MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABBES LAGHROUR KHENCHELA

FACULTE DES SCIENCES ET DE LA TECHNOLOGIE

Département de mathématique et informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master (L.M.D)

Spécialité : Informatique

Option : sécurité et technologies web

Représentation sémantique en utilisant Word embeddings pour la désambiguïsation lexicale

Réalisé par :

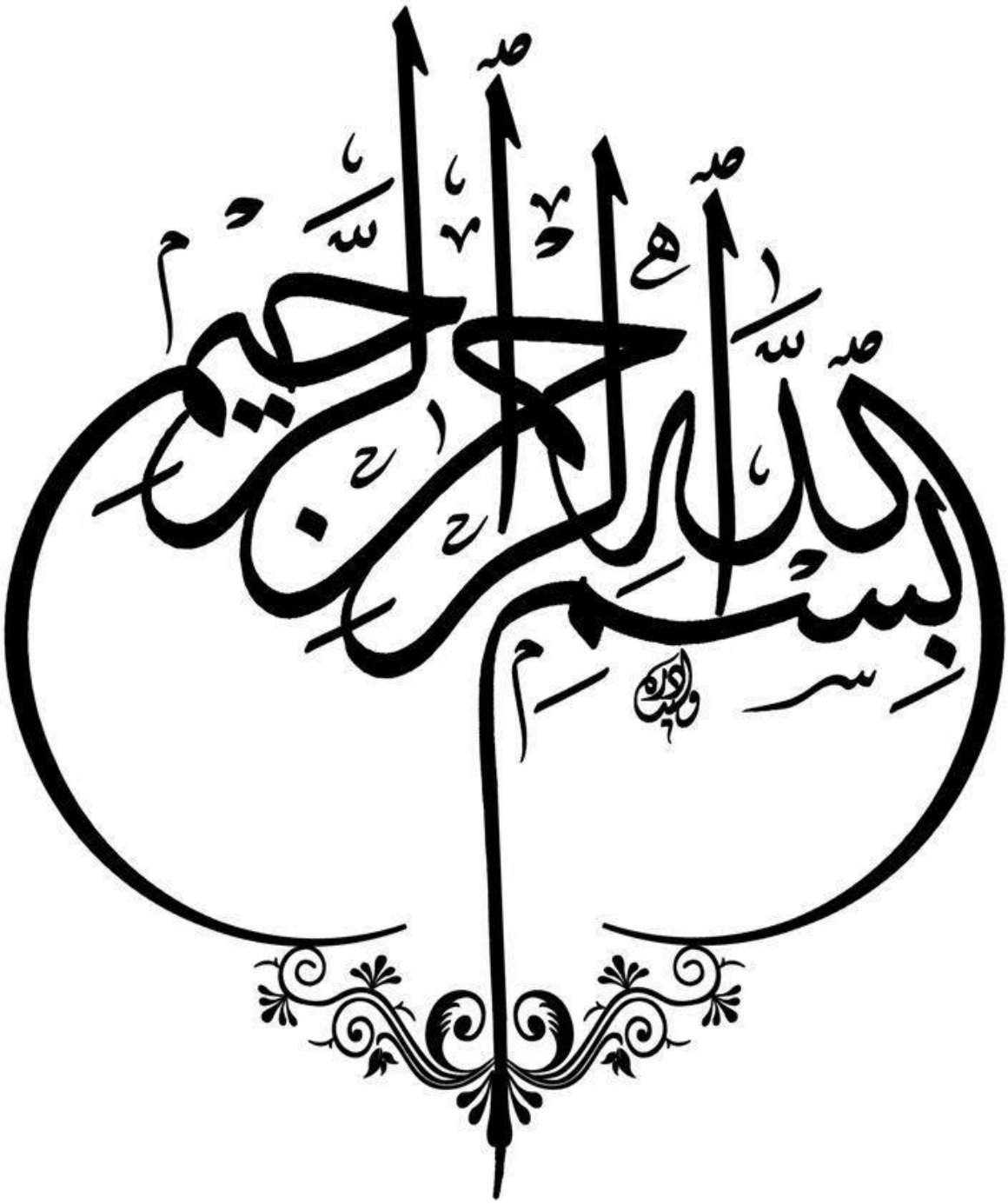
Khalfaoui Nora.

Mordjane nessrine .

Dirigé par :

Dr. Bakhouche. Abdelali.

Année Universitaire : 2020/2021


بسم الله الرحمن الرحيم

Dédicace

A nos très chers parents,

A nos chers frères et sœurs,

A nos Enseignants

A nos collègues de travail,

A nos meilleurs amis.

Nous dédions ce mémoire

.

Remerciement

Nous tenions à remercier Monsieur ABDELAALI BEKHOUCHE, qui nous a encadrés Tout au long de cette thèse et qui nous a fait partager ses brillantes intuitions. Qu'il soit aussi remercié pour sa gentillesse, sa disponibilité permanente et pour les nombreux encouragements qu'il nous a prodiguée.

Résumé

La désambiguïsation lexicale (DL) est une tache centrale dans traitement automatique des langues (TAL), basé sur mesurer la similarité sémantique est la base de nombreuses applications, elle joue un rôle important dans divers domaines tel que la recherche d'information, la traduction automatique, l'extraction d'information ou la détection de plagiat.

Dans ce mémoire, nous proposons une approche fondée sur le plongement de mot (Word embedding). Ce système est destiné de mesurer la similarité sémantique entre les vecteurs de sens des voisins et les vecteurs de sens des mots ambigües, l'idée principale est d'exploiter la représentation des sens des mots par des vecteurs dans l'espace multidimensionnel.

Mots clés : La désambiguïsation lexicale (DL), traitement automatique des langues (TAL), Word embedding.

Abstract

Lexical disambiguation (DL) is a central task in automatic language processing (NLP), the basis of countless applications and plays an important role in diverse areas, such as information retrieval, plagiarism detection, information extraction and machine translation.

In this chapter propose an approach based on word embedding. This system is intended to measure the semantic similarity between the meaning vectors of neighbors and the meaning vectors of ambiguous words, the main idea is to exploit the representation of the meanings of words by vectors in multidimensional space.

ملخص:

يعد توضيح الغموض المعجمي (DL) مهمة مركزية في المعالجة التلقائية للغة (NLP) ، استنادًا إلى قياس التشابه الدلالي وهو أساس العديد من التطبيقات ، فهو يلعب دورًا مهمًا في مجالات مختلفة مثل استرجاع المعلومات أو الترجمة الآلية أو استخراج المعلومات أو اكتشاف الانتحال . في هذه الأطروحة ، نقترح نهجًا يعتمد على تضمين الكلمات. يهدف هذا النظام إلى قياس التشابه الدلالي بين متجهات معنى الجيران وناقلات المعنى للكلمات الغامضة ، والفكرة الرئيسية هي استغلال تمثيل معاني الكلمات بواسطة المتجهات في الفضاء متعدد الأبعاد

Table des matières

Chapitre I: La désambiguïsation lexicale

Chapitre II: Word embedding

Chapitre III: Conception et réalisation d'un système de désambiguïsation sémantique

Listes des figures

Liste des tableaux:

Abréviations

- WE: Word Embeddings
- CBOW: Continious Bag Of Words
- TAL : Traitement Automatique de la langue
- ADT : Analyse de données textuelles
- RD : Recherche Documentaire
- TM : TextManing
- STS : SemanticTextualSimilarity
- NLTK: Natural Language Toolkit
- CNRI: Corporation for National Research Initiatives
- TF-IDF: Term Frequency–Inverse Document Frequency
- Word2vec : représentation de mot par un vecteur
- POS : Partie du discours

Introduction générale

Introduction générale

Les langues naturelles sont caractérisées par l'omniprésence de l'ambiguïté lexicale qui constitue l'une des sources de richesse et de souplesse des langues. Dans la communication interhumaine, l'ambiguïté ne présente pas un réel problème, mais dans le cadre de traitement automatique de la langue, l'ambigüité lexicale constitue toujours un défi. L'ambiguïté est la propriété d'un mot, d'une suite de mots ou d'un concept ayant plusieurs significations ou plusieurs analyses grammaticales possibles. On peut rencontrer différents types d'ambigüité en fonction de niveau d'analyse linguistiques (morphologique, syntaxique, sémantique, pragmatique). Les difficultés liées à la problématique de la désambiguïsation sémantique ont très tôt été identifiées. Toutefois, les solutions qui ont été proposées dans chaque domaine sont également multiples et très diverses, en fonction des besoins et des savoirs afférents à chacune des matières concernées. La définition du problème elle-même ne fait pas l'unanimité. En effet, si un consensus est atteint pour définir la désambiguïsation sémantique comme l'association d'un mot apparaissant dans un contexte avec sa signification ou sa définition qui peut être distinguée des autres définitions qu'on peut attribuer à ce mot, en revanche le même accord n'existe pas pour d'autres tâches. La désambiguïsation lexicale est une pierre de base pour de nombreuses applications du traitement automatique des langues. Le processus de désambigüisation lexicale (Word Sense Disambiguation) consiste à sélectionner les sens corrects d'instance contextualisé des mots ambigus, parmi l'ensemble de leurs sens possibles. Notre travail s'inscrit dans le cadre des études visant à la désambiguïsation lexicale de manière automatique. Il existe aujourd'hui plusieurs approches

proposées pour représenter le sens tel que le Word Embeddings qui est une méthode

d'apprentissage d'une représentation de mots utilisée notamment en traitement automatique des langues (TAL). Le principe général de cette technique est la représentation de chaque mot d'un dictionnaire par un vecteur de nombres réels.

➢ Ce manuscrit s'articule autour de trois chapitres :

✓ Nous revenons d'abord dans le premier chapitre de présenter et d'expliquer le concept principal de la désambiguïsation.

✓ Ensuite dans le second chapitre on va présenter et d'expliquer le contexte d'étude et le concept de base utilisés dans ce travail le Word embeddings, avant de décrire, d'une manière plus approfondie, les différents types de Word Embeddings.

✓ Dans le troisième chapitre, on va présenter notre propre modèle, nous illustrerons les phases nécessaires pour sa conception et sa réalisation, en utilisant des exemples illustratifs.

Chapitre I :

La désambiguïsation lexicale

1.1 Introduction :

La désambiguïsation lexicale est une partie très importante pour le traitement automatique des langues, qui permet d'améliorer de nombreuses applications en traitement automatique des langues (TAL) comme la recherche d'information, la traduction automatique, la simplification lexicale de textes, l'extraction d'informations multilingues et le résumé automatique.L'objectif de désambiguïsation lexicale a déterminé le sens le plus proche de chaque mot du texte dans un inventaire prédéfini.Par exemple dans la phrase « La souris mange le fromage. », l'animal devrait être préféré au dispositif électronique.

Dans ce contexte le sens le plus proche a le mot souris c'est un préféré.

1.2 Désambiguïsation lexicale

Les mots que l'on emploie peuvent avoir plusieurs sens, qui sont déduits généralement de leur contexte. Par exemple : « La souris mange le fromage. », on pense tout de suite, pour le mot souris, au sens du petit rongeur, plutôt que celui du périphérique informatique [Vial,2020]. L'être humain définit directement le sens souhaité, tandis que la machine trouver une ambiguïté, pour cela on propose une approche pour lever l'ambiguïté de mots dans un contexte quelconque, en utilise Word embedding.

Dans ce chapitre, nous allons d'abord faire un rapide historique de la tâche et décrire le processus de mise en œuvre, avant de voir quelles sont les différentes approches pour la DL ainsi que les ressources nécessaires à leur mise en œuvre., et enfin les méthodes pour leur évaluation.

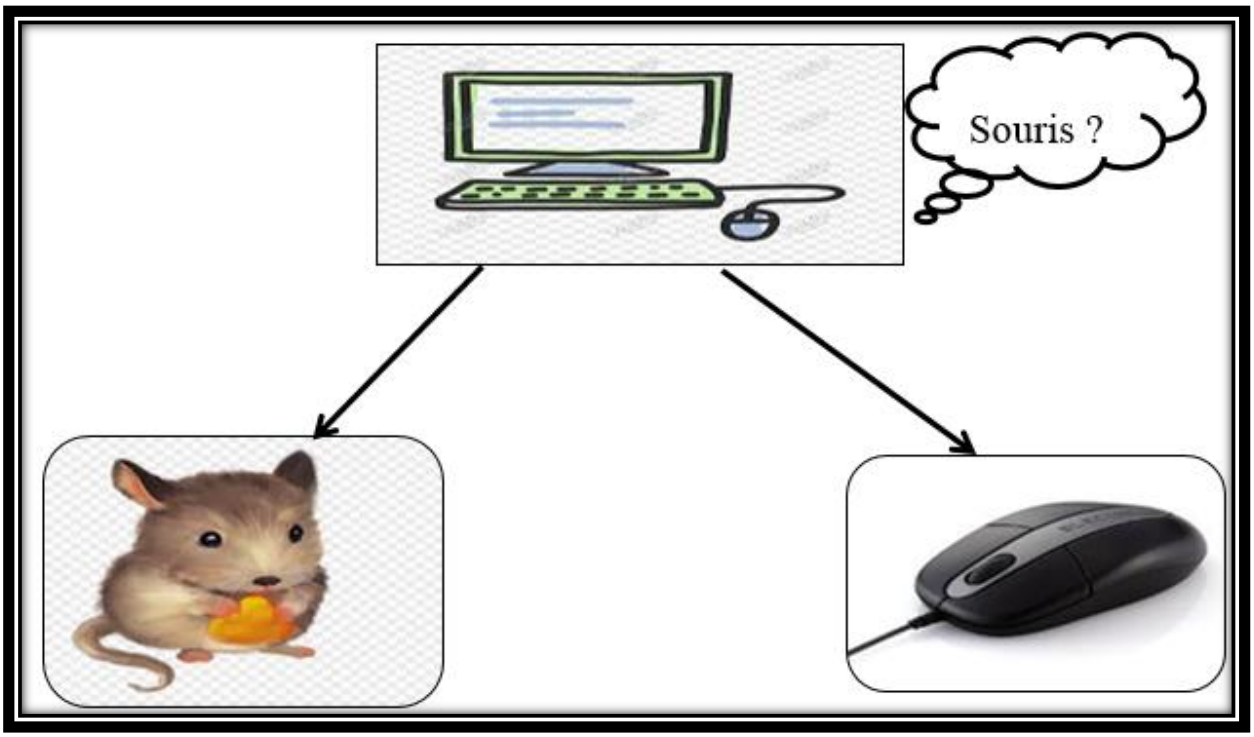


Figure 1.1: l'ambigüité par a pour a la machine

1.3 Historique

La désambiguïsation lexicale, comme elle concerne une notion fondamentale qui est le sens, est un élément central dans de nombreux champs du TAL et pour la compréhension de la langue en général. Comme l'écrit Navigli (2009), elle apparaît d'abord comme une des tâches principales pour la traduction automatique dès les années 1950 (Weaver, 1955), et elle a ensuite été considérée très rapidement comme un problème à part entière au vu de sa grande difficulté. En effet, même si la DL peut être vue comme un simple problème de classification (tel sens est assigné à tel mot dans tel contexte), elle soulève en fait de nombreuses questions fondamentales qu'il est nécessaire de se poser. Par exemple :

Comment définir précisément un sens ? Est-ce que l'existence d'un sens dépend d'une langue ou d'un domaine d'application ? Quel niveau de granularité est souhaité pour distinguer des sens proches ? etc.

Dans les premiers travaux sur la DL, cette difficulté était accrue par le manque de ressource sémantique disponible. Sans inventaire de sens standard, base de données

lexicale ou campagne d'évaluation, la comparaison des systèmes n'était pas évidente, et cette situation est d'ailleurs toujours d'actualité pour la majorité des langues autres que l'anglais (voir par exemple les travaux de Hadj Salah et al.(2018) sur la DL de l'arabe).

