

Mémoire :

**Optimisation du référencement d'un produit
sur une Marketplace : étude de cas Amazon**

Rédigé par **Olivier RANDAEL**

Master 2 : IASD (Intelligence Artificielle, Système et Données)
en collaboration avec l'AFIA CFA

Mémoire encadré par **Mr Claude Vincent** tuteur enseignant, **Mr Mickaël Avoledo**
chargé d'alternance, au sein de l'entreprise M13H et **Mme Patricia Lavagna**,
responsable pôle alternance pour l'AFIA CFA



Rédigé en Août 2020, année universitaire 2019-2020

Sommaire

| | | |
|----------|---------------------------------------------------------------------------------------------------------|-----------|
| 1 | Introduction | 3 |
| 2 | Avant de commencer | 4 |
| 2.1 | Remerciements | 4 |
| 2.2 | Vocabulaire | 5 |
| 3 | L'outil : motivation et choix | 6 |
| 3.1 | Présentation de l'outil | 6 |
| 3.2 | Concurrence sur le marché | 8 |
| 4 | Extraction de la donnée | 11 |
| 4.1 | Extraction à l'aide de Selenium, Scrapy ou BeautifulSoup | 11 |
| 4.2 | Méthode d'extraction retenue | 11 |
| 5 | Préparation de la donnée | 12 |
| 5.1 | Retrait des mots et caractères parasites | 12 |
| 5.2 | Stemming et lemmatisation | 13 |
| 6 | Extraction de mot-clefs : méthodes statistiques | 16 |
| 6.1 | Différents outils statistiques | 16 |
| 6.1.1 | TF-IDF | 16 |
| 6.1.2 | Rapid Automatic Keyword Extraction : Rake | 17 |
| 6.1.3 | TextRank | 18 |
| 6.2 | Présentation de la méthode retenue | 19 |
| 7 | Expliquer le référencement des produits par l'algorithme d'Amazon | 21 |
| 7.1 | Méthodes utilisées | 21 |
| 7.2 | Expérimentations | 23 |
| 7.2.1 | Étude statistique préliminaire | 23 |
| 7.2.2 | Création de modèles | 25 |
| 8 | Déterminer les similarités entre les mots-clefs : méthodes utilisant l'intelligence artificielle | 28 |
| 8.1 | Points communs entre les différentes méthodes : | 28 |
| 8.1.1 | Apprentissage profond | 28 |
| 8.1.2 | Word embedding | 30 |
| 8.2 | Présentation des différentes méthodes : | 31 |
| 8.2.1 | Word2Vec | 31 |
| 8.2.2 | GloVe | 34 |
| 8.2.3 | Transfer learning | 35 |
| 8.3 | Évaluation d'un modèle | 36 |
| 8.4 | Présentation de la démarche scientifique et de la méthode retenue | 38 |
| 8.4.1 | Expérimentations | 38 |
| 8.4.2 | Méthode retenue | 43 |

| | |
|---------------------------------------------------------------|-----------|
| 9 Réduction de dimension | 46 |
| 9.1 Différentes méthodes de réduction de dimensions | 46 |
| 9.1.1 PCA | 46 |
| 9.1.2 T-SNE | 47 |
| 9.2 Méthode de réduction retenue | 49 |
| 10 Démonstration de l'outil | 51 |
| 11 Automatisation de l'outil | 57 |
| 11.1 Implémentation | 57 |
| 11.2 Coûts et performance | 61 |
| 12 Conclusion | 62 |
| 13 Annexe | 65 |

1 Introduction

Ce mémoire a été réalisé en entreprise, encadré par les membres de l'équipe de data science de M13H. L'agence M13H évolue dans le secteur du data marketing. Elle accompagne ses clients pour mieux vendre leurs produits sur la toile. Elle possède une expertise pluridisciplinaire forte de ses deux entités techniques et marketing. L'équipe marketing conseille sur le choix de plateforme de gestion clients online et offline (*data management plateforme DMP*), elle prépare des études marketing d'attributions multileviers, effectue des campagnes de publicités ciblées et mesure l'impact de ces dernières. L'équipe data science met à disposition des dashboards pour la prise de décision, elle met en place les connexions et se charge de la gestion de la donnée. Elle conseille et implémente des datalakes reposant sur une technologie cloud de GCP (Google Cloud Plateform) et effectue des analyses de clustering pour isoler des populations. En 2018, M13H a été rachetée par Labelium, une agence internationale. Labelium propose une expertise marketing et un accompagnement des marques au travers des services de Google. Ce mémoire s'inscrit dans un objectif de rapprochement entre les différentes entités du groupe Labelium. En effet, Labelium possède un département SEO (Search Engine Optimization). Cette équipe gère le référencement des pages internet sur Google. Labelium a racheté l'entreprise Feed-Manager. Dans ce projet, Feed-Manager m'accompagne par l'intermédiaire de son expertise SEO sur la marketplace Amazon. L'outil développé pour ce mémoire permettra d'accompagner leurs clients pour le référencement de leurs produits sur Amazon. Dans la prochaine partie, l'outil sera présenté en détail, avant de mettre en avant une démonstration client à la fin de ce mémoire.



Figure 1: Illustration des missions de M13H au sein du groupe Labelium

Ce mémoire aura pour but de répondre à ces problématiques :

- Quel moyen utiliser pour extraire au mieux les mots les plus révélateurs d'un texte ?
- Quelle technique est la plus pertinente pour trouver des similitudes entre des mots ?
- Comment améliorer le référencement naturel d'un site ou d'un produit sur une marketplace en s'aidant des deux points cités précédemment ?

2 Avant de commencer

2.1 Remerciements

Ce mémoire s'inscrit dans une collaboration étroite avec Labelium et l'équipe de data science de M13H. Ce rapprochement a permis de créer un outil pertinent et utile à l'entité SEO de Labelium. Je remercie tout particulièrement l'équipe SEO de Labelium. Tout au long de ce projet, leurs conseils et leur dévouement ont permis de constamment challenger l'outil et de l'améliorer. De plus, j'ai eu la chance d'échanger et de suivre des présentations exposées par un expert de la marketplace d'Amazon : Mr Harold Christensen. Enfin, j'ai récemment pu collaborer avec Johann Lucas de FeedManager pour corriger et ajuster l'outil.

Évidemment, tout ce projet n'aurait pas vu le jour sans le soutien de M13H. Au cours de ce dernier mois, j'ai pu consacrer mon travail au développement de mon projet. Je tiens à remercier Mr Mickaël Avoledo, mon maître d'alternance, pour son suivi et ses conseils pertinents. Également, l'expertise de l'équipe data science m'a permis de développer cet outil de façon performante et rigoureuse. Leurs conseils ont été très précieux tant sur le plan technique que marketing. Je n'oublie évidemment pas l'ensemble de l'équipe M13H avec qui j'ai pu travailler. Ce mémoire souligne l'accomplissement d'une année de travail entourée de personnes exceptionnelles. J'ai eu grand plaisir à travailler au sein de cette société et je souhaite même expérience à quiconque cherche une entreprise à taille humaine, où esprit d'équipe rime avec bonne humeur et rigueur.

Enfin, je remercie amoureusement ma compagne Aloïse Uséo, pour son soutien, sa relecture attentive, ses nombreux conseils et sa patience. De plus, je tiens à saluer le travail professionnel de Mme Patricia Lavagna : l'encadrement de l'AFIA a permis de réaliser, malgré la conjoncture particulière, une année de master complète et rigoureuse. Ce dernier point n'aurait évidemment pas pu être réussi sans la détermination et le sérieux de l'ensemble des enseignants du master IASD et le dévouement de son directeur Mr Cazenave. Mes derniers remerciements vont pour mon tuteur enseignant, Mr Claude Vincent, sa disponibilité, ses nombreux conseils et son expérience m'ont permis d'accomplir ce mémoire.

Grand MERCI à l'ensemble des personnes ayant collaborées de loin comme de prêt à ce mémoire :



2.2 Vocabulaire

Ce mémoire détaille l'implémentation technique d'un outil. Il décrit des processus d'automatisation dans le cloud, des moyens d'extraire de la données sur le web ainsi que des méthodes statistiques et de machine learning pour apporter des solutions à un besoin marketing. Voici une liste de définitions importantes que le lecteur doit avoir en tête pour comprendre au mieux ce mémoire. Enfin, au cours de ce mémoire les sources seront spécifiées ainsi [18], et sont à retrouver avant l'annexe.

- **SEO** Search Engine Optimization : branche du marketing digital permettant d'améliorer le référencement naturel d'un site sans recourir à des moyens sponsorisés (payant). Par ce biais, il s'oppose au SEA (Search Engine Advertising)
- **Référencement** : sur une marketplace, comme sur un moteur de recherche, il caractérise la position d'un produit ou d'une page suivant la recherche d'un utilisateur. Cette position est déterminée par l'algorithme de référencement de la plate-forme (A9 dans le cas d'Amazon). Le SEO et le SEA permettent par un biais gratuit ou payant d'améliorer ce positionnement. Des entreprises du marketing digital sont spécialisées dans le conseil et développent des stratégies pour rendre plus visible un produit ou une page. Enfin, un meilleur référencement permet d'augmenter le nombre de visites et donc le nombre de ventes.
- **Scraping** : action de récupérer de façon automatique le contenu d'une page web
- **Epoch** : en apprentissage artificiel, lorsqu'un modèle de prédiction est créé, le modèle est amélioré plusieurs fois au cours de l'apprentissage. Il est donc amélioré à chaque epoch.
- **Checkpoint** : utilisé en machine learning lors de l'entraînement d'un modèle, à chaque epoch, différentes métriques sont calculées, si le modèle obtenu est meilleur que le meilleur modèle précédemment calculé alors ce dernier est sauvegardé ainsi que l'ensemble des métriques
- **Caractère** : il définit l'ensemble des éléments contenus dans un texte (mots, guillemets, ponctuations, smiley..). Une fois le corpus nettoyé, le nombre de caractères du corpus peut être appelé nombre de mots. En effet, la méthode de nettoyage de la donnée utilisée dans ce mémoire retire l'ensemble des caractères pour garder uniquement les mots du corpus.
- **Template** : c'est un anglicisme qui désigne un exemple sur lequel on se base. Une étude est souvent templatisée, sa mise en page finale est identique pour toutes les études.
- **Container** : un espace d'exécution dédié à une application logicielle
- **Docker** : un logiciel libre permettant de lancer des applications dans des conteneurs logiciels.
- **API Application Programming Interface** : application permettant de communiquer entre plusieurs logiciels. Il est possible d'envoyer des informations par l'intermédiaire d'un lien "http" (lien internet).
- **Bibliothèque** : en informatique, une bibliothèque permet au développeur d'accéder à des codes déjà implémentés. Ces codes sont open-source.

3 L'outil : motivation et choix

3.1 Présentation de l'outil

M13H souhaiterait développer un nouveau service pour ses clients. Cet outil a pour but d'accompagner des clients présents sur la marketplace d'Amazon. L'analyse résultante permettra d'améliorer le référencement d'un produit pour que celui-ci apparaisse sur la première page ou en meilleure position suite à une recherche d'un utilisateur sur le site d'e-commerce. Cette méthode utilise l'enrichissement sémantique de la page décrivant le produit. Cette recommandation est basée sur l'ensemble des compétiteurs identifiés lors de l'analyse. L'optimisation du SEO sur Amazon permet de mieux référencer un produit lors d'une recherche par mot clef. Ce référencement générera plus de visites et donc plus de ventes. Il est important d'avoir en tête que les utilisateurs regardent rarement les produits n'apparaissant pas dans la première page et cliquent beaucoup plus souvent sur les 3 premiers produits. Pour étayer notre choix voici plusieurs statistiques [26] mettant en avant le potentiel d'Amazon : 23% des acheteurs cherche l'inspiration sur cette marketplace, 120 millions de produits sont en vente sur cette plateforme, 4000 produits sont vendus chaque minute, enfin Amazon c'est 151 millions d'utilisateurs mobile aux États-Unis et il regroupe 2.5 millions de vendeurs.

Cette partie illustre l'utilisation de l'outil proposé par M13H. Un client fournit à M13H l'url de sa page produit. Le client souhaite mieux référencer son produit sur Amazon. C'est à dire qu'il aimerait que son produit soit présent sur de plus nombreuses recherches et plus haut dans les résultats de la première page. L'outil développé devra, dans un premier temps, récupérer le contenu de la fiche produit à l'aide d'une méthode de scraping. Ensuite il analysera les mots les plus fréquents du texte par l'intermédiaire d'une méthode statistique. Cela implique l'utilisation du RAKE et du TF-IDF. Ensuite, des recherches sur Amazon seront effectuées automatiquement pour chaque groupe de mots clefs. Parmi la liste de produits concurrents, un corpus sera constitué. A l'aide d'une méthode d'intelligence artificielle, je déduirai des mots similaires entre chaque produit. Cela implique l'utilisation de la méthode Word2Vec. Ainsi les mots obtenus permettront d'enrichir la fiche d'un produit. De plus, cet outil permettra au client d'acquérir plus de connaissances sur la concurrence de son produit sur la marketplace d'Amazon. L'objectif de ce mémoire est de parvenir à créer un outil d'une façon performante et de pouvoir mesurer les améliorations. Voici une illustration de ce cas client.

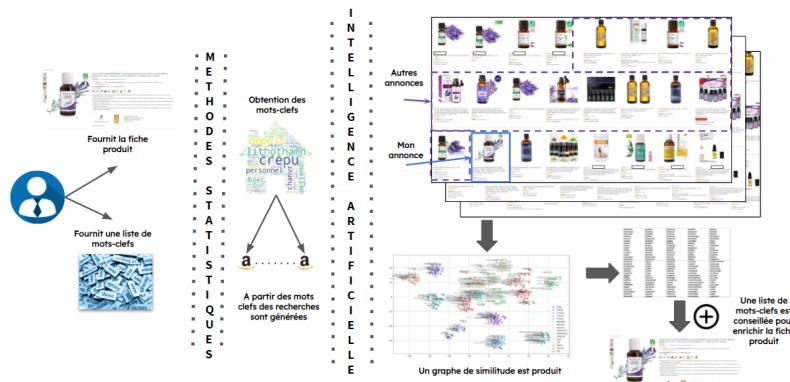


Figure 2: Illustration simplifiée de l'outil développé

D'un point de vue technique l'outil se décompose en deux parties. Une partie génère l'analyse, tandis que l'autre fait en sorte que les analyses soient les plus pertinentes. Dans ce mémoire, les différentes parties seront détaillées en présentant tout d'abord l'état de l'art puis une synthèse décrivant la méthode retenue. Mais avant d'explorer chaque détail de cet outil, voici un condensé des fonctionnalités implémentées.

L'outil, pour répondre à l'analyse demandée par un client Amazon, est implanté sur le cloud et a pour objectif d'être entièrement automatisé. L'analyse se découpe en deux parties. La première détermine les recherches Amazon et fournit des statistiques pour permettre à l'utilisateur de choisir les recherches les plus pertinentes. Le choix de l'utilisateur détermine l'ensemble des produits constituant le corpus et l'analyse concurrentielle réalisée dans la seconde partie. Les différentes étapes sont décrites ci-dessous.

Script 1 :

1. **scraping** : extraction de la donnée depuis la page produit du client
2. **data preprocessing** : traitement de la donnée par stemming et suppression des mots parasites
3. **group of Keywords scoring** : calcul d'un score pour chaque candidat à une recherche Amazon à l'aide d'une méthode statistique
4. **scraping** : récupération de statistiques sur les recherches recommandées pour aider l'utilisateur à choisir les recherches pertinentes. L'étude de la concurrence se basera uniquement sur le choix de ces recherches.

Script 2 :

1. **scraping** : extraction de la donnée depuis l'ensemble des pages produits correspondant aux recherches Amazon choisies
2. **data preprocessing** : traitement de la donnée par stemming et suppression des mots parasites
3. **keywords similarities** : calcul d'un score de similarité entre mots à l'aide de la méthode d'intelligence artificielle Word2Vec

Le dernier point est crucial et le corpus pour réaliser de la similarité entre mots ne peut être constitué uniquement des pages concurrentes. Donc il est nécessaire de construire un modèle se basant sur de très nombreuses pages produits Amazon. Ce modèle sera évalué suivant trois métriques construites pour ce mémoire. De plus, il sera ré-entraîné à un intervalle de temps régulier pour être constitué uniquement de pages produits récemment scrapées sur Amazon. Cette pipeline sera hébergée sur une machine virtuelle disponible dans le cloud. Lors de l'exécution du script plusieurs étapes se succèdent pour obtenir plusieurs modèles. Le data scientist doit alors choisir à l'aide trois graphes le modèle qu'il souhaite utiliser pour l'ensemble des analyses. En effet ce modèle permettra d'effectuer du transfert learning. L'obtention de ce modèle suit l'ensemble des étapes décrites ci-dessous :

1. **scraping** : récupération de l'ensemble des urls des pages catégories depuis la page d'accueil d'Amazon.
2. **scraping** : extraction des pages produits présentes sur les pages catégories

3. **scraping** : récupération du contenu des pages produits si la page produit est non présente dans notre base de données ou que son dernier scraping a été effectué il y a plus de 30 jours.
4. **data preprocessing** : traitement de la donnée par stemming et suppression des mots parasites
5. **model training** : entraînement du modèle avec trois checkpoints implémentés pour permettre à un data scientist de choisir le meilleur modèle.

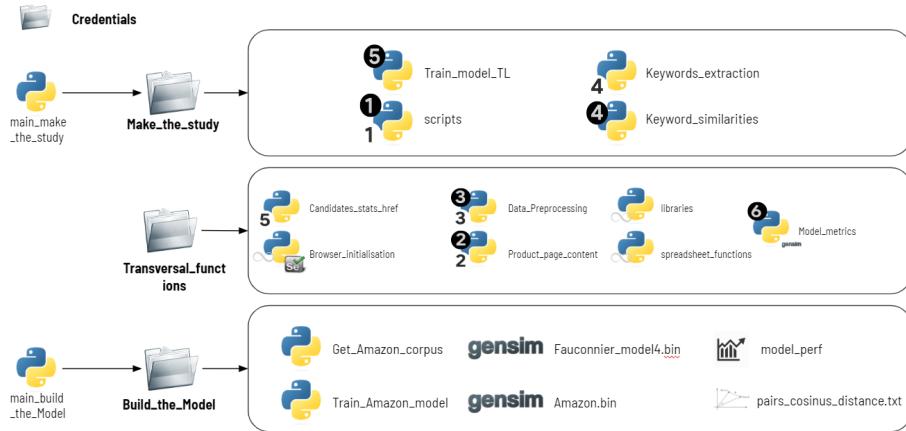


Figure 3: Architecture de la bibliothèque créée pour ce projet, les numéros détaillent l'ordre des étapes pour chaque partie de l'étude

3.2 Concurrence sur le marché

Des recherches sur internet ont permis d'identifier différents concurrents indirects et directs de la solution apportée. D'une façon générale le thème "Amazon product SEO" est très bien documenté sur google. On retrouve 51 millions de résultats pour cette recherche. Certains sites apportent des conseils pour mieux référencer un produit d'Amazon. Ils s'appuient sur le fonctionnement de l'algorithme de référencement utilisé par Amazon : A9. D'autres sites proposent des solutions complètes pour suivre des ventes, améliorer la logistique de livraison, suivre les tendances et améliorer le référencement d'un produit. Cette partie sera découpée en trois points, un premier présentant l'algorithme de classement d'Amazon, un deuxième mettant en avant les optimisations SEO, et enfin deux entreprises, proposant une offre complète en SEO, seront présentées. L'objectif sera de comprendre en quoi l'outil proposé est différent et apporte une approche plus personnalisée que ses concurrents.

Intéressons-nous rapidement au fonctionnement de l'algorithme A9 d'Amazon [15]. Cet algorithme a été conçu dans le but de répondre, en quelques microsecondes, à la requête d'un utilisateur parmi un catalogue regroupant des centaines de millions d'articles. Au contraire d'une recherche Google où l'on peut chercher des informations, une photo ou acheter, une recherche Amazon a pour but exclusif de rechercher un produit à acheter. Cet algorithme doit donc maximiser les ventes. L'algorithme évalue le potentiel de millions d'articles et leur donne un ordre d'apparition, suivant la probabilité d'être acheté par un utilisateur. L'algorithme se découpe en deux parties, un filtre est effectué par l'intermédiaire de la

recherche de l'utilisateur puis un ordre est appliqu     la aide de plusieurs m triques. La premi re  tage se pr occupe uniquement des mots clefs, un produit peut appara tre uniquement si celui-ci contient des mots clefs pertinents selon la recherche de l'utilisateur. La seconde  tage se base sur les performances de chaque produit : CTR (Click-Through-Rate) le nombre de cliques divis  par le nombre d'apparitions, le CR (Conversion Rate) le pourcentage d'achat apr s consultation de la page produit, les ventes, les avis, le taux de commandes dfectueuses (ODR).. Il est important de ne pas oublier qu'un produit aura plusieurs CR et CTR suivant la recherche de l'utilisateur. Enfin, les mots clefs repr sentent une part tr s importante du r f rencement sur Amazon et leur optimisation sera abord e dans le point suivant.

On retrouve de nombreux guides SEO pour les produits Amazon sur internet [15], [16]. Ils conseillent sur l'utilisation d'images, l'ergonomie de la fiche produit et l'ajout de mots clefs pertinents. Ce dernier point est central dans ce m moire. Les guides expliquent qu'il faut tout d'abord identifier les mots clefs pertinents concernant notre produit. L'outil *auto-complete* d'Amazon permet de voir les mots les plus proches d'un mot cherch  en barre de recherche. Cela permet d'identifier la liste de produits concurrents. De plus,   l'aide des commentaires sur les produits on peut s lectionner certains mots clefs. L'utilisateur de l'outil peut analyser les mots clefs des fiches produits concurrentes. Enfin, le vendeur peut ajouter des mots clefs en arri re-plan, ils ne seront pas visibles mais permettront   l'algorithme de mieux comprendre le produit. Ces recommandations apparaissent assez triviales mais la tâche est fastidieuse voire impossible   effectuer sur une liste de centaines de mots clefs pour une centaine de produits. Cette constatation am ne un vendeur   se tourner vers une solution compl te. Deux solutions seront abord es dans la partie suivante.

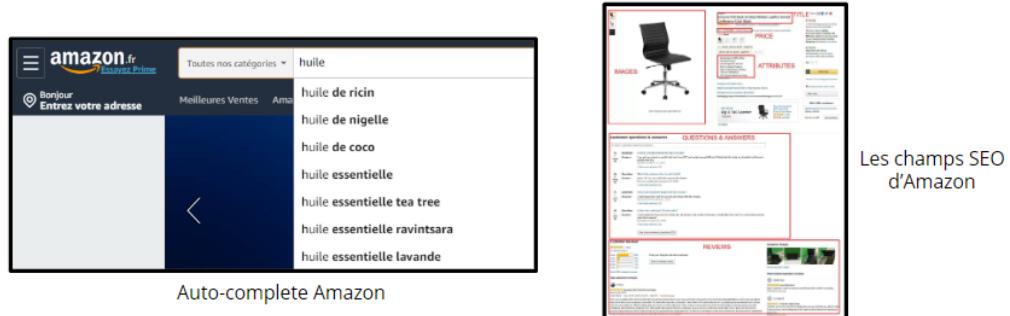


Figure 4: Illustration de l'outil de recommandations d'Amazon et des diff rents champs disponibles sur une page produit

La premi re solution est vendue par l'entreprise *Sellcics* [7]. Cette soci t  se pr sente comme le leader pour accompagner une marque, une agence ou un vendeur sur Amazon. Cette entreprise est implant e dans trois continents : Europe, Asie et Am rique du Nord. La solution propos e est vaste, elle permet d'acc der   un dashboard regroupant les ventes, les visites et commentaires clients. De plus, Sellcics propose une am lioration du r f rencement d'un produit gr ce   un outil de recherches de mots clefs nomm  *sonar-tool.com* disponible gratuitement. Cet outil propose des synonymes, des traductions, ou des mots proches d'un mot clef. Cet outil est bas  uniquement sur les recherches des utilisateurs d'Amazon et apporte la m trique "volume de recherches" pour chaque mot clef. Sellcics propose d'autres fonctionnalit s comme de l'A/B test automatis  pour  valuer les performances de deux visuels pour la

vente d'un même produit. Une autre fonctionnalité nommé *listing optimizer*, permet de comparer le contenu entre notre produit et des concurrents. L'outil offre des recommandations basées sur les différents champs de la page produit.

La deuxième solution est vendue par l'entreprise *JungleScout* [6]. Elle propose un outil similaire au précédent, le retailer peut retrouver ses ventes, suivre ses stocks, découvrir des opportunités basées sur les recherches et achats des usagers sur Amazon. Il permet aussi de consulter des avis, de répondre aux consommateurs et enfin d'optimiser le référencement à l'aide de mots clefs. Ce dernier est effectué à l'aide de l'outil nommé *keyword scout* et conseille sur l'achat de mots clefs. Enfin, les solutions présentées débutent aux prix de 57\$ pour Sellics et 39\$ pour JungleScout par mois.

