Projet 7:
Topic Modelling - Evaluation des performances

Edward Levavasseur

Ingénieur Machine Learning

De nombreuses méthodes et algorithmes ont été développés pour classer des textes de manière non-supervisée. Ce papier repose sur l'application de différentes méthodes à des questions de Stack Overflow, pour évaluer de manière systématique les performances des différentes méthodes. J'applique différente méthodes de word embeddings (CountVectorizer, TF-IDF), puis différentes méthodes de réduction des dimensions (LDA, LSA, NNMF) pour générer des topics associés aux questions de Stack Overflow. Enfin, j'applique un score de cohérence à chaque Topic pour évaluer de manière systématique les performances des différents modèles. Les résultats montrent que le LSA génère des résultats meilleurs que LDA et NNMF sur un petit nombre de topics, et avec une plus grande rapidité de calculs. Pour finir, j'applique également 2 algorithmes récents (BERT et Top2Vec) qui effectuent automatiquement le word embedding et génèrent le nombre optimal de topics.

Keywords: *Topic modeling, word embeddings, dimensionality reduction, coherence*

1. Introduction

La classification de textes de manière non-supervisé est devenu un des segments phares du Machine Learning. Elle est utilisée dans un grand nombre de secteurs d'activité, dont la finance ou le marketing.

La classification de textes peut se faire de manière supervisée (Topic Classification) et de manière non-supervisée (Topic Modeling). Ici je choisi de me focaliser exclusivement sur le *topic modeling* car il permet de classer n'importe quel texte, et d'en identifier un thème sous-jacent (Topic). Par ailleurs, c'est aussi le topic modeling qui de part sa nature non-supervisée est le plus susceptible de générer des résultats contestables, voir absurdes.

Ainsi, une double problématique est liée à la classification non-supervisée de textes: une nécessité accrue de mesurer la qualité de la classification, et une difficulté inhérente à mesurer cette qualité de classification.

Il existe de nombreuses méthodes de topic modeling, dont une bonne partie sont classées par Kherwa and Bansal (2020), selon qu'elles soient des models probabilistes ou pas. Dans le premier cas, les méthodes reposent sur des hypothèse sous-jacentes concerant les lois de probabilités que suivent les topics, et les mots au sein de ces topics. Dans le second cas, de telles hypothèses ne sont pas nécessaires, car les méthodes reposent sur des décomposition de matrices, sous forme de produit de matrices de dimension inférieures.

Dans cet article je ne cherche pas à classer les différentes méthodes selon leur nature, mais plutôt à évaluer leurs performances de manière systématique. Je mène donc une étude comparative des différentes approches, en appliquant une mesure unique de la qualité d'identification des topics: le score de cohérence. Dans un premier temps j'applique des méthodes de *word embedding* à des questions de Stack Overflow, pour représenter chaque question dans un espace vectoriel, où chaque dimension correpond à un mot utilisé dans le corpus des questions. J'essaye d'abord la méthode de CountVectorizer, puis celle de TF-IDF. Dans un second temps, j'applique des méthodes de réduction de dimensionalité pour identifier les Topic sous-jacents aux questions. J'essaye d'abord la méthode du *Latent Dirichlet Allocation* (LDA), puis celle du *Non-Negative Matrix Factorization* (NNMF), et enfin celle du *Latent Semantic Analysis* (LSA). Pour finir, ayant généré un certain nombre de topics avec chaque méthode, j'applique un score de cohérence sur chaque topic, pour en évaluer la qualité.

Dans la partie suivante de cet article je décris les méthodes de word embeddings, puis celles de réduction des dimensions. Dans la troisième partie je présente une comparaison des résultats obtenus avec CountVectorizer et avec TF-IDF, puis une comparaison des résultats obtenus avec LDA, NNMF, et LSA. Dans la quatrième partie je présente de nouveaux algorithmes - BERT et Top2Vec - qui effectuent l'intégralité du travail: le cleaning, le word embedding et la production de topics. Dans la dernière partie je conclue cette étude.