C'est dans les années 1990 qu'arrivent les premières bases de données lexicales, et notamment Word Net (Miller et al, 1990), l'inventaire de sens pour l'anglais de référence, toujours utilisé de nos jours dans la plupart des travaux de la DL. Dans la même période est créé le premier corpus annoté en sens à grande échelle, le SEM Cor (Miller et al., 1993), puis les premières campagnes d'évaluation comme SensEval (Kilgarriff, 1998), qui permettent finalement la création et l'évaluation de systèmes de DL de façon automatique et à grande échelle. Il existe aujourd'hui une multitude d'approches pour la DL, qu'on différencie principalement par la nature et par la quantité des données sur lesquelles elles s'appuient. On trouve ainsi principalement d'un côté les approches supervisées, et de l'autre, les approches à base de connaissances. Elles, s'appuient sur des ressources lexicales telles que des dictionnaires, des thésaurus ou des réseaux lexicaux, couplées à des méthodes telles que des calculs de similarité ou de parcours de graphe.

1.4 Processus de mise en ouvre

La mise en place d'un système de désambiguïsation lexicale se déroule en trois étapes

1.4.1 constitution de ressources génériques :

Plusieurs ressources non dédiées à la désambiguïsation lexicale sont envisageables : dictionnaires, encyclopédies, corpus non annotés, corpus annotés, bases lexicales, . . . Certaines sont construites automatiquement parfois en utilisant d'autres ressources. Cette étape est optionnelle et est souvent réalisée par des équipes spécialisées. Elle est celle qui demande le plus de supervision humaine.

1.4.2 Constitution d'une ressource dédiée à la DL :

Utilisation d'une ou plusieurs ressources génériques pour associer une représentation informatique à chacun des sens d'un mot. Ces sens sont soit directement

définis à partir de l'expertise humaine pour certaines ressources génériques comme les bases lexicales, soit induits à partir des contextes d'utilisation dans les textes (induction de sens). Structurellement, la ressource peut être, par exemple, un graphe, des chaines de caractères ou des représentations vectorielles.

1.4.3 Utilisation de la ressource dédiée pour désambiguïser des textes :

Il s'agit de l'algorithme de désambiguïsation proprement dit. Plusieurs paramètres peuvent être définis pour le traitement. Certains sont communs à tous les algorithmes comme la taille du contexte considéré pour un mot cible (par exemple quelques mots avant ou après la cible, la phrase qui contient la cible, voire le texte) tandis que d'autres dépendent du type d'algorithme mis en œuvre (par exemple la limite à considérer pour la profondeur de la recherche dans un graphe ou encore les paramètres à considérer pour des algorithmes stochastiques). Ainsi, selon ce point de vue, (Schwab et al., 2013) utilisent Word Net comme ressource générique, une représentation sous forme de sacs de mots issus des définitions des sens et de leurs liens comme ressource dédiée, un algorithme à colonies de fourmis et une mesure de proximité entre les sacs de mots comme algorithme de désambiguïsation lexicale. Roberto Navigli et son équipe (Navigli&Ponzetto, 2012) utilisent BabelNet comme ressource générique, une représentation sous forme de graphe issu des sens et de leurs liens comme ressource dédiée, des algorithmes de graphes (Pagerank, Degree, . . .) comme algorithmes de désambiguïsation. (Nasiruddin, Tchechmedjiev, Blanchon, Schwab,2015)

1.5 Ressources

Selon l'approche employée, diverses ressources sont nécessaires pour la DL. Les deux grandes catégories de ressources utilisées sont les corpus annotés en sens, indispensables pour les approches supervisées, et les bases de données lexicales, matière première des approches à base de connaissances. On trouve aussi des ressources plus génériques comme des corpus bruts (non annotés) ou des vecteurs de mot en complément dans certaines approches.

1.5.1 Bases de données lexicales

On parle généralement de base de données lexicale pour toute source de connaissances structurée qui apporte des informations sur les mots et les sens d'une ou plusieurs langues et qui est accessible par logiciel. On retrouve parmi ces bases de données des dictionnaires, des thésaurus, des ontologies, des graphes de connaissances, etc. Dans cette section, nous présentons deux principales bases de données lexicales qui sont utilisées en DL : WordNet et BabelNet.

1.5.1.1 WordNet

Le Princeton Word-Net (Miller et al., 1990), souvent appelé simplement Word-Net, est une base de données lexicale créée au laboratoire des sciences cognitives de l'université de Princeton, qui a la particularité de regrouper les différents sens des mots de la langue anglaise en ensembles de synonymes appelés *synsets* (contraction de *synonyme sets*). Ainsi, pour le nom *mouse* qui possède par exemple quatre sens, son deuxième sens (un *mouse* au sens d'un oeil au beurre noir) partagele même *synset*que le premier sens du nom *shiner*et le premier sens du groupe nominal *black eye*.WordNet repose sur des principes psycholinguistiques, qui ont pour conséquence une granularité très fine dans la distinction des sens. Par exemple, le premier et le deuxième sens du mot *snow*se distinguent par le fait que le premier (*precipitationfallingfromclouds in the form of icecrystals*) se réfère à la neige qui tombe du ciel, tandis que le second (*a layer of snowflakes (white crystalsoffrozen water) covering the ground*) se réfère à la neige qui couvre le sol. Les motspeuvent ainsi avoir un nombre de sens très grand, allant jusqu'à 59 sens différents pour le verbe *break*.

La richesse de cette base lexicale, en plus de sa granularité fine, vient aussi de son vaste réseau sémantique. En effet, les *synsets*sont reliés entre eux par des relations sémantiques telles que l'antonymie, l'hyperonymie, la méronymie, etc.

Le réseau sémantique de WordNet classe les mots en quatre parties du discours : les noms, les verbes, les adjectifs et les adverbes. Les principales relations sémantiques possibles entre les sens sont :

— la synonymie, qui regroupe les sens en *synsets*;

— l'antonymie, qui définit deux sens opposés (par exemple les adjectifs *petit* et*grand*);

— l'hyperonymie et l'hyponymie, qui définissent respectivement la généralisation et la spécialisation d'un sens (par exemple *voiture* est un hyponyme de *véhicule*, et à l'inverse *animal* est un hyperonyme de *chat*) ;

— la méronymie et l'homonymies, qui définissent la « partie de » (par exemple *main* est à la fois un méronyme de *bras* et un holonyme de *doigt*).

Word Net s'est rapidement imposé comme un standard *de facto* en DL de l'anglais, si bien que son inventaire de sens est utilisé pour l'annotation de la grande majorité des corpus anglais, tant pour l'apprentissage que pour l'évaluation des systèmes. Il est aussi au cœur de nombreuses approches à base de connaissances.

1.5.1.2 BabelNet

BabelNet (Navigli et Ponzetto, 2010) est une base de données lexicale multilingue,créée à l'université La Sapienza de Rome, qui repose sur le concept de *Babel synsets*. Un *Babel synset*représente un concept et regroupe des entrées lexicales provenant de multiples bases de données lexicales telles queWikipedia,Wiktionary,WordNet, etc.

Cette ressource est, à la différence deWordNet, générée de façon automatique,à l'aide d'un algorithme qui aligne les différents inventaires de sens en s'appuyant sur différentes informations propres à la base de données cible (titres et liens entre les pagesWikipedia, relations sémantiques entre les *synsets*deWordNet, etc.). Le résultat est un vaste réseau sémantique mis à jour continuellement. Il couvre, dans sa version 4.0, 284 langues et près de 16 millions de *Babel synsets*7.

On retrouve des utilisations notables de BabelNet dans certains travaux comme l'algorithme de Babelfy (Moro et al., 2014) qui exploite le graphe de BabelNet pour permettre de désambiguïser du texte dans n'importe quelle langue présente dans la base de données. Les campagnes d'évaluation SemEval 2013 (Navigli et al.,2013) et SemEval 2015

(Moro et Navigli, 2015) ont aussi consacré une tâche de désambiguïsation lexicale multilingue utilisant l'inventaire de sens de BabelNet.

1.5.2 Corpus

Selon Habert et al. (1998), un corpus est « une collection de données langagières qui sont sélectionnées et organisées selon des critères linguistiques explicites pour servir d'échantillon du langage ». Un corpus regroupe généralement plusieurs documents qui peuvent contenir plusieurs millions de mots, lesquels peuvent être annotés avec plusieurs informations (lemmes, parties du discours, sens, etc.).

1.5.2.1 corpus bruts

Parmi les corpus « bruts », c'est-à-dire sans annotation, on trouve par exemple le *Brown Corpus* (Francis et Kucera, 1964) (un million de mots), le *British NationalCorpus*(Burnard, 1998) (100 millions de mots) ou encore l'*American NationalCorpus*(Ide et Macleod, 2001) (20 millions de mots). Les textes proviennent de diverses sources comme des journaux, des livres, des encyclopédies ou du Web.

1.5.2.2Un corpus annoté en sens

Est un corpus dans lequel tous les mots ou seulement une partie d'entre eux sont annotés avec une étiquette de sens issue d'un inventaire de sens particulier. Par exemple, le corpus SemCor, DSO, OMSTI,ect…

1.6 Approches

Il existe différentes approches de désambiguïsation lexicale :

Les algorithmes supervisés utilisent des techniques d'apprentissage automatique. Ils apprennent un classificateur sur les corpus annotés en sens en utilisant des classificateurs(Hadj Salah ,2018),il existe plusieurs méthodes comme : arbres de décision, les classifications naïves bayésiennes, les réseaux de neurones,les approches de type plus proches voisins, les machines à vecteurs de support, etc.Les méthodes supervisées sont plus précises pour donner de meilleurs résultats (en anglais) que les méthodes Non supervisé, tant en termes de rapidité que de qualité, est son principal inconvénient Est-ce que cela nécessite une grande quantité de données commentées manuellement Pour inventorier une signification spécifique, une langue spécifique ou un champ spécifique Étant donné. (Schwab,2013)

1.6.2 Approches non supervisées

Dans l'approche non supervisée utilisent des corpus non annotés pour construire des vecteurs de mots ou des graphes de cooccurrences tandis que d'autres utilisent des sources de connaissance externes

(dictionnaires, thésaurus, bases lexicales...).Où la similitude peut être utilisée dans le sens en mesurant la distance (Schwab,2013).

1.6.3: Approches semi-supervisées

Une catégorie intermédiaire, constituée des approches semi-supervisées, utilise quelques données annotées comme, par exemple, un sens par défaut issu d'un corpus annoté lorsque l'algorithme principal échoue.

1.7 Évaluation

L'évaluation des différents systèmes de DL peut se faire de deux manières : d'un côté l'évaluation *in vivo* consiste à mesurer l'apport de la DL au sein d'une autre tâche, comme par exemple l'amélioration d'un système de traduction automatique ou de recherche d'information, de l'autre côté, l'évaluation *in vitro* consiste à mesurer directement les performances d'un système de DL en comparant ses prédictions avec celles d'un humain sur des corpus d'évaluation.

En pratique, l'évaluation *in vivo* a rarement été utilisée. On peut citer les travaux de Vickrey et al. (2005), qui introduisent la tâche de « traduction de mots », dans laquelle un algorithme semblable à un algorithme de DL doit trouver la bonne traduction d'un mot en fonction du contexte. Les différents « sens » d'un mot sont donc les différentes traductions possibles, et la méthode est évaluée sur un ensemblede corpus parallèles anglais-français. On peut aussi citer la tâche de subtitution lexicale (McCarthy, 2002) ou le *Word-in-Context*(Pilehvar et Camacho-Collados, 2019) qu'on décrira plus loin.

1.8 Conclusion

Dans ce chapitre nous avons définit la tâche de désambiguïsation lexicale en général en présentant le processus de mise en œuvre, les ressources utiles, ainsi que les approches pour la mise en œuvre. Nous avons identifié les deux manières pour évaluer un système de désambiguïsation lexicale.

Chapitre II :

Word embedding

2.1 Introduction

Word embeddings sont devenues des outils les plus utilisés et les principaux moteurs des réalisations étonnantes des tâches d'intelligence artificielle qui nécessitent le traitement de langages naturels tels que la parole ou les textes.

Avant on commence décrire ce modèle, on va voir quelque exemple ci-dessous :

- Quand on ouvre Google et recherche sur un article dans un domaine quelconque on va obtenir des certaines de résultats à ce sujet.

- On tape une phrase dans google traduction en anglais et on va obtenir une conversion en équivalente en arabe ou bien d'autre langue.

Nous notons dans deux exemples ci-dessus cela dépend de traitement de texte pour effectuer différentes tâches, qui besoin une quantité énorme de texte telles que le regroupement dans l'exemple de recherche Google, et la traduction automatique dans le deuxième. On sait que l'humain peut traiter et comprendre le format de texte mais ne peut pas effectuer les deux taches précédentes, Il n'est ni évolutif ni efficace.

Alors, comment pouvons-nous faire en sorte que les ordinateurs d'aujourd'hui effectuent le regroupement, la classification, etc. sur des données textuelles puisque nous savons qu'ils sont généralement inefficaces pour gérer et traiter des chaînes ou des textes pour des sorties fructueuses ?

Nous voulons représenter les mots de telle manière qu'ils saisissent leur sens comme le font les humains. Pas le sens exact du mot mais un sens contextuel. C'est à dire, quand on recherche dans google sur le mot « Apple » Comment faites-vous comprendre à un ordinateur que Apple est une entreprise et non un fruit ? comment pouvons-nous faire en sorte que les ordinateurs vous parlent du football ou de Ronaldo lorsque vous recherchez Messi. La réponse aux Questions ci-dessus réside dans la création d'une représentation des mots qui capturent leurs relations sémantiques et les différents types de contextes dans lesquels ils sont utilisés.

Et tout cela est implémenté en utilisant **Word Embeddings**ou des représentations numériques de textes afin que les Ordinateurs puissent les traiter.

2.2 Que sont Word embeddings de mots ?

Les algorithmes et modèles de Machine Learning utilisent des entrées numériques, nous avons besoin des chiffres comme entrées pour effectuer tout type de travail, que ce soit une classification, une régression...etc. puisque nous travaillons avec des espaces topologiques voire métriques (La Région-Ile-France, 2021).

En termes généraux et avec la quantité énorme de données présentés dans le format texte. Il est impératif d'extraire des connaissances et de créer des applications textuelles comme l'analyse de l'opinion, des critiques d'Amazon, la classification de documents ou l'actualités par Google, … etc.

Alors comment faire pour représenter du texte ?

Lorsqu'on traite des données brutes à grande dimensions comme des images ou du spectre audio, on utilise directement des vecteurs de coefficients associées aux intensités de pixel dans le premier cas ou les coefficients de densité spectrales dans le second cas.

Le problème avec un texte, c'est que l'on va traiter les mots et groupes de mots comme des entités symboliques non porteuses de sens, c'est à dire indépendantes les unes des autres et pour résoudre ce problème on va montrer une technique de représentation d'un mot ou un ensemble de mots en vecteurs de dimension inférieure, qui s'appelle Word embedding « plongement de mot ».

Alors, Word embeddings sont devenues l'un des outils les plus utilisés pour convertir les textes en nombres

L'objectif principal de Word embedding est de générer des représentations vectorielles denses des mots qui portent des significations sémantiques, Il est capable de capturer le contexte d'un mot dans un document, et traduisent les similarités ou relations entre les sens de ces mots.

Chaque de mot vecteur peut avoir plusieurs certaines de dimensions tel qu'un mot unique dans corpus se voit attribuer un vecteur dans l'espace.

Représentation vectorielle(mathématique) qui est, en même temps cohérente avec la sémantique.

Il sera aussi possible d'établir une cohérence entre les opérations arithmétiques (l'addition et la soustraction) sur les vecteurs et la sémantique des mots qu'ils représentent.

Définissons maintenant formellement les Word Embeddings, Word embedding tente généralement de mapper un mot à l'aide d'un dictionnaire sur un vecteur, Pour avoir une vue claire, prenons cet exemple **une phrase = "Word Embeddings are Word convertedintonumbers"** le mot dans cette phrase peut être '**Embeddings**' or '**numbers**'

Le vocabulaire se présente alors sous forme d'un dictionnaire, qui contient tous les mots d'une phrase,Ainsi on peut représenter un dictionnaire['Word','Embeddings','are','converted','into','numbers']

Une représentation vectorielle d'un mot peut être codé de format 'one hot' tel que 1 représente la position où le mot existe et 0 partout ailleurs. La représentation vectorielle du mot «numbers » dans ce format, selon le dictionnaire ci-dessus, est 'numbers' = [0,0,0,0,1] et «converted » est [0,0,0,1,0]. Avec cette représentation, tous les mots ont la même distance et la même similitude. L'encodage one hot n'apporte donc qu'une information selon laquelle un mot est différent d'un autre.

