux clients peu contents

III.2.iii. Distributions des sites marchands de *compagny* avec filtre *star* ou filtre *source*

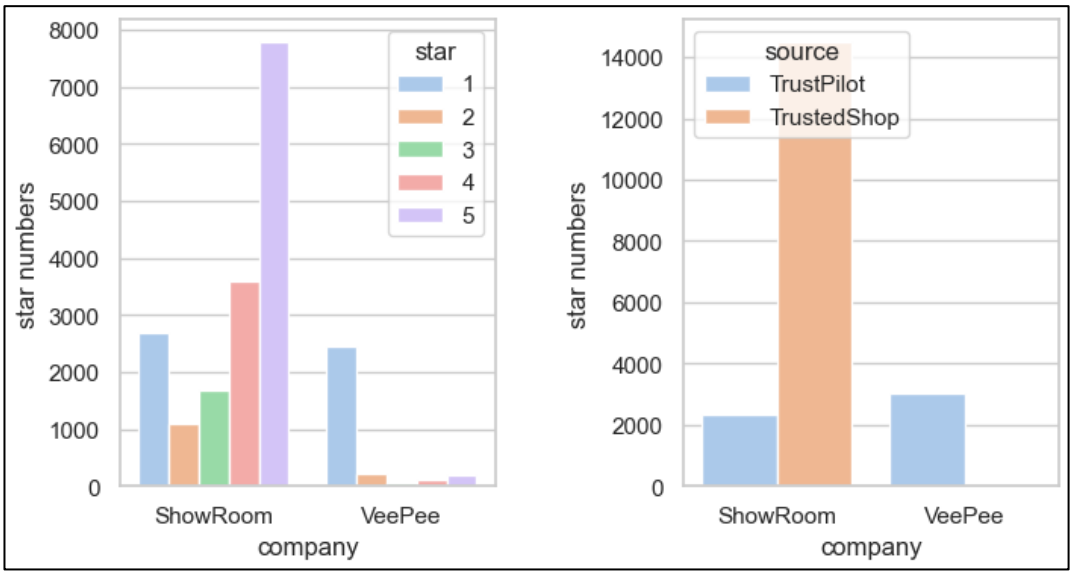

Figure 5 Distributions des sites marchands

- Site marchand showroom
  - a la grande majorité de notations étoiles

- a une grande majorité de clients très satisfaits et satisfaits, avec cependant un nombre non négligeable de clients moyennement satisfaits à très mécontents
- les sources de notations proviennent des 2 entreprises 'TrustPilot' et 'TrustedShop' avec une grande majorité pour ce dernier

- Site marchand VeePee
  - a peu de notations, comparé à son concurrent 'showRoom'
  - la grande majorité des notations concerne les clients très mécontents
  - l'entreprise 'VeePee' est la seule source de données de notations

III.2.iv. Analyse de *ville* de la clientèle

Un décompte des villes répertoriées dans nos données indique qu'il y a 2458 villes différentes.
En voici le top 20 :

```
il y a  2458  villes différentes
----------------------------------
top 20:
paris                140
marseille             66
lyon                  62
toulouse              49
strasbourg            39
nice                  38
lisboa                33
brive la gaillarde    32
chatenay malabry      31
bor                   26
toulon                22
portugal              22
nantes                20
nancy                 20
robert                19
porto                 19
metz                  19
fort                  18
caen                  18
rennes                18
Name: ville, dtype: int64
```

Figure 6 Top 20

- la colonne ville a un taux de NaN de 75.53%
- le top 20 est probablement une conclusion biaisée au vu du pourcentage élevé des valeurs manquantes
- il semblerait malgré tout que les clients renseignés dans cette colonne proviennent de France mais c'est un top 20 sur 2458 villes.

Écart en jours entre *date_commande* et *date* des notations ou commentaires clients

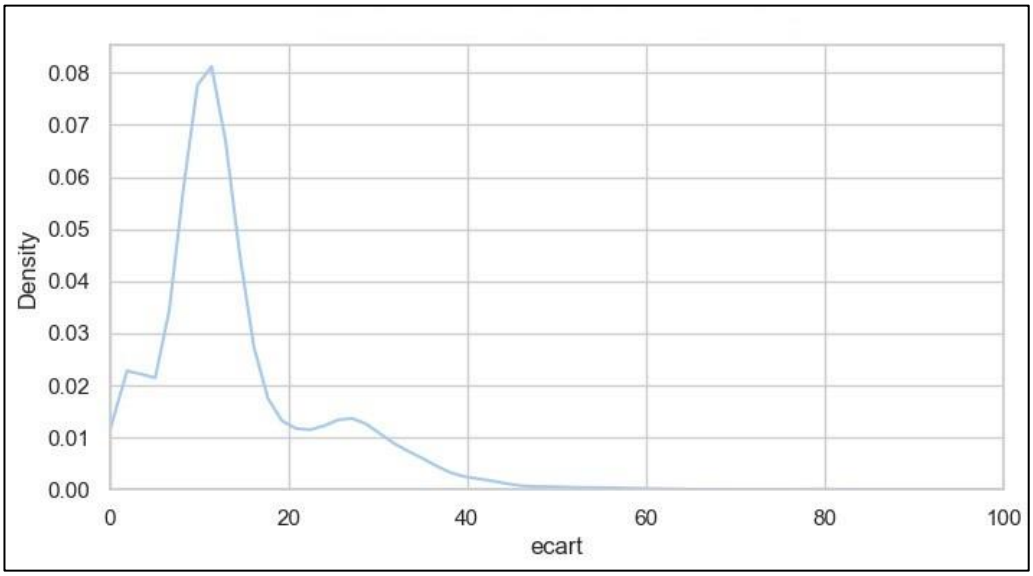

Figure 7 Écart en jours entre entre la date de commande et de commentaire

- Le délai entre la date de commande et l'avis du client reçu est proche de 15 jours
- ce résultat est biaisé par le fort pourcentage de NaN à 66.34%
- il y a aussi le biais du délai de livraison qui n'est pas rendu compte