2. Méthodes

2.1. Word Embedding

La première étape dans une classification de documents consiste généralement à nétoyer les textes, à savoir supprimer les *stopwords* - mots courrant de la langue anglaise - à lemmatizer les mots - supprimer les pluriels ou conjuguaisons - et enfin à identifier les *features* - mots les plus importants du corpus de textes.

Ici le travail de préparation des données est déjà fait: chaque question a déjà un *bag of words* qui lui est attaché. La deuxième étape consiste donc à utiliser le bag of words de chaque texte pour représenter les textes dans un espace vectoriel, où chaque dimension est un mot du corpus de questions. Les deux méthodes les plus fréquentes, et que j'utilise ici sont le *CountVectorizer* et le *Term frequency - inverse document frequency* (TF-IDF).

Le CountVectorizer est la plus simple et la plus intuitive des approches. Elle consiste simplement à compter le nombre de fois que chaque feature à été utilisée dans chaque bag of words, et à représenter ce nombre dans une matrice. Les lignes de cette matrice sont les textes, les colonnes de cette matrice sont les features, et les cases de la matrice sont le nombre de fois que le mot de la colonne a été utilisé dans le texte de la ligne. L'avantage de cette approche est qu'elle est intuitive. Son inconvénient est que le CountVectorizer est incapable de distinguer si le mot est beaucoup utilisé dans un texte parce que c'est un mot très courrant de la langue anglaise, ou s'il est beaucoup utilisé parce que c'est un mot crucial pour la signification du texte.

La méthode de TF-IDF corrige en parti cette limite. En effet la méthode de TF-IDF produit également une matrice avec les textes en lignes et les features en colonnes, mais con-

trairement au CountVectorizer ne se contente pas de compter le nombre de fois que le mot a été utilisé dans un texte. Le TF-IDF indique à la place:

$$P(terme \mid document) \times log\Big(\frac{1}{P(documents\, avec\, terme \mid tous\, documents)}\Big)$$

La première partie de la formule correspond au *term frequency* et correspond à la fréquence d'une feature dans un texte donné, et la seconde partie du produit correspond à l'*inverse document frequency* et correspond au log de l'inverse du nombre de documents contenant le mot, sur le nombre de documents total. Par conséquent, ceteris paribus, plus un mot se trouve dans un petit nombre de textes, plus il aura un poids important au sein du texte qui l'utilise. Un mot comme "Python" aura donc plus de poids que le mot "the" dans un même texte, même s'ils étaient utilisés 2 fois chacun dans le texte.

Une fois que l'une ou l'autre des méthodes de word embedding a été appliqué aux bag of words, j'applique différentes méthodes de réduction des dimensions à l'espace vectoriel généré. Chaque nouvelle dimension produite sera considérée comme un Topic.

2.2. Latent Dirichlet Allocation (LDA)

La méthode de topic modeling la plus courramment utilisée aujourd'hui s'appelle le *Latent dirichlet allocation model*, et a été dévelopé par Blei et al. (2003). Contrairement aux deux méthodes qui suivent (*NNMF* et *LSA*), le LDA est une méthode probabiliste (Kherwa and Bansal (2020)) qui repose sur l'hypothèse que les topics associés à un document suivent une loi de dirichlet, et les mots associés à un topics suivent eux-même une loi de dirichlet. La résolution analytique de l'identification des topics assez complexe et ne peut pas être détaillée ici (voir: *https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation*).

L'algorithme du LDA identifie k topics sous-jacents au corpus de textes, puis va créer un simplex de probabilité dans lequel representer chaque document. Ainsi, contrairement aux approches suivantes, la méthode du LDA ne consiste pas à proprement parler d'une réduction de dimension, mais plutôt au fait d'attribuer une probabilité à chaque texte d'appartenir à chaque topic. Le résultat final est donc une matrice n (nombre de textes) $\times$ k (nombre de topics), où les éléments d'une ligne se somment à 1.