Cette idée est apparue pour la première fois dans l'hypothèse de distribution de Harris (1954), déclarant que : "les mots dans des contextes similaires ont des significations similaires ». Une erreur courante dans la compréhension des Word Embeddings est que les mots proches dans l'espace vectoriel sont synonymes. Alors qu'en réalité, ces mots sont juste syntaxiquement et / ou sémantiquement similaire Mikolov et al. (2013a). Sur cette base, la distance entre les mots est la clé de la reconnaissance des phrases et des similarités entre documents Levyet al. (2015).

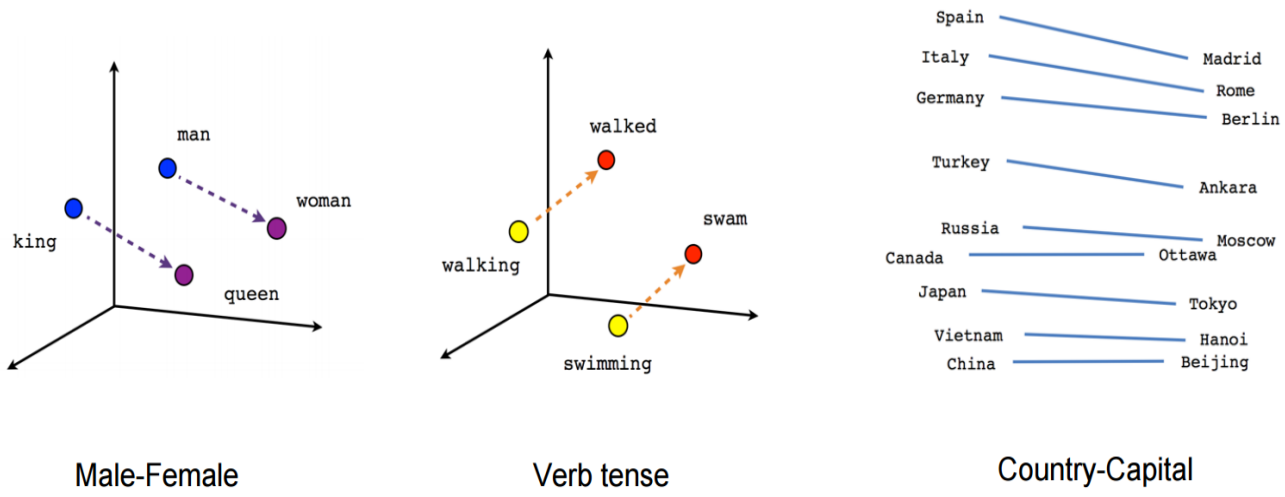


Figure 2.2: Relations entre les mots selon les plongements de mots Mikolov et al. (2013a)

- **Similarité sémantique sur des données textuelles (STS :** SemanticTextual-Similarity)

Évaluer la similarité entre entités textuelles est un des problèmes centraux dans plusieurs disciplines tel que l'analyse de données textuelles, Le Plagiat, la recherche documentaire ou l'extraction de connaissances à partir de données textuelles (TextMining). Dans chacun de ces domaines, les similarités sont en effet utilisées pour une large variété de traitements :

- en analyse de données textuelles (ADT), les similarités sémantiques sont utilisées pour la description et l'exploration de données, pour l'identification de structures cachées et pour la prédiction.

- en recherche documentaire (RD), l'évaluation des similarités entre documents et requêtes est utilisée pour identifier les documents pertinents par rapport à des besoins d'information exprimés par les utilisateurs.

- en TextMining (TM), les similarités sont utilisées pour produire des représentations synthétiques de vastes collections de documents, dans le cadre de procédures d'extraction d'information à partir de données textuelles.

Les techniques mises en œuvre pour calculer les similarités sont dans une même approche générale :

Les entités textuelles sont tout d'abord associées à des représentations spécifiques qui vont servir de base au calcul des similarités.

Comme nous voyons un schéma illustrant la similarité sémantique entre des entités textuelles :

Figure 2.3: Utilisation de STS dans une conversation.

2.3 Différents types de Word Embeddings

Les différents types de Word Embeddings peuvent être classés en deux catégories :

- Embeddings basée sur la fréquence.

- Embeddings basée sur la prédiction.

Essayons de comprendre chacune de ces méthodes en détail.

2.3.1 Embeddings basée sur la fréquence

Il existe généralement trois types de vecteurs que nous rencontrons dans cette catégorie.

- **Vecteur par comptage**
- **Vecteur TF-IDF**
- **Vecteur de co-occurrence**

On va expliquer chacune de ces types en détail avec des exemples.

2.3.1.1 Vecteur par comptage

Le modèle vectoriel apprend un vocabulaire à partir de tous les documents, puis modélise chaque document en comptant le nombre de fois où chaque mot apparaît. Par exemple, considérons que nous avons des documents **D** et que **T** est le nombre de mots différents dans notre vocabulaire, alors la taille de la matrice vectorielle de comptage sera donnée par D * T. Prenons les deux phrases suivantes :

Document 1: "The cat sat on the hat"

Document 2: "The dog ate the cat and the hat "

A partir de ces deux documents, notre vocabulaire est le suivant :

{the, cat, sat, on, hat, dog, ate, and}

Donc D = 2, T = 8

On va compter le nombre de fois où chaque mot apparaît dans chaque document. Dans le document 1, « le » apparaît deux fois, et « chat », « sat », « on » et « hat » apparaissent chacun une fois, de sorte que le vecteur caractéristique des documents est : {the, cat, sat, on, hat, dog, ate, and} la matrice vectorielle de comptage est comme suivant :

	the	cat	sat	on	hat	dog	ate	and
Document1	2	1	1	1	1	0	0	0
Document2	3	1	0	0	1	1	1	1

On peut dire maintenant, la colonne est comprise comme un vecteur de mot pour le correspondant dans la matrice de M.Par exemple, le vecteur de mot pour 'cat' dans la matrice ci-dessus est [1,1] et ainsi de suite.

On remarque ci-dessus, les lignes correspondent aux documents du corpus et les colonnes correspondent aux jetons du dictionnaire. Par exemple la deuxième ligne de la matrice contient « hat » une seule fois, « the » trois fois etc.

En fait, le problème c'est qu'il peut y avoir pas mal de variations lors de la préparation de la matrice M.

Généralement La façon dont le dictionnaire est préparé.

Pourquoi ? Parce que dans les applications du monde réel, nous pourrions avoir un corpus contenant des millions de documents. Et avec des millions de documents, nous pouvons extraire des centaines de millions de mots uniques.Donc, fondamentalement, la matrice qui sera préparée comme ci-dessus sera très maigre et inefficace pour tout calcul. Nous pouvons extraire certains mots principaux du vocabulaire en fonction de la fréquence et utiliser comme nouveau vocabulaire ou nous pouvons utiliser les deux méthodes.

2.3.1.2 Vectorisation TF-IDF :

Dans un corpus de texte volumineux, certains mots seront très présents (par exemple « the », « a », « is » en anglais) et donc très peu d'informations significatives sur le contenu réel du document. Si nous devions transmettre les données de comptage direct directement à un classificateur, ces termes très fréquents masqueraient les fréquences de termes plus rares mais plus intéressants.

Nous souhaiterions minimiser les mots courants présents dans presque tous les documents et accorder plus d'importance aux mots figurant dans un sous-ensemble de documents. Il est très courant d'utiliser la transformation tf – idf.

Cette méthode prend en compte non seulement l'occurrence d'un mot dans un seul document mais dans l'ensemble du corpus. Par exemple, un document A sur Lionel Messi va contenir davantage d'occurrences du mot « Messi » par rapport à d'autres documents. Mais des mots courants tels que « the », etc. vont également être

présents à une fréquence plus élevée dans presque tous les documents. Donc cette méthode résoudre ce problème.

- TF-IDF travaille en pénalisant ces mots courants en leur attribuant des poids plus faibles tout en donnant une importance aux mots comme Messi dans un document particulier. Alors, comment fonctionne exactement **TF-IDF** ?

Définissons maintenant quelques termes liés à TF-IDF.

Tf signifie **terme-fréquence** tandis que tf – idf signifie terme-fréquence multiplié par l'inverse **de la fréquence du document.**

TF = (Nombre de fois que le terme t apparaît dans un document) / (Nombre de termes dans le document)

IDF = log (N / n), où N est le nombre total de documents et n le nombre de documents dans lesquels un terme t est apparu.

TF-IDF (t, document) = TF (t, document) * IDF (t)

On va prendre un exemple qui donne le nombre de termes (mots / mots) dans deux documents

Document 1

Term	Count
This	1
is	1
about	2
Messi	4

Document 2

Term	Count
This	1
is	2
about	1
Tf-idf	1

Tel que : TF (This, Document1) = 1/8

TF (This, Document2) = 1/5

Cela dénote la contribution du mot dans le document, c'est-à-dire que les mots relatifs au document doivent être fréquents Exemple : un document sur Messi doit contenir le mot 'Messi' en grand nombre.

Calculons l'**IDF** pour le mot 'Messi'.

IDF (Messi) = log (2/1) = 0.301. IDF(This) =log (2/2) = 0.

Alors, comment expliquer le raisonnement derrière IDF ? Idéalement, si un mot est apparu dans tout le document, alors probablement ce mot n'est pas pertinent pour un document particulier. Mais s'il est apparu dans un sous-ensemble de documents, le mot a probablement une certaine pertinence pour les documents dans lesquels il est présent. Comparons maintenant TF-IDF pour un mot commun **"This"** et un mot **"Messi"** qui semble être pertinent pour le Document 1.

TF-IDF (This, Document1) = (1/8) * (0) = 0

TF-IDF (This, Document2) = (1/5) * (0) = 0

TF-IDF (Messi, Document1) = (4/8) * 0,301 = 0,15.

Nus remarquons que, la méthode TF-IDF pénalise fortement le mot 'This' mais attribue un poids plus important à 'Messi'. Cela peut donc être compris comme « Messi » est un mot important pour Document1 dans le contexte du corpus entier.

2.3.1.3 Matrice de co-occurrence avec une fenêtre contextuelle fixe

L'idée générale de cette méthode c'est que les mots similaires ont tendance à apparaître ensemble et auront un contexte similaire,par exemple Pomme est un fruit. La mangue est un fruit. La pomme et la mangue ont tendance à avoir un contexte similaire, à savoir le fruit Avant de plonger dans les détails de la construction d'une matrice de co-occurrence, il doit convient de clarifier deux concepts :

laco-occurrence et la fenêtre de contexte.

- Co-occurrence : Pour un corpus donné, la co-occurrence d'une paire de mots disons w1 et w2 est le nombre de fois où ils sont apparus ensemble dans une fenêtre de contexte.

- Fenêtre contextuelle : La fenêtre contextuelle est spécifiée par un nombre et la direction. Alors, que signifie une fenêtre de contexte de 2 (autour de) ? laisse nous voir un exemple ci-dessous,

Quick BrownFoxJumpOverTheLazyDog

Les mots verts sont une fenêtre contextuelle de 2 (autour de) pour le mot 'Fox' et pour le calcul de la co-occurrence, seuls Ces mots seront comptés. Voyons la fenêtre de contexte pour le mot 'Over'.

QuickBrownFoxJumpOverTheLazyDog

Pour bien illustrer on prend un autre exemple comme :

Considérons un corpus composé des documents suivants :

penny wise and pound foolish

a penny saved is a penny earned

En laissant compte (w (next) | w (current)) représenter le nombre de fois que le mot w (next) suit le mot w (current), nous pouvons résumer les statistiques de cooccurrence pour les mots « a » et « penny » comme suitvant :

	a	and	earned	foolish	is	penny	pound	saved	wise
a	0	0	0	0	0	2	0	0	0
penny	0	0	1	0	0	0	0	1	1

On remarque dans le tableau que le mot « a » est suivi deux fois le mot « penny » ainsi « penny » suivi les mots « earned », « saved », « wise » une seule fois dans notre corpus.

En réalité Le décompte indiqué ci-dessus est appelé fréquence bigramme il ne regarde que le mot suivant d'un mot courant. Étant donné un corpus de N mots, nous avons besoin d'une table de taille NxN pour représenter les fréquences bigrammes de toutes les paires de mots possibles. Une telle table est très clairsemée car la plupart des fréquences sont égales à zéro. En pratique, les nombres de cooccurrences sont convertis en probabilités. Il en résulte que les entrées de ligne pour chaque ligne s'ajoutent à une dans la matrice de co-occurrence, cette matrice de co-occurrence est décomposée en utilisant des techniques telles que PCA, SVD, etc. en facteurs et la combinaison de ces facteurs forme la représentation vectorielle de mot.

-Avantages de la matrice de co-occurrence : Il préserve la relation sémantiqueentre les mots. Autrement dit, l'homme et la femme ont tendance à être plus proches que l'homme et la pomme.

21

- Inconvénients de la matrice de co-occurrence : Il faut beaucoup de mémoire pour stocker la matrice de co-occurrence

2.3.2 Embeddings basée sur la prédiction

Il existe plusieurs approches de Word embedding. Les premiers remontent aux années 1960 et reposent sur les méthodes de réduction de dimensionnalité**.** Plus récemment, de nouvelles techniques basées sur des modèles probabilistes et des réseaux de neurones, comme Word2Vec permis d'obtenir de meilleures performances (Ria Klushther, 2019).

Word2Vec est un modèle de représentation distribué des mots peu profond (par rapport un réseau de neurone traditionnel) où le principe de générer des vecteurs dimensionnels à partir d'un apprentissage sur des données d'entrer non étiquetées. En fait, il prédit les mots en fonction de leur contexte en utilisant l'un de deux modèles neuronaux distincts : CBOW et Skip-Gram.

2.3.2.1 Continuous Bag-of-Words (CBOW)

Un **contexte** peut être un mot unique ou un groupe de mots. Mais pour plus de simplicité, On va prendre un mot de contexte unique et essayons de prédire un mot cible unique. Nous supposonsd'avoirun corpus **C = "Hey, this is sample corpus using only one context Word."** Et que nous avons défini une **fenêtre de contexte** égale à 1. Ce corpus peut être converti à un ensemble d'entraînement pour un modèle **CBOW** comme suit.

L'entrée est indiquée ci-dessous. La matrice qui existe à droite dans l'image ci-dessous contient le code **one-hot (0 et 1)** de l'entrée de gauche.

Input	Output			Hey	This	is	sample	corpus	using	only	one	context	word
Hey	this		Datapoint 1	1	0	0	0	0	0	0	0	0	0
this	hey		Datapoint 2	0	1	0	0	0	0	0	0	0	0
is	this		Datapoint 3	0	0	1	0	0	0	0	0	0	0
is	sample		Datapoint 4	0	0	1	0	0	0	0	0	0	0
sample	is		Datapoint 5	0	0	0	1	0	0	0	0	0	0
sample	corpus		Datapoint 6	0	0	0	1	0	0	0	0	0	0
corpus	sample		Datapoint 7	0	0	0	0	1	0	0	0	0	0
corpus	using		Datapoint 8	0	0	0	0	1	0	0	0	0	0
using	corpus		Datapoint 9	0	0	0	0	0	1	0	0	0	0
using	only		Datapoint 10	0	0	0	0	0	1	0	0	0	0
only	using		Datapoint 11	0	0	0	0	0	0	1	0	0	0
only	one		Datapoint 12	0	0	0	0	0	0	1	0	0	0
one	only		Datapoint 13	0	0	0	0	0	0	0	1	0	0
one	context		Datapoint 14	0	0	0	0	0	0	0	1	0	0
context	one		Datapoint 15	0	0	0	0	0	0	0	0	1	0
context	word		Datapoint 16	0	0	0	0	0	0	0	0	1	0
word	context		Datapoint 17	0	0	0	0	0	0	0	0	0	1

Figure 2.4: Présentation de code one-hot des mots.

