Rapport :

Cognitive Computing

Table de matière :

Chapitre 1 : Contexte général de projet

1.1 Problématique

Le modèle standard de Spacy est un modèle qui génère par défaut plusieurs informations spécialement des entités des mots ou des textes, mais ce modèle reste très limité et ne peut pas prédire certaines entités comme : COVID-19 en étant une maladie (disease) ou autres termes soit très spécifiques ou nouveaux.

1.2 Objectifs de la recherche

L'objectif principal de la recherche est de couvrir les étapes de cycle de vie pour la construction d'un graphe de connaissance en se basant sur les technologies du Deep learning et NLP.

- Objectifs intermédiaires :
  - Étudier l'état de l'art sur l'utilisation de l'IA dans les méthodes d'extraction d'information qui couvre le cycle de vie de construction des graphes de connaissances.
  - Proposer et Appliquer des techniques d'intelligence artificielle pour construire un graphe de connaissance en utilisant des informations textuelles des articles médicaux.
  - Appliquer des techniques de théorie des graphes pour analyser le graphe construit afin de faire sortir des nouvelles connaissances.
  - Créer un nouveau modèle Spacy qui traite des mots ou termes nouveaux et spécifiques dans l'entrée.

1.3 La méthodologie de recherche

Pour atteindre les objectifs et répondre aux questions de recherche de ce projet, on va premièrement choisir des articles médicaux en tant que données d'entrées, pour appliquer ensuite le modèle de Spacy standard, et rectifier la nature des entités en créant un tout nouveau modèle basé sur des nouveaux concepts.

Chapitre 2 : Cadre théorique et revue de la littérature

Introduction

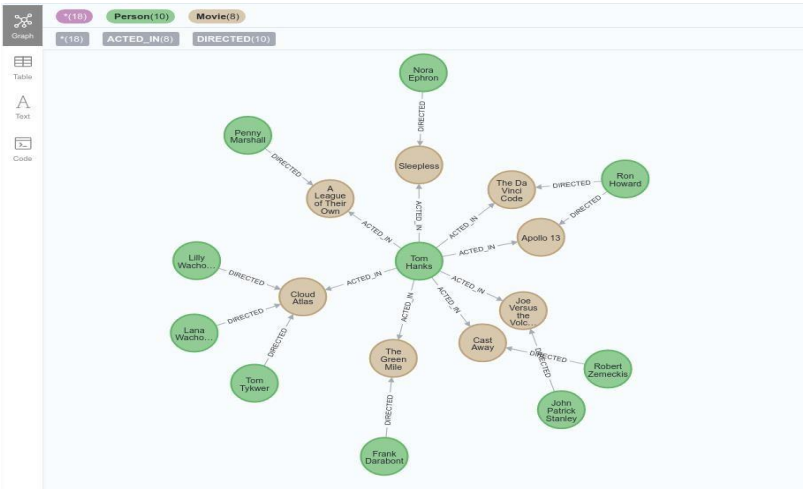
Dans la suite de ce rapport, nous allons consacrer ce chapitre pour présenter le cadre théorique du projet, où nous allons définir les différents concepts et notions traités à savoir le traitement du langage naturel (TLN, ou NLP en anglais).

1 Les graphes des connaissances

1.1 Définition

Un graphe de connaissance est une représentation de connaissances relatives à un domaine sous une forme exploitable par la machine. C'est un graphe étiqueté dans lequel les étiquettes ont des significations bien définies. Un graphe étiqueté se compose de nœuds, d'arêtes et d'étiquettes et des attribues. N'importe quoi peut faire type de nœud, par exemple, des personnes, une entreprise, un ordinateur, etc.

Une arête relie une paire de nœuds et capture la relation d'intérêt entre eux, par exemple, une relation d'amitié entre deux personnes, une relation client entre une entreprise et une personne, ou une connexion réseau entre deux ordinateurs. Les étiquettes donnent la signification de la relation, par exemple, la relation d'amitié entre deux personnes.



Un sous-graphe d'un graphe de connaissances des films et ses acteurs à grande échelle

1.2 Représentation

Le graphe de connaissances (KG) représente une collection de descriptions reliées entre elles d'entités - objets et événements[1] du monde réel où :

- Les descriptions ont une sémantique formelle qui permet aux personnes et aux ordinateurs de les traiter de manière efficace et sans ambiguïté.
- Les descriptions d'entités contribuent les unes aux autres, formant un réseau, où chaque entité représente une partie de la description des entités qui lui sont liées, et fournit un contexte pour leur interprétation.

1.3 Difficulté d'analyse

La création d'une base de connaissances est en fait un véritable défi, en partie en raison des difficultés et des subtilités du langage et de la nature éthérée et transitoire de la connaissance (par exemple, les faits et les connaissances évoluent en permanence).

Malgré toute la compréhension du langage évoquée dans les conférences et les récents développements de l'apprentissage profond, il n'existe toujours pas d'algorithme universel pour analyser et distiller une compréhension approfondie et non ambiguë du texte.

1.4 Application des graphes de connaissance

Les graphes de connaissances sont utilisés pour un large éventail d'applications allant de l'espace, du journalisme, de la biomédecine au divertissement, à la sécurité des réseaux et aux produits pharmaceutiques, parmi ses applications on trouve.

- La recherche d'informations contextuelles qui fournit une vue d'ensemble des informations sur un sujet donné en enrichissant les résultats.
- Les systèmes de recommandation proactifs.
- Robots conversationnels ou chatbots : un graphe de connaissances est utilisé pour relier les mots et les concepts afin de fournir la réponse la plus pertinente à la question de l'utilisateur.

