

Institut National des Langues et Civilisations Orientale

Département Textes, Informatique, Multilinguisme

Projet de programmation itérative/réursive

MASTER

TRAITEMENT AUTOMATIQUE DES LANGUES

Parcours :

Ingénierie Multilingue

par

Sheherazade NINEB

Xinyi SHEN

Directeur de mémoire :

Patrick Paroubek

Année universitaire 2020/2021

TABLE DES MATIÈRES

Résumé 3

Abstract 3

INTRODUCTION 4

1. Présentation générale 4
2. Problématique de recherche 4
3. Plan 4

Première partie 5

Contexte général 5

Chapitre 1 6

ÉTAT DE L'ART 6

Chapitre 2 8

DESCRIPTION DU PROJET 8

2.1 Corpus 8

2.2 Outils principaux 9

Deuxième partie 12

Expérimentations 12

Chapitre 3 13

MÉTHODES 13

3.1 Traitement du corpus xml 13

3.2 Complétion d'annotation des occurrences 14

3.3 Evaluation de reconnaissance d'ingrédients 19

3.4 Complexité en temps et corrélation entre le niveau du recette 20

Chapitre 4 22

RÉSULTATS 22

4.1 Résultats 22

4.2 Difficultés et conclusion 38

Troisième partie 42

Annexe 42

BIBLIOGRAPHIE 43

Résumé

Nous présentons ici un modèle de calcul de complexité en temps des programmes informatiques appliqués aux recettes de cuisine en français. Dans un premier temps, afin de bien identifier les différents éléments des recettes utiles au calcul – *et sans lesquels la complexité ne peut être calculée* – à savoir les ingrédients, les ustensiles de cuisine utilisés et les opérations culinaires effectuées, nous avons imbriqué deux méthodes : une consistant à extraire du vocabulaire relatif au domaine à travers internet et quelques fonctionnalités de spaCy pour constituer un premier lexique et une seconde méthode qui tente d'identifier ces éléments de la façon la plus automatisée possible grâce à un plongement des mots et à un algorithme non supervisé de type clustering. Ces étapes permettent la création de nouvelles entités nommées françaises par rapport aux recettes : VerbeCulinaire, Ingredient et Ustensile et donc l'apprentissage d'un modèle. A partir de la reconnaissance des entités nommées, nous avons calculé pour chaque recette sa complexité en temps et avons fait la corrélation entre la complexité en temps et le niveau difficulté de chaque recette.

Mot-clés : plongement des mots, algorithme non-supervisé, apprentissage, entité nommée, calcul de complexité

Abstract

We present here a model for calculating the time complexity of computer programs applied to cooking recipes in French. First, in order to recognize the recipes, we will create the new french entities named in relation to the recipes: VerbeCulinaire, Ingredient and Ustensile. And then, from the named entities, we calculated for each recipe its complexity in time. In the end, we made the correlation between the complexity of the time and the level of each recipe.

Keywords : name entity, complexity calculation

INTRODUCTION

1. Présentation générale

Une recette de cuisine est l'ensemble des ingrédients et des opérations nécessaires pour effectuer une préparation alimentaire en cuisine à l'aide d'ustensiles¹.

A ce jour, un volume important de recettes de cuisine est disponible que ce soit à partir des livres de cuisine ou d'Internet. On comprend aisément que la tâche qui consiste à trier toutes les recettes par rapport à leur complexité manuellement ne soit pas simple.

Ici, nous proposons un modèle de calcul de complexité en temps et en espace des programmes informatiques appliqué aux recettes de cuisine en français.

2. Problématique de recherche

L'objectif de ce projet est d'apporter une réflexion sur ces quelques questions :

- Comment compléter les annotations des occurrences des opérations élémentaires dans le texte des recettes ?
- Comment faire un entraînement de la performance de la fonction de reconnaissance d'ingrédients par rapport à la liste d'ingrédients mentionnée dans la recette avec les mesures de précision et de rappel ?
- Comment calculer la complexité en temps ?

3. Plan

Le chapitre 1 contient un état de l'art des différentes approches de la reconnaissance des recettes culinaires. Le chapitre 2 est une description du projet et les outils principaux pour réaliser le projet. Dans le chapitre 3, nous décrivons les méthodes employées pour produire le projet. Dans le chapitre 4, nous présentons les résultats et les difficultés que nous avons rencontrés.

¹ Recette de cuisine : https://fr.wikipedia.org/wiki/Recette_de_cuisine

Première partie

Contexte général

Chapitre 1

ÉTAT DE L'ART

La structure des recettes se composent souvent d'au moins trois parties :

- La section « ingrédients », qui liste les ingrédients nécessaires et leur quantité. La syntaxe de ses phrases ne répond pas à une « grammaire classique » et fait plutôt penser à la structure concise des tweets.
- La section « préparations », qui liste une série d'actions (techniques/process) (souvent des verbes) à appliquer sur les ingrédients et les ustensiles. Cette section est écrite dans un langage naturel classique.
- Et le titre de la recette, qui est souvent une phrase concise dans laquelle on retrouve parfois les ingrédients principaux. A noter, en particulier pour la langue française, l'art de « concocter des titres en cuisine » qui utilise un vaste espace figuratif du goût par exemple : « Carré frais et ses compagnons d'été ».

Les tâches d'extraction et de structuration des informations dans les recettes de cuisine ont été préalablement étudiées. La reconnaissance d'entités nommées dans les recettes est souvent coûteuse car il s'agit généralement d'un processus d'annotation par l'humain.

Rahul Agarwal, Kevin Miller [10] ont proposé un modèle Maximum Entropy Classification pour identifier les mots clés dans les ingrédients et un SRL (semantic role labelling) pour les préparations. Cependant, pour bien identifier toutes les entités nommées, l'aide humaine a été essentielle. Après la première SRL, les humains ont du faire la correction des résultats pour mieux extraire les entités nommées.

Nuno Silva, David Ribeiro et Liliana Ferreira [9] ont proposé un modèle CRF(Conditional Random Field) pour l'extraction des informations, y compris les ingrédients, les quantités, les unités et les commentaires dans les parties ingrédients afin de créer les entités nommées. Un corpus d'entraînement annoté a été le prérequis pour cette étape. Pour identifier les actions (verbes) dans les parties préparations, ils ont utilisé le POS (Part Of Speech) tagging et une analyse de la structure grammaticale.

Mori Shinsuke, Maeta Hirokuni, Yamakata Yoko et Sasada Tetsuro [4] ont rapporté les détails du corpus de diagramme de flux construit à partir de textes de recettes en japonais. Ils ont choisi le graphe acyclique dirigé enraciné comme représentation de la signification des textes de recette. Yuzhe Chen [6] a aussi utilisé cette méthode : à l'état de départ, une collection d'ingrédients est introduite. Ensuite, une série d'actions sont effectuées sur ces ingrédients (avec les ustensiles de cuisine). Un système informatique capable d'extraire de telles structures à partir de recettes alimentaires peut avoir de nombreuses applications utiles.

Jushaan Kalra, Devansh Batra, Nirav Diwan et Ganesh Bagler [3] ont utilisé NER avec Jaccard Similarity et Unit Mapping sur une grande base de données contenant plus de 118 000 recettes pour fournir des estimations précises des profils nutritionnels malgré des données extrêmement bruyantes et variées.

Nirav Diwan, Devansh Batra et Ganesh Bagler [1] ont construit un vecteur similaire au word2vec et puis créé les clusters dans les vecteurs pour détecter tous les formats dans les ingrédients pour diminuer les travaux de l'annotation et créé les entités nommées. Pour les préparations, ils ont utilisé les arbres d'analyse des dépendances pour reconnaître les ingrédients, les actions (verbes) et les ustensiles. Nous nous sommes inspirés de ces travaux et avons utilisé des fonctionnalités de Spacy en plus d'une correction manuelle de certains résultats afin d'éviter l'annotation manuelle.

Chapitre 2

DESCRIPTION DU PROJET

Dans cette partie, nous allons présenter le corpus du projet ainsi que les outils principaux de notre projet.

2.1 Corpus

Le corpus de notre projet contient 23071 recettes sous le format xml.

Dans chaque fichier xml, l'élément racine « recette » inclut six éléments : « titre », « type », « niveau », « cout », « ingredient » et « préparation ».

L'élément « titre » représente le titre de la recette, par exemple : « Recette de lapin à la bière Quintine et 2 raisins ».

L'élément « type » représente le type de cette recette, par exemple : « Desert », « Plat principal », etc...

L'élément « niveau » représente le niveau de la difficulté par rapport à cette recette, par exemple : « moyennement difficile ».

L'élément « cout » est le niveau du coût de cette recette, par exemple : « Moyen », « bon marché », etc...

Les éléments « ingredient » et « préparation » sont les éléments essentiels dans la description d'une recette. L'élément « ingredient » contient les éléments enfants « p ».

Chaque élément « p » représente un ou plusieurs ingrédient(s) pour cette recette.

L'élément « préparation » contient la section CDATA (Character Data). La section CDATA inclut les étapes détaillées des opérations de cette recette sur une ou plusieurs lignes.

La figure 2.1 est un exemple du fichier XML de notre corpus.

Figure 2.1 - Exemple d'une recette du corpus

```

<?xml version="1.0" encoding="utf-8"?>
<recette id="230010">
  <titre>Recette de lapin à la bière Quintine et 2 raisins</titre>
  <type>Plat principal</type>
  <niveau>Moyennement difficile</niveau>
  <cout>Moyen</cout>
  <ingredients>
    <p>2 lapins de 1,5 kg sans les viscères</p>
    <p>2 grappes de raisin blanc sans pépins</p>
    <p>2 grappes de raisin noir sans pépins</p>
    <p>2,3 l de bière Quintine ou Équivalent (bière artisanale de préférence titrant minimum 8% d'alcool)</p>
    <p>huile d'olive</p>
    <p>beurre</p>
    <p>sel et poivre</p>
  </ingredients>
  <preparation>
    <![CDATA[
      Faire saisir le lapin dans une grande marmite avec l'huile d'olive et le beurre.
      Ensuite enlever le lapin de la marmite, déglaçer avec la bière (les 7 bouteilles).
      Incorporer à la sauce tout le raisin le blanc et le noir, remettez-y également le lapin.
      Laissez mijoter de 1 heures à 2 heures suivant que vous désirez une viande plus ferme ou plus
      tendre.
      Bières Artisanales, Vin Rouge Château la tour Figeac
    ]]>
  </preparation>
</recette>

```

2.2 Outils principaux

Afin de réaliser ce projet, nous avons utilisé différentes bibliothèques de Python pour chaque étape. Ici, nous présentons les outils principaux: `xml.dom` et `xml.etree` pour parser le corpus original et récupérer les contenus; Spacy et NLTK pour les opérations d'analyse des textes, du traitement et de l'apprentissage automatique des langues, Genism pour le plongement des mots avec `word2vec`, Sklearn mais aussi NLTK pour la partie machine learning.

2.2.1 `xml.dom`

Le corpus de recettes culinaires est un ensemble de fichiers xml. Notre objectif dans cette étape est de récupérer l'élément « ingredients », l'élément « preparation » et l'élément « titre ». Pour cela, nous avons choisi `xml.dom` et `xml.etree`.

Le Document Object Model, ou "DOM," est une API inter-langage du World Wide Web Consortium (*W3C*) pour accéder et modifier les documents XML.² Il représente xml comme un arbre et il permet d'accéder au xml et de trouver tous les éléments.

Pour notre corpus, la structure est bien construite et assez simple. Donc, nous n'avons pas besoin d'un module compliqué pour analyser et récupérer le contenu. Ici, nous avons donc choisi `xml.dom.minidom` et `xml.etree`. Ces modules sont plus légers et plus simples à utiliser. C'est idéal pour notre projet. Les fichiers suivants : *TraitementXML.py* et *fromXML2PandaDf.py* créées soit des fichiers pour chaque section soit un fichier JSON qui contient ces trois sections pour un chargement automatique dans un dataframe.

2.2.2 spaCy

Après avoir récupéré le contenu dans les fichiers xml, nous avons traité les textes avec l'outil spaCy.

SpaCy est une bibliothèque logicielle Python de traitement automatique des langues développée par Matt Honnibal et Ines Montani. SpaCy est livré avec des pipelines pré-

² `xml.dom` — L'API Document Object Model: <https://docs.python.org/fr/3/library/xml.dom.html>, 14/05/2021

formés et prend actuellement en charge la tokenisation et la formation pour plus de 60 langues, y compris la langue française. Il propose des modèles de vitesse et de réseau neuronaux de pointe pour le marquage, l'analyse, la reconnaissance d'entités nommées, la classification de texte et plus encore, un apprentissage multi-tâches avec des transformateurs pré-entraînés comme BERT, ainsi qu'un système de formation prêt pour la production et un modèle facile pour le packaging, le déploiement et la gestion des workflows.

