CentraleSupélec

Recherches et bibliographie

Identification des thèmes d'un livre par NLP

Client : Timothée BOHÉ & Nicolas GAUDEMENT - Onepoint

Amine CHERIF HAOUAT
Tom LABIAUSSE
Cyrine NABI
Pierre OLLIVIER

Sommaire :

1 - Bref historique du NLP

Le **Traitement Automatique du Langage Naturel (TALN)** ou ***Natural Language Processing*** **(NLP)** porte sur la compréhension d'un langage naturel humain par une machine. Cette discipline prend naissance dans les années 40/50 avec les débuts de l'informatique moderne. Elle consiste alors principalement en la traduction de moins d'une centaine de phrases entre deux langues[T1]. Les années 60/70 voient l'émergence des premiers *chatbots* comme ELIZA développée entre 1964 et 1966 et qui simule un comportement d'un psychothérapeute particulier face à un patient en reformulant des affirmations en questions[T2]. Cependant, la véritable révolution du NLP se produit à la fin des années 1980 avec l'introduction d'algorithmes de *Machine Learning* et l'augmentation de la puissance de calcul. Aujourd'hui, la diversité des domaines de recherches associés au NLP s'est considérablement accrue et le *Deep Learning*, utilisé dans de nombreuses tâches, continue à faire preuve de ses performances.

2 - Topic Modeling[T3]

Les champs d'applications du NLP sont, de nos jours, très larges. On peut en particulier citer la traduction automatique, l'analyse de sentiments, les *chatbots*, les correcteurs de textes, les résumés automatiques, la classification de textes… Dans le cadre du projet d'identification automatique du thème d'un livre à partir de la lecture de son résumé, nous nous intéresserons tout particulièrement au ***Topic Modeling*** ou « modélisation de sujet ». Ce domaine du NLP se rapproche de la classification de textes (*Topic Classification*) selon différents thèmes tout en présentant une différence fondamentale. En effet, les algorithmes de *Topic Classification* se base sur des techniques d'apprentissage supervisé. Les textes fournies à la machine lors de sa phase d'entraînement doivent être au préalable associés aux différents thèmes qu'ils abordent. Au contraire, le *Topic Modeling* repose sur des méthodes d'apprentissage non-supervisés où seul le nombre de thèmes à identifier parmi un ensemble de textes est au préalable fixé. La machine tente alors de trouver à la fois la répartition optimale des thèmes entre les textes mais aussi des mots associés à chaque thème. De plus, le *Topic Modeling* permet d'associer plusieurs thèmes prédominants à un même texte, ce qui n'est pas réalisable avec un algorithme de *Topic Classification* qui n'affecterait qu'un seul thème à un texte. Durant ce projet, nous nous intéresserons donc aux algorithmes de *Topic Modeling* qui permettent d'exploiter de grandes quantités de données lors de la phase d'entraînement et donc d'accroître les capacités de compréhension acquises par la machine.

On dit d'une méthode de *Topic Modeling* qu'elle doit identifier les variables latentes d'un jeu de données. Ces variables latentes sont des informations qu'on ne peut pas directement lire à l'aide des données, typiquement les thèmes. Les variables latentes doivent alors être inférées grâce aux données à disposition justement à l'aide d'une méthode de *Topic Modeling*.

Un modèle boîte noir de *Topic Modeling* est défini par les éléments suivants :

- ENTREE : un corpus de textes et un nombre de thèmes à identifier

- SORTIE : plusieurs ensembles de mots correspondant chacun à un thème identifié par l'algorithme ainsi que des regroupements de textes selon les thèmes précédents

Tout problème de NLP se décompose en deux aspects essentiels :

- La **partie linguistique**, qui consiste à pré-traiter et transformer les informations en entrée en un jeu de données exploitable.
- La **partie apprentissage**, qui porte sur l'exécution d'algorithmes sur le jeu de données exploitable pour aboutir aux sorties désirées.
-

3 - Pré-traitement des données C3

Avant de pouvoir traiter nos données et en tirer les thèmes, il est impératif d'effectuer des étapes de pré-traitement ou de « preprocessing » sur les données textuelles brutes :

- **Normalisation** : On nettoie le texte brute pour avoir des formes canoniques. On enlève certains caratères comme les chiffres et les ponctuations et on effectue des modifications simple sur les caractères comme par exemple tout mettre en minuscule pour éviter que « hello » et « Hello » soient considérés comme 2 éléments différents.
- **Tokenisation** ou **segmentation** : On divise le texte en petits morceaux, souvent en mots ou en phrases selon l'analyse qui va suivre. Ces petits morceaux sont appelés « jetons » (*tokens* en anglais) et constituent les éléments ou les unités sémantiques de nos données.
- **Stop words** : On supprime les mots vides (*stopwords* en anglais). Ce sont les mots très courants dans la langue et qui n'apportent pas de valeur informative pour la compréhension d'un document. En effet, ils sont très fréquents et donc les garder ne ferait qu'alourdir nos données et ralentir notre travail. En français, c'est les mots comme « et », « à », « la »… et en anglais c'est par exemple « to », « a », « the »…
- **Lemmatisation** : Dans cette étape, on cherche à représenter les mots sous leur forme canonique. Par exemple pour un verbe, ce sera son infinitif. Pour un nom ou un adjectif, son masculin singulier. L'idée étant encore une fois de ne conserver que le sens des mots utilisés dans le corpus.
- **Stemming** ou **racinisation** : C'est un processus qui a la même fonction que la lemmatisation mais qui conserve les racines des mots étudiés. L'idée étant de supprimer les suffixes, préfixes et autres des mots afin de ne conserver que leur origine. C'est un procédé plus simple que la lemmatisation et plus rapide à effectuer puisqu'on tronque les mots essentiellement contrairement à la lemmatisation qui nécessite d'utiliser un dictionnaire.

4 – Méthodes de *Topic Modeling* de type « *bag of words* »

4.1 Représentation « *bag of words* » (terme-document)

Une fois le pré-traitement des données réalisé, il est alors possible d'opter pour un modèle **« sac de mots » (*bag of words* or BoW)**. Un tel modèle considère chaque mot d'un texte sans tenir compte de sa position. Ainsi, un texte peut simplement être représenté par une liste de valeurs donnant le nombre d'occurrence de chaque mot. Pour un corpus de textes donné, on peut alors construire un tableau d'occurrences regroupant les listes d'occurrence de chaque texte. On parle alors plutôt de matrice d'occurrence et on emploie le plus souvent le terme ***term-document matrix***. Cette représentation matricielle de l'ensemble des textes est alors beaucoup plus « traitable » pour la machine que ne l'était le corpus original. Cependant la matrice terme-document obtenue est très probablement de taille très importante et très creuse.

Il est important de préciser qu'il existe plusieurs manières de construire une matrice terme-document. En effet, au lieu de compter simplement les occurrences de chaque mot, on peut aussi préférer prendre en compte la rareté d'un mot au sein du corpus pour lui attribuer de l'importance. En effet, des mots courants tels que « personne » (ou *people*) n'étant pas considérés comme des *stop-words*, il est fort probable que l'on retrouve ce dernier dans la quasi-totalité des textes d'un corpus. Comme on vient de le voir avec la méthode LSA, il serait alors bien d'abaisser le poids de ce types de mots dans la matrice terme-document. La représentation **TF-IDF pour *term frequency - inverse document frequency*** permet alors justement de prendre en compte cet aspect. Le nombre d'occurrences du mot indexé par *i* au sein du document *j* est alors remplacé par $w_{i,j}$ exprimé selon la formule présentée en *figure 1*.

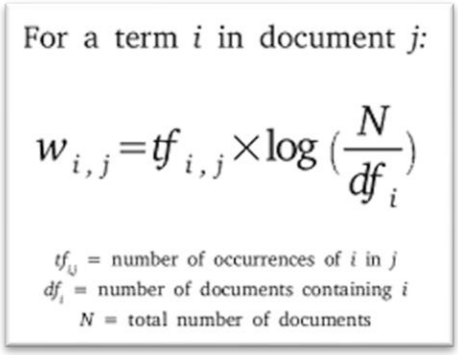

Figure 1 : Terme (i,j) de la matrice terme-document en représentation TF-IDF

4.2 - Méthode LSA[T4]

L'algorithme **LSA pour *Latent Semantic Analysis*** basé sur une représentation BoW permet d'exploiter l'information contenue dans la matrice terme-document. Cette méthode consiste à appliquer une décomposition en valeur singulière sur la matrice terme-document comme on peut le voir sur la *figure 2* où on a noté *A* ladite matrice. On peut alors identifier les

t dimensions les plus discriminantes, *t* étant le nombre de thèmes souhaité. Les matrices *U* et *V* permettent alors de relier respectivement les thèmes identifiés aux mots dont ils sont principalement composés et les documents aux thèmes prépondérants.