III.2.v. Chronologie des notations par Années

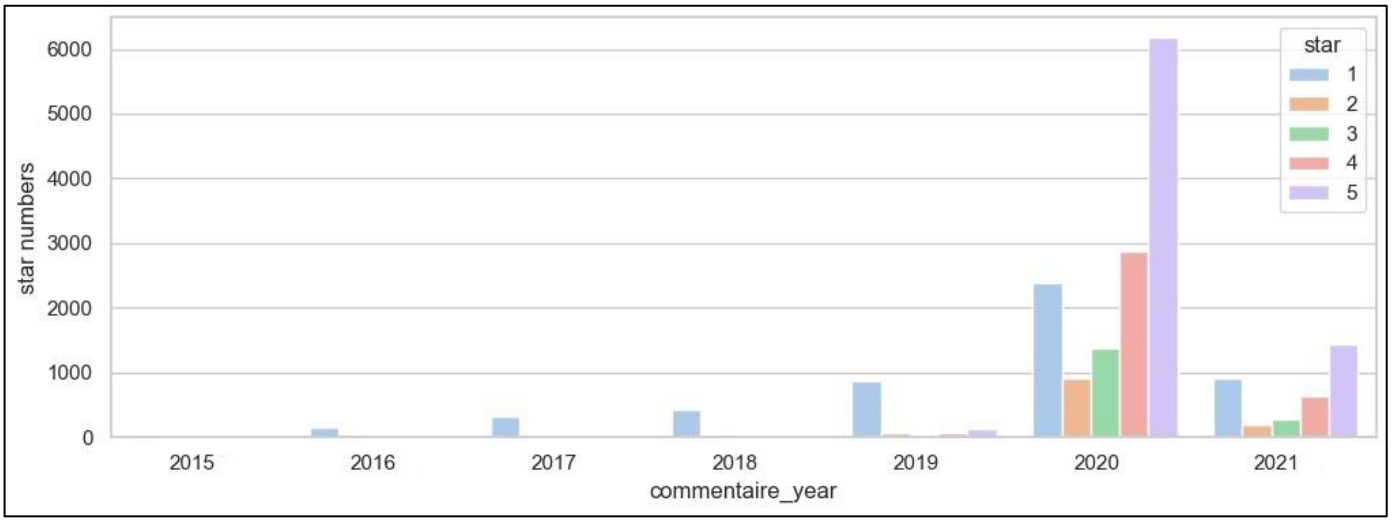

Figure 8 Date de commentaire

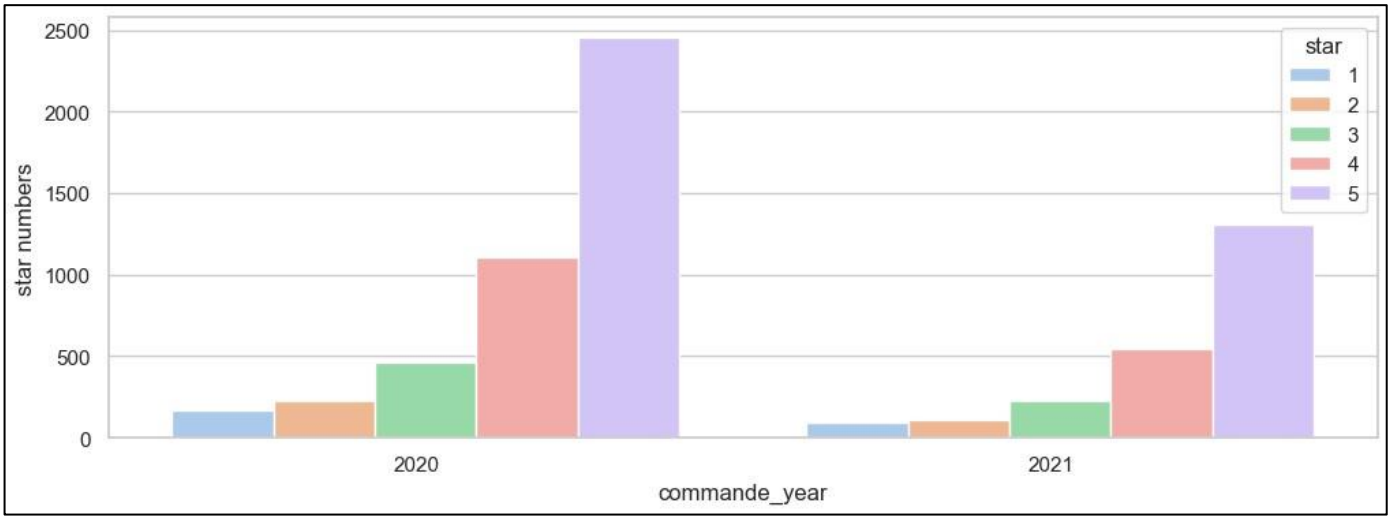

Figure 9 Date de commande

- Il y a 66.34% de NaN sur date_commande, le graphe présente essentiellement les informations en 2020 et 2021
- Evolution lente des commandes et notations clients à moins de 1000 par an jusqu'en 2019
- à partir de 2020, il y a une explosion des commandes, avec une majorité relative de notations positives et cependant bon nombre de clients très mécontents
- l'année 2021 semble biaisée par manque de données : à vérifier les dates les plus récentes de notations 'star' où absence de NaN
- on ne peut pas utiliser les dates de commandes pour compléter les dates de commentaires.

III.2.vi. client et maj

Ces 2 colonnes nous sont inutiles pour notre étude :

- *client* nous donne les noms et prénoms des clients, la colonne est donc à supprimer
- *maj* contient 99.97% de NaN est de type date mais non exploitable, colonne à éliminer également

III.2.vii. variable commentaire

L'analyse de cette colonne est réalisée après la préparation de nos données. C'est notre **variable principale** appelée **cible** (target). Une sélection de variables conjointes avec une préparation des données est indispensable pour éliminer la présence de NaN et les variables jugées inutiles afin de faciliter son analyse.

IV. Préparation des données et 1ère analyse de la variable *commentaire*

La partie D est à retrouver en **code python** sous format Jupyter NoteBook dans le fichier nommé : PP1_Exploitation_Visualisation_&_Préparation_données.ipynb

IV.1. Nettoyage des données

L'analyse de l'exploration des données nous préconise **d'éliminer les colonnes** suivantes :

- *client* : info personnelle
- *date_commande* : taux de NaN important, ne propose pas plus des informations
- *maj* : taux de nulle importante 99.98%
- *ecart* : intervalle entre les dates. ne propose pas plus des informations
-

Dont les colonnes à **conserver** avec modification éventuelle :

- Index original : pour pouvoir retrouver à tout moment la totalité de la ligne concernée
- commentaire : notre variable principale
- star original : pour vérifier à tout moment la pertinence des traitements et prédictions
- star : variable modifiée en sentiments positifs ou négatifs
- source
- company
- date

Nous **éliminons** également les lignes **doublons** inutiles.

Extraction des commentaires Français et 1ère sauvegarde

Après une **mise en minuscule** de la variable commentaire, nous utilisons la librairie langid afin de décompté le nombre de langues présent et leur proportion.

```
fr   0.871
en   0.042
it   0.021
es   0.021
pt   0.019
wa   0.006
de   0.004
oc   0.002
mt   0.002
nl   0.002
Name: language, dtype: float64
```

Figure 10 Distribution des langages

Plus de 87% des commentaires sont répertoriés en Français, mais après une analyse visuelle sur les résultats, il y a beaucoup de commentaires Français qui sont classés comme Anglais.

Des erreurs de classement « fr »sont également possible, mais la grande quantité de lignes dans ce corpus implique plutôt une vérification et élimination sur les erreurs réalisées par nos modèles de prédiction.

Un **filtrage manuel sur le petit corpus Anglais** peut donc être effectué pour le réinjecter dans le corpus Français et ne considérer que ce dernier. Ce filtrage est réalisé à l'aide d'un petit fichier de 1Ko appelé drop_list_en.csv et qui est joint au présent rapport.

Nous sauvegardons le dataset entièrement Français de **17.199 lignes** et 8 colonnes suivant nommé reviews_trust_fr_VF.csv et qui est joint au présent rapport.

Index	Commentaire	star	source	company	ndex_org	star_org	date
0	bonjour , ca doit faire 5 ans environ que j…	1	TrustPilot	ShowRoom	0	1	2021-06-20…
1	vente lacoste article manquant photo prise …	1	TrustPilot	ShowRoom	1	1	2021-06-20…
2	vente lacoste honteuse , article erroné , a…	1	TrustPilot	ShowRoom	2	1	2021-06-19…
3	j'ai commandé des mules de la marque moosef…	2	TrustPilot	ShowRoom	3	2	2021-06-19…

Figure 11 reviews_trust_fr_VF.csv

IV.2. Visualisation par WordCloud

WordCloud pour une première mise en bouche de la variable commentaire.

Le nuage de mots WordCloud[1] est une représentation visuelle qui complète une section de texte pour aider les lecteurs à mieux visualiser la présence de mots clés du texte concerné. Pour cela, nous avons besoin de Tokeniser[2] notre corpus commentaire.

Nous utilisons la méthode RegexpTokenizer pour obtenir le wordcloud suivant :

Figure 12 Wordcloud – Commentaires

V. Traitement de la variable commentaire

La partie E est à retrouver en **code python** sous format Jupyter NoteBook dans le fichier nommé : PP2 _Vectorisation_&_Filtrage_ commentaires_V2.ipynb

V.1. Choix de numérisation des étoiles *star*

Après avoir recharger notre fichier de sauvegarde reviews_trust_fr_VF.csv pour ce nouveau notebook, nous décidons de simplifier la notation de client allant de 1 à 5 étoile en classification binaire comme suit :

Star d'origine	Sentiment du client	Star binaire
1	client très mécontent	0
2	client mécontent	0
3	client moyennement satisfait	1
4	client satisfait	1
5	client très satisfait	1

Tableau 2 Regroupement des notations en classe binaire

Ainsi nous ramenons la prédiction de nos modèles à une classification de clients mécontents ou bien satisfaits, en considérant que la note de 3 étoiles médiane avait globalement plus de clients satisfaits que mécontents. Bien que ce choix reste subjectif, nous verrons par la suite si nos modèles de prédiction remplissent leur mission quant au choix présent.

V.2. Analyse du Verbatim

L'analyse du verbatim[3] du corpus *commentaire* est longue et fastidieuse, mais c'est un passage obligé afin de trouver ici des mots clés, des combinaisons de mots ou des erreurs d'interprétation à éliminer si l'on veut que nos modèles comprennent et classent correctement chaque *commentaire* en sentiment positif (numérisé 1) ou négatif (numérisé 0).

En observant de nombreuses lignes de commentaires nous avons fait les constats suivants :

- les mauvaises notations se réfèrent souvent à de très longs commentaires comportant de nombreuses ponctuations du type « !!!! », « !?!? » et leurs variantes.
- Les commentaires très courts de 1 ou 2 mots sont ici bien plus difficiles à classer, surtout pour des expressions positives comme « ras » signifiant « rien à signaler » qui sont parfois mals classées quand on les confonds avec le sentiment négatif « ras le bol ». De plus les nombreuses variantes de ce petit mots (r.a.s , r à d etc…) font que de nombreuses parties à venir de nos traitements peuvent causer problème par tokenisation et élimination des mots courts par exemple.
- Certains mots comme « commande » ou « livraison » par exemple apparaissent très souvent, autant dans des commentaires positifs que négatifs.

- Enfin, nous remarquons une grande quantité de fautes d'orthographe inévitables dans ce genre de commentaires ou des mots incongrus comme « soihzoekhfôi » qu'il sera difficile d'éliminer ou d'interpréter quoiqu'il en soit vu la taille de notre corpus.

L'ensemble de ces remarques étant posées, elles vont nous servir pour une stratégie de nettoyage de notre corpus et les tests à effectuer sur différents modèles de classifications.