Pour compléter les annotations, nous avons utilisé son pos-tagging, la lemmatisation, la reconnaissance d'entités nommées ainsi que l'ajout de nouvelles entités nommées, de règles de patrons linéaires, et surtout avons entraîné le modèle de reconnaissance d'entités nommées suite à un important prétraitement qui a permis de fournir des listes d'ingrédients, de verbes et d'ustensiles. Ce module nous fournit alors le modèle de sortie, ainsi que les mesures de précision et de rappel pour l'évaluation de la performance du modèle.

2.2.3 NLTK

Afin de faire la tokenisation, la stemmatisation et le clustering via K-Means, nous avons utilisé une autre bibliothèque de traitement automatique des langues, NLTK. NLTK a été développé par Steven Bird et Edward Loper au Département d'informatique et des sciences de l'information de l'Université de Pennsylvanie. NLTK comprend des démonstrations graphiques et des exemples de données. Il est accompagné d'un livre qui explique les concepts sous-jacents derrière les tâches de traitement du langage prises en charge par la boîte à outils, et d'un livre de traitement sur des recettes.

2.2.4 Gensim

Avant passer au processus de machine learning et de clustering, nous avons créé les vecteurs de word-embedding grâce à word2vec du module Genism. Gensim est conçu pour gérer de grandes collections de texte à l'aide de flux de données et d'algorithmes en ligne incrémentiels, ce qui le différencie de la plupart des autres logiciels d'apprentissage automatique qui ne ciblent que le traitement en mémoire.

2.2.5 scikit-learn (sklearn)

Pour entraîner notre modèle de reconnaissance d'informations dans les recettes culinaires, nous avons choisi le module scikit-learn en python. Il est basé sur NumPy, sciPy et matplotlib et il permet de faire divers algorithmes de classification, de régression et de clustering, y compris les machines vectorielles de support, les forêts aléatoires, l'augmentation de gradient, les k-means et DBSCAN.

Pour notre étude, nous avons utilisé K-Means pour clustérer nos données et avons utilisé pour visualiser et analyser nos données : une analyse en composantes principales (PCA) ainsi qu'un algorithme de réduction de dimensionnalité appelé t-distributed stochastic neighbor embedding (t-SNE), algorithme non-linéaire de “feature extraction”, qui construit une nouvelle représentation des données de telle sorte que les données proches dans l'espace original aient une probabilité élevée d'avoir des représentations proches dans le nouvel espace.

Deuxième partie

Expérimentations

Chapitre 3

MÉTHODES

3.1 Traitement du corpus xml

A partir de notre corpus initial, nous constituons deux corpus : un premier qui comprend toutes les données de la section « ingredients » de toutes les recettes et un second qui comprend toutes les descriptions de préparation de chaque recette.

Nous avons mis en place deux méthodes d'extraction de ces données : une qui génère ces corpus dans des fichiers .txt séparés, une seconde qui génère un fichier unique JSON facilement chargeable dans un dataframe (JSON qui comprend en plus tous les titres de recettes).

La première méthode utilise le code du fichier suivant : *TraitementXML.py*.

Pour cela, nous avons créé une boucle pour passer tous les fichiers xml et dans la boucle, premièrement, nous avons utilisé le module `xml.dom` pour faire parser xml. Et puis, nous avons utilisé “`xml_file.documentElement`” pour trouver tous les éléments dans xml. Ici, les deux éléments utiles sont “`p`” et “`preparation`”.

Dans chaque fichier xml, chaque élément “`p`” représente un ingrédient dans la recette. Et l'élément “`preparation`” indique toutes les étapes nécessaires de la recette.

Pour récupérer toutes les données d'ingrédients, nous avons choisi “`firstChild.data`”. Par contre, l'élément “`preparation`” contient la section CDATA, autrement dit, “`character data`”. Pour enlever CDATA et n'avoir que le contenu nécessaire, ici nous avons utilisé “`firstChild.wholeText`”. Une recette culinaire correspond à une ligne de ce nouveau corpus.

Ensuite, nous avons enregistré chacun dans deux listes différentes et puis nous avons créé deux fichiers txt, “`ingredients.txt`” et “`recettes.txt`”, afin de sauvegarder les données.

Figure 3.1 un extrait du fichier “`ingredients.txt`” et “`recettes.txt`”

8 grandes tranches de pain de mie(special sandwich) Une boule de mozzarella 4 tranches de saumon fumé Fromage frais Aneth 4 oeuf frais 500 g de champignons 1 citron 1 l de bouillon de volaille 85 g de beurre 85 g de farine 10 cl de crème 1 échalote 1 petit oignon persil sel poivre 2 lapins 2 têtes d'ail 25 cl de lait 2 cuillères à soupe de crème fraîche 1 cuillère à soupe de thym séché sel, poivre et noix de muscade 12 tomates 450 g de dés de jambon 450 g de gouda 30 g de persil haché	1-Dans une casserole d'eau bouillante plonger délicatement les œufs à l'aide d'une cuillère à soupe et les laisser cuire 5min (Ils doivent rester coulant à l'intérieur)Les écailier, réserver.2-Couper les tranches de saumon fumé en plusieurs morceaux et la mozzarella en tranches.3-Dans un bol, mélanger 4 cuillères à soupe rases de fromage frais avec de l'aneth à votre convenance.4-Toaster les tranches de pain de mie,les tartiner de fromage frais,disposer les tranches de mozzarella et de saumon fumé au centre ainsi que les œufs pochés.5-Dans un appareil à croque monsieur, faire doré les clubs sandwiches.la mozzarella doit être fondu Couper les clubs sandwiches en 2 et servir avec une salade verte.Ce club sandwich ravira tous vos convives par son originalité et le mariage de saveurs qu'il propose.A servir avec une salade composée. Si vous avez peur que les œufs éclatent lors de la cuisson dans l'appareil à croque monsieur vous pouvez les partager en 2 ! Couper le pied des champignons puis les laver dans plusieurs bains d'eau légèrement vinaigrée. Les napper ensuite de citron, puis les hacher. Peler et hacher l'échalote et l'oignon puis les mettre à cuire dans une poêle beurrée. Incorporer ensuite les champignons, saler, poivrer puis les laisser cuire jusqu'à évaporation complète du jus de cuisson. Dans une casserole, faire fondre le reste du beurre avec la farine et mouiller le tout de bouillon froid. Verser ensuite la crème sur les champignons cuits, puis laisser cuire encore 10 min environ. Ajouter la crème puis saupoudrer de persil haché et servir bien chaud dans une soupière. Vous pouvez également l'agrémenter de
---	---

La seconde méthode utilise le code du fichier suivant : *fromXML2PandaDf.py*. Ce code parcourt toutes les recettes, crée un dictionnaire contenant les sections pointées de chaque recette et met bout à bout dans une liste ces dictionnaires. Cette liste est ensuite enregistrée dans un unique JSON *jsonCorpusRecettes.json*.

The screenshot shows a Jupyter Notebook interface. The top cell displays the JSON content of 'jsonCorpusRecettes.json', which contains two recipe entries. The bottom cell shows the Python code used to load this JSON into a Pandas DataFrame and display its head.

```

{
    "titre": "Poulet tandoori rouge",
    "preparation": "A préparer la veille !Mélanger l'épice Tandoori (épice indienne",
    "ingredients": [
        "4 escalopes de poulet",
        "4 cuillères à soupe d'épice Tandoori",
        "25 cl de lait (ou de yaourt grec)",
        "3 cuillères à soupe d'huile",
        "3 cuillères à soupe de jus de citron"
    ],
    "titre": "Lasagnes végétariennes (facile)",
    "preparation": "Si vous utilisez des oignons, faites-les revenir (dans une saut",
    "ingredients": [
        "6 tomates fraîches (ou pelées en boîte, à défaut)",
        "4 ou 5 courgettes",
        "sel, poivre, basilic frais, herbes de Provence",
        "lasagnes",
        "400 g de gruyère râpé",
        "200 g de coulis de tomates",
        "4 oignons de taille moyenne",
        "50 cl de béchamel ou (si on est pressé) de crème fraîche épaisse ou semi-ép
    ],
    "titre": "Terrine de foie de porc",
    "preparation": "- hacher (pas trop fin) le foie, le lard, les ...",
    "ingredients": [
        "400 grammes de foie de porc , 200 grammes de ..."
    ],
    "titre": "Lapin au vin blanc (à la cocotte)",
    "preparation": "Faire revenir les échalotes et les lardons dan...",
    "ingredients": [
        "[1,8 kg de lapin (7 à 8 morceaux), 1 bouteille...]"
    ],
    "titre": "Escalopes lucullus à la cocotte",
    "preparation": "Coupez l'oignon en petits morceaux, faites rev...",
    "ingredients": [
        "[4 escalopes, 4 tranches de bacon, 4 tranches ...]"
    ]
}

```

Load Data

```

import pandas as pd

data= pd.read_json('jsonCorpusRecettes.json')

data.head()

```

	titre	preparation	ingredients
0	Poulet tandoori rouge	A préparer la veille !Mélanger l'épice Tandoor...	[4 escalopes de poulet, 4 cuillères à soupe d'...
1	Lasagnes végétariennes (facile)	Si vous utilisez des oignons, faites-les reven...	[6 tomates fraîches (ou pelées en boîte, à déf...
2	Terrine de foie de porc	- hacher (pas trop fin) le foie, le lard, les ...	[400 grammes de foie de porc , 200 grammes de ...
3	Lapin au vin blanc (à la cocotte)	Faire revenir les échalotes et les lardons dan...	[1,8 kg de lapin (7 à 8 morceaux), 1 bouteille...
4	Escalopes lucullus à la cocotte	Coupez l'oignon en petits morceaux, faites rev...	[4 escalopes, 4 tranches de bacon, 4 tranches ...]

Figure 3.2 un extrait du fichier “*jsonCorpusRecettes.json*” et son chargement instantané.

3.2 Complétion d’annotation des occurrences

3.2.1 Récupération des listes d’ustensiles, des verbes culinaires et la liste d’ingrédients

Nous avons pris parti de ne pas faire d’annotations manuelles ni via InCeption ni via l’outil d’annotation de spaCy ni via tout autre outil pourtant la majorité des études parcourues posent ce prérequis. Pour rappel, notre corpus est constitué de 23071 recettes et par expérience nous savons que pour obtenir de bons scores c’est à dire un modèle qui détecte suffisamment bien les nouvelles entités nommées, il nous faut

un nombre important d'annotations or cette tâche, bien qu'elle soit très lourde et sans difficulté, est surtout extrêmement couteuse en temps.

Nous avons donc suivi tout en modifiant et combinant plusieurs approches dont celles proposées par [1] Nirav Diwan, Devansh Batra et al, et celle proposée par William Mattingly [11]. En d'autres termes, nous avons tenté une approche qui combine apprentissage non supervisé, rectification manuelle des résultats, utilisation des patrons et des rule-based matching de spaCy afin d'extraire des entités mais aussi de créer nos data d'entraînement et de tests pour un entraînement final du modèle de reconnaissance via spaCy.

L'idée première n'était pas de passer par de l'embedding et un algorithme non-supervisé mais de créer/d'extraire de sites internet (sans passer par une étape de scrapping) trois listes de lexiques culinaires : une première pour les ingrédients, une seconde pour les ustensiles et une dernière et troisième pour les verbes relatifs aux techniques/process de cuisine qui auraient permis une construction directe des data d'entraînement. Première constatation : la pauvreté de bases de données en français. Aucune centralisation de ce type de données n'a été trouvée. Nous avons alors collecté ces lexiques sur différentes pages du web. Première conclusion : nos lexiques sont pauvres en termes de volume pour pouvoir couvrir l'ensemble ou du moins une partie importante du vocabulaire employé dans notre corpus. Néanmoins nous disposons d'une liste pour les ingrédients (*IngredientParSpacy.txt*), d'une liste pour les verbes(*verbesCuisineSurInternet.txt*) et d'une liste pour les ustensiles(*ustensilesCuisineSurInternet.txt*).

Nous nous sommes concentrées sur les ingrédients dans une première étape.

Nous avons alors investigué le champ des patterns que spaCy propose (spaCy propose une déclinaison importante et intéressante de fonctionnalités qui permettent d'extraire des informations à partir de l'analyse morphosyntaxique des phrases) afin de les appliquer sur le corpus « ingredients.txt » dans un premier temps. Seconde constatation : l'analyse de spaCy contient beaucoup d'erreurs et ne permet pas une extraction du vocabulaire sans introduire de nombreuses fautes. Prenons le cas de la fonction pos_tag : nous avons observé que pour beaucoup de schéma « Nom Adjectif », comme dans *sucré vanillé*, l'analyse de spaCy retournait « Nom Nom ». Il devient alors difficile d'extraire de *sucré vanillé* le nom *sucré*, qui nous intéresse, à partir de son POS tag sachant que *vanillé* a le même. Pour contourner ce problème, nous avons créé deux schémas : « NOUN » et « NOUN NOUN » pour extraire tous les noms mais aussi les schémas « Nom Adjectif ». Ils ont été enregistrés dans deux listes.