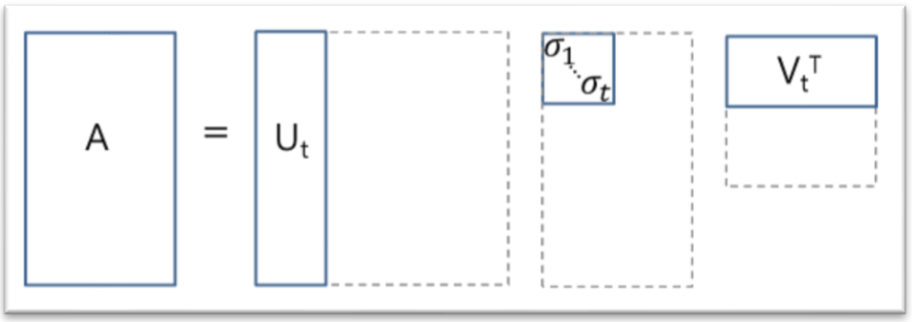

Figure 2 : SVD sur la matrice terme-document

On peut alors aisément mesurer la similarité entre mots ou documents entre réalisant des produits scalaires des vecteurs représentatifs de ces données dans l'espace à *t* dimensions des thèmes identifiés. En regardant les mots prépondérants dans chaque thème, on peut également manuellement « reconnaître un thème ». A titre d'exemple, si, une fois les composantes ordonnées par ordre décroissant de leur coefficient, le vecteur associé au thème t_i s'écrit :

$$t_i = 1.21*dog + 0.93*prey + 0.86*bird + 0.85*horse…$$

Alors on peut raisonnablement associer le thème i au thème « Animal ». Une fois les thèmes reconnus, tout nouveau document décomposé dans l'espace des thèmes peut donc être catégorisé dans un ou plusieurs de ces derniers.

L'étape de reconnaissance des thèmes par un humain est commune aux méthodes de *Topic Modeling*. En plus d'être délicate dans la plupart des cas, elle est de plus uniquement basé sur le mécanisme mathématique derrière la SVD dans le cas de la méthode LSA. Ce manque d'interprétabilité autre que mathématique du modèle est une des principales critiques pouvant être formulées à l'encontre de cette dernière. L'utilisation d'une représentation BoW est également très critiquable, le modèle est par exemple totalement imperméable aux subtilités du langage liées à la polysémie. Plus grave encore, l'enchaînement des mots étant totalement perdu par la représentation en matrice terme-document, le sens même des phrases n'est pas du tout pris en considération.

Comparé aux méthodes présentées par la suite, LSA offre tout de même un bon compromis entre résultats et complexité étant donné les capacités de calculs modernes permettant de réaliser des SVD en un temps raisonnable.

Parmi les multiples application de LSA, on peut en particulier citer le travail réalisé par *T.Landauer* et *S.Dumais* en 1997[T12]. Leur objectif a été de simuler l'acquisition du vocabulaire anglophone pour une être humain entre 2 et 20 ans. Ils ont pour ce faire appliquer la méthode LSA avec une encyclopédie électronique de plus de 30 000 articles. Ils ont ensuite testé les capacités de reconnaissance sémantique de leur modèle sur 80 questions de reconnaissance de synonyme du TOEFL. Chaque question consistant à apparier un mot cible avec le mot

sémantiquement le plus proche choisi parmi quatre propositions. Le modèle LSA obtient alors un total de 64,4 % de bonnes réponses, comparable à la moyenne des sujets non anglophones admis dans les universités américaines (64,5 %). LSA peut ainsi être considéré comme un modèle plausible de l'acquisition de connaissances à partir de données textuelles.

4.3 - Méthode pLSA

La méthode **pLSA (*probabilistic Latent Semantic Analysis*[T6,T7,T8])** développée en 1999 par Th.Hofmann permet de construire un modèle probabiliste apportant un sens aux résultats purement mathématiques obtenus avec LSA. On considère tout d'abord que le corpus de texte comme peut être exprimé par trois ensembles :

- Documents : **D** = $\{d_1,d_2,\ldots,d_N\}$
- Mots (*Words*) : **W** = $\{w_1,w_2,\ldots,w_M\}$
- Thèmes (*Topics*) : **Z** = $\{z_1,z_2,\ldots,z_K\}$

Comme précédemment, le nombre de documents *N* ainsi que le nombre de mots différents utilisés dans l'ensemble de ces documents *M* est propre au corpus tandis que le nombre de thèmes *K* est au préalable fixé. La méthode pLSA utilise elle aussi un modèle BoW traduit en termes probabilistes de la manière suivante :

$$P(\mathcal{D},\mathcal{W}) = \prod_{(d,w)} P(w|d) = \prod_{d\in\mathcal{D}} \prod_{w\in\mathcal{W}} P(w|d)^{n(d,w)}$$

n(d,w) représentant le nombre de fois où le mot w apparaît dans le document d.

Autrement dit, la probabilité d'observation d'un mot au sein d'un document ne dépend pas des autres mots du document en question. Le modèle utilisé suppose aussi que les mots et documents ne sont liés que par les thèmes et sont donc indépendants étant donné un thème fixé. Plus simplement, on peut écrire :

$$P(w,d|z) = P(w|z)P(d|z) \qquad \text{ou} \qquad P(w|d,z) = P(w|z)$$

Cela permet alors d'écrire :

$$P(w|d) = \sum_{z\in\mathcal{Z}} P(w|z)P(z|d)$$

On peut alors identifier à partir de cette équation les paramètres du modèle pLSA qui sont les probabilités **P(w|z)** au nombre de *(M-1)K* et **P(z|d)** au nombre de *(K-1)N*. L'objectif étant alors maintenant d'évaluer au mieux ces probabilités conditionnelles à partir du corpus fourni. Pour ce faire, pLSA s'appuie sur l'algorithme **EM *(Expectation-Maximization*[T7]**) afin de maximiser la log-vraisemblance des données qui s'écrit grâce aux relations précédentes de la manière suivante :

$$\log \prod_{(d,w)} P(w|d) = \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} n(d,w) \log \sum_{z \in \mathcal{Z}} P(w|z)P(z|d)$$

En reprenant les notations matricielles utilisées pour présenter la méthode LSA, on peut alors relier la matrice *U* aux termes **P(w|z)** quantifiant les relations entres thèmes et mots. De la même manière, on relie la matrice *V* aux termes **P(z|d)** quantifiant les relations entre thèmes et documents. La méthode pLSA fournit donc bien une interprétation probabiliste à la démarche de LSA. Cependant, les deux méthodes ne fournissent pas pour autant les mêmes résultats. En effet, la méthode LSA ne garantit pas que les paramètres issus des matrices *U* et *V* obtenues par SVD soient positifs. Au contraire, les termes **P(w|z)** et **P(z|d)** de la pLSA sont par définitions compris entre 0 et 1. Dans sa présentation originale de la méthode pLSA[T9], *Thomas Hofmann* précise même que cette dernière présente de meilleurs résultats que LSA face aux mots polysémiques.

Cependant, la croissance linéaire du nombre de paramètres d'un modèle pLSA avec le nombre de documents N au travers du calculs des termes **P(z|d)** expose tout particulièrement le modèle au risque de sur-apprentissage sur des corpus de textes de taille trop importante.

4.4 - Méthode LDA

La **méthode LDA (*Latent Dirichlet Allocation*[A1])** est une technique de *Topic modeling*. Le terme « *Latent* » indique le fait que la liste des topics n'est pas fixée en amont. Cette technique permet, en prenant en entrée le nombre de topics souhaité, d'obtenir des catégories regroupant des mots de la base de données. Le modèle s'appuie sur l'hypothèse que la distribution des topics dans un document et la distribution de mots dans les topics suivent une distribution de Dirichlet[A2]. *« Allocation »* indique la distribution de topics dans le document.