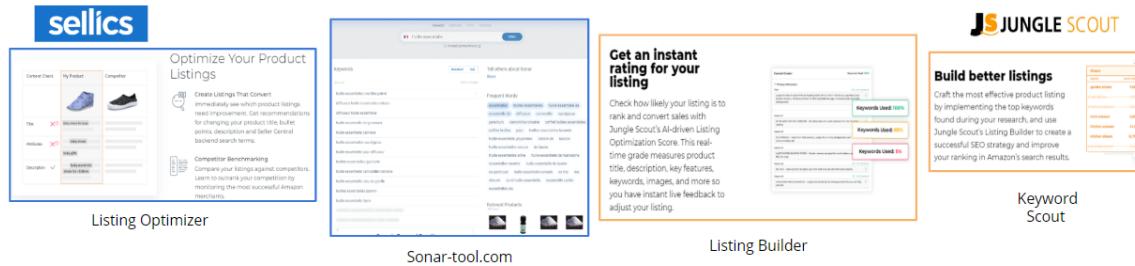


Figure 5: Présentation des outils pour les deux concurrents identifiés

En conclusion, on remarque que Sellics et JungleScout sont des concurrents directs de notre solution. Ils offrent deux outils pour l'amélioration du référencement d'un produit, par l'enrichissement de la fiche produit, avec des mots clefs pertinents. Cependant, cette recommandation est basée sur les recherches des clients et sur les produits concurrents, or peu de détails sont communiqués sur la détection des concurrents. Mais, la solution proposée par M13H semble prendre en compte une variété de produits plus importante et se base aussi sur l'avis des clients. Les deux outils concurrents ne mettent en avant aucune utilisation d'intelligence artificielle contrairement à la solution développée par M13H, cet aspect rendra certainement plus pertinente et variée la liste de mots clefs. De plus, notre outil offre une prédiction à l'aide d'une base de données peuplée de plusieurs milliers de pages produits récemment scrapées. Notre solution imite le comportement d'un humain et chaque donnée scrapée est fiable. Aucune approximation n'est effectuée, l'analyse se base sur le référencement réel sur Amazon au moment de l'analyse. L'utilisateur a la possibilité d'explorer l'ensemble de ses concurrents, et une synthèse lui est offerte pour comprendre quels sont les marques et produits les plus similaires et concurrentiels. Enfin, aujourd'hui aucun modèle Word2Vec, entraîné sur les fiches produits Amazon, n'est disponible en open source, notre volonté est donc de mettre à profit la puissance du scraping pour réaliser un modèle regroupant les pratiques de référencement les plus précises et récentes pour conseiller nos clients. L'analyse est totalement transparente, l'utilisateur a accès à l'entièreté des informations scrapées et utilisées pour améliorer le référencement de sa page produit. Cependant, ces deux concurrents ont mis en avant qu'il serait intéressant de scraper le contenu des suggestions Amazon lorsque l'utilisateur écrit une partie d'un mot. Notre outil de scraping le permet et il est alors possible d'utiliser les mots présents dans le titre de l'annonce. Ces recommandations seraient ajoutées à la suite des groupes de mots clefs recommandés à l'aide des méthodes statistiques. De plus, le nombre de recherches Amazon associé à une recherche est un indicateur pertinent qui pourrait être récupéré au moyen d'une API comme keywordtool.io/amazon. Mais cet outil est payant et sera peut-être inclus dans une prochaine version.

4 Extraction de la donnée

4.1 Extraction à l'aide de Selenium, Scrapy ou BeautifulSoup

Pour extraire la donnée sur Amazon, plusieurs solutions s'offrent au développeur. Il est possible de s'enregistrer auprès d'Amazon pour accéder à leur API. Cependant cela ne permet pas de récupérer le contenu de page produit, elle permet uniquement à une entreprise de suivre ses ventes et de modifier sa page produit. Une autre démarche consiste à passer par des bibliothèques comme *Beautifulsoup* ou *Selenium* ou *Scrapy*. Dans le cadre, de projets antérieurs j'ai pu utiliser *Beautifulsoup*, cette bibliothèque permet de récupérer le code (html) source d'une page. Ses fonctionnalités sont assez limitées et ne permettent pas d'exploiter les expressions *xpath* utiles pour récupérer une donnée spécifique sur une page. Les deux autres options sont abordées dans cette article [28]. L'auteur met clairement en avant les avantages de *Scrapy*, selon ses propos la méthode asynchrone facilite la récupération de la donnée, la bibliothèque est efficace, performante mais la documentation est difficile pour les débutants. Cependant, le choix s'est naturellement porté sur *Selenium*, suite à une mission réalisée pour M13H sur la récupération du data-layer (contient les trackers pour récupérer le comportement d'un utilisateur sur le web) d'un site web. Dans une démarche d'amélioration, il pourrait être intéressant de voir s'il est mieux de migrer vers *Scrapy*.

4.2 Méthode d'extraction retenue

Selenium possède une documentation [27] très facile à prendre en main. Un développeur peu familier du scraping peut facilement automatiser la récupération de la donnée à l'aide de cette bibliothèque. *Selenium* utilise un navigateur installé sur l'ordinateur de l'utilisateur. Deux modes sont possibles, la récupération de la donnée peut se faire en tâche de fond ou avec un navigateur ouvert et dirigé par une machine. *Selenium* permet de récupérer des champs spécifiques à l'aide de *XPath* (XML Path Language). Ce dernier est un langage pour effectuer des requêtes dans des données HTML, il permet de sélectionner des noeuds. Voici une démonstration assez simple pour récupérer de la donnée, ci-dessous, le but est de récupérer le prix de l'article. Dans un premier temps, l'utilisateur inspecte l'élément, à l'aide d'un clic-droit, puis d'un même clique sur la partie qu'il désire il sélectionne *XPath*. Enfin, le langage python permet de dialoguer avec *Selenium* et de récupérer la donnée.



Figure 6: Exemple d'extraction de données avec Selenium sur une page produit Amazon

5 Préparation de la donnée

La préparation de la donnée est une étape très importante en NLP. Il est en effet impossible de concevoir un modèle mathématique sans avoir une donnée normalisée. Dans le cadre de ce projet, les données sont des documents contenant du texte (fiches produits Amazon). Pour pouvoir manipuler facilement les données avec le langage python, on utilise les dataframes et le texte est séparé en phrases. Une fois cette étape effectuée, il est alors possible d'exploiter les données pour retirer les caractères parasites et mots inutiles. Enfin, les mots seront modifiés pour pouvoir être mieux comparés entre eux. Cette section s'inspire des informations contenues dans cette article : [20].

Pour imager les différentes manipulations de données décrites dans la suite de ce document, le texte suivant sera utilisé au titre de corpus [36] :

Comment l'intelligence artificielle va changer nos vies ? À l'occasion du sommet sur l'intelligence artificielle qui se tient le jeudi 29 mars au Collège de France et à l'issue duquel le président de la République doit présenter la stratégie nationale dans ce domaine, (re)découvrez tous nos contenus sur cette technologie qui pourrait à terme « augmenter » presque chaque moment de notre vie. Reconnaissance faciale, assistant intelligent, voiture autonome, où en est l'intelligence artificielle ? Laisserons-nous ces systèmes interpréter nos scanners médicaux ou rendre la justice ? L'automatisation du travail touchera-t-elle tout le monde ? Comment éviter les dérives éthiques ?

```
1 def corpus_to_sentences(data) :
2     data = data.replace('!', '.').replace('?', '.').split('.')
3     return data
```

5.1 Retrait des mots et caractères parasites

Pour chaque phrase, il est possible de retirer la ponctuation. Pour se faire, on utilise communément la bibliothèque *string.punctuation*. Ensuite, on peut enlever les accents par l'intermédiaire de la bibliothèque *unicode*.

```
1 def remove_punctuation_and_lowering(data) :
2     for i in range(len(data)) :
3         data[i] = data[i].translate(str.maketrans(' ', ' ', '!#$%&\()*+,.:/;<=>@[\\\]^_`{|}~'))
4         #data[i] = unidecode.unidecode(data[i])
5     return data
```

Enfin, une liste de mots parasites peut être constituée pour enlever les mots inutiles. Dans la littérature [20], il est souvent conseillé d'utiliser la bibliothèque *nltk.corpus*. Cette bibliothèque propose de nombreuses fonctions pour traiter automatiquement les données textuelles. La liste de mots contenue dans *stopword* permet de constituer une base. Néanmoins, il est important d'avoir à l'esprit que cette liste va être enrichie lors de la préparation de la donnée et devra être maintenue à jour pour permettre

de conserver les performances de l'algorithme. Par exemple, depuis 2018 et la mise en place de la Réglementation Générale sur la Protection des Données (RGPD), on retrouve une barre de consentement sur la plupart des sites. Ces informations peuvent s'avérer inutiles et gênantes dans le cadre du calcul d'une métrique de similitude entre deux sites.

```

1 def remove_stopwords(data, list_stopwords = list_stopwords) :
2     for i in range(len(data)) :
3         remove_word = re.findall(list_stopwords, data[i])
4         for word in remove_word :
5             data[i] = re.sub(r"\{}".format(word), ' ', data[i])
6         if len(re.findall(list_stopwords, data[i])) > 0 :
7             data[i] = stopwords_list(data[i])
8     return data

```

5.2 Stemming et lemmatisation

Ensuite, une méthode de préparation de la donnée propose de normaliser les mots en créant des mots plus courts. Cette méthode permet de réduire le nombre de mots différents. Cela permet de mieux les comparer et de trouver des similitudes de façon plus facile. Deux procédés existent et génèrent des mots différents, on retrouve une description complète depuis cette article [17]. Ces procédés utilisent tous deux la bibliothèque *NLTK*.

Le stemming est un procédé permettant de récupérer le radical d'un mot. Le mot alors obtenu peut définir un vrai mot ou un mot inexistant. En effet, cette fonction retire le suffixe ou le préfixe d'un mot. Dans le cadre de ce projet, des données textuelles en français seront utilisées. La bibliothèque *NLTK.stem.snowball* fournit des classes permettant d'effectuer ce travail. En effet, des règles simples sont implémentées comme par exemple : retirer les "s" à la fin des mots ou les terminaisons que l'on trouve dans les conjugaisons des mots. On retrouve la documentation complète en suivant ce lien : [29].

```

1 def stemming(data) :
2     from nltk.stem.snowball import FrenchStemmer
3     stemmer = FrenchStemmer()
4     stemmed = []
5     for i in range(len(data)) :
6         data[i] = data[i].split(" ")
7         for word in data[i] :
8             if len(word) > 0 :
9                 stemmed.append(stemmer.stem(word))
10    stemmed = " ".join(stemmed)
11    return stemmed

```

Un second procédé existe et est nommé lemmatisation. Il permet de transformer le mot en un mot existant. Il retourne la forme canonique d'un mot. Par exemple pour un verbe, l'algorithme retournera son infinitif, pour un nom féminin pluriel on obtiendra le masculin singulier. Cette algorithme est assez lourd et demande la création d'un dictionnaire de mots pour faire correspondre le mot et son lemme. Cependant, la bibliothèque *NLTK* ne propose pas de version française pour réaliser la lemmatisation. Mais, une

autre source [30] propose d'utiliser la bibliothèque *spacy*. C'est un modèle statistique se basant sur le corpus des pages wikipedia. La précision est cependant montrée comme moins bonne que pour le stemming. Cette méthode ne sera donc pas nécessairement utilisée dans ce projet. En effet, la stemmatisation est moins sensible aux fautes d'orthographies que la lemmatisation : la stemmatisation peut réussir si la faute ne perturbe pas la détection du radical et si celui-ci n'est pas modifié. La stemmatisation n'a pas besoin du contexte pour fonctionner. Elle ne requiert que le mot à traiter et la langue dans laquelle ce mot est écrit.

```

1 def all_(data , list_stopwords) :
2     data = corpus_to_sentences(data)
3     data = remove_punctuation_and_lowering(data)
4     print("\n no_punctuation : \n", "\n".join(data))
5     data = remove_stopwords(data)
6     data = remove_stopwords(data)
7     print("\n remove_stopwords : \n", "\n".join(data))
8     data = stemming(data)
9     print("\n stemmed : \n", data)
10    return print(done)
11 all_(text , list_stopwords)

```

| Original_text | No_punctuation | Remove_stopwords | Stemmed |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Comment l'intelligence artificielle va changer nos vies ? | comment l'intelligence artificielle va changer nos vies | comment intelligence artificielle va changer vies | comment intelligent artificiel va chang vi |
| À l'occasion du sommet sur l'intelligence artificielle qui se tient le jeudi 29 mars au Collège de France et à l'issue duquel le président de la République doit présenter la stratégie nationale dans ce domaine, (re)découvrez tous nos contenus sur cette technologie qui pourrait à terme « augmenter » presque chaque moment de notre vie. | à l'occasion du sommet sur l'intelligence artificielle qui se tient le jeudi 29 mars au collège de france et à l'issue duquel le président de la république doit présenter la stratégie nationale dans ce domaine redécouvrez tous nos contenus sur cette technologie qui pourrait à terme « augmenter » presque chaque moment de notre vie | occas sommet intelligence artificielle tient jeudi 29 mars college france issue duquel president republique doit presenter strategie nationale domaine redecouvr contenus cette technologie pourraut terme augmenter presque chaque moment vie | occas sommet intelligent artificiel tient jeud 29 mar colleg franc issu duquel president republ doit present strateg national domain redecovr contenus cet technolog pourr term augment presqu chaqu moment vi |
| Reconnaissance faciale, assistant intelligent, voiture autonome, où en est l'intelligence artificielle ? | reconnaissance faciale assistant intelligent voiture autonome où en est l'intelligence artificielle | reconnaissance faciale assistant intelligent voiture autonome intelligence artificielle | reconnaisss facial assist intelligent voitur autonom intelligent artificiel |
| Laisserons-nous ces systèmes interpréter nos scanners médicaux ou rendre la justice ? | laisserons-nous ces systèmes interpréter nos scanners médicaux ou rendre la justice | laisserons systemes interpreter scanners medicaux rendre justice | laiss system interpret scanner medical rendr justic automatis travai touch mond |
| L'automatisation du travail touchera-t-elle tout le monde ? | l'automatisation du travail touchera-t-elle tout le monde | automatisation travail touchera monde | comment évit dériv éthiqu |
| Comment éviter les dérives éthiques ? | comment éviter les dérives éthiques | comment éviter dérives éthiques | |

Figure 7: Évolution du corpus lors des différentes étapes de data preprocessing

De façon générale, il est important de faire attention à l'ordre dans lequel ces différentes étapes sont effectuées. En effet, la stemmatisation utilise des règles reposant sur des mots accentués pour déduire le radical. Donc il sera important d'orchestrer ces méthodes de façon intelligente. Pour se faire, la littérature scientifique et l'observation empirique seront utilisées.

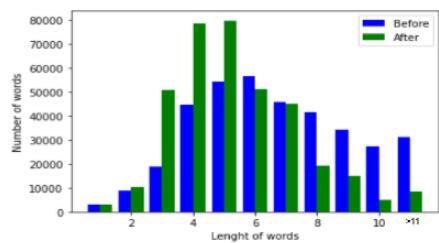
Dans le cadre de ce mémoire, la pertinence de du data preprocessing a été évaluée à l'aide d'un corpus résultant d'une analyse sur le produit : Clean&Steam de Rowenta vendu sur Amazon. Cette donnée a été récupérée suite à l'exécution de l'outil. Le scraping a permis de récupérer le contenu de 403 pages produits. Le tableau ci-dessous montre le nombre de caractères présents avant et après nettoyage de la donnée. Une fois la donnée nettoyée, il est possible d'parler en nombre de mots et non de caractères. De plus, les effets du stemming sont mis en évidence par ces deux graphes. En effet, le nombre de mots ayant une taille élevé diminue et la fréquence des mots augmente.

Cleaning

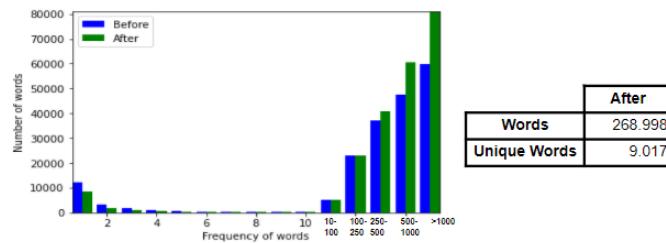
| | Avant | Après |
|--------------------|---------|---------|
| Caractères | 481.863 | 268.998 |
| Caractères uniques | 36.894 | 15.468 |

The number of words and unique words vanished

Stemming



The proportion of words with a large length decreases



The frequency of each word increases

Figure 8: Évolution du nombre de caractères avant et après data preprocessing et évolution de la distribution de la longueur des mots et de la fréquence avant et après stemming

6 Extraction de mot-clefs : méthodes statistiques

Dans cette partie, l'objectif est de présenter des métriques différentes pour déterminer des mots clefs dans un texte. Trois méthodes statistiques distinctes sont présentées ci-dessous. Elles permettent d'extraire des mots ou des groupes de mots caractérisant au mieux le texte. D'une façon générale, plus le score de la métrique sera élevé plus le mot clef sera considéré comme important.

6.1 Différents outils statistiques

6.1.1 TF-IDF

Cette explication utilise les notions présentées dans ce document: [31].

Le TF-IDF est communément appelé "term frequency-inverse document frequency". Comme présenté précédemment, il est ingénieux de retirer les mots inutiles (stopwords) et de stemmatiser le texte. Ces deux points permettent de réduire le nombre de mots et de pouvoir réaliser une analyse plus robuste. Le calcul du TF-IDF repose sur un calcul statistique utilisant uniquement des formules mathématiques simples.

Avant toutes choses, il faut définir certains termes et poser les variables mathématiques. Le *corpus* est constitué de l'ensemble des *documents* du texte. Un document est donc une phrase. Le *corpus* est constitué de m mots distincts et de n documents. Chaque document est composé de k mots $[k^{(1)}, k^{(n)}]$.

Le calcul de l'indicateur se déroule ainsi, en trois étapes :

- Tout d'abord, il faut créer une liste de tous les mots constituant notre corpus. L'utilisateur est amené à construire une matrice : M^{n*m} , m nombre de mots et n nombre de documents. On note ainsi $i \in [1, n]$ et $j \in [1, m]$. Chaque composante de la matrice est notée x_{ij} avec i un document et j un mot.
- Ensuite, la première formule consiste à calculer la fréquence des termes ("term frequency"). Dans chaque document, il faut compter le nombre de fois où apparaît chaque mot. Puis diviser ce nombre par le nombre de mots du document.

$$TF_{ij} = \sum_{l=1}^{k^{(i)}} \frac{1_{j=l}}{k^{(i)}}$$

- Puis, la deuxième formule calcul l>IDF c'est à dire le nombre de documents divisé par le nombre de documents contenant le mot. Cette formule utilise le logarithme dans le but d'atténuer l'effet du ratio.

$$IDF_{ij} = \log \frac{n}{\sum_{l=1}^n 1_{l=j}}$$

- Enfin, le résultat final est donné par le produit des deux formules précédentes.

$$x_{ij} = TF_{ij} * IDF_{ij} = \sum_{l=1}^{k^{(i)}} \frac{1_{j=l}}{k^{(i)}} * \log \frac{n}{\sum_{l=1}^n 1_{l=j}}$$

De façon synthétique, le TF-IDF permet d'évaluer la fréquence d'un mot dans un texte. Il pénalise les mots dont la fréquence est élevée à la fois dans un corpus et un document. Au contraire, un mot présent de nombreuses de fois dans peu de documents aura un poids plus important.

Exemple : Reprenons le corpus précédent en partant du texte stemmatisé. On compte 53 mots uniques et 6 phrases. Je construis quatre matrices TF, IDF, TF*IDF et SUM de taille 6*53 pour les trois premières et 1*53 pour la dernière. Le code est à retrouver en **Annexe 1**. Les mots les plus importants de ce corpus sont alors ceux ayant le plus haut score.

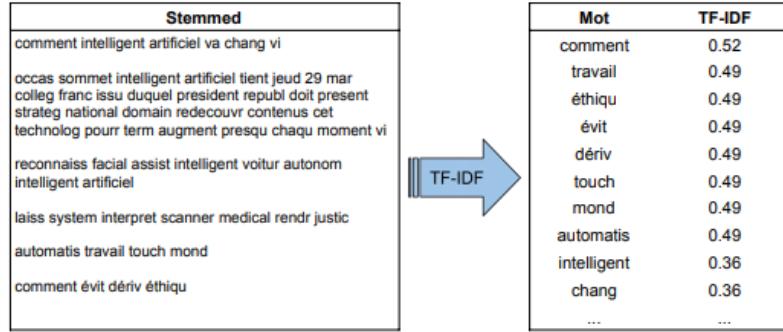


Figure 9: Illustration de l'entrée et de la sortie fournies par l'algorithme TF-IDF

6.1.2 Rapid Automatic Keyword Extraction : Rake

Cette explication reprend de façon succincte ces deux articles : [33] et [5]. La méthode RAKE permet de constituer des mots clefs composés d'au moins deux mots. Au contraire du TF-IDF, la ponctuation et les stopwords permettent de séparer les éléments d'une phrase et de constituer des potentiels groupes de mots clefs. La détermination de ces mots clefs se fait à l'aide d'une matrice de co-occurrence . Elle met en avant deux éléments, le nombre de fois où un mot apparaît dans un même mot-clef qu'un autre et le nombre de fois où le mot-clef est présent. On calcule donc deux métriques à partir de cette matrice :

- la fréquence d'un mot : $freq(w)$, le nombre de fois où un mot apparaît
- le degré d'un mot : $deg(w)$, le nombre de fois où le mot apparaît plus le nombre fois où il apparaît avec un autre mot

Enfin, on utilise ces métriques pour obtenir notre indicateur : $\frac{deg(w)}{freq(w)}$

Exemple : La bibliothèque `rake_nltk` permet de lister les mots les plus fréquents. Dans notre exemple, je souhaite obtenir des mots-clefs constitués de 2 mots. Rake utilise une liste de stopwords française, il est important de l'enrichir, si les résultats obtenus contiennent des mots parasites. Cette liste sera donc constamment complétée. Les mots dans cet exemple n'ont pas été stemmatisés, il sera important de comparer dans la phase expérimentale des corpus stemmatisés ou non. Ci-dessous, voici le code et les résultats.

```

1 from rake_nltk import Rake
2 r = Rake()
3 stopwords = stopwords
4 language="french",
5 min_length=2,
6 max_length=2 )
7 r.extract_keywords_from_text(text.replace("<< ","").replace(" >>",""))
8 r.get_ranked_phrases()

```

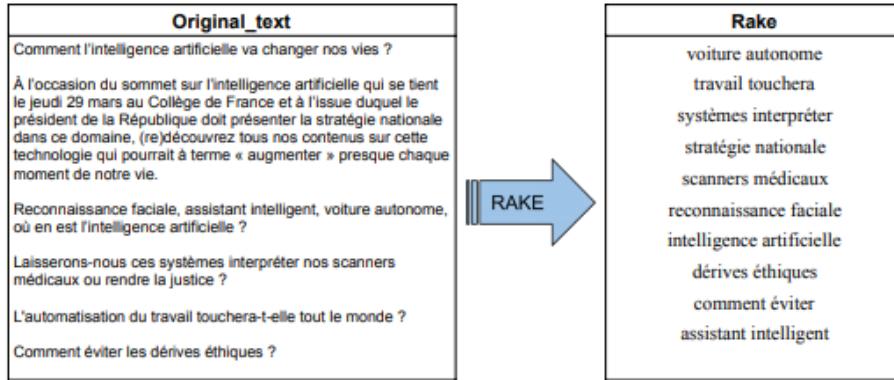


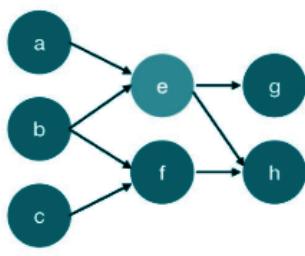
Figure 10: Illustration de l'entrée et la sortie fournies par l'algorithme RAKE

6.1.3 TextRank

Cette partie reprend l'explication de cet article [24]. TextRank est un algorithme basé sur PageRank, la différence principale réside dans le fait que PageRank classe des sites internets et TextRank des mots dans un texte. Tout d'abord il faut comprendre ce que fait PageRank. Il est utilisé pour calculer la valeur (le poids) d'une page internet. Plus cette métrique est élevée, plus la page est importante et de qualité. Les moteurs de recherche comme Google utilisent cette métrique pour référencer un site. Pour comprendre cette métrique, il suffit de représenter un graphe où chaque noeud V_i est un site web et les connections entre deux sites sont des liens présents sur les pages. Des liens peuvent être entrants $In(V_i)$ ou sortants $Out(V_i)$. Si le site ne possède aucun lien sortant alors sa valeur est égale au facteur $d = 0.85$. La valeur d'un noeud suit cette formule:

$$S(V_i) = (1 - d) + d * \sum_{j \in In(V_i)} \frac{S(V_j)}{\#Out(V_j)}$$

Ce calcul se fait à l'aide d'un algorithme avec une initialisation des valeurs à 1 pour chaque noeud. Après plusieurs itérations, une fois que la valeur des noeuds est stable l'algorithme s'arrête.



Calculons la valeur des noeuds avec d=0.85 :
 ➤ Pour e :

$$In(v_e) = \{a, b\}, j \in \{a, b\}$$

$$S(V_e) = (1 - d) + d * \left(S(V_a) + \frac{1}{2} S(V_b) \right)$$

➤ On construit une matrice

$$\frac{S(V_j))}{\#Out(V_j)} = \begin{bmatrix} a & b & c & e & f & g & h \\ a & 0 & 0 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 0 & 0 \\ e & 1 & 0.5 & 0 & 0 & 0 & 0 \\ f & 0 & 0.5 & 1 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0.5 & 0 & 0 \\ h & 0 & 0 & 0 & 0.5 & 1 & 0 \\ \end{bmatrix}$$

➤ Calcul itératif des valeurs

$$S(V)_0 = [1, 1, 1, 1, 1, 1, 1]$$

$$S(V)_1 = [0.15, 0.15, 0.15, 1.42, 1.42, 0.57, 1.42]$$

$$S(V)_2 = [0.15, 0.15, 0.15, 0.34, 0.34, 0.29, 0.58]$$

$$S(V)_3 = [0.15, 0.15, 0.15, 0.34, 0.34, 0.29, 0.58]$$

$S(V)_2 = S(V)_3$ on arrête.

$$S(V_e) = 0.34$$

Figure 11: Explication mathématique du calcul de TextRank avec un exemple à 7 noeuds

TextRank analyse les mots d'un corpus. Le corpus est divisé en phrase et compte n mots distincts. Communément, on peut utiliser la bibliothèque *spacy* pour sélectionner les natures de mots les plus pertinentes et retirer les stopwords. Pour chaque phrase, on prend une fenêtre k et chaque mot de la fenêtre possède une connexion. Le graphe ainsi construit permet de calculer la valeur de chaque mot. Dans ce mémoire l'algorithme présenté par BrambleXu [3] sera utilisé et adapté à la langue française.