Nous supposons que dans plupart des cas, le schéma "NOM Adjectif" est plus commun que le schéma "Adjectif NOM", et à partir de là, nous avons récupéré le deuxième "NOM" du schéma proposé par spaCy "NOUN NOUN" et supposé qu'ils

sont tous adjetifs. Nous avons enregistré les adjetifs dans une liste et après enlevé les adjetifs dans le schéma “NOUN” (qui était déjà enregistré dans une liste). Et puis, parfois, nous pouvons voir les mots ustensiles dans les phrases de l'ingrédients, donc, nous avons créé la liste d'ustensiles que nous avons récupéré sur Internet et les avons enlevés aussi par le schéma “NOUN”. À ce moment, nous avons eu une liste d'ingrédients pas propre, et nous avons passé le processus manuel de nettoyer la liste, et finalement, nous avons créé un fichier : *IngredientParSpacy.txt*.

Les expérimentations sous spaCy et le peu de visibilité sur les probables résultats finaux nous ont dirigé en parallèle vers le champ de l'apprentissage non-supervisé.

Le fichier *nerW2V\IngredientSeqLevel.py* étudie la partie « ingrédients » du corpus et fournit en résultat une liste d'ingrédients.

Nous avons cherché à mettre en place une méthode semi-automatique de détection des ingrédients afin de simplifier, rendre l'annotation optimale (dans le sens où il est impossible d'annoter un nombre significatif de recettes: cette méthode rendra alors l'annotation optimale quant à la sélection des données à annoter) tout en la diminuant au maximum.

L'idée ici est que les phrases qui décrivent/experimentent les quantités, les ingrédients nécessaires à la recette répondent à un pattern d'une façon générale.

Pour mettre en évidence ces patterns nous avons opté pour :

1. un prétraitement du texte : normalisation, extraction des ponctuations, stemmatization (équivalent à lemmatisation pour enlever les doublons dus au pluriel, les formes dérivées...),
2. un embedding (de type skipgram) à l'échelle des phrases sur le texte prétraité,
3. l'utilisation de la méthode Elbow afin d'identifier/sélectionner le nombre de cluster qui répondrait le mieux à notre corpus,
4. l'application ensuite d'un apprentissage non-supervisé de type KMeans,
5. une estimation des résultats via un score et un silhouette_score,
6. une visualisation des résultats via une PCA en 2 dimensions (N.B.: nous n'avons pas utilisé la PCA pour accélérer l'apprentissage)

Nous pensions dans un premier temps sélectionner un pourcentage de phrases répondant à des patterns différents suite à cette étude pour couvrir le plus de formes possibles avec l'intuition que cela optimiserait la phase d'annotation manuelle qui aurait ensuite été menée. Nous avons finalement opté pour une extraction directe de ces données suite à l'analyse des formes et à leurs régularités. Une rectification

manuelle des mauvaises extractions a cependant été nécessaire mais cela nous semblait plus souple à mettre en œuvre que l'annotation.

L'embedding à l'échelle de la phrase s'est fait suite à un filtre sur le corpus (méthode *filter_docs*) afin de conserver le vocabulaire présent dans le dictionnaire *word2vec*. Ce après quoi il a été possible de tester la méthode Elbow qui devait nous aider à définir un nombre de clusters. Nous avons enfin utilisé l'algorithme non-supervisé de clustering : K-Means en choisissant un partitionnement sur 7 clusters. Les résultats sont présentés dans le chapitre suivant.

Concernant l'extraction des verbes et ustensiles, nous avons mené une étude similaire en supprimant les stopwords et une liste non significative d'adjectifs et d'adverbes. Cette fois-ci l'embedding (pour expérience) s'est fait à l'échelle des mots sur le corpus « préparation ». A ce stade, il semble difficile d'extraire des informations à cette échelle. Il faudrait, car nous n'avons pas eu le temps, ajouter l'embedding à l'échelle des phrases pour, de la même façon que les ingrédients, réussir à extraire des patterns de structure grammaticale de phrases afin d'extraire les actions (souvent similaire à des verbes) et les ustensiles. La liste des ingrédients pourrait soit être extraite par la même occasion une seconde fois soit être exclue des extractions grâce à la liste existante. (Le fichier *nerW2VPreparationWordLevel.py* étudie la partie « préparation » du corpus)

Nous avons malgré tout tenté d'entrainer un modèle en prenant en considération des verbes, des ustensiles, des ingrédients et des quantités. Nous sommes donc retournées vers spaCy et avons utilisé ses patrons que nous avons adapté suite à nos analyses pour extraire des techniques/actions (que nous avons restreint à une identification de verbes correspondant aux « root » des syntagmes nominaux qui répondent aux tags « obj » et « conj ») et les quantités qui répondent aux schémas « NUM » ou « NUM UNIT » en fournissant manuellement une liste d'unités (grammes, cuillère, etc..). Les résultats de sortie de spaCy n'étaient pas parfaits, une correction manuelle lourde a été menée pour extraire les verbes (passage d'une liste à plus de 5000 mots à une liste de 226 verbes le résultat étant le fichier *verbesSpacyCorrec.txt*. On note une difficulté même humaine à trancher parmi le vocabulaire pour exclure les mots qui ne seraient pas totalement de l'ordre des techniques de réalisation des recettes.).

Nous avons alors utilisé ces listes et celles initialement créées pour entraîner un modèle à reconnaître ces 4 entités nommées : INGREDIENT, ACTION, USTENSIL et QUANTITY.

<pre> ingredients = "4 tomates rondes \n \ 150g de lamelles de saumon fumé\n \ 150 g de crevettes roses décortiquées\n \ 3 Carrés Frais\n \ ciboulettes hachée + quelques feuilles pour la déco\n \ 1 cuillère d'huile d'olive, sel, poivre\n" #création d'un doc à partir du modèle nlp doc = nlp(ingredients) for entity in doc.ents: print(entity.text,entity.start_char, entity.end_char,entity.label_) 4 0 1 DIGIT 150g 23 27 QUANTITY 150 g 60 65 QUANTITY 3 103 184 DIGIT 1 cuillère 179 189 QUANTITY </pre>	<pre> grill 5 10 ACTION poêl 19 23 USTENSILE rôti 24 27 INGREDIENT brochet 28 35 INGREDIENT serv 36 40 USTENSILE sauc 41 45 INGREDIENT menth 46 51 INGREDIENT épic 10 14 INGREDIENT tikk 15 19 INGREDIENT plat 16 20 USTENSILE salad 21 26 INGREDIENT tranch 27 33 INGREDIENT oignon 38 44 INGREDIENT vin 0 3 INGREDIENT oignon 7 13 INGREDIENT reven 14 19 ACTION sautéux 20 27 USTENSILE wok 28 31 USTENSILE </pre>
--	---

Figure 3.2.1 Exemple d'extraction des quantités et d'annotations

```

listVerb2 = []
#iterate over the corpus to extract verbs
for sentence in var:
    doc = nlp(sentence)

    #extract verbs
    for sent in doc.sents:
        for chunk in sent.noun_chunks:
            if chunk.root.dep_ in ['obj','conj']:
                if chunk.root.head.text != ' ':
                    listVerb2.append(chunk.root.head.text)
                    print([chunk.text, chunk.root.text, chunk.root.dep_,chunk.root.head.text])

listVerb2=list(set(listVerb2))

[ 'la veille', 'veille', 'obj', 'préparer']
[ 'le lait ou de yaourt grec, l huile et le jus de citron', 'lait', 'obj', 'mélanger']
[ 'les escalopes de poulet', 'escalopes', 'obj', 'découper']
[ 'le gras', 'gras', 'obj', 'enlever']
[ 'les', 'les', 'obj', 'mélanger']
[ '24h', '24h', 'obj', 'reposer']
[ 'les morceaux à la poêle ', 'morceaux', 'obj', 'faire']
[ 'ou encore un plat avec une salade, des tranches crues d oignons doux', 'plat', 'conj', 'entrée']
[ 'des oignons', 'oignons', 'obj', 'utilisez']
[ 'les tomates', 'tomates', 'obj', 'coupez']
[ 'les courgettes en rondelles', 'courgettes', 'obj', 'coupez']
[ 'et 1 cuillère à café de sucre en poudre ( plus en hiver,', 'cuillère', 'conj', ' ' ]
[ 'à feu moyen', 'feu', 'conj', 'mijoter']
[ 'des carottes', 'carottes', 'obj', 'rajouter']
[ 'la dernière couche de gruyère par du comté', 'couche', 'obj', 'remplace']
[ 'un robot à ajouter 1 oeuf', 'robot', 'obj', 'aide']
[ 'le mélange pas', 'mélange', 'obj', 'verser']
[ 'la terrine', 'terrine', 'obj', 'remplir']
[ 'les échalotes', 'échalotes', 'obj', 'revenir']
[ 'les morceaux de laniñ', 'morceaux', 'obj', 'ajouter' ]

```

Figure 3.2.1 Exemple de champ investigué pour extraire les process de cuisine

3.2.2 Ajout d'entités nommées et création des dataset d'entraînement et d'évaluation

Pour retrouver tous les ingrédients et les autres entités nommées utiles, nous avons décidé de créer quatre types d'entités nommées et de les appliquer sur le corpus de préparation des recettes avec les fonctions de spaCy.

N.B. : Le premier modèle a été entraîné sur les ingrédients uniquement. Le second modèle sur le tout.

Le fichier *fromDataClusterToDataSpacy.ipynb* permet de générer les data d'entraînement et d'évaluation suite au travail préalable d'identification des ingrédients.

Le fichier *fromDataClusterAndNetToDataSpacy.ipynb* permet de générer les data d'entraînement et d'évaluation suite au travail d'identification et de correction d'une liste de verbes, d'identification des quantités et des listes d'ingrédients et d'ustensiles.

SpaCy nous permet de créer les nouvelles entités nommées grâce à ses fonctions `spacy.pipeline`, `EntityRuler` et `spacy.language`. Il suffit de créer le nom de l'entité nommée et de lui associer une liste (d'ingrédients, de verbes, d'ustensiles, ou d'unités) et d'ajouter cet *EntityRuler* dans le pipeline. Ceci permet alors de créer les fichiers d'entraînement et de test.

On relève par la même occasion une fonctionnalité intéressante de visualisation de spaCy (fonction *displacy*) qui nous permet de vérifier les annotations simplement.

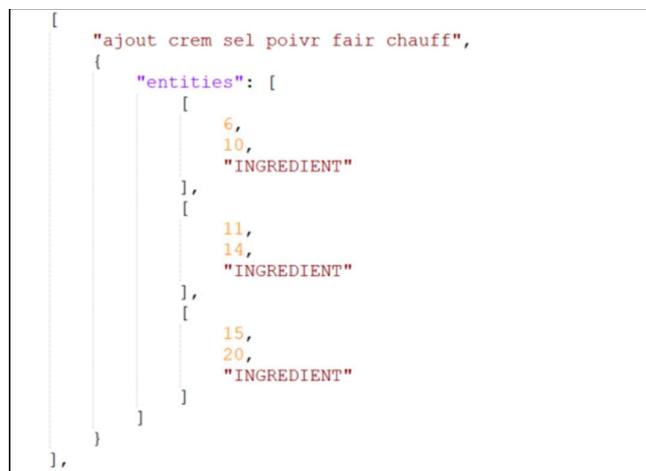


Figure 3.2.2 Illustration d'un extrait du fichier d'entraînement

```
[23] > [REDACTED] M4
doc=nlp1('lasagn v\u00e9g\u00e9tarien facil terrin foi porc lapin vin blanc cocott escalop lucullus cocott tart oignon carr coq vin facil ')
[24] > [REDACTED] M4
spacy.displacy.render(doc, style="ent", jupyter=True)
```

lasagn INGREDIENT v\u00e9g\u00e9tarien facil terrin INGREDIENT foi INGREDIENT porc INGREDIENT lapin INGREDIENT vin INGREDIENT blanc cocott escalop
INGREDIENT lucullus cocott tart INGREDIENT oignon INGREDIENT Carr INGREDIENT coq INGREDIENT vin INGREDIENT facil

Figure 3.2.2 Exemple de visualisation de résultat d'évaluation

3.3 Evaluation de reconnaissance d'ingrédients

Nous avons entraîné notre modèle de reconnaissance des ingrédients sur le corpus de « préparation » des recettes (*ingr_train.spacy*). L'évaluation a été faite sur les données test du même corpus (*ingr_valid.spacy*) mais aussi sur les titres de recettes

de cuisine (*ingrTitre_eval.spacy*). Les mesures de précision et de rappel sont fournies dans le chapitre suivant.

3.4 Complexité en temps et corrélation entre le niveau du recette

3.4.1 Calcul de la complexité en temps

Les unités du résultat de la fonction de complexité en temps ne sont pas des unités de temps, mais un nombre d'opérations de base. La formule de ce calcul est :
(nombre d'opération / nombre d'ingrédient) * nombre d'ingrédient * temps moyen pour chaque étape

Ici, nous supposons que le temps moyen pour chaque étape est d'une minute. Il faut donc calculer le nombre d'ingrédients ainsi que le nombre d'opérations.