2.3. Non-Negative Matrix Factorization (NNMF)

La méthode du *Non-Negative Matrix Factorization* est plus ancienne que le LDA, car elle a été developpée en mathématiques, bien avant l'existence de la classification de textes. Néanmoins elle reste une méthode assez courrante. Elle consiste à décomposer la matrice issue du word embeding X comme le produit de matrices de dimensions infereieurs W et H:

$$\underset{n\times m}{X} = \underset{n\times k}{W} \times \underset{k\times m}{H}$$

Où X est la matrice issue du word embedding avec n textes et m features, et W est la matrice avec n textes et k topics. H est une matrice indiquant comment les k topics sont associés au m

features d'origine.

L'identification des matrices est obtenue en résolvant le problème de minimization suivant:

$$\min_{W,H} ||X - WH||^2 \quad s.c\ W \geq 0,\ H \geq 0$$

Bien qu'il soit difficle de trouver une solution analytique à ce problème de minimiszation, un algorithme peut trouver une approximation de la solution grace à la descente de gradient.

2.4. Latent Semantic Analysis (LSA)

Tout comme le NNMF, la méthode du *Non-Negative Matrix Factorization* est aussi plus ancienne que le LDA, car elle prédate le topic modelling. Néanmoins elle reste une méthode assez courrante, qui repose sur une décomposition SVD. Elle consiste à décomposer la matrice issue du word embeding comme le produit de matrices de dimensions inféreieurs. Plus précisément, elle consiste à identifier les valeurs propres de la matrice de word embedding, puis les vecteurs propres correpondant, et à effectuer une décomposition comme suit:

$$\underset{n\times m}{X} = \underset{n\times k}{U} \times \underset{k\times k}{\Sigma} \times \underset{k\times m}{V}$$

Où X est la matrice issue du word embedding avec n textes et m features, Σ est une matrice diagonale composée des k valeurs propres, V est une matrice composée des k vecteurs propres dans les m dimensions (features). Enfin, U est la matrice avec les n textes représentés dans un espace à k dimensions, qui seront considérés comme les k topics du corpus de textes.

Il existe une solution analytique à cette décomposition, que l'on peut trouver sur internet: *https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm.*

3. Résultats

Il y a plusieurs méthodes pour évaluer les performances d'un modèle de topic modeling. Deux des méthodes les plus courantes sont la *Perplexity* et la *Coherence*. La *Perplexity* consiste à séparer les données en *training set* et *test set*, puis à évaluer les performances d'un modèle calibré sur les données training sur les données test. Le problème de cette approche est que chaque méthode (LDA, NNMF, LSA) génère des résultats sous une forme différente (simplex de probabilité, réduction de dimension), et la comparabilité de la *Perplexity* entre les différentes méthodes est questionable. Par ailleurs Chang et al. (2009) touvent qu'il peut dans certains cas y avoir une corrélation inverse entre la mesure de *Perplexity*, et la qualité sémantique lorsqu'elle est évaluée par des être humains. Par conséquent je choisi d'utiliser la méthode de la *Coherence*.

3.1. Evaluation des performances: Score de Cohérence

Une étude comparative des différentes méthodes a déjà été menée par Stevens et al. (2012). Dans ce papier ils comparent les performances obtenues avec différents modèles et différents corpus de textes, en utilisant notament le score de cohérence.

La cohérence est une mesure qui s'applique sur chaque Topic identifié. Par conséquent, lorsqu'on identifie k topics, il se peut que certains topics aient une bonne cohérence, et que d'autres topics aient une mauvaise cohérence. Le but étant de maximizer la cohérence pour chaque topic identifié, il faut trouver un moyen d'aggréger les différents scores de cohérence. Ici, je choisi de retenir la cohérence moyenne des topics identifiés, ainsi que le coéficient de variation de ces topics. Mais comment évaluer la cohérence d'un topic?

Le score de cohérence d'un topic, consiste à évaluer la similarité entre les mots qui ont été identifié comme faisant partie d'un même topic. Je choisi donc les 10 mots les plus significatifs d'un topic, puis j'évalue 2 à 2, la similarité entre chaque paire de mots. Ce score de similarité est évalué en utilisant le score de $UMass$:

$$UMass(w_i, w_j) = log\Big(\frac{D(w_i,w_j)+\epsilon}{D(w_i)}\Big)$$

Où w_i et w_j sont deux mots différents d'un même topic, et où $D(w_i, w_j)$ et $D(w_i)$ sont respectivement le nombre de textes coutenant les mots w_i et w_j, et le nombre de textes contenant les mots w_j. Par défault ϵ vaut 1.