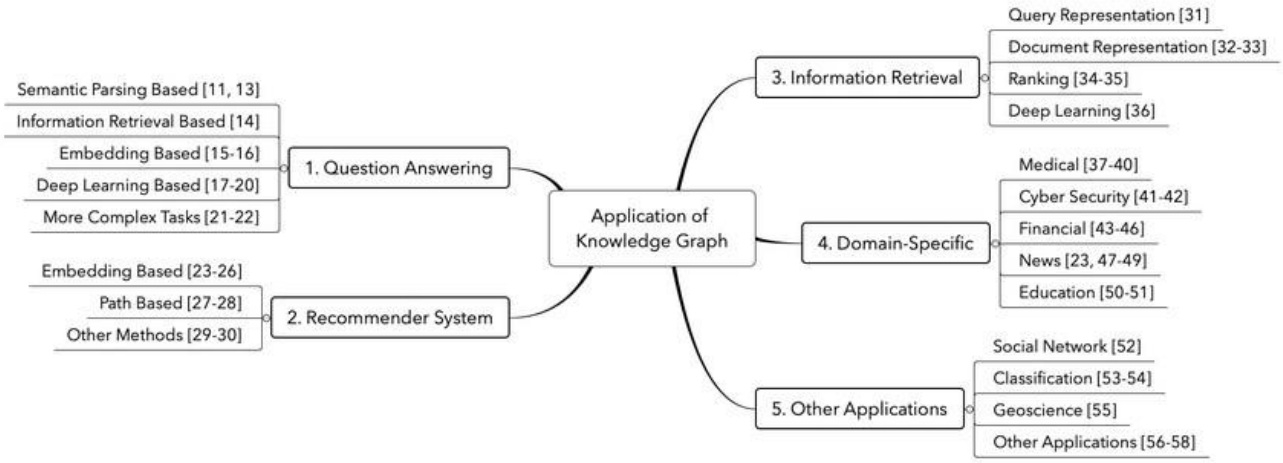


Diagramme représente les domaines d'applications pour les graphes de connaissances[2]

2 Construction des graphes de connaissance

2.1 Collection et extraction de l'information

Pour construire un graphe de connaissance à partir des données textuelles, on a besoin de techniques de traitement automatique du langage qui traitent :

- Extraction des entités
- Extraction des relations
- On construit le graphe en intégrant les concepts définis dans le modèle conceptuel des données.

2.2 Vérification et inférence

La dernière étape consiste à inférer de nouvelles relations, c'est à dire des nouveaux liens entres les nœuds existants dans la base des faits.

3 Deep learning pour la construction d’un graphe de connaissance

3.1 Traitement automatique des langues naturelles

3.1.1 Définition

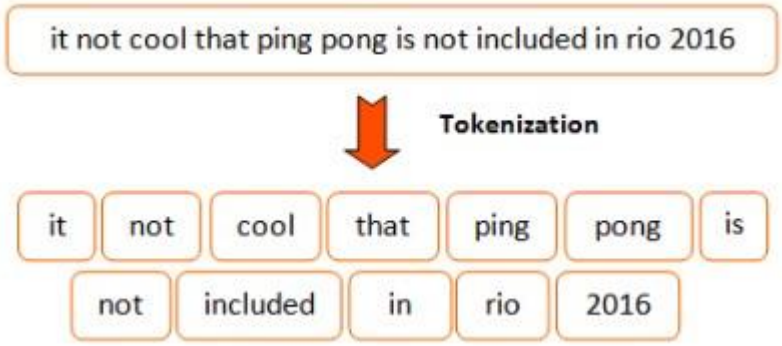
Le traitement du langage naturel (NLP) est une branche de l'intelligence artificielle qui aide les ordinateurs à comprendre, interpréter et manipuler le langage humain. Le NLP s'appuie sur de nombreuses disciplines, notamment l'informatique et la linguistique informatique, pour combler le fossé entre la communication humaine et la compréhension informatique.

Les machines d'aujourd'hui peuvent analyser plus de données linguistiques que les humains, sans fatigue et de manière cohérente et impartiale. Compte tenu de la quantité stupéfiante de données non structurées générées chaque jour, des dossiers médicaux aux médias sociaux, l'automatisation sera essentielle pour analyser pleinement et efficacement les données textuelles et vocales.

Le rôle du traitement du langage naturel est d'aider les ordinateurs à communiquer avec les humains dans leur propre langue et à effectuer d'autres tâches liées au langage. Par exemple, le traitement du langage naturel permet aux ordinateurs de lire un texte, d'entendre un discours, de l'interpréter, et de mesurer le sentiment.

3.1.2 Tokenisation

La tokenisation est l'une des tâches les plus courantes lorsqu'il s'agit de travailler avec des données textuelles.
La tokenisation consiste essentiellement à diviser une expression, une phrase, un paragraphe ou un document texte entier en unités plus petites, comme des mots ou des termes individuels. Chacune de ces petites unités est appelée token.
Les tokens peuvent être des mots, des chiffres ou des signes de ponctuation.



Un exemple de processus de tokenisation d'une phrase

3.1.3 Segmentation des phrases

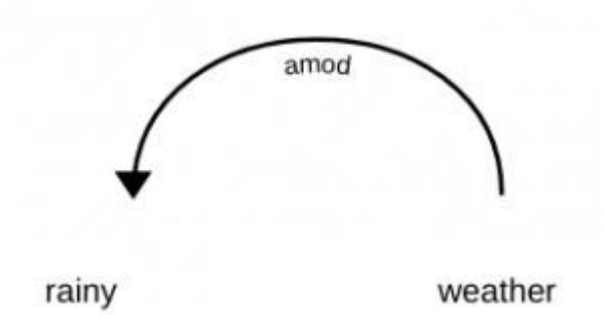
La segmentation des phrases (également appelée tokenisation des phrases) est le problème de la division d'une chaîne de langage écrit en ses phrases constitutives. L'idée ici semble très simple. En français et dans certaines autres langues, nous pouvons séparer les phrases chaque fois que nous voyons un signe de ponctuation.

Cependant, même en français, ce problème n'est pas trivial en raison de l'utilisation du caractère point pour les abréviations qui peuvent nous conduire à une une affectation incorrecte des limites de phrases.

3.1.4 Analyse des dépendances

L'analyse syntaxique des dépendances est le processus d'analyse de la structure grammaticale d'une phrase basée sur les dépendances entre les mots de la phrase.

Dans l'analyse syntaxique par dépendance, plusieurs balises représentent la relation entre deux mots dans une phrase. Ces balises sont les balises de dépendance. Par exemple, dans la phrase 'rainy weather', le mot “rainy” modifie le sens du nom “weather”. Par conséquent, il existe une dépendance de weather -> rainy dans laquelle weather agit en tant que tête et rainy agit en tant que dépendant ou enfant. Cette dépendance est représentée par la balise amod, qui représente le modificateur adjectival.