La cible pour un **datapoint**unique, par exemple, le point de données 4 est indiquée ci-dessous

Hey this is sample corpus using Only One context Word

0 0 0 1 0 0 0 0 0 0

Cette matrice illustrée dans l'image ci-dessus est envoyée dans un réseau de neurones avec trois couches : Une couche d'entrée, une couche cachée et une couche de sortie. La couche en sortie est une couche **softmax** est utilisée pour additionner les probabilités obtenues dans la couche en Sortie La somme de ces probabilités décimales doit être égale à 1. Voyons d'abord une représentation schématique du modèle CBOW.

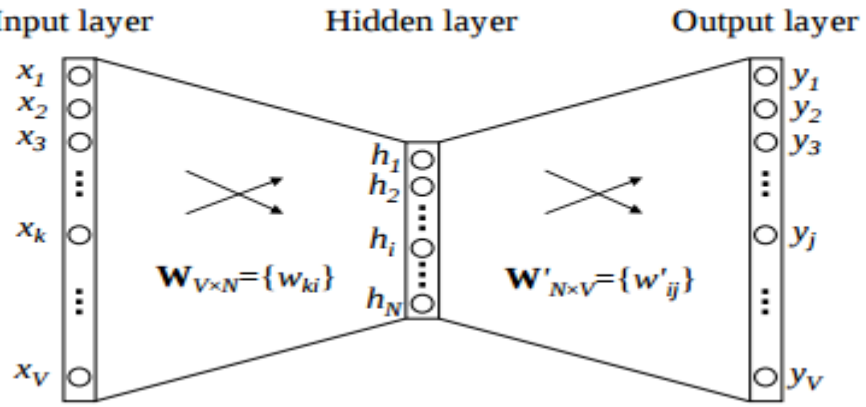


Figure 2.5: représentation schématique du modèle CBOW

Et voyons maintenant comment la propagation en avant fonctionnera pour calculer l'activation de la couche cachée. La représentation matricielle de l'image ci-dessus pour un seul point de données est présentée ci- dessous.

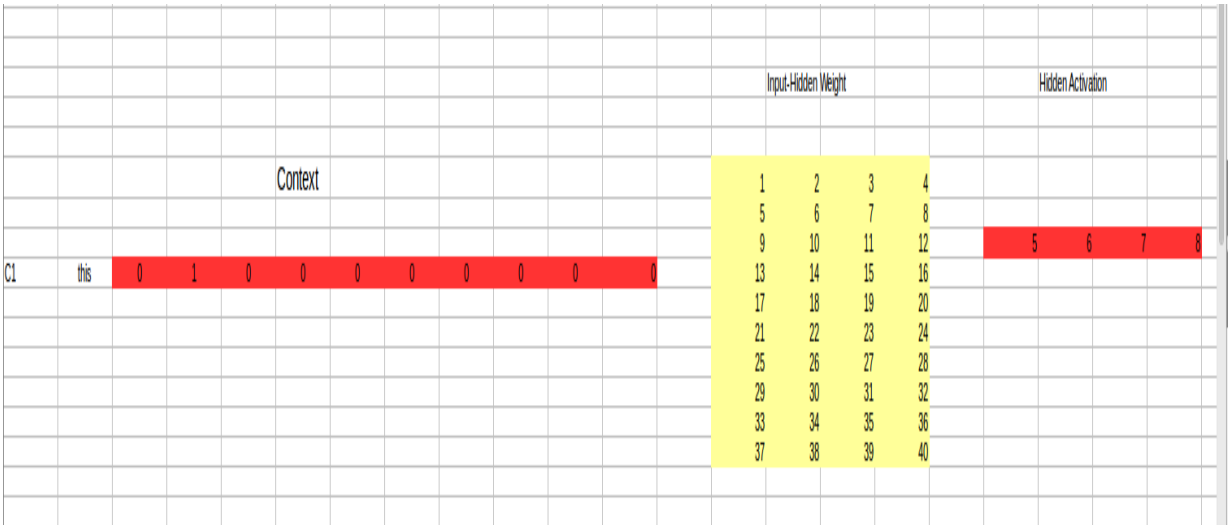


Figure 2.6: La représentation matricielle du model CBOW

1- La couche d'entrée et la cible les deux codées à one-hot la taille [1 * V].

Ici V = 10.

2- Il y a deux séries de poids. L'un se trouve entre la couche d'entrée et la couche cachée et l'autre entre la couche cachée et la couche de sortie.

Taille de la matrice de couche cachée en entrée = [V*N], taille de matrice cachée en sortie = [N*V] : tel que N est le nombre de dimensions dans lesquelles nous

choisissons de représenter notre mot. Il est arbitraire et constitue un hyper-paramètre pour un réseau neuronal.

De plus, **N est le nombre de neurones** dans la couche cachée.

3- Ici, N=4. Il n'y a pas de **fonction d'activation** entre les couches (plus précisément,Je fais référence à l'activation linéaire).

4- L'entrée est multipliée par les poids cachée entrés et appelée **activation cachée**. C'est simplement la ligne correspondante dans la matrice cachée en entrée copiée.

5- L'entrée cachée est multipliée par les poids de sortie cachés et calculée la sortie.

6- L'erreur entre la sortie et la cible est calculée et renvoyée pour réajuster les poids.

7- Le poids entre la couche cachée et la couche en sortie est considéré comme une représentation vectorielle du mot. Nous avons vu les étapes ci-dessus pour un seul mot de contexte. Maintenant, qu'en est-il si nous avons plusieurs mots de contexte? L'image ci-dessous décrit l'architecture de plusieurs mots de contexte.

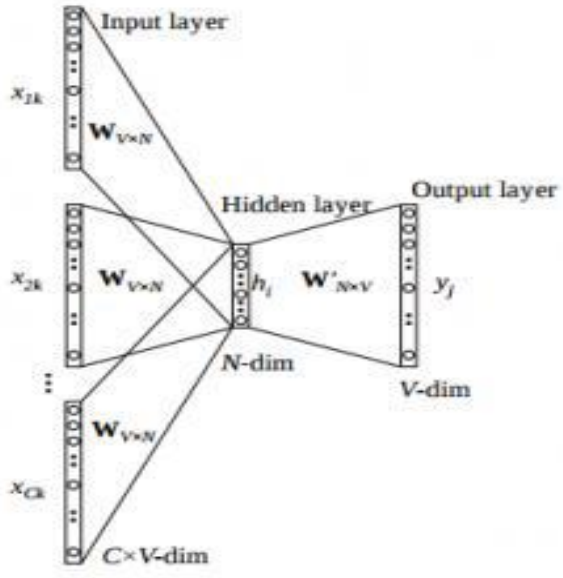


Figure 2.7: Une représentation d'architecture de plusieurs mots de contexte.

Vous trouverez ci-dessous une représentation matricielle de l'architecture ci-dessus pour faciliter la compréhension

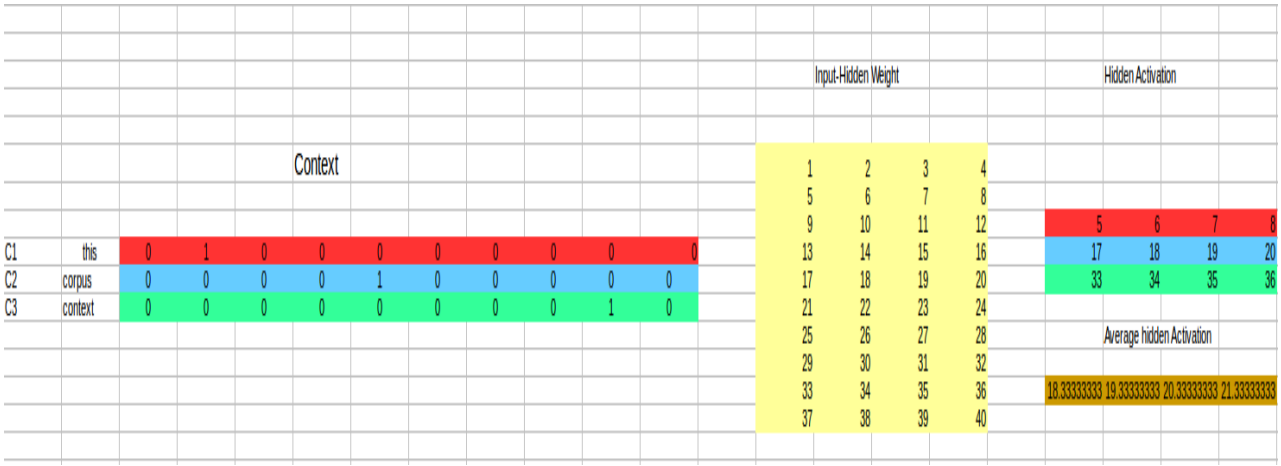


Figure 2.8: Une représentation matricielle de l'architecture

L'image ci-dessus prend 3 mots de contexte et prédit la probabilité d'un mot cible. L'entrée peut être considérée comme trois vecteurs codés à une séquence dans la couche d'entrée, comme indiqué ci-dessus en rouge, bleu et vert. Ainsi, la couche d'entrée aura 3 [1 * V] vecteurs dans l'entrée comme indiqué ci-dessus et 1 [1 * V] dans la couche de sortie. Le reste de l'architecture est identique à celui d'un CBOW à 1 contexte. Les étapes restent les mêmes, et change le calcul de l'activation cachée. Au lieu de simplement copier les lignes correspondantes de la matrice de poids cachée en entrée dans la couche cachée, et on va prendre la moyenne sur toutes les lignes correspondantes pour la matrice. Nous pouvons comprendre cela avec la figure ci-dessus. Le vecteur moyen calculé devient l'activation cachée. Donc, si nous avons trois mots de contexte pour un seul mot cible, nous aurons trois activations cachées initiales qui seront ensuite moyennées pour obtenir l'activation finale.

À la fois pour un mot de contexte unique et pour un mot de contexte multiple, nous avons montré les images jusqu'au calcul des activations cachée, car cette partie où CBOW diffirent d'un simple réseau MLP. Les étapes après le calcul de la couche cachée sont identiques à celles du MLP.

La différence entre MLP et CBOW sont mentionnées ci-dessous pour clarification :

1- La fonction objective dans MLP est une MSE (moyenne erreur quadratique) alors que dans CBOW, elle est une probabilité logarithmique négative d'un mot donné à un ensemble de contexte i.e. **log (p (wo / wi)), où p (wo / wi)** est donné comme

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

wo : mot de sortie

wi : mots de contexte.

Le gradient d'erreur en ce qui concerne les poids de sortie cachés et les poids de la couche en entrée est différent puisque MLP a des activations **sigmoïdes** (généralement) mais CBOW a des activations **linéaires**. Cependant, la méthode de calcul du gradient est identique à celle d'un MLP.

- Avantages de CBOW :

Étant probabiliste est la nature, il est supposé être supérieure aux méthodes déterministes (en général). Il manque de mémoire. Il n'a pas besoin d'énormes besoins en RAM, comme celui d'une matrice de co-occurrence, où il doit stocker trois énormes matrices.

Inconvénients de CBOW :

CBOW prend la moyenne du contexte d'un mot (comme indiqué ci-dessus dans le calcul de l'activation cachée). Par exemple, Apple peut être à la fois un fruit ou une entreprise, mais CBOW prend une moyenne des deux contextes et le place entre un cluster pour les fruits et les entreprises. Entrainer d'un CBOW à partir de zéro peut prendre une éternité s'il n'est pas optimisé correctement.

2.3.2.2 Skip-gram

L'architecture skip-gram est l'inverse de CBOW. Le but de **skip-gram** est de prédire le contexte donné à un mot. Prenons le même corpus que celui sur lequel nous avons construit notre modèle CBOW. **C = " Hey, this is sample corpus using only one context Word.**" Construisons les donnéesd'apprentissage.

Input	Output(Context1)	Output(Context2)
Hey	this	<padding>
this	Hey	is
is	this	sample
sample	is	corpus
corpus	sample	corpus
using	corpus	only
only	using	one
one	only	context
context	one	word
word	context	<padding>

Le vecteur d'entrée pour **skip-gram** sera similaire à un modèle **CBOW** à 1 contexte. De plus, les calculs jusqu'à l'activation de la couche cachée seront les mêmes. La différence sera dans la variable cible. Comme nous avons défini une fenêtre de contexte de 1 sur les deux côtés, il y aura «**two» une variable cible codée one hot** et **« tow » sorties correspondantes,** comme l'indique la section bleue de l'image. Deux erreurs distinctes sont calculées par rapport aux deux variables cibles et les deux vecteurs d'erreur obtenus sont ajoutés élément par élément pour obtenir un vecteur d'erreur final qui est propagé en retour pour mettre à jour les poids. Les poids entre la couche d'entrée et la couche cachée sont prises comme représentation du mot vecteur après L'entrainement. La fonction de perte ou l'objectif est du même type que le modèle **CBOW**.

L'architecture skip-gram est illustrée ci-dessous.

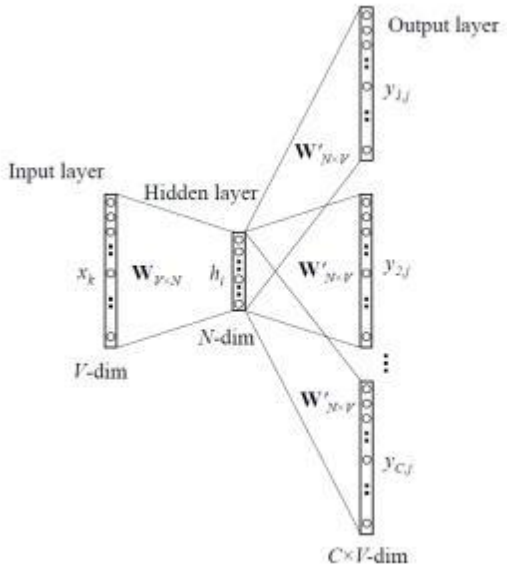


Figure 2.9: représentation schématique du modèle SKIP-GRAM

Pour une meilleure compréhension, la structure de style matricielle avec le calcul est présentée ci-dessous.

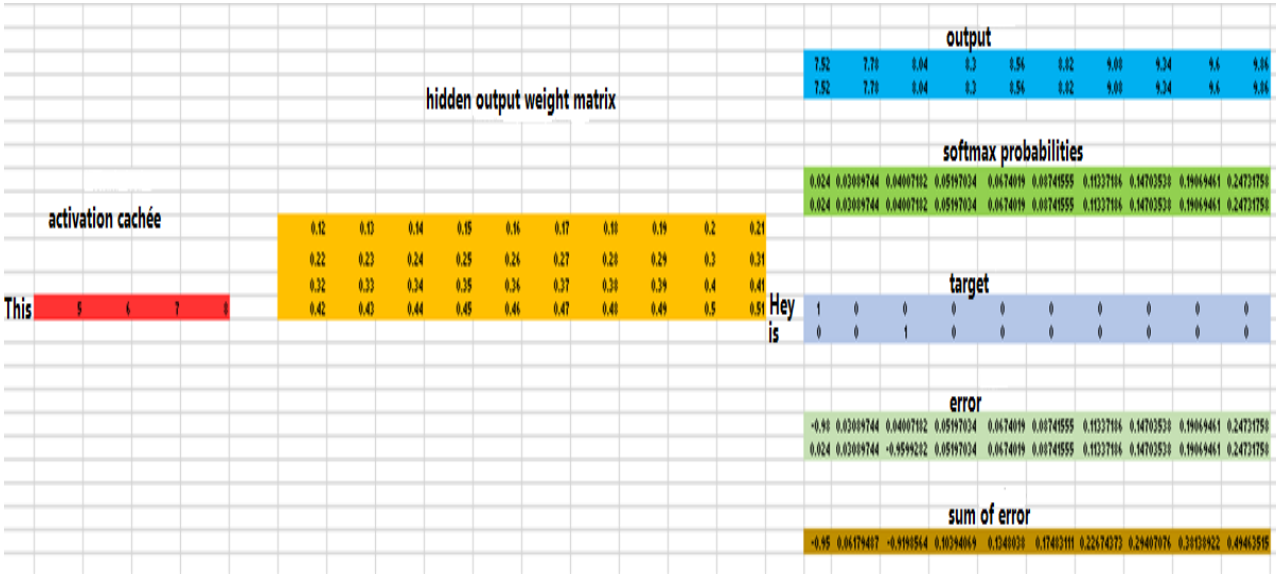


Figure 2.10: La représentation matricielle du model CBOW

Laissez-nous décomposer l'image ci-dessus. Taille de la couche d'entrée [1 * V], taille de la matrice de poids cachée [V * N], nombre de neurones dans la couche cachée N, taille de la matrice de poids de la sortie cachée [N * V]. Taille de la couche de sortie C [1 * V]. Dans l'exemple ci-dessus, C : est le nombre de mots de contexte = 2, V = 10, N = 4.