Le score de similarité d'un topic T est simplement la somme des $UMass$ de chaque paires de mots, parmis les 10 mots les plus significatifs du topic:

$$Coherence(T_i) = \sum_{i=1}^{10}\sum_{j=1}^{10} UMass(w_i, w_j)$$

Enfin, puisqu'il y a plusieurs topics évalués, je calcule la cohérence moyenne, puis le coéficient de variation des cohérences. S'il y a k topics:

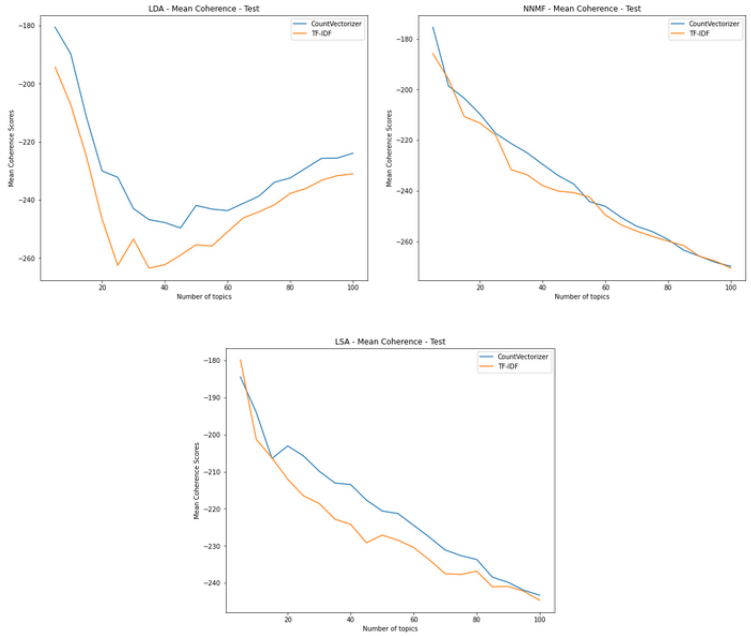
$$Average\_Coherence = \frac{\sum_{i=1}^{k} Coherence(T_i)}{k}$$

$$CV\_of\_Coherence = \frac{\sqrt{\frac{\sum_{i=1}^{k}\Big(Coherence(T_i)-Average\_Coherence\Big)^2}{k}}}{Average\_Coherence}$$

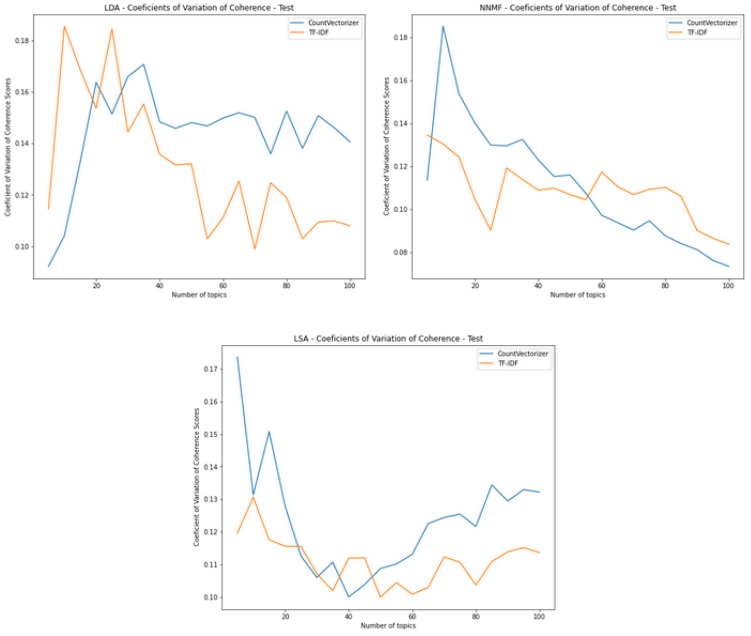
La cohérence moyenne sera toujours négative: plus elle est négative, moins les topics sont cohérents, et plus elle est proche de 0, plus les topics sont cohérents. La Coéficient de variation des cohérences - qui correspond à l'écart-type des cohérences, divisé par la moyenne des cohérences - est généralement entre 0 et 1. Plus la cohérence moyenne est élevée, plus les topics sont inégalement cohérents. Il est donc optimal d'avoir une cohérence moyenne la plus élevée possible, et un coéficient de variation de la cohérence le plus faible possible.