V.3. Nettoyage du corpus

V.3.i. Problématique des « ras »

En utilisant la méthode de compilateur regex[4] , nous partons à la recherche de tous les « ras » et ses variantes présente dans le corpus afin de les comptabiliser et étudier leur note en star originale pour prendre une décision éventuelle de remplacement. Nous obtenons la synthèse suivante :

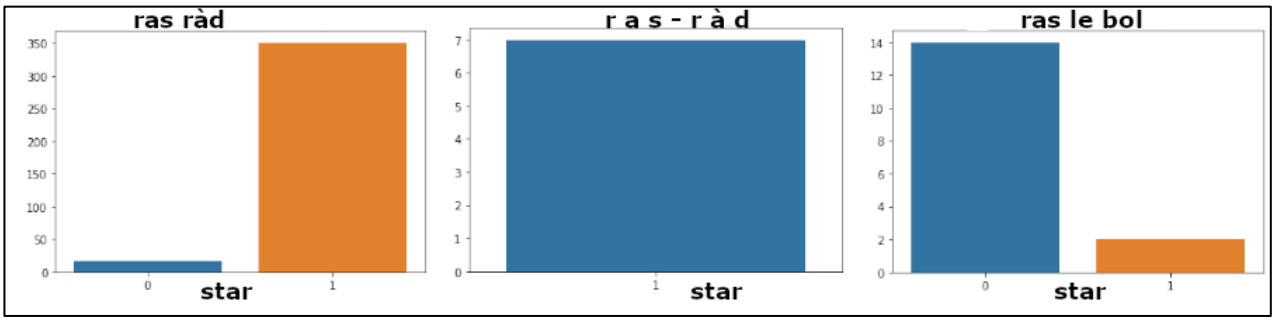

Figure 13 Proportions des « ras », ses variantes et « ras le bol » selon la notation binaire

Cela confirme que les « rien à signaler ou rien à dire » représentent en général des notations positives (+1) contrairement aux « ras le bol » a tendance majoritaire négative (0)

De plus cette problématique de « ras » représente en tout plus de 390 lignes de ce petit mot utilisé souvent comme seul commentaire, soit 2.3% de notre corpus.

Nous décidons de transformer les « ras le bol » en « mauvais » et les « ras » seuls ou leurs variantes en « bien » afin qu'ils ne soient pas éliminés par la suite et qu'il n'y ait plus de confusion possible.

4 expressions regex et substitutions ont été utilisées pour effectuer cette tâche réalisée en 1.3 seconde sur tout le corpus.

V.3.ii. Tokenisation

Nous décidons de tokeniser[2] notre corpus en **éliminant toutes les ponctuations, smileys, nombres et espaces** comme base de travail pour nos modèles, en gardant à l'esprit de tester par la suite un modèle sur la présence des ponctuations selon l'analyse précédente de nos verbatims.

Sont donc conservés tous les mots avec lettres accentuées et particulières suivantes :

- é è ê à â î ï à ù ç ô ë û æ œ

Ce choix a été fait pour dénaturer le moins possible les mots présents et réaliser au mieux selon nous les autres méthodes à venir.

Pour ce 1er nettoyage et ceux qui vont suivre, nous avons réalisé un ajout de 2 colonnes à notre dataset Français original, avec 1 colonnes des mots traités et 1 colonne de total de ces mots afin de comparer et vérifier chaque process. Il y a un total de **691.347 mots** tokeninés.

Index	Commentaire	star	source	company	index_org	star_org	date	tokens	total1
0	bonjour , ca doit faire 5 ans enviro…	0	TrustPi…	ShowRoom	0	1	2021-06…	['bonjour', 'ca', 'doit', 'faire', 'ans'…	188
1	vente lacoste article manquant photo…	0	TrustPi…	ShowRoom	1	1	2021-06…	['vente', 'lacoste', 'article', 'manquan…	20
2	vente lacoste honteuse , article err…	0	TrustPi…	ShowRoom	2	1	2021-06…	['vente', 'lacoste', 'honteuse', 'articl…	194
3	j'ai commandé des mules de la marque…	0	TrustPi…	ShowRoom	3	2	2021-06…	['j', 'ai', 'commandé', 'des', 'mules', …	26
4	commande téléphone etat a+ . livrais…	0	TrustPi…	ShowRoom	4	1	2021-06…	['commande', 'téléphone', 'etat', 'a', '…	44

Figure 14 Ajout de la colonne tokens et du total des tokens

La colonne tokens crée est donc une liste de de liste de mots.

Nous décidons de créer une fonction extend_list() pour transformer ce genre de colonnes en une seule chaine de caractères. L'idée est de pouvoir parcourir à la volée une partie des mots traités en pouvant tous les visualiser comme dans un unique paragraphe.

```
Out[11]: 'bonjour ca doit faire ans environ que je suis membre showrooprive je n ai jamais eu de
soucis en particulier à part petites bricoles par contre depuis ces derniers mois une vraie
catastrophe j ai eu commandes annulées d affiliés à se demander si les ventes sont bien réelles
histoires de nous faire payer et que showroom joue avec notre argent en se faisant des intérêts
bancaires sur notre dos la ère commande machines et cafés annulé mois après la e commande don t
call me jennifer annulé mois après la e commande techwood annulé mois après je l apprends toujours
par mail le même message bateau malgré tous les efforts bla bla bla bien évidemment pas de
```

Figure 15 Extrait de 500 caractères vus par la fonction extend_list

V.3.iii. Lemmatisation

La lemmatisation (lemmatisation en anglais) est une technique similaire à la racinisation.

Elle consiste à transformer des mots en leur lemme (lemma en anglais).

Le lemme correspond à un terme issu de l'usage ordinaire des locuteurs de la langue.

Nous utilisation pour ce faire la librairie spacy qui permet, après chargement d'un module dédié à la langue Française, de transformer par exemple le mot « voulais » en lemme « vouloir ».

La lemmatisation du corpus complet met bien plus de temps, soit **environ 6 minutes**, pour donner le tableau dataset complété suivants :

Index	Commentaire	star	ource	mpar	dex_o	ar_or	date	tokens	total1	lemmatiser	tot_lem
3	j'ai comm…	0	Tru…	Sho…	3	2	2021-06…	['j', 'ai', 'commandé', 'des', 'mu…	26	['j', 'avoir', 'commander', 'un'…	25
4	commande …	0	Tru…	Sho…	4	1	2021-06…	['commande', 'téléphone', 'etat', …	44	['commande', 'téléphone', 'etat'…	43
5	commande …	0	Tru…	Sho…	5	1	2021-06…	['commande', 'passée', 'pour', 'un…	35	['commande', 'passer', 'pour', '…	34
6	annulatio…	0	Tru…	Sho…	6	1	2021-06…	['annulation', 'de', 'commande', '…	47	['annulation', 'de', 'commande',…	47
7	extrêmeme…	0	Tru…	Sho…	7	1	2021-06…	['extrêmement', 'déçue', 'de', 'la…	66	['extrêmement', 'décevoir', 'de'…	65

Figure 16 Ajout de la colonne lemmatiser et son total

Cette méthode prenant du temps de calculs conséquent, nous sauvegardons le Dataset constitué dans un fichier review_trust_fr_lemmantiser_VF-19-09-22.csv qui sera remplacé en fin de traitement.

V.3.iv. Retirer les tokens de 2 lettres et moins

A ce stade, noter corpus contient encore **464.241 tokens** lemmatisés contre 691.347 après la 1ère tokenisation. Nous décidons d'éliminer tous les mots de 1 ou 2 lettres sans signification particulière afin de réduire le nombre de tokens restants et faciliter les calculs de modélisation à venir.

Une simple ligne de code sur les longueurs de mots > 2 nous donne le résultat suivant :

tokens	total1	lemmatiser	tot_lem	words+3	tot_+3
['bonjour', 'ca', 'doit', 'faire', 'an…	188	['bonjour', 'ca', 'devoir', 'faire', …	187	['bonjour', 'devoir', 'faire', 'envi…	113
['vente', 'lacoste', 'article', 'manqu…	20	['vente', 'lacost', 'article', 'manqu…	19	['vente', 'lacost', 'article', 'manq…	15
['vente', 'lacoste', 'honteuse', 'arti…	194	['vente', 'lacost', 'honteux', 'artic…	193	['vente', 'lacost', 'honteux', 'arti…	119
['j', 'ai', 'commandé', 'des', 'mules'…	26	['j', 'avoir', 'commander', 'un', 'mu…	25	['avoir', 'commander', 'mule', 'marq…	16
['commande', 'téléphone', 'etat', 'a',…	44	['commande', 'téléphone', 'etat', 'av…	43	['commande', 'téléphone', 'etat', 'a…	29

Figure 17 Ajout de la colonne words+2 et son total

V.3.v. Filtrage des mots corbeille *'No-Stop-Words'*

Nous décidons d'analyser la fréquence des tokens de tout le corpus ainsi traité afin de déterminer une liste de mots à éliminer et nous avons décidé d'éliminer tous les mots uniques du corpus.