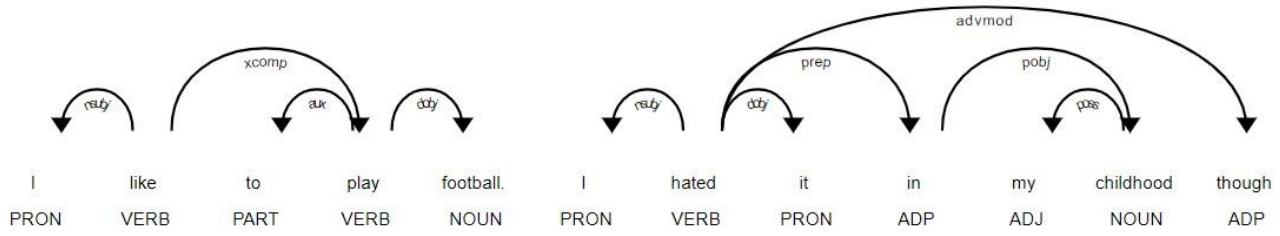


Exemple de dépendances entre deux mots

3.1.5 Étiquetage morpho-syntaxique

Le marquage POS (Part-of-speech) est un processus populaire de traitement du langage naturel qui consiste à classer les mots d'un texte (corpus) en fonction d'une partie particulière du discours, selon la définition du mot et son contexte.

Les balises POS permettent aux outils de traitement automatique de texte de prendre en compte la partie du discours de chaque mot. Cela facilite l'utilisation de critères linguistiques en plus des statistiques.



Un exemple d'un arbre de dépendance pour les balises POS pour une phrase

3.1.6 La reconnaissance d'entités nommées

La reconnaissance d'entités nommées (NER) (également connue sous le nom d'identification d'entités nommées, de découpage d'entités et d'extraction d'entités) est une sous-tâche de l'extraction d'informations qui vise à localiser et à classer les entités nommées mentionnées dans un texte non structuré dans des catégories prédéfinies telles que des noms de personnes, des organisations, des lieux,

des codes médicaux, des expressions temporelles, des quantités, des valeurs monétaires, des pourcentages, etc.

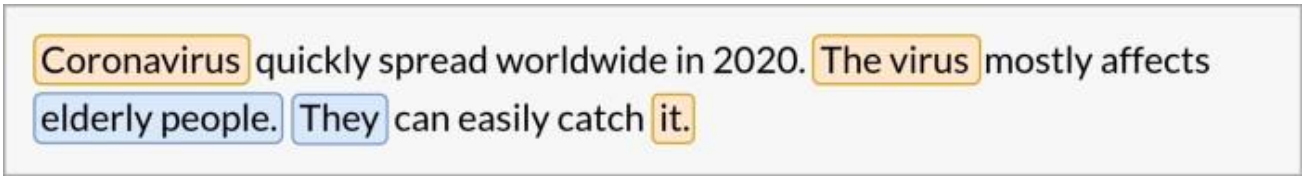
Les trois classes principales sont : la classe des entités (telles que le nom de la personne, le nom du lieu et le nom de l'institution), la classe du temps (telle que la date et l'heure) et la classe des nombres (par exemple, la monnaie et le pourcentage). Ces classes peuvent être étendues pour s'adapter à des domaines d'application spécifiques.

Ici un exemple d'un système de reconnaissance d'entités nommées qui met en évidence et classifie un texte médical sur COVID-19.

3.1.7 La résolution de la coréférence

La résolution de coréférence (CR) est une tâche du traitement du langage naturel (NLP). Elle vise à regrouper les expressions qui font référence à la même entité du monde réel afin d'obtenir un texte moins ambigu. Elle est utile dans des tâches telles que la compréhension de textes, la réponse à des questions et le résumé.

Grâce à la résolution des coréférences, nous voulons obtenir une phrase non ambiguë, qui ne nécessite pas de contexte supplémentaire pour être comprise. Le résultat attendu est illustré dans l'exemple simple suivant :

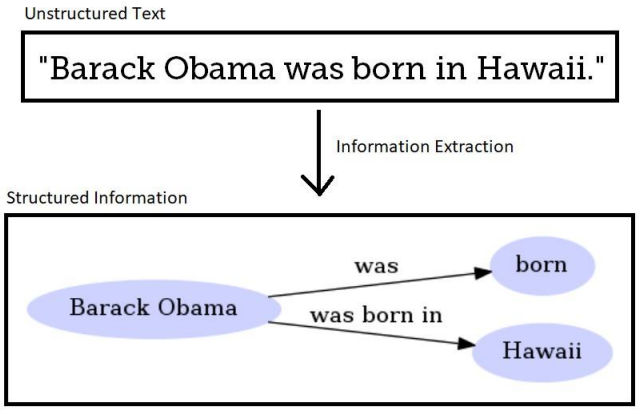


Un exemple de la résolution de la coréférence

Regrouper les entités détectées avec d'autres mentions/entités de mots originaux dans les phrases restantes.

3.1.8 Extraction des relations

L'extraction de relations consiste à extraire des relations sémantiques dans un texte. Les relations extraites se produisent généralement entre deux ou plusieurs entités d'un certain type (par exemple, une personne, une organisation, un lieu) et sont classées dans un certain nombre de catégories sémantiques (par exemple, marié à, employé par, vit à).



Un exemple d'extraction des relations dans une phrase.[3]

Chapitre 3 : Outils et Environnement d'Exécution

1 Langage de programmation

➢ Python pour data science :

- Python est le langage de programmation informatique le plus populaire et le plus utilisé, notamment dans le domaine de la Data Science et du Machine Learning.
- Il s'agit d'un langage de programmation interprété, qui ne nécessite donc pas d'être compilé pour fonctionner.

Logo de langage de programmation python

Ce langage est connu par sa performance dans tous les problèmes de Deep Learning, vu qu'il offre un large choix de bibliothèques afin de répondre à tous les besoins collectés.

2 Outils de développement

2.1 Solution Deep Learning

Outils de développement

Bibliothèques Python	Mission
sklearn	Scikit-learn est une librairie pour Python spécialisée dans l'apprentissage automatique, son module sklearn.metrics implémente des fonctions d'évaluation de l'erreur de prédiction à des fins spécifiques.