A présent je présente les résultats obtenus en utilisant les 2 méthodes de word embedding (TF-IDF et CountVectorizer), puis les résultats obtenus en utilisant les 3 méthodes pour générer des topics (LDA, NNMF, LSA).

3.2. TF-IDF, CountVectorizer



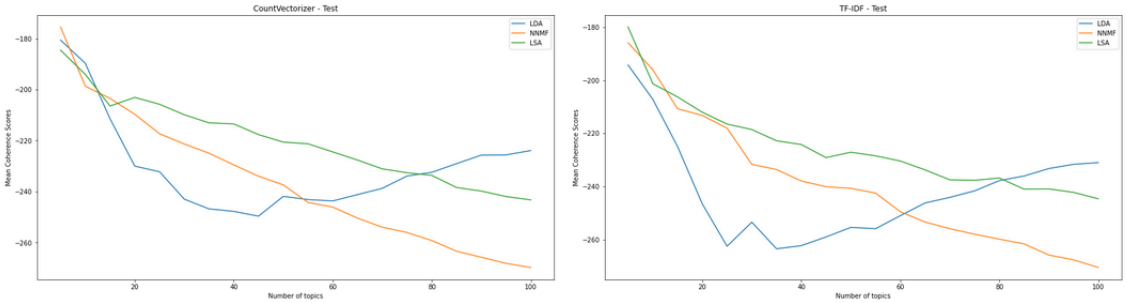
On voit ici que l'utilisation du CountVectorizer génère des score de cohérence moyens qui sont plus élevés que le TF-IDF, qu'ils soient appliqués sur LDA, NNMF et LSA. Ce résultat reste vrai, quel que soit le nombre de topic identifiés. A présent, on peut à présent regarder le coéficient de variation des cohérences des topics:



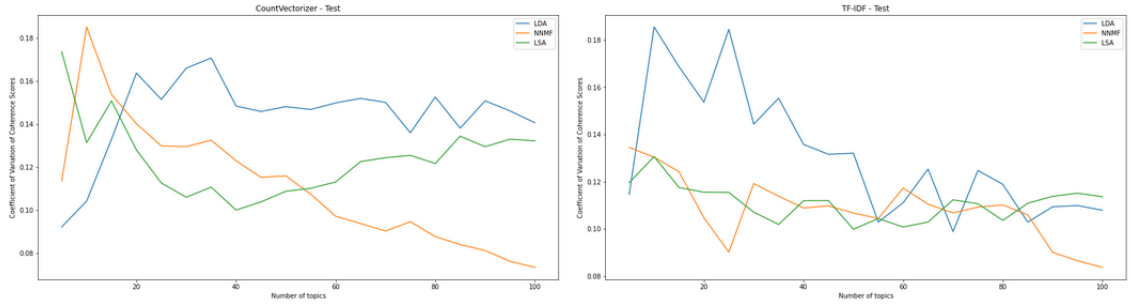
Ici on voit généralement avec le coéficient de variation de la cohérence des topics, qu'aucune des deux méthodes ne produit systématiquement une plus grande égalité dans la cohérence des différents topics.

Enfin, si l'on regarde le temps de calcul qui est mis avec le TF-IDF et avec le CountVectorizer, on remarque que le LDA fonctionne 30% plus vite avec le TF-IDF, et qu'à l'inverse le NNMF fonctionne 30% plus vite sur un CountVectorizer.

3.3. LDA, NNMF, LSA



On voit ici que le LSA génère des résultats qui sont plus cohérents que le NNMF, quel que soit le nombre de topics utilisés. Le LDA génère une meilleure cohérence moyenne que le LSA, seulement pour un grand nombre de topics identifiés.