En effet, les **mots de fréquence unique** sont en nombre de **6.725** ! Ils représentent essentiellement des fautes de frappe ou d'orthographe, des mots étrangers à la langue Française et parfois même des mots délibérément écris sans aucune signification.

Nous utilisons pour cela notre fonction extend_list() pour obtenir une seule liste de tokens, puis nous créons un dictionnaire des fréquences d'apparition de chacun d'eux. Voici un extrait du résultat :

```
print(sortedDict)

13050
[('être', 22603), ('avoir', 20174), ('pour', 7517), ('commande', 7299), ('livraison', 5100), ('très', 5052), ('plus', 4800), ('fa
ire', 4166), ('mais', 3838), ('recevoir', 3712), ('client', 3594), ('produit', 3533), ('bien', 3531), ('vous', 3443), ('site', 34
38), ('colis', 3337), ('article', 3307), ('service', 3299), ('tout', 3174), ('dans', 2850), ('commander', 2782), ('vente', 2772),
('avec', 2752), ('dire', 2374), ('rien', 2065), ('même', 2058), ('leur', 2032), ('remboursement', 2018), ('jour', 1895), ('cela',
1887), ('toujours', 1861), ('aucun', 1858), ('mois', 1852), ('après', 1821), ('fois', 1757), ('retour', 1730), ('livrer', 1721),
('donc', 1705), ('aller', 1696), ('nous', 1643), ('mail', 1616), ('jamais', 1581), ('depuis', 1444), ('attendre', 1434), ('passe
r', 1425), ('priver', 1409), ('autre', 1359), ('achat', 1336), ('votre', 1327), ('sans', 1319), ('devoir', 1312), ('rembourser',
```

Figure 18 Extrait du dictionnaire des tokens par fréquence de présence

Nous avons donc **13.662 tokens différents** qu'il nous reste à filtrer en dernière intention.

Pour cela, nous créons une feuille excel afin de lister les tokens à éliminer pour des raison pratique de tri et recherche simplifié sur cette liste au besoin.

Nous sauvegardons alors liste_no-stop-words_tokens_unique.xlsx qui est joint au Github du projet.

Voici un extrait du fichier en question qui comporte **6.725 tokens unique** à éliminer :

	A
1	machiner
2	call
3	courageux
4	chantier
5	marteau
6	nuisance
7	moosefield
8	inaudible
9	diagonale
10	dolore
11	promesa
12	dévier
13	interré

Figure 19 Extrait du fichier liste_no-stop-words_tokens_unique.xlsx

Le *No-Stop-Words* est réalisé grâce au package nltk (natural language toolkit) et son module à charger stopwords.

Nous obtenons après traitement les colonnes *no_stop_words* et tot_sw ajoutées suivantes :

lemmatiser	tot_lem	words+2	tot_+2	no_stop_words	tot_sw
['bonjour', 'ca', 'devoir', '…	187	['bonjour', 'devoir', 'faire…	133	['bonjour', 'devoir', 'faire…	131
['vente', 'lacost', 'article'…	19	['vente', 'lacost', 'article…	16	['vente', 'lacost', 'article…	16
['vente', 'lacost', 'honteux'…	193	['vente', 'lacost', 'honteux…	134	['vente', 'lacost', 'honteux…	130
['j', 'avoir', 'commander', '…	25	['avoir', 'commander', 'mule…	17	['avoir', 'commander', 'mule…	16
['commande', 'téléphone', 'et…	43	['commande', 'téléphone', 'e…	32	['commande', 'téléphone', 'e…	31

Figure 20 Ajout de la colonne no_stop_words et son total

Ceci constitue le dernier processus de filtrage réalisé sur l'ensemble du dataset.

Il reste un total de **248.607 tokens** contre 691.347 avant la 1ère tokenisation. Nous avons donc réduit le nombre de mots à 35% de sa proportion initiale !

Nous sauvegardons à présent notre filtrage final dans un fichier en éliminant préalablement les colonnes de totaux inutiles .

Ce fichier de sauvegarde est nommé review_trust_fr_lemmantiser_word+2_VF.csv et comporte **17.199 lignes** et 12 colonnes (8 colonnes de départ + tokens, lemmatiser, words+2 et no_stop_words).

VI. Modélisations, Prédictions et Analyses des résultats

La partie F est à retrouver en **code python** sous format Jupyter NoteBook dans le fichier nommé : PP3_Modelisations.ipynb
Nous avons décidé de lancer divers modèles de Machine learning et de Deep Learning sur les 2 datasets suivant :

- reviews_trust_fr_VF.csv : où un simple nettoyage des NaN et lignes doublons a été effectué. Un filtrage particulier sera adapté à chaque modèle utilisé.
- review_trust_fr_lemmantiser_word+2_VF.csv : où un filtrage complet a déjà été réalisé comme indiqué dans notre rapport de présentation.

L'objectif est de pouvoir comparer les différents résultats et en sortir une analyse complète afin d'orienter nos recherches ultérieures sur la ou les meilleures solutions.

VI.1. Modèle 1 : Logistic Regression

Nous avons vu précédemment, avant tout processus de filtrage, au paragraphe « Analyse du Verbatim », que visuellement, beaucoup de commentaires négatifs avaient en commun une grande quantité de mots ainsi que bon nombre de signes de ponctuations.

Nous souhaitons donc commencer tout simplement par vérifier ce constat et tenter le modèle de classification le plus basique : la régression logistique[5] basé sur la régression mathématique binomiale.

Nous décidons de travailler sur le dataset non filtré reviews_trust_fr_VF.csv et commençons par quelques 1ers traitements et analyses.

Trouver les signes de ponctuation et tailles des commentaires

Avec 3 nouvelles et simples méthodes regex, nous parvenons à détacher les points d'exclamations, les points d'interrogations et les cascades de points enchainés que nous avions remarqué chez de nombreux clients mécontents.

Nous avons regroupé les résultats en fonction de notations 0 pour les clients mécontents et 1 pour les clients satisfaits dans le tableau qui suit :

star	exclamation	interrogation	etc	nb_caracter
0	12901	1232	4339	2947360
1	2549	218	1147	990510

Figure 21 Décompte des ponctuations et nombre de mots selon la notation star

Ce tableau confirme que dans chaque cas, les personnes mécontentes emploient entre 3 et 5 fois plus de point d'exclamations, d'interrogations , de suspension et de caractères dans leurs commentaires.

Nous décidons donc de conserver ces 4 caractéristiques en ajoutant leur décompte par commentaire dans notre dataset de départ.

Ajout de colonnes de dates

Nous décidons d'ajouter également au dataset des colonnes de dates par année, mois, et jours afin d'éliminer la variable initiale *date* et la remplacer par plus de colonnes utiles à nos modèles.

Le but est finalement d'éliminer toutes les variables de départ en les remplaçant par d'autres plus « parlantes » et numérisées comme ce qui suit.

Numérisation des colonnes source et compagnie

Par la méthode get_dummies() de la librairie pandas, nous numérisons par des colonnes binaires les classes des variables *source* et *compagnie* pour éliminer ces dernières par la suite.

Nous sauvegardons le data_set entier après numérisation pour une utilisation ultérieure si besoin en le nommant reviews_trust_fr_LR.csv.

Notre dataset est maintenant prêt à l'emploi pour nos modèles et en voici un aperçu :

exclamation	interrogation	etc	nb_caracter	year	month	weekday	source_TrustPilot	source_TrustedShop	company_ShowRoom	company_VeePee
21	2	1	1182	2021	6	6	1	0	1	0
0	0	0	138	2021	6	6	1	0	1	0
1	0	0	1191	2021	6	5	1	0	1	0
2	0	0	148	2021	6	5	1	0	1	0
6	0	0	272	2021	6	5	1	0	1	0

Figure 22 Variables conservées pour la modélisation Logistic regression

Lancement du modèle LogisticRegression

Après séparation des données en 80% de un **jeu d'entrainement (train_set)** et un **jeu de test** de 20% des données **(test_set)**, nous instancions notre modèle, l'entrainons par la méthode fit() sur le train_set et obtenons la matrice de confusion et les scores suivants :

predict	0	1
real		
0	895	277
1	109	2159

	precision	recall	f1-score	support
0	0.89	0.76	0.82	1172
1	0.89	0.95	0.92	2268
accuracy			0.89	3440
macro avg	0.89	0.86	0.87	3440
weighted avg	0.89	0.89	0.89	3440

Figure 23 Matrice de confusion - LogisticRegression et Classification report

Analyse des résultats

L'**exactitude** (accuracy), soit le taux de prédiction correct est de 89%.

La **précision** est également de 89% pour les clients mécontents (0) comme pour les clients satisfaits(1)

Le **rappel** (recall), soit la sensibilité de la prédiction est meilleure pour les clients satisfaits avec 95% que pour les clients mécontents avec 76%, ce qui se retrouve dans la matrice de confusion avec près de 2.5 fois plus d'erreurs sur les faux satisfaits (277) que les faux mécontents (109).