Pandas	Il s'agit d'une bibliothèque open source pour la manutention et l'analyse de données. Elle offre en particulier des structures de données et des traitements pour des tableaux numériques et de séries temporelles.
Numpy	Il s'agit d'une bibliothèque incontournable pour Python, conçue pour traiter les matrices ou les tableaux multidimensionnels et exploiter les fonctions mathématiques sur ce type de données.
Spacy	Spacy est une bibliothèque python en libre accès utilisée dans le traitement avancé du langage naturel et l'apprentissage automatique. Elle est utilisée pour construire des systèmes d'extraction d'information, de compréhension du langage naturel, et pour pré-traiter du texte pour l'apprentissage profond.
os	Le module OS de Python fournit des fonctions permettant d'interagir avec le système d'exploitation. OS fait partie des modules utilitaires standard de Python. Ce module fournit un moyen portable d'utiliser les fonctionnalités dépendantes du système d'exploitation.

2.2 Solution pour éditer le code

2.2.1 Visual Code

Visual Studio Code (plus connu sous le nom de VS Code) est un éditeur de texte open source gratuit de Microsoft. Il est disponible pour Windows, Linux et macOS. Bien que l'éditeur soit relativement léger, il comprend des fonctionnalités puissantes qui ont fait de VS Code l'un des outils d'environnement de développement les plus populaires ces derniers temps.

Le logo de VScode

2.3 Environnement du travail :

Pour la réalisation de ce projet, nous avons disposé d'un ordinateur LENOVO caractérisé par :

- Processeur : Intel(R) Core(TM) I7 2.4 GHz
- Mémoire : 8 Go de RAM.
- Disque dur : 255 Go.
- Système d'exploitation : Windows 10

CHAPITRE 4 : Réalisation de la solution

Le présent chapitre, sera dédié aux différentes étapes de la mise en place de notre approche du travail, à savoir les outils/bibliothèques déployées, les différents fonctions et modèles utilisées. Et l'étape suivante sera la présentation des 4 étapes de notre approche d'étude.

1-Named Entity recognition :

- Commençons par importer les bibliothèques requises et les modèles nltk standards.

```
import spacy
import random
import pandas as pd
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\LENOVO\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\LENOVO\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\LENOVO\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\LENOVO\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
True
```

- On crée deux dataframes, le premier pour les caractéristiques grammaticales(pos) et le deuxièmme pour les entités des textes(label).

```
df1 = pd.DataFrame(columns = ['Text', 'pos'])
df2 = pd.DataFrame(columns = ['Text', 'Label'])
```

- Une fonction annotate1 pour extraire les caractéristiques grammaticales par le modèle spacy standard et les ajouter au dataframe df1.

```
def annotate1(text):
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(text)

    rows = []
    for token in doc:
        print(token.text, token.pos_)
        rows.append([token.text, token.pos_])
    df1 = pd.DataFrame(rows, columns=['Text', 'pos'])
    #df.to_csv('C:\Users\LENOVO\Desktop\cours\S5\Ingénierie cognitive\Projet\data.csv')
    return df1
```

- Une fonction annotate2 pour extraire les entités des textes par le modèle spacy standard et les ajouter au dataframe df2.

```
def annotate2(text):
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(text)

    rows = []
    for ent in doc.ents:
        print(ent.text, ent.label_)
        rows.append([ent.text, ent.label_])
    df2 = pd.DataFrame(rows, columns=['Text', 'label'])
    #df.to_csv(r'C:\Users\LENOVO\Desktop\cours\S5\Ingénierie cognitive\Projet\data2.csv')
    return df2
```

- Importer un texte par open() et le lire par read().

```
f = open(r'C:\Users\LENOVO\Desktop\Final livrable\NER\Article1_ressources\Article1_text.txt', 'r',encoding="utf8")
txt = f.read()
```

- Afficher les caractéristiques grammaticales de chaque mot du texte importé avec un tableau df1 pour simplifier la lecture des résultats.

```
annotate1(txt)
```

```
Output exceeds the size limit. Open the full output data in a text editor
Malaria PROPN
and CCONJ
COVID-19 NOUN
may AUX
have VERB
similar ADJ
aspects NOUN
and CCONJ
seem VERB
to PART
have VERB
a DET
strong ADJ

 SPACE
potential NOUN
for ADP
mutual ADJ
influence NOUN
. PUNCT
They PRON
have AUX
already ADV
caused VERB
millions NOUN
```

	Text	pos
0	Malaria	PROPN
1	and	CCONJ
2	COVID-19	NOUN
3	may	AUX
4	have	VERB
...	...	...
802	preventive	ADJ
803	measure	NOUN
804	for	ADP
805	both	DET
806	disease	NOUN

- Afficher les entités pour chaque mot du texte importé avec un tableau df1 pour simplifier la lecture des résultats.

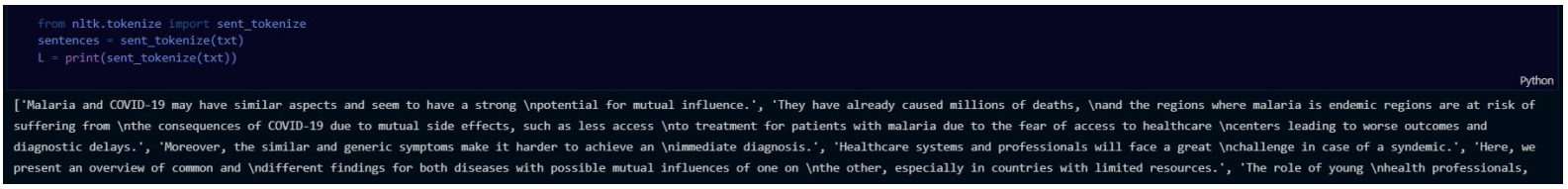
```
annotate2(txt)
```

```
Malaria GPE
COVID-19 PERSON
millions CARDINAL
Healthcare ORG
one CARDINAL
COVID-19 PERSON
WHO ORG
COVID-19 PRODUCT
WHO ORG
COVID-19 PERSON
COVID-19 PERSON
the next month DATE
COVID□19 ORG
COVID-19 PERSON
16 CARDINAL
COVID-19 PERSON
1 CARDINAL
2 CARDINAL
3 CARDINAL
4 CARDINAL
TB ORG
```

	Text	label
0	Malaria	GPE
1	COVID-19	PERSON
2	millions	CARDINAL
3	Healthcare	ORG
4	one	CARDINAL
5	COVID-19	PERSON
6	WHO	ORG
7	COVID-19	PRODUCT
8	WHO	ORG
9	COVID-19	PERSON
10	COVID-19	PERSON
11	the next month	DATE

- Détection des entités par un modèle de reconnaissance d'entités nommées.

- Tokenisation des phrases.



