STIC-B545 - Traitement automatique de corpus

Etudiant

NGOUFACK NANFAH Angelot

Soumettez sur l'UV un court rapport (2-3 pages, format PDF) présentant brièvement vos découvertes

Plongé dans l'univers du Tac, cette semaine comme annoncée était la plus chargée, mais surtout la plus significative. Le clustering, le plongement lexical et l'entrainement de modèle sont à mettre en pratique après l' introduction de l'apprentissage non supervisé. Des méthodes du machine Learning, qui permettent à un ordinateur d'en apprendre lui-même sur les données qui lui sont fourni.

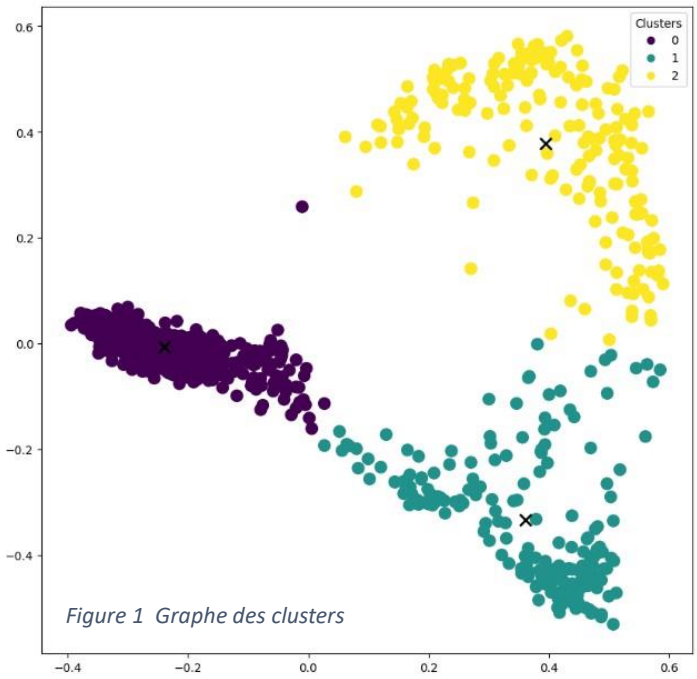

Figure 1 Graphe des clusters

D'entrée de jeu le clustering : avec la KMeans, une des méthodes centroïdes très simple dès l'abord il faut seulement choisir une variable K qui détermine le nombre de groupe souhaite, bien que ce n'est pas toujours optimal. Car un bon clustering nécessite de centres très distancés et des données concentrées autour de ces points.
Pour ce faire à partir d'une décennie « 1960 » sont les 1000 fichiers textes on applique un fonction de prétraitement et on les instancie avec tfidf pour les vectoriser. Puis on les transforme en matrice pour mieux représenter leur dimensions. Puis on applique la fonction fitpredict pour générer les cluster. A la base le nombre de dimension mais pour le visualiser on applique une autre fonction APC pour les réduire en 2 dimensions, mieux interprétable pour l'être humain.
Pour la décennie que j'ai choisie, j'émets la conclusion que K=3 est acceptable avec des centres éloignés et des donnes pour le centre du cluster **0** par exemple son très concentrées autour.
A noter que TF-IDF est ici pour voir dans quelle mesure le terme représente bien le document. Dont (max_df et min_df) sont à paramètre pour éliminer les stop-words. Un max_df de 0.9 tend à éliminer la totalité de stop-words.

Word embeddings plongement lexical et l'entrainement de modèle
Le Word embeddings ou plongement lexical prend forme dans le sens que chaque mot d'un corpus de textes est représenté par un vecteur, des nombres dans un espace vectoriel de dimension 'x' en se basant sur différents critères tel de contextualité. Il désigne un ensemble de méthode d'apprentissage visant à représenter les mots d'un texte par des vecteurs de nombres réels. Effectivement en se débarrassant du « fléau de la dimension » (curse of dimensionality) : on réduit les milliers de dimensions d'un mot à un nombre fixe moins élevé (dizaines/centaines) Intuitivement, les mots aux sens similaires seront proches dans l'espace vectoriel.
Word2Vec
Pour apprendre des termes similaires dans un corpus non supervisé donné, Word2Vec est très utilisé.
Que ce soit la sémantique, ou pour créer des incorporations de mots par des représentations sémantiques distributionnelles dans les applications NLP, Semantic Analysis, Text Classification et bien d'autres.

Mise à part, sa limitation uni-gramme. Word2Vec est très performant, pour prédire un mot donné en fonction de son contexte (CBOW), ou de prédire un contexte environnant en fonction d'un mot donné (Skip-Gram).

Dans la pratique, au point de départ on se sert d'un objet **phrases** pour extraire les N-grammes de mots formant un dictionnaire.

Notre corpus détient 15850647 clés qui sont des termes observe entre autres, possible grâce avec l'objet phrases. Et n'importe clé pris au hasard tombe sur un mot exemple la clé 76123 est docteur_, et son score d'occurrence est de « 2 ». Mais dans un contexte plus poussé l'outils **phraser** produit des clés sous forme groupe soit des bi-grammes ou tri-grammes concaténées pourvu que cela fasse du sens.

De ce fait en réduisant la dimension de la matrice de capturer le contexte, la similarité sémantique et syntaxique (genre, synonymes, …) d'un mot. De faire de arirthmeqtique dans la mesure du possible. Dans mon modele du tp avec word2vec on peut apprécier ces propriétés.