Enfin, les temps de calculs du modèle et de ses prédictions est extrêmement rapide, ce qui est un plus.

Nous sommes très satisfaits de ces résultats et même quelque peu surpris d'un aussi bon score de prédiction en ayant fait jouer notre modèle non pas sur les commentaires (donc sans aucune analyse des mots) mais sur de simples variables temporelles, les 2 sites vendeurs *ShowRoom* et *VeePee*, les 2 sources de récoltes des avis *TrustedShops* et *Trustpilot*, ainsi que sur le nombre de mots écrits et les seules ponctuations des commentaires !

VI.2. Modèle 2 : Gradient Boosting Classifier en 2 tests distincts

Nous poursuivons par un classifier un peu plus élaboré, le GradientBoostingClassifier[6] qui va s'auto-évaluer et se corriger pas à pas. Nous allons appliquer ce modèle par 2 méthodes sur la base du fichier original non filtré reviews_trust_fr_VF.csv.

Nous séparons notre dataset chargé en un train_set et test_set comme précédemment.

1er test

Nous commençons tout d'abord par vectoriser nos commentaires par la méthode CountVectorizer[7] afin de transformer les mots en vecteurs utilisables par le modèle, puis nous entrainons notre modèle sur le train_set.

Le temps de calculs de cette opération est assez long (près de 6 minutes pour nous, mais dépend des performances de votre PC) et nous décidons de sauvegarder ce modèle pré-entrainé avec des mots-vecteurs sous le format pickle adapté à la situation afin de le réutiliser facilement après un chargement ne prenant plus quelques secondes.

La prédiction se réalise comme toujours sur le test_set et nous obtenons les résultats suivants :

Classe prédite	0	1
Classe réelle		
0	895	277
1	109	2159

	precision	recall	f1-score	support
0	0.89	0.76	0.82	1172
1	0.89	0.95	0.92	2268
accuracy			0.89	3440
macro avg	0.89	0.86	0.87	3440
weighted avg	0.89	0.89	0.89	3440

Figure 24 Matrice de confusion - GradientBoostingClassifier 1 et Classification report

Analyse des résultats

Nous avons **exactement les mêmes résultats** qu'au 1er modèle, ce qui est très étonnant. Même accuracy à 98%, précision sur les 2 classes à 89% et recall de 76% pour les mécontents contre 95% pour les satisfaits.

La matrice de confusion nous donnant les erreurs de prédiction est aussi étrangement exactement la même que le 1er modèle.

Nous décidons qu'à la fin des tests de tous nos modèles, un **tableau regroupant les erreurs** de chacun nous sera peut-être utile pour comprendre ces erreurs de prédiction et tenter de remédier à certaines d'entre elles.

2sd test

Nous reprenons notre fichier original non filtré reviews_trust_fr_VF.csv et décidons cette fois-ci de lui appliquer 2 filtres seulement qui ont été vus précédemment :

- l'élimination des mots de 2 lettres et moins
- le stop-word des tokens uniques

Tous les autres processus sont les mêmes avec les même paramètres qu'au 1er test de ce modèle.

Et nous obtenons les résultats suivants :

Classe prédite	0	1
Classe réelle		
0	901	271
1	135	2133

	precision	recall	f1-score	support
0	0.89	0.76	0.82	1172
1	0.89	0.95	0.92	2268
accuracy			0.89	3440
macro avg	0.89	0.86	0.87	3440
weighted avg	0.89	0.89	0.89	3440

Figure 25 Matrice de confusion - GradientBoostingClassifier 2 et Classification report

Analyse des résultats

Seule la **matrice de confusion** a cette fois-ci **légèrement changé** en perdant un peu de qualité de bonne prédiction et en ayant des faux-positif ou faux-négatifs aussi quelque peu différent.

C'est ce qui nous intéresse finalement à ce stade, obtenir des modèles assez performants comme ceux jusqu'ici, mais ayant des erreurs de prédiction différentes.

Nous pourrons d'une part mieux comprendre comment et pourquoi certaines erreurs sont faites ou pas grâce à notre tableau de comparaison finale, mais il y a aussi une idée derrière tout ça que nous allons vous dévoiler après l'étude de notre prochain modèle.

VI.3. Modèle 3 : Naïve Bayes Classifier (Classification Naïve Bayésienne)

La classification naïve bayésienne est un type de classification bayésienne probabiliste simple basée sur le théorème de Bayes avec une forte indépendance des hypothèses. Elle met en œuvre un classifieur bayésien naïf, ou classifieur naïf de Bayes, appartenant à la famille des classifieurs linéaires.

Nous reprenons ici notre data_set vectorisé reviews_trust_fr_LR.csv sauvegardé au 1er modèle pour gagner du temps, et nous lui appliquons les filtres supplémentaires du 2ème test du Gradient Boosting pour observer les résultats.

Classe prédite	0	1
Classe réelle		
0	908	264
1	140	2128

Figure 26 Matrice de confusion - Naïve Bayes Classifier

Analyse et 1ères conclusions

Nous n'avons pas présenté le rapport de classification qui n'a toujours pas changé mais seulement la matrice de confusion qui a peu varié pour information.

Nous pouvons en tirer plusieurs enseignements :

- Sur 3 modèles de classifications différents, nous obtenons les même bons résultats résumés à 89% de taux d'exactitude des prédictions, et les erreurs sont très proche en nombre bien qu'il nous faille encore le vérifier visuellement à ce stade.
- Que nous prenions un jeu de donné filtré préalablement ou non, cela ne change rien aux scores obtenus
- Notons que nous n'avons fait aucune recherche spécifique sur les paramètres de nos modèles pour tenter de marquer de vraies différences et que nous avons pris les modèles implémentés par défaut de la librairie scikit-learn.

Nous décidons d'effectuer des recherches sur internet, où de nombreuses ressources sont disponibles sur tous les sujets de Machine Learning et de Classification dans le but de changer radicalement de stratégie. Nous voulons obtenir absolument des modélisations qui ont des scores et des erreurs de prédictions bien différents, même si les résultats sont moins bon !

La stratégie derrière ces 1ers tests

Il y a tout d'abord ces tableaux d'erreurs de nos modèles que nous voulons réaliser afin de comparer efficacement nos modèles et améliorer les filtrages de la colonne commentaire, mais il y a bien plus.

Nous souhaitons faire appelle à la Sagesse de la foule que l'on traduit par the Wisdom of the Crowd[8]

Qu'est-ce que la **Sagesse de la Foule** ?

C'est une propriété Mathématique très ancienne qui nous dit que si on a une foule d'individus qui font tous, et à coup sûr, des prédictions juste au-dessus de 50%, alors nous pouvons utiliser tous leurs avis, faire une moyenne de leurs prédictions, et nous obtiendrons un score optimisé dans chaque cas.

Dit plus simplement par Ken Blanchard : *"Aucun de nous n'est aussi intelligent que nous tous".*

Et puisque nous avons appris dans notre formation DataScientest des techniques d'ensembles de modèles tels que le Voting, le Bagging ou le Boosting qui reposent sur ce principe, nous souhaiterions obtenir des modèles de prédiction bien différents, mais qui ont tous une accuracy d'au moins 50.1% (et bien sûr s'ils ont plus, ce n'est que mieux 😊)

Nos recherches sur internet, via les exemples de codes sur GitHub mais aussi les modules de cours de DataScientest sur les Réseaux de Neurones (spécialité du Deep Learning), nous ont amené à trouver des approches nouvelles et extrêmement formatrices.

VI.4. Modèle pré-entrainé Wikipedia2Vec[9]

Ce que nous avons fait jusqu'alors, c'est d'entrainer un modèle sur un pré-traitement de notre corpus de texte, et demander à ce modèle de nous réaliser des prédictions sur un test_set inconnu mais provenant de notre base de données.

En fouillant sur le net, nous sommes tombés sur une **vidéo YT** d'un tutoriel de Master-2-Sise-Data-Science, intitulé *analyse de sentiments*, **Text Mining – Modèle pré-entrainé Word2Vec** *avec Gensim*.

Le Modèle pré-entrainé s'appelle wikipedia2vec et il a été entrainé sur l'énorme base textuelle de Wikipédia, non pas pour l'analyse de sentiments, mais pour l'apprentissage des imbrications de mots et d'entités de Wikipédia. Les imbrications apprises mappent des mots et des entités similaires proches les uns des autres dans un espace vectoriel continu.