6.2 Présentation de la méthode retenue

Cette partie présente l'ensemble des traitements et méthodes utilisées pour obtenir une liste de groupe de mots pertinente pour effectuer des recherches Amazon à partir de la fiche produit d'un client uniquement. Je pars de cette hypothèse pour construire notre méthode: l'étude de la fiche produit permettra de dégager des recherches pertinentes pour le moteur de recherche d'Amazon.

Tout d'abord, la fiche produit du client est scrapée sur le site d'Amazon, les données sont réparties dans différents champs. Il est alors possible de créer 18 colonnes décrivant l'url, le titre de l'annonce, ASIN (identifiant Amazon du produit), le nom de la marque, la note attribuée par les clients au produit, le nombre d'avis client, le prix, le texte de la description, les caractéristiques sous forme de tirets, la catégorie, les questions et réponses, les avis des clients, le vendeur et l'expéditeur du produit. Ensuite, la donnée textuelle subit plusieurs transformations : les majuscules sont retirées, la ponctuation est remplacée par des points, les emojis fréquents dans les avis clients sont retirés. Enfin, le corpus est divisé en phrases et chaque mot est stemmatisé. La donnée nettoyée peut être enfin exploitée. Pour rappel l'algorithme RAKE utilise une liste de stopwords pour retirer des mots parasites et constituer des candidats qu'il évalue. Le score attribué ainsi que les candidats proposés sont apparus très décevants avec la bibliothèque *rake_nltk*. Cette méthode n'a donc pas été retenue, néanmoins, l'idée de candidats a été conservée pour réaliser des recherches Amazon. Une fonction nommé "candidat" a été implémentée, celle-ci permet de créer des groupes de mots constitués d'un nombre minimal et maximal de mots précisés en paramètre. Les phrases sont alors découpées en plusieurs groupes. Une fois les candidats définis on les met de coté et on s'occupe de la méthode de scoring pour obtenir des mots clefs pertinents.

L'hypothèse précédente ne permet pas d'effectuer une analyse statistique sur des groupes de mots. En effet, il est commun en NLP d'effectuer un TF-IDF sur des groupes de mots (BOW : Bag Of Words). Dans notre cas, cela est impossible car le nombre de mots présents dans la fiche produit est beaucoup trop faible et des groupes de mots ne se répètent presque jamais. Il a donc été décidé de se concentrer sur tous les mots de la donnée nettoyée. Les méthodes TF-IDF et TextRank ont été implémentées et chaque mot se retrouve avec deux scores.

Pour attribuer un score à chaque candidat précédemment défini j'utilise les scores de chaque mot et effectuons une moyenne de chaque méthode. Enfin, je retire les groupes de mots contenant les mêmes mots mais dans un ordre différent. Ainsi à partir d'une fiche produit, un traitement du corpus a été effectué pour exploiter le texte, des candidats à des recherches Amazon ont été déterminés et enfin un score a été attribué pour chaque candidat. En outre, l'utilisateur peut ajouter d'autres mots clefs conseillés par le client et filtrer les candidats non pertinents. De plus, à la suite de l'étude sur la concurrence effectuée en amont, il a été décidé d'ajouter les suggestions d'Amazon à partir de l'écriture de mots dans la barre de recherches Amazon. Les 5 premières suggestions relatives aux 5 mots les plus pertinents du titre au sens du TF-IDF ont été récupérées par scraping. On remarque que cette méthode débouche sur une étape manuelle . Voici une illustration reprenant chaque étape décrite ci-dessus :

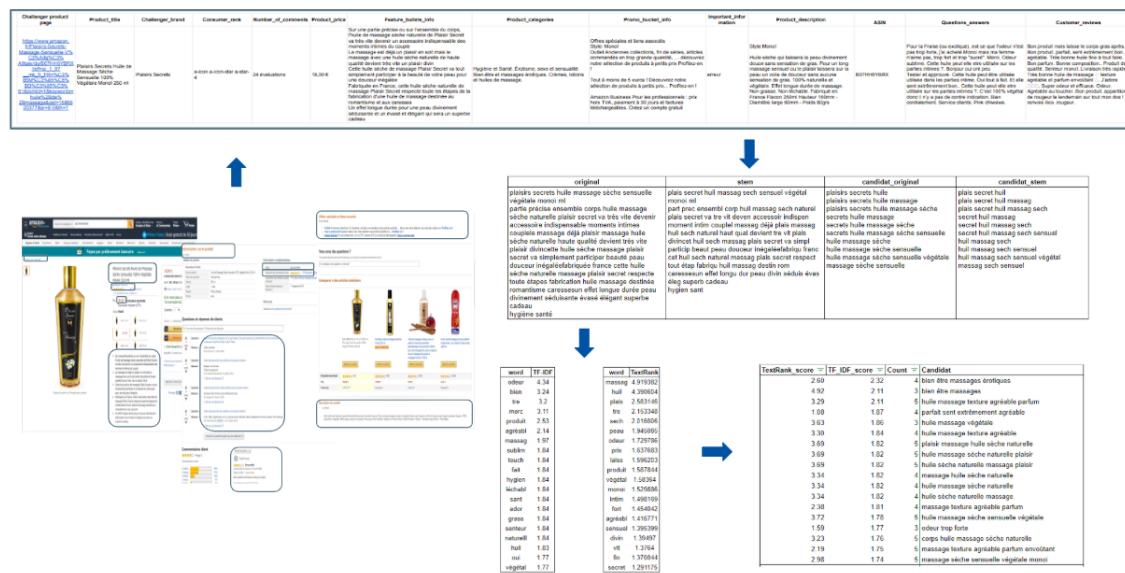


Figure 12: Détail des différentes étapes pour détecter les recherches Amazon pertinentes pour l'étude concurrentielle

7 Expliquer le référencement des produits par l'algorithme d'Amazon

Après avoir scrapé l'ensemble des pages produits, on aboutit à une liste de plusieurs milliers de produits. Le contenu de chaque produit est alors récupéré automatiquement et on obtient un certain nombre d'informations. L'objectif est d'exploiter toutes ces variables et de créer d'autres variables pertinentes pour réaliser plusieurs modèles. Ces modèles auront pour but d'expliquer quelles variables sont significatives pour l'algorithme de référencement d'Amazon. On souhaite prédire si une page produit sera dans le Top 10 pour une recherche Amazon. Cette partie sera découpée en deux. D'une part l'aspect théorique des différentes méthodes employées sera présenté rapidement. D'autre part, une phase expérimentale détaillera les variables choisies et les résultats obtenus.

7.1 Méthodes utilisées

D'une part, un modèle de régression logistique va être utilisé à l'aide de la bibliothèque de `sklearn`. La régression logistique est utilisée pour effectuer une classification. On note y la cible et x_k les variables explicatives ($k \in [1, p]$). En partant de la formule de la régression linéaire, on obtient la probabilité d'avoir la classe 1 pour une régression logistique :

- Régression linéaire : $\hat{y}^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)}$
- Régression logistique : $\sigma(\eta) = \frac{1}{1+exp(-\eta)}$
- $P(y^{(i)} = 1) = \frac{1}{1+exp(-(\beta_0+\beta_1x_1^{(i)})+\dots+\beta_px_p^{(i)}))}$

Enfin, il faut calculer la fonction de coût à optimiser.

$$\begin{aligned} Loss &= \prod_{i=1}^n P(Y = 1 | X = x_i \beta) \\ Loss &= \prod_{i=1}^n \sigma(x_i \beta)^{y_i} * (1 - \sigma(x_i \beta))^{1-y_i} \\ Log(Loss) &= \sum_{i=1}^n y_i * log(\sigma(x_i \beta)) + (1 - y_i) * log((1 - \sigma(x_i \beta))) \end{aligned}$$

On applique la dérivée et obtient :

$$\nabla Log(Loss) = \frac{1}{n} (Y - \sigma(X \beta)) X$$

On applique la descente de gradient pour obtenir le minimum (α_t : stepsize):

$$w_{t+1} = w_t + \alpha_t * \nabla f(w_t)$$

Si on s'intéresse à la bibliothèque de `sklearn`, on utilise la fonction `LogisticRegression`. Cette fonction possède plusieurs paramètres qui pourront être testés : des méthodes pour régulariser (l1,l2) et le paramètre `class_weight` utile lorsque un dataset comporte plus d'éléments d'une classe par rapport à l'autre. Nous verrons que le dataset utilisé dans ce point a une distribution non-équivalente. Il faut alors spécifier "balanced" pour le paramètre pour éviter que le modèle ne prédisse une unique classe.

```

1 from sklearn.linear_model import LogisticRegression
2 y = df1["Top_10_Position"].cat.codes
3 x = df1.select_dtypes(np.number)
4 X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.10, random_state
    =42)
5 clf = LogisticRegression(penalty='none', random_state=3, class_weight = 'balanced').fit(
    X_train, y_train)
6 clf.score(X_train, y_train), clf.score(X_test, y_test)

```

D'autre part, un modèle de random forest va être utilisé à l'aide de la bibliothèque de `sklearn`. Ce modèle implique le calcul de nombreux arbres de décision. Chaque arbre attribut une classe suivant les caractéristiques de la donnée. Random forest prend alors la classe majoritaire sur l'ensemble des arbres calculés. Enfin, pour la bibliothèque de `sklearn`, on utilise la fonction `RandomForestClassifier`. Cette fonction possède de nombreux paramètres dont la combinaison pourra être testée à l'aide de l'algorithme `GridSearch`.

```

1 from sklearn.ensemble import RandomForestRegressor
2 clf = RandomForestClassifier(verbose=1, max_depth=2, random_state=0, class_weight =
    'balanced').fit(X_train, y_train)
3 clf.score(X_train, y_train), clf.score(X_test, y_test)

```

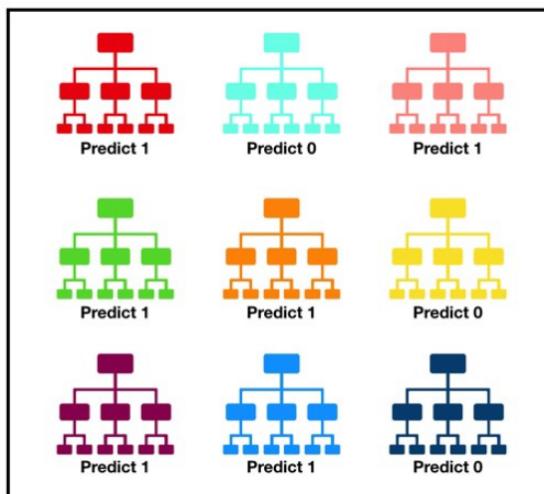


Figure 13: Random Forest : la labélisation obtenue est 1 (6 arbres prédisent 1 et 3 attribuent 0)

De plus, avant de passer à la phase expérimentale, il reste à définir la méthode `GridSearch` et les métriques utilisées pour évaluer nos modèles.

Tout d'abord, `GridSearch` est un algorithme couramment employé pour tester différentes combinaisons

de paramètres pour un modèle de machine learning. La meilleure combinaison est retenue et donne le modèle. La bibliothèque sklearn fournit une fonction nommée *GridSearchCV*. Cette fonction sera utilisée pour déterminer le meilleur modèle calculé par Random Forest. Il est alors possible de spécifier différents paramètres comme le nombre d'arbres de décision, le nombre de caractéristiques à prendre en compte à chaque noeud, la profondeur maximale de l'arbre, le nombre minimum de données pour créer un nouvelle feuille, le nombre minimum de données pour chaque feuille ou encore utiliser la méthode bootstrap.

Ensuite, pour évaluer les modèles, la métrique de précision sera utilisée. De plus, il est possible d'afficher la courbe ROC, l'aire sous la courbe indique la proportion de prédictions Fausses-Positives en fonction des Vraies-Positives. Ces deux métriques sont obtenues à partir de la matrice de confusion.

Enfin, une dernière méthode nommée *Information Value* décrite dans cette article [21], permet de déterminer si les variables utilisées dans le modèle sont significatives. En effet, cette méthode permet de sélectionner des variables. Il construit 10 groupes, pour chaque variable, il calcule le nombre de données pour chaque classe (*non-events* et *events*). Puis, applique la formule suivante en sommant sur les classes et variables.

$$\sum (Event\% - NonEvent\%) * \ln\left(\frac{Event\%}{NonEvent\%}\right)$$

Le tableau ci-dessous décrit l'interprétation des résultats.

| Information Value (IV) | Predictive Power |
|------------------------|-----------------------------------|
| < 0.02 | useless for prediction |
| 0.02 to 0.1 | weak predictor |
| 0.1 to 0.3 | medium predictor |
| 0.3 to 0.5 | strong predictor |
| > 0.5 | suspicious or too good to be true |

Figure 14: Information Value : interprétation des résultats

7.2 Expérimentations

7.2.1 Étude statistique préliminaire

Avant d'aborder la création de modèles, il est important de choisir rigoureusement les variables constituant le modèle. Après avoir généré plusieurs études, nous avons accès à 6505 pages produits scrappées sur la première page pour une recherche d'un utilisateur. (Pour rappel une liste de pages produits pour une recherche comporte au maximum 16 ou 56 produits. Le rendu sous 16 ou 56 est aléatoire.) Ces pages produits regroupent différentes informations : le texte rangé par champs d'une page produit (Titre, Description, Information en points (FBI), catégorie, questions et réponses, commentaires clients), ainsi que le nombre de commentaires, le classement du produit par les clients, le prix, l'expéditeur, la marque, le vendeur, le stock et la position du produit pour une recherche client donnée. Pour rappel, l'objectif est de prédire si une page produit sera dans le Top 10 pour une recherche Amazon effectuée par un utilisateur. Avant toute chose, j'ai créé une liste de variables en deux catégories : une compare la part des mots contenus dans la recherche de l'utilisateur (keywords) et chacun des champs, la seconde s'intéresse au méta-data de la page produit : la note du produit, le nombre de commentaires, le stock.. Cette expérimentation, résumée dans la photo ci-dessous, décrit la distribution des variables et oppose les statistiques présentes sur la première page et celle du Top 10.

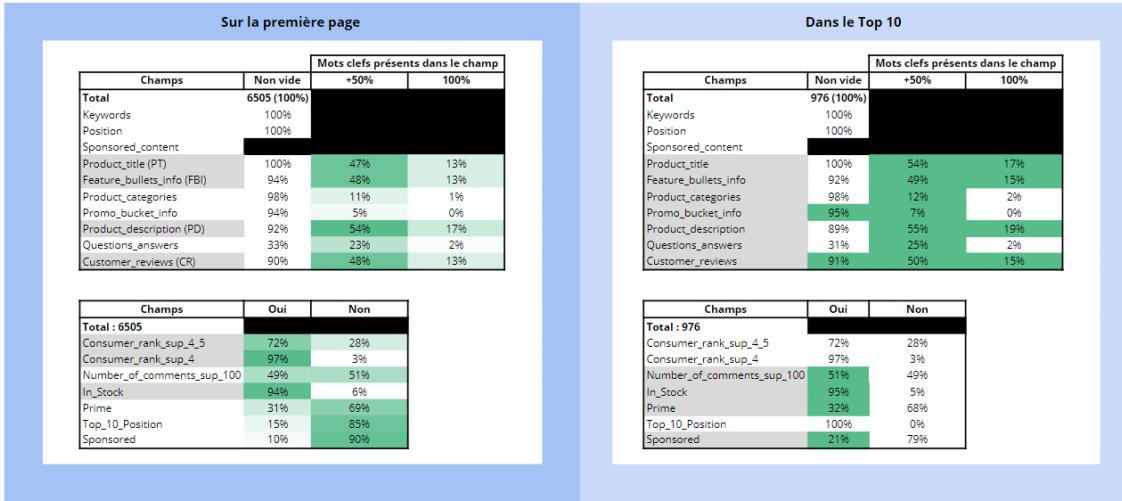


Figure 15: Distribution de l'ensemble des métriques et part des mots de la recherche présents dans chaque champ. Évolution des variables entre le Top 10 et les produits de la première page

Le tableau de gauche se concentre sur l'ensemble des produits présents sur la première page pour une recherche donnée. L'échantillon étudié fait 6505 produits tous scrappés à l'aide d'une recherche utilisateur et dont le titre et la position sont renseignés. Pour ces différents produits, on remarque que tous les champs ne sont pas systématiquement renseignés : "questions et réponses" est absent dans 67% des cas. De plus, l'objectif a été de mesurer la part de mots contenus dans la recherche de l'utilisateur et dans les différents champs. Cette deuxième observation permet de voir que la description, le titre, les avis des consommateurs et les informations en points (FBI) regroupent le plus souvent plus de 50% des mots. Le second tableau sur la gauche, compare la distribution des métadata. Sur la première page, les produits sont majoritairement en stock (94%), les produits possèdent une note supérieure à 4/5 et un tiers des produits visibles sur la première page sont éligibles à Amazon Prime (expédiés et vendus par Amazon). Ces observations permettent de montrer que certaines variables sont primordiales pour être sur la première page. En effet, l'algorithme d'Amazon privilie, sur la première page, les produits en stock, ayant une note supérieure à 4 et éligibles à Amazon Prime. Au contraire, le seuil de 100 commentaires clients n'est pas pertinent car sa distribution est équivalente (49% - 51%). De plus, d'un point de vue sémantique, il semble que le contenu du champ "Promo Bucket Info" ne soit pas significatif par rapport à la recherche de l'utilisateur (seul 5% des pages produits possède 50% des mots de la recherche dans ce champ), au contraire, la description, le titre, les avis des consommateurs et les informations en points (FBI) semblent pertinents pour un bon référencement.

La seconde partie de l'illustration, sur la gauche, montre l'évolution de l'ensemble des variables lorsque l'on se focalise sur le top 10 des produits pour des recherches utilisateurs. On remarque d'un coup d'œil que l'ensemble des valeurs, en sur-brillance vert, sont plus élevées que sur le tableau de gauche. Cette observation prouve que l'algorithme A9 d'Amazon favorise les annonces dont le contenu sémantique concorde avec la recherche de l'utilisateur. Cette preuve souligne l'intérêt d'avoir une démarche SEO et de développer notre outil. Enfin, le tableau en bas à droite montre une faible augmentation (en vert) de

certaines valeurs. Cela corrobore l'observation précédente : l'algorithme d'Amazon porte plus d'intérêt à la sémantique d'une annonce pour différencier des annonces dans le Top 10 et celles sur la première page. En conclusion, une annonce placée sur la première page de résultats mais absente du top 10, devrait être optimisée par une démarche SEO affectant son contenu sémantique plutôt qu'une démarche client pour augmenter le nombre de commentaires ou les notes. En effet, enrichir certains champs aura plus d'impact que de créer une campagne de communication pour recueillir des avis consommateurs. Enfin, ce dernier tableau illustre la combinaison de plusieurs champs avec plus de 50% des mots de la recherche présents dans plusieurs champs à la fois. (La limite à 100% a été retirée au vue d'une distribution réduite de la part de pages produits ayant 100% des mots de la recherche dans leurs champs. De plus, l'évolution entre le Top 10 et la première page pour la limite de 100% est faible.) Dans ce tableau, on observe que le Top 10 est mieux enrichi sémantiquement. Comme observé précédemment, il est donc important que les mots de la recherche soient présents dans les différents champs de la page produit. Un enrichissement SEO est donc pertinent.

| Champs (+ 50%) | PT | PT & FBI | PT & FBI & PD | PT & FBI & PD & CR |
|----------------------|-----|----------|---------------|--------------------|
| Sur la première page | 47% | 31% | 26% | 16% |
| Dans le Top 10 | 54% | 36% | 29% | 19% |

Figure 16: Combinaisons de plusieurs champs où plus de 50% des mots d'une recherche apparaissent dans les champs d'une page produit

7.2.2 Création de modèles

Abordons maintenant la réalisation de modèles. L'objectif est de prédire à l'aide d'un modèle de machine learning si un produit sera dans le Top 10 d'une liste de pages produits pour une recherche d'un utilisateur. Dans cette partie, j'ai choisi les variables suivantes : notre label Top 10 et les variables explicatives. Ces dernières sont : note du produit supérieure à 4.5, Prime, 50% des mots de la recherche de l'utilisateur sont dans les informations en points (FBI), la description du produit, les commentaires clients ou le titre du produit. On a donc 6 variables explicatives. Il est judicieux avant d'entraîner tout modèle de regarder les corrélations entre les variables. La photo expose ci-dessous les corrélations selon Spearman et Pearson. Cette illustration montre peu de corrélation entre notre label et les variables explicatives.

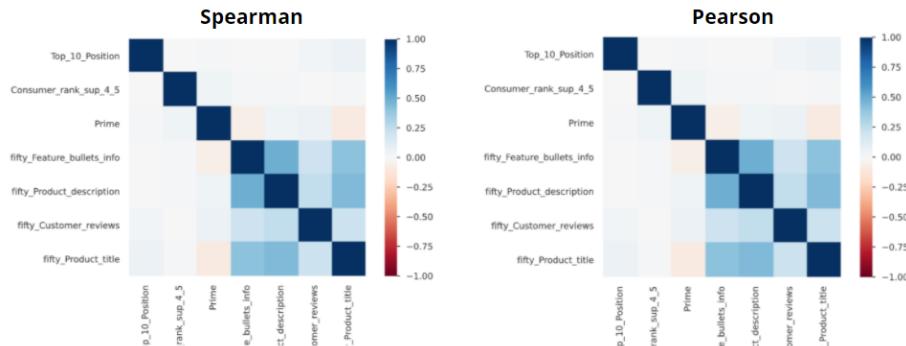


Figure 17: Corrélation de Spearman et de Pearson

Le premier modèle entraîné est une régression logistique binaire. Les résultats obtenus pour ces trois modèles sont peu concluants. Nos modèles ne font guerre mieux qu'une fonction uniforme. Tous trois donnent une précision de 54% et de 55% pour le train et test set et les p_values correspondant à chaque variable ne sont pas inférieures à 5% sauf pour la variable "fifty_Product_title". Enfin, les métriques ROC et la matrice de confusion donnent de mauvais résultats.

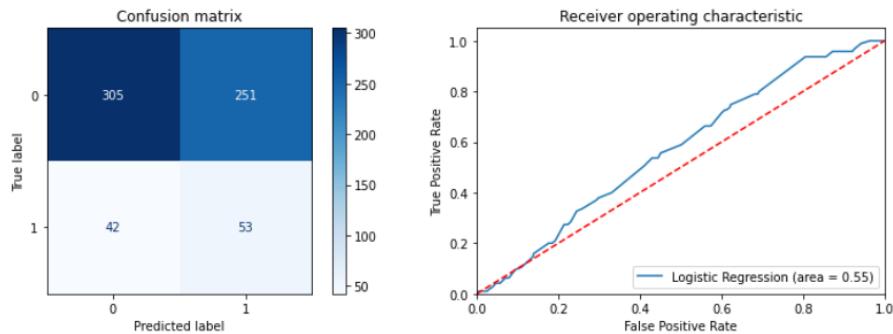


Figure 18: Matrice de confusion et courbe ROC

```

1 clf = LogisticRegression(penalty='none', random_state=3, class_weight = 'balanced').fit(
2     X_train , y_train)
3 clf = LogisticRegression(penalty='l1', random_state=3, class_weight = 'balanced', solver=
4     "liblinear", l1_ratio = 0.5).fit(X_train , y_train)
5 clf = LogisticRegression(penalty='l2', random_state=3, class_weight = 'balanced',
6     l1_ratio = 0.2).fit(X_train , y_train)

```

Le second modèle utilise RandomForest. Malgré l'utilisation d'un GridSearch, les résultats ne sont pas meilleurs : 53% et de 51% pour le train et test set.

```

1 from sklearn.model_selection import GridSearchCV
2 from sklearn.ensemble import RandomForestRegressor
3 from sklearn.model_selection import RandomizedSearchCV
4 n_estimators = [int(x) for x in np.linspace(start = 10, stop = 1000, num = 10)] # Number
   of trees in random forest
5 max_features = ['auto', 'sqrt'] # Number of features to consider at every split
6 max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]# Maximum number of levels in
   tree
7 max_depth.append(None)
8 min_samples_split = [2, 5, 10] # Minimum number of samples required to split a node
9 min_samples_leaf = [1, 2, 4] # Minimum number of samples required at each leaf node
10 bootstrap = [True, False] # Method of selecting samples for training each tree
11 random_grid = { 'n_estimators': n_estimators,
12                 'max_features': max_features,
13                 'max_depth': max_depth,
14                 'min_samples_split': min_samples_split ,
15                 'min_samples_leaf': min_samples_leaf ,
16                 'bootstrap': bootstrap} # Create the random grid
17 rf = RandomForestClassifier(class_weight='balanced')
18 rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter
   = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
19 rf_random.fit(X_train, y_train)

```

De plus, à l'aide du colab disponible par ce lien [21]. Il est possible de calculer l'Information Value. Les résultats obtenus montrent que les variables choisies semblent peu pertinentes.

| | VAR_NAME | IV | Predictive Power |
|---|----------------------------|----------|------------------------|
| 0 | Consumer_rank_sup_4_5 | 0.000157 | useless for prediction |
| 1 | Prime | 0.002865 | useless for prediction |
| 2 | fifty_Customer_reviews | 0.004567 | useless for prediction |
| 3 | fifty_Feature_bullets_info | 0.000048 | useless for prediction |
| 4 | fifty_Product_description | 0.000147 | useless for prediction |
| 5 | fifty_Product_title | 0.022466 | weak predictor |

Figure 19: Résultats pour la métrique : Information Value

En conclusion, de ces expérimentations, il n'a pas été possible de construire un modèle de prédiction performant. Deux explications sont envisageables : les variables choisies ne sont pas assez corrélées ou pertinentes pour prédire la position d'un produit ou bien le jeu de données n'est pas assez important. Ce premier point souligne la complexité de l'algorithme A9 d'Amazon. En effet, il est ambitieux de penser que 6 variables vont réussir à expliquer la méthode de référencement. Le second point peut être facilement corrigé en scrappant plus de contenu. Il est donc prévu de refaire une analyse lorsque plus d'études auront été lancées. Si rien ne change alors d'autres variables devront être utilisées en gardant en tête qu'il est peu probable que la précision soit très bonne. Enfin, l'étude préliminaire est concluante et prouve que l'algorithme d'Amazon A9 porte plus d'importance au contenu sémantique de la page produit plutôt que les métadata pour référencer un article dans le Top 10 suivant une recherche d'un utilisateur.