Pour calculer le nombre d'ingrédients pour chaque recette, nous avons parcouru les fichiers xml de la recette et dans chaque recette, nous avons enregistré tous les éléments "ingredient" dans une liste par la reconnaissance d'entité nommée du spaCy. Nous pouvons signaler qu'il existe deux types de quantité d'ingrédients. Le premier, ce sont les quantités tels que g, kg, cl, ml, etc. Ils sont souvent très grands (ex, 500g), ça va nous empêcher de calculer la complexité. Pour ce genre de quantités, nous traitons comme une unité. S'il y a 500 g de lait dans la recette, nous traiterons une unité de lait. Le deuxième genre de la quantité est tel que boîte, bouteille, etc. Pour cela, nous avons gardé leurs nombres précis.

En plus, parfois, nous pouvons voir le nombre comme "½" qui ne se reconnaît pas par Python. Pour normaliser, nous l'avons transformé en 0.5.

Après avoir eu tous les nombres, nous avons eu le résultat du nombre d'ingrédients par la recette et nous avons enregistré les données dans une liste.

Ensuite, pour le nombre d'opérations, pour chaque recette, nous avons tout d'abord coupé les recettes par virgule et point et les enregistré dans la liste. La figure 3.3.1 est un exemple d'une recette

Figure 3.4.1 Exemple du résultat après avoir coupé une recette

```
['Couper le haut des tomates', 'Enlever la chair des tomates', 'et en réserver les trois-quarts dans une coupelle', 'Mélanger la chair avec les Carrés frais', "l'huile d'olive", 'le sel', 'le poivre', 'et la ciboulette hachée', 'Rajouter le saumon et les crevettes', 'Disposer ce mélange dans les tomates', 'et décorer avec quelques feuilles de ciboulettes', 'Variante : vous pouvez changer les ingrédients et utiliser du surimi ou un mélange avocat/crevettes', 'Mettre au frais 30 min', 'Déguster frais', 'avec quelques feuilles de mâche', 'Miam', 'miam ...', '']
```

Nous avons observé que chaque élément dans la liste, soit il a une entité nommée INGREDIENT, soit il a une entité nommée VERBE, soit il a les deux.

Si c'est le cas d'avoir deux entités nommées, nous supposons que ce VERBE est une opération pour ce INGREDIENT. Par exemple, si la quantité de cet ingrédient est 3 et le verbe est "couper", pour cette partie, les nombres d'opération est $3 * \text{couper}$.

Si l'entité nommée VERBE est toute seule dans l'élément, nous supposons que c'est une opération pour l'ensemble d'ingrédients. Par exemple, ici le verbe tout seul est "salez", les nombres d'opération est $1 * \text{salez}$.

Si l'entité nommée INGREDIENT est toute seule dans l'élément, nous supposons que cet ingrédient traite au moins une fois.

Pour cette étape, nous avons tout d'abord créé trois listes différentes qui contiennent les phrases du recette pour les trois cas différents.

Et puis, nous avons obtenu un dictionnaire, les clés sont les ingrédients, les valeurs sont leurs quantités par l'élément ingrédients dans les fichiers xml.

La figure 3.4.2 est un extrait du dictionnaire.

Figure 3.4.2 Extrait du dictionnaire de l'ingrédients et ses quantités

```
{'sandwich': 8, 'mozzarella': 1, 'saumon': 4, 'Aneth': 1, 'oeuf': 1}
{'citron': 5, 'volaille': 1, 'beurre': 1, 'farine': 8, 'crème': 8, 'persil': 1, 'sel': 1, 'poivre': 1}
{'lapins': 2, 'lait': 2, 'soupe': 1, 'crème': 2, 'thym': 1, 'sel': 1, 'poivre': 1}
{'dés': 1, 'gouda': 4, 'persil': 4, 'sel': 3, 'poivre': 1}
{'souris': 8, 'agneau': 8, 'pruneaux': 4, 'raisins': 1, 'cannelle': 2, 'olive': 1, 'soupe': 1, 'cumin': 1, 'café': 1, 'poivre': 1, 'grains': 1}
{'courgettes': 7, 'sucre': 1, 'vanillé': 2, 'farine': 2, 'lait': 3, 'beurre': 1, 'Pour': 1, 'soupe': 1}
{'raves': 1, 'terre': 1, 'rutabagas': 6, 'chou': 3, 'rave': 4, 'racines': 2, 'persil': 3, 'carottes': 2, 'panais': 1, 'clous': 2, 'girofle': 2, 'échine': 2, 'laurier': 7, 'grains': 1, 'Beurre': 1, 'tournesol': 1}
{'riz': 1, 'risotto': 3, 'olive': 1, 'poivre': 4, 'beurre': 1, 'crème': 2, 'volaille': 1, 'basilic': 1}
{'belles': 4, 'terre': 1, 'provence': 1, 'Luzerne': 1}
{'soles': 3, 'pépins': 2, 'crème': 3, 'poivron': 1, 'céleri': 1, 'thym': 1, 'laurier': 2, 'café': 1, 'cerfeuil': 1, 'beurre': 1, 'Farine': 1}
{'macaronis': 1, 'rosette': 1, 'oeuf': 1, 'sel': 1, 'poivre': 1, '': 1, 'gruyère': 1}
{'farine': 1, 'beurre': 1, 'sucre': 1, 'soupe': 3, 'miel': 1, 'liquide': 2, 'cerneaux': 2, 'café': 1, 'cannelle': 1}
{'lait': 1, 'sucre': 1, 'vanillé': 1, 'farine': 4, 'maizena': 4, 'pâte': 1}
{'viande': 8, 'agneau': 2, 'bourghoul': 3, 'pignons': 1, 'olive': 2, 'beurre': 1, 'sel': 1, 'poivre': 6}
```

Car notre programme de reconnaître les ingrédients n'est pas parfait, nous n'arrivons pas à lier tous les ingrédients avec leurs quantités et nous avons rencontré un problème pour créer un dictionnaire. Nous présenterons ce problème dans la partie de difficulté.

Maintenant, nous avons tous les éléments pour calculer la complexité en temps, et nous avons créé une liste pour sauvegarder chaque complexité.

3.4.2 Corrélation entre le niveau de recette et la complexité

Dans un premier temps, nous avons récupéré tous les niveaux de recette avec le module `xml.dom` et enregistré les niveaux dans une liste. La liste des niveaux et la liste des complexités sont toutes en même ordre, donc leurs indices sont identifiés. Pour visualiser le niveau et la complexité, nous avons mis les deux listes dans un dictionnaire, sa clé est le niveau, sa valeur est le résultat de la complexité.

Chapitre 4

RÉSULTATS

4.1 Résultats

4.1.1 Identification des ingrédients

Remarque préliminaire :

L'automatisation des tâches de prétraitement des textes (*lemmatisation, tokenisation, normalisation,..*) rencontre l'univers humain et les diverses fautes d'orthographies, d'oubli d'espace séparant les termes, etc... par exemple : « moyennepour », « gruyèrepour », « émincépour », « oeuf » pour « œuf », etc... l'automatisation ne pourra pas rectifier ce type d'erreur et cela créera forcément du bruit quant à l'analyse et l'utilisation des données.

L'input data pour cette identification est *ingredient.txt* c'est-à-dire le corpus de la section « ingredients ».

Au préalable une petite étude du corpus chiffrée nous semble importante :

- le nombre de mots dans le corpus avec répétition s'élève à 926444
- le nombre de mots dans le corpus sans répétition s'élève à 5768
- les 10 tokens les plus fréquents dans le corpus sont les suivants:

```
[('de', 126338), ('l', 51462), ('g', 41610), ('à', 27357), ('2', 26856), ('d', 26135),  
('cuiller', 21546), ('soup', 15402), ('sel', 13186), ('4', 11446)]
```

- le nombre de mots ayant une fréquence de répétition inférieure ou égale à 5 dans le corpus est 3723. En conclusion pour le word-embedding : nous avons utilisé une fenêtre minimum de 1 car 5 aurait réduit drastiquement le vocabulaire.

Les premiers résultats qu'il nous semble intéressant de regarder sont les calculs de similarité des mots suite au plongement réalisé par Word2Vec :

```
In [729]: model.wv.most_similar('pain')  
Out[729]:  
[('rass', 0.7995592951774597),  
('campagn', 0.7881822262573242),  
('brinach', 0.6794731616973877),  
('compact', 0.6791952252388),  
('baguet', 0.6735643148422241),  
('rassis', 0.671642784200745),  
('toast', 0.6615371784101562),  
('mi', 0.6592831717300415),  
('viennois', 0.641882598480116),  
('écrout', 0.6327947378158569)]
```

```
In [732]: model.wv.most_similar('farin')  
Out[732]:  
[('polent', 0.6822400093078613),  
('maiz'en', 0.6610264778137207),  
('tamis', 0.6509989484678345),  
('froment', 0.652597963889967),  
('t', 0.639442503452301),  
('t55', 0.635487618293762),  
('semoul', 0.6222544312477112),  
('fécul', 0.6175669597625732),  
('t45', 0.5858898758888245),  
('tapio', 0.5794791579246521)]
```

```
In [738]: model.wv.most_similar('eau')  
Out[738]:  
[('21*c', 0.7033263444980513),  
('chaud', 0.6857132315635681),  
('tied', 0.6715463399887085),  
('35*c', 0.644341766834259),  
('bouill', 0.6366538656814575),  
('dison', 0.6270766854286194),  
('gazeux', 0.6248333318710327),  
('20em', 0.6181488633155823),  
('addition', 0.6076226830482483),  
('petr', 0.5863208174785585)]
```

```

In [742]: model.wv.most_similar('saumon')
Out[742]:
[('truite', 0.7467071413993835),
 ('hareng', 0.6858268244743347),
 ('flétan', 0.678666353225708),
 ('églefin', 0.65255809357452393),
 ('haddock', 0.6471438659294128),
 ('kangourou', 0.6416419744491577),
 ('maquereau', 0.6394262579421997),
 ('esturgeon', 0.63274880774353),
 ('pav', 0.6284867525100708),
 ('cabillaud', 0.6267490983800338)]
```

```

In [746]: model.wv.most_similar('sel')
Out[746]:
[('poivre', 0.6667030453681946),
 ('cayennet', 0.621978193782698),
 ('mignonnet', 0.5948704914749573),
 ('guérard', 0.5739715899334717),
 ('pouvr', 0.5675575733184814),
 ('muscad', 0.560724675655365),
 ('poivrel', 0.5598229169845581),
 ('moulin', 0.559339702129364),
 ('2citron', 0.5518892480483687),
 ('brindill', 0.5517013669013977)]
```

```

In [748]: model.wv.most_similar('vern')
Out[748]:
[('sel', 0.6540687680244446),
 ('dl', 0.6438122256278992),
 ('bouchon', 0.6225137114524241),
 ('louch', 0.6057952046394348),
 ('mug', 0.60165935754776),
 ('tabl', 0.597412800788879),
 ('whisky', 0.585345983505749),
 ('rasad', 0.5832783579826355),
 ('v', 0.5884946422576984),
 ('charent', 0.57979816198349)]
```

Figure 4.1 Exemple de résultats de calcul de similarité

Lorsque l'on regarde de plus près on relève quelques erreurs mais globalement on trouve une classification assez bonne : saumon par exemple est rapproché à truite, hareng, maquereau, cabillaud... Farine est même rapproché à T45. Pour l'eau par contre, on constate bien que l'ingrédient n'est pas forcément lié à un ingrédient mais aussi à tout ce qui en est proche ou qui apparaît souvent pour l'eau des indicateurs tels que la température, la quantité, le type (gazeuse/plate) apparaissent dans ce calcul de similarité.

L'étape suivante a consisté à essayer de mettre en évidence des clusters de type de phrases (des patterns en quelques sortes).

Au préalable, pour choisir le bon nombre de clusters, nous avons utilisé la technique « Elbow Method » qui consiste à tracer l'évolution du coût de notre modèle en fonction du nombre de clusters. Il s'agit de détecter dans ce graphique une zone de « coude ». Cette zone (théoriquement) nous indique le nombre de clusters optimal c'est-à-dire celui qui nous permet de réduire au maximum le coût de notre modèle tout en conservant un nombre raisonnable de clusters.

Nous avons défini la rangée de valeurs que nous voulions tester (de 1 à 20 puis nous sommes montées par curiosité jusqu'à 80). Nous avons alors entraîné le modèle KMeans sur nos données X pour chaque k dans [1,20] et [1,80]. Le coût (inertia_) a été conservé dans une liste pour affichage du coût en fonction du k-range.

Nous obtenons les graphes ci-dessous : il est finalement difficile de déterminer une valeur pour k. Augmenter le nombre de k n'aurait pas de sens non plus car nous augmenterions aussi le nombre de clusters. Nous avons choisi de prendre 7 clusters suite aux résultats affichés dans [1].