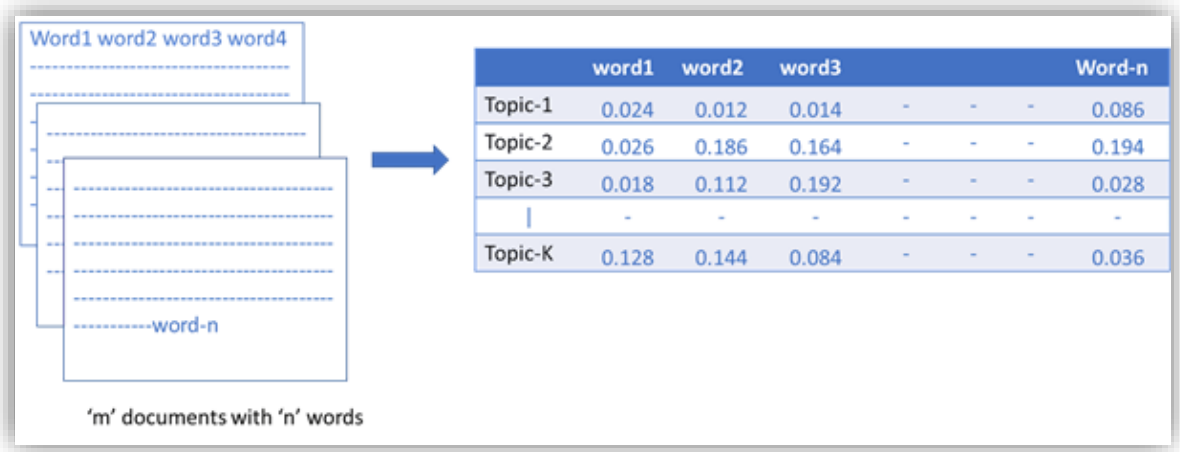

Figure 3 : En case (i,j), la probabilité que le mot j appartienne au topic i

Le principe de cette méthode est d'assigner chaque mot à plusieurs topics. En clair la *figure 3* indique la probabilité que le mot w_j appartienne au topic t_k : une ligne doit donc sommer à 1.

Une fois que ces probabilités sont estimées, il y a 2 manières de trouver l'ensemble de mots qui représente un topic :

- Soit on sélectionne, pour une catégorie donnée, les r mots qui la représentent de la manière la plus probable. En clair, on sélectionne dans la ligne correspondant au topic les r mots associés aux cases avec les plus grandes valeurs.
- Soit on fixe un seuil de probabilité pour ne pas avoir des topics qui contiennent les mots les plus probables parmi une liste de mots tous improbables.

Chaque document peut donc être représenté par la formule suivante :

$$D_i = w_{i,1}*Topic_1 + w_{i,2}*Topic_2 + \dots$$

Où chaque terme représente la proportion de mots $w_{i,j}$ qui représentent le topic j.

Hyptohèses du modèle LDA

L'algorithme suppose que chaque document est généré suivant un modèle statistique : chaque document est associé un à un certain nombre de catégories, chacune avec un certain poids. De même, chaque topic est associé à un certain nombre de mots, chacun avec un certain poids. Un exemple avec 3 catégories et un total de 10 mots est présenté en en *figure 4* :

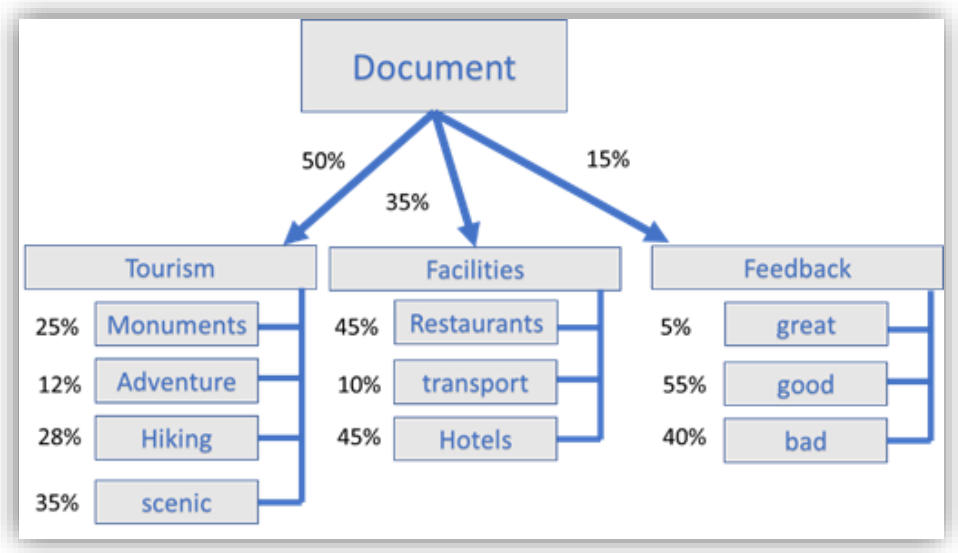

Figure 4 : Hypothèses de génération statistique d'un document

Les hyperparamètres de LDA

La méthode LDA comprend 3 hyperparamètres :

1. La densité α de topics par document (ie le paramètre de la loi de Dirichlet régissant la distribution de topics pour un document)
2. La densité β de mots par topic (ie le paramètre de la loi de Dirichlet régissant la distribution de mots pour un topic)
3. Le nombre de topics K

On choisit généralement une faible densité de topics par document afin de ne pas multiplier le nombre de topics par document pour éviter d'attribuer trop de catégories à un livre par exemple et de lui associer des topics qui ne lui seraient sûrement pas attribués par un humain en temps normal. De même, on choisit une densité de mots par topic assez faible pour désigner un topic avec des mots clés plutôt qu'un groupement de mots d'importance variable. Ainsi les valeurs de α et β sont généralement choisies en dessous de 1 pour l'algorithme LDA.

Fonctionnement de LDA

L'entraînement de LDA sur un corpus de texte consiste théoriquement à identifier les paramètres du modèle génératif ayant produit ces textes (proportions de la *figure 4*). L'algorithme LDA doit en un sens effectuer l'opération inverse de génération d'un corpus suivant le modèle génératif précédemment décrit.

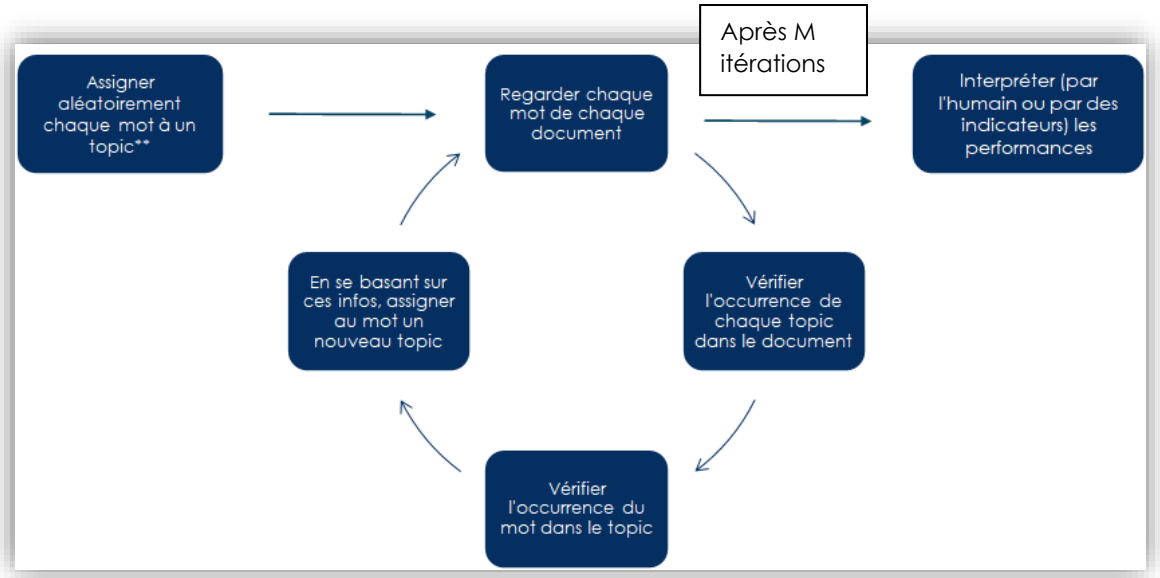

Figure 5 : Algrithme itératif du LDA

L'assignation se fait selon les lois de Dirichlet de paramètres respectifs α et β décrits plus haut.