8 Déterminer les similarités entre les mots-clefs : méthodes utilisant l'intelligence artificielle

Cette phase est très importante dans ce mémoire. Elle contribue à l'obtention des similarités entre mots calculées lors de la seconde phase de l'analyse. De plus, elle a soulevé de nombreuses problématiques spécifiques à cette technique de machine learning. En effet, au contraire de la labellisation d'images ou la catégorisation sémantique de documents, dans notre cas, l'apprentissage est non supervisé. Cette situation met en avant un défi de taille sur l'évaluation du modèle. Cette partie expliquera la démarche scientifique entreprise et détaillera l'ensemble des indicateurs sélectionnés pour mesurer la précision de notre modèle. Enfin, M13H a pour ambition de créer son modèle de référence pour exploiter le potentiel du transfer learning. Ce dernier point nécessite une rigueur sur l'automatisation car cela requiert de nombreuses heures de calculs pour une machine. Il est donc important de paramétrier correctement les checkpoints. Avant d'expliquer la phase d'industrialisation de l'outil, nous allons dans ce point détailler plusieurs aspects du machine learning. Une première partie abordera ce qu'est un réseau de neurones profonds et expliquera comment il est possible de convertir du texte en nombre. Une seconde partie décrira les méthodes Word2Vec, GloVe et de transfer learning. Enfin, une dernière partie s'intéressera de près au choix des métriques pour évaluer nos modèles. Suite à cette introduction théorique, une phase expérimentale sera présentée et enfin, la méthode retenue sera exposée.

8.1 Points communs entre les différentes méthodes :

8.1.1 Apprentissage profond

Cette section reprend les informations contenues dans plusieurs articles : [1], [35] et [4]. L'apprentissage profond est une sous-partie de l'intelligence artificielle. Il est communément comparé à notre cerveau. En effet, l'apprentissage profond repose sur une architecture de neurones répartis sur plusieurs couches inter-connectées. D'une façon générale, un réseau de neurones prend en entrée une donnée qu'il souhaite par exemple classifier. Un réseau de neurones est un modèle d'intelligence artificielle, il est tout d'abord entraîné avec des données labellisées. Ces données permettent de faire une série de calculs pour estimer au mieux les poids tout en minimisant l'erreur d'estimation. L'objectif de cette partie est d'expliquer comment sont calculés les poids.

La valeur de chaque neurone est constituée d'une combinaison linéaire des entrées, de poids et d'un biais. Une fonction d'activation est appliquée sur cette fonction affine dans le but de normaliser les valeurs : $n_j = g(\sum_{i=1}^n w_i x_i + b_i)$. Cette valeur ainsi obtenue constitue la mémoire ou connaissance répartie du réseau. On retrouve plusieurs types de fonction d'activation. L'ensemble des fonctions, pour la bibliothèque *keras*, est disponible par ce lien : [12]. Les plus connues sont les fonctions suivantes :

- Relu (Rectified linear unit) : $g(x) = \max(0, x)$ et $g'(x) = 1$ si $x > 0$ sinon 0
- sigmoïd : $g(x) = \frac{1}{1+\exp -x}$ et $g'(x) = 1$ if $x > 0$ sinon 0
- softplus : $g(x) = \ln(1 + \exp x)$ et $g'(x) = g(x)(1 - g(x))$

Ces fonctions d'activation possèdent un avantage intéressant car elles sont différentiables. Cet aspect permet d'introduire la *Backpropagation*, c'est à dire la rétropropagation du gradient. C'est une méthode pour calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones, en partant de la

dernière couche vers la première. L'objectif est d'optimiser les poids de telle sorte que n'importe quelle entrée soit correctement classée. Cette optimisation a lieu à chaque epoch lors de l'apprentissage.

Voici un exemple simple communément utilisé pour expliciter la rétropropagation du gradient. Soit un réseau de neurones avec deux entrées 0.05 et 0.10, deux neurones cachés et deux sorties 0.01 et 0.99. La première étape consiste à attribuer des poids de façon aléatoire. Ensuite, il faut calculer le *net input* pour chacun des neurones cachés. Puis, il faut normaliser cette valeur à l'aide de la fonction d'activation (sigmoid). Enfin, on calcule l'erreur quadratique moyenne. La seconde étape vise à minimiser cette erreur. Cela se fait par rétropropagation du gradient. Il permet de mettre à jour les poids dans le but de minimiser l'erreur. On utilise la différentiation. Si on souhaite calculer le nouveau poids w_i alors il suffit de dériver l'erreur par rapport à w_i . Et de mettre à jour cette valeur grâce à la formule de descente du gradient avec un learning rate η : $w_i = w_i - \eta * \frac{\delta \text{Erreur}}{\delta w_i}$. Une fois les poids actualisés, il faut reprendre la première étape et minimiser à nouveau l'erreur tant que celle-ci est supérieure à $\epsilon = 0.1$.

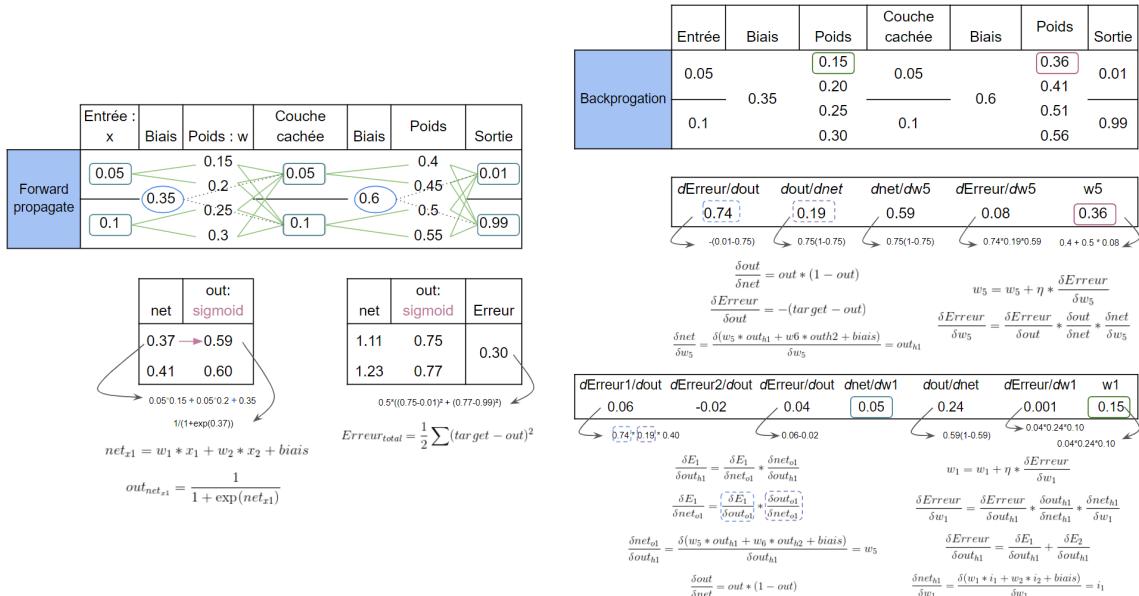


Figure 20: Explication mathématique des calculs effectués pour la mise à jour des poids à chaque epoch dans un réseau de neurones

Un réseau de neurones peut être composé de plusieurs types de couches. Ces couches ont différent objectifs et peuvent être régularisées.

- La couche **dense** : elle connecte simplement tous les neurones d'une couche à une autre. Elle peut être régularisée par un régulariser l1 (Lasso) ou l2 (Ridge). Cela permet de simplifier le modèle, c'est à dire de réduire le nombre de poids ou d'ajouter une pénalité à la fonction loss pour éviter le surapprentissage. Il est aussi possible d'ajouter un Dropout. Cette technique permet de supprimer une partie des neurones à chaque epoch de façon aléatoire.

- Les couches **convolutives** sont utilisées dans le traitement d'image ou le traitement du langage. Elles permettent de se concentrer sur une partie d'une image et de détecter des courbes. Un ou plusieurs neurones traitent une partie de l'image appelée caractéristique. Chaque caractéristique peut être vu comme un filtre. Les filtres peuvent se chevaucher.
- La couche **Pooling** est placée le plus souvent entre deux couches de convolutions. Elle reçoit les caractéristiques et applique une réduction d'image tout en essayant de garder les caractéristiques importantes (récupère la valeur maximale). Cette couche est utile pour réduire le nombre de paramètres et de calculs dans le réseau. Généralement cela baisse le sur-apprentissage.
- La couche **globalMaxPooling2D** est similaire au pooling. Elle réduit la dimension en prenant seulement les valeurs les plus élevées de chaque couche. Dans le cas d'une image on compte trois couches pour RGB (trois couleurs).
- La couche **Flatten** elle permet lors de la classification de transformer une matrice en un seul vecteur correspondant à la prédiction pour les différents labels.
- La couche **embedding** permet en traitement du langage d'ajouter un bruit (gaussien par exemple). Ce bruit blanc permet de mieux généraliser et de baisser le sur-apprentissage.

8.1.2 Word embedding

Cette partie est centrale dans ce projet, en effet toutes les méthodes de machine learning présentées dans la suite se baseront sur cette transformation de la donnée. Cette description de la technique *word embedding* s'appuie sur ces deux sources : [2] , [22] .

Le terme word embedding encadre la représentation vectorielle de mots. Les mots d'un texte sont alors transformés en valeurs numériques à l'aide de nombres réels. Cette technique a pour but de manipuler ces mots avec des réseaux de neurones comme vu dans le point précédent. Le word embedding est utilisé dans le but de déduire une relation sémantique entre les mots d'un texte. Les métriques présentées dans la partie 6 relative aux méthodes statistiques, ne permettent pas de capturer le sens d'un mot et son contexte. Ce procédé est utilisé dans tout traitement automatique du langage, il permet, accompagné de l'utilisation du machine learning, d'effectuer une analyse de sentiments ou une analyse sémantique. Pour illustrer cette notion, nous pouvons prendre ces exemples. Dans un corpus composé de nombreux avis consommateurs, on souhaite identifier les mots faisant référence à un commentaire négatif et d'autres positifs. Dans une analyse sémantique classique, on souhaiterait rapprocher tous les mots permettant de caractériser le sentiment d'un utilisateur. Ce premier exemple montre que dans le cas d'une analyse de sentiment on souhaite opposer les mots "beau" et "moche" ou encore "mauvais" et "bon", alors qu'une analyse sémantique les rapprocherait.

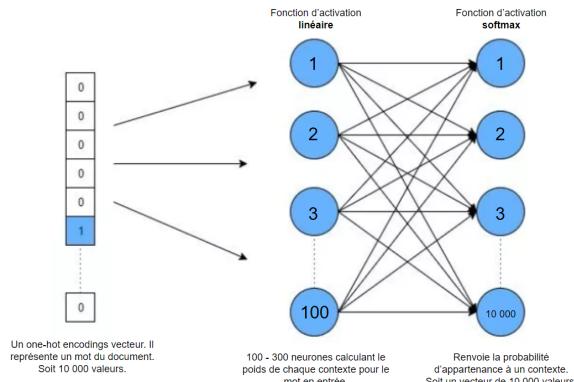
Différentes approches existent pour vectoriser des mots. Prenons un corpus de n mots. Une méthode naturelle serait d'attribuer à chaque mot un nombre entre 1 et n . Chaque mot serait représenté par un vecteur composé de $n - 1$ valeurs nulles et d'un 1. Cette première technique se nomme en anglais *one-hot encodings* mais elle possède plusieurs limites. En effet, il est difficile de représenter des similarités entre les mots. La représentation trop simple ne permet pas à l'aide d'opérations vectorielles de déduire des analogies entre mots. De plus, chaque mot est représenté par un vecteur très grand, l'augmentation du nombre de mots aura un impact direct sur la grandeur des vecteurs et donc sur le nombre de paramètres

à calculer. La complexité du modèle deviendra exponentiellement plus importante. De plus, la mémoire est souvent un facteur limitant d'une machine, les données doivent donc être judicieusement formatées pour tenir en RAM. Cette méthode n'apparaît pas envisageable pour pouvoir déduire des similitudes entre mots. En effet, la similarité entre deux points x et y est souvent calculée à l'aide la similarité cosinus :

$$\frac{x^T y}{\|x\| * \|y\|}$$

Or, la similarité cosinus entre deux *one-hot vectors* correspond au produit des deux vecteurs donne 0, donc il n'est pas possible avec cette méthode de calculer la similarité entre ces deux vecteurs, c'est à dire deux mots.

Le réseau de neurones est la clef pour enrichir la représentation vectorielle d'un mot. Il est capable de déterminer le sens d'un mot à l'aide d'un vecteur possédant une centaine de valeurs. Ce type de modèle prend en entrée un vecteur *one-hot encodings* (le même que vu précédemment) et à l'aide du réseau on calcule la probabilité que des mots appartiennent à son contexte. La fonction d'activation pour la dernière couche est une softmax car elle donne la probabilité que chaque mot appartienne à un contexte, c'est à dire une classe. Une fois le modèle entraîné la dernière couche du réseau est supprimée et la représentation vectorielle du mot en entrée est définie par la dernière couche cachée. L'ensemble des poids de cette couche caractérise la représentation vectorielle d'un mot.



Réseau de neurones pour la représentation vectorielle d'un mot

8.2 Présentation des différentes méthodes :

8.2.1 Word2Vec

Cette partie reprend les notions abordées dans ce cours de l'université de Stanford disponible par ce lien: [10] qui lui-même s'est inspiré de ces deux sources [[WordVecVideo000](#)], [[WordVecVideo00](#)]. Word2Vec est un algorithme utilisant les réseaux de neurones pour fournir une similarité entre les mots d'un document. Il permet de déduire des synonymes ou un champ lexical proche. Ce modèle se base sur l'hypothèse de distribution sémantique : les mots qui apparaissent dans le même contexte ont un sens proche. Deux implémentations de Word2Vec sont disponibles dans la bibliothèque *gensim* :

- CBOW (Continuous Bag Of Words) : il prend en entrée le contexte d'un mot, c'est à dire les mots se trouvant avant et après lui suivant une fenêtre définie en paramètre et il retourne le mot au centre.
- Skip-Gram : il prend en entrée un mot et prédit son contexte, c'est à dire les mots se trouvant autour. De la même manière, l'utilisateur donne une fenêtre pour retourner le nombre de mots souhaités.

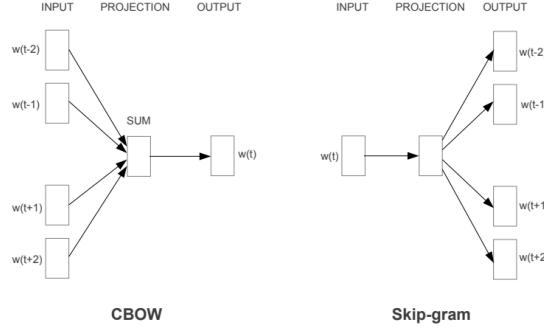


Figure 21: Architecture CBOW et Skip-Gram disponible sur la bibliothèque Word2Vec Gensim

Si on s'intéresse de plus près à l'algorithme utilisant *Skip-Gram*, le réseau de neurones prend en entrée l'ensemble du document. Pour chaque mot du document le réseau optimise les poids dans le but de renvoyer le contexte du mot. Deux approches sont possibles pour mettre à jour les poids.

Une première approche se définit ainsi, comme vu dans la partie précédente, il est possible d'utiliser une fonction d'activation softmax et de renvoyer les N probabilités les plus élevées. Dans ce cas le réseau de neurones maximise cette fonction :

$$\frac{1}{N} \sum_{n=1}^N \sum_{-f \leq j \leq f, j \neq 0} \log P(x_{n+j} | x_n)$$

On remarque que plus la fenêtre f est grande plus la précision devrait augmenter. Une fois la fonction de coût définie, il suffit de détailler la fonction softmax avec $p(x_{n+j}|x_n)$ la probabilité que le mot x_n appartienne au contexte j défini par le mot x_{n+j} (la classe j) :

$$P(x_{n+j} | x_n) = \frac{\exp(v_{n+j}^T v_n)}{\sum_{k=1}^K \exp(v_k^T v_n)} \quad (1)$$

Enfin, l'algorithme effectue une descente de gradient avec une optimisation SGD, la mise à jour des vecteurs se fait à chaque epoch à l'aide de la formule de la dérivée ci-dessous.

$$\begin{aligned} \log P(x_{n+j} | x_n) &= (v_{n+j}^T v_n) - \log \left(\sum_{k=1}^K \exp(v_k^T v_n) \right) \\ \frac{\log P(x_{n+j} | x_n)}{\delta v_n} &= v_{n+j} - \frac{(\sum_{l=1}^K \exp(v_l^T v_n) v_l)}{\sum_{i=1}^K \exp(v_i^T v_n)} = v_{n+j} - \sum_{l=1}^K P(x_l | x_n) v_l \end{aligned}$$

Le numérateur, dans l'égalité (1), effectue un produit entre les deux vecteurs. Si la similarité est forte alors la valeur sera élevée : $u^T v = \sum_{i=1}^n u_i v_i$. De plus, il est important de remarquer qu'un mot est alors décrit par deux vecteurs : son contexte v_{n+j}^T et lui même v_n . Cette opération est très coûteuse en effet, le calcul du dénominateur calcule tous les produits vectoriels entre le mot en question et les K autres mots. De plus, pour chaque donnée d'entraînement les poids seront très peu modifiés à part celui correspondant au mot cible, or l'objectif d'un réseau de neurones est de mettre à jour l'ensemble des poids à chaque epoch. Une autre idée, développée dans la suite, repose sur le calcul d'un échantillon et permet d'optimiser le temps d'apprentissage.

Une méthode nommée *negative sampling* permet de modifier seulement certains poids. Le but est de prédire la probabilité que le mot en question soit voisin d'un autre. On évalue alors, le mot en question et un mot de son contexte comme une régression logistique binaire. La fonction de coût devient alors $J(\theta) = \frac{1}{N} \sum_{n=1}^N J_n(\theta)$ avec N le nombre de fenêtres,

$$J_n(\theta) = \log(\sigma(v_{n+j} v_n)) + \sum_{j \sim P(x)} [\log \sigma(-v_j^T v_n)]$$

Dans cette formule, le premier terme est caractérisé par la fonction sigmoïde notée σ , elle retourne 1 ou 0. L'objectif est alors de maximiser la log probabilité de ces deux termes d'apparaître. Le second terme calcule la somme, sur un échantillon de la donnée, que le mot en question et d'autres mots apparaissent. Dans ce second cas on souhaite minimiser la probabilité d'apparition simultanée. Cet échantillon de données est calculé par l'intermédiaire de cette formule $P(x) = \frac{\text{Unif}(x)^{\frac{3}{4}}}{N}$. Soit une loi uniforme caractérisant le nombre de fois où un mot apparaît dans le document, la puissance a été choisie de manière empirique et permet de donner moins de poids aux stopwords. Le nom de cette méthode *negative sampling*, caractérise le nombre de mots utilisés dans ce second terme. Généralement on choisit moins de 5 mots pour un corpus avec de nombreux mots et entre 5-20 dans le cas contraire.

Du côté de l'implémentation, la bibliothèque *gensim* sera utilisée comme présenté ci-dessous. La fonction d'entraînement possède plusieurs hyper-paramètres. Le `min_count` caractérise le nombre de fois où chaque mot doit être présent au minimum pour être intégré au modèle. La taille (`size`) définit la dimension du vecteur représentant chaque mot après apprentissage. C'est la sortie de notre modèle. Le terme `worker` décrit le nombre de coeurs utilisés pour effectuer le travail, ce choix est spécifique à la configuration de la machine. Plus ce nombre est grand plus l'apprentissage est rapide, attention un nombre élevé n'aura pas d'effet si les caractéristiques de la machine ne le permettent pas. Le `window` définit le nombre de mots appartenant au contexte du mot central. Enfin, `sg` renseigne la méthode utilisée Skip-Gram ou CBOW. Il est aussi possible de modifier des paramètres par défaut comme le *negative sampling*.

```

1 from keras.layers import dot
2 !pip install gensim
3 from gensim.models import Word2Vec
4
5 similarity_data = stemming(remove_stopwords(corpus_to_sentences(
6     remove_punctuation_and_lowering(texte))))[1]
7 model = Word2Vec(similarity_data, min_count=2, size= 300, workers=3, window =3, sg = 1)
8 words2 = model.wv.vocab
9 print(model.wv.vectors.shape)

```

Dans le cas de l'algorithme utilisant CBOW l'implémentation est un peu différente. Cette implémentation va être détaillée ci-dessous et reprend cette article : [23] On s'intéresse alors à la probabilité du mot central sachant la fenêtre f utilisée avec $j \neq 0$ et $j \in [-f, f]$:

$$P(x_n|x_{n-f}, \dots, x_{n+f}) = \frac{\exp(\frac{1}{2f}v_n^T * (\sum_{j=-f}^f v_{n-j}))}{\sum_{k=1}^K \exp(\frac{1}{2f}v_k^T * (\sum_{j=-f}^f v_{n-j}))}$$

De manière simplifiée, en posant $\bar{v}_n = \frac{\sum_{j=-f}^f v_{n-j}}{2f}$ et $X_F = \{x_{n-f}, \dots, x_{n+f}\}$:

$$P(x_n|X_F) = \frac{\exp(v_n^T \bar{v}_n)}{\sum_{k=1}^K \exp(v_k^T \bar{v}_n)}$$

La fonction de coût à minimiser pour les K mots est donc la suivante :

$$-\sum_{k=1}^K \log P(x^{(k)}|x^{(k-f)}, \dots, x^{(k+f)})$$

Il suffit alors de calculer la dérivée pour pouvoir faire la mise à jour avec la descente de gradient :

$$\begin{aligned} \log P(x_n|X_F) &= v_n^T \bar{v}_n - \log(\sum_{k=1}^K \exp(v_k^T \bar{v}_n)) \\ \frac{\delta \log P(x_n|X_F)}{\delta v_{n_k}} &= \frac{1}{2f}(v_n - \frac{\sum_{j=1}^K \exp(u_j^T \bar{v}_n) v_j}{\sum_{k=1}^K \exp(v_k^T \bar{v}_n)}) = \frac{1}{2f}(v_n - \sum_{j=1}^K P(x_j|X_F) v_j) \end{aligned}$$

Enfin, pour visualiser les coordonnées d'un mot avec la méthode CBOW, on utilise le vecteur de contexte plutôt que le vecteur du mot. L'implémentation se fait à l'aide de la bibliothèque *gensim* de cette manière.

8.2.2 GloVe

Word2Vec étudie le contexte d'un mot suivant une fenêtre définie en paramètre. Cette méthode est statistiquement peu efficace car requiert de nombreux calculs pour savoir quand deux mots apparaissent simultanément. Il serait donc judicieux de calculer la matrice de co-occurrences sur l'ensemble du document. Néanmoins, il est important de négliger l'apport des mots très fréquents.

La méthode Global Vector nommée GloVe permet de combiner les deux points précédents. Tout d'abord définissons P_{ij} la matrice de co-occurrences. Elle découpe le document en phrases, et calcule le nombre de fois où un mot apparaît en même temps qu'un autre dans une phrase.

Le but est alors de minimiser cette fonction de coût :

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^K f(P_{ij})(v_i^T v_j - \log P_{ij})^2$$

Cette fonction parcourt deux à deux tous les K mots du document. Le premier terme est composé de la fonction f . Cette fonction permet de diminuer l'importance des mots très fréquents (stopwords) telle que $f(x) = \min((\frac{x}{x_{max}})^\alpha, 1)$ avec $x > 0$ et $\alpha = 0.75$. Le second terme calcule la distance carrée

entre le produit des deux vecteurs et le log du nombre de fois où les deux mots apparaissent dans la même phrase. Au contraire du Word2Vec, le calcul est beaucoup plus rapide car ne requiert pas une optimisation par fenêtre. Le but est alors de faire converger le deuxième terme vers 0 pour minimiser la fonction de coût.

8.2.3 Transfer learning

Le transfer learning est une méthode d'apprentissage visant à utiliser un modèle pré-entraîné pour entraîner à nouveau ce modèle sur notre data set. Il n'est en effet pas obligatoire de ré-entraîner ce modèle car il est capable de labelliser suivant une entrée. Dans notre cas, nous souhaitons trouver les mots similaires à un mot donné dans le contexte de notre analyse, il est donc intéressant d'entraîner à nouveau le modèle uniquement avec le corpus fraîchement scrapé. Cette méthode permettra de garantir des résultats fiables selon les métriques exposées plus bas.