- Tokenisation des mots.

```
from nltk.tokenize import word_tokenize
for word in sentences:
    token_sentences = word_tokenize(word)
```

```
token_sentences
```

```
Output exceeds the size limit. Open the full output data in a text editor
['Finally',
 ',',
 'from',
 'a',
 'global',
 'perspective',
 ',',
 'it',
 'is',
 'necessary',
 'to',
 'increase',
 'and',
 'join',
 'efforts',
 'in',
 'order',
 'to',
 'develop',
 'an',
 'effective',
 'vaccine',
```

- Part of speech (POS) tagging

```
for word in sentences:
    text = word_tokenize(word)
    pos_sentences = nltk.pos_tag(text)
```

```
pos_sentences
```

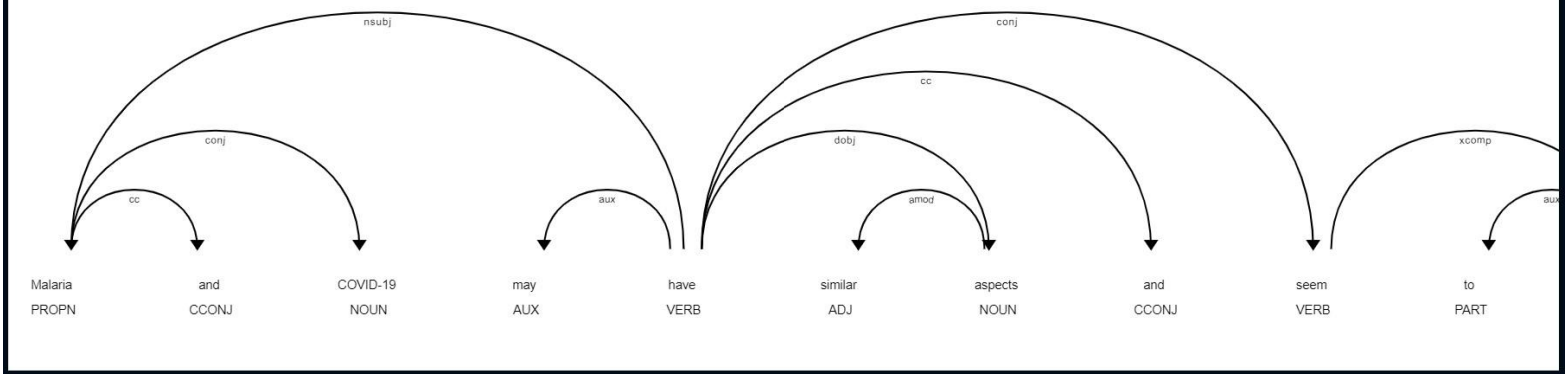
Output exceeds the size limit. Open the full output data in a text editor

```
[('Finally', 'RB'),
 (',', ','),
 ('from', 'IN'),
 ('a', 'DT'),
 ('global', 'JJ'),
 ('perspective', 'NN'),
 (',', ','),
 ('it', 'PRP'),
 ('is', 'VBZ'),
 ('necessary', 'JJ'),
 ('to', 'TO'),
 ('increase', 'VB'),
 ('and', 'CC'),
 ('join', 'VB'),
 ('efforts', 'NNS'),
 ('in', 'IN'),
 ('order', 'NN'),
 ('to', 'TO'),
 ('develop', 'VB'),
 ('an', 'DT'),
 ('effective', 'JJ'),
 ('vaccine', 'NN'),
 ('and', 'CC'),
 ('it', 'PRP'),
 ('make', 'VBP'),
```

- Analyse des dépendances pour chaque phrase.

```
for i in sentences:
    doc = nlp(i)
    displacy.render(doc, style="dep" , jupyter=True)
```

Python



2-Relation extraction :

- On commence par l'extraction des entités. La fonction ci-dessous extrait le sujet et l'objet (entités) d'une phrase.

```
def get_entities(sent):
  ent1 = ""
  ent2 = ""

  prv_tok_dep = ""
  prv_tok_text = ""

  prefix = ""
  modifier = ""

  for tok in nlp(sent):
    if tok.dep_ != "punct":
      if tok.dep_ == "compound":
        prefix = tok.text
        if prv_tok_dep == "compound":
          prefix = prv_tok_text + " "+ tok.text

      if tok.dep_.endswith("mod") == True:
        modifier = tok.text
        if prv_tok_dep == "compound":
          modifier = prv_tok_text + " "+ tok.text

      if tok.dep_.find("subj") == True:
        ent1 = modifier +" "+ prefix + " "+ tok.text
        prefix = ""
        modifier = ""
        prv_tok_dep = ""
        prv_tok_text = ""

      if tok.dep_.find("obj") == True:
        ent2 = modifier +" "+ prefix +" "+ tok.text


      prv_tok_dep = tok.dep_
      prv_tok_text = tok.text

  return [ent1.strip(), ent2.strip()]
```

- Afficher toutes les entités de chaque phrase par des listes.

```
for i in sentences:
    print(get_entities(i))
```

```
['Malaria', 'mutual  influence']
['where  malaria', 'worse side outcomes']
['it', 'immediate  diagnosis']
['Healthcare systems', 'great \n syndemic']
['Here  we', 'limited  resources']
['role', 'high \n health malaria']
['', 'such  fever']
['malaria health worker diagnosis', 'endemic \n countries']
['', 'potential \n transmission']
['malaria control efforts', 'COVID-19 response']
['challenges COVID-19', 'public health crisis']
['impact', 'health financing whole']
['economic', 'also  COVID-19']
['next  it', 'real malaria control prevention']
['is', 'consequent  morbidity']
['characteristics', 'other \n diseases']
['endemic COVID-19', 'following']
['that', 'together  strikes']
['prevention  measures', 'protective hospital equipment']
['Community engagement', 'contact funeral tracing']
['', 'healthcare  structures']
['Data management', '']
['It', 'routine health services use']
['this', 'preventive  disease']
```

- Cette fonction extrait les paires d'entités pour toutes les phrases de nos données.

```
entity_pairs = []

for i in tqdm(sentences):
  entity_pairs.append(get_entities(i))
```

```
100%|██████████| 24/24 [00:00<00:00, 82.13it/s]
```

```
entity_pairs[10:20]
```

```
[['challenges COVID-19', 'public health crisis'],
 ['impact', 'health financing whole'],
 ['economic', 'also  COVID-19'],
 ['next  it', 'real malaria control prevention'],
 ['is', 'consequent  morbidity'],
 ['characteristics', 'other \n diseases'],
 ['endemic COVID-19', 'following'],
 ['that', 'together  strikes'],
 ['prevention  measures', 'protective hospital equipment'],
 ['Community engagement', 'contact funeral tracing']]
```