Applications

La technique de LDA a fait ses preuves à de nombreuses reprises et s'impose comme la plus puissante méthode de *Topic Modeling* ne reposant pas sur le *Deep Learning*. En effet, LDA s'avère moins sujette au sur-apprentissage que LSA et pLSA. Certains travaux ont même pu exhiber de meilleurs résultats avec cette technique que les deux précédentes[A3]. Cependant, confronter ces méthodes ne peut se faire en pratique que sur un jeu de données. On ne peut donc pas réellement tirer de conclusion générale sur la supériorité théorique de l'une d'entre elles. Néanmoins, il est intéressant de constater que la méthode LDA présente de bons résultats sur des problèmes présentant des données qui se rapprochent des nôtres (extraits de livres). On peut par exemple citer les travaux réalisés à l'université Humboldt de Berlin qui ont permis de développer un système de recommandation de livres basé sur une classification réalisée par LDA[A4]. De plus, LDA s'est même révélée très performante sur dans des domaines autres que le traitement du langage naturel humain comme celui de la

classification de l'expression des gènes en biologie[A5]. LDA présente en effet de meilleurs résultats d'accuracy, de spécificité et de sensibilité que d'autres méthodes de classification plus classiques comme SVM ou Random Forest.

5 – Méthodes de NLP avec *Deep Learning*

5.1 – Représentation « *word embedding* » (plongement lexical)[T13]

Afin d'élaborer des modèles plus sophistiqués d'analyse du langage naturel, on a aujourd'hui le plus souvent recours au *Deep Learning*. Les réseaux de neurones ont en effet démontré leur efficacité dans de nombreux domaines comme celui de la vision par ordinateur. Cependant, afin d'exploiter au maximum la puissance de ces derniers, il est nécessaire de fournir en entrée l'information la plus riche possible. Ainsi, la représentation BoW ne convient plus dans le sens où elle ne prend pas en compte le sens des mots et retourne une image très appauvrie d'un texte. A titre d'exemple, on peut remarquer que des homographes jouent des rôles totalement identiques dans une représentation BoW. On ne prend pas non plus en compte les relations qui peuvent exister entre les mots d'une même phrase. Enfin, la matrice terme-document obtenue est dans la plupart des cas creuse, ce qui est un indice supplémentaire de la non-optimalité de cette représentation. Pour toutes ces raisons, il est donc nécessaire de développer une représentation numérique plus riche des mots du langage naturel.

De nos jours, la technique privilégiée de représentation du langage dans les modèles de pointe en NLP est le *word embedding* ou plongement lexical. Le principe de cette méthode est de représenter chaque mot d'un texte par un vecteur numérique dense (non creux) de taille fixé en amont. Il existe des méthodes de plongement lexical non-contextuelles comme **word2vec** (2013) ou **GloVe** (2014) et contextuelles telles que **ELMo** (2018) ou **BERT** (2018). Un plongement est dit contextuel lorsqu'il prend en compte le sens des mots c'est-à-dire leur contexte. Autrement dit, deux homographes n'auront probablement pas la même représentation en avec un plongement contextuel. Ainsi, on obtient par exemple des représentations différentes pour les deux occurrences de « left » dans la phr*ase suivante :*

$$« \, I \ \boldsymbol{left} \ my \ phone \ on \ the \ \boldsymbol{left} \ side \ of \ the \ table. \, »$$

Plus formellement, une représentation par plongement de taille *N* consiste à représenter chaque occurrence d'un mot dans un texte par un vecteur dans un espace à *N* dimensions. On « plonge » les mots dans un espace réel N-dimensionnel. Un des avantages de cette représentation réside dans la capacité à réduire les dimensions des données avec un choix adéquate du paramètre *N*. Il devient également possible d'effectuer des opérations entre les vecteurs représentants des mots. On peut par exemple chercher à savoir si deux mots ont un sens ou un champ lexical proche en comparant leurs représentations. La métrique la plus souvent utilisée pour faire cela est la similarité cosinus définie par :

$$cosine\_similarity(u, v) = \frac{u \cdot v}{||u|| \times |v||}$$

Il devient également possible d'écrire des égalités compréhensibles pr la machine telles que :

$$roi - homme + femme = reine$$

Une fois le plongement déterminé, il s'agit encore d'utiliser la représentation du langage obtenu pour réaliser la tâche de NLP attendue. Cependant, une grande partie du travail repose sur la qualité de la représentation obtenue. En réalité, tous les plongements lexicaux au sens strict du terme sont non-contextuels. En effet, il s'agit de manière simplifié de représenter tous les mots d'un dictionnaire

5.2 - Utilisation de RNN (*Recurrent Neural Networks*)[T10,T11]

Les réseaux de neurones récurrents diffèrent des réseaux de neurones classiques sur leur capacité à traiter des entrées de taille variable. Cette compétence est tout simplement essentielle lorsqu'il s'agit de traiter une entrée de taille non pré-définie comme du texte. De manière simplifiée, on peut dire qu'un RNN traite chaque mot comme une entrée à un instant *t* et utilise la valeur d'activation à l'instant *t-1* comme une entrée additionnelle. Cela permet alors au modèle d'élargir sa capacité de compréhension d'un texte en considérant les relations entre mots consécutifs. De plus, le nombre de paramètres d'un RNN est fixé par la structure du réseau et ne dépend pas de la taille des données en entrée diminuant ainsi le risque de sur-apprentissage par rapport aux modèles de type BoW.

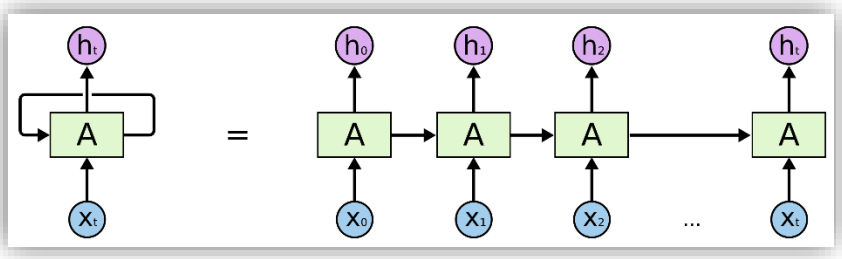

Figure 6 : Représentation imagée d'un RNN

De telles architectures de RNN ne peuvent cependant capturer les dépendances du langage que dans une direction correspondant au sens de lecture du texte par l'algorithme. On considère alors que les mots suivants n'ont aucune influence sur le sens des mots précédents. Évidemment, cette hypothèse est largement critiquable dans le cadre du traitement du langage naturel. Les RNN ne sont non plus pas adaptés pour traiter les dépendances de long-terme (*long term dependencies*) au sein d'un texte. L'information transmise d'une étape à l'autre a en effet tendance à tendre vers des valeurs extrêmes (0 ou ∞) et on parle dans ce cas de *vanishing/exploding gradients*. C'est un problème récurrent des réseaux de neurones. Enfin, les architectures de RNN sont en général lentes à entraîner du fait de l'impossibilité de paralléliser les tâches lors de cette phase d'entraînement.

5.3 - Architectures LSTM (*Long Short-Term Memory networks*)[T10,T11]

Pour remédier au problème de l'absence de mémoire à long-terme dans les RNN classiques, *S.Hochreiter et J.Schmidhuber* développent en 1997 des architectures de RNN améliorées et appelées LSTM. De manière simplifiée, ce type de structure octroie à l'algorithme une « case mémoire » qui est mise à jour à la lecture de chaque nouveau mot. L'information tirée d'un mot lu peut donc être réemployée bien plus longtemps après par la machine qu'avec une simple structure neuronale de RNN. Concrètement, un réseau LSTM remplace la cellule neuronale A de la figure 6 par la cellule de la figure 7. Le pipeline d'information traversant le haut de la cellule incarnant la « mémoire » du réseau qui est mieux préservée que dans une cellule RNN classique puisqu'elle contourne le traitement de la nouvelle donnée x_t.

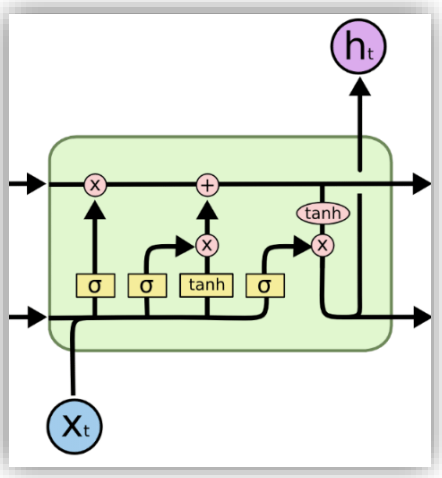

Figure 7 : Représentation d'une cellule LSTM

Bien que les réseaux LSTM présentent de meilleures performances lorsqu'il s'agit de capturer les dépendances de long-terme d'un texte, ces derniers sont aussi plus complexes que de simples RNN et donc encore plus lents à entrainer. En effet, les données sont toujours traitées de manière séquentielle.