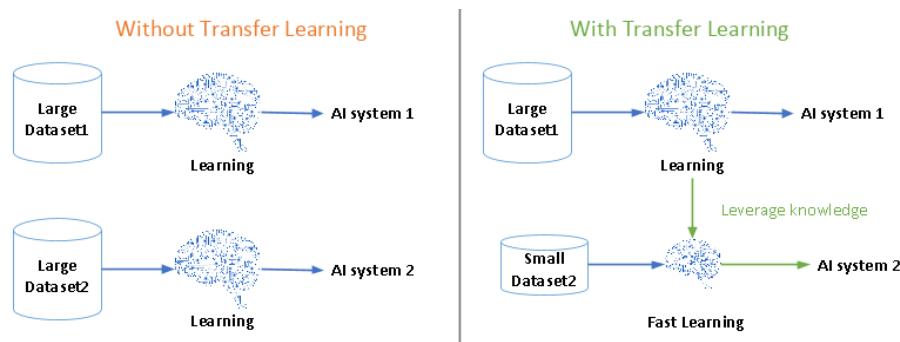


Figure 22: Croquis présentant la différence entre une méthode utilisant du transfer learning ou non

Word2Vec regroupe, dans la bibliothèque de gensim, des fonctions permettant d'effectuer du transfer learning à partir d'un modèle déjà entraîné. Dans ce mémoire, un des modèles en open source de Mr Fauconnier [11] sera utilisé. Le choix de ce modèle sera détaillé ultérieurement. La méthode de transfer learning est montré ci-dessous :

```

1   :param corpus: list of lists (sentence&word) corpus of the study
2   :param min_count: int it used to train words contained in the corpus of the study
3
4   model_path = "frWiki_no_lem_no_postag_no_phrase_1000_skip_cut100.bin" # Fauconnier's
5   model
6   model = Word2Vec(window=4, size=1000, min_count=min_count, iter=10)
7   model2 = KeyedVectors.load_word2vec_format(model_path, binary=True) # Load Fauconnier
8   's model
9   model.build_vocab(corpus)
10  model.build_vocab([list(model2.vocab.keys())], update=True)
11  model.intersect_word2vec_format(model_path, binary=True, lockf=1.0)
12  epoch = 30
13  model.train(corpus, epochs=epoch, total_examples=model.corpus_count, compute_loss=
14      True)

```

8.3 Évaluation d'un modèle

L'idée principale est maintenant d'évaluer un modèle entraîné par nos soins ou un modèle pré-entraîné. Il faut avant tout définir une ou plusieurs métriques pour comparer correctement ces modèles. Dans la littérature l'évaluation d'un modèle non-supervisé est toujours un challenge. En effet, il est difficile de trouver une unique métrique pour évaluer un modèle. Cette partie exposera quatre métriques couramment utilisées, trois d'entre-elles seront exploitées et implémentées à l'aide de checkpoints. Au cours de l'apprentissage, à chaque epoch les métriques seront calculées. Si une amélioration est observée alors le modèle est automatiquement enregistré. Pour rappel, le data scientist aura pour travail de choisir le meilleur modèle au regard de trois graphes. Ce dernier aspect sera détaillé dans la phase expérimentale.

Tout d'abord, pour un modèle entraîné sur un corpus anglais il est commun d'utiliser l'ensemble des dataset ci-dessous comme présenté dans cette article [**Evaluation**].

TABLE I
WORD SIMILARITY DATASETS USED IN OUR EXPERIMENTS WHERE PAIRS
INDICATE THE NUMBER OF WORD PAIRS IN EACH DATASET.

| Name | Pairs | Year |
|---------------------|-------|------|
| WS-353 [40] | 353 | 2002 |
| WS-353-SIM [41] | 203 | 2004 |
| WS-353-RiL [41] | 252 | 2009 |
| MC-30 [42] | 30 | 1991 |
| RG-65 [43] | 65 | 1965 |
| Rare-Word (RW) [44] | 2034 | 2013 |
| MEN [45] | 3000 | 2012 |
| MTurk-287 [46] | 287 | 2011 |
| MTurk-771 [47] | 771 | 2012 |
| YP-130 [48] | 130 | 2006 |
| SimLex-999 [49] | 999 | 2014 |
| Verb-143 [50] | 143 | 2014 |
| SimVerb-3500 [51] | 3500 | 2016 |

The WordSimilarity-353 Test Collection

| # Word 1 | Word 2 | Human (mean) |
|----------|----------|--------------|
| love | sex | 6.77 |
| tiger | cat | 7.35 |
| tiger | tiger | 10.00 |
| book | paper | 7.46 |
| computer | keyboard | 7.62 |
| computer | internet | 7.58 |

Figure 23: Liste de datasets anglais pour valider un modèle Word2Vec

Ces datasets sont composés de mots en anglais. Certains sont disponibles en d'autres langues mais seul un dataset présente une similitude entre mots français : FR-WS-353. Naïvement, on pourrait penser que traduire ces mots en français et garder le même score pourrait suffire, mais cet article [19] montre que cela est bien plus compliqué et relève d'un travail important pour montrer que la simple traduction est valable. Cependant dans cette article les deux auteurs parviennent à démontrer que la traduction de l'anglais à l'allemand ou de l'anglais au français du dataset : Rubenstein and Goodenough's donne des corrélations pertinentes. Le dataset WS-353 présente une métrique basée sur une similarité relative au jugement humain. La définition du score est détaillée dans l'article mais cela a moins d'importance dans ce mémoire.

Dans la bibliothèque de Gensim, une évaluation est couramment utilisée. D'une part on peut calculer la similarité entre deux vecteurs à l'aide de la similarité cosinus. Chaque paire de mots est caractérisée par une valeur entre -1 (opposé) et 1 (identique). La distance cosinus et le jugement humain sont deux métriques différentes mais comparables à l'aide de la corrélation de Spearman et de Pearson. La corréla-

tion de Spearman évalue la monotonie des scores de chaque paire, tandis que la corrélation de Pearson caractérise la linéarité.

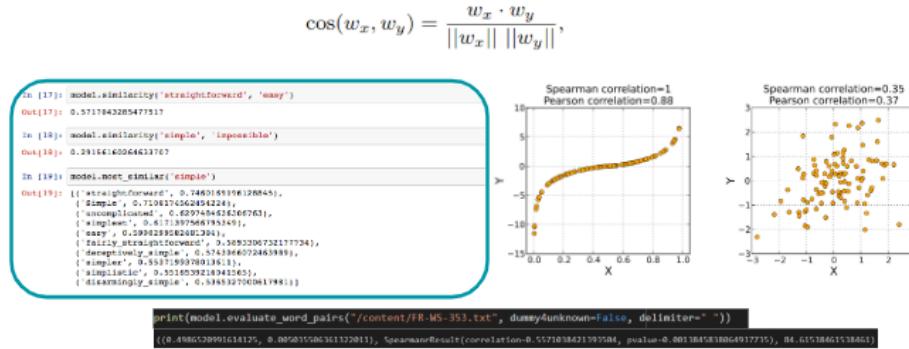


Figure 24: Métrique de similarité cosinus entre des paires de mots

L'objectif est donc d'avoir une p-value inférieur à 5% garantissant que notre modèle possède des relations entre mots pertinents au regard du test choisi. De plus, il est très important de faire attention au taux de match entre le jeu de test et notre modèle. En effet, il se peut que notre modèle et le jeu de test aient très peu de mots en commun et cela pourrait rendre l'évaluation non pertinente (sur l'image ci-dessus 84% des paires ne sont pas retrouvées ce qui est trop important).

Une seconde métrique est souvent utilisée. Une idée développée dans un cours de Stanford University [23] est d'utiliser une analogie entre les vecteurs des mots. Prenons les mots homme (a), femme (b) et roi (c). Si homme se ramène à femme alors on souhaite que roi se rapproche de reine (d). Comme présenté dans cette article [18], on peut utiliser cette formule mathématique : $d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$. Cette distance cosinus permet de capturer des analogies syntaxiques intéressantes. Il s'avère que le vecteur ayant une similarité cosinus la plus importante suivant le premier terme de cette formule est une métrique fiable pour évaluer différents modèles.



Figure 25: Métrique d'analogie

Enfin, deux autres métriques sont envisageables. Comme précédemment exposé dans la partie "Transfer learning", il est possible d'utiliser un modèle pré-entraîné. Ce modèle peut être évalué au regard du

dataset FR-WS-353. Une fois ce modèle validé, on peut l'utiliser comme modèle de référence. Il est alors possible de constituer deux fichiers d'évaluation. Un fichier regroupera des paires de mots et leur similarité cosinus, et un autre donnant les 10 mots les plus similaires à un mot donné.

Figure 26: Exemple de dataset d'évaluation utilisé pour évaluer nos modèles

8.4 Présentation de la démarche scientifique et de la méthode retenue

8.4.1 Expérimentations

Plusieurs expériences ont été menées dans le but d'illustrer le comportement de Word2Vec et de montrer les performances d'un modèle de similarités simples. Pour entraîner ces modèles, j'ai utilisé un corpus résultant d'une étude réalisée sur le produit : Clean&Steam de Rowenta vendu sur Amazon. Suite à l'exécution des scripts pour l'analyse, j'obtiens un corpus constitué de 481.863 caractères et 36.894 caractères uniques. Ce scraping rassemble le contenu de 403 pages produits. Le tableau ci-dessous expose les effets du data preprocessing.

| | Avant | Après |
|--------------------|---------|---------|
| Caractères | 481.863 | 268.998 |
| Caractères uniques | 36.894 | 15.468 |

Figure 27: Évolution du nombre de caractères avant et après avoir effectué un data processing de la donnée

Avant de détailler les expériences, j'ai créé des métriques à l'aide d'un modèle de référence. L'objectif fut de trouver un modèle Word2Vec entraîné dont le corpus était français. Après quelques recherches sur le web, le site de Mr Fauconnier est apparu [11]. Celui-ci expose une série de 8 modèles entraînés sur 600 millions de mots provenant d'articles Wikipedia. Ces modèles diffèrent suivant le choix des hyperparamètres. Comme évoqué précédemment le fichier FR-WS-353 a permis de choisir un modèle. Tout d'abord, à l'aide de la corrélation de Spearman et de Pearson j'ai éliminé deux modèles. Ensuite l'écart cosinus a permis de sélectionner un modèle (seules les dimensions 1000 sont conservées, il serait mathématiquement faux de comparer des similarités cosinus pour des dimensions différentes). En effet, le choix s'est porté sur le modèle 4 non lemmatisé (car notre corpus ne l'est pas), comportant un nombre

de mots élevé et ayant une différence cosinus la moins importante. A partir de ce modèle j'ai extrait 3000 paires de mots communs à notre dataset avec leur similarité cosinus.

| Model | Dimension | Type | Min_count | Lemmatizer | Number of words | Spearman_corr | Spearman_pvalue | Person_corr | Person_pvalue | Common_pairs |
|--------|-----------|-----------|-----------|------------|-----------------|---------------|-----------------|-------------|---------------|--------------|
| model1 | 1000 | Skip_Gram | 200 | TRUE | 31.548 | 0.7 | 2.44E-11 | 0.02 | 1.49E-08 | 69 |
| model2 | 500 | Cbow | 10 | TRUE | 66.819 | 0.09 | 5.26E-11 | 0.02 | 1.19E-08 | 70 |
| model3 | 1000 | Skip_Gram | 200 | FALSE | 50.883 | 0.7 | 2.44E-11 | 0.02 | 1.43E-08 | 58 |
| model4 | 1000 | Skip_Gram | 100 | FALSE | 66.274 | 0.83 | 5.98E-09 | 0.57 | 2.38E-07 | 70 |
| model5 | 1000 | Cbow | 200 | FALSE | 50.883 | 0.22 | 8.80E-02 | 0.34 | 4.65E-03 | 58 |
| model6 | 1000 | Cbow | 100 | FALSE | 66.274 | 0.14 | 2.00E-01 | 0.26 | 2.60E-02 | 70 |
| model7 | 700 | Cbow | 100 | TRUE | 36.382 | 0.06 | 3.82E-10 | 0.59 | 7.68E-08 | 70 |
| model8 | 1000 | Skip_Gram | 100 | TRUE | 36.382 | 0.06 | 4.02E-11 | 0.61 | 2.16E-08 | 70 |

| Model | Dimension | Type | Min_count | Lemmatizer | Number of words | D<=0.1 |
|--------|-----------|-----------|-----------|------------|-----------------|--------|
| model1 | 1000 | Skip_Gram | 200 | TRUE | 31.548 | 26% |
| model3 | 1000 | Skip_Gram | 200 | FALSE | 50.883 | 26% |
| model4 | 1000 | Skip_Gram | 100 | FALSE | 66.274 | 29% |
| model5 | 1000 | Cbow | 200 | FALSE | 50.883 | 22% |
| model6 | 1000 | Cbow | 100 | FALSE | 66.274 | 14% |
| model8 | 1000 | Skip_Gram | 100 | TRUE | 36.382 | 26% |

Figure 28: Comparaison des modèles de Fauconnier suivant les métriques de corrélation et de distance cosinus

Premièrement, il a été décidé de construire un auto-encoder. Ce modèle est un réseau de neurones profond. Son but est de réduire la dimension d'une donnée tout en ne dégradant pas la donnée. L'auto-encoder possède deux parties, une encode la donnée entrante et l'autre décode cette donnée. L'objectif est de réduire la loss mesurant l'erreur entre la donnée en entrée et la donnée résultant du décodeur. Il est possible d'utiliser un set de validation pour évaluer le sur-apprentissage ou le sous-apprentissage du modèle. L'auto-encoder est souvent utilisé pour compresser des images, dans ce cas la donnée entrante est l'ensemble des pixels constituants l'image. Dans notre cas, nous souhaitons réduire la dimension d'un mot. Comme vu précédemment le onehot encoding est peu intéressant. J'ai créé une matrice de co-occurrences suivant un window. Cette approche est similaire à celle de la méthode GloVe. De plus, comme pour Word2Vec l'objectif sera d'encoder les mots avec une plus faible dimension que la donnée entrante. Ce vecteur est alors utilisé pour comparer les mots. La similarité cosinus fournit alors une proximité entre les mots du corpus. Pour cette expérience, j'ai utilisé un réseau constitué uniquement de couche denses. Deux métriques ont été utilisées. La loss sur un échantillon de validation est de train. La seconde métrique évalue la différence cosinus entre des paires de mots. Cette similarité a été comparée au cours de l'apprentissage. Les deux graphiques ci-dessous, montrent l'évolution de la loss et la proportion de paires de mots dont la différence cosinus est inférieure et supérieure à 0.3. On observe que la loss diminue très faiblement au cours de l'apprentissage et la proportion de mots avec une similarité supérieure à 0.3 ne varie pas et stagne autour de 85%. En conclusion, cette méthode n'est pas concluante car les résultats ne sont pas bons au regard de nos deux métriques. Enfin, les similitudes entre mots obtenues sont très discutables (ce résultat n'est pas illustré).

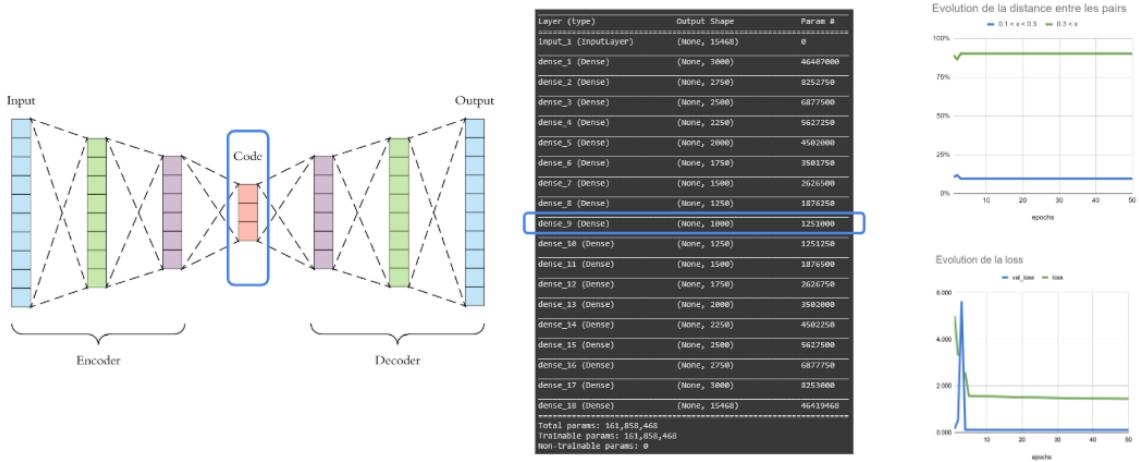


Figure 29: Architecture d'un auto-encoder avec Keras et évolution des métriques au cours de l'apprentissage

Une seconde expérience vise à créer un modèle CBOW à l'aide de Keras. La structure du modèle a été reprise de cet article [8]. Les checkpoints ont néanmoins été modifiés pour afficher les métriques souhaitées. De plus, la génération de la donnée a été codée par mes soins (vous pouvez retrouver l'implémentation en **Annexe 2**). Le réseau de neurones est relativement simple, il est composé d'une couche d'embedding initialisée avec des vecteurs aléatoires. Puis cet embedding est passé dans une couche lambda où la moyenne des mots constituant le contexte est calculée. Enfin, une troisième et dernière couche permet de prédire le mot central à l'aide de la fonction softmax. Dans cette expérience les mêmes métriques sont utilisées seule la précision est ajoutée. Les résultats obtenus montrent une amélioration de la précision tout au long de l'apprentissage, mais le set de validation semble stagner à 26%. On peut constater (entouré en noir) un sur-apprentissage avec la loss. Enfin, la dernière métrique montre que la proportion de mots dont la différence cosinus est supérieure à 0.3 augmente. Ce dernier point montre un manque de similarité avec le modèle de Fauconnier. Cependant, il est important d'avoir en tête que notre corpus est constitué de mots appartenant au champs sémantique des "aspirateurs". Il est donc très probable que le modèle de référence situe les mots "aspirateur" et "salon" proche alors que notre modèle CBOW lui, situera ces mots éloignés. Enfin, chaque epoch prend 30 secondes à se calculer pour un corpus restreint et une machine puissante (Colab) pourvue d'un GPU. En conclusion, cette méthode n'est pas viable pour notre outil mais montre l'efficacité de CBOW.

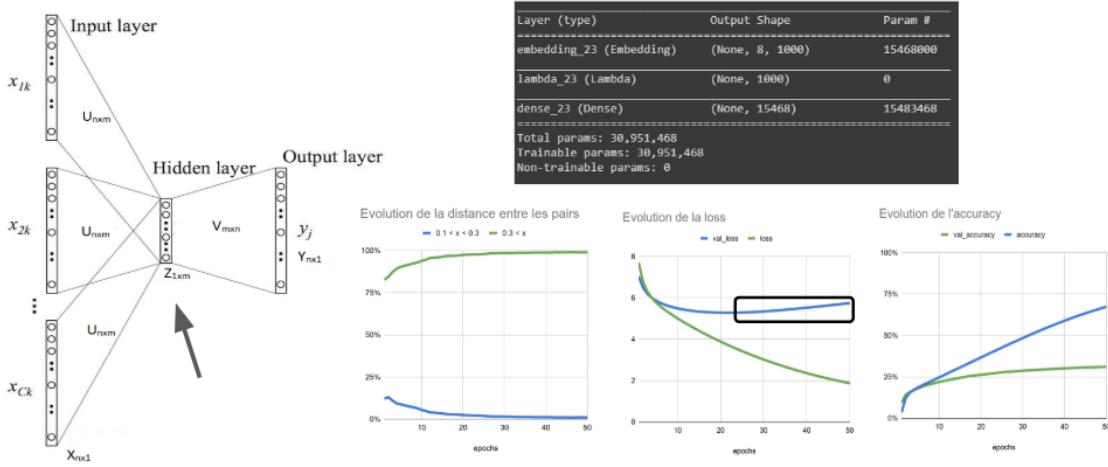


Figure 30: Architecture du modèle CBOW avec Keras et évolution des métriques au cours de l'apprentissage

Une troisième expérience consiste à utiliser la fonction de Word2Vec. Il suffit alors d'implémenter un checkpoint similaire aux précédents pour obtenir les métriques souhaitées. Word2Vec propose différents paramètres pour entraîner un modèle. Le modèle de Fauconnier de référence possède une dimension égale à 1000. J'ai donc choisi d'entraîner deux modèles avec un min_count=1, un window=4 et l'un avec la méthode CBOW et l'autre Skip-Gram. Les résultats semblent meilleurs pour le modèle de CBOW tant sur la corrélation que la proportion de mots dont l'écart cosinus est faible. En effet, la corrélation stagne après 10 epochs autour de 0.2. De plus, comme l'indique la ligne verticale sur le troisième graphique, il semble qu'un optimum ait été atteint lors de l'apprentissage autour de l'époque 4. Ces observations sont autant de checkpoints qui peuvent être paramétrés par la suite pour enregistrer de façon automatique un modèle dont les métriques s'améliorent.

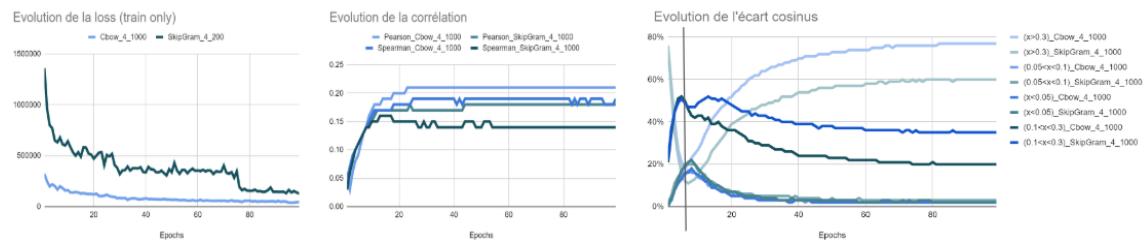


Figure 31: Comparaison de la loss, évolution de la corrélation et de l'écart cosinus pour un modèle CBOW et Skip-Gram

Une dernière expérience consiste à utiliser la fonction de Word2Vec avec du transfer learning. Dans ce cas les métriques utilisées sont identiques mais nous allons observer une dégradation des métriques. En effet, les poids sont initialisés avec le modèle de Fauconnier. Les métriques étant créées à partir de ce modèle, la première époque affichera une corrélation de 1 et une similarité cosinus parfaite. Il faut donc maîtriser la détérioration des métriques, c'est à dire entraîner à nouveau le modèle pour qu'il s'adapte à notre cas pratique sans trop fausser la prédiction. De plus, notre modèle de référence étant de type Skip-Gram, l'expérience compare deux modèles entraînés avec Skip-Gram, l'un entraîné en utilisant du Transfert Learning et l'autre non. Les graphes suivants sont cohérents avec les points soulevés précédemment. En effet, les poids étant initialisés non aléatoirement, la loss démarre beaucoup plus bas. De plus, on remarque que la différence cosinus se détériore. Il peut être intéressant de garder un écart cosinus pour $x < 0.1$ de plus de 80%. Cet état est atteint autour de 20 epochs.

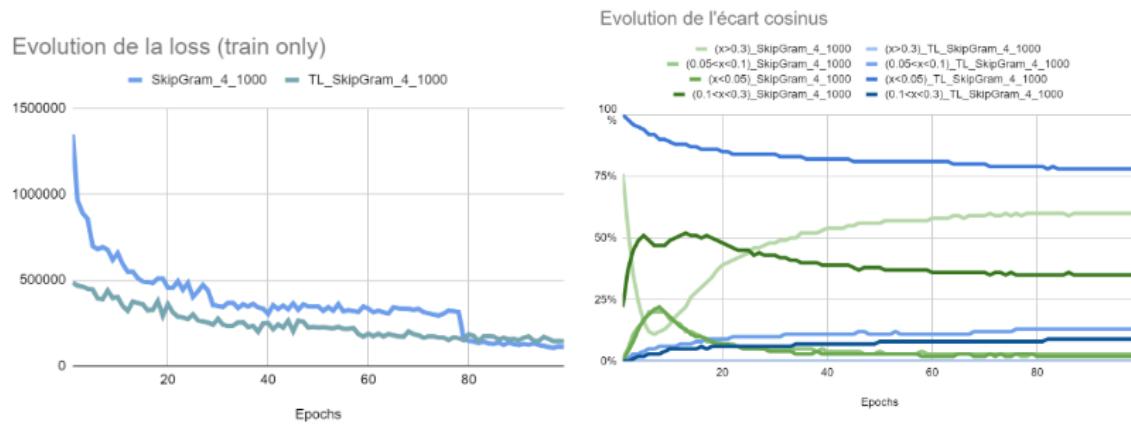


Figure 32: Comparaison des métriques Loss et Evolution écart cosinus pour un modèle de Transfer Learnig avec un modèle sans

De toutes ces expériences plusieurs apprentissages ressortent et permettront d'évaluer notre modèle en production. D'une part, il semblerait que le modèle CBOW soit plus performant que Skip-Gram. De plus, la métrique de différence cosinus a peu de chance d'être bonne si les corpus comparés sont très différents. Il est donc souhaitable de recourir à d'autres métriques comme la corrélation ou la comparaison de la liste de mots associée à un mot. De plus, ces expériences ont permis de mettre en avant qu'il était possible d'apprendre à l'aide de différentes méthodes simples. Enfin, la méthode Word2Vec entraînée sans transfer learning semble obtenir un optimum autour de 5 epochs. Cette élément indique, qu'un entraînement long n'aura très certainement pas d'effet positif et qu'il n'est donc pas nécessaire de réaliser un apprentissage long. La prochaine partie mettra en avant le processus retenu pour calculer la similarité entre mots d'une manière efficace et précise. Cette partie ne détaille pas les expériences réalisées avec GloVe, cette méthode s'est avérée peu pertinente, en effet la valeur d'entrée est une matrice de co-occurrences utilisant une grande quantité de RAM et aucun modèle de référence pertinent n'a été trouvé pour comparer notre modèle à la manière de Word2Vec.