1. La ligne en rouge correspond à l'activation cachée correspondant au vecteur codé one hot unique en entrée. C'est fondamentalement la ligne correspondante de la matrice cachée en entrée.
2. La matrice jaune est le poids entre la couche cachée et la couche en sortie.
3. La matrice bleue est obtenue par la multiplication de la matrice **d'activation cachée** et des **poids de sortie cachés.Output = (hidden activation) * (hidden output weights matrix)**

Example **Output [0] = 5*0.12 + 6*0.22 + 7*0.32 + 8*0.42 = 7.52**

Output [1] = 5*0.13 + 6*0.23 + 7*0.33 + 8*0.43= 7.78

Il y aura deux lignes calculées pour deux mots cibles (contextuels).

4. Chaque ligne de la matrice bleue est convertie en ses Probabilités **softmax**individuellement, comme indiqué dans la zoneverte.$\sum_{k=1}^{10} softmax[\mathbf{k}] = 1$

5. La matrice grise contient les vecteurs codés one-hot des deux mots de contexte **(cible). Hey** ET **This**

6. L'erreur est calculée en soustrayant la première ligne de la matrice grise (cible) de la première ligne de la matrice verte (sortie) par élément.

Error[i] = softmax[i] – target[i]

Ceci est répété pour la ligne suivante. Par conséquent, pour **n** mots de contexte cible, nous aurons **n** vecteurs d'erreur.

7. La somme par élément est prise sur tous les vecteurs d'erreur pour obtenir un vecteur d'erreur final.

8.Ce vecteur d'erreur est renvoyé pour mettre à jour les poids.

- Avantages du modèle Skip-Gram

Le modèle de Skip-gram peut capturer deux sémantiques pour un seul mot. C'est-à-dire qu'il aura deux représentations vectorielles de l'Apple. Le premier pour l'entreprise et un autre pour le fruit.

Le skip-gram avec sous-échantillonnage négatif est supérieur à toutes les autres méthodes. Un excellent outil interactif pour visualiser CBOW et ignorer gramme en action. Puisque le Word Embeddingssont des représentations numériques de similarité contextuelles entre lesmots, ils peuvent être manipulés et utilisés pour effectuer des tâches incroyables.

2.4 Conclusion :

Les plongements de mots (Word embeddings) sont les représentations les plus utilisées actuellement dans les dernières méthodes de traitement du langage.

Nous avons vu dans ce chapitre certains modèles de Word embedding qui nous aide à déterminer le sens correct d'un mot à partir de son contexte dans une phrase ou un texte.

Différentes manières ont été introduites pour calculer les plongements de mot. La première est basée sur la fréquence qui contient trois types de vecteurs :

- Vecteur par comptage qui prend tous les vocabulaires qui apparait une seule fois dans les documents.
- Vecteur TF-IDF calcule la fréquence du chaque mot dans les documents
- Vecteur de co-occurrence qui construit une matrice pour les mots qui apparait dans le même contexte.

La deuxième basée sur la prédiction qui obtenus à partir des modèles probabilistes et des réseaux de neurones comme word2vec qui prédit les mots en fonction de leur contexte en utilisant l'un de deux modèles neuronaux distincts les deux modèles utilisent des mots voisins afin d'extraire la sémantique des mots dans les plongements.

- Dans Skip-Gram, nous essayons de prédire les mots de contexte en utilisant le mot cible.

- Dans CBOW (Continuous Bag of Words), le modèle est vraiment similaire, mais fait l'opération inverse. À partir des mots de contexte, nous voulons que notre modèle prédise le mot cible.

Chapitre III:

Conception et réalisation d’un système de désambiguïsation sémantique

3.1 Introduction :

Il est connu que les humains ont la capacité, dans la plupart des cas et en s'aidant du contexte, à désambiguïser sans trop d'efforts. Cependant, pour le traitement automatique des langues naturelles, nous avons déjà présenté dans le chapitre précédent des méthodes pour affecter aux mots les sens corrects vis à vis du contexte. Dans ce dernier chapitre nous présentons une approche non-supervisé n'utilisent pas de corpus annotés qui nous avons adopté dans notre étude. Certaines de ces méthodes utilisent des corpus non annotés pour construire des vecteurs de mots ou des graphes de cooccurrences tandis que d'autres utilisent des sources de connaissance externes (dictionnaires, thésaurus, bases lexicales...) et consistent à donner un score censé refléter la proximité des objets linguistiques (généralement des mots ou des sens de mots) comparés. Ce chapitre est organisé de la manière suivante. Dans la partie 3.2 décrit les ressources et les outils utilisés dans notre étude. La partie 3.3 décrit les étapes nécessaires pour la conception et la réalisation notre système :

- Prétraitement du texte et la construction de base de données.

- Création des vecteurs des mots ambiguës.

- Création des vecteurs contextuels. - La mesure de la similarité à partir la similarité cosinus.

-Nous présentons par la suite les données des expériences menées ainsi que les résultats de l'évaluation des différents algorithmes.

-Nous terminons par une conclusion.

3.2 3.2 Les outils et ressources utilisés:

3.2.1 Environnement :

Anaconda est une distribution libre et open source2 des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique) (Anaconda(distribution python), 2021). En particulier,

Anaconda fournit un environnement de travail adapté à l'enseignement et au calcul scientifique, **spyder**, que nous avons utilisé dans notre étude.

SPYDER :

1- Spyder est un environnement de développement pour Python

2- Libre (Licence MIT) et multiplateforme (Windows, Mac OS, GNU/Linux)

3- Créé et développé par Pierre Raybaut en 2008 (Spyder(logiciel), 2021)

4- Spyder est extensible avec des plugins

5- La version est utilisée dans cet étude : version 4

L'environnement **spyder**est disponible sur le **Navigateur anaconda** dont est une interface graphique (GUI) incluse dans la distribution Anaconda, et qui permet aux utilisateurs de lancer des applications. Quand on ouvre le navigateur anaconda, il apparait ci-dessousPuis on passe à l'application spyder clique sur le bouton Launch ouvrir l'interface spyder :

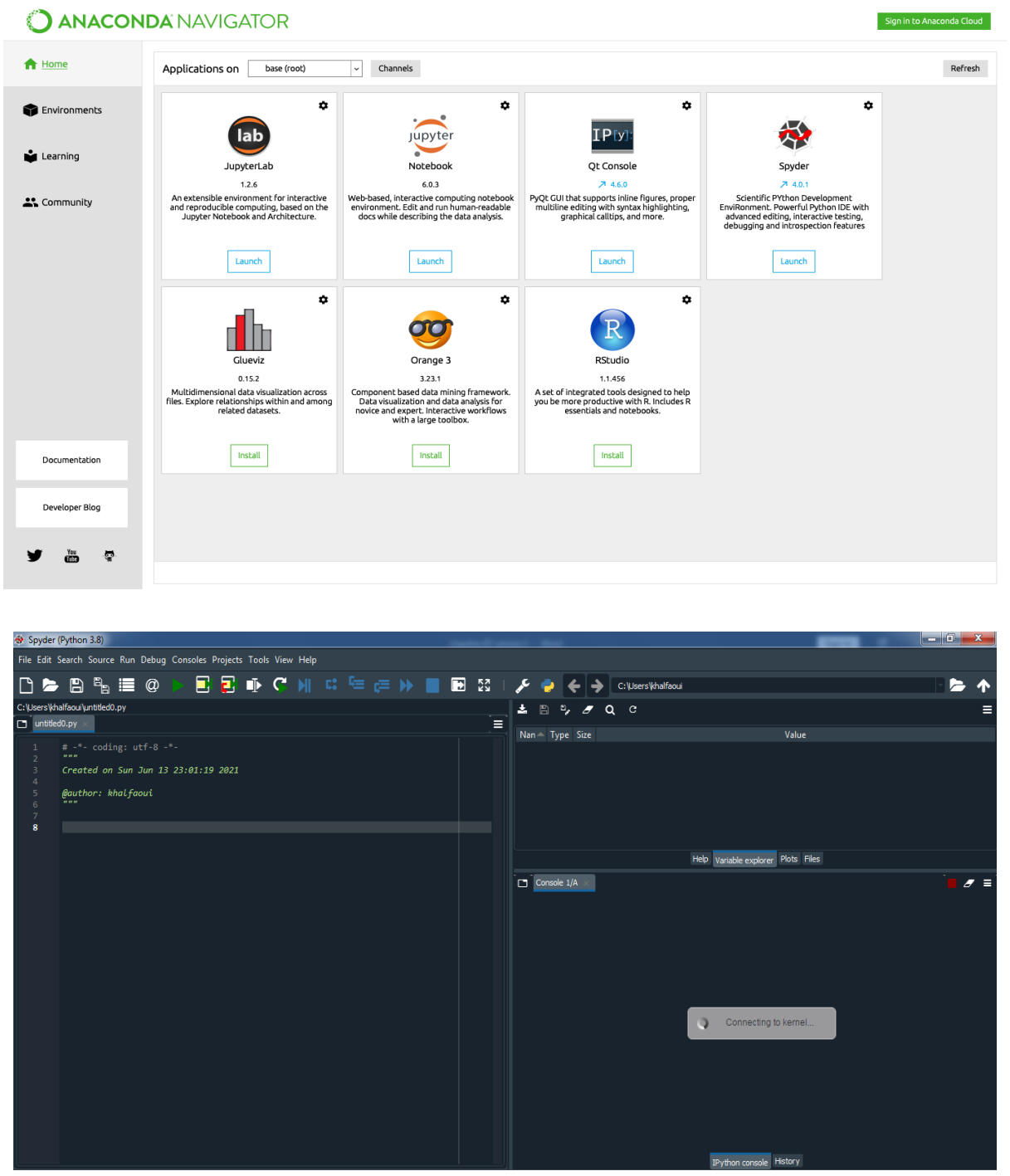


Figure 3.11: L'interface graphique de spyder (IDE)

D'autre part ouvrir Anaconda Prompt saisir l'instruction suivante : spyder et clique sur le bouton enté pour ouvrir l'interface qui apparait ci-dessus.

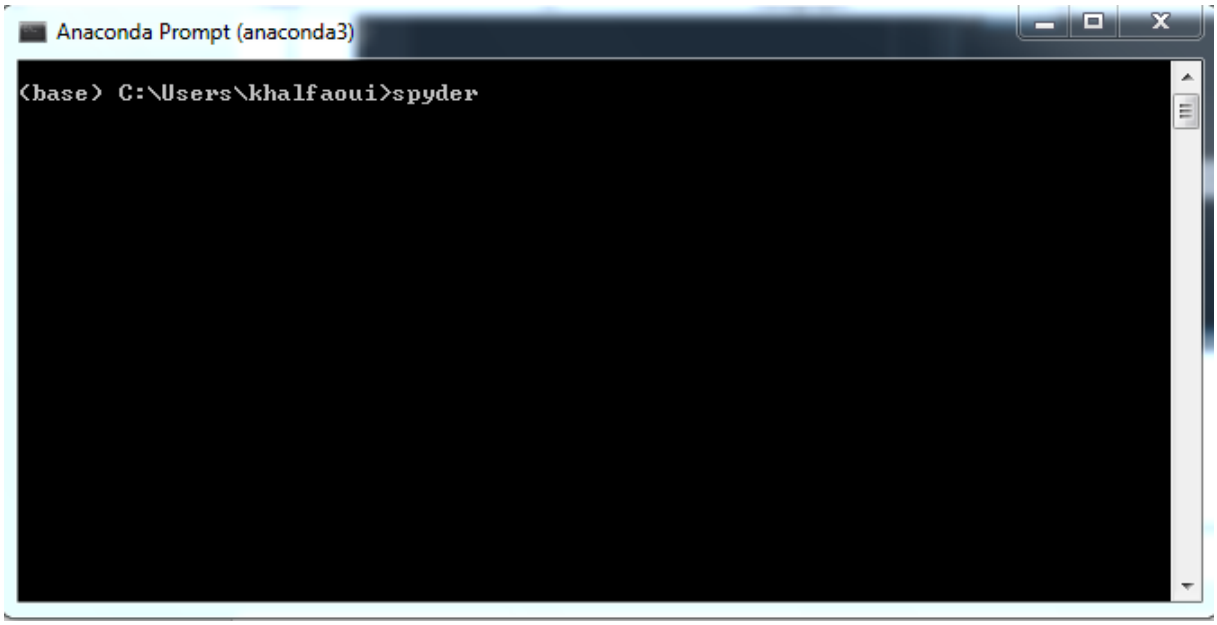


Figure 3.12: Anaconda Prompt(anaconda3).

Présentation de l'environnement SPYDER :

Spyder est constitué de 3 fenêtres :

1. L'éditeur qui permet de rédiger des programmes.

2. La console qui permet de tester des commandes (onglet : Console IPython) et qui renvoie les résultats des programmes rédigés dans l'éditeur (autres onglets)

3. L'explorateur avec pour onglets :

(a) L'inspecteur d'objets qui donne des informations sur l'utilisation des fonctions activées.

(b) L'explorateur de variables qui donne la liste et les valeurs de toutes les variables qui ont été crées.

(c) L'explorateur de fichiers qui donne accès au disque dur.

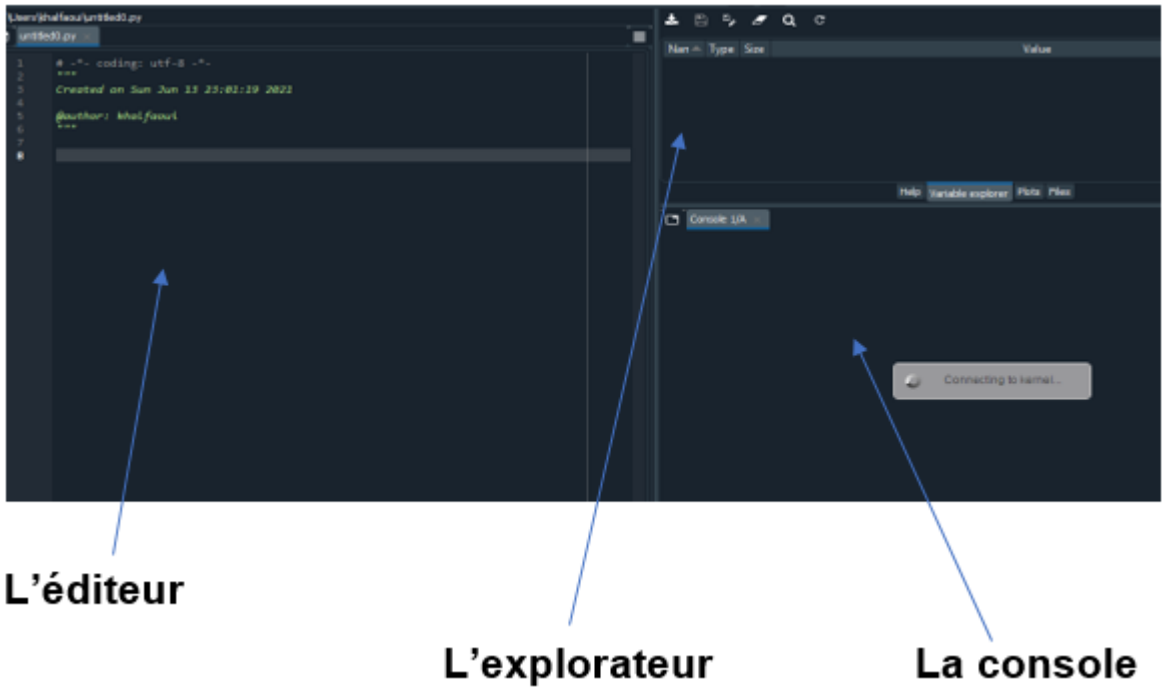


Figure 3.13: les fenêtres de spyder (IDE).