5.4 - Utilisation de *Transformers* et modèle BERT[T13,P1 P2]

5.4.1 – Présentation générale de BERT

L'utilisation de *Deep Learning* en NLP a nottament pour objectif de produire des plongements lexicaux de qualité croissante. En 2021, les meilleurs résultats dans le domaine sont obtenus grâce à des architectures inspirés des *Transformers* (combinaisons encodeur-décodeur). C'est en particulier le cas du modèle **BERT (Bidirectional Encodeur Representation for Transformers)** publié en 2018 par Google. Ce modèle a permis de grandes avancées dans nombre d'applications du NLP du fait de la qualité du plongement lexical qu'il est possible de générer avec BERT.[T14] Ce succès a de nombreuses raisons techniques parmi lesquelles on peut citer :

- L'analyse bidirectionnelle du langage (dépendance entre les mots observée dans le sens de la lecture mais aussi dans l'autre sens)
- La mise en œuvre du processus d'attention (quantification des liens d'un mot avec tous les autres mots de la phrase) permet des performances plus élevés par rapport aux autres modèles et permet d'interpréter le fonctionnement de BERT (possibilité de visualiser graphiquement le processus d'attention)
- Analyse non-séquentielle du langage : évite les problèmes de *vanishing/exploding gradients* rencontrés avec les RNN et LTSM. Permet aussi de paralléliser les calculs et donc d'exploiter le potentiel des GPU et de réduire la durée nécessaire à l'entraînement.
- Code *open source* qui a rendu très rapide la diffusion, l'utilisation et le test du modèle

Derrière ces points de fonctionnement techniques se cache une raison théorique qui fait de BERT une petite révolution dans le domaine du NLP. En effet, les algorithmes tels que word2vec ou GloVe permettent de construire un plongement lexical qui sera ensuite utilisé par un autre système pour réaliser une tâche de NLP comme de l'analyse de sentiment, de la traduction automatique ou du *Topic Modeling*. BERT permet de réaliser à la fois le plongement lexical mais aussi d'obtenir un **modèle de langage**. On peut alors utiliser BERT pour générer des phrases puisque la structure même du langage a été apprise par l'algorithme. BERT est aussi en mesure d'appréhender de nouveau mots lorsque ceux-ci sont contextualisés. Cette compréhension plus profonde est par la suite employée pour répondre à différentes problématiques du NLP parmi lesquelles le *Topic Modeling*. Cela est, en particulier, rendu possible par **BERTopic**[T15], une technique de *Topic Modeling* basée sur BERT et développée par *Maarten Grootendorst* (*data scientist* titulaire d'un master en *data science* à l'université JADS aux Pays-Bas) en open source en 2020 et reconnu par la communauté Python.

BERT est donc un modèle de langage entraînable[T17] permettant de réaliser un plongement lexical contextualisé de haute performance. Il existe deux versions principales de BERT appelées $BERT_{base}$ et $BERT_{large}$ contenant respectivement 110 et 335 millions de paramètres. La taille de ces architectures implique le plus souvent qu'il n'est pas raisonnable de ré-entraîner l'intégralité du modèle. Il existe en effet déjà un modèle de BERT pré-entraîné par Google sur BookCorpus (environ 11 000 livres dans 16 genres différents) ainsi que sur les pages anglaises de Wikipédia (environ 2,5 milliards de mots). Plus largement, l'entraînement de BERT sur l'ensemble de Wikipédia a permis l'adaptation du réseau à 104 langues dont le français. Plus récemment, CamemBERT a également vu le jour et est une adaptation de BERT à la langue française entraînée sur la partie française du corpus OSCAR (environ 138Go de texte) présentant des performances légèrement supérieures à celles de la version multilingue de BERT. De nombreux modèles pré-entrainés de BERT sont disponibles et téléchargeables via *Hugging Face* à l'adresse : https://huggingface.co/models?sort=downloads

BERT est un encodeur multi-couches bidirectionnel. RoBERTa est une évolution de BERT qui, en plus d'avoir été entraînée plus longtemps, utilise le *masked-language model* (comme on le verra ci-dessous). CamemBERT, quant à lui, se base sur RoBERTa et se décline en deux versions : $CamemBERT_{base}$ (12 couches, 110 millions de paramètres) et $CamemBERT_{large}$ (24 couches, 335 millions de paramètres). $CamemBERT_{base}$ est trois fois plus petit, mais il est donc aussi plus rapide.

CamemBERT utilise comme données d'entraînement la partie française du corpus OSCAR (…). Le texte en entrée est pré-processé en utilisant SentencePiece.

5.4.1 – Architecture de BERT[T13]

En 2017, une équipe de chercheurs de Google publie un article qui fera date dans l'histoire du NLP. Cet article se nomme « Attention is all you need »[T18] et présente une nouvelle architecture de réseau de neurone appelée *Transformer* et présenté en *figure 8*. Ce modèle a pour objectif de répondre à la recherche de performances dans le domaine de la traduction automatique de texte d'une langue à une autre. En clair, l'objectif du réseau est, étant donné une phrase écrite dans une langue A, de renvoyer la traduction de cette même phrase dans une langue B. La différence cruciale faîte par le *Transformer* par rapport à un RNN réside dans sa capacité à considérer une phrase entière en tant qu'entrée là où le RNN traite la phrase séquentiellement. Avec un *Transformer*, chaque mot passé en entrée entre en relation avec tous les autres mots de la phrase à laquelle il appartient. L'apprentissage d'un tel réseau consiste donc, en quelques sortes, à identifier les liens qui peuvent exister entre les différents mots d'un phrase aussi éloignés soient-ils les uns des autres, c'est ce qu'on appelle un **mécanisme d'attention**. Un exemple d'application de ce processus peut être visualisé sur la *figure 8*.

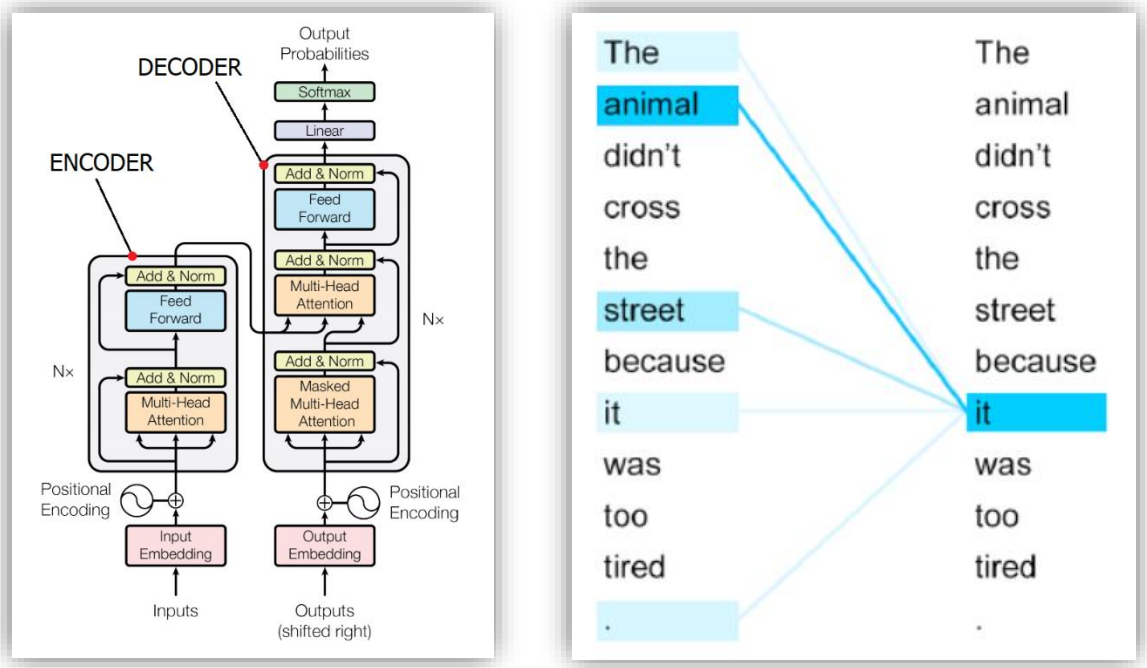

Figure 8 : Architecture d'un Transformer proposé dans « Attention is all you need » (gauche) et visualisation d'un processus d'auto-attention par rapport au mot « it » (droite)

L'architecture d'un Transformer se compose d'une partie décodeur (blocs de gauche) et d'une partie encodeur (bloc de droite). L'encodeur a pour objectif de réaliser un plongement lexical des mots de la phrase passée en *Inputs* nottament grâce à la couche *Multi-*

Head Attention qui met en œuvre le mécanisme d'attention. On remarque la présence d'un *Positional Encoding* en amont de l'encodeur. Cette entrée est en réalité combinée avec les *Inputs* et permet de fournir l'information concernant l'ordre des mots de la phrase à l'encodeur. Ce codage positionel est crucial pour la prise en compte du contexte. En sortie de l'encodeur on obtient un plongement lexical contextualisé de chaque terme de la phrase passée en entrée. En pratique, pour une tâche de traduction, la représentation encodée est injectée dans le décodeur prenant en entrée, grâce à un bouclage, les mots qu'il a lui-même déjà traduit.