8.4.2 Méthode retenue

L'objectif de cette partie est d'exposer la méthode de similarité implémentée au cours de la seconde phase de l'étude. En effet, pour rappel, après avoir choisi la liste de recherches Amazon, les pages produits associées sont scrapées et le corpus obtenu est nettoyé de sa donnée parasite. Aucun stemming n'est effectué dans le but d'avoir des métriques fiables. En effet, le modèle de référence de Fauconnier n'est ni lemmatisé ni stommatisé. De plus, dans cette phase empirique de plusieurs mois, trois modèles distincts génèrent trois recommandations de similarité. Un premier modèle regroupe uniquement les données scrapées pour l'étude en question, un min_count de 1, un modèle CBOW et une dimension de 1000. Le second modèle est un modèle de transfer learning effectué avec le modèle de Fauconnier. Enfin le dernier modèle s'entraîne par transfer learning effectué sur notre modèle Amazon. Ce dernier contient l'ensemble des pages produits jusqu'à ce jour. Comme expliqué lors de la partie consacrée à la présentation de l'outil. L'objectif est de développer un modèle pertinent sur l'ensemble du corpus Amazon. Ce choix de trois modèles est provisoire, en effet une fois que notre modèle sera assez précis, il sera possible de l'utiliser seul.

A la fin de notre étude trois modèles sont donc générés. Chaque modèle crée une recommandation de mots clefs que nous détaillerons dans la partie consacrée à la démonstration de l'outil. Au vu des expérimentations réalisées au-dessus, le modèle Word2Vec composé uniquement du corpus de l'étude est entraîné sur 10 epochs. Ceci est assez arbitraire mais provisoire. Le second modèle est entraîné sur le modèle Fauconnier pour limiter la dégradation de la métrique, l'entraînement se fait sur 20 epochs. Enfin, le même nombre d'epochs est choisi sur le dernier modèle.

En amont, il faut donc entraîner de façon pertinente notre modèle Amazon. Pour se faire, deux cas pratiques vont être exposés. Les métriques choisies sont aux nombres de trois. A l'aide de trois graphes, il sera alors possible de choisir le modèle à conserver, en effet des checkpoints sont implémentés pour sauvegarder les modèles où les métriques s'améliorent. La première métrique est la corrélation cosinus. La seconde étudie la différence de similarité cosinus entre de paires de mots. Enfin, la dernière compare la liste des 5 mots les plus similaires à un mot, cette liste est comparée à celle générée par notre modèle pour un mot identique. Pour exposer la démarche scientifique, deux corpus sont exploités. L'objectif est de comprendre l'évolution et l'amélioration des métriques suite à l'agrandissement du corpus. Nos métriques sont générées grâce à notre modèle de référence Fauconnier. Deux fichiers sont créés, l'un avec 5000 paires de mots et leur similarité cosinus et l'autre avec 3000 mots et leur liste de 5 mots similaires. Les deux datasets utilisés ont été réalisés en scrappant 15 000 pages Amazon. Ce tableau montrent les différences entre nos deux datasets.

| | 15.000 pages produits | | 10.500 pages produits | |
|--------------------|-----------------------|---------|-----------------------|---------|
| | Avant | Après | Avant | Après |
| Caractères | 13,909,377 | 377,403 | 9,180,189 | 249,086 |
| Caractères uniques | 7,881,722 | 151,798 | 5,282,432 | 123,646 |

| min_count | 5 | 10 | 20 | 30 | 50 | 100 |
|-----------------------|--------|--------|--------|--------|--------|-------|
| 15.000 pages produits | 45,399 | 30,409 | 20,563 | 16,166 | 11,924 | 7,636 |
| 10.500 pages produits | 37,052 | 24,738 | 16,437 | 12,768 | 9,318 | 5,896 |

Figure 33: Présentations des deux datasets : calcul du nombre de mots uniques après et avant data preprocessing et évolution du nombre de mots uniques suivant le min_count

Pour rappel, l'objectif est d'obtenir un modèle semblable à celui de Fauconnier mais entraîné depuis le corpus d'Amazon et non de Wikipédia. Il a donc été décidé de scraper régulièrement de la donnée d'Amazon. Le but à long terme est de pouvoir entraîner un modèle Word2Vec avec les mêmes hyperparamètres que celui de Fauconnier. On souhaite donc avoir un min_count de 100 et un nombre associé de mots uniques de 66.000. Dans un premier temps nous nous intéresserons à l'analyse du meilleur modèle CBOW ou Skip-Gram pour un corpus identique. Ensuite, nous observerons l'évolution des métriques pour la configuration choisie entre le corpus complet et l'autre avec un tiers de données en moins.

Cette première comparaison montre qu'un modèle CBOW est plus pertinent. En effet, tous les indicateurs lui sont favorables. La part de mots ayant 0 mot en commun est beaucoup plus faible, la corrélation maximum est obtenue plus vite avec un niveau plus proche de 1. Enfin, l'évaluation de la différence cosinus montre une proportion de mots plus importante dans l'intervalle 0.1 et 0.3 que celui entre 0.3 et 0.5. Ces résultats confirment les éléments lus dans la littérature, en effet Skip-Gram est reconnu pour mieux marcher sur des grands corpus tandis que CBOW lui est pertinent sur les petits. Il faudra donc continuer d'étudier ces métriques à chaque entraînement du modèle. De plus, il est important de noter que l'on peut choisir le meilleur modèle à l'aide de checkpoints. Lors de l'entraînement si une des métriques s'améliore alors le modèle est sauvegardé. Le premier graphe à gauche met en avant un optimum atteint dès la 17ème epoch, le second graphe montre une stagnation dès la 5ème epoch et enfin le dernier se détériore dès la 3ème epoch. Ces observations permettent de choisir le modèle entraîné à la 17ème. En effet, la corrélation est identique à l'optimum et il est plus intéressant de maximiser le nombre de mots en commun dans le top 5 que de favoriser une distance. La distance est très sensible à la composition du corpus tandis que la similarité est la fonctionnalité finalement utilisée pour prédire des similitudes entre mots pour l'étude. Ce modèle permettra alors de réaliser du transfer learning pour nos études.

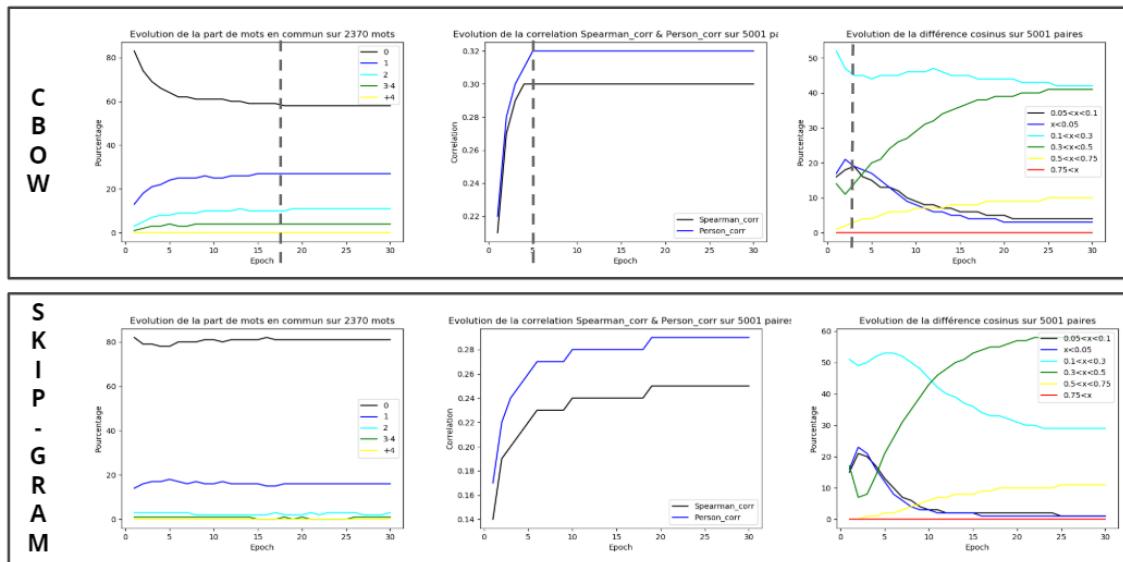


Figure 34: Comparaison des méthodes CBOW et Skip-Gram pour un corpus identique

Dans ce second point, le but est de voir l'évolution des métriques pour le modèle CBOW pour un corpus de 10.500 pages produits et un autre de 15.000 pages produits. Ces deux expériences montrent un apprentissage plus rapide lorsque le corpus est plus grand. En effet, chaque optimum est atteint en moins de temps. En outre, on remarque une légère amélioration des métriques sur le graphe de gauche et droite pour le corpus avec plus de mots. Ces observations encouragent le fait de vouloir construire un corpus constitué de plusieurs dizaines de milliers de pages produits. L'objectif d'un `min_count=100` avec 66.000 mots peut améliorer notre prédiction. Enfin, le temps d'apprentissage n'a pas beaucoup évolué entre les deux expériences, on remarque une augmentation de 10% du temps de calcul.

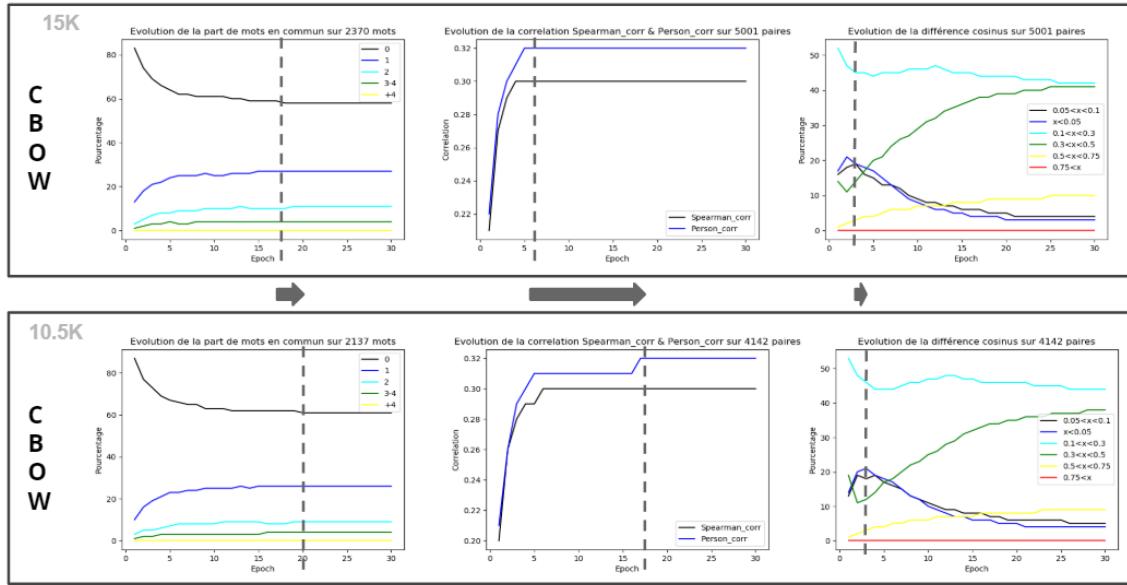


Figure 35: Évolution des métriques pour la méthode CBOW avec un corpus 33% plus grand

9 Réduction de dimension

La réduction de dimension trouve son intérêt dans la visualisation de la donnée. En effet, elle permet de se rendre compte dans un plan orthonormé 2D ou 3D. Dans ce projet, nous souhaitons visualiser la proximité entre plusieurs mots. Comme vu précédemment les différents algorithmes d'intelligence artificielle fournissent une représentation vectorielle d'un mot correspondant à la dernière couche cachée. La dimension est un paramètre à préciser au lancement de l'apprentissage. Cette dimension est supérieure à 3 donc il faut la réduire pour la visualiser informatiquement par l'homme. Cette partie propose d'aborder plusieurs techniques pour réduire la dimension d'un vecteur. Un site propose de visualisée de la donnée à l'aide de la bibliothèque *tensorflow* de façon automatique. Ce site [34] pourra être utilisé dans la partie expérimentale de ce projet. Dans cette section deux méthodes seront présentées PCA et TSNE. L'une est une approche linéaire ayant pour objectif de séparer au mieux les données. L'autre est une approche probabiliste essayant de grouper au mieux les données. Ces deux méthodes ont leurs avantages et seront utilisées pour visualiser la similitude entre les données. Dans la suite une approche mathématique de chacune des méthodes sera présentée.

9.1 Différentes méthodes de réduction de dimensions

9.1.1 PCA

Cette partie reprend les éléments présents dans ces sources : [9]. Cette méthode linéaire, nommée Principal Component Analysis, permet de réduire la dimension d'un jeu de données en supprimant les caractéristiques les moins importantes. Elle possède deux utilisations principales, réduire le temps d'apprentissage par un algorithme sur un jeu de données ou pour visualisation de la donnée. Cette technique a pour but de conserver au mieux les variations du jeu de données tout en réduisant le nombre de caractéristiques.

Les différentes étapes du PCA sont les suivantes. Dans un premier temps les labels sont retirés de la donnée. Ensuite la moyenne de chaque dimension est calculée. La matrice de covariance est calculée. Puis chaque vecteur propre est calculé pour chaque valeur propre. Dans le cadre d'une réduction à 3 dimensions, on conserve uniquement les 3 vecteurs propres avec les valeurs propres les plus élevées. Ainsi on obtient une matrice avec n observations et 3 caractéristiques.

Voici un exemple pour illustrer la méthode PCA. Cette exemple utilise un échantillon de la donnée nommée *iris_dataset*. On retrouve très facilement ces données sur le web.

| Caractéristiques | | | | Label | 5.PCA | | $PCA = W * VP$ |
|------------------|-------------|--------------|-------------|-----------------|--------|-------|----------------|
| sepal length | sepal width | petal length | petal width | | PC 1 | PC 2 | |
| 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa | -27.99 | 7.34 | |
| 5 | 3.6 | 1.4 | 0.2 | Iris-setosa | -31.66 | 7.41 | |
| 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa | -33.98 | 8.61 | |
| 5 | 3.4 | 1.5 | 0.2 | Iris-setosa | -30.65 | 7.67 | |
| 5.6 | 3 | 4.5 | 1.5 | Iris-versicolor | -27.80 | 16.33 | |
| 6.2 | 2.2 | 4.5 | 1.5 | Iris-versicolor | -25.77 | 17.00 | |
| 5.6 | 2.5 | 3.9 | 1.1 | Iris-versicolor | -26.08 | 14.65 | |
| 5.9 | 3.2 | 4.8 | 1.8 | Iris-versicolor | -29.23 | 17.55 | |
| 5.8 | 2.8 | 5.1 | 2.4 | Iris-virginica | -26.26 | 18.80 | |
| 6.4 | 3.2 | 5.3 | 2.3 | Iris-virginica | -30.01 | 19.64 | |
| 6.5 | 3 | 5.5 | 1.8 | Iris-virginica | -29.75 | 19.71 | |
| 7.7 | 3.8 | 6.7 | 2.2 | Iris-virginica | -36.28 | 23.80 | |
| 5.81 | 3.14 | 3.87 | 1.30 | 1.Moyenne | | | |

| 2.Calcul de la matrice de covariance : $COV(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$ | 4.Vecteurs propres | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|--------------|--------------|-------------|------|------|------|------|------|------|-------|-------|------|-------|------|------|------|-------|------|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|------|-------|------|-------|-------|------|------|------|------|
| <table border="1"> <thead> <tr> <th>sepal length</th> <th>sepal width</th> <th>petal length</th> <th>petal width</th> </tr> </thead> <tbody> <tr><td>0.63</td><td>0.01</td><td>1.26</td><td>0.52</td></tr> <tr><td>0.01</td><td>0.23</td><td>-0.22</td><td>-0.11</td></tr> <tr><td>1.26</td><td>-0.22</td><td>3.16</td><td>1.40</td></tr> <tr><td>0.52</td><td>-0.11</td><td>1.40</td><td>0.67</td></tr> </tbody> </table> | sepal length | sepal width | petal length | petal width | 0.63 | 0.01 | 1.26 | 0.52 | 0.01 | 0.23 | -0.22 | -0.11 | 1.26 | -0.22 | 3.16 | 1.40 | 0.52 | -0.11 | 1.40 | 0.67 | <table border="1"> <thead> <tr> <th>PC 1</th> <th>PC 2</th> </tr> </thead> <tbody> <tr><td>-3.04</td><td>0.92</td></tr> <tr><td>-4.82</td><td>-0.15</td></tr> <tr><td>0.48</td><td>2.25</td></tr> <tr><td>1.00</td><td>1.00</td></tr> </tbody> </table> | PC 1 | PC 2 | -3.04 | 0.92 | -4.82 | -0.15 | 0.48 | 2.25 | 1.00 | 1.00 |
| sepal length | sepal width | petal length | petal width | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.63 | 0.01 | 1.26 | 0.52 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.01 | 0.23 | -0.22 | -0.11 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1.26 | -0.22 | 3.16 | 1.40 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.52 | -0.11 | 1.40 | 0.67 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PC 1 | PC 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| -3.04 | 0.92 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| -4.82 | -0.15 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.48 | 2.25 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1.00 | 1.00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3.Valeurs propres $\lambda_1=0.022$ $\lambda_2=0.085$ $\lambda_3=0.281$ $\lambda_4=4.308$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

$$\det(A - \lambda I) = 0$$

Figure 36: Illustration mathématique de la méthode PCA

Enfin, dans le cadre de ce projet nous utiliserons la bibliothèque `sklearn.decomposition` et le code suivant. Le code suivant reprend l'article [13].

```

1 from sklearn.decomposition import PCA
2 pca = PCA(n_components=2)
3 points = pca.fit_transform(x)
4 print("Répartition de la variation entre les deux caractéristiques :", pca.
      explained_variance_ratio_)
5 #Répartition de la variation entre les deux caractéristiques : [0.72770452 0.23030523]
6
7 plot_iris_2d(
8     x = points[:,0],
9     y = points[:,1],
10    title = 'Iris dataset visualized with PCA')

```

9.1.2 T-SNE

T-Distributed Stochastic Neighbor Embedding est un algorithme de réduction de dimension développé par Lauren van der Maaten et Geoffrey Hinton. Cette section reprend les informations contenues dans ces articles : [32] et [25]. Cette méthode est une méthode non linéaire, elle repose sur le calcul de similarité entre deux points du jeu de données à l'aide d'une approche probabiliste. La similarité entre les points x_i et x_j est la probabilité conditionnelle, notée $p_{j|i}$. Cela mesure la proximité entre x_i et x_j en considérant une loi normale centrée en x_i et de variance σ_i^2 . Cette variance est calculée pour chaque point de telle sorte que la variance soit élevée pour des points proches. On écrit $(p_{i|i} = 0)$:

$$p_{j|i} = \frac{\exp(-|x_i - x_j|^2 / 2\sigma_i^2)}{\sum_{k \neq j} \exp(-|x_i - x_k|^2 / 2\sigma_i^2)}$$

D'une façon générale, deux points proches auront une probabilité conditionnelle élevée. De façon similaire, on appelle $q_{j|i}$ la probabilité conditionnelle de similarité entre y_i et y_j dans l'espace de moindre dimension. On choisit une variance égale à $\frac{1}{\sqrt{2}}$ et on écrit $(q_{i|i} = 0)$:

$$q_{j|i} = \frac{\exp(-|y_i - y_j|^2)}{\sum_{k \neq j} \exp(-|y_i - y_k|^2)}$$

L'objectif du TSNE est de représenter la même similarité dans un espace de grande et de petite dimension donc $q_{j|i} \simeq p_{j|i}$. Une métrique est communément utilisée, elle se nomme Kullback-Leibler divergence. Elle permet de mesurer la divergence entre tous les points du jeu de données à l'aide d'une méthode de descente de gradient. On écrit la fonction de coût ainsi :

$$C = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

P_i représente la distribution de probabilité conditionnelle sur tous les points pour le point x_i , de la même manière pour Q_i concernant les points dans l'espace de plus petite dimension pour y_i . Ensuite il suffit de dériver la fonction de coût pour obtenir cette formulation :

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i}) + p_{i|j} - q_{i|j})(y_i - y_j)$$

Il ne reste plus qu'à mettre à jour la valeur de nos données dans l'espace de plus petite dimension. Dans la formule ci-dessous on retrouve η le learning rate et $\alpha(t)$ le momemtum à l'itération t . Ce dernier est utilisé à des fins d'optimisation algorithmique.

$$y^{(t+1)} = y^{(t)} + \eta \frac{\delta C}{\delta y} + \alpha(t)(y^{(t)} - y^{(t-1)})$$

L'algorithme s'arrête dès que la différence maximale entre deux itérations est inférieure à une valeur epsilon proche de zéro. Enfin, dans le cadre de ce projet nous utiliserons la bibliothèque `sklearn.manifold` et le code suivant.

```

1 from sklearn.manifold import TSNE
2 RANDOM_STATE = 42
3 tsne = TSNE(n_components=2, n_iter=1000, random_state=RANDOM_STATE)
4 points = tsne.fit_transform(iris[features])
5 plot_iris_2d(x = points[:, 0], y = points[:, 1],
6                 title = 'Iris dataset visualized with t-SNE',
7                 xlabel="1st dimension", ylabel="2nd dimension")

```

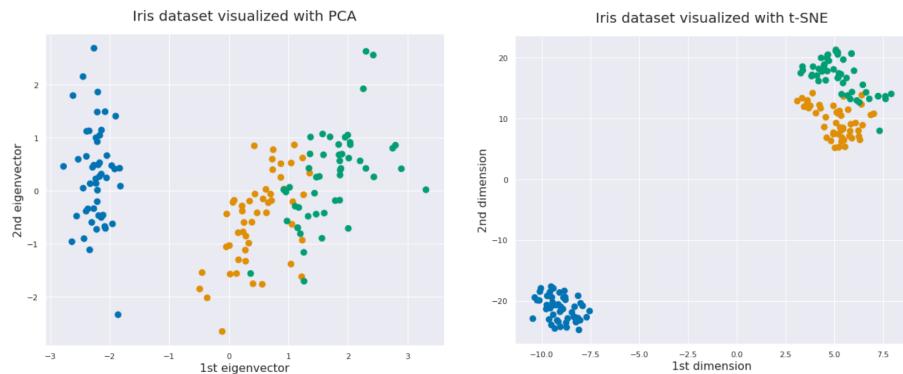


Figure 37: Visualisation d'un même dataset suivant PCA et t-SNE

9.2 Méthode de réduction retenue

La méthode de réduction n'est pas indispensable dans la conception de l'outil et pour déterminer un modèle de machine learning. En effet, cela permet de visualiser l'environnement d'un mot. Cependant, un exemple ne garantit pas la véracité d'un modèle. Les trois métriques présentées dans la partie précédente suffisent pour choisir le modèle à mettre en production. Enfin, lors de la phase expérimentale, j'ai été amené à utiliser TSNE pour afficher un graphe à chaque epoch. Un checkpoint affiche les 30 mots les plus proches d'un mot en entrée. Cette visualisation permet d'observer le résultat fourni par Word2Vec à chaque epoch. Cette représentation est souvent utilisée pour expliquer le fonctionnement de Word2Vec.

```
1 from sklearn.manifold import TSNE
2 import matplotlib.pyplot as plt
3 from gensim.models import Word2Vec, KeyedVectors
4 from mpl_toolkits.mplot3d import Axes3D
5
6
7 def tsne_plot(word, words, number):
8     "Creates and TSNE model and plots it"
9     model = KeyedVectors.load_word2vec_format("Amazon_models/word2vec_Cbow.bin", binary=True)
10    labels, tokens = [], []
11    tokens.append(model[word])
12    labels.append(word)
13    for simil in model.wv.most_similar(word, topn=number):
14        tokens.append(model[simil[0]])
15        labels.append(simil[0])
16    tsne_model = TSNE(perplexity=40, n_components=3, init='pca', n_iter=2500,
17                      random_state=23)
18    new_values = tsne_model.fit_transform(tokens)
19    x, y, z = [], [], []
20    for value in new_values:
21        x.append(value[0])
22        y.append(value[1])
23        z.append(value[2])
24    ax = plt.figure(figsize=(16, 16)).gca(projection='3d')
25    ax.scatter(x, y, z)
26    for i in range(len(x)):
27        if labels[i] == word:
28            ax.text(x[i], y[i], z[i], labels[i], size=14, zorder=1,
29                    color='green')
30        elif labels[i] in words:
31            ax.text(x[i], y[i], z[i], labels[i], size=14, zorder=1,
32                    color='blue')
33        else :
34            ax.text(x[i], y[i], z[i], labels[i], size=10, zorder=1,
35                    color='k')
36    plt.show()
37
38 if __name__ == '__main__':
39     tsne_plot("aspirateur", ["aspirer", "nettoyeur"], 30)
```

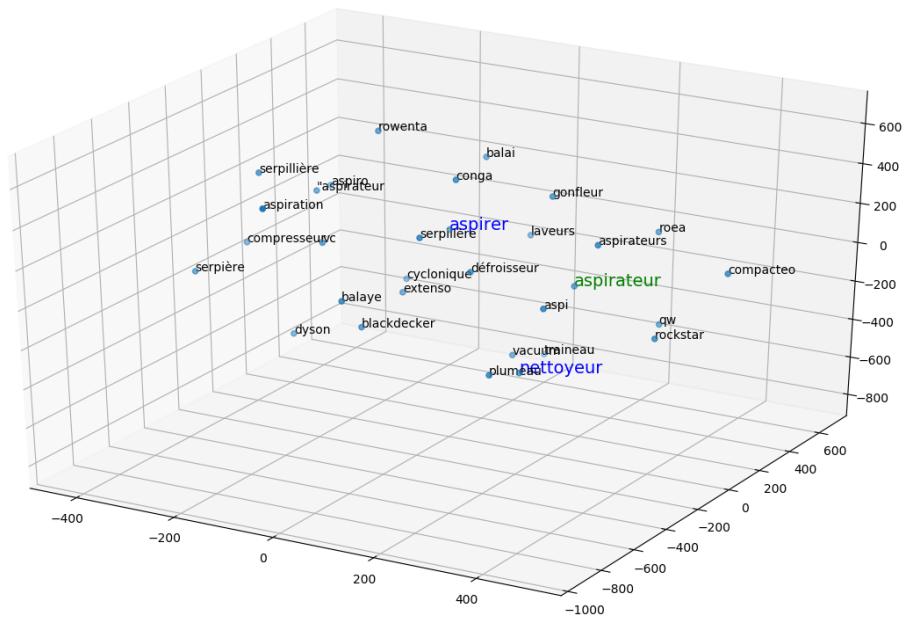


Figure 38: 30 mots les plus proches du mot "aspirateur" et visualisation des mots "aspirer" et "nettoyer" suite à l'apprentissage d'un modèle par Word2Vec

10 Démonstration de l'outil

Cette partie a pour but de présenter l'outil dans son implémentation finale. Bien entendu des améliorations et des changements opéreront suite aux remontées des clients ou des collaborateurs. En effet, cet outil a une visée commerciale, il sera utilisé directement par le client ou par un collaborateur de l'équipe marketing. L'étude se lance à partir d'un spreadsheet. Ce spreadsheet offre un template pour l'exécution de l'étude.