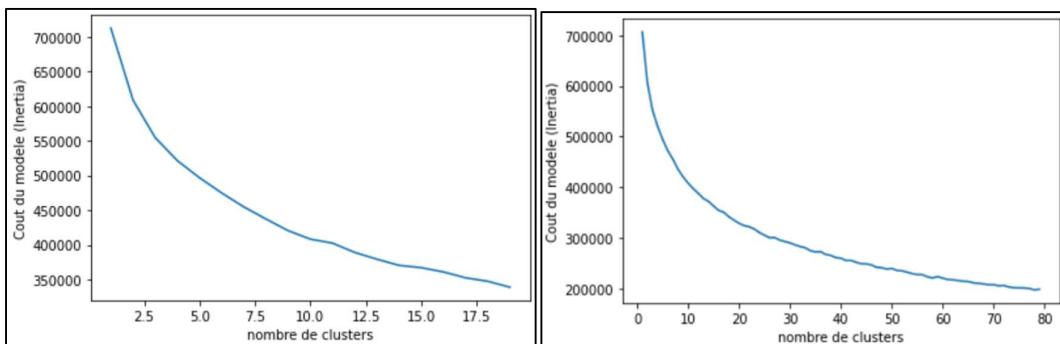


Figure 4.2 Coût du modèle K-Means en fonction du nombre de clusters
L'étape suivante a consisté à entraîner le modèle K-Means avec K=7.

Le score obtenu est le suivant: -454933.4375 (*Score : Opposite of the value of X on the K-means objective which is Sum of distances of samples to their closest cluster center*). Cette valeur nous semble difficile à expliquer.

Le coefficient de silhouette, *Silhouette_score*, obtenu vaut 0.15875801. Ce coefficient est une mesure de qualité d'une partition d'un ensemble de données. Il varie entre -1 (pire classification) et 1 (meilleure classification). Dans notre cas le score n'est pas mauvais mais ce n'est pas non plus un bon score : il pourrait certainement être amélioré suite à des expériences.

Le clustering donne le graphe suivant :

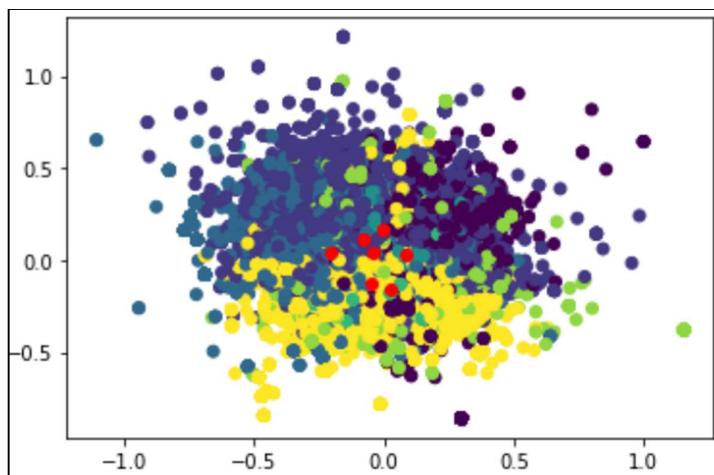


Figure 4.3 Clusterisation du modèle K-Means (K=7)

En rouge nous visualisons les centroïdes de chaque cluster : ils sont proches. Le graphe est difficilement interprétable, nous utilisons donc une approche d'analyse en composantes principales (PCA) pour projeter dans un espace 2D.

On passe ainsi d'un X.shape (189971, 100) à un X.shape (189971, 2).

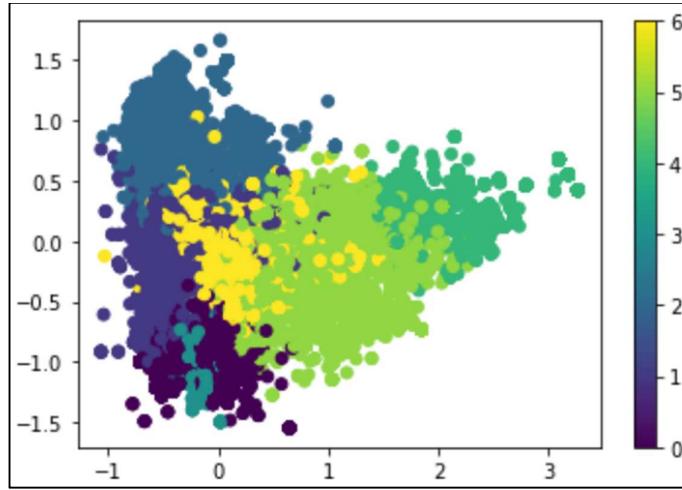


Figure 4.4 Visualisation des clusters par PCA

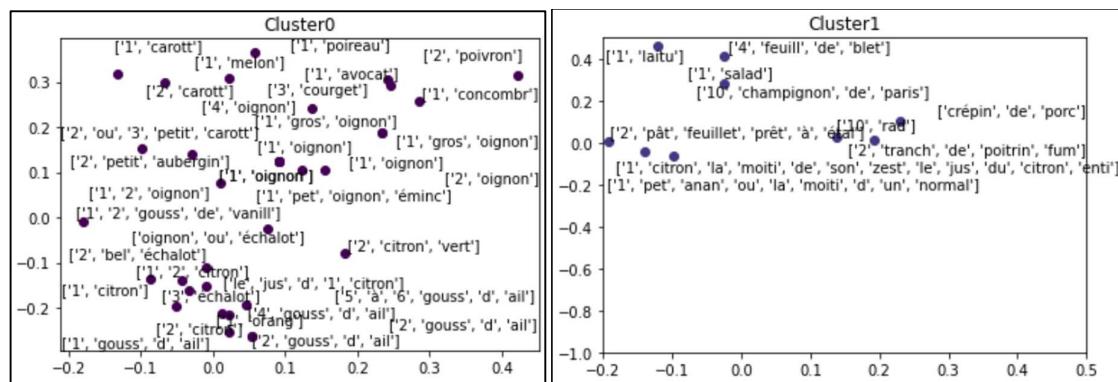
Si l'on regarde les résultats de variances des composantes on obtient :

`model_pca.explained_variance_ratio_ array([0.17714245, 0.10542039])`

Soit une somme sur les deux composantes qui donne à peu près 28%, ce qui est faible.

Une fois la clusterisation effectuée nous nous sommes intéressées à ces clusters en les analysant afin de mettre en relief des patterns et d'en extraire le plus aisément possible nos ingrédients.

Ci-dessous quelques échantillons de clusters : des tendances se dégagent clairement sauf pour le cluster1. Il y a même une sorte de spécialisation des clusters.



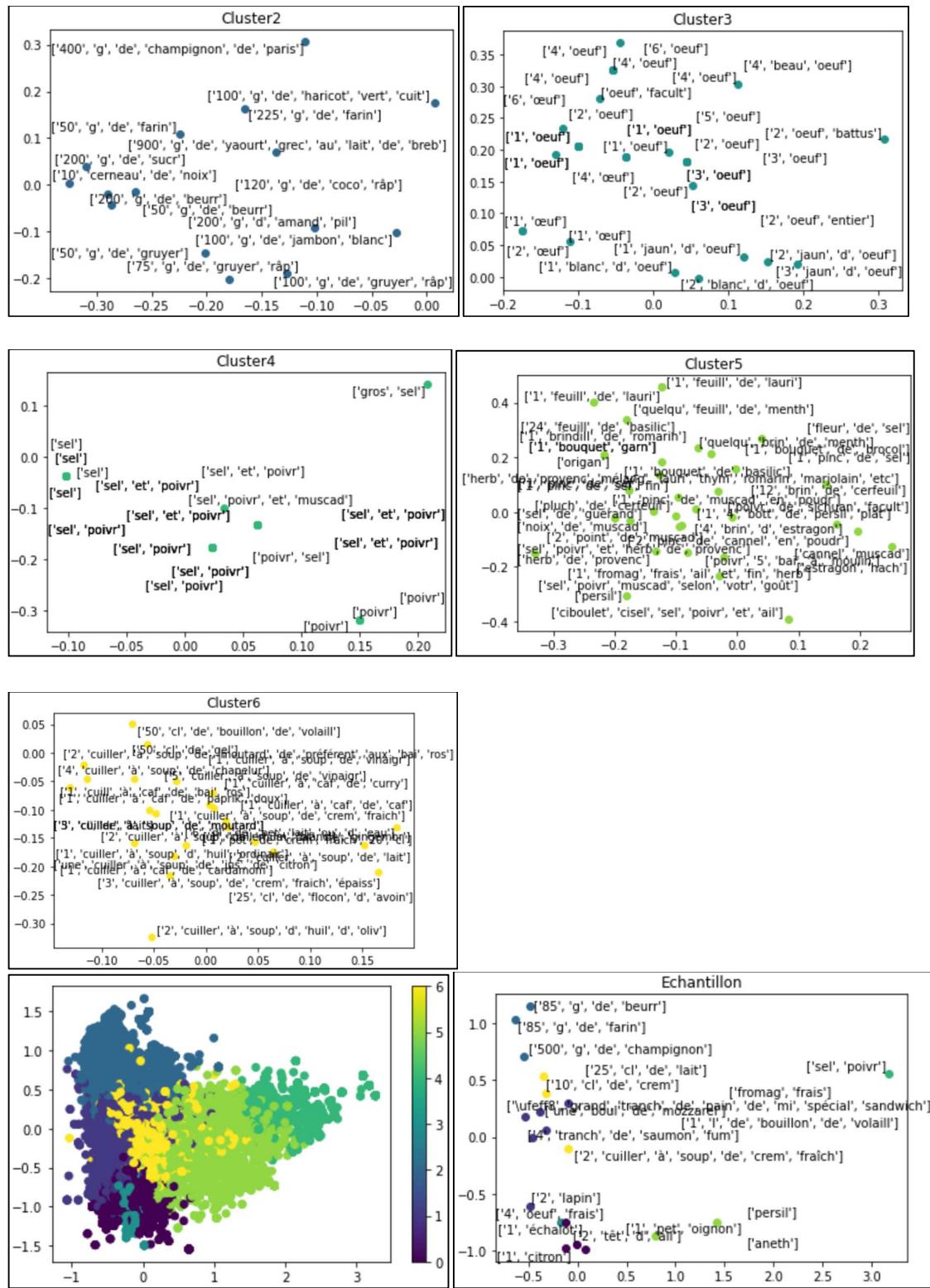


Figure 4.5 Visualisation d'échantillons de phrases par cluster

Nous avons ainsi poussé l'analyse de ces clusters pour sélectionner des textes représentatifs qui permettent une extraction simplifiée des ingrédients. Nous répondons donc à quelques questions pour aller dans ce sens.

Tout d'abord quelle est la dimension de chaque cluster ?

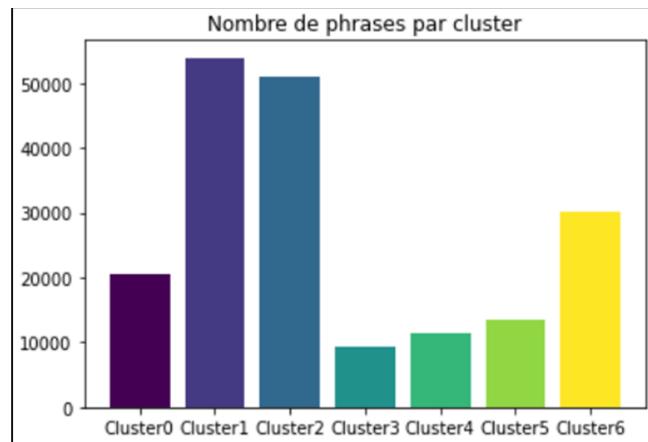
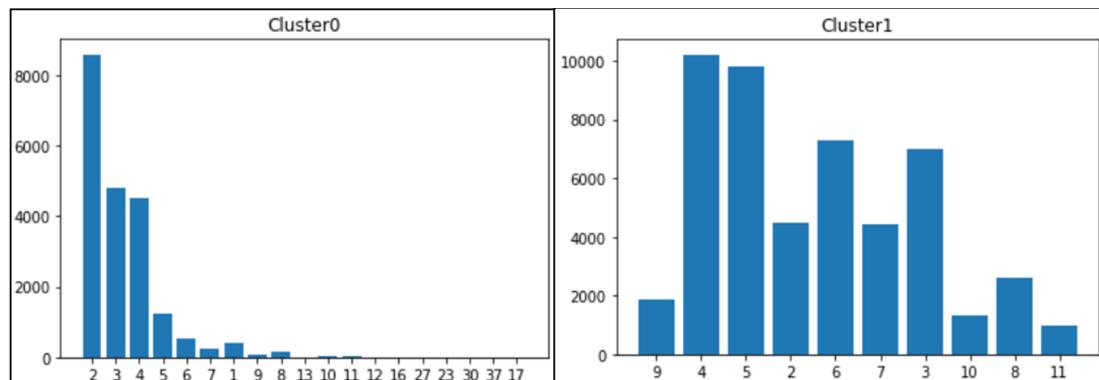


Figure 4.6 Nombre de phrases par cluster

Est-ce que chaque Cluster est composé de phrases de la même taille (même nombre de tokens) ? On présente ici quelques résultats dans les graphes suivants : il y a selon les clusters différentes syntaxes des phrases.