La structure et le fonctionnement du décodeur ne sera pas étudiée de plus près ici puisqu'il s'avère que BERT n'utilise uniquement que des encodeurs inspirés de l'architecture d'un *Transformer*. La structure de $BERT_{base}$est présentée en *figure 9*.

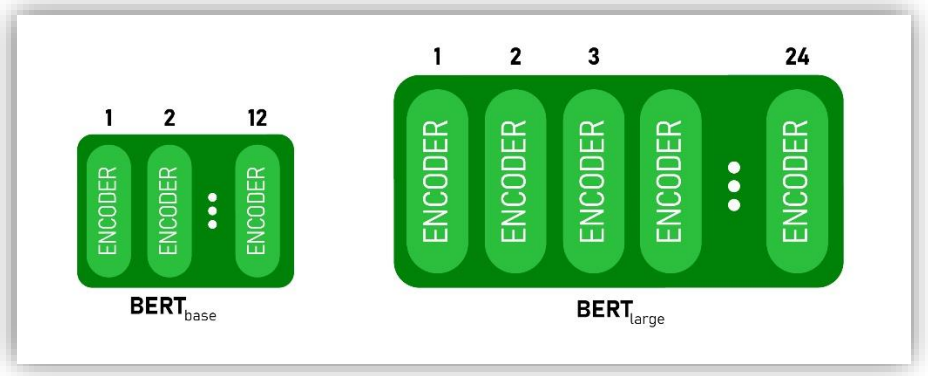

Figure 9 : Structure de $BERT_{base}$ et $BERT_{large}$

BERT n'est donc « simplement » qu'une superposition de plusieurs encodeurs, chacun produisant une sortie servant d'entrée au suivant. C'est justement cette superposition d'encodeurs qui permet au réseau de répéter l'application de processus d'attention et d'obtenir une compréhension de plus en plus profonde de ses entrées. Précisons encore un peu plus la manière dont une phrase en anglais par exemple est injectée dans BERT. Tout d'abord, BERT fonctionne par *tokens* qui sont soient des mots soient des parties de mots. Par exemple, *tokeniser* « Hello » donnera [Hello] tandis que « playing » donnera [play, ##ing]. $BERT_{base}$ recense environ 30 000 *tokens* qu'on appelle son « vocabulaire ». BERT assigne à chaque *token* un numéro et peut donc, à partir d'une phrase, *tokeniser* puis numériser les tokens pour produire un vecteur qui, combiné au codage positionel, devient digeste pour le premier encodeur.

On a supposé jusqu'ici que BERT ne prenait en entrée que des phrases. Il est cependant possible de lui fournir plusieurs phrases d'un seul coup. Le modèle s'occuper alors d'ajouter un *segment embedding* permettant de savoir de quelle phrase provient chaque token. Il devient alors possible de directement fournir des textes en entrée du réseau. Cependant, l'architecture de BERT est construite de telle manière qu'un décodeur ne peut prendre que 512 *tokens* en entrée d'un seul coup. BERT n'est donc pas encore très adaptée à la gestion de longs textes.

5.4.3 – Entraînement de BERT avec MLM et NSP MANQUE NSP = Next Sentence Prediction

Le *masked-language model* fonctionne à partir d'une séquence d'entrée de taille N : on a N tokens $x_1, x_2, \ldots, x_N$. On sélectionne 15 % de ces tokens pour un possible remplacement : parmi ces 15 % sélectionnés, 80 % sont remplacés par le token spécial <MASK>, 10 % restent inchangés et 10 % sont remplacés par un token aléatoire. Le modèle est ensuite entraîné pour retrouver les tokens initiaux (en utilisant la fonction loss d'entropie croisée).

Les tokens d'entrée sont soit des mots, soit des sous-mots (n'utiliser que des mots – Whole-World Masking, WWM – permet d'améliorer les performances à terme, mais, en contrepartie, l'entraînement est nettement plus difficile).

Pour l'optimisation, on utilise Adam (avec comme paramètres $\beta_1 = 0{,}9$ et $\beta_2 = 0{,}98$, 100k steps, une batch size de 8 192 séquences, chaque séquence contenant au plus 512 tokens).

Le pré-entraînement a été fait avec 100 000 backpropagation steps sur 256 GPU Nvidia V100 (32 Go de RAM chaque) pendant une journée (les performances auraient été encore meilleures avec une durée plus longue, mais ce n'était pas possible d'un point de vue pratique pour les auteurs de la thèse).

5.4.4 – Méthode de *Topic Modeling* avec BERTopic

L'approche du problème de *Topic Modeling* proposée par BERTopic est différente de tout ce qui a pu être vu avec les modèles BoW mais se trouve, en un sens, être plus naturelle. BERTopic s'appuie en effet sur un plongement lexical des mots pour construire une représentation vectorielle de chaque document puis identifier des clusters entre ces dernières. La *figure 10* issue de la page GitHub de présentation de BERTopic[T16] par M.Grootendorst présente schématiquement les différentes étapes de l'algorithme.

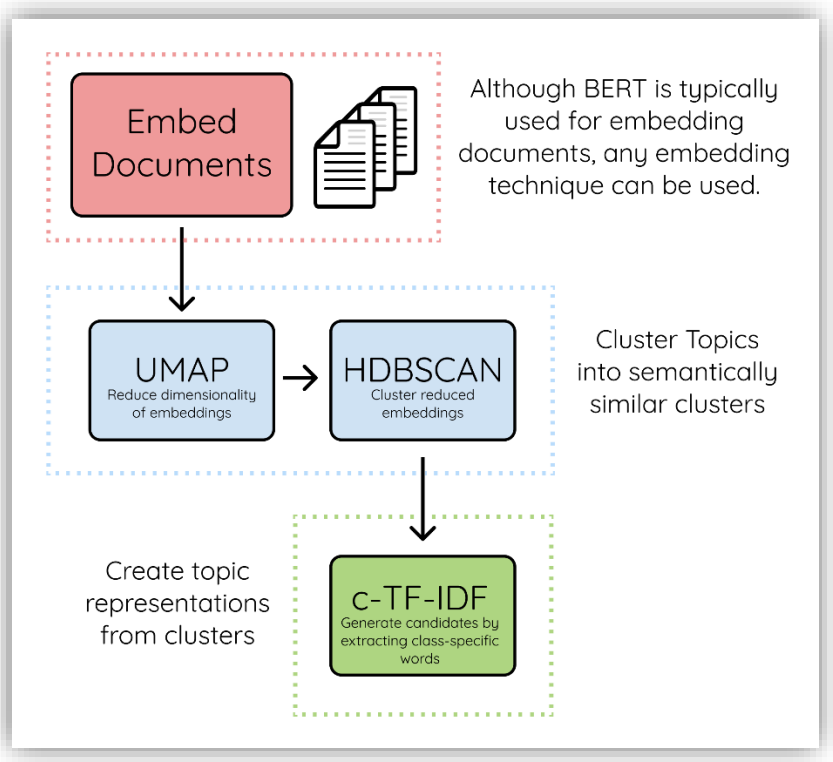

Figure 10 : Présentation de l'algorithme BERTopic par M.Grootendorst

Détaillons un peu plus cette architecture dans laquelle on peut identifier les quatre grandes étapes suivantes :

- [1] Plongement lexical des documents grâce à BERT et un ***Sentence Transformer***
- [2] Réduction de la dimensionnalité des représentations avec **UMAP**
- [3] Clustering des représentations réduites avec **HDBSCAN**
- [4] Identification des mots clefs de chaque cluster avec **c-TF-IDF**

[1] ***Sentence Transformer***[T11]

La première étape utilise un modèle pré-entrainé de BERT afin de réaliser un plongement lexical des mots présents dans le corpus de documents. Chaque document peut donc être représenté par une suite de vecteurs ayant tous la même dimension. Un *Sentence Transformer* est une méthode de pooling sur ces différents vecteurs permettant *in fine* d'obtenir un seul vecteur représentant chaque document. La technique la plus simple est par exemple de réaliser un *mean pooling* qui consiste à attribuer au document le barycentre de toutes les représentations des *tokens* le composant. Cette procédure n'est malheureusement pas la plus performante et on a plutôt recours à du NLI (*Natural Language Inference*) et du STS (*Sentence Test Similarity*). De manière simplifiée, ces techniques permettent d'assurer la cohérence entre les représentations de chaque documents en comparant ces dernières entre elles. NLI est, par exemple, chargée d'identifier des liens d'implication, de contradiction ou de neutralité entre phrases. STS permet de tester la similarité sémantique entre deux phrases. Au final, lorsqu'on applique BERTopic, chaque document est représenté par un vecteur à 768 coordonnées.