Tout d'abord, l'utilisateur ouvre le spreadsheet et arrive sur l'onglet nommé "Information". Il doit faire une copie de ce template, ensuite sur l'onglet "Information", il doit renseigner son mail ainsi que l'url Amazon où figure la page produit de son article. Puis, il suffit d'appuyer sur le bouton correspondant au script numéro 1. Cette première étape envoie un appel à l'API de Cloud Run et exécute les étapes sur lesquelles nous repasserons lors de la partie consacrée à l'automatisation. Cette première étape dure moins de 20 minutes et génère plusieurs onglets. Un premier nommé "Product Page Info" regroupe l'ensemble des informations récupérées depuis la page produit. On retrouve la liste des éléments énumérés dans la partie consacrée à l'extraction de mots-clefs. Un second onglet nommé "Relevant Researches" donne une liste de recherches Amazon pertinentes. Cette liste est donnée grâce à la méthode statistique décrite précédemment et regroupe 25 recherches Amazon résultantes de l'outil de suggestion d'Amazon. Cette première partie débouche donc sur une étape manuelle où l'utilisateur doit choisir les recherches Amazon. Ces recherches généreront l'étude concurrentielle. Pour faciliter le choix plusieurs KPI ont été ajoutés suite à un pré-scraping de l'ensemble des recherches recommandées. Ces KPI sont aux nombres de 4, ils permettent de savoir en fonction d'une recherche si notre produit est présent sur la première page, le nombre de pages produits présentes sur la première page, le nombre de liens sponsorisés et le nombre total de produits correspondants à la recherche. D'un point de vue marketing il serait intéressant d'étudier la concurrence sur des recherches pertinentes où le produit est absent de la première page. Enfin, un dernier onglet se nomme : "Collected information", il rassemble l'ensemble des pages produits scrapées sur la première page de chaque liste de produits. On retrouve le classement du produit dans la page, son identifiant ASIN, il est possible de savoir si c'est un lien sponsorisé, un produit labelisé Amazon meilleur choix ou Amazon meilleure vente. L'ensemble de ces informations sont capitales pour l'étude concurrentielle générée après avoir appuyé sur le second bouton. L'utilisateur reçoit donc un email dès que les résultats sont apparus. Il doit donc sélectionner la liste de recherches sur laquelle l'étude se basera. Il lui est possible d'ajouter d'autres recherches absentes de la liste.

The screenshot shows the SEO Amazon Tool interface. At the top, there's a header with tabs: 'Information', 'Keywords Recommendation', and 'Full-Collected Information'. Below the header, there's a section titled 'Read carefully :'. It contains instructions for using the tool, including steps to copy the URL of the product page, run the study, and receive an email with results. There are two main buttons: 'Run the study (Part 1)' and 'Run the study (Part 2)'. To the right of these buttons is a clock icon with the text 'environ 15 min' (approximately 15 minutes). Below the buttons, there's a section titled 'Please provide below information' with fields for 'Amazon product page' and 'The number of unique words in the URL' (set to 1000). A progress bar indicates 'The job has been searched at this date'. At the bottom of this section is a button labeled 'Applying sheet formatting'. The 'Product Page Info' tab is active, showing a table with columns: A (Challenger_product_page), B (Product_Title), C (ASIN), D (Challenger_brand), E (Consumer_rank), F (Number_of_comments), G (Product_price), and H (Feature_bulletin_info). One row is visible, detailing a Roventa vacuum cleaner. The 'Keywords Recommendation' tab is also visible at the bottom.

Figure 39: Résultats obtenus suite à l'exécution de la première étape

Une fois, ce choix effectué, l'utilisateur presse le second bouton. Cette action entraîne un appel API vers l'instance de Cloud Run. Les différentes étapes seront rappelées dans la partie consacrée à l'automatisation de l'outil. A la fin de cette exécution, un email est envoyé à l'utilisateur. Deux onglets ont été créés et l'onglet "Information" a été mis à jour. L'onglet nommé "Full-Collected Information" regroupe l'ensemble des informations sur les pages produits scrapées suivant la liste de recherches choisies. Ensuite un autre onglet "Keywords Recommandation" passe en revue l'ensemble des champs de notre page produit et propose des mots similaires à un mot de ce champ. Les mots recommandés se trouvent sur les autres pages produits dans le champ correspondant. Une séparation est effectuée entre les mots similaires présents à la fois sur la page produit des concurrents et ma page et ceux uniquement présents chez les concurrents. Dans un angle marketing, deux métriques permettent à l'utilisateur de savoir comment modifier la page produit. En effet, chaque mot présent dans mon annonce est associé à la fréquence d'apparition de ce mot, dans ce champ chez mes concurrents. L'autre métrique donne le nombre de fois où le mot apparaît. Il est donc intéressant de remplacer un mot de mon champ par un mot similaire non présent, si mon mot possède une fréquence faible et un nombre peu élevé d'occurrences. De plus, lorsque ma page ne possède pas un champ alors il est impossible de proposer des mots similaires. La méthode statistiques TF-IDF permet donc de proposer des mots présents chez les concurrents, une seconde ligne propose des groupes de mots. Il est alors possible de créer un champ à partir de ces informations.

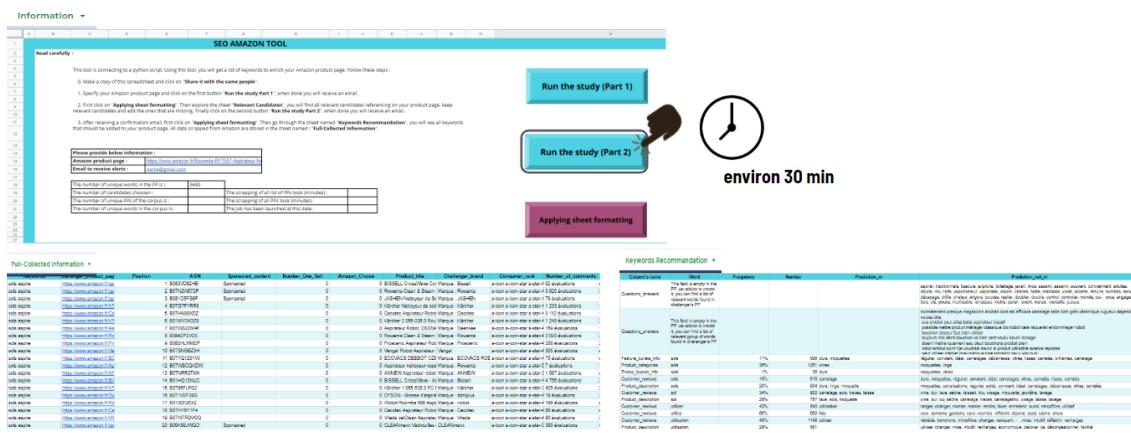


Figure 40: Résultats obtenus suite à l'exécution de la deuxième étape

Enfin, l'onglet "Information" se complète d'une partie nommée "Learnings" et détaille des informations relatives aux paramètres de l'analyse. Ce dernier se décrit sous la forme d'un tableau regroupant le nombre de mots uniques dans la page produit, le nombre de candidats choisis après la phase 1, le nombre de pages produits scrapées parmi les recherches Amazon et enfin le nombre de mots uniques dans le corpus. Un autre volet plus technique détaille le temps d'exécution de chaque phase et la date de l'étude. La photo ci-dessous détaille un exemple. Le cas pratique utilisé ici, est celui d'un produit concernant des couverts de table. On remarque que la page produit est composée d'un nombre restreint de mots, et que l'étude se base sur 41 recherches produisant 1036 pages produits uniques.

| | | | |
|-----------------------------------------------|-------|------------------------------------------------------------|------------|
| The number of unique words in the PP is : | 327 | The time to get the results of the first part (minutes) : | 20 |
| The number of candidates choosen : | 41 | The time to get the results of the second part (minutes) : | 54 |
| The number of unique PPs of the corpus is : | 1036 | The job has been launched at this date : | 2020-08-03 |
| The number of unique words in the corpus is : | 10470 | | |

Figure 41: Information détaillant les paramètres de l'étude

Ensuite, sur ce même onglet, il est possible d'explorer trois tableaux détaillant l'environnement concurrentiel du produit. Un premier cadran met en avant une comparaison par marque. Il est alors possible de comparer une marque par rapport à la notre à l'aide de plusieurs métriques. Seul le top 15 des marques est affiché, il est possible de connaître le prix médian, le nombre de commentaires médian, le classement du produit médian, le top 3 des catégories, le vendeur et l'expéditeur le plus fréquent, le nombre de fois où la marque apparaît en top vendeur Amazon ou top choix Amazon, le nombre de fois où la marque est sponsorisée et enfin la position médiane de la marque. Dans un objectif SEO, il est donc intéressant de comparer sa marque à une autre dont la gamme de prix est similaire et dont la marque est peu ou pas sponsorisée. De plus, si Amazon est expéditeur et vendeur du produit alors celui-ci est éligible à Amazon Prime et c'est un facteur très important pour la compétitivité d'un produit. En effet, la livraison est donc gratuite et rapide pour le consommateur. Enfin, en plus de cette comparaison par marque, il est possible

de visualiser un cadran par produit. On retrouve les mêmes KPIs avec un champ complémentaire détaillant le classement du produit selon Amazon. Chaque produit se voit attribuer un classement macro correspondant à la catégorie globale et micro pour la sous-catégorie. Il est alors possible de comprendre où se situe notre produit par rapport aux autres. L'exemple ci-dessous, met en avant que la marque "Couzon" est concurrente de "Villeroy Boch" et "Degrenne" avec une gamme de prix similaire. Cependant, il est intéressant de voir que le top produit comprend aussi un produit de la marque "Degrenne". Il est alors possible d'explorer de plus près cette annonce et de comprendre pourquoi celle-ci possède un meilleur classement Amazon que notre produit. Après étude, il semblerait que les informations contenues sont plus pertinentes que notre annonce et qu'il serait donc judicieux de compléter notre annonce.

| The Top 15 challenger brands | Number of displayed | Median Price | Median Position | Median Customer Rank | Median number of Consumer reviews | Top 3 Product Category | Number of Amazon label N°1 Seller - Amazon Choice | Top 1 Sender - Seller | Number of time(s) sponsored |
|------------------------------|---------------------|--------------|-----------------|----------------------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|-------------------------------------|-----------------------------|
| Marque : COUZON | 85 | 140 | 25 | 5 | 2 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Cuillères. Cuillères à café. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteaux. Porte-couteaux. | 0 - 0 | Amazon - Amazon. 20 - 20 | 0 |
| Amefa | 112 | 36 | 34 | 4 | 35 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteaux. Couteau à steak. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Fourchettes. Fourchettes de table. | 16 - 0 | Amazon - Amazon. 19 - 19 | 0 |
| Exact | 74 | 24 | 30 | 4 | 598 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Cuillères. Cuillères pour enfants. | 0 - 0 | Amazon - Exact. 16 - 16 | 3 |
| Pradel Excellence | 72 | 30 | 25 | 4 | 102 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Couverts et ustensiles de cuisine. Couteaux de cuisine. Ensemble de couteaux de cuisine. | 4 - 0 | Amazon - Amazon. 21 - 21 | 0 |
| WUJO | 60 | 28 | 12 | 4 | 218 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. | 0 - 0 | Amazon - Wujo. 7 - 7 | 0 |
| Laguiole Production | 58 | 28 | 17 | 4 | 150 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteaux. Couteau à steak. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Cuillères. Cuillères à thé. | 0 - 1 | Amazon - Amazon. 7 - 7 | 0 |
| Villeroy & Boch | 52 | 133 | 28 | 4 | 10 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteau. Couteau à steak. | 1 - 0 | Amazon - Amazon. 24 - 24 | 6 |
| AmazonBasics | 42 | 40 | 22 | 4 | 691 | Cuisine et Maison. Vaisselle et arts de la table. Services de vaisselle. Services de table. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Services de vaisselle. Services complets. | 8 - 0 | Amazon - Amazon. 17 - 17 | 24 |
| LAGUIOLE | 40 | 30 | 39 | 4 | 4 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteaux. Couteau à steak. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Fourchettes. Fourchettes de table. | 0 - 0 | Amazon - Amazon. 9 - 9 | 1 |
| COUZON | 38 | 82 | 35 | 5 | 3 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteau. Couteau de table. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Fourchettes. Fourchettes de table. | 0 - 0 | Amazon - Amazon. 21 - 21 | 0 |
| Raylans | 31 | 28 | 33 | 4 | 2 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Aménagement et décoration. Linen et linge de maison. Linge de table. Sets de table. | 0 - 0 | borate - borate. 5 - 5 | 1 |
| Elegant Life | 30 | 33 | 28 | 4 | 828 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteaux. Couteau à steak. | 0 - 0 | Amazon - HOBO-FR. 5 - 5 | 0 |
| Velaze | 30 | 38 | 31 | 4 | 24 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Couteaux. Couteau à steak. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Cuillères. Cuillères à thé. | 0 - 0 | Amazon - JOVIAL SOAR LIMITED. 6 - 6 | 6 |
| Paganohome | 29 | 64 | 27 | 4 | 8 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. | 0 - 0 | Amazon - paganohome. 3 - 3 | 0 |
| Buyer Star | 28 | 21 | 38 | 4 | 105 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Couverts servant ensembles. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Cuillères. Cuillères à café. | 0 - 0 | Amazon - Buyer Star. 14 - 14 | 1 |
| DEGRENNIE | 26 | 256 | 30 | 5 | 3 | Cuisine et Maison. Vaisselle et arts de la table. Couverts. Ménagères. Ménagères assorties. Cuisine et Maison. Vaisselle et arts de la table. Couverts. Cuillères. Cuillères à café. | 0 - 0 | Amazon - Amazon. 5 - 5 | 0 |

Figure 42: Environnement concurrentiel par marque

Figure 43: Environnement concurrentiel par produit

Un dernier tableau met en valeur des métriques importantes pour le suivi du produit avant et après la modification. Il est alors possible de voir que sur l'ensemble des recherches proposées, le produit est présent 36 fois en première page soit 34% du temps. De plus, le nombre de marques uniques parmi les 1036 pages produits est de 414 marques. De plus, Amazon possède deux configurations pour lister ses produits sur son site. Il est possible de les visualiser par 16 ou 56 produits par page, cela est indépendant de l'utilisateur et géré par l'algorithme d'Amazon. Il est donc intéressant de savoir le nombre fois où la liste est complète, lorsque mon produit est présent. Enfin, lorsque notre produit est présent en première page pour une recherche, il est intéressant de connaître le nombre médian de liens sponsorisés ainsi que le nombre médian de résultats générés par l'ensemble de ces recherches. Toutes ces caractéristiques doivent être maximisées suite à l'amélioration sémantique de la page.

| | |
|-----------------------------------------------------------------|----------------------------------------|
| The number of times the product is presented through candidates | 36 - (34.0%) |
| The number of challenger brands : | 526 |
| Stats when the product is presented on the first page | Number of page list fully fill |
| | Median of number of sponsored products |
| | Median of number of total products |

Figure 44: Métriques utilisées pour le suivi après et avant modification

Cet outil a récemment été utilisé pour une prospection client. L'apport de connaissances sur l'environnement concurrentiel permet de montrer à un nouveau client des éléments spécifiques à la marketplace d'Amazon en rapport avec sa marque. De plus à l'avenir il est envisagé de créer une version data studio permettant de visualiser de manière plus facile et personnalisée la donnée résultant de l'analyse. A ce jour, il n'a pas encore été possible d'évaluer les KPIs cités précédemment avant et après modification d'une page produit.

11 Automatisation de l'outil

11.1 Implémentation

Cette partie présente l'industrialisation de l'outil. Après avoir testé à plusieurs reprises le bon fonctionnement de l'outil en local et une fois le template d'étude validé, l'objectif est de rendre l'outil utilisable par un client ou un collaborateur dont les compétences ne permettent pas de comprendre du code. Cette étape, au moment de rendre mon mémoire n'est pas finalisée. En effet, comme le montre la démonstration du produit, nous sommes en cours de définition des bons KPI. Cependant, en parallèle de ces tests, un plan d'industrialisation a été préparé. La mise en production de l'application utilisera un système de route grâce à la bibliothèque *Flask*. Chaque route est une fonctionnalité de l'API. Cette application sera déployée au sein de l'univers Google Cloud Platform (GCP). Ce dernier offre des fonctionnalités vastes pour déployer une API et mettre en service une application complexe. Tout d'abord un projet nommé "M13h-amazon-SEO" a été créé. Ce compte est rattaché au compte de facturation de M13H. Un projet permet d'avoir accès à l'ensemble des services de GCP. Voici la liste des services utilisés.

- **Google Cloud Storage (GCS)** : ce service permet de stocker de la donnée dans le cloud
- **Cloud Identity and Access Management (IAM)** : un compte de service procure une clef pour accéder à l'ensemble des services de GCP
- **BigQuery** : est une base données pour le Big Data. Elle permet de stocker l'ensemble des données scrapées.
- **Cloud Run** : fournit des containers initialisés à l'aide d'une configuration docker. Cette fonctionnalité offre 2go de RAM et 15 min d'utilisation maximum. Chaque call API démarre un nouveau container.
- **Virtual Machine (VM)** : Ordinateur virtuel accessible depuis n'importe quel autre ordinateur. Il est possible de moduler la puissance et le stockage alloués à la machine.
- **Cloud Function (CF)** : permet d'exécuter du code à l'aide d'un appel API. Cet appel peut être automatique ou exécuté par un utilisateur. Ses fonctionnalités sont très similaires à Cloud Run, le temps d'exécution ne doit pas dépasser 9 minutes et ce service est composé de fonctions et ne se présente pas sous la forme d'un container.
- **Cloud Scheduler (CS)** : Service permettant de déclencher de façon automatique et programmée tout type d'exécution. Ce module sera utilisé pour déclencher une VM.
- **DataStudio** : similaire à spreadsheet il permet de visualiser d'une manière plus personnalisée et jolie les données. Son avantage repose sur sa solution dynamique permettant de comparer facilement des études sans avoir à changer de spreadsheet. L'accès à BigQuery est direct et très facile. Sur le marché, on trouve d'autres solutions comme la plus connue : Tableau.

Notre application repose sur 3 blocs. 2 éléments concernent l'analyse et le dernier permet d'entraîner notre modèle Amazon. Ce qui va suivre va donc détailler les différents appels générés suite à l'action de l'utilisateur depuis le spreadsheet. L'utilisateur de l'outil peut déclencher deux boutons :

Le premier bouton entraîne ces différentes étapes :

1. **scraping** : extraction de la donnée depuis la page produit du client

2. **data preprocessing** : traitement de la donnée par stemming et suppression des mots parasites
3. **group of Keywords scoring** : calcul d'un score pour chaque candidat à une recherche Amazon à l'aide d'une méthode statistique
4. **scraping** : récupération de statistiques sur les recherches recommandées pour aider l'utilisateur à choisir les recherches pertinentes. L'étude de la concurrence se basera uniquement sur le choix de ces recherches.

Les limitations de temps et de mémoire de Cloud Run obligent le développeur à faire preuve d'ingéniosité pour monitorer les différentes étapes d'exécution. Le croquis ci-dessous présente les étapes. Chaque rectangle correspond à un appel API vers le container de Cloud Run. Chaque container doit donc être exécuté en moins de 15min et avec moins de 2GO de RAM. Lors de l'exécution des étapes 1 et 4 des containers "main" sont ajoutés. Ceux-ci sont utiles car scraper une page prend en moyenne 5 secondes. Ce container va donc gérer la bonne exécution simultanée des scrapers et faire en sorte que celle-ci soit inférieure à 15min. De plus, chaque scraper génère une table temporaire sur BigQuery, toutes ces tables seront regroupées par "main". Les autres étapes sont rapides et ne demandent pas plus de 15 min.

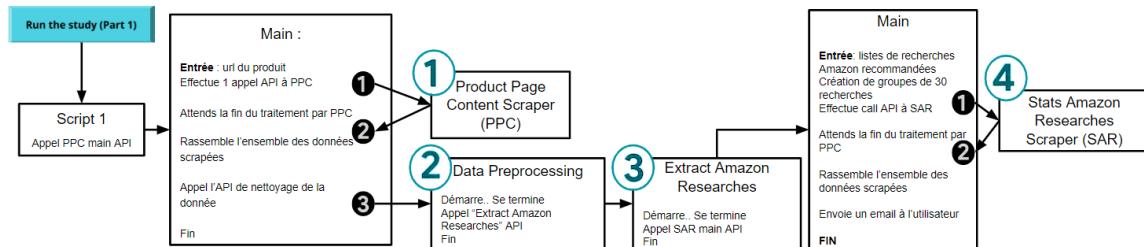


Figure 45: Architecture d'automatisation de la première étape sur Cloud Run

Le second bouton opère après que l'utilisateur ait effectué un choix de recherches Amazon. Ces recherches Amazon sont là pour générer la concurrence de notre produit. Voici les différentes étapes suite à l'exécution de la deuxième étape :

1. **scraping** : si l'utilisateur ajoute des recherches, celles-ci sont scrapées
2. **scraping** : extraction de la donnée depuis l'ensemble des pages produits correspondant aux recherches Amazon choisies
3. **data preprocessing** : traitement de la donnée par suppression des mots parasites
4. **keywords similarities** : calcul d'un score de similarité entre mots à l'aide de la méthode d'intelligence artificielle Word2Vec.

Les rectangles sont à interpréter de la même manière que précédemment et possèdent les mêmes fonctionnalités de regroupement de la donnée scrapée. Lorsque l'industrialisation sera réalisée un unique modèle sera entraîné lors de la quatrième phase. Le modèle Amazon de transfer learning sera utilisé pour effectuer du transfer learning.

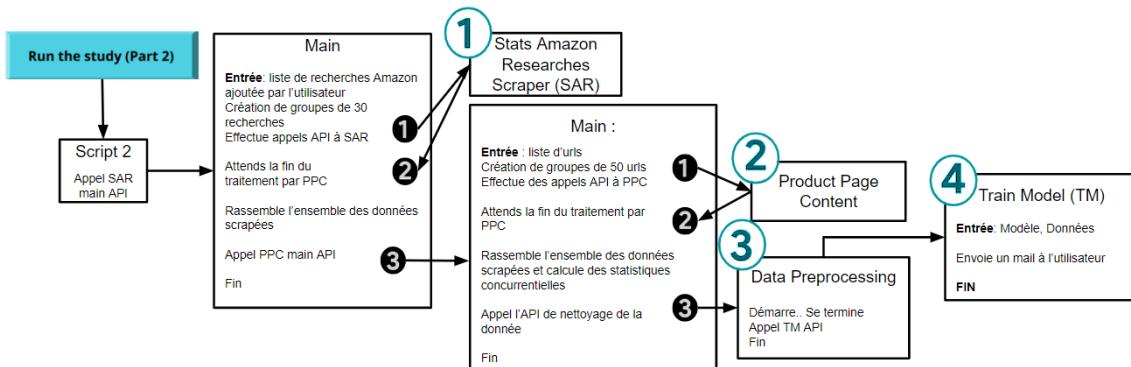


Figure 46: Architecture d'automatisation de la seconde étape sur Cloud Run

La dernière fonctionnalité à industrialiser est l'entraînement automatique de notre modèle Amazon. Cette exécution est effectuée une fois par semaine à l'aide du déclencheur automatique Cloud Scheduler. L'objectif est de récupérer des données fraîches et d'entraîner notre modèle seulement sur les données récupérées dans les 30 derniers jours. Ce modèle a pour but de rendre pertinente l'analyse de similarités effectuée dans la deuxième partie de l'analyse. La bibliothèque de Gensim permet d'entraîner sur des modèles très grands. Néanmoins le temps de calcul peut être élevé, et sera monitoré au cours de l'utilisation du modèle. Voici les différentes étapes :

1. **scraping** : récupération de l'ensemble des urls des pages catégories depuis la page d'accueil d'Amazon.
2. **scraping** : extraction des pages produits présentes sur les pages catégories
3. **scraping** : récupération du contenu des pages produits si la page produit est non présente dans notre base de données ou que son dernier scraping a été effectué il y a plus de 30 jours.
4. **data preprocessing** : traitement de la donnée par stemming et suppression des mots parasites
5. **model training** : entraînement du modèle avec trois checkpoints implémentés pour permettre à un data scientist de choisir le meilleur modèle.

Les différentes étapes de récupération décrites ci-dessus, prennent aux alentours de 12h. Une machine virtuelle sera donc utilisée de série N2 avec 7.5 GO de RAM, un CPU et 10GO de stockage. Celle-ci sera allumée lors de l'exécution du script par une Cloud Function précédemment déclenchée par un appel provenant de Cloud Scheduler. Le processus se déclenche à la suite d'une façon itérative.

Enfin, au cours de ces 3 modules, l'ensemble des données scrapées sont sauvegardées sur BigQuery, on les retrouve dans un dataset nommé "Amazon-SEO". Il contient aussi les tables utilisées pour prédire la position d'un produit Amazon dans une liste (partie 7). On retrouve aussi un dataset par étude nommé par l'ASIN du produit. Ce dataset regroupe les données présentes dans chaque spreadsheet. Ces données sont utiles dans le cas, où l'on souhaite migrer la visualisation de spreadsheet à DataStudio.