3.2.2 Langage :

Le langage qui a été adopté dans notre étude python est un langage portable, dynamique, extensible, gratuit, qui permet sans l'imposer une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles (Cordeau, s.d.). Python est un langage de programmation généraliste, interprété, facile à apprendre et rapide à mettre en œuvre (Jachym, s.d.). Il possède de nombreuses caractéristiquesont cité :

- PYTHON peut être utilisé dans tous les domaines : écriture d'applications pour le Web (serveur d'application Zope, framework Django), programmes de calculs mathématiques (biblithèqueSciPy), programmation de scripts systèmes etc.

- PYTHON dispose d'une très large bibliothèque standard qui offre au programmeur des outils très diverspour : la gestion réseau (librairie socket), la manipulation du format xml etc.

- PYTHON est un langage open source, licence open source CNRI, de version 3.7

-PYTHON syntaxe claire et cohérente,

- PYTHON typage dynamique fort, pas de déclaration.

3.2.3 Les outils et les ressources :

Dans notre étude nous utilisons des bibliothèques suivantes :

3.2.3 Bibliothèques

3.2.3.1 NLTK :

Natural LanguageToolkit (NLTK) est une boîte-à-outil permettant la création de programmes pour l'analyse de texte. Cet ensemble a été créé à l'origine par Steven Bird et Edward Loper, en relation avec des cours de linguistique informatique à l'Université de Pennsylvanie en 2001 (Ali, 2017).

Commençons par installer la librairie NLTK pour démarrer nos prochaines études en désambiguïsation lexical de mots.Son installation est assez simple

On utilise Anaconda Prompt et saisit l'instruction suivante :pipinstallnltk

La première chose à faire pour utiliser NLTK est de télécharger ce qui se nomme le NLTK corpora. On va télécharger tout le Corpus.

Dans votre éditeur de spyder, écrivez ceci :

```
1  import nltk
2  nltk.download()
```

Figure 3.14: L'installation de NLTK

Dans ce cas précis, une interface graphique s'affiche, vous permettant de définir la destination des fichiers et de sélectionner, cliquez sur le bouton **Download** dans le coin inférieur gauche de la fenêtre, et patientez jusqu'à ce que tout soit téléchargé dans votre dossier de destination.

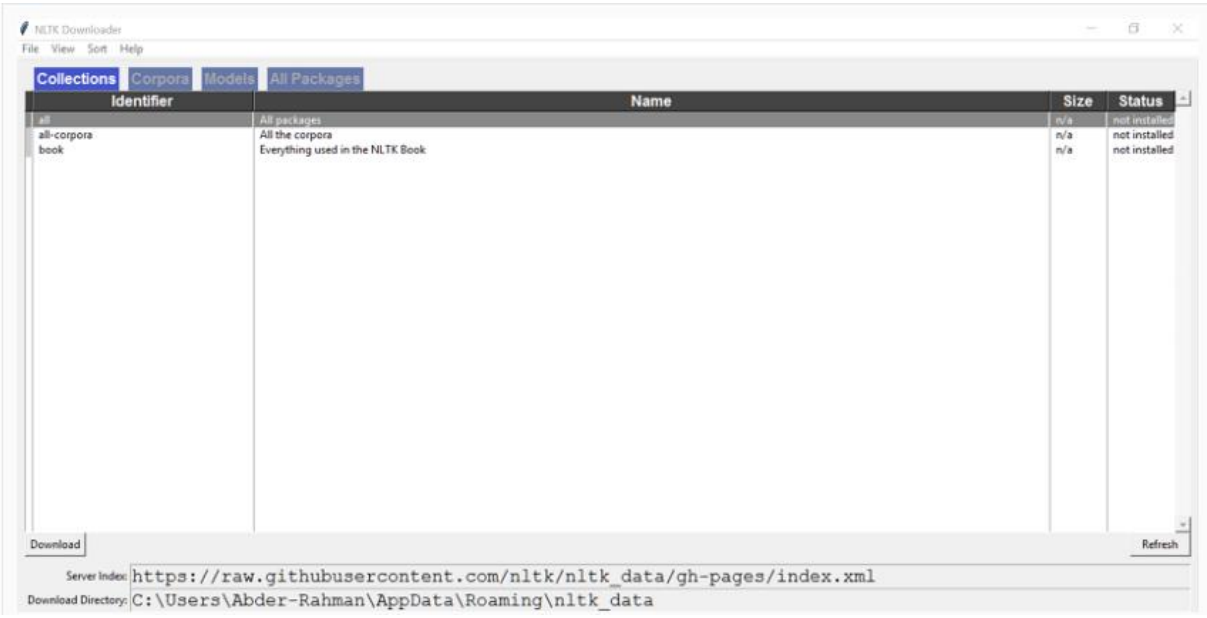


Figure 3.15: L'installation de collections

3.2.3.2 Wordnet :

- Une base de données lexicographiques qui couvre la grande majorité des noms, verbes, adjectifs et adverbes de la langue anglaise [7]. Développée par le Cognitive Science Laboratory (Princeton University) sous la direction de George A. Miller et Christiane Fellbaum

-Ressource libre et bien documentée

- il est devenu une des ressources les plus utiles pour de nombreuses applications de compréhension et d'interprétation automatique des langues, telles que la désambiguïsation sémantique, ... etc.

- Il définit le sens des mots par deux moyens : Les synsets (synonym set) : ensembles de synonymes. Les relations entre synsets : hyperonymie- hyponymie (is-a), méronymie, implication, dérivation morphologique ....

- **synsets** : sorte de « classe d'équivalence » sémantique, représentant un sens (un concept) particulier.Chaquesynset est accompagné : d'une description du sens qu'il représente : "City" et d'exemples d'usage.

Exemple : le mot "City" a 3 sens avec ses définitions :

1.city, metropolis, urban center -- (a large and densely populated urban area; may include several independant administrative districts; etc.)

2. city -- (an incorporated administrative district established by state charter)

3. city, metropolis -- (people living in a large densely populated municipality)

3.2.3.3 Gensim :

Est une bibliothèque de traitement du langage naturel (NLP) open source populaire utilisée pour modélisation de sujets non supervisée. Il utilise des modèles académiques de pointe et un apprentissage automatique statistique moderne pour effectuer tel que Création de documents ou de vecteurs de mots etc. (HebergementWebs, 2020).

- Gensim a été utilisé et cité dans plus de mille applications commerciales et académiques par exemple :fastText, Word2vec, …etc.

Concepts de base de Gensim :

- **Document** - ZIt fait référence à du texte.
- **Corpus** - Il fait référence à une collection de documents.
- **Vecteur** - La représentation mathématique d'un document est appelée vecteur
- **Modèle** - Il fait référence à un algorithme utilisé pour transformer des vecteurs à partir d'une représentation à un autre.

Il y a plusieurs méthodes pour télécharger Gensim parmi eux est d'utiliser l'environnement ***conda.*** Exécutez la commande suivante dans votre terminal Anaconda Prompt

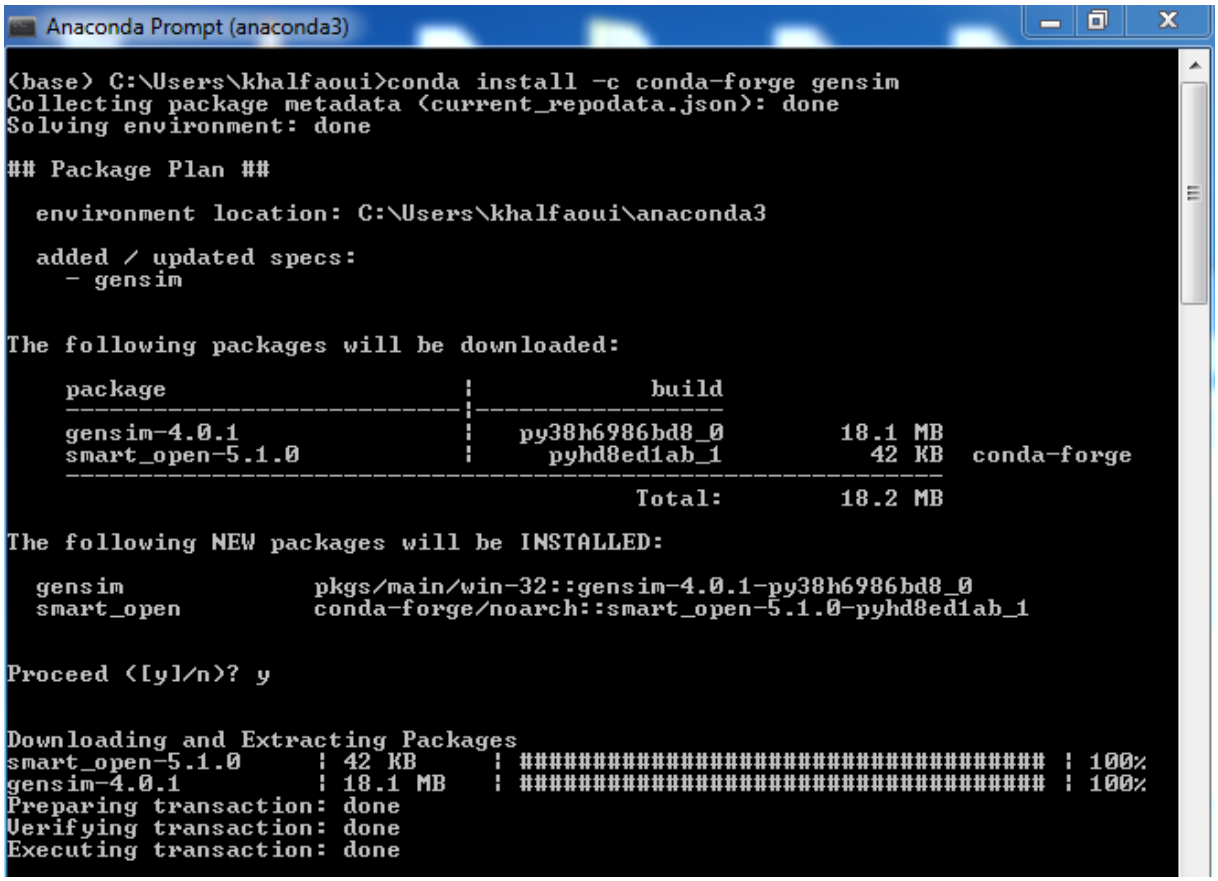


Figure 3.16: L'installation de Gensim

3.2.4 Matériel :

Après nous avons vu tous les outils et les ressources qui nous a aidé dans notre étude qui a été exécuté sur un ordinateur de type Toshiba d'un Processeur AMD E1-2100 APU with Radeon™ HD Graphic, vitesse 1.00 GHz. Espace mémoire de 2 GB, Disque dur de 5,9 GB.

3.3 Les étapes du système

A partir l'architecture suivante montre les étapes nécessaires pour implémenter l'approche.

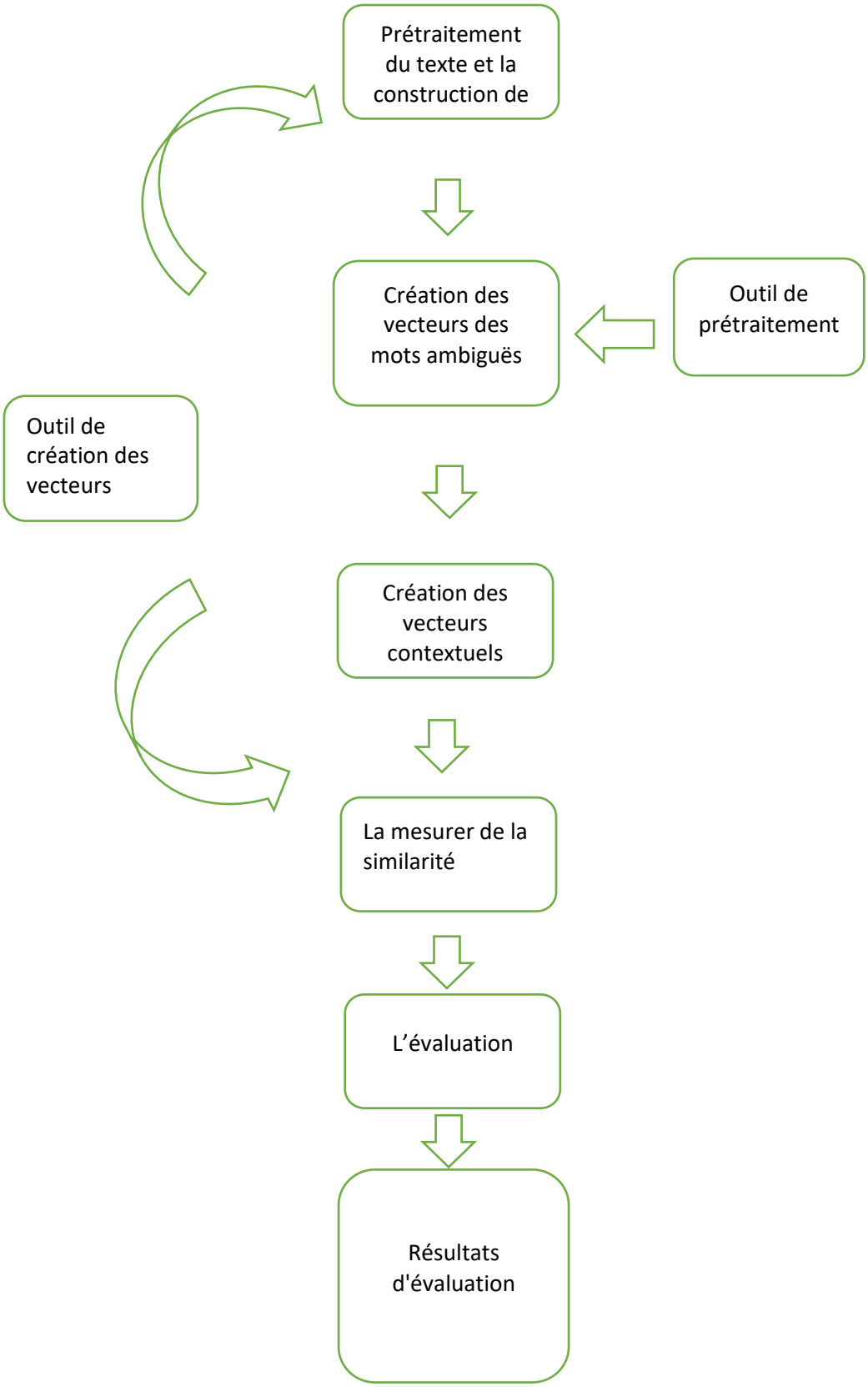


Figure 3.17: l'architecture d'implémentation de l'approche

3.3.1 Le prétraitement du texte et la construction de base de données :

3.3.1.1 Prétraitement et Extraction des définitions :

Nous travaillons sur quatre exemples différents, comme suit :

a: The mouse ate some toxin

b: I saw bats

c: willows lined the bank

d: I work in a bank

On remarque dans ces exemples Il existe des mots ont plusieurs sens quand recherche les sur google comme le mot mouse a plusieurs sens, d'autre part il est destiné est un animal ou un périphérique matériel de l'ordinateur, on trouve les mots : bats, bank ont plusieurs de sens dans le dictionnaire.

3.3.1.1TOKENISE :

La tokenisation est la tâche de la découper en morceaux, appelés jetons, peut-être en même temps de jeter certains caractères, comme la ponctuation.

Dans cette étape, nous allons convertir chaque exemple précédent en liste de jetons tout en supprimant la ponctuation. La Figure 3.2.2 montre le code de tokenise.

```
import nltk
from nltk.tokenize import word_tokenize
a_tok=word_tokenize(a_s)
b_tok=word_tokenize(b_s)
c_tok=word_tokenize(c_s)
d_tok=word_tokenize(d_s)
print("\n ----------------------tokenze-text------------------")
print("\n a_tok:",a_tok)
print("\n b_tok : ",b_tok)
print("\nc_tok : ",c_tok)
print("\n d_tok : ",d_tok)
```

```
----------------------tokenze-text------------------

a_tok: ['mouse', 'ate', 'toxin']

b_tok :  ['saw', 'bats']

c_tok :  ['willows', 'lined', 'bank']

d_tok :  ['work', 'bank']
```

Figure 318: Prétraitement (Tokenise).