- Pour construire un graphe de connaissances, nous avons besoin d'arêtes pour connecter les nœuds (entités) les uns aux autres. Ces arêtes sont les relations entre une paire de nœuds.
- La fonction ci-dessous est capable de capturer de tels prédicats à partir des phrases.

```
def get_relation(sent):

  doc = nlp(sent)

  # Matcher class object
  matcher = Matcher(nlp.vocab)

  #define the pattern
  pattern = [{'DEP':'ROOT'},
             {'DEP':'prep','OP':"?"},
             {'DEP':'agent','OP':"?"},
             {'POS':'ADJ','OP':"?"}]

  matcher.add('matching_1', [pattern])

  matches = matcher(doc)
  k = len(matches) - 1

  span = doc[matches[k][1]:matches[k][2]]

  return(span.text)


relations = [get_relation(i) for i in tqdm(sentences)]
```

- Nombre de chaque relation trouvée par get_relation().

```
pd.Series(relations).value_counts()[:50]
```

```
have similar        1
caused              1
is of fundamental   1
neglected           1
be able             1
is crucial          1
applied for         1
focus on            1
focused on          1
indicate            1
result in           1
be possible         1
happened with       1
```

3-Knowledge Graph :

- Créer un dataframe d'entités et de prédicats.

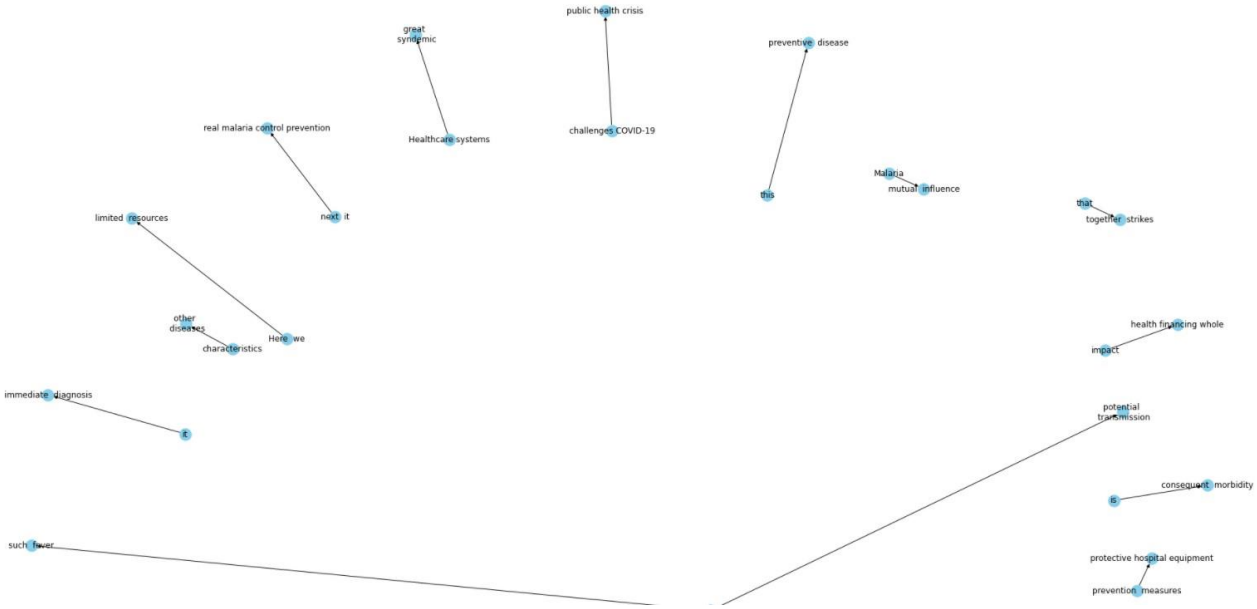
```
source = [i[0] for i in entity_pairs]

target = [i[1] for i in entity_pairs]

kg_df = pd.DataFrame({'source':source, 'target':target, 'edge':relations})
```

- Créer un graphe orienté à partir d'un dataframe.

```
G=nx.from_pandas_edgelist(kg_df, "source", "target", edge_attr=True, create_using=nx.MultiDiGraph())
```

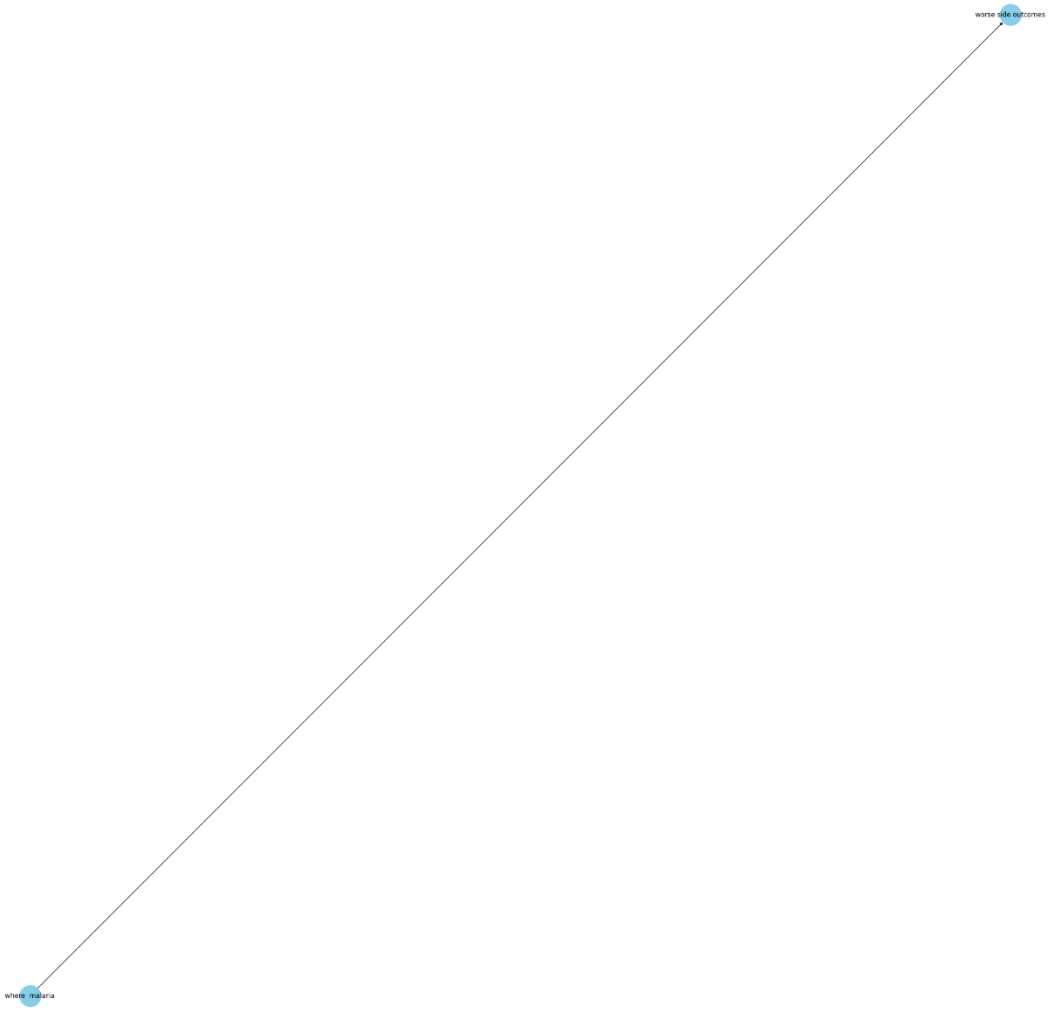
- Tracer le réseau.

```
plt.figure(figsize=(30,30))
pos = nx.spring_layout(G)
nx.draw(G, with_labels=True, node_color='skyblue', edge_cmap=plt.cm.Blues, pos = pos)
plt.show()
```