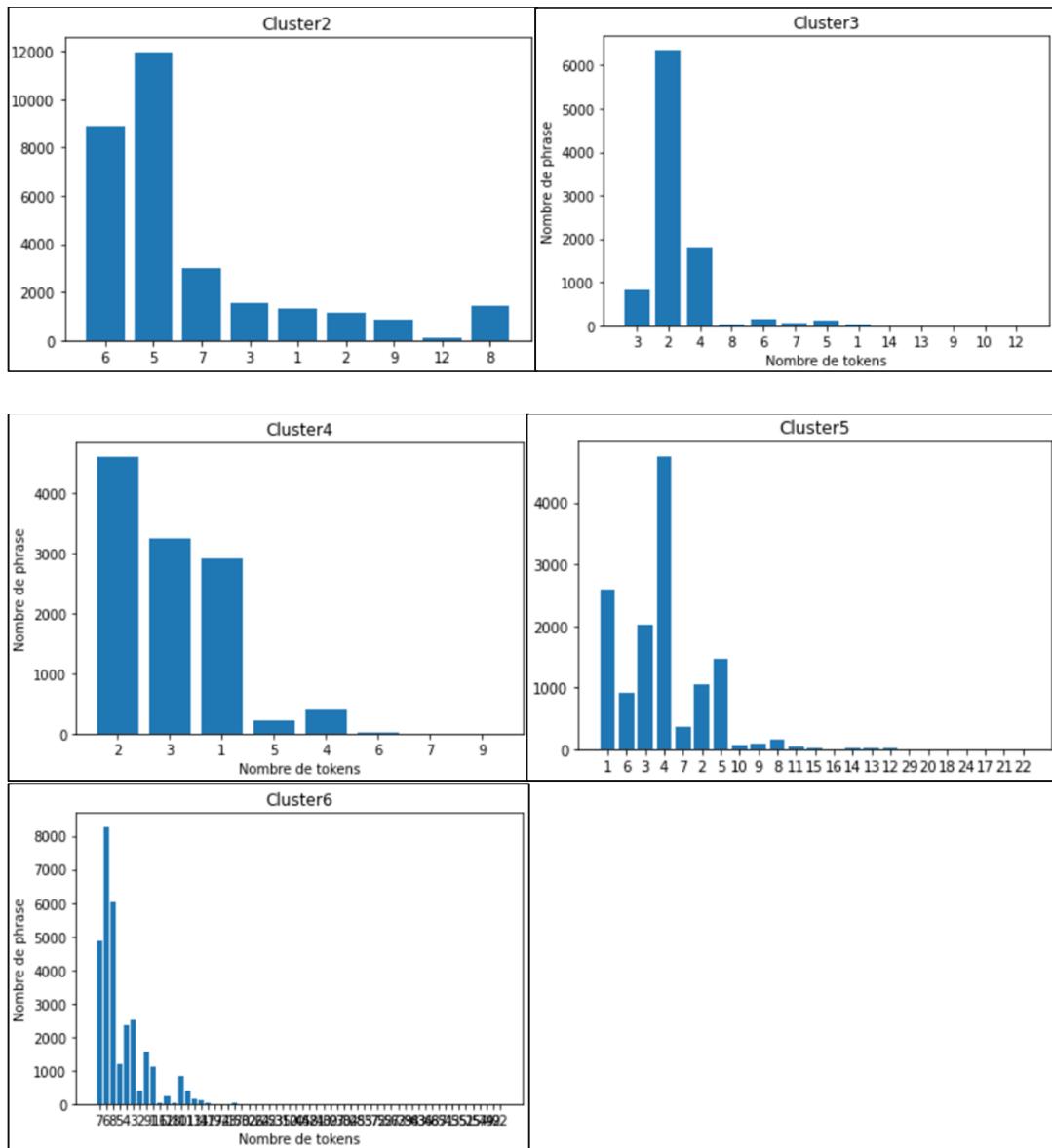


Figure 4.6 Nombre de phrases en fonction de son nombre de tokens par cluster

On ne présente qu'une partie des phrases pour le cluster 1 et 2 (celles les plus nombreuses).

Comment avons-nous extrait les ingrédients ?

Prenons l'exemple du cluster 0. Si l'on regarde les phrases composées de :

- 2 tokens : elles sont sous la forme : ['1', 'oignon'], ['oignon', 'blanc'], ['1', 'carott'], ['ail', 'hach'].
- 3 tokens : elles suivent le schéma suivant : « NUM NOM » ou « NOM ADJ » ou « ADJ NOM » ['1', 'gros', 'oignon'], ['1', 'poivron', 'roug'], ['2', 'citron', 'confit']..
- 4 tokens : elles suivent le schéma suivant : ['1', 'gouss', 'd', 'ail'], ['1', 'gouss', 'de', 'vanill'], ...

Un parcours des clusters et de ses sous-ensembles permet à partir de règle simple d'extraire les ingrédients à condition d'enrichir une liste qui exclu des tokens (en particulier les token, les adjectifs, ...).

Au final nous avons extrait une liste de 1127 ingrédients entre automatisme et intervention humaine manuelle. Nous avons fait le choix de réduire le périmètre de prospection en fonction de la fréquence de chaque sous-ensemble. Nous n'excluons pas la possibilité que la liste ne soit pas 100% correcte cependant elle est suffisamment propre et importante pour éviter une annotation manuelle du corpus (*listIngredientsToSave.csv*).

4.1.2 Création des data de train et de test

L'étape suivante est la constitution d'un set de train et de test. Le code suivant *fromDataClusterToDataSpacy.ipynb* permet de créer ces données dans le format souhaité par spaCy.

Ce code prétraite le texte des recettes et des titres, crée l'entité INGREDIENT qui prend en entrée la liste des ingrédients extraits par le clustering et génère les fichiers JSON pour entraînement et évaluation.

4.1.3 Entraînement du modèle

Les data sous le format JSON correspondent à la version 2 de spaCy or la version 3 connaît une mise à jour importante qui ne permet plus de lancer les calculs d'entraînement de la même manière. Aussi cette version demande un fichier de configuration, des données binaires en entrée et une interaction sous forme de ligne de commandes. Le fichier suivant *nerIngredientsSpacy3.ipynb* permet cette conversion grâce à la méthode *create_training*.

Un fichier de configuration dit « de base » (*base_config.cfg*) se crée à partir d'une sélection que l'on effectue sur le site de spaCy(<https://spacy.io/usage/training#config>), il est ensuite légèrement adapté pour prendre en compte les chemins de nos données binaires. Il faut alors interagir avec la console de commandes pour générer un nouveau fichier de configuration (*config.cfg*) mais aussi pour entraîner. Les commandes sont fournies sur le site. A noter qu'il est possible qu'il faille changer *spacy.MultiHashEmbed.V2* en *.V1* selon les versions. L'optimiseur par défaut est Adam : nous l'avons conservé.

La dimension de nos data : nous avons créé deux ensembles train et test à partir du corpus « preparation » suivant la proportion 80/20 soit : 144443 pour le train et 35568 pour le test. Nous avons ensuite créé un autre set d'évaluation qui prend tous les titres de recette soit 21648 (certaines recettes n'ayant pas de titre).

Le temps de calcul a été assez important : un peu plus d'une heure trente.

Les résultats suite au calcul sont fournis dans la Figure 4.7. On relève une très bonne convergence et un très bon score.

```
Using CPU
=====
[2021-05-21 13:23:34,221] [INFO] Set up nlp object from config
[2021-05-21 13:23:34,261] [INFO] Pipeline: ['tok2vec', 'ner']
[2021-05-21 13:23:34,278] [INFO] Created vocabulary
[2021-05-21 13:23:34,279] [INFO] Finished initializing nlp object
[2021-05-21 13:27:27,622] [INFO] Initialized pipeline components: ['tok2vec', 'ner']

Initialized pipeline
=====
Training pipeline
=====
Pipeline: ['tok2vec', 'ner']
Initial learn rate: 0.001
E   #      LOSS TOK2VEC LOSS NER ENTS_F ENTS_P ENTS_R SCORE
---- -----
0    0      0.00    75.06  0.00   0.00   0.00   0.00   0.00
0    200    56.81   3429.74 96.91  95.90  97.94  0.97
0    400    30.62   706.19  98.63  99.06  98.21  0.99
0    600    27.40   522.98  99.15  99.36  98.95  0.99
0    800    34.25   411.97  99.41  99.45  99.37  0.99
0   1000    45.16   395.95  99.53  99.51  99.55  1.00
0   1200    72.21   378.63  99.64  99.70  99.59  1.00
0   1400   116.80   354.18  99.73  99.70  99.77  1.00
0   1600   174.10   363.20  99.80  99.73  99.87  1.00
0   1800   218.20   296.99  99.84  99.80  99.89  1.00
0   2000   267.69   307.88  99.88  99.91  99.86  1.00
0   2200   319.55   322.82  99.91  99.90  99.92  1.00
1   2400   344.05   258.98  99.90  99.85  99.95  1.00
1   2600   309.29   117.66  99.92  99.88  99.96  1.00
1   2800   403.11   143.18  99.94  99.92  99.95  1.00
1   3000   358.35   89.88  99.92  99.88  99.95  1.00
1   3200   434.56   109.84  99.93  99.92  99.94  1.00
2   3400   358.19   83.29  99.92  99.88  99.97  1.00
2   3600   289.53   55.13  99.93  99.89  99.97  1.00
2   3800   327.99   54.57  99.94  99.91  99.97  1.00
2   4000   293.31   55.10  99.93  99.88  99.97  1.00
2   4200   293.77   50.50  99.94  99.91  99.97  1.00
3   4400   277.23   57.35  99.91  99.85  99.97  1.00
3   4600   186.23   32.95  99.92  99.85  99.98  1.00
3   4800   309.22   45.03  99.92  99.87  99.98  1.00
3   5000   278.34   47.86  99.93  99.88  99.98  1.00
3   5200   316.65   30.18  99.93  99.88  99.98  1.00
4   5400   322.33   29.68  99.93  99.88  99.98  1.00
Saved pipeline to output directory
output\model-best
```

Figure 4.7 Résultats en termes de score, rappel et précision de l'entraînement du modèle de reconnaissance des ingrédients.

4.1.4 Evaluation du modèle de reconnaissance des ingrédients sur les titres de recettes

Lors de l'entraînement différents modèles sont enregistrés dans le répertoire output en particulier le meilleur modèle rencontré pendant le process d'entraînement mais aussi le dernier. On trouve également les composantes NER, tok2vec. Lors de l'évaluation, on charge un modèle, ici le model-best cf. Figure 4.8.

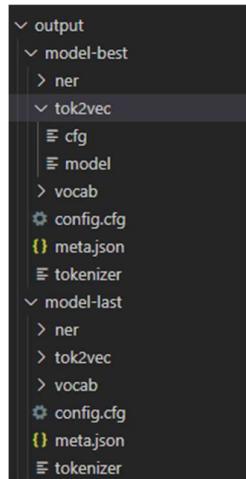


Figure 4.8 Enregistrement des modèles lors de l'entraînement.

La performance de notre modèle de reconnaissance d'ingrédients a été testé suite à une évaluation des résultats d'identification des ingrédients sur les titres de recette. La Figure 4.9 donne les mesures de précision et de rappel obtenues. Les Figures 4.10 et 4.11 fournissent une visualisation des identifications.
(Il s'agit ici encore du fichier *nerIngredientsSpacy3.ipynb*.)

```
Using CPU
=====
Results =====
TOK    100.00
NER P  99.79
NER R  99.89
NER F  99.84
SPEED  5205

===== NER (per type) =====
          P      R      F
INGREDIENT 99.79  99.89  99.84

Saved results to --displacy-path
```

Figure 4.9 Résultats en termes de score, rappel et précision de évaluation du modèle de reconnaissance des ingrédients sur les titres des recettes.