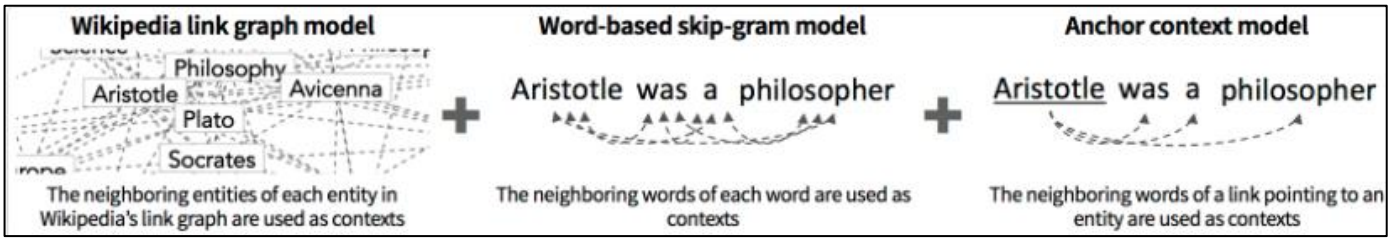

Figure 27 Schéma de principe de wikipedia2vec

Voilà donc un modèle déjà entrainé, sur une grande quantité de textes, que nous pourrions utiliser pour nos prédictions. Nous avons donc suivi pas à pas ce tuto dédié (voir toutes les liens en annexes du rapport). Nous avons dû évidement adapter le tuto à notre cas d'études.

Installation et Chargement du modèle wikipedia2vec

Le téléchargement du modèle est assez compliqué, son installation aussi car peu de liens explicatifs nous aidaient pour cela, mais nous avons réussi et décidé de donner toutes les pistes trouvées sur le notebook PP3 dédié aux modèles. Ce modèle fait 1.9Go, nous avons choisi le plus petit modèle pour commencer.

Le temps de chargement du modèle ensuite est relativement long (environ 6 à 8 min pour nous), aussi nous avons rapidement décidé d'en faire une archive pickle exploitable en 10 secondes environ sous le nom trained.pickle disponible dans le Github et de vous laisser le code entier de chargement pour information dans le notebook.

Chargement de notre data_set d'étude et 1er essai sur 4 mots

Nous décidons de commencer par le fichier filtré et archivé préalablement dans le paragraphe E review_trust_fr_lemmantiser_word+2_VF.csv.

Nous testons immédiatement notre wikipedia2vec pour vérifier son fonctionnement sur le mot 'papa' et lui demandant le 1er mot similaire : la réponse du modèle est 'maman'.

Puis nous indiquons au modèle de trouver le mot manquant si on lui indique que 'roi' et 'femme' sont 2 mots positifs et que 'homme' est un mot négatif : le mot 1er mot manquant négatif trouvé est 'reine'.

Ce petit test est un classique pour vérifier que notre modèle est bien chargé et fonctionne correctement.

Lancement de wikipedia2vec sur notre corpus filtré *no_stop_words*

Nous récupérons notre dernière colonne de texte filtré du fichier csv nommé *no_stop_words*, et entrainons wikipedia2vec sur la totalité du corpus avant séparation des données en train_set et test_set comme le tuto YT nous l'indique. C'est la méthode à utiliser avec tout modèle de pré-entrainement. Ce pré-traitement du corpus dure quelques secondes.

La colonne du corpus, de dimension vectorielle (17.199, 1), comporte 17.199 lignes de commentaires pour 1 seule colonne est alors transformée en **matrice vectorielle (17.199, 100).**

Il y a donc 100 colonnes numériques maintenant dont chaque ligne correspond au codage vectoriel d'un seul commentaire (c'est signe que tout a bien fonctionné puisque nous avons choisi le modèle version Française de wikipedia2vec de conversion en 100 vecteurs).

Puis nous séparons notre corpus matriciel pré-entrainé lié à notre variable cible star encodée (0 ou 1), en train_set et test_set afin de dérouler nos modèles de prédictions plus classiques comme vus précédemment.

Voici le schéma global d'utilisation d'un modèle pré-entrainé :

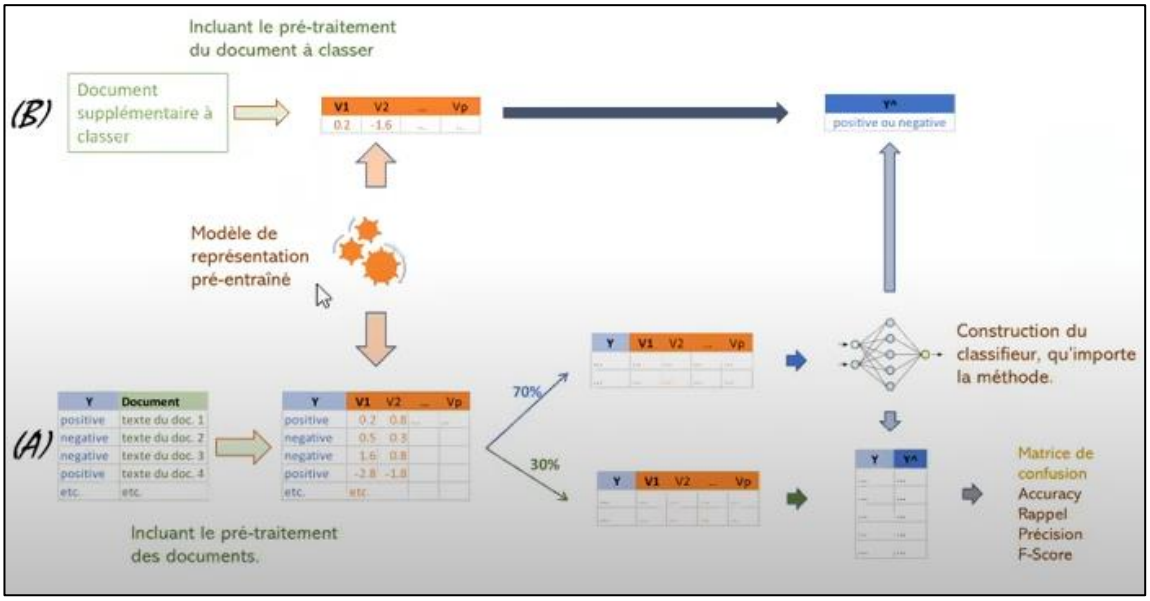

Figure 28 Schéma d'utilisation d'un modèle pré-entrainé (source tuto TY)

VI.5. Modèle 4 : Support Vector Machine (SVM) +_wikipedia2vec

Nous choisissons comme 1er modèle à lancer le SVM, que nous entrainons sur le train_set, ce qui prend moins d'une minute, et nous sauvegardons sous format pickle ce modèle entrainé sous le nom SVM.pickle.

Analyse des résultats

A l'exception de la précision sur la classe 1 de 90% des clients satisfait, toutes les autres mesures de scores sont en-deçà des résultats précédents mais non mauvais pour autant.

Classe prédite	0	1
Classe réelle		
0	961	224
1	310	1945

	precision	recall	f1-score	support
0	0.76	0.81	0.78	1185
1	0.90	0.86	0.88	2255
accuracy			0.84	3440
macro avg	0.83	0.84	0.83	3440
weighted avg	0.85	0.84	0.85	3440

Figure 29 Matrice de confusion et classification_report du SVM après wikipedia2vec

Nous remarquons cependant dans la matrice de confusion bien des changements, qui est bon signe pour comparer les erreurs de nos modèles

VI.6. Modèle 5 : Gradient Boosting Classifier + wikipedia2vec

Nous choisissons comme 2ème modèle le GBC déjà vu précédemment.

La méthode est la même que pour le SVM et nous obtenons les résultats suivants :

Classe prédite	0	1
Classe réelle		
0	866	319
1	258	1997

	precision	recall	f1-score	support
0	0.77	0.73	0.75	1185
1	0.86	0.89	0.87	2255
accuracy			0.83	3440
macro avg	0.82	0.81	0.81	3440
weighted avg	0.83	0.83	0.83	3440

Figure 30 Matrice de confusion et classification_report du GBC après wikipedia2vec

Les résultats sont à la baisse, mais toujours bien au-dessus des 50.1% recherchés.

VI.7. Modèle 6 : Réseaux de Neurones Artificiel (ANN[11]) + wikipedia2vec

Nous changeons désormais de registre pour passer du Machine Learning vers le **Deep Learning** avec le plus simple des réseaux de neurones artificiels.

La structure unitaire d'un neurone artificiel est appelée Perceptron[12], dont le fonctionnement s'est basé sur notre compréhension des vrais neurones naturels depuis les années 1957 et son invention par F. Rosenblatt.

Dans le principe, un réseau neuronal informatique est un ensemble de perceptrons qui prennent des informations en entrée et, en conjonction avec des informations provenant d'autres nœuds de perceptrons, développent des sorties sans règles programmées. Essentiellement, ils résolvent les problèmes par essais et erreurs.

Les modèles de réseaux de neurones se construisent très facilement avec les modules keras de la librairie Tensorflow, où nous indiquons pour chaque couche du réseaux, le nombre de neurones à considérer, depuis la 1ère couche d'entrée de données, jusqu'à la dernière couche de sortie servant à réaliser la prédiction souhaitée.

Puis nous lançons l'exécution des calculs qui ajustent automatiquement les paramètres du modèle, en réalisant un nombre de passage dans les données d'entrainement, appelé epoch, que nous avons fixé à 500 passages pour permettre un bon ajustement.