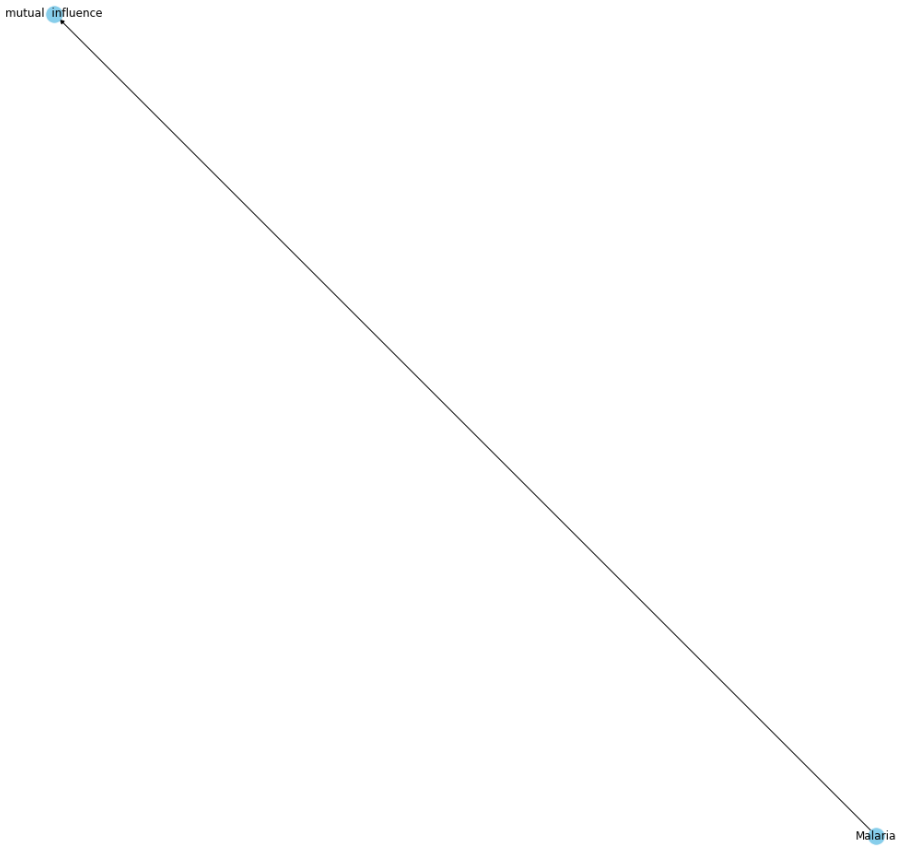
- Utiliser juste quelques relations importantes pour visualiser un graphique.

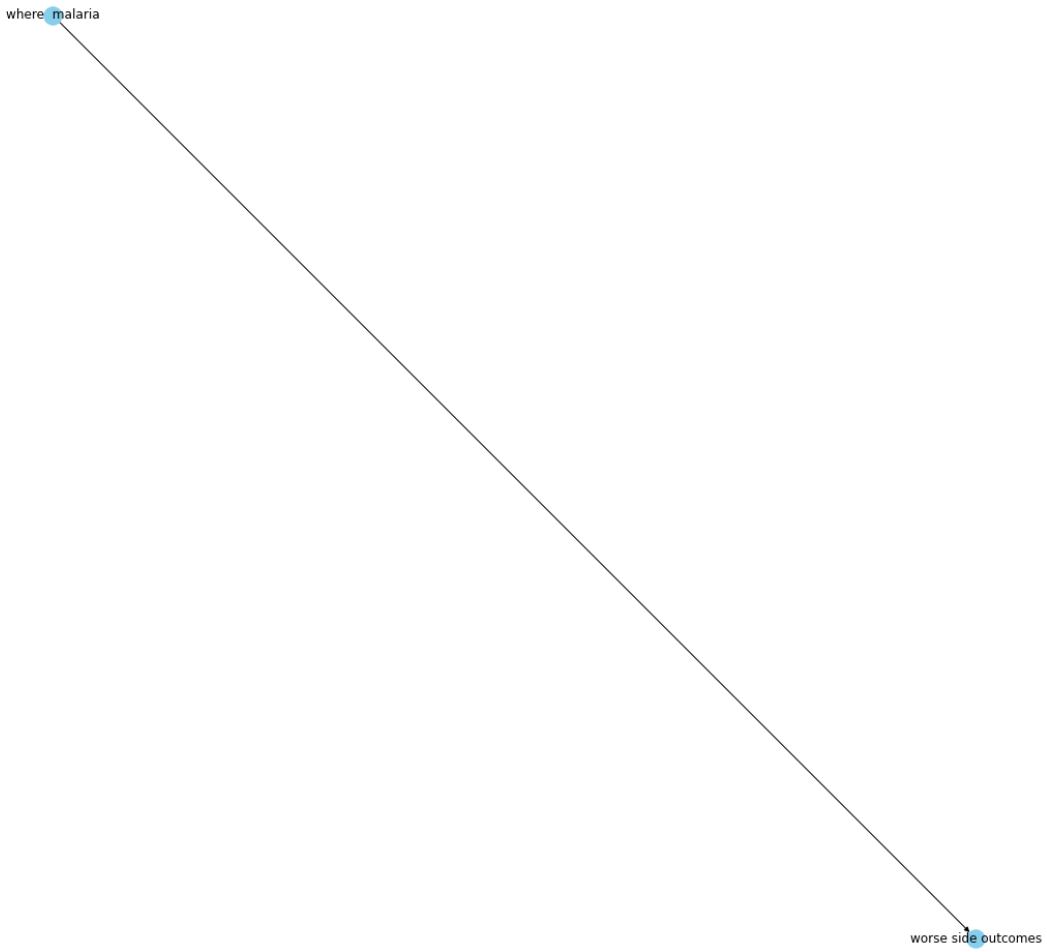
```
G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="caused"], "source", "target", edge_attr=True, create_using=nx.MultiDiGraph())
plt.figure(figsize=(30,30))
pos = nx.spring_layout(G, k = 0.5)
nx.draw(G, with_labels=True, node_color='skyblue', node_size=1500, edge_cmap=plt.cm.Blues, pos = pos)
plt.show()
```