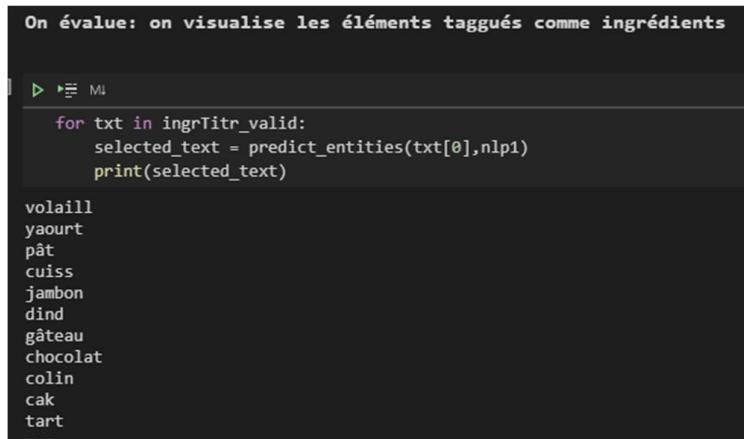
```
# load the best model
nlp1 = spacy.load("output\model-best")

#doc=nlp1('lasagn végétarien facil terrin foi porc lapin vin blanc cocott escalop lucullus cocott tart oignon carr coq vin facil')
spacy.displacy.render(doc, style="ent", jupyter=True)
```

The output shows the rendered entities for the sentence "lasagn végétarien facil terrin foi porc lapin vin blanc cocott escalop lucullus cocott tart oignon carr coq vin facil". Entities are highlighted in boxes:

- lasagn (INGREDIENT)
- végétarien (INGREDIENT)
- facil (INGREDIENT)
- terrin (INGREDIENT)
- foi (INGREDIENT)
- porc (INGREDIENT)
- lapin (INGREDIENT)
- vin (INGREDIENT)
- blanc (INGREDIENT)
- cocott (INGREDIENT)
- escalop (INGREDIENT)
- lucullus (INGREDIENT)
- cocott (INGREDIENT)
- tart (INGREDIENT)
- oignon (INGREDIENT)
- carr (INGREDIENT)
- coq (INGREDIENT)
- vin (INGREDIENT)
- facil (INGREDIENT)

Figure 4.10 Visualisation d'une évaluation sur quelques titres de recette.



```
On évalue: on visualise les éléments taggués comme ingrédients

for txt in ingrTitr_valid:
    selected_text = predict_entities(txt[0],nlp1)
    print(selected_text)

volaill
yaourt
pât
cuiss
jambon
dind
gâteau
chocolat
colin
cak
tart
```

Figure 4.11 Visualisation des extractions/prédictions sur le corpus de titre.

4.2.1 Identification des techniques/process culinaires

Une étude similaire avait été entamée sur le corpus des « preparation » mais à l'échelle des mots. Elle n'a pas été finalisée par manque de temps. Nous présentons les quelques résultats obtenus.

Il s'agit du fichier *nerWord2PreparationWordLevel.py*.

L'input data est *recettesParPhrases2.txt* c'est-à-dire le corpus de la section « preparation ». Il a été généré par le fichier *fromDataClusterToDataSpacy.ipynb* qui a produit les data d'entraînement sur ce corpus. Le texte est prétraité (les stopwords et une liste d'adjectifs et adverbes non exhaustives a été utilisée *adjectifCulinnaire.txt* et *stopwords-fr.json*).

Au préalable une petite étude du corpus chiffrée nous semble importante :

- le nombre de mots dans le corpus avec répétition s'élève à 1 334 969
- le nombre de mots dans le corpus sans répétition s'élève à 15574
- les 10 tokens les plus fréquents dans le corpus sont les suivants:

```
[('ajout', 30505), ('fair', 26421), ('mélang', 23111), ('coup', 17064), ('cuir', 16553), ('four', 16165), ('mettr', 14558), ('pât', 14244), ('min', 13242), ('beurr', 13092),
```

- Nous avons fait le choix pour une première analyse de réduire drastiquement le vocabulaire en utilisant une fenêtre de 10 rendant le format de X (3377, 100).

Le plongement réalisé par Word2Vec était de type BOW (Bag Of Word). L'étape suivante a consisté à entraîner le modèle K-Means avec K=3 en supposant dans le meilleur des mondes que les clusters répartiraient les ustenciles, les ingrédients et les actions.

Le score obtenu est le suivant : -92497.296875 (*Score : Opposite of the value of X on the K-means objective which is Sum of distances of samples to their closest cluster center*). Cette valeur nous semble difficile à expliquer.

Le coefficient de silhouette, *Silhouette_score*, obtenu vaut quant à lui 0.20009983. Dans notre cas le score n'est pas mauvais mais ce n'est pas non plus un bon score : il pourrait certainement être amélioré suite à des expériences.

Le clustering donne le graphe suivant :

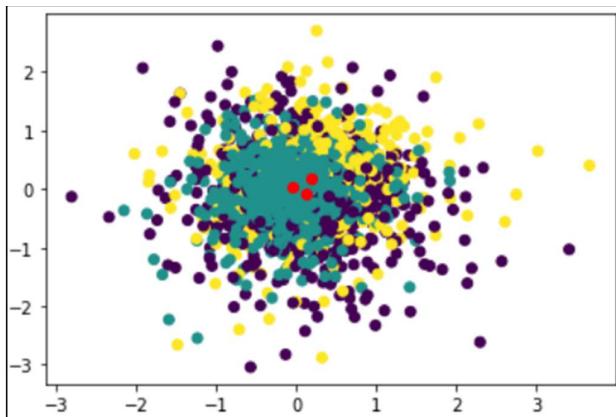


Figure 4.12 Clusterisation du modèle K-Means (K=3)

En rouge nous visualisons les centroïdes de chaque cluster : ils sont proches. Le graphe est difficilement interprétable, nous utilisons donc une approche T-SNE et une approche d'analyse en composantes principales (PCA) pour projeter dans un espace 2D.

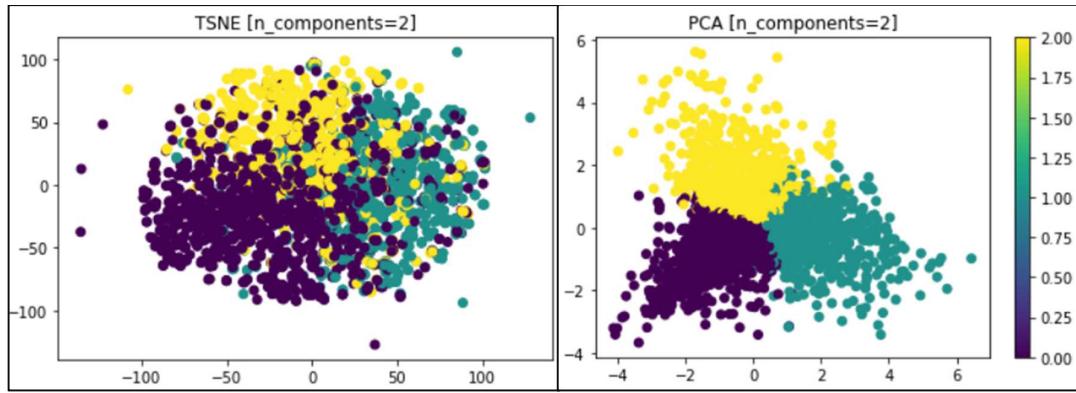


Figure 4.13 Visualisation des clusters (T-SNE et PCA)

Si l'on regarde les résultats de variances des composantes on obtient :
`model_pca.explained_variance_ratio_ array([0.05702673, 0.04489018])`
 Soit une somme sur les deux composantes qui donne à peu près 10%, ce qui est faible.

Une fois la clusterisation effectuée nous nous sommes intéressées à ces clusters en essayant d'analyser de plus près. Nous présentons un exemple d'échantillon de données :

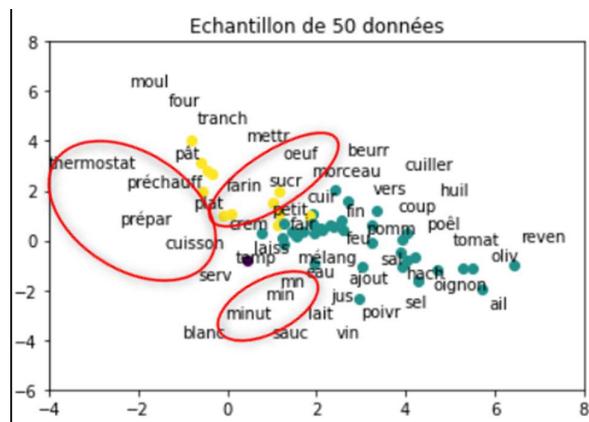


Figure 4.13 Visualisation de quelques tokens

A première vue, il semble un peu compliqué d'extraire automatiquement des tendances/liens pour automatiser cette tâche à partir des calculs obtenus. En effet, bien que l'embedding rapproche parfois les verbes ou ustenciles ou ingrédients ou même unités, il reste cependant beaucoup trop de mélange. L'embedding à ce stade à l'échelle des mots n'est pas concluant. Peut-être faudrait-il jouer sur les valeurs

fournies par le embedding mais aussi faire une clusterisation à l'échelle de la phrase comme pour les ingrédients. C'est ce que propose à peu près [1].

Nous avons malgré tout tenté d'entrainer un modèle en prenant en considération des verbes, des ustensiles, des ingrédients et des quantités que nous avions créé à partir de spaCy, de notre résultat d'entraînement mais aussi à partir du web. Les résultats de convergence sont donnés par la Figure 4.14. Son évaluation est donnée par les Figures 4.15 et 4.16. Une visualisation des résultats d'évaluation est donnée par les Figures 4.17 et 4.18. Le modèle semble détecté la majorité des entités nommées créées mais on constate quelques défauts : *huile arachide* étant compté comme deux ingrédients par exemple.

===== Training pipeline =====							
Pipeline: ['tok2vec', 'ner']							
Initial learn rate: 0.001							
E	#	LOSS TOK2VEC	LOSS NER	ENTS_F	ENTS_P	ENTS_R	SCORE
0	0	0.00	75.08	0.02	8.15	0.01	0.00
0	200	213.47	6498.78	92.74	92.29	93.21	0.93
0	400	199.10	2131.11	94.43	94.87	93.98	0.94
0	600	189.63	2186.06	94.93	94.93	94.94	0.95
0	800	176.30	2149.54	95.43	95.43	95.42	0.95
0	1000	184.72	2433.20	95.71	95.65	95.77	0.96
0	1200	182.81	2633.79	95.95	95.90	96.00	0.96
...							
7	10200	705.09	3900.78	97.65	97.58	97.72	0.98
7	10400	656.66	3859.07	97.64	97.63	97.64	0.98
8	10600	654.18	3938.94	97.49	97.50	97.48	0.97
8	10800	723.79	3693.13	97.57	97.59	97.55	0.98
8	11000	738.48	3817.08	97.59	97.63	97.56	0.98
8	11200	805.78	3820.33	97.61	97.55	97.67	0.98
8	11400	750.01	3797.12	97.62	97.61	97.64	0.98

Figure 4.14 Entrainement d'un modèle de reconnaissance des 4 EN

===== Results =====							
TOK	100.00						
NER P	99.39						
NER R	99.61						
NER F	99.50						
SPEED	8131						
===== NER (per type) =====							
		P	R	F			
INGREDIENT	99.52	99.91	99.72				
USTENSILE	98.45	99.86	99.15				
ACTION	98.50	97.86	98.18				
UNIT_QUANTITY	94.72	72.97	82.43				
MISC	0.00	0.00	0.00				

Figure 4.15 Mesures d'évaluation du modèle NER sur les titres des recettes

Results			
TOK	100.00		
NER P	97.70		
NER R	97.73		
NER F	97.71		
SPEED	9894		
NER (per type)			
	P	R	F
ACTION	98.87	93.16	95.93
USTENSILE	95.21	99.15	97.14
UNIT_QUANTITY	98.01	97.57	97.79
INGREDIENT	97.77	99.34	98.55
QUANTITY	100.00	100.00	100.00
MISC	91.67	55.00	68.75
LOC	0.00	0.00	0.00
PER	0.00	0.00	0.00
ORG	0.00	0.00	0.00

Figure 4.16 Mesures d'évaluation du modèle NER sur les préparations des recettes

```

> ▶ M4
doc=nlp1('lasagn végétarien facil terrin foi porc lapin vin blanc cocott escalop lucullus cocott tart oignon carr coq vin facil ')
> ▶ M4
spacy.displacy.render(doc, style="ent", jupyter=True)
[lasagn INGREDIENT végétarien ACTION facil ACTION terrin USTENSILE foi INGREDIENT porc INGREDIENT lapin INGREDIENT vin INGREDIENT blanc USTENSILE cocott INGREDIENT escalop INGREDIENT lucullus USTENSILE cocott USTENSILE tart INGREDIENT oignon INGREDIENT carr INGREDIENT coq INGREDIENT vin INGREDIENT facil ACTION]
couvert

```



```

> ▶ M4
doc=nlp1('coup viand porc dé fair reven mijot cocott feu huil arachid couvert')
spacy.displacy.render(doc, style="ent", jupyter=True)
[coup ACTION viand INGREDIENT porc INGREDIENT dé ACTION fair ACTION reven ACTION mijot ACTION cocott USTENSILE feu INGREDIENT huil INGREDIENT arachid INGREDIENT couvert ACTION]

```

Figure 4.17 Mesures d'évaluation du modèle sur les titres et les recettes

```

> ▶ M4
for txt in nerTitr_valid:
    selected_text = predict_entities(txt[0],nlp1)
    print(selected_text)

chinois
porc
saindoux
quich
asiat
magret
brioch
tart
noix
frais
tart
pomm
mouss

```



```

> ▶ M4
for txt in ner_test:
    selected_text = predict_entities(txt[0],nlp1)
    print(selected_text)

1
lav
coup
2
3
réserv
reven
sal
reven
color
viand
serv
plat

```

Figure 4.18 Mesures d'évaluation du modèle sur les titres et les recettes

4.3 Résultat du calcul de la complexité en temps et la corrélation entre le niveau de recettes et la complexité

Ici, nous avons deux versions du calcul complexité et sa corrélation entre le niveau du recette et la complexité (*AjoutEntiteNommee+CalculComplexite.ipynb*).