3.3.1.2SUPPRIMER LES MOTS D'ARRÊT (stop Word) :

Les mots vides sont des mots courants qui n'apportent que peu ou pas de valeur à la signification du texte par exemple : « the », « i », « in » et « somme »

Différents ensembles de mots vides peuvent être nécessaires en fonction du domaine auquel le texte est lié. Dans cette étape, nous tirerons parti du corpus de mots vides de nltk. Vous pouvez définir votre propre ensemble de mots vides ou enrichir les mots vides standard en ajoutant des termes communs appropriés au domaine du texte. La Figure 3.2.3 montre le code suivant :

```
from gensim.parsing.preprocessing import remove_stopwords
```

```
print("\n ----------------------remove_stopwords-----------------------")
li_vec=[]
a_s=remove_stopwords(a)
b_s=remove_stopwords(b)
c_s=remove_stopwords(c)
d_s=remove_stopwords(d)
print("\n a_s : ",a_s)
print("\n b_s : ",b_s)
print("\n c_s : ",c_s)
print("\n d_s : ",d_s)
```

```
----------------------remove_stopwords-----------------------

a_s :  mouse ate toxin

b_s :  saw bats

c_s :  willows lined bank

d_s :  work bank
```

Figure 3.19: prétraitement (stop words).

3.3.1.3 Analyses des vocabularies:

Pour extraire les relations sémantiques à partir des exemples. On utilise les différentes ressources de connaissance pour extraire les lemmes et les classes grammaticales des mots le résultat de ces outils est présenté dans le tableau suivant :

Mot	Stemm	Catégorie grammaticale
mouse	mous	NOUN
ate	at	VERB
toxin	toxin	NOUN
saw	saw	VERB
bats	bat	NOUN
willows	willow	VERB
lined	line	NOUN
bank	bank	NOUN
work	work	VERB

Tableau 3.1: Exemple de résultat de l'analyseur morphosyntaxique

La base de données est présente sur forme suivante :

Id	Mot	Stemm	POS
1	mouse	mous	NOUN
2	Ate	at	VERB
3	Toxin	toxin	NOUN
4	Saw	saw	VERB
5	Bats	bat	NOUN

6	Willows	Willow	NOUN
7	Lined	Line	VERB
8	Bank	Bank	NOUN
9	work	work	VERB

Tableau 3.2: Exemple de tableau de base de données

3.3.2 Création des vecteurs des mots ambigus :

3.3.2.1 Extraction des définitions :

Maintenant on passe pour extrait les définitions de chaque mot ambigu ont été constituées à partir de WordNet (voir la partie 3.1.3.2 Wordnet), ensuite la préparation ces définitions (prétraitement de chaque définitions).

Le mot ambigu	Les définitions et prétraitement
« mousse »	la définition 1 de mouse : [['numerous', 'small', 'rodents', 'typically', 'resembling', 'diminutive', 'rats', 'having', 'pointed', 'snouts', 'small', 'ears', 'elongated', 'bodies', 'slender', 'usually', 'hairless', 'tails']] la définition 2 de mouse : [['hand-operated', 'electronic', 'device', 'controls', 'coordinates', 'cursor', 'screen', 'pad;', 'device', 'ball', 'rolls', 'surface', 'pad'], [';']]
« bank »	déffinition 1 de bank : ['sloping', 'land', '(especially', 'slope', 'body', 'water)', ';', '(', ')', 'especially', 'water'] déffinition 2 de bank :['financial', 'institution', 'accepts', 'deposits', 'channels', 'money', 'lending', 'activities', ';', '(', ')']

« bats »	la définition 1 de bats : [['nocturnal', 'mouselike', 'mammal', 'forelimbs', 'modified', 'form', 'membranous', 'wings', 'anatomical', 'adaptations', 'echolocation', 'navigate']] la définition 2 de bats : [['club', 'hitting', 'ball', 'games'], [';']]

Tableau 3.3: L'extraction des définitions de mot ambigu.

3.3.2.2 Vecteurs de mots ambigües :

Dans ce cas, on va créer chaque définition de mot ambigu en vecteur, exploitent des modèles de vecteurs de mot différents modèle Word2Vec pré-entraîné et proposé par (Mikolov et al. (2013)). L'objectif de cette partie est pour la facilité de comparaison sémantique de mots. Nos vecteurs de sens sont calculés comme la somme normée de tous les vecteurs des termes présents dans la définition du sens donné, tous basés sur le vocabulaire et les définitions de WordNet 3.0 (voir la partie 3.1.3.2). Ce modèle a été entraîné sur environ 100 milliards de mots issus du corpus de nouvelles de Google. La taille du vocabulaire est d'environ de 3 millions de mots et les vecteurs ont une dimension de 300.

Exemple de vecteur de définition de mot « mouse »:

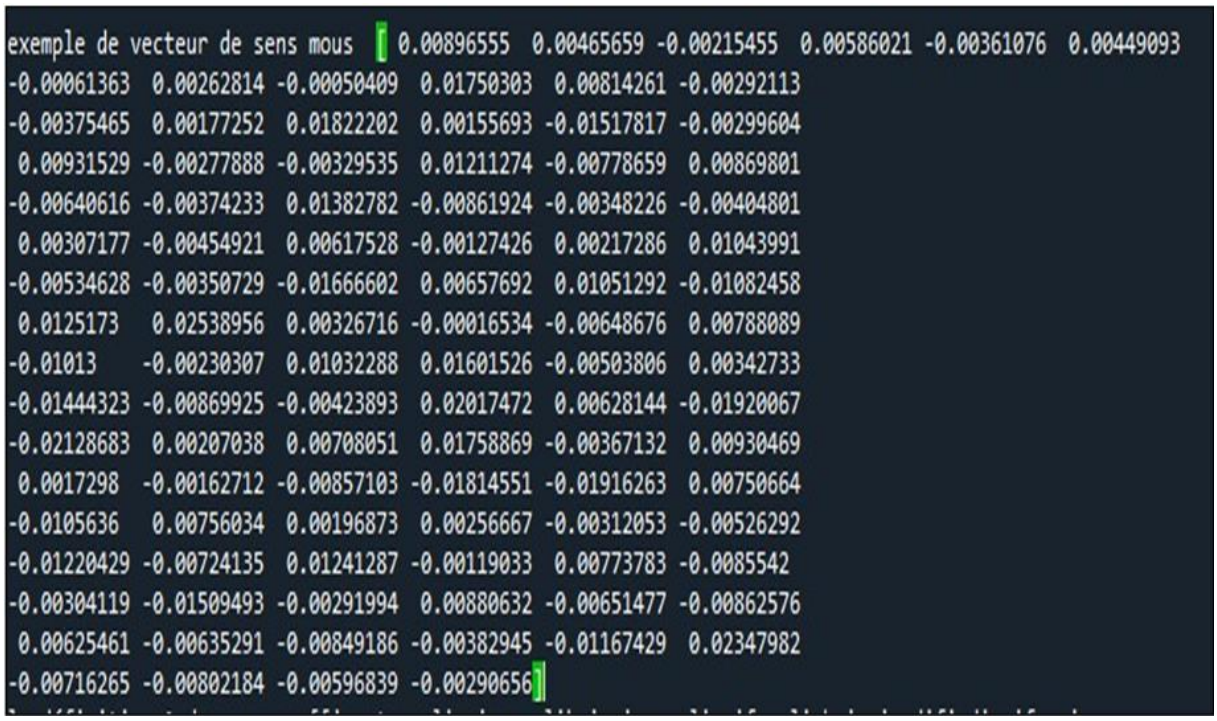


Figure 3.20: Exemple d'un vecteur

Le mot ambigu	Définition1	Définition2
« mousse »	Vecteur de mousse : Word2Vec(vocab=17, size=100, alpha=0.025)	Vecteur de mousse : Word2Vec(vocab=13, size=100, alpha=0.025)
« bats »	Vecteur de bats : Word2Vec(vocab=12, size=100, alpha=0.025)	Vecteur de bats : Word2Vec(vocab=5, size=100, alpha=0.025
« bank »	Vecteur de bank : Word2Vec(vocab=11, size=100, alpha=0.025)	Vecteur de bank : Word2Vec(vocab=11, size=100, alpha=0.025)

Tableau 3.4: Création des vecteurs des mots ambiguës

3.3.3 Création des vecteurs contextuels :

Désambiguïser tous les mots pleins d'un corpus dont le contexte représente un paragraphe est une tâche qui demande beaucoup de temps si on se base sur un algorithme exhaustif simple. La clé de notre approche de désambiguïsation est

l'observation des voisins de chaque mot polysémique dans le texte : au lieu de comparer chaque sens d'un mot à désambiguïser avec tous les sens de tous les mots qui se trouvent dans le texte, nous faisons une comparaison uniquement avec les sens des voisins plus proche pour mot ambigu.

3.3.3.1 Les voisins de mot ambigu et prétraitement :

Déterminer les voisins de chaque mot ambigu et extrait pour chaque voisin sa définitions dans les exemples précédents :

```
liste= ['mouse', 'ate', 'toxin']

liste= ['saw', 'bats']

liste= ['willows', 'lined', 'bank']

liste= ['work', 'bank']

----------------------début de tratment----------------------------------

les deffinition  6

les deffinition  [['eat', 'meal', ';', 'meal'], ['poisonous', 'substance', 'produced',
'metabolism', 'growth', 'certain', 'microorganisms', 'higher', 'plant', 'animal',
'species'], ['perceive', '(', 'an', 'idea', 'situation', ')', 'mentally'], ['textile',
'machine', 'having', 'revolving', 'spikes', 'opening', 'cleaning', 'raw', 'textile',
'fibers'], ['line', 'with', ';', 'form', 'line'], ['applying', 'mind', 'learning',
'understanding', 'subject', '(', 'especially', 'reading', ')']]
```

Figure 3.21: Extraire les voisins avec ses définitions

3.3.3.2 Vecteur de chaque voisin :

Le même cas précédent : le code montre comment créer de chaque définition d'un mot un vecteur :

```
    from gensim.models import Word2Vec
    import warnings
    warnings.filterwarnings('ignore')
    mymodel = Word2Vec(t, min_count=1 )
    # summarizing the loaded model
    print("mymodel ",mymodel)
# summarize vocabulary
    words = list(mymodel.wv.vocab)
# summarize vocabulary
    print("words ",words)
```

Figure 3.22: Exemple de création des vecteurs.

La définition de voisin	Vecteur
D1 = : [['eat'], ['meal'], [';'], ['meal']]	V1: Word2Vec(vocab=3, size=100, alpha=0.025)
D2 = : [['poisonous'], ['substance'], ['produced'], ['metabolism'], ['growth'], ['certain'], ['microorganisms'], ['higher'], ['plant'], ['animal'], ['species']]	V2 :Word2Vec(vocab=11, size=100, alpha=0.025)
D3 = : [['perceive'], ['('], ['an'], ['idea'], ['situation'], [')'], ['mentally']]	V3 : Word2Vec(vocab=7, size=100, alpha=0.025)
D4 = : [['textile'], ['machine'], ['having'], ['revolving'], ['spikes'], ['opening'], ['cleaning'], ['raw'], ['textile'], ['fibers']]	V4: Word2Vec(vocab=9, size=100, alpha=0.025)
D5= : [['line'], ['with'], [';'], ['form'], ['line']]	V5 : Word2Vec(vocab=4, size=100, alpha=0.025)
D6 = : [['applying'], ['mind'], ['learning'], ['understanding'], ['subject'], ['('], ['especially'], ['reading'], [')']]	V6: Word2Vec(vocab=9, size=100, alpha=0.025)

Tableau 3.5: Création des vecteurs contextuels

Tableau 3.5:..

3.3.3.3 La partie du discours (POS) et Inverse Document Frequency (ou IDF) :

Nos vecteurs de sens sont calculés comme la somme normée de tous les vecteurs des termes présents dans la définition du sens donné. Ces vecteurs sont pondérés en fonction de leur partie du discours (Part Of Speech, ou POS) : nom, verbe, adjectif ou ad-

verbe, et également pondérés par leur Inverse Document Frequency (ou IDF) : l'inverse de leur nombre d'occurrence dans le dictionnaire entier (FERRERO, 2017). Plus formellement nous avons :

— D(S) = {w0, w1, w2, . . . ,wn} la définition du sens S dans le dictionnaire

— pos(wn) = {n, v, a, r} la partie du discours du terme wn (nom, verbe, adjectif, ou adverbe)

— weight(pos) le poids associé à une partie du discours

— idf(wn) la valeur IDF de wn La définition du vecteur du sens S, notée φ(S) correspond à :

φ(S) = Xn i=0 (φ(wn) × weight(pos(wn)) × idf(wn))

φ(S) est ensuite normé afin d'avoir la même taille que les vecteurs de mots (qui sont eux aussi normés). Les poids attribués aux POS sont les mêmes que ceux utilisés par Ferrero et al. (2017) [9].

Le poids d'un mot :

Un mot est peut-être un nom, verbe ou adjectif afin que je puisse distinguer entre les trois types nous avons accordé un poids à chaque type, on connait que le verbe est le pivot de la phrase : les fonctions fondamentales (primaires) dépendent de lui.

Il est le centre du prédicat (= ce que l'on dit du sujet). Il exprime un procès : quelque chose qui se déroule dans le temps (bbouillon.free, s.d.) et partir les études antérieures disent que le verbe possède en effet un pouvoir combinatoire spécifique encodé dans la composante grammaticale du lexique (Dubost, 1977, 1983).

Multiplier le résultat d'un mot comme suite :

- Si le mot est un nom résultat = résultat * 1
- Si le mot est un verbe résultat = résultat * 0.6
- Si le mot est un adjectif résultat = résultat * 0.8

Calculer la valeur IDF de chaque mot :

Alors, comment expliquons-nous le raisonnement derrière IDF ? Idéalement, si un mot est apparu dans Tout les documents, alors ce mot n'est probablement pas pertinent pour un document particulier. Mais s'il est apparu dans un sous-ensemble de documents, alors le mot est probablement **pertinent** pour les documents dans lesquels il est présent. Calculons l'IDF pour chaque mot. La figure 3.2.7 montre le code de calcul la valeur IDF :

```
96      def idf(word, liste_document):
97          import math
98          return math.log(len(liste_document) / (1. + n_containing(word, liste_docume
99      li_idf=[]
100     idf1=idf('ate', liste_document)
101     li_idf.append(idf1)
102     idf1=idf('toxin', liste_document)
103     li_idf.append(idf1)
104     idf1=idf('saw', liste_document)
105     li_idf.append(idf1)
106     idf1=idf('willows', liste_document)
107     li_idf.append(idf1)
108     idf1=idf('lined ', liste_document)
109     li_idf.append(idf1)
110     idf1=idf('work', liste_document)
```

Figure 3.23: calcule la valeur IDF.

3.3.4 La mesure de la similarité : La similarité cosinus est l'une des mesures permettant de mesurer la similarité textuelle entre deux documents, quelle que soit leur taille dans le traitement du langage naturel. Un mot est représenté sous forme vectorielle. Les documents textuels sont représentés dans un espace vectoriel à n dimensions. Mathématiquement, la métrique de similarité cosinus mesure le cosinus de l'angle entre deux vecteurs à n dimensions projetés dans un espace multidimensionnel. La similarité cosinus de deux documents va de 0 à 1. Si le score de similarité cosinus est de 1, cela signifie que les deux Vecteurs ont la même orientation. La valeur plus proche de 0 indique que les deux documents ont moins de similarité comme La Figure 3.2.8.