[2] **UMAP**

Le recours à une méthode de réduction est rendu nécessaire par la taille trop importante de la représentation de chaque document pour effectuer directement un clustering. En effet, les méthodes de clustering s'appuyant sur des distances pour comparer la proximité des éléments, il est avéré que des mesures comme celle d'Euclide ou de Manhattan deviennent inexploitables en dimension élevée.[T19] UMAP (*Uniform Manifold Approximation & Projection*) est une méthode stochastique non-linéaire de réduction de dimensions dont l'application au problème qui nous intéresse est cité dans la documentation.[T20] UMAP est en particulier apprécié pour sa rapidité d'exécution sur une machine classique en 2018 (quelques secondes).

[3] **HDBSCAN**

HDBSCAN (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*) est une méthode de clustering s'appuyant sur la densité des données plutôt que sur la recherhce de forme convexe pour former des clusters. Une de ses particularité réside dans la spécification « *with Noise* » signifiant que l'algorithme n'établit pas une partition des données. Ce dernier regroupe certaines données pour former un cluster et en laisse d'autres sans attribution considérant qu'il n'est pas pertinent des les associer à aucun groupement. HDBSCAN est également reconnu pour fonctionner de manière complémentaire avec UMAP et s'exécuter en un laps de temps similaire.

[4] **c-TF-IDF**

Une fois les cluster de documents identifiés, il convient de s'en faire une représentation. Pour ce faire, on souhaite identifier les mots prépondérants dans chacun d'entre eux. On s'appuie alors sur la représentation TF-IDF déjà présenté en section 4.1. L'idée étant désormais de considérer tous les documents d'un cluster comme un unique super-document. On utilise alors la méthode c-TF-IDF pour *classed-based* TF-IDF ou c-TF-IDF en appliquant, pour chaque mot *t* au sein de chaque cluster *i*, la formule suivante :

$$c - TF - IDF_i = \frac{t_i}{w_i} \times log \frac{m}{\sum_{j=1}^{m} t_j}$$

Avec les notations :

- t_i : Nombre d'occurrences du mot *t* dans le cluster *i*
- w_i : Nombre de mots distincts dans le cluster *i*
- m : Nombre total de clusters construits par HDBSCAN

Chaque cluster peut donc maintenant être interprété comme un thème donc les mots caractéristiques sont ceux possédant les scores c-TF-IDF les plus élevés.

Utilisation de BERTopic

Du fait de sa relative nouveauté, il n'existe pas beaucoup d'exemples d'application de l'algorithme. Cependant, l'auteur *M.Grootendorst* a tout de même appliqué sa méthode[T16] au corpus *20 Newsgroup* regroupant plus de 18 000 articles rédigés en anglais et issus de 20 groupes de presse différents pour un total d'environ 72 Mo de texte. L'exemple est reproductible et il est possible grâce à BERTopic de classifier plus des 2/3 du corpus dans environ 200 topics. La *figure 11* présente quelques topics qu'il est possible d'obtenir et qui répondent très bien aux attentes de cohérence et de spécificité qu'on pourrait avoir.

```
top_n_words[7][:10]

[('game', 0.010457064574205876),
 ('team', 0.009330623698817741),
 ('hockey', 0.008341477022610098),
 ('games', 0.006831457696895118),
 ('players', 0.006753830927421891),
 ('play', 0.006293209317999615),
 ('season', 0.006227752030983029),
 ('baseball', 0.0060984850344868195),
 ('year', 0.005789161738305711),
 ('nhl', 0.005736180378958607)]
```

```
top_n_words[43][:10]

[('dos', 0.014029062524679042),
 ('windows', 0.010633883955998472),
 ('problem', 0.007022143162108724),
 ('help', 0.005238622429981215),
 ('disk', 0.005146911575725927),
 ('thanks', 0.005086509908619605),
 ('file', 0.0049983442954192),
 ('program', 0.004945085186682861),
 ('pc', 0.004936689541825903),
 ('files', 0.004886658254378234)]
```

```
top_n_words[12][:10]

[('nasa', 0.019843378603487997),
 ('space', 0.019422587182152878),
 ('gov', 0.01211931116301047),
 ('henry', 0.00885814675356012),
 ('launch', 0.008563915961385683),
 ('orbit', 0.00826107575349062),
 ('moon', 0.007999346663885938),
 ('earth', 0.007568500945765267),
 ('shuttle', 0.0075630804907155895),
 ('jpl', 0.007471242802444713)]
```

```
top_n_words[41][:10]

[('jesus', 0.017941948810926055),
 ('god', 0.017222920386950193),
 ('church', 0.011782741914097233),
 ('christian', 0.011198341988130087),
 ('christians', 0.010921245789061267),
 ('christ', 0.010734557826855092),
 ('bible', 0.01067142555458017 3),
 ('faith', 0.010616988475347046),
 ('sin', 0.007795019887970474),
 ('christianity', 0.007527424973714497)]
```

Figure 11 : Exemples de topics obtenus avec BERTopic sur 20 Newsgroup

La *figure 12* permet de visualiser une projection sur deux dimensions principales des différents topics obtenus. On peut alors observer des clusters bien séparés les uns des autres et il devient même possible de regrouper « à la main » plusieurs d'entre eux afin d'en réduire le nombre.

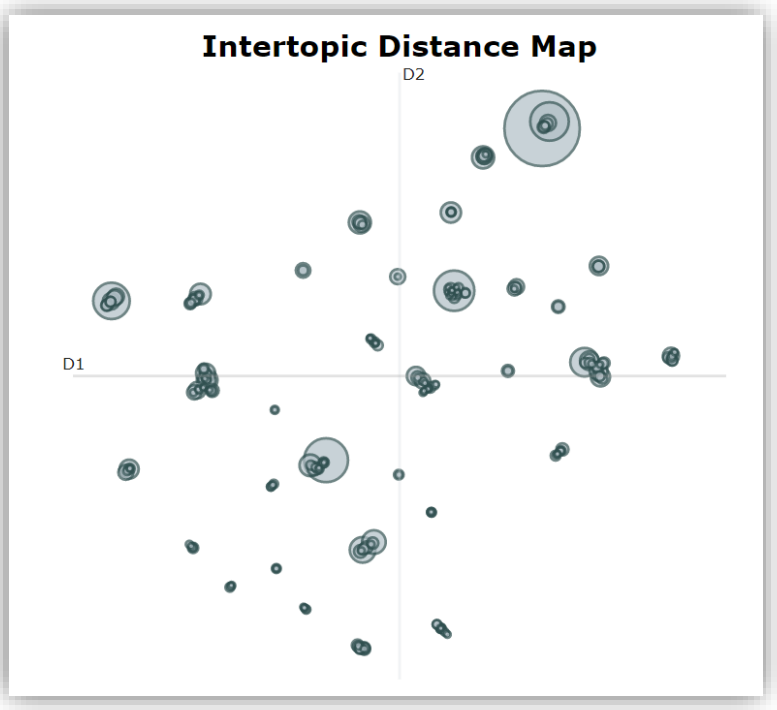

Figure 12 : VIsualitation en 2D des topics obtenus par BERTopic sur 20 Newsgroup

6 – Évaluation des résultats en Topic Modeling C1 C2

Il existe plusieurs façons d'évaluer les méthodes de topic modeling:

- Evaluation par jugement humain :
  - Basée sur l'observation, par exemple en observant les 'N' premiers mots d'un sujet.
  - Basée sur l'interprétation, ces approches demandent plus d'efforts que les approches fondées sur l'observation, mais produisent de meilleurs résultats. On a par exemple la méthode de « l'intrusion de mots » (*Word Intrusion*) ou la méthode de « l'intrusion des sujets » (*Topic Intrusion*) pour identifier les mots ou les sujets qui n'ont pas leur place dans un sujet ou un document. Dans le cas de l'intrusion de mots par exemple, on présente aux sujets des groupes de N mots, dont N-1 appartiennent à un sujet donné et un mot "intrus" puis on demande aux sujets d'identifier le mot intrus. Ainsi, plus la tâche est simple pour les sujets, plus le sujet en question est cohérent.