Les différences entre deux versions sont les fichiers de la liste d'ingrédients et de la liste de verbes.

Dans la première version, nous avons utilisé le fichier *IngredientParSpacy.txt* et *verbesCuisineSurInternet.txt* pour les entités nommées "INGREDIENT" et "VERBE".

Figure 4.1 Extrait du résultat de la corrélation de la première version

print(dict_correlation) #le résultat	
{'Facile': 26.0}	
{'Facile': 39.0}	
{'Facile': 18.0}	
{'Très facile': 17.0}	
{'Facile': 33.0}	
{'Très facile': 26.0}	
{'Facile': 78.0}	
{'Facile': 17.0}	
{'Facile': 9.0}	
{'Moyennement difficile': 22.0}	
{'Très facile': 12.0}	
{'Facile': 13.0}	
{'Très facile': 19.0}	
{'Facile': 52.0}	
{'Très facile': 17.0}	
{'Très facile': 29.000000000000004}	
{'Facile': 25.0}	
{'Facile': 20.0}	
{'Très facile': 12.0}	

Dans la deuxième version, nous avons utilisé le fichier *listIngredientsToSave.csv* et *verbesSpacyCorrect.txt* pour les entités nommées "INGREDIENT" et « VERBE ».

Figure 4.1 Extrait du résultat de la corrélation de la deuxième version

```

print(dict_correlation) #le résultat
{'Facile': 17.0}
{'Facile': 22.0}
{'Facile': 15.0}
{'Très facile': 9.0}
{'Facile': 27.0}
{'Très facile': 10.0}
{'Facile': 30.0}
{'Facile': 23.0}
{'Facile': 11.0}
{'Moyennement difficile': 18.0}
{'Très facile': 9.0}
{'Facile': 12.0}
{'Très facile': 6.0}
{'Facile': 18.0}
{'Très facile': 25.0}
{'Très facile': 10.0}
{'Facile': 12.0}
{'Facile': 8.0}
{'Très facile': 0.0}

```

Les deux résultats ne sont pas pertinents. Nous présenterons les difficultés pour cette partie dans la section Difficultés. Ici, nous avons imprimé des résultats juste pour montrer que le programme marche.

4.2 Difficultés et conclusion

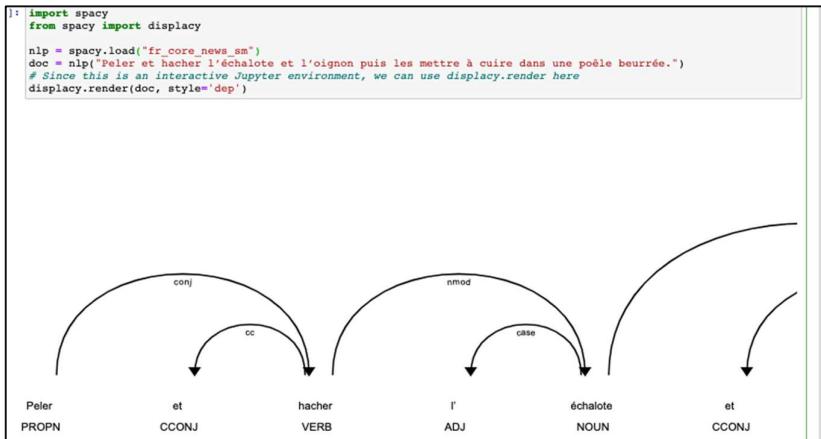
4.2.1 Identification des entités

Cette partie a finalement été traité au fil de l'eau dans les chapitres/sections précédents.

4.2.2 Calcul de la complexité en temps et de l'espace

Pour calculer la complexité en temps, il faut savoir le nombre d'ingrédients et le nombre d'opérations. Le premier est plus simple. Nous pouvons retrouver les nombres exacts dans les listes d'ingrédients mentionnées par les fichiers xml. Par contre, pour le nombre d'opérations, il faut tout d'abord lier les listes d'ingrédients et les textes de recettes afin de reprendre les nombre d'ingrédients. Et puis, il faut aussi lier les verbes culinaires avec les ingrédients. Par exemple, "Coupez les lapins en morceaux (1 morceau par convive), huilez-les et saupoudrez-les de sel, de poivre et de thym." Ici, nous devons savoir combien de lapins dans les ingrédients, et aussi, nous devons connecter "couper -> lapins", "couper -> lapins", "saupoudrez avec sel, poivre, thym". La première étape, nous pouvons résoudre par lire les fichiers xml, pourtant, la deuxième étape, il faut trouver la relation dépendante entre l'ingrédient et le verbe. Nous avons essayé de visualiser la relation dépendante avec spaCy, mais le problème du spaCy en français, c'est qu'il y a trop d'erreurs. Il n'arrive pas à identifier tous les postaggings correctement (par exemple, il a toujours traité les adjectifs comme les noms) et le postagging est la base de la relation dépendante. Nous avons visualisé quelques textes de recettes en dépendance, et le résultat était très insatisfaisant.

Figure 4.2.2.1 Résultat de la dépendance



Nous avons bloqué ici, et finalement, nous n'avons pas trouvé une solution pour résoudre ce problème et nous n'avons pas eu le choix, mais faire un calcul de la complexité en temps subjectif que nous avons déjà représenté dans le chapitre 3. Néanmoins, pendant ce processus, nous avons aussi rencontré un problème. Afin de calculer le nombre d'opérations, nous avons créé un dictionnaire, sa clé est l'ingrédients et sa valeur est les quantités. Avant de créer ce dictionnaire, les clés et les valeurs sont deux listes séparées. Nous voulions utiliser la méthode zip pour bien transmettre les deux listes en un dictionnaire. Pour cela, la longueur de deux listes doit être identifiée. À cause du mal reconnaître tous les ingrédients, malheureusement, les longueurs n'étaient pas pareilles.

Pour continuer le programme, nous avons supprimé les derniers éléments de la liste pour avoir les mêmes longueurs.

La figure 4.2.2.2 montre ce processus.

Figure 4.2.2.2 Extrait du programme

```
if len(num) > len(textes):
    num = num[0:len(textes)]
if len(num) < len(textes):
    textes = textes[0:len(num)]
```

Normalement, ces 4 lignes n'affichent pas dans le programme. Ici, on ajoute les 4 lignes juste pour montrer que le reste du programme du calcul de la complexité marche bien et si nous reconnaissons tous les ingrédients, nous n'avons pas besoin de ce processus.

Dans `fromDataClusterAndNetToDataSpacy`, il est proposé une expérience intéressante qui tente de lier les INGREDIENT aux ACTION à partir des syntagmes nominaux et des tags : pour certains ingrédients le résultat est pertinent pour la majorité ce n'est pas le cas. Nous n'avons pas eu le temps d'aller plus loin mais il faudrait creuser cette piste en y filtrant à partir des EN que nous réussissons maintenant à détecter.

```

'sapin': ['Ajouter'],
'vin': ['Verser', 'Ajouter'],
'oignon': ['Coupez',
'Coupez',
'finement',
'Mettre',
'revenir',
'Emincer',
'hacher',
'Pelez',
'mettez'],

```

```

dicIngrVerb = {}
for sent in sents:
    for entity in sent.ents:
        print(entity.text,entity.start_char, entity.end_char,entity.label_)
        if entity.label_ == 'INGREDIENT':
            for chunk in sent.noun_chunks:
                if chunk.root.dep_ in ['obj','conj']:
                    if chunk.root.head.text != '':
                        if entity.text in chunk.text:
                            if (entity.text not in dicIngrVerb):
                                dicIngrVerb[entity.text] = list()
                                dicIngrVerb[entity.text].append(chunk.root.head.text)
                            else:
                                dicIngrVerb[entity.text].append(chunk.root.head.text)

                print([chunk.text, chunk.root.text, chunk.root.dep_,chunk.root.head.text])

```

Figure 4.2.2.3 Extrait de dépendance qui marche via un parcours de l'arbre de dépendance

4.2.3 Conclusions et perspectives

Les calculs de complexité en temps et en espace nécessitent au préalable une détection des EN qui interviennent dans les calculs mais aussi la création de dictionnaires qui permettent de relever les dépendances/liens entre ces différentes entités en particulier (INGREDIENT, QUANTITE, ACTION) et (INGREDIENT, USTENSILE). La première idée a été de chercher à identifier dans un premier temps ces entités. Nous avons rencontré différentes difficultés dans cette tâche d'identification, cependant nous avons réussi à mettre en place diverses méthodologies, techniques qui semblent donner de très bons résultats en termes de modèle NER. Nous avons relevé plusieurs défauts dans ce modèle, qui doivent être modifiés en particulier la stemmatisation qui réduit l'identification à ces formes que le modèle a appris. Aussi, il faudrait prendre en considération les noun chunck comme « huile d'olive » ou les verbes tels que « faire revenir » .., il s'agit là de quelques pistes concernant la partie purement modélisation du modèle NER pour son amélioration (idem pour les digit qui embarquent les températures qu'il faudrait finalement supprimées).

La détection de ces EN n'est pas suffisante pour les calculs de complexités : les liens de dépendances sont capitaux. Plusieurs expérimentations ont été mises en place reposant sur l'analyse morphosyntaxique de l'outil Spacy mais ces outils bien que très

utiles en TAL fournissent un nombre de résultats erronés beaucoup trop important (en français d'ailleurs bien plus qu'en anglais car ces algorithmes sont moins entraînés ou alors autant mais sur des données beaucoup moins volumineuses). Quelques pistes intéressantes d'extraction de lien via spaCy ont été fournies mais surtout un code a été développé pour le calcul de la complexité en considérant les liens déjà existants. Les résultats ne sont pas probants. Une piste à suivre serait celle fournie par [1] qui considère que le lien entre les ingrédients et les ustensiles se fait à travers les verbes. Il propose que l'on aille un cran plus loin que l'utilisation d'arbres de dépendances fournis par spaCy comme nous avons commencé à faire Cf. Figure 4.2.2.3. Cette partie relèvera plus de programmation récursive comme le proposé ce sujet de projet.

Troisième partie

Annexe

BIBLIOGRAPHIE

- [1] Nirav Diwan, Devansh Batra, Ganesh Bagler, A Named Entity Based Approach to Model Recipes, arXiv:2004.12184 [cs]
- [2] Elham Mohammadi, Louis Marceau, Eric Charton, Leila Kosseim, Luka Nerima, et al. Du bon usage d'ingrédients linguistiques spéciaux pour classer des recettes exceptionnelles. 6e conférence conjointe Journées d'Études sur la Parole (JEP, 33e édition), Traitement Automatique des Langues Naturelles (TALN, 27e édition), Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL, 22e édition). Volume 2 : Traitement Automatique des Langues Naturelles, 2020, Nancy, France. pp.81-94. fhal-02784757v3
- [3] Jushaan Kalra, Devansh Batra, Nirav Diwan, Ganesh Bagler, Nutritional Profile Estimation in Cooking Recipes, arXiv:2004.12286v1 [physics.soc-ph] 26 Apr 2020
- [4] Mori, Shinsuke, Maeta, Hirokuni, Yamakata, Yoko, Sasada, Tetsuro, Flow Graph Corpus from Recipe Texts, Proceedings of the Ninth International Conference on Language Resources and Evaluation, European Language Resources Association (ELRA), 2014, pp.2370–2377
- [5] Mori, Shinsuke, T. Sasada, Yoko Yamakata and Koichiro Yoshino, A Machine Learning Approach to Recipe Text Processing, 2012
- [6] Chen, Yuzhe. A Statistical Machine Learning Approach to Generating Graph Structures from Food Recipes, 2017
- [7] Losing, Viktor, Fischer, Lydia ,Jörg Deigmöller, Proceedings of the 11th Global Wordnet Conference, University of South Africa (UNISA), Global Wordnet Association, pp.81-90, 2021
- [8] Cyril Grouin, Patrick Paroubek, Pierre Zweigenbaum, DEFT2013 se met à table présentation du défi et résultats , In Actes de DEFT, 2013
- [9] Nuno Silva, David Ribeiro, and Liliana Ferreira. 2019. Information Extraction from Unstructured Recipe Data. In <i>Proceedings of the 2019 5th International Conference on Computer and Technology Applications, 2019, Association for Computing Machinery, New York, NY, USA, 165–168.
- [10] Rahul Agarwal, Kevin Miller, Information Extraction from Recipes
- [11] Mattingly, William. Introduction to Names Entity Recognition with a case study of Holocaust NER, 2020. ner.pythonhumanities.com.