Cette exécution prend un peu de temps, selon la quantité de données à traiter, le nombre de perceptrons et donc de paramètres à ajuster qui en découle, et le nombre d'epoch indiqué. Pour nos PC assez peu performants, cela dure environ **4 minutes.**

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 100)]	0
Dense_1 (Dense)	(None, 50)	5050
Dense_2 (Dense)	(None, 20)	1020
Dense_3 (Dense)	(None, 8)	168
Dense_4 (Dense)	(None, 2)	18

Total params: 6,256
Trainable params: 6,256
Non-trainable params: 0

Figure 31 Couches de neurones (Layers) et nombre de neurones par couche de notre modèle ANN

```
Epoch 499/500
217/217 [==============================] - 0s 2ms/step - loss: 0.3049 - accuracy: 0.8658 - val_loss: 0.4386 - val_accuracy: 0.8173
Epoch 500/500
217/217 [==============================] - 1s 2ms/step - loss: 0.2990 - accuracy: 0.8640 - val_loss: 0.4126 - val_accuracy: 0.8248
Wall time: 4min 8s
```

Figure 32 Extrait de la fin des calculs d'epoch et son temps d'exécution

Nous obtenons les résultats de confusion et scoring suivants avec **936 erreurs** relevées :

```
[[1301  462]
 [ 474 2923]]
Wall time: 368 ms
```

	precision	recall	f1-score	support
0	0.73	0.74	0.74	1763
1	0.86	0.86	0.86	3397
accuracy			0.82	5160
macro avg	0.80	0.80	0.80	5160
weighted avg	0.82	0.82	0.82	5160

Figure 33 Matrice de confusion et classification report du ANN après wikipedia2vec

VI.8. Modèle 7 : Réseaux Récurrents RNN avec couche LSTM + wikipedia2vec

Le Deep Learning est en constante évolution. Ainsi, pour rester dans cette approche de reproduire le fonctionnement de vrais neurones naturels, les cellules à mémoire internes (cell) ont vu leur équivalent informatique par l'intermédiaire notamment des LSTM[13] (Long Short Term Memory) afin de doter les perceptron d'une mémoire longue à court terme.

Cette exécution prend beaucoup de temps, près d'**1 heure** dans notre cas !

Voici pour information les paramètres d'implémentation du modèle réalisé, une vue des epochs et le score obtenu :

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 300)          2039400

 dropout (Dropout)           (None, 300, 300)          0

 lstm (LSTM)                 (None, 100)               160400

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 2,199,901
Trainable params: 160,501
Non-trainable params: 2,039,400
```

Figure 34 Paramètres d'implémentation du modèle LSTM

```
Epoch 1/8
13/13 [==============================] - 165s 12s/step - loss: 0.7017 - accuracy: 0.5943 - val_loss: 0.6331 - val_accuracy: 0.6679 - lr: 0.0010
Epoch 2/8
13/13 [==============================] - 204s 16s/step - loss: 0.6255 - accuracy: 0.6559 - val_loss: 1.0784 - val_accuracy: 0.6679 - lr: 0.0010
Epoch 3/8
13/13 [==============================] - 244s 19s/step - loss: 0.5031 - accuracy: 0.7229 - val_loss: 1.0060 - val_accuracy: 0.6679 - lr: 0.0010
```

Figure 35 Extrait des epochs et scores intermédiaires lors de l'apprentissage du modèle