- Afficher une visualisation séparée pour chaque phrase.

```
entity_pairs1 = []
relations1 = []

for i in tqdm(sentences):
    entity_pairs1.append(get_entities(i))
    relations1.append(get_relation(i))
    source = entity_pairs1[0][0]
    target = entity_pairs1[0][1]
    kg1_df = pd.DataFrame({'source':source, 'target':target, 'edge':relations1})
    G1=nx.from_pandas_edgelist(kg1_df, "source", "target", edge_attr=True, create_using=nx.MultiDiGraph())
    plt.figure(figsize=(15,15))
    pos = nx.spring_layout(G)
    nx.draw(G1, with_labels=True, node_color='skyblue', edge_cmap=plt.cm.Blues)
    plt.show()
    entity_pairs1 = []
    relations1 = []
```





4-Customed spaCy model :

- Premièrement, nous allons charger les données, initialiser les paramètres et créer ou charger le modèle NLP. Commençons par importer les bibliothèques requises et chargeons le jeu de données.
- Puis, nous allons créer un modèle s'il n'y a pas de modèle existant, sinon nous chargerons le modèle existant.

```python
import spacy
import random


model_file = None
iterations= 20

# Training data
TRAINING_DATA = [('COVID-19', {'entities': [(0, 8, 'disease')]}), ('Iman', {'entities': [(0, 4, 'Person')]}), ('Khalid', {'entities': [(0, 6, 'Person')]}),]

# Testing sample data
test_sample='what is COVID-19'
test_sample2='Khalid and Iman'


if model_file is not None:
    " "
    nlp = spacy.load(model_file)
    print("Load Existing NER Model ", model_file)
else:
    nlp = spacy.blank('en')
    print("Created blank NLP model")
```

- Dans cette étape, nous allons créer un pipeline NLP. Tout d'abord, nous vérifions s'il existe un pipeline, puis nous utilisons le pipeline existant, sinon nous créerons un nouveau pipeline.

```python
if 'ner' not in nlp.pipe_names:
    ner_pipe = nlp.add_pipe('ner')
else:
    ner_pipe = nlp.get_pipe('ner')
```

- Dans cette étape, nous ajouterons les étiquettes des entités au pipeline. Tout d'abord, nous itérons l'ensemble de données d'apprentissage, puis nous ajoutons chaque entité au modèle.

```python
for text, annotations in TRAINING_DATA:
    for entity in annotations.get('entities'):
        ner_pipe.add_label(entity[2])
```

- Dans cette étape, nous allons former le modèle NER. Tout d'abord, nous désactivons tous les autres pipelines, puis nous ne suivons que la formation NER. Dans la formation NER, nous créerons un optimiseur. Après cela, nous mettrons à jour le modèle PNL en fonction du texte et des annotations dans l'ensemble de données d'entraînement. Ce processus se poursuit jusqu'à un nombre défini d'itérations.

```
from spacy.training.example import Example

other_pipes = [pipe for pipe in nlp.pipe_names if pipe != 'ner']

with nlp.disable_pipes(*other_pipes):  # only train NER
    optimizer = nlp.begin_training()
    for itn in range(iterations):
        print("Iteration Number:" + str(itn))
        random.shuffle(TRAINING_DATA)
        losses = {}
        for text, annotations in TRAINING_DATA:
            example = Example.from_dict(nlp.make_doc(text), annotations)
            nlp.update([example],
                drop=0.2,
                sgd=optimizer,
                losses=losses)
        print("Loss:",losses['ner'])
```

```
Output exceeds the size limit. Open the full output data in a text editor
Iteration Number:0
Loss: 2.675879657268524
Iteration Number:1
Loss: 2.5540661811828613
Iteration Number:2
Loss: 2.3213374614715576
Iteration Number:3
Loss: 1.933038353919983
Iteration Number:4
Loss: 1.4558714926242828
Iteration Number:5
Loss: 0.8661706820130348
```

- Pour les tests, nous devons d'abord convertir le texte de test en objet PNL pour les annotations linguistiques puis on test le modèle.

```
test_document = nlp(test_sample)
for ent in test_document.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
what 0 4 Person
is 5 7 Person
COVID-19 8 16 disease
```

```
test_document = nlp(test_sample2)
for ent in test_document.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
Khalid 0 6 Person
and 7 10 disease
Iman 11 15 Person
```

Conclusion

Les graphes de connaissances en tant que sujet de recherche est de plus en plus populaire pour représenter les relations structurelles entre les entités. Ces dernières années ont vu la publication de divers graphes de connaissances open source et pris en charge par les entreprises avec une croissance spectaculaire dans l'application de la représentation des connaissances et du raisonnement dans différents domaines tels que le traitement du langage naturel et la vision par ordinateur.