En termes de coeficient de variation des topics, le LDA génère des topics dont la cohérence est plus inégalement répartie que le NNMF ou le LSA.

Enfin, en regardant le temps de calcul des 3 méthodes, on remarque que le LSA (20 minutes) est 3 fois plus rapide que le LDA (40 à 60 minutes) ou le NNMF (40 à 60 minutes).

Par conséquent, on peut affirmer ici que le LSA est une méthode qui est à la fois plus rapide en termes de calcul que les autres, et qui génère des topics plus cohérents que le LDA ou le NNMF, lorsque le nombre de topics n'est pas trop élevé. Etonnamment, le papier de Kherwa and Bansal (2020) trouve que les scores de cohérence observés avec le LSA sont moins élevés qu'avec le LDA et le NNMF, mais que les autres indices et ainsi que les évaluation de qualité sémantique faites par des êtres humains, confirmaient les meilleures performances du LSA. Il se peut donc que les topics de Stack Overflow soient naturellement plus facilement identifiables que les topics des données utilisées dans leur papier.

Enfin, il existe de nouveaux algorithmes permettant de construire automatiquement des topics, à partir de textes, sans qu'il n'y ait besoin de paramêtrer le nombre de topics, sans qu'il

n'y ait besoin de choisir une méthode de word embedding, et sans même qu'il y ait besoin de néttoyer les textes et de les organiser en *bag of words*. Dans la partie suivante, j'applique 2 de ces nouvelles méthodes (BERT et Top2Vec), et j'évalue leurs performances avec le score de cohérence.

4. Nouvelles Méthodes: BERT, Top2Vec

4.1. Top2Vec

Un des algorithmes les plus récemment développées est le Top2Vec (Angelov (2020)), qui effectue l'ensemble du travail en plusieurs étapes. La première étape consiste à représenter les mots et les documents dans un même espace, à savoir le *semantic embedding*. D'abord le *Top2vec* repose sur l'algorithme du *Word2Vec* pour calculer la probabilité que chaque mot du corpus soit suivi de chaque autre mot du corpus. Cette étape qui repose sur un petit réseau de neurones, permet de ne pas traîter les mots comme des entités seules, mais comme des bi-grammes ou n-grammes, dont on peut mesurer la similarité. Par conséquent, cela permet d'évaluer la similarité de 2 phrases, et plus largement de 2 textes. Dans un second temps, la méthode applique l'algorithme du *Doc2Vec* qui représente les textes (n-grammes avec de nombreux mots) et les mots (n-grammes avec un mot) dans un même espace multi-dimensionnel.

Une fois que ce *semantic embedding* est terminé, *Top2Vec* procède à une réduction de dimension avec le *Uniform Manifold Approximation and Projection* (UMAP). Enfin la dernière étape de l'algorithme consiste à appliquer un HDBSCAN pour identifier des clusters de textes et de points, dans l'espace réduit par UMAP. Chaque cluster va contenir des textes et des mots. Les différents textes d'un cluster formeront un topic, et les mots inclus dans ce cluster seront les features du topic.

J'ai appliqué cet algorithme à mes données nettoyées de Stack Overflow:

- Nombre de topics: 279
- Coherence Moyenne : -367.08
- Coéficient de Variation de la Coherence : 0.113

J'ai appliqué cet algorithme aux titres bruts (sans nettoyage) de Stack Overflow:

- Nombre de topics: 374
- Coherence Moyenne : -364.858
- Coéficient de Variation de la Coherence : 0.194

Les performances de Top2Vec sont inférieures aux algorithmes précédents, avec des scores de cohérence plus faibles, un coeficient de variation comparable, et un grand nombre de topics (qui n'est pas idéal pour une réduction de dimensionalité). Compte-tenu du grand nombre de topics, il est fort probable que certains topics soient très bien identifiés, et que d'autres le soient très mal.

4.2. BERT

L'algorithme de BERT, développé par Devlin et al. (2018) est rapidement devenu un des plus popoulaire en NLP. C'est une des variantes de BERT qui est appliqué ici, du nom de *BERTtopic*. BERT fonctionne d'une manière similaire à *Top2Vec*, mais avec pour différence dans la première étape de calculer non-seulement la probabilité du prochain mot, mais également la probabilité du mot précédent.: on parle d'un encoder bi-directionnel.