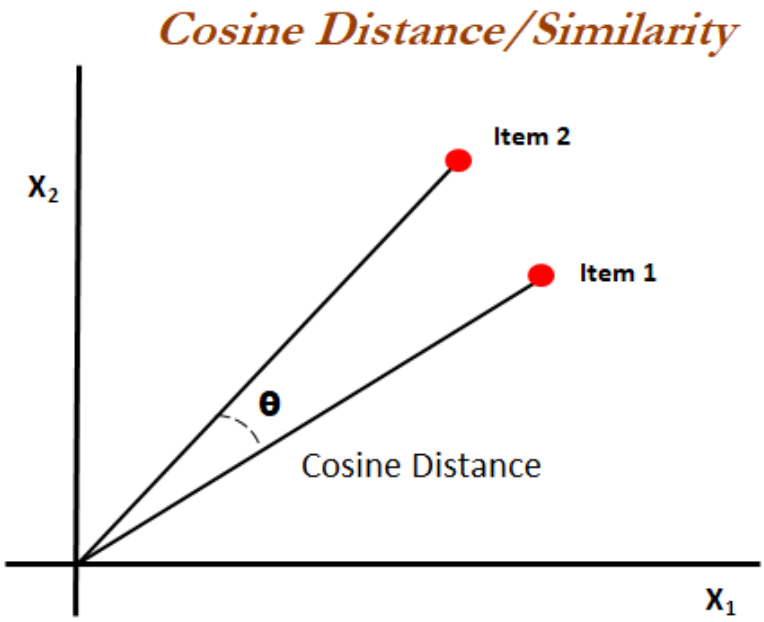


Figure 3.24: La similarité cosinus

Dans cette section, nous calculons la distance cosinus entre chaque vecteur de voisins avec les vecteurs de chaque mot ambigu, le code ci-dessus montre le calcul de la distance cosinus

```
273     #-------------------------------------------------cos mouse-------------------
274     import numpy as np
275     import scipy.spatial.distance as distance
276     c = 1 - distance.cosine(np.array(li_vec[0]),  np.array(vcsm1))
277     print("\n ate df 1",c)
278     c = 1 - distance.cosine(np.array(li_vec[0]), np.array(vcsm2))
279     print("\n ate df 2",c)
280     c = 1 - distance.cosine(np.array(li_vec[1]),  np.array(vcsm1))
281     print("\n toxin df 1",c)
282     c = 1 - distance.cosine(np.array(li_vec[1]),  np.array(vcsm2))
283     print("\n toxin df 2",c)
284     print("\n\n la deffinition lplus proch a deffinition mouse et 1")
```

Figure 3.25: Calcule de distance cosinus

Le mot ambigu 'mousse' dans l'exemple 1 :

Les vecteurs	La distance cosinus
-Vecteur de 'ate ' avec vecteur df 1 de 'mousse'	=-0.028101954609155655
- Vecteur de 'ate ' avec vecteur df 2 de 'mousse'	=0.08219128847122192
- Vecteur de 'toxin ' avec vecteur df 1 de 'mousse'	
- Vecteur de 'toxin ' avec vecteur df 2 de 'mousse'	=0.07244835048913956
	= 0.10583444684743881

Tableau 3.6: la distance similarité de mot ambigu « mousse ». Alors, on remarque que la définition la plus proche de mot « mouse » est la définition 1.

Alors, on remarque que la définition la plus proche de mot « mouse » est la définition1.

Le mot ambigu 'bats' dans l'exemple 2 :

Les vecteurs	La distance cosinus
-Vecteur de 'saw' avec vecteur df 1 de 'bats'	= -0.0345577709376812
- Vecteur de 'saw ' avec vecteur df 2 de 'bats'	=0.07015740871429443

Tableau 3 7: la distance similarité de mot ambigu « bats ».

Alors, on remarque que la définition la plus proche de mot « bats » est la définition 1.

Le mot ambigu 'bank' dans l'exemple 3 :

Les vecteurs	La distance cosinus
-Vecteur de 'willows' avec vecteur df 1 de 'bank'	= 0.013755553402006626
- Vecteur de 'willows' avec vecteur df 2 de 'bank'	= 0.12866650521755219
-Vecteur de 'lined' avec vecteur df 1 de 'bank'	
- Vecteur de 'lined' avec vecteur df 2 de 'bank'	=-0.04639745503664017
	= -0.10190597921609879

Tableau 3.8: la distance similarité de mot ambigu « bank ».

Alors, on remarque que la définition la plus proche de mot « bank » est la définition 1.

Le mot ambigu 'bank' dans l'exemple 4 :

Les vecteurs	La distance cosinus
-Vecteur de 'work ' avec vecteur df 1 de 'bank'	= 0.3190750181674957
- Vecteur de 'work' avec vecteur df 2 de 'bank'	= 0.18553799390792847

Tableau 3.9: la distance similarité de mot ambigu « bank »

Alors, on remarque que la définition la plus proche de mot « bank » est la définition 2.

La figure suivante montre les résultats des tableaux précédents :

```
ate df 1 -0.028101954609155655
ate df 2 0.08219128847122192
toxin df 1 0.07244835048913956
toxin df 2 0.10583444684743881

la deffinition lplus proch a deffinition mouse et 1
-----------phrase 2 ------------------------------------------------
saw df 1 -0.0345577709376812
saw  df 2 0.07015740871429443

la deffinition lplus proch a deffinition bats et 1
-----------phrase 3 ------------------------------------------------
willows df 1 0.013755553402006626
willows  df 2 0.12866650521755219
lined df 1 -0.04639745503664017
lined   df 2 -0.10190597921609879

la deffinition lplus proch a deffinition bank et 1
-----------phrase 4 ------------------------------------------------
work df 1 0.3190750181674957
work df 2 0.18553799390792847
```

Figure 3.26 : Résultat de calcule la distance cosinus

3.3.4 Evaluation : Métriques Les mesures utilisées

Pour l'évaluation quantitative des résultats de notre méthode de WSD sont les taux de précision et de rappel :

$$Précesion = \frac{\text{nombre de prédictions correctes}}{\text{nombre de prédictions faites par le système}}$$

Ce critère permet de mesurer la pertinence des réponses retournées par la méthode évaluée.

$$Rappel = \frac{\text{nombre de prédictions correctes}}{\text{nombre de nouvelles instances}}$$

Ce critère permet de mesurer l'exhaustivité des bonnes réponses dans l'ensemble des réponses générées. Le taux de rappel indique la proportion des nouvelles instances d'un mot ambigu qui sont correctement désambiguïsées, tandis que la précision indique, quant à elle, la proportion des prédictions de désambiguïsation faites par le système qui sont correctes. Nos résultats sont ensuite comparés aux résultats issus d'une méthode de base (Baseline). Les résultats de cette méthode correspondent aux deux scores, précision et rappel, comme nous allons l'expliquer dans le paragraphe

suivant. Pour faciliter la comparaison entre nos résultats et la Baseline, nous avons également eu recours à la f-mesure (Van Rijsbergen, 1979), score combinant précision et rappel en une mesure unique. Il s'agit, plus précisément, de la moyenne harmonique du rappel et de la précision, calculée selon la formule suivante :

$$f - mesure = \frac{2 * (précision * rappel)}{précision + rappel}$$

Résultat montre dans le tableau suivant :

Critère	Résultat
Précision	1
Rappel	0,5
F_mesure	0,66

Tableau 3.10: Résultat de l'évaluation.

Le résultat de notre méthode est 66 %.

3.4 Conclusion:

Ce chapitre décrit les étapes nécessaires de notre méthode sur la désambiguïsation lexicale de mot. La méthode que nous avons testée s'appuie sur une sélection des voisins les plus proches selon le contexte du mot à désambiguïser par comparaison les sens de mot ambigu avec les sens de voisins pour déterminer le sens exact de mot dans un contexte quelconque.

Conclusion générale

Conclusion Générale :

Il était bénéfique pour nous d'avoir vécu l'expérience de la mise en place d'un système de désambiguïsation lexicale. Nous avons décrit le modèle proposé pour la représentation du sens des mots dans un espace vectoriel basé sur les modèles de Word Embeddings. Ce modèle est inspiré des modèles vectoriels utilisés dans le domaine de la recherche documentaire, plus précisément le modèle des vecteurs conceptuels. Notre méthode consiste à sommer les vecteurs de mots présents dans la définition, pondérés en fonction de leur partie du discours ainsi que de leur fréquence, nous avons calculé les distances entre les vecteurs qui représente les mots, et ce pour mesurer la similarité des vecteurs d'un mot ambigu avec ses voisins pour déterminer le sens exact du mot dans un contexte quelconque, et enfin nous avons utilisé ce modèle pour la désambiguïsation lexicale du mot dans un texte ou document. Au stade de l'aboutissement de notre recherche, nous estimons que notre apport qui explore la problématique de l'ambigüité de mots bien que modeste comparé aux travaux de nos prédécesseurs. Nous avons trouvé que les voisins sont aussi des mots ambigus, ont des plusieurs sens dans le WorNet par apport le mot cible dans notre étude. Nous souhaitons que notre travail ouvre d'autres pistes de recherche pour approfondir les travaux relatifs à ce problème.

Bibliographie

Bibliographie:

[1] - Jérémy Ferrero : Similarités textuelles sémantiques translingues : vers la détection automa-tique du plagiat par traduction

[2] Busse Winfried et Dubost Jean-Pierre. 1983 (1977). *FranzösischesVerblexicon. Die Konstruk-tion der VerbenimFranzösischen.* Stuttgart : Klett.

[3] - La Région Île-de-France. OpenClassrooms. Disponible sur : https://openclassrooms.com/fr/courses/4470541-analysez-vos-donnees-textuelles/4855006-effectuez-des-plongements-de-mots-word-embeddings > (Consulté le 30/04/2021)

[4] - Manjeet Singh. Medium. Disponible sur : <https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285> (Consulté le 14 /10/2017)

[5] – NSS. AnalyticsVidhya. Disponible sur :< https://www.analyticsvid-hya.com/blog/2017/06/word-embeddings-count-word2veec/> (Consulté le 04/06/2017)

[6] – RiaKulshrestha. Towards Data Science.Disponible sur : https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314 > (Consulté le 29/11/2019)

[7] Jachym, Marc, "Cours de Programmation avec le langage Python : Niveau débu-tant en programmation », Licence professionnelle – Métrologie dimensionnelle et qua-lité IUT de St Denis, Université Paris 13.

[8] Cordeau, Bob, Introduction à Python 3, version 2.71828.

[9] wikipédia.Anaconda (distribution Python).Disponible sur :< https://fr.wikipedia.org/wiki/Anaconda_(distribution_Python)>(9 /04/ 2021)

[10] wikipédia. Spyder(logiciel). Disponible sur :< https://fr.wikipedia.org/wiki/Spyder_(logiciel)> (20 /05/2021)

[11] Abder-Rahman Ali. Tust. Introduction au Natural LanguageToolkit (NLTK) : Disponible sur <https://code.tutsplus.com/fr/tutorials/introducing-the-natural-language-toolkit-nltk--cms-28620> (03/05/2017)

[12] FERRERO J., BESACIER L., SCHWAB D. & AGNÈS F. (2017). CompiLIG at SemEval-2017 Task 1: Cross-Language Plagiarism Detection Methods for Semantic Textual Similarity. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval 2017), Vancouver, Canada.

[13] Hebergementwebs. Gensim_guide_rapie. Disponible sur :<https://www.hebergementwebs.com/tutoriel-gensim/gensim-guide-rapide> (17/11/2020)

[14] Loïc Vial. Modèles neuronaux joints de désambiguïsation lexicale et de traduction automatique. Intelligence artificielle [cs.AI]. Université Grenoble Alpes [2020-..], 2020. Français. P31

[15] Roberto Navigli.Wordsensedisambiguation : A survey. ACMComputing Surveys,

41(2) :10 :1–10 :69, feb. 2009. ISSN 0360-0300. doi : 10.1145/1459352.1459

355. URL http://doi.acm.org/10.1145/1459352.1459355.

[16] Warren Weaver. Translation. Machine translation of languages, 14 :15–23, 1955.

[17] Marwa Hadj Salah, Loïc Vial, Hervé Blanchon, Mounir Zrigui, Benjamin Lecouteux, et Didier Schwab. Traduction automatique de corpus en anglais annotés en sens pour la désambiguïsation lexicale d'une langue moins bien dotée, l'exemple de l'arabe. In 25e conférence sur le Traitement Automatique des Langues Naturelles, Rennes, France, May 2018. URL https://hal.archivesouvertes.

fr/hal-01781185.

[18] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, et Katherine Miller. Wordnet : An on-line lexical database. International Journal of Lexicography, 3 :235–244, 1990.

[19] George A. Miller, Claudia Leacock, RandeeTengi, et Ross T. Bunker. A semantic

concordance. In Proceedings of the workshop on Human Language Technology,

HLT '93, pages 303–308, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.ISBN 1-55860-324-7.doi : 10.3115/1075671.1075742.

URL http://dx.doi.org/10.3115/1075671.1075742.

[20] Adam Kilgarriff. Senseval : An exercise in evaluating word sense disambiguation

programs. In First international conference on language resources and evaluation (LREC), 1998.

[21] Didier Schwab — Jérôme Goulian —AndonTchechmedjiev, Désambiguïsation lexicale de textes :efficacité qualitative et temporelle d'un algorithme à colonies de fourmis, Univ. Grenoble Alpes, Laboratoire d'Informatique de Grenoble équipe GETALP 41 rue des mathématiques, BP 53 38041 Grenoble Cedex 9,p2.

[22] R. Navigli and S. P. Ponzetto.BabelNet : The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence, 193 :217–250, 2012.

[23] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, et Katherine Miller. Wordnet : An on-line lexical database. International Journal of Lexicography, 3 :235–244, 1990.

[24] Roberto Navigli et Simone Paolo Ponzetto. Babelnet : Building a very large multilingual semantic network. In Proceedings of the 48th annual meeting of the association for computational linguistics, pages 216–225. Association for Computational Linguistics, 2010.

[25] Andrea Moro, Alessandro Raganato, etRobertoNavigli. Entity linking meetsword-sense disambiguation : a unified approach. TACL, 2 :231–244, 2014.

[26] Roberto Navigli, David Jurgens, et Daniele Vannella. SemEval-2013 Task 12 :MultilingualWord Sense Disambiguation. In Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2 : Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 222–231, 2013. URL http://www.aclweb.org/anthology/S13- 2040.

[27] Andrea Moro et Roberto Navigli. Semeval-2015 task 13 : Multilingual all-words sense disambiguation and entity linking. In Proceedings of the 9th International

Workshop on Semantic Evaluation (SemEval 2015), pages 288–297, Denver,

Colorado, June 2015. Association for Computational Linguistics.URL http://www.aclweb.org/anthology/S15-2049.

[28] Benoit Habert, Cécile Fabre, et Fabrice Issac. De l'écrit au numérique. Constituer, normaliser et exploiter les corpus électroniques. Elsevier Masson, 1998.

[29] W. N. Francis et H. Kucera. A Standard Corpus of Present-Day Edited American

English, for use with Digital Computers (Brown).Technical report, Brown University, Providence, Rhode Island, 1964. Lou Burnard. The british national corpus, 1998

[30] Nancy Ide et Catherine Macleod. The american national corpus : A standardized resource of american english. In Proceedings of Corpus Linguistics 2001, volume 3, 2001.

[31] Roberto Navigli, Kenneth C. Litkowski, et Orin Hargraves. Semeval-2007 task 07:

Coarse-grained english all-words task. In SemEval-2007, pages 30–35, Prague,

Czech Republic, June 2007.

[32] David Vickrey, Luke Biewald, Marc Teyssier, et Daphne Koller. Word-sense disambiguation for machine translation. In Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, HLT '05, pages 771–778, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.doi : 10.3115/1220575.1220672. URL

http://dx.doi.org/10.3115/1220575.1220672.

[33] Diana McCarthy. Lexical substitution as a task for WSD evaluation. In Proceedings of the ACL-02 Workshop on Word Sense Disambiguation : Recent Successes and Future Directions, pages 089–115. Association for Computational Linguistics, July 2002. doi : 10.3115/1118675.1118691. URL

https://www.aclweb.org/anthology/W02-0816

[34] Mohammad Taher Pilehvar et Jose Camacho-Collados. WiC : the word-in-context dataset for evaluating context-sensitive meaning representations. In Proceedings of the

2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers), pages 1267–1273, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi : 10.18653/v1/N19-1128.