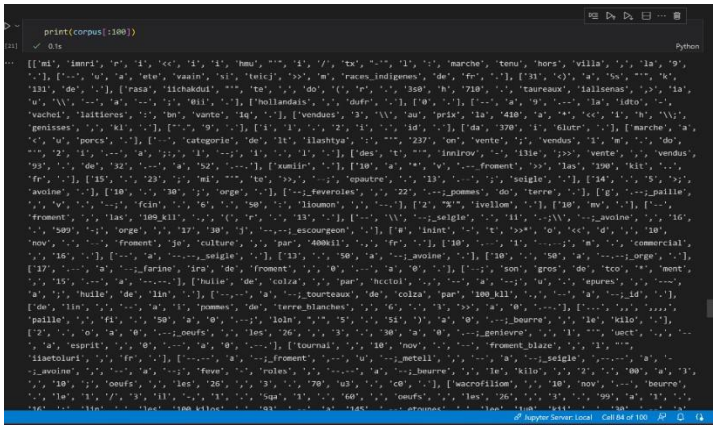

Figure 2 - Visualisation de 100 clés extraites par Phraser

Puis vient l'entrainement du modèle.

Du corpus généré , word2vec en mode exécution entraine le modèle. Avec bien sur des paramètres à fixer : dans mon cas je les ai définis ainsi :

- Le nombre de dimensions dans lesquelles le contexte des mots devra être réduit, mis a 32 une limite raisonnable vector_size=32,
- La taille du "contexte", ici 5 mots avant et après le mot observé window=6,
- On ignore les mots qui n'apparaissent pas au moins 6 fois dans le corpus min_count=6,
- Pour paralléliser l'entraînement du modèle en 3 threads workers=3,
- Nombre d'itérations du réseau de neurones sur le jeu de données pour ajuster les paramètres avec la descente de gradient. epochs=5,

Dans la pratique Word2vec, on utilise un réseau de neurones à 3 couches (1 couche d'entrée, 1 couche cachée, 1 couche de sortie) :

Observation :

1) Impression de vecteur de mots sachant que le vector-zise est 32.

La figure 3 montre le vecteur du terme « Bruxelles » avec un matrice de 32 float.

Figure 3 visualisation d'un vecteur @ 32 dimensions « Bruxelles »

2) Calcul de la similarité entre 2 termes données.

	Coefficient	Commentaires

model.wv.similarity("black", "noir")	0.6086001	Le contexte de l'utilisation de « black » selon moi perd le sens de couleur pour se calquer des mots anglais dans le corpus. ('love', 0.8405863046646118), ('duke', 0.8402025103569031), ('sky', 0.8389651775360107), ('lady', 0.838699460029602),
model.wv.similarity("invasion", "immigration")	0.8164259	Cette similarité n'est pas évidente, j'ai des réserve par rapport.
model.wv.similarity("droit", "privilege")	0.8898432	Un bon cas de similarité, mais encore il y a lieu de se questionner. Si on regarde les 10 termes les plus proches. ('refus', 0.8388926982879639),
		('visa', 0.8321605324745178), ('consentement', 0.827886700630188), ('pouvoir', 0.8267961740493774),
model.wv.similarity("policier", "gendarme")	0.93085563	Un bon cas de similarité, simple à comprendre.
model.wv.similarity("police", "violence")	0.58114463	Mais cela devient vite questionnable. Violence est plus proche de policier que le mot ordre.
model.wv.similarity("police", "ordre")	0.48012277	
model.wv.similarity("pere", "papa")	0.7287459	Selon moi le principe de l'arithmétique des vecteurs prend un coup.
model.wv.similarity("mere", "maman")	0.8472999	

3) Recherche de mots les plus rapproches d'un terme donnée.

model.wv.most_similar("soir", topn=10)	model.wv.most_similar("noir", topn=10)	model.wv.most_similar("foot", topn=10)
[('som', 0.8401994705200195), ('soih', 0.8251874446868896), ('matin', 0.815732479095459), ('soib', 0.8002303242683411), ('mutin', 0.7949638366699219), ('malin', 0.794888973236084), ('fascicule', 0.7852529287338257), ('jour', 0.7813154458999634), ('careme', 0.7802088260650635), ('feuilleton', 0.7775447964668274)]	[('gris', 0.9457598924636841), ('blanc', 0.9406753778457642), ('brun', 0.907046377658844), ('bleu', 0.8958034515380859), ('jaune', 0.886952817440033), ('vert', 0.8822004795074463), ('blano', 0.8741109371185303), ('marron', 0.8734044432640076), ('fonce', 0.8708969354629517), ('ecossais', 0.8577011227607727)]	[('basket', 0.8813017010688782), ('ball', 0.8323959708213806), ('volley', 0.8300209641456604), ('matoh', 0.7948707342147827), ('lawn', 0.7844277024269104), ('qui_opposera', 0.7791548371315002), ('tcnnis', 0.7770369052886963), ('cricket', 0.7731530070304871), ('leopold_club', 0.7665836215019226), ('cross', 0.7646769881248474)]
Je m'attendais pour matin et jour des négatifs. Mais l'explication est de mise.	**Là je me perd.**	**On comprend que c'est régional**

4) Effectuer des recherches complexes à travers de corpus vectorise. C'est exécuté dans VS code.

En somme c'était une exercice attrayant et très pratique, 1960 peut être clustériser moyennant trois groupes. L'entrainement du modèle a pris toute une journée si on compte le nombre de fois que le PC s'est planté. Néanmoins je peux dire que j'ai beaucoup appris sur l'apprentissage non supervise en machine Learning.

PS : Vous noterez que j'ai 2 notebook pour le tp3 , c'est juste pour faire le test avec spikle en ce qui a trait à la sauvegarde du modèle.