- Mesures quantitatives :
  - Calculs de la perplexité.
  - Calculs de la cohérence.

- Approches mixtes : Combinaison d'approches quantitatives et basées sur le jugement.

Method	Description
Human judgment approaches	
Observation-based	Observe the most probable words in the topic
Interpretation-based	Word intrusion and topic intrusion
Quantitative approaches	
Perplexity	Calculate the held out log-likelihood
Coherence	Calculate the conditional likelihood of co-occurrence

Figure 13 : Méthodes d'évaluation du Topic Modeling

6.1 – Mesure de perplexité C4

Il s'agit d'une mesure statistique de la capacité d'un modèle à prédire un échantillon. Cela se fait généralement en divisant l'ensemble de données en deux parties : l'une pour la formation, l'autre pour le test. Pour LDA, un ensemble de test est une collection de documents inconnus $\mathbf{w}_d$, et le modèle est décrit par la matrice de topics $\mathbf{\Phi}$ et l'hyperparamètre α pour

la distribution des sujets des documents. Par conséquent, nous devons évaluer la log-vraisemblance d'un ensemble de documents inconnus **Wd** étant donné les sujets **Φ** et l'hyperparamètre **α** pour la distribution des sujets **θd** des documents :

$$\mathcal{L}(\boldsymbol{w}) = \log p(\boldsymbol{w}|\boldsymbol{\Phi}, \alpha) = \sum_d \log p(\boldsymbol{w}_d|\boldsymbol{\Phi}, \alpha)$$

La vraisemblance des documents inconnus peut être utilisée pour comparer les modèles ; une vraisemblance plus élevée implique un meilleur modèle.

La mesure de perplexité est construite à partir de la vraisemblance de la façon suivante :

$$\text{perplexity}(\text{test set } \boldsymbol{w}) = \exp\left\{-\frac{\mathcal{L}(\boldsymbol{w})}{\text{count of tokens}}\right\}$$

qui est une fonction décroissante de la log-vraisemblance L(w) des documents inconnus wd ; plus la perplexité est faible, meilleur est le modèle.

Cependant, la vraisemblance p(wd|Φ,α) d'un document est mal interprétable, ce qui rend l'évaluation de L(w), et donc la perplexité, difficilement interprétable également.

6.2 – Mesures de cohérence

Un ensemble d'affirmations ou de faits est dit **cohérent**, s'il se soutient mutuellement. Ainsi, un ensemble cohérent de faits peut être interprété dans un contexte qui couvre tous ou la plupart des faits. Voici un exemple d'ensemble cohérent de faits : "le jeu est un sport d'équipe", "le jeu se joue avec un ballon", "le jeu exige de grands efforts physiques"...

Les mesures de **Topic Coherence** évaluent un seul sujet en mesurant le degré de similarité sémantique entre les mots à score élevé du sujet. Ces mesures aident à faire la distinction entre les sujets qui sont sémantiquement interprétables et les sujets qui sont des artefacts d'inférence statistique.

6.3 – Mesure de C_v

La mesure **C_v** est basée sur une fenêtre glissante, une segmentation en un ensemble des mots les plus importants et une mesure de confirmation indirecte qui utilise l'information mutuelle normalisée (NPMI) et la similarité en cosinus.

7 - Références bibliographiques

T1 Georgetown-IBM experiment – *Wikipedia* 2020 - https://en.wikipedia.org/wiki/Georgetown%E2%80%93IBM_experiment

T2 ELIZA – *Wikipedia* 2021 - https://fr.wikipedia.org/wiki/ELIZA

T3 Topic Modeling : An Introduction – *Frederico PASCUAL* – 2019 - https://monkeylearn.com/blog/introduction-to-topic-modeling/

T4 Topic Modeling with LSA, pLSA, LDA & lda2Vec – *Joyce XU* – 2018 - https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

T5 Using TF-IDF to Determine Word Relevance in Document Queries – *Juan RAMOS* – Department of Computer Science, Rutgers University, New Jersey - https://www.researchgate.net/publication/228818851_Using_TF-IDF_to_determine_word_relevance_in_document_queries

T6 Probabilistic Latent Semantic Analysis – Dan ONEATA - https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/oneata.pdf

T7 A Note on EM Algorithm for Probabilistic Latent Semantic Analysis - *Qiaozhu MEI, ChengXiang ZHAI* - Department of Computer Science University of Illinois - http://times.cs.uiuc.edu/course/598f16/plsa-note.pdf

T8 Improving pLSA with PCA – *Ayman FARAHAT, Francine CHEN* – https://aclanthology.org/E06-1014.pdf

T9 pLSA – *Thomas HOFMANN* – University of California, Berkeley 1999/2001 - https://www.researchgate.net/publication/220343825_Hofmann_T_Unsupervised_Learning_by_Probabilistic_Latent_Semantic_Analysis_Machine_Learning_421-2_177-196

T10 Understanding LTSM networks – *colah's blog : Christopher Olah* - http://colah.github.io/posts/2015-08-Understanding-LSTMs/

T11 Transformer Neural Networks – *Youtube : CodeEmporium (Ajay Halthor)* - https://www.youtube.com/playlist?list=PLTl9hO2Oobd_bzXUpzKMKA3liq2kj6LfE

T12 A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge – *T.Landauer, S.Dumais* - *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.333.7403&rep=rep1&type=pdf*

T13 FROM Pre-trained Word Embeddings TO Pre-trained Language Models - Focus on BERT – *Adrien Sieg* – 2019 - https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-343815627598

T14 How BERT and GPT models change the game for NLP – *IBM : Demi Ajayi* – 2020 - https://www.ibm.com/blogs/watson/2020/12/how-bert-and-gpt-models-change-the-game-for-nlp/

T15 BERTopic – *Maarten Grootendorst* – 2020 - https://maartengr.github.io/BERTopic/

T16 Topic Modeling with BERT – *Maarten Grootendorst* – 2020 - https://towardsdatascience.com/topic-modeling-with-bert-779f7db187e6

T17 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding – *Google IA : Jacob Devlin Ming-Wei Chang, Kenton Lee, Kristina Toutanova* – 2019 - https://arxiv.org/pdf/1810.04805.pdf

T18 Attention is all you need – Google Research/Brain : *Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin* – 2017 - https://arxiv.org/pdf/1706.03762.pdf

T19 Clustering sentence embeddings to identify intents in short text – *David Borelli* – 2019 - https://towardsdatascience.com/clustering-sentence-embeddings-to-identify-intents-in-short-text-48d22d3bf02e

T20 Document embedding using UMAP – *Leland McInnes* – 2018 - https://umap-learn.readthedocs.io/en/latest/document_embedding.html

A1 Understanding Latent Dirichlet Allocation (LDA) - *Great Learning Team & Arun K Sharma* - https://www.mygreatlearning.com/blog/understanding-latent-dirichlet-allocation/

A2 Dirichlet Distribution, Dirichlet Process and Dirichlet Process Mixture – *Leon Gu* - https://www.cs.cmu.edu/~epxing/Class/10701-08s/recitation/dirichlet.pdf

A3 LSA & LDA Topic Modeling classification : comparisaon study on E-books - *Shaymaa H. Mohammed, Salam Al-augby* - https://www.researchgate.net/publication/338817648_LSA_LDA_Topic_Modeling_Classification_Comparison_study_on_E-books

A4 Building a LDA-based Book Recommender System - *Andrew McAllister, Ivana Naydenova, Quang* - https://humboldt-wi.github.io/blog/research/information_systems_1819/is_lda_final/#Building-a-LDA-based-Book-Recommender-System

A5 A Novel Approach for Classifying Gene Expression Data using Topic Modeling – *Soon Jye Kho, Himi Yalamanchili, Michael L. Raymer, Amit Sheth* - https://corescholar.libraries.wright.edu/cgi/viewcontent.cgi?article=2515&context=knoesis

P1 Les modèles de langue contextuels CamemBERT pour le français : impact de la taille et de l'hétérogénéité des données d'entraînement – *Louis Martin et al.* – 2020.

P2 CamemBERT : a TAsty French Language Model – *Louis Martin et al.*

C1 https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0

C2 https://highdemandskills.com/topic-model-evaluation/

C3 https://ichi.pro/fr/tokenisation-pour-le-traitement-du-langage-naturel-177543891237588

C4 http://qpleple.com/perplexity-to-evaluate-topic-models/