```
ACCURACY: 0.6779069900512695
LOSS: 0.4827301800251007
108/108 [==============================] - 42s 387ms/step
Wall time: 47.7 s
```

Figure 36 Score final du LSTM

Le score obtenu est le plus faible sur tous les modèles réalisés dans cette étude, il nous reste encore à travailler les paramètres, produire un tableau d'erreurs réalisées, et faire une prédiction éventuelle à la demande, mais le temps de calculs du modèle rend difficile cette poursuite sous nos installations actuelles.

VI.9. Tableaux comparatif des erreurs des modèles, analyses

Nous nous replongeons à présent dans l'analyse du verbatim en cherchant à étudier pourquoi les modèles font des erreurs, sur quel type de tokens ou sens de phrases etc…

Pour ce faire, nous réalisons à présent pour chaque modèle un tableau, qui ne comportera que les erreurs de prédictions, avec toutes les informations utiles pour notre étude.

Le modèle 1 de Logistique Regression nous donne le tableau d'analyse d'erreurs suivant :

Index	index_1	star_1	pred_1	bool_1	comment_vo
0	14776	1	0	False	problème avec un réveil qui ne fonctionnait plus.une étiquette retour m ' …
1	15017	1	0	False	excellent site de e commerce , imbattable , tout simplement ! !
2	10844	0	1	False	cette commande a cumulé les problèmes . j'ai été livrée très tard , bien p…
3	7751	0	1	False	je n'ai reçu que la moitié de ma commande pourtant validée par vos soins .…
4	15263	1	0	False	je n'ai jamais été déçu . .. j'achète en toute confiance

Figure 37 Extrait du tableau des erreurs de prédictions du modèle 1 LR

Ce tableau comporte **387 lignes d'erreurs** et pour rappel, ce modèle 1 n'a aucun filtrage en mode tokens.

La colonne *index_1* représente les indexes origine des commentaire afin de récupérer ces derniers.
La colonne *star_1* et *pred_1* représentent respectivement les vraies notations clients et leur prédiction par notre modélisation n°1.

Nous conserverons ensuite ce type de notation pour nous y retrouver dans les tableaux des autres modèles, et surtout pour les comparer entre eux dans un tableau général.

Le modèle 2 Gradient Boosting Classifier nous donne le tableau d'analyse d'erreurs suivant :

Index	index_2	star_2	pred_2	bool_2	comment_vo
0	14776	1	0	False	problème avec un réveil qui ne fonctionnait plus.une étiquette retour m ' a …
1	15017	1	0	False	excellent site de e commerce , imbattable , tout simplement ! !
2	10844	0	1	False	cette commande a cumulé les problèmes . j'ai été livrée très tard , bien plu…
3	7751	0	1	False	je n'ai reçu que la moitié de ma commande pourtant validée par vos soins . d…
4	15263	1	0	False	je n'ai jamais été déçu . .. j'achète en toute confiance

Figure 38 Extrait du tableau des erreurs de prédictions du modèle 2 GBC

Ce tableau comporte **336 lignes d'erreurs** et pour rappel, ce modèle 2 n'a aucun filtrage en mode tokens.

Il semble a 1ère vue que les erreurs soient les mêmes que pour le modèle 1 mais il y a quand même **51 erreurs de moins**. Ce qui peut-être utile par la suite pour comprendre ces erreurs.

Le modèle 3 Naïve Bayes Classifier nous donne le tableau d'analyse d'erreurs suivant :

Index	index_3	star_3	pred_3	bool_3	comment_vo
0	14776	1	0	False	problème avec un réveil qui ne fonctionnait plus.une étiquette retou…
1	15017	1	0	False	excellent site de e commerce , imbattable , tout simplement ! !
2	10844	0	1	False	cette commande a cumulé les problèmes . j'ai été livrée très tard , …
3	7751	0	1	False	je n'ai reçu que la moitié de ma commande pourtant validée par vos s…
4	15263	1	0	False	je n'ai jamais été déçu . .. j'achète en toute confiance

Figure 39 Extrait du tableau des erreurs de prédictions du modèle 3 NBC

Ce tableau comporte **386 lignes d'erreurs** et pour rappel, ce modèle 3 n'a aucun filtrage en mode tokens. A une erreur près, il y en a autant que pour le modèle 1 et les erreurs semblent les mêmes.

Nous devrons vérifier si ces erreurs sont similaires pour ces 3 premiers modèles, et n'en choisirons qu'un seul si c'est le cas pour nos études à venir dans un possible Voting sur les modèles retenus.

Le modèle 4 SVM sur wikipedia2vec nous donne le tableau d'analyse d'erreurs suivant :

Index	index_4	star_4	pred_4	bool_4	comment_vo
0	10792	1	0	False	ravissantes boucles d'oreilles qui ont mis beaucoup de temps pour être livrées .
1	13463	0	1	False	lors de ma commande g mis un article dans mon panier qui etait d…
2	10437	1	0	False	plus qu'un colis qui arrivent en plusieurs livraisons . j'attends toujours le colis 2 que je n'ai pas encore reçu .
3	3331	1	0	False	déçue qu'il manque un article alors que le stock était encore va…
4	7577	0	1	False	j'ai reçu ma commande 4 jours après la date prévue . il manquait…

Figure 40 Extrait du tableau des erreurs de prédictions du modèle 4 SVM sur wikipedia2vec

Ce tableau comporte **534 lignes d'erreurs** soit 148 erreurs de plus que le modèle précédent et pour rappel, ce modèle 4 est basé sur le fichier archivé review_trust_fr_lemmantiser_word+2_VF.csv qui comporte un filtrage complet de notre colonne *commentaire*.

VI.10. Prédictions en direct, à la demande

Pour nous aider à mieux comprendre les erreurs de nos modèles, nous avons décidé de leur demander un prédiction instantanée sur un texte que nous écrivons nous-même.

Pour cela, nous utilisons les fichiers au format pickle, qui ont sauvegardés chacun de nos modèles déjà entrainés.

Un chargement rapide du modèle souhaité, puis une ligne de code pour vectoriser un texte imaginé dans l'instant, et nous obtenons la prédiction satisfait 1 ou mécontent 0 en moins d'une seconde.

Prédictions sur le modèle Gradient Boosting Classifier 1 :

Le texte saisi « mauvais produit » nous donne une prédiction 1 = client satisfait !

Malgré 89% de précision et exactitude de ce modèle, nous voyons clairement que peu de mots induisent facilement une erreur de prédiction comme nous l'avons remarqué dans le tableau d'erreurs.

Prédictions sur le modèle SVM pré-entrainé sur wikipedia2vec :

Le texte saisi « mauvais produit » nous donne une prédiction 1 = client satisfait également !

Nous devons faire plus d'exploration à ce stade pour comprendre si nous faisons une erreur dans notre code de prédiction à la demande (qui est différents dans les 2 cas) ou bien si le résultat est normal et cohérent avec les erreurs que les modèles produisent tous 2.

Cette façon d'explorer nos modèles est dans tous les cas à conserver, puisqu'elle nous donne la possibilité de tester les modèles sur les commentaires qui sont bien prédits comme sur les mauvaises prédictions. Nous aimerions inclure cela dans notre présentation streamlit de soutenance

Prédictions sur le réseau de neurones ANN et tout autre modèle

Dans la partie réseaux de neurones du code, nous avons créé un fonction spécifique nommée prediction qui récupère un texte donné, et nous donne une prédiction selon le modèle de notre choix. Nous souhaitons la tester de multiples façon encore pour la présenter prochainement dans la soutenance streamlit accompagnée de quelques idées supplémentaires que nous souhaitons développer.

VII. Conclusions et perspectives

Traiter un sujet comme celui-ci a été passionnant et nous aimerions le poursuivre encore.

Nous y avons mis beaucoup de cœur, à découvrir toutes ces notions à travers nos modules de cours et à dérouler une grande quantité de modèles pour trouver des approches et ressources différentes.

Nous avons le sentiment d'en être qu'au tout début de notre étude, tant il nous reste de questions à résoudre, de tests à réaliser et de nouvelles pistes à explorer.

Nous comprenons que nous avons peut-être été assez redondants, dans cette synthèse de travaux, mais nous avons voulu rester fidèle à la réalité de « comment nous avons mené ce projet ».

Dans le déroulement comme dans les questionnements, nous avons procédé ainsi pour obtenir les solutions que nous avons proposé.

Nous espérons que sa lecture vous a été agréable et vous recommandons de parcourir les vidéos et liens donnés en annexe si vous souhaitez en savoir plus sur le sujet.

Vous pourrez retrouver tous les codes sous format notebook Jupyter, les fichiers de sauvegardes et autres ressources aux adresses suivantes :

- Github -> https://github.com/DataScientest-Studio/The-Satispy-Project/
- Google Drive -> https://drive.google.com/file/d/1OSTry-yIQkKjX0osWfhBoA2eaLCoain-/view?usp=sharing

Nos codes sont libres de droits et d'accès, vous pouvez donc en profiter pour les améliorer à votre guise et pourquoi pas nous aider à encore apprendre sur le sujet.

VIII. Remerciements

Nous tenons à remercier toute l'équipe de DataScientest pour son écoute et leurs conseils, et particulièrement notre mentor de projet Antoine qui nous a suivi chaque semaine en réunion zoom avec un sourire et une patience admirable.

Merci par avance à vous aussi cher lecteur, pour tous vos aimables commentaires à venir.

Annexe 1 : compléments d'informations ou références utilisées

WordCloud[1] : La bibliothèque WordCloud implémente un algorithme permettant d'afficher un nuage de mots d'un texte. Cet algorithme regroupe les étapes suivantes :

- Tokeniser le texte passé en paramètre
- Filtrer les mots vides
- Calculer la fréquence des mots
- Représenter visuellement les mots-clefs les plus fréquents sous forme de nuage de mots

(source: https://train.datascientest.com/)

Tokeniser[2] : méthode permettant de découper une ou plusieurs phrases en mots uniques appelés jetons (token en Anglais).

Il y a deux méthodes principales de tokenization :

- la méthode word_tokenize (sous package nltk.tokenize) : qui découpe une ou plusieurs phrases en mots uniques.
- la méthode RegexpTokenizer (sous package nltk.tokenize.regexp) : qui découpe des phrases en mots en ne conservant que les mots remplissant une condition donnée par expression regex.

(source Quan L et Fred Z ;-)

Verbatim[3] : Étude mot pour mot d'un texte afin d'en connaître les liens et contexte essentiels.

Regex[4] : une expression régulière est une chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles. Les expressions régulières sont également appelées regex (source https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re)

régression logistique[5] : En statistiques, la régression logistique ou modèle logit est un modèle de régression binomiale. Comme pour tous les modèles de régression binomiale, il s'agit d'expliquer au mieux une variable binaire par des observations réelles nombreuses, grâce à un modèle mathématique. (source https://fr.wikipedia.org/wiki/R%C3%A9gression_logistique)

GradientBoostingClassifier[6] : Le Boosting permet de construire un modèle, pas à pas en demandant au modèle de corriger les erreurs de son prédécesseur. Ainsi, ce modèle de classifier va s'auto-corriger par degré (gradient) successifs et s'évaluer tour à tour pour obtenir le meilleur score possible.

(source Datascientest)

CountVectorizer[7] : est un excellent outil fourni par la bibliothèque scikit-learn en Python. Il est utilisé pour transformer un texte donné en un vecteur sur la base de la fréquence de chaque mot qui apparaît dans l'ensemble du texte. Les mots n'étant pas compréhensibles par un ordinateur, il s'agit là de les rendre numérique sous forme de vecteur afin de pouvoir les comparer entre eux par exemple.

(source Quan L et Fred Z ;-)

the Wisdom of the Crowd[8] : La Sagesse le la Foule, une propriété mathématique bien séduisante que nous avons expliqué dans les grandes lignes dans ce rapport (paragraphe F) mais qui vous sera détaillée avec brio par Guillaume dans cette vidéo de 26 min : https://YouTube/7C_YpudYtw8

Wikipedia2Vec[9] : est un outil d'apprentissage des imbrications de mots et d'entités de Wikipédia. Les imbrications apprises mappent des mots et des entités similaires proches les uns des autres dans un espace vectoriel continu.

- Le site https://wikipedia2vec.github.io/wikipedia2vec/intro/
- Le tuto vidéo https://YouTube/FwSD1EM2Qkk

SVM[10] : Machines à Vecteurs de Support (Support Vector Machine)

ANN[11] : Réseaux de Neurones Artificiels (Artificiel Neuronal Network)

Perceptron[12] : Le perceptron est inventé en 1957 par F. Rosenblatt. Le perceptron est un neurone formel, le plus petit réseau neuronal possible, dont la fonction d'activation est une fonction échelon également appelé Linear Threshold Function, ce qui fait du perceptron une unité linéaire à seuil (ou Linear Threshold Unit). Découvrez la magnifique vidéo YT de Guillaume expliquant très simplement avec de belles illustration comment cela fonctionne : https://youtu.be/VlMm4VZ6lk4

LSTM[13] : La mémoire longue à court terme (LSTM) est un réseau de neurones artificiels utilisé dans les domaines de l' intelligence artificielle et de l'apprentissage profond . Contrairement aux réseaux de neurones à anticipation standard , LSTM a des connexions de rétroaction. Un tel réseau neuronal récurrent (RNN) peut traiter non seulement des points de données uniques (tels que des images), mais également des séquences entières de données (telles que la parole ou la vidéo).
https://en.wikipedia.org/wiki/Long_short-term_memory

Annexe 2 : liens pour retrouver tous nos codes

Fichiers joints

- Github du projet
  - Codes : (dossier “notebooks”)
    1. PP1-Exploitation_Visualisation et Préparation des données.ipynb
    2. PP2 - Vectorisation & Filtrage des commentaires.ipynb
    3. PP3_Modelisations.ipynb
  - Données : (dossier “data”)
    1. reviews_trust.csv
    2. drop_list_en.csv
    3. reviews_trust_fr_VF.csv
    4. review_trust_fr_lemmantiser_VF-19-09-22.csv
    5. liste_no-stop-words_tokens_unique.xlsx
    6. review_trust_fr_lemmantiser_word+2_VF.csv
    7. reviews_trust_fr_LR.csv
  - Modèles : (dossier “data” -> “Modèles”)
    1. ANN.pickle
    2. GBC_1.pickle
    3. GBC_2.pickle
    4. GBC_3.pickle
    5. SVM.pickle
    6. ber.pickle
    7. vectoriser_GBC_1
    8. vectoriser_GBC_2
    9. vectoriser_ber

- Google Drive
  trained.pickle
  https://drive.google.com/file/d/1OSTry-yIQkKjX0osWfhBoA2eaLCoain-/view?usp=sharing