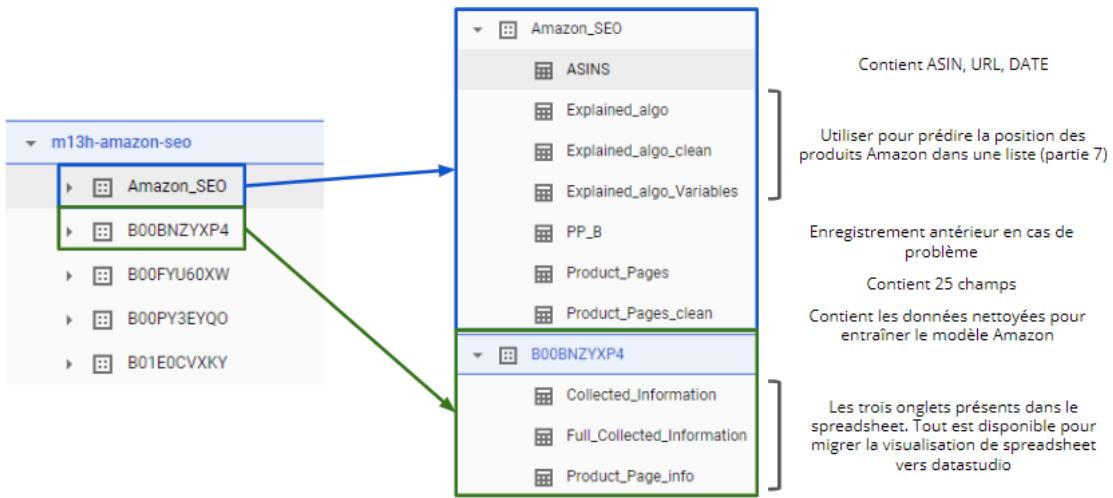


Figure 47: Tables BigQuery contenues dans le projet M13h-amazon-SEO

De plus, lors de l'entraînement d'un nouveau modèle, un dossier est créé dans le bucket nommé "amazon-model" de Google Cloud Storage. Il est défini par une date et contient l'ensemble des modèles enregistrés à l'aide des checkpoints. Le data scientist choisit à l'aide des graphes le meilleur modèle est le copie dans le dossier "Use_In_Production". Enfin, lors de l'exécution d'une étude, un dossier est créé, celui-ci donne accès à trois graphes. Ils illustrent l'évolution des métriques par transfer learning (TL).

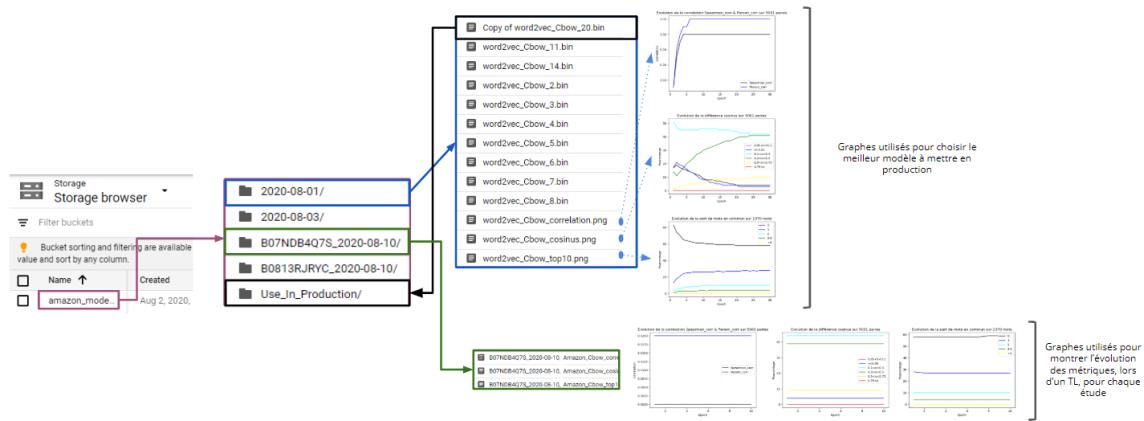


Figure 48: Documents enregistrés sur GCS pour le projet M13h-amazon-SEO

11.2 Coûts et performance

La phase d'industrialisation n'étant pas effectuée à ce jour, il est uniquement possible de donner une estimation des coûts. GCP offre une documentation complète à ce sujet [14], et propose un outil de monitoring appelé **Cloud Monitoring**. Les deux premiers scripts utilisent Cloud Run ce service est payant uniquement lors de son exécution, aucun coût n'est facturé si aucun appel n'est effectué. L'exécution d'une instance avec 2go de RAM coûte 0.000029\$ la seconde. Une cloud function est un peu moins chère. L'exemple présenté ci-dessous permet d'évaluer le coût d'une étude. L'exemple utilisé est générique, les paramètres sont souvent de cet ordre de grandeur. De plus, au cours de l'exécution l'API BigQuery est utilisée. Le coût pour un tera de données requêté est de 5\$. Dans notre cas, nous manipulons des mégaoctets (MO) de données. Donc ce coût est négligeable. Enfin GCS permet de stocker de la donnée, le coût est de 0.02\$ par GB par mois. Ce coût est aussi négligeable, car 10.000 pages produits regroupent 50 MO. Le tableau ci-dessous détaille l'ensemble des containers déclenchés au cours des deux phases d'une étude. Le temps moyen est donné et permet de donner une estimation du prix d'une étude. Le temps facturé est d'environ 3h pour une étude, néanmoins, beaucoup d'actions se font en même temps, cela réduit le temps réel à 45 min. Enfin, l'étude est facturée environ 25 centimes de dollars.

| | Paramètres | Temps (min) | Nombres de containers simultanés | Temps par paramètre |
|-----------------------------|-------------------|-------------|----------------------------------|---------------------|
| Main PPC | 1 url | 3 | 1 | |
| R PPC Scrapers | 1 url | 1 | 1 | 5 secondes |
| U Data Preprocessing | 1 Page Produit | 1 | 1 | 2 secondes |
| N Extract Amazon Researches | 1 Page Produit | 3 | 1 | |
| 1 Main SAR | 100 recherches | 7 | 1 | |
| SAR Scrapers | 30 recherches | 6 | 4 | 10 secondes |
| Main SAR | 10 recherches | 3 | 1 | |
| R SAR Scrapers | 10 recherches | 2 | 1 | 10 secondes |
| U Main PPC | 1000 url | 8 | 1 | |
| N PPC Scrapers | 50 urls | 5 | 20 | 5 secondes |
| 2 Data Preprocessing | 1000 Page Produit | 10 | 1 | 2 secondes |
| Train Model | 1 modèle | 10 | 1 | |
| Temps total (min) | | 45 | | |
| C Temps facturé (min) | | 172 | | |
| O Temps facturé (secondes) | | 10320 | | |
| U Temps facturé (secondes) | | | | |
| T Prix par seconde | | \$0.000024 | | |
| Prix Total | | \$0.25 | | |

Figure 49: Évaluation du temps et du coût d'une étude à l'aide de Cloud Run

L'entraînement du modèle s'effectue sur une Machine Virtuelle. Le temps d'exécution est très variable suivant le nombre de pages produits à scraper. Cela diffère suivant le nombre de produits ayant une date de scraping antérieure à 30 jours et le nombre de nouveaux produits répertoriés. Le temps de scraping pour une page web est de 5 secondes grâce à la bibliothèque de Selenium. Enfin, une estimation du temps d'exécution donne entre 12h et 24h par semaine, soit 4 jours par mois. Le module de calcul Google Cloud Pricing Calculator nous donne une estimation de 17\$ par mois. Ce temps de calcul serait équivalent sur une machine en local.

Sur le plan de la performance, l'exécution sur le cloud de la première partie de l'étude devrait prendre environ 14 minutes. La seconde étape devrait s'exécuter en moins de 30 minutes. Le temps sur une machine en locale est égale au temps facturé soit environ 3h. Enfin, le temps de scraping d'une page produit est de 5 secondes et le temps pour récupérer les statistiques et les liens sur les listes de produits prend 10 secondes par recherche Amazon.

12 Conclusion

Ce mémoire a été réalisé dans le but de mettre à profit l'ensemble des compétences acquises au cours du master IASD. Ce travail s'est consacré au domaine spécifique du traitement du langage en intelligence artificielle. L'outil finallement obtenu permet aux équipes de M13H et de Labelium d'explorer la concurrence et d'apporter une recommandation SEO à un client concernant leurs produits sur la marketplace d'Amazon. Ce mémoire regroupe la théorie dont découle l'ensemble des implémentations et il détaille l'implémentation sur GCP. Enfin, il permet de répondre aux problématiques exposées. En effet, pour extraire au mieux les mots les plus révélateurs d'un texte, j'ai utilisé deux techniques statistiques (TF-IDF et TextRank) et une technique SEO (Amazon Auto-complete). De plus, les modèles créés à partir de Word2Vec permettent de donner des similitudes pertinentes entre des mots, à l'aide d'une technique de transfer learning se basant sur un modèle entraîné sur des dizaines de milliers de pages produits récentes. Ces deux méthodes donnent lieu à un outil unique sur le marché et permettent à M13H de posséder un modèle Word2Vec performant sur les données d'Amazon.

Ces données ouvrent le champ des possibles. Il est alors envisageable de mieux comprendre l'algorithme de référencement A9 d'Amazon comme exposé dans la partie 7. Néanmoins, le scraping est une méthode nécessitant une constante attention car elle est très dépendante de la structure du site d'Amazon. En effet, un data scientist devra suivre les remontées et identifier des erreurs de scraping. Cette tâche pourra être facilitée par un système d'alertes par mail. L'outil développé au cours de ce mémoire évoluera au fur et à mesure de son utilisation. Son automatisation est en cours et permettra à un utilisateur d'obtenir une étude de concurrence et une recommandation SEO en moins de 30 minutes. Des améliorations comme l'ajout de la dimension mobile, et une migration vers DataStudio ont été identifiées. Enfin, l'outil permet d'enrichir le contenu d'une page web pour effectuer une recommandation SEO, cependant il est toujours important d'avoir une démarche globale et d'ajouter une dimension SEA. L'outil devra donc être pourvu d'un module SEA permettant de recommander sur l'achat de mots-clés pour être mieux référencé. Ces deux stratégies devraient garantir aux produits d'un client une présence dans les 5 premières annonces pour une recherche Amazon optimisée à l'aide de l'outil. En outre la méthode, exposée dans ce mémoire, est adaptée pour la market place d'Amazon, mais il est totalement possible de la transposée à toutes autres market places (Cdiscount, Darty..).

References

- [1] Akshay Kulkarni et Adarsha Shivananda. "Natural Language Processing Recipes". In: (2019), pp. 185–202. doi: 978-1484242667.
- [2] Akshay Kulkarni et Adarsha Shivananda. "Natural Language Processing Recipes". In: (2019), pp. 82–93. doi: 978-1484242667.
- [3] BrambleXu. *TextRank4Keyword.py*. URL: <https://gist.github.com/BrambleXu/3d47bbdbd1ee4e6fc695b0ddb88cbf> file-textrank4keyword-py. (accessed: 06.05.2020).
- [4] Tristan Cazenave. *Cours Apprentissage Profond, Master 2 IASD*.
- [5] Hackage company. *rake: Rapid Automatic Keyword Extraction (RAKE)*. URL: <https://hackage.haskell.org/package/rake>. (accessed: 03.03.2020).
- [6] JungleScout company. *Your all-in-one platform for selling on Amazon*. URL: <https://www.junglescout.com/>. (accessed: 30.03.2020).
- [7] Sellics company. *The Leading All-in-One Amazon Software specially built for Agencies*. URL: <https://sellics.com/agency-edition/>. (accessed: 30.03.2020).
- [8] RedHat. Dipanjan Sarkar. *Implementing Deep Learning Methods and Feature Engineering for Text Data: The Continuous Bag of Words (CBOW)*. URL: <https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-cbow.html>.
- [9] Akash Dubey. *The Mathematics Behind Principal Component Analysis*. URL: <https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643>. (accessed: 26.03.2020).
- [10] Stanford University School of Engineering. *Lecture 2 | Word Vector Representations: word2vec*. URL: <https://www.youtube.com/watch?v=ERibwqs9p38>. (accessed: 27.03.2020).
- [11] Jean-Philippe Fauconnier. *About*. URL: <https://fauconnier.github.io/>. (accessed: 18.07.2020).
- [12] Naruu et Gabriel Demarmiesse. *Usage of activations*. URL: <https://keras.io/activations/>. (accessed: 16.03.2020).
- [13] Michael Galarnyk. *PCA using Python (scikit-learn)*. URL: <https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60>. (accessed: 26.03.2020).
- [14] GCP. *Pricing*. URL: <https://cloud.google.com/run/pricing>.
- [15] Robin Hanna. *Amazon SEO: The Ultimate Guide to Higher Product Ranking in 2020 (With Tips Examples)*. URL: <https://sellics.com/blog-amazon-seo/>. (accessed: 30.03.2020).
- [16] Robin Hanna. *How to Optimize Your Amazon Product Listings*. URL: <https://sellics.com/blog-amazon-seo/>. (accessed: 06.04.2020).
- [17] Hafsa Jabeen. *Stemming and Lemmatization in Python*. URL: <https://www.datacamp.com/community/tutorials/stemming-lemmatization-python>. (accessed: 09.03.2020).
- [18] Christopher D. Manning Jeffrey Pennington Richard Socher. *GloVe: Global Vectors for Word Representation*. URL: <http://nlp.stanford.edu/pubs/glove.pdf>. (accessed: 30.03.2020).
- [19] Colette Joubarne and Diana Inkpen. *Comparison of Semantic Similarity for Different Languages using the Google n-gram Corpus and Second Order Co-Occurrence Measures*. URL: www.researchgate.net/publication/221442282_Comparison_of_Semantic_Similarity_for_Different_Languages_Using_the_Google_n-gram_Corpus_and_Second_Order_Co-occurrence_Measures/link/5435c7550cf2643ab9869a04/download. (accessed: 18.06.2020).

- [20] Rachel Koenig. *NLP for Beginners: Cleaning Preprocessing Text Data*. URL: <https://towardsdatascience.com/nlp-for-beginners-cleaning-preprocessing-text-data-ae8e306bef0f>. (accessed: 09.03.2020).
- [21] Sundar Krishnan. *Weight of evidence and Information Value using Python*. URL: <https://medium.com/@sundarstyles89/weight-of-evidence-and-information-value-using-python-6f05072e83eb>.
- [22] Natasha Latysheva. *Why do we use word embeddings in NLP?* URL: <https://towardsdatascience.com/why-do-we-use-embeddings-in-nlp-2f20e1b632d2>. (accessed: 21.03.2020).
- [23] Dive Into Deep Learning. *Word Embedding (word2vec)*. URL: https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html. (accessed: 30.03.2020).
- [24] Xu LIANG. *Understand TextRank for Keyword Extraction by Python*. URL: <https://towardsdatascience.com/textrank-for-keyword-extraction-by-python-c0bae21bcec0>. (accessed: 06.05.2020).
- [25] Laurens van der Maaten et Geoffrey Hinton. *Visualizing Data using t-SNE*. URL: <http://www.cs.toronto.edu/~hinton/absps/tsne.pdf>. (accessed: 26.03.2020).
- [26] Maryam Mohsin. *10 Amazon Statistics You Need to Know in 2020 (Amazon Stats Infographic)*. URL: <https://www.oberlo.com/blog/amazon-statistics>. (accessed: 30.03.2020).
- [27] Baiju Muthukadan. *Selenium with Python*. URL: <https://selenium-python.readthedocs.io/>. (accessed: 06.05.2020).
- [28] Sri Manikanta Palakollu. *Scrapy Vs Selenium Vs Beautiful Soup for Web Scraping*. URL: <https://medium.com/analytics-vidhya/scrapy-vs-selenium-vs-beautiful-soup-for-web-scraping-24008b6c87>. (accessed: 06.05.2020).
- [29] Martin Porter. *French stemming algorithm*. URL: <https://www.datacamp.com/community/tutorials/stemming-lemmatization-python>. (accessed: 09.03.2020).
- [30] Martin Porter. *French stemming algorithm*. URL: <https://spacy.io/models/fr>. (accessed: 09.03.2020).
- [31] Juan Ramos. *Using TF-IDF to Determine Word Relevance in Document Queries*. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf>. (accessed: 06.03.2020).
- [32] Cyrille Rossant. *An illustrated introduction to the t-SNE algorithm*. URL: <https://www.oreilly.com/content/an-illustrated-introduction-to-the-t-sne-algorithm>. (accessed: 26.03.2020).
- [33] Nick Cramer Stuart Rose Dave Engel and Wendy Cowley. *Automatic keyword extraction from individual documents*. URL: https://www.researchgate.net/publication/227988510_Automatic_Keyword_Extraction_from_Individual_Documents. (accessed: 03.03.2020).
- [34] Wikistat. *Embedding Projecor*. URL: <https://projector.tensorflow.org/>. (accessed: 16.03.2020).
- [35] Wikistat. *Réseaux de neurones*. URL: <https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-rn.pdf>. (accessed: 16.03.2020).
- [36] Charline Zeitoun. *Comment l'intelligence artificielle va changer nos vies*. URL: <https://lejournal.cnrs.fr/dossiers/comment-lintelligence-artificielle-va-changer-nos-vies>. (accessed: 15.03.2020).

13 Annexe

Annexe 1 : Implémentation TF-IDF (page : 17)

```
1 def TF_IDF(corpus, stopwords):
2     """
3         This is a homemade implementation of TF_IDF. A TF_IDF matrix is returned
4         for each sentence.
5             A sum for each word is made and it results in an overall corpus TF-IDF
6             per word
7                 :param corpus: list of sentences
8                 :param stopwords: list of stopwords
9                 :return dataframe that contains TF_score, IDF_score and TF_IDF_score per
10                word
11                """
12
13    corpus_int = []
14    for document in corpus:
15        if len(document) > 1:
16            corpus_int = corpus_int + document.split(' ')
17            corpus_unique = list(set(corpus_int))
18
19    TF = np.zeros((len(corpus), len(corpus_unique)))
20    IDF_int = np.zeros((len(corpus), len(corpus_unique)))
21    TF_IDF = np.zeros((len(corpus), len(corpus_unique)))
22
23    count = 0
24    for document in corpus:
25        if len(document) > 1:
26
27            document = document.split(' ')
28            for word in document:
29                if len(word) > 1 and word not in stopwords:
30                    TF[count][corpus_unique.index(word)] += 1
31                    IDF_int[count][corpus_unique.index(word)] = 1
32                    TF[count] = TF[count] / len(document)
33                    count += 1
34
35    count = 0
36    for document in corpus:
37        if len(document) > 1 and word not in stopwords:
38            document = document.split(' ')
39            for word in document:
40                IDF = np.log(len(corpus) / max(1, np.sum(IDF_int, axis=0)[corpus_unique.
41                index(word)]))
42                TF_IDF[count][corpus_unique.index(word)] = TF[count][corpus_unique.index(
43                word)] * IDF
44                count += 1
45
46    def matrix_to_df(matrix, corpus_unique, name):
47        SUM = [np.sum(matrix, axis=0)]
48        SUM = pd.DataFrame(data=SUM, columns=corpus_unique).transpose()
49        SUM.columns = [name]
50        SUM = SUM.sort_values(by=name, ascending=False, na_position='first').round(2)
51        return SUM[~np.isnan(SUM[name])].reset_index()
52
53    TF_IDF = matrix_to_df(TF_IDF, corpus_unique, "score_TFIDF")
54    TF = matrix_to_df(TF, corpus_unique, "score_TF")
55    IDF = matrix_to_df(IDF_int, corpus_unique, "score_IDF")
```

```

49     TF_IDF = TF_IDF.join(TF, how="inner", lsuffix='_caller').drop(columns=["index_caller"])
50     df = TF_IDF.join(IDF, how="inner", lsuffix='_caller').drop(columns=["index_caller"])
51     return df

```

Annexe 2 : Implémentation CBOW (page : 40)

```

1  #####CBOW implementation#####
2  from keras.preprocessing.text import Tokenizer
3  from keras.utils import to_categorical
4  import random
5
6  #https://realpython.com/python-keras-text-classification/#using-pretrained-word-
7  #embeddings
8  num_words = len(list(set([j for i in text_no_stem for j in i])))
9  tokenizer = Tokenizer(num_words=num_words)
10 tokenizer.fit_on_texts(text_no_stem)
11 random.shuffle(text_no_stem)
12
13 sentences_train = text_no_stem[:int(len(text_no_stem)*0.90)]
14 sentences_test = text_no_stem[int(len(text_no_stem)*0.90):]
15
16 X_train = tokenizer.texts_to_sequences(sentences_train)
17 X_test = tokenizer.texts_to_sequences(sentences_test)
18 vocab_size = len(tokenizer.word_index) + 1 # Adding 1 because of reserved 0 index
19
20 for word in sentences_train[0]:
21     print(' {}: {}'.format(word, tokenizer.word_index[word]))
22 print()
23
24 def create_label_train_set(data,window) :
25     d = {"value" : [], "label" : []}
26     for sentence in data :
27         for word_num in range(len(sentence)) :
28             d["label"] += [sentence[word_num]]
29             l = []
30             for i in range(1,window+1) :
31                 if word_num - i >= 0 and word_num - i < len(sentence) :
32                     l.append(sentence[word_num-i])
33                 else :
34                     l.append(0)
35                 if word_num + i >= 0 and word_num + i < len(sentence) :
36                     l.append(sentence[word_num+i])
37                 else :
38                     l.append(0)
39             d["value"] += [l]
40     return d
41
42
43 h = create_label_train_set(sentences_train , 4)
44 print(h["value"][:10] , h["label"][:10])
45
46
47 d1 = create_label_train_set(X_train , 4)
48 d2 = create_label_train_set(X_test , 4)

```

```

49
50 from keras.callbacks import Callback
51 from sklearn.metrics.pairwise import cosine_similarity
52
53 class NewCallback(Callback):
54     """ NewCallback descends from Callback
55     """
56     def __init__(self, model, X_train, encoding_dim, input_dim, epoch):
57         """ Save params in constructor
58         """
59         self.input_dim = input_dim
60         self.model = model
61         self.encoding_dim = encoding_dim
62         self.X_train = X_train
63         self.epoch = epoch
64         self.current_epoch = 0
65         self.l = {"inf_005" : [], "inf_01" : [], "inf_03" : [], "sup_03" : []}
66
67
68
69     def on_epoch_end(self, epoch, logs={}):
70         weights = self.model.get_weights()[0]
71         df = pd.DataFrame(data=weights[1:,1:],      # values
72                            index=weights[1:,0],    # 1st column as index
73                            columns=weights[0,1:])
74         similarity_df = pd.DataFrame(cosine_similarity(df))
75         inf_005, inf_01, inf_03, sup_03 = 0, 0, 0, 0
76         count = 0
77
78         for i,j,k in zip(model_8_word1_word2_score_3000["word1"],
79                            model_8_word1_word2_score_3000["word2"], model_8_word1_word2_score_3000["score"]):
79             try :
80                 t = np.abs(similarity_df[tokenizer.word_index[i]][tokenizer.word_index[j]]-
81 float(k))
82                 if t < 0.05 :
83                     inf_005 += 1
84                 elif t < 0.1 and t >= 0.05 :
85                     inf_01 += 1
86                 elif t < 0.3 and t >= 0.1 :
87                     inf_03 += 1
88                 else :
89                     sup_03 += 1
90                 count+=1
91             except :
92                 pass
93             self.current_epoch += 1
94             self.l["inf_005"] += [inf_005/count]
95             self.l["inf_01"] += [inf_01/count]
96             self.l["inf_03"] += [inf_03/count]
97             self.l["sup_03"] += [sup_03/count]
98             if self.current_epoch == self.epoch :
99                 pd.DataFrame(self.l).to_csv("cbow_"+str(self.epoch)+".csv")
100                print([round((inf_005/count)*100,0), round((inf_01/count)*100,0), round((inf_03/
count)*100,0), round((sup_03/count)*100,0)])
101
102    def generator(d, num_words, only_x=False):
103        j = 0
104        while True : # a generator must be infinite for Keras
105            y = to_categorical(d["label"][j:j+1000], num_classes=num_words)

```

```

104     x = np.array(d["value"][:j:j+1000])
105     j+=1000
106     if j+1000 >= len(d["label"]) :
107         j = 0
108         if only_x :
109             yield x
110             yield x, y
111
112
113 from keras.models import Sequential
114 from keras import layers
115 maxlen = 8
116 embedding_dim = 1000
117
118
119 import keras.backend as K
120 from keras.models import Sequential
121 from keras.layers import Dense, Embedding, Lambda
122 vocab_size = num_words
123
124 # build CBOW architecture
125 cbow = Sequential()
126 cbow.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxlen))
127 cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embedding_dim,)))
128 cbow.add(Dense(vocab_size, activation='softmax'))
129
130 cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=[ 'accuracy'])
131
132 # view model summary
133 print(cbow.summary())
134 batch_size = 1
135 epochs = 100
136 model_8_word1_word2_score_3000 = read_scorefile("/content/
    model4_word1_word2_score_5000_2020-07-24.txt", stem=False)
137 history = cbow.fit_generator((generator(d1,num_words)),
138                             epochs=epochs,
139                             verbose=True,
140                             validation_data=(generator(d2,num_words)),
141                             steps_per_epoch = len(d1["value"])//1000//batch_size,
142                             validation_steps = len(d2["value"])//1000//batch_size,
143                             callbacks = [NewCallback(cbow, generator(d1,num_words, True),
144                                         embedding_dim, vocab_size, epochs)] )
144 print(history.history)
145 weights = cbow.get_weights()[0][1:]
146 print(weights.shape)

```