Ensuite, les textes sont représentés dans un espace multi-dimensionnel, sur lequel est appliqué un UMAP pour réduire le nombre de dimensions, puis un HDBscan pour identifier clusters. Enfin, contrairement au Top2Vec, les mots les plus importants de chaque cluster (donc de chaque topic), sont identifié en utilisant un TF-IDF au sein de chaque cluster.

J'ai appliqué BERT à mes données nettoyées de Stack Overflow:

- Nombre de topics: 321
- Coherence Moyenne : -341.208
- Coéficient de Variation de la Coherence : 0.080

J'ai appliqué BERT aux titres bruts (sans nettoyage) de Stack Overflow:

- Nombre de topics: 594
- Coherence Moyenne : -249.961
- Coéficient de Variation de la Coherence : 0.284

L'algorithme de BERT sur les titres brutes de Stack Overflow génère un très grand nombre de topics (594 sur titres bruts), avec un score de cohérence moyen élevé, et un coéficient de variation qui est également élevé. En regardant de plus près, l'algorithme de BERT génère des Topics qui sont extrêmement cohérents, et d'autres qui le sont moins, d'où le Coeficient de Variation de la cohérence des topics qui est élevé. En revanche, lorsqu'il est appliqué sur les mêmes bag of words que les autres, il génère des résultats comparables au Top2Vec.

5. Conclusion

Dans ce papier J'ai évalué les performances de plusieurs algorithmes de topic modeling (LDA, NNMF, LSA) combiné avec plusieurs méthodes de word embedding (TF-IDF, CountVectorizer). J'ai trouvé que le CountVectorizer génère des scores de cohérence moyen qui sont plus élevés (légèrement) qu'avec TF-IDF. En revanche l'utilisation du CountVectorizer ralentit légèrement le temps de calcul pour le LDA, mais accélère légèrement le temps de calcul avec le NNMF.

En comparant les performances du LDA, du NNMF et du LSA, j'ai remarqué que le LSA produisait des topics qui étaient en moyennes toujours plus cohérents que ceux identifiés avec un NNMF, quel que soit le nombre de Topics choisi. De même le LSA génère des topics plus cohérents que le LDA, sauf lorsqu'on choisi un grand nombre de topics (> 80). De plus, le temps de calcul du LSA s'est avéré 2 à 3 fois plus rapide que le LDA ou le NNMF.

Les résultats suggèrent donc que le *Latent Semantic Allocation* (LSA) est la méthode de réduction de dimenstionalité la plus commode et la plus efficace dans la majorité des cas.

Pour finir, j'ai appliqué des algorithmes plus récents (BERT et Top2Vec). L'avantage de ces méthodes est qu'elles effectuent l'intégralité du travail (nettoyage, identification des features, semantic embedding, réduction de dimensionalité, clustering). L'inconvenient de ces méthodes est qu'elles reposent sur des méthodes de clustering (HDBSCAN) qui ne permettent pas de sélectionner le nomber de cluster (et donc de topics). Appliqué sur les données de Stack Overflow, les deux méthodes ont identifié des centaines de Topics, certains avec une bonne cohérence, d'autres moins cohérents. En revanche, je trouve que sur des données brutes les performances de BERT sont significativement meilleures que celles de Top2Vec, qui performe moins bien que les méthodes plus classiques (LDA, LSA, NNMF).

References

Angelov, D. (2020). Top2vec: Distributed representations of topics. *arXiv preprint arXiv:2008.09470*.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

Chang, J., Boyd-Graber, J., Wang, C., Gerrish, S., and Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. In *Neural information processing systems*, volume 22, pages 288–296. Citeseer.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Kherwa, P. and Bansal, P. (2020). Topic modeling: a comprehensive review. *EAI Endorsed transactions on scalable information systems*, 7(24).

Stevens, K., Kegelmeyer, P., Andrzejewski, D., and Buttler, D. (2012). Exploring topic coherence over many models and many topics. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 952–961.