Traitement automatique de la langue naturelle

IFT-7022

Session Automne 2020

Rapport du TP2

Destinataire

Département d'informatique et de génie logiciel

Luc Lamontagne

Date de remise : lundi 28 Décembre 2020

Israël Akobi **111 252 705**

I. Introduction

Le traitement de langage naturel regroupe plusieurs tâches dans le but d'analyser et de comprendre et de faire imiter certaines compétences linguistiques humaines par les machines. Parