Université Paris Descartes

Module : Sciences des données

Synthèse et Implémentation

How to make words with vectors: Phrase generation in distributional semantics

Réalisé par :

Ait Bachir Samy

Ayad Wafa

Hernandez Matthieu

I. Introduction

Dans ce projet, il s'agit de lire, comprendre, résumer puis implémenter une publication dans le domaine la sémantique distribuée, rédigé par Georgiana Dinu et Marco Baroni chercheurs à l'université de Trento en Italie. L'intitulé de ce papier, daté de 2014, est « How to make words with vectors: Phrase generation in distributional semantics » qui traduit en français peut être lu « Comment créer des phrases à l'aide de vecteurs : La génération de phrase dans la sémantique distribuée ».

Ce rapport est divisé en deux majeurs parties :

1. Un bref résumé de l'article, mettant en exergue les propositions des chercheurs et leurs explications.
2. Une partie ou sera expliquée l'implémentation en python du concept, les librairies utilées et les problèmes rencontrés.

Pour commencer donc, dans le chapitre suivant, nous proposons une synthèse de l'article en français, qui se veut la plus compréhensible possible.

II. Synthèse de la publication

Dans la publication intitulée « How to make words with vectors: Phrase generation in distributional semantics », Georgiana Dinu et Marco Baroni parlent de la génération de phrases grâce à la méthode dite sémantique distribuée.

En introduction, Dimu et Baronmi nous explique que les méthodes de sémantique distribuée se base sur une approximation du sens d'une expression sous forme de vecteurs, et que plus récemment, des opérations de composition ont été utilisés sur deux vecteurs de sens, pour générer un vecteur approximant le sens combiné des deux vecteurs sources.

Toujours dans l'introduction, les deux chercheurs de l'université de Trento posent la problématique de la bidirectionnalité de la relation entre sens et expression. Dans cette optique la méthode sémantique distribuée doit être capable à partir de vecteurs de sens, de générer des phrases ou expression ayant le sens sémantique visé. Par la suite, ils énumèrent quelques cas pratiques d'utilisation de génération de phrases à partir de vecteurs de sens, par exemple dans la traduction ou description d'images. Par la suite, ils expliquent que la génération d'un seul mot à partir d'un vecteur est simple, il suffit de chercher dans le lexique sur lequel on s'appuie le mot correspondant au vecteur le plus proche du vecteur générateur, et cette méthode est appelée « Génération par synthèse ». Cependant cette méthode n'est pas applicable pour la génération de phrases, car on se heurte à très vite à une explosion combinatoire. D'où le but de ce papier : effectuer une décomposition d'un vecteur en une séquence de vecteurs plus petit, qui sont soit matché au lexique pour générer un mot, ou de nouveau décomposés (donnant des phrases plus longues).

Suivant l'introduction, Dinu et Baroni expliquent qu'à leur connaissance, aucune autre recherche similaire à la leur n'a été effectuée précédemment. Ils s'attellent cependant à énumérer certains papiers publiés dans le même domaine de recherche. Parmi ces derniers, ils parlent de la publication de Andreas and Ghahramani en 2013, ou les chercheurs ont mis au point une technique probabiliste ayant certaines lacunes au niveau de la génération de phrases.

Pour la suite, dans le 3e paragraphe de la publication, ils rentrent dans le vif du sujet expliquent tout d'abord le concept connu de composition de vecteurs de sens, tel que défini par Guevara (2010) puis Zanzotto et al. (2010) : la composition de deux vecteurs de sens est l'application de combinaisons linéaires sur les deux vecteurs, combinaison linéaire dont l'équation est équivalente à :

$$\mathrm{f}_{\mathrm{comp_R}}(\vec{u}, \vec{v}) = W_R[\vec{u}; \vec{v}]$$

Ou *u* et *v* sont les vecteurs de sens des mots à combiner (Concaténés verticalement), et W est une matrice, relative à une relation syntaxique r, que l'on obtient à partir d'un corpus, en résolvant le problème des moindres carrés suivant :

$$\min_{W_R \in \mathbb{R}^{d \times 2d}} \|P - W_R[U; V]\|$$

La ou U et V sont des matrices composées de vecteurs de sens de suite de mots satisfaisant la relation syntaxique r, et P est la matrice des de combinaison vecteurs observés des chacune des combinaisons de mots représentées par les vecteurs de U et V.

Les deux chercheurs justifient en suite le choix de l'approximation de vecteurs observés de phrases, par le fait que cette méthode permet d'appréhender les mots possédant plusieurs sens selon le contexte dans lequel ils apparaissent.

Ils enchaînent en suite par la proposition d'une méthode de décomposition similaire ayant pour but de générer des phrases (de deux mots pour commencer). Cette méthode de décomposition se base sur la résolution du problème des moindres carrés défini par :

$$\min_{W'_R \in \mathbb{R}^{2d \times d}} \|[U; V] - W'_R P\|$$

Ou la matrice W'r est obtenue exactement de la même de la même façon que la matrice Wr dans la composition.

Pour éviter l'apprentissage sur un corpus, et dans le cas ou nous avons déjà la matrice de composition pour la relation syntaxique r, il est possible d'essayer d'inverser le processus de combinaison, et ce en résolvant le problème des moindres carrés :

$$\min_{W'_R \in \mathbb{R}^{2d \times d}} \|[U; V] - W'_R W_R[U; V]\|$$

Les deux méthodes sont dites utilisable, cependant la première est plus apte à détecter les différents sens que peut avoir un mot, alors que la seconde inversera plus fidèlement une composition effectuée à l'aide de la matrice Wr, ce qui la rend très attractive pour le passage d'une langue à une autre par exemple.

Une fois la décomposition expliquée, Dinu et Baroni montrent la façon de générer des phrases à l'aide de cette méthode. Basiquement, en décomposant un vecteur de sens p en deux vecteurs de sens u et v, ils arrivent à former une phrase de deux mots en utilisant, par deux fois (pour chacun de u et v), la méthode du plus proche voisin, voisins disposés dans un lexique ou chaque vecteur est appairé à un mot et vice versa, trouvant ainsi les deux mots qui composent donc la phrase.

Avant de passer à la phase de test, une dernière chose est mentionnée par les deux chercheurs : il s'agit de la composition et décomposition de plus grande phrase. Pour effectuer cette tâche, ils

proposent une approche triviale récursive. Plus clairement, pour composition de plus de deux vecteurs, il suffit de combiner les deux premiers, puis de combiner le résultat de cette combinaison avec le 3^{e} vecteur, et ainsi de suite. Pour la décomposition, on décompose le premier vecteur en 2 vecteurs, puis on décompose ceux-ci encore une fois, puis on répète encore cette opération, jusqu'à atteindre la longueur de phrase souhaitée.

Par la suite, Dinu et Baroni détaillent la construction de leur environnement de test. Ils utilisent tout d'abord deux méthodes pour la création des vecteurs de sens de mots : le modèle *cbow,* pour lequel ils utilisent l'outil de réseaux de neurones Word2Vect, et un modèle standard appelé count basé sur la construction de matrices de cooccurrence de mots dans une fenêtre symétrique de 2 mots (2 mots avant et deux mots après). Pour les deux approches, ils disposent de corpus de 2,8 milliards de tokens en anglais, de plus qu'un corpus 1,6 milliards de token en Italien, pour tester le passage d'une langue à une autre. Puis ils illustrent leur proposition en testant tour à tour la composition, la décomposition simple, la décomposition récursive, puis la traduction.

Pour terminer, Dinu et Baroni assurent la qualité de leur proposition preuve à l'appui, admettent qu'il reste fort à faire et se positionnent dans le milieu de la recherche, soulignant l'importance théorique de leur travail.

III. Implémentation

Pour l'implémentation en Python des méthodes de composition et décomposition de vecteurs de sens données dans l'article résumé dans le chapitre précédent de ce document, plusieurs librairies ont été utilisées, certains problèmes d'implémentation ont été rencontrés, et plusieurs classes ont été créées. Lors de ce chapitre nous allons détailler chacun de ces points.

III.1 Librairies utilisées

L'une des forces du langage de programmation Python est la disponibilité de librairies très variées, et qui nous permette de ne pas avoir à réinventer la roue à chaque fois que l'on doit implémenter un programme.

Lors de l'implémentation de ce projet, nous avons eu besoin de réaliser certaines opérations sur des données textuelles et sur des représentations matricielles de ces données-là, pour effectuer ces opérations nous avons eu recours aux librairies suivantes :

- **NLTK :** La librairie *Natural Language ToolKit* est l'une des bibliothèques les plus connues dans le domaine du traitement automatique du langage naturel, elle propose des fonctionnalités très variées, allant des stemmers (Récupération de racines de mots) aux POSTagger (Étiqueteurs morpho-syntaxiques). Nous utilisons NLTK dans ce projet pour les corpus de textes étiquetés qu'il propose.
- **NumPy :** Tout aussi connue que la librairie NLTK, NumPy est sans doute la bibliothèque la plus connue par les programmeurs et les chercheurs qui sont amenés à implémenter des calculs algébriques simples ou complexes. Ici NumPy est utilisée pour ses méthodes de calculs de produits matriciels, et pour sa méthode de résolution de problèmes de moindres carrées.
- **Gensim :** Moins connue que les deux précédentes librairies, Gensim propose plusieurs modèles de machine learning basés sur des travaux réalisés par des chercheurs. Nous n'utilisons que la méthode Word2Vec de la librairie, permettant la création de vecteurs de sens de mots suivant la méthode *cbow*.

III.2 Problèmes rencontrés

Au début de la réalisation du projet, l'idée était de ré-implémenter aussi fidèlement que possible les méthodes proposées par Dinu et Baroni ainsi que l'environnement d'évaluation qu'ils proposent. Cependant nous nous somme très vite heurtés à des barrières difficilement franchissables. Ces problèmes sont énumérés de façon non-exhaustive dans ce qui suit :

- En premier lieu, nous devions comprendre les méthodes de création de vecteurs de sens que les deux scientifiques ont utilisés. Dinu et Baroni donnent les noms des méthodes en laissant le soin au lecteur d'aller voir les articles publiés précédemment au sujet de ces méthodes. Par manque de temps nous avons décidé de sélectionner l'une des deux méthodes, et notre choix c'est posé sur la méthode *cbow* (Celle qui apporte de meilleurs résultats selon l'article). *Cbow*

se base sur un réseau de neurones sur 3 couches (Une couche d'entrée, une couche cachée et une couche de sortie) pour créer les vecteurs de sens de mots. Le nombre de neurones utilisés est déterminé par la dimension du vecteur de sens voulue. Dans la littérature, il est dit que des vecteurs de sens de dimension 100 à 300 sont à considérer, plus le nombre de token disponibles pour l'entraînement est grand, plus on s'approche de la borne supérieure de cet intervalle pour avoir de meilleurs résultats.

- Le deuxième problème rencontré est l'implémentation d'un solveur de problème de moindres carrés matriciel. En effet sur les formules proposées par Dinu et Barouni se rapportent à un problème B – XA, hors dans le cas matriciel B – XA est différent du problème classique et implémenté par les librairies B – AX, car le produit matriciel n'est pas une opération commutative. Si les deux chercheurs sont restés fidèles à la description de leurs formules, ils ont du implémenter un solveur, résultat d'une annulation de dérivée, trop complexe mathématiquement pour que l'on s'y attarde dans le temps imparti. Le contournement du problème proposé se base une inversion des formules proposées, là où la formule de composition utilisées dans l'article s'écrit sous la forme P = W [U;V], nous écrivant notre formule sous la forme P = [U;V] W. Ceci respecte la logique posée par les chercheurs, tout en facilitant le calcul de minimisation de l'erreur d'approximation de l'apprentissage en retombant sur une forme classique du problème des moindres carrés en B – AX, dont la solution est implémentée par plusieurs librairies, dont *NumPy*.
- Un 3^{e} obstacle rencontré est l'obstacle de la puissance de calcul et l'espace mémoire : Dinu et Barouni ont effectué leurs tests sur des corpus gigantesques, de 2,8 milliards de token en anglais (en combinant ukWaC, Wikipédia et BNC), et 1,8 milliards de token en Italien (itWaC). Sur des machines conventionnel il est impossible d'exécuter les différents algorithmes sur d'aussi grands ensemble de données. Pour contourner ce problème, nous proposons de tester les composition et décomposition sur une seule langue, en entraînant le réseau de neurones Word2Vec et en calculant les matrices de composition et décomposition sur le corpus de mots étiquetés morpho-syntaxiquement *Brown* proposé dans la librairie NLTK, en se limitant, encore une fois par manque de puissance de calcul, sur la catégorie science-fiction de ce corpus.
- Un autre problème auquel nous sommes opposés lors de l'implémentation de ce projet, problème qui n'est pas des moindres, est que Dinu et Baroni ne mentionne à aucun moment comment ils obtiennent les vecteurs de sens de phrases composées de plusieurs mots pour calculer les matrices W de composition ou décomposition. Les implémentations classiques de Word2Vec ne permette en aucun cas de réaliser un réseau de neurones donnant des représentations vectorielles à une phrase. Les recherches que nous avons effectués nous conduisent à un paradoxe du type « l'œuf ou la poule » : pour pouvoir obtenir une méthode de composition, nous avons besoin d'une composition effectuée par une méthode de composition. Pour palier à ce problème, nous proposons de créer de nouvelles phrases à partir des phrases du corpus, en reliant deux à deux les mots les composant. En d'autres termes pour une phrase du corpus de taille N, N-1 phrases sont générées, chacune d'entre elle est similaire en tout point à

la phrase de base, sauf que deux mots y sont reliés par un « _ » comme pour l'exemple suivant :
This is the best moment of my life donne :

- *This_is the best moment of my life*
- *This is_the best moment of my life*
- *This is the_best moment of my life*
- *This is the best_moment of my life*
- *This is the best moment_of my life*
- *This is the best moment of_my life*
- *This is the best moment of my_life*

Cette méthode permet d'entraîner le réseau de neurones sur des bigrammes et avoir leurs vecteurs de sens comme s'ils n'étaient qu'un mot unique, qui seront utilisés comme phrase à 2 mots pour calculer les matrices de composition et décomposition des règles syntaxique représentées par les concaténations des étiquettes morpho-syntaxiques des mots composant le bigramme. Évidement cette opération est un gouffre de ressources et c'est ce qui nous a poussé à n'utiliser que la catégorie science-fiction du corpus Brown de NLTK.
Cette concaténation de mots en bigrammes et la gestion de leurs étiquettes sont implémentés dans la classe *NltkBigrams*, détaillée plus loin.

III.3 Classes implémentées

Python est un langage orienté objets, il va de soit que l'implémentation d'un projet tel que celui-ci se fasse sur plusieurs classes. Ces différentes classes sont :

- **NltkBigrams :** Cette classe a été créer dans le seul et unique but de pouvoir obtenir des représentations vectorielles de bigrammes de corpus NLTK ou basés sur ces corpus. Les méthodes de cette classe permette la génération de bigrammes à partir des mots de corpus NLTK, de bigrammes étiquetés par la concaténation des étiquettes des mots composant les bigrammes en question, de phrases contenant des mots simple et un bigramme (Voir chapitre des problèmes rencontrés) et de ces mêmes phrases ou les mots et les bigrammes sont étiquetés mopho-syntaxiquement.
- **DistributionalSemantics :** Cette classe contient un ensemble de méthodes statiques permettant de calculer les matrices de composition et de décomposition, et permettant également de calculer des compositions de vecteurs de sens (de mots ou de phrases) ainsi que des décompositions de vecteurs de sens de phrases et le mot le plus proche au sens vectoriel.

Finalement un fichier main.py a été implémenter pour permettre le test des méthodes implémentées. Par manque de temps et de puissance de calculs ce ne sont que des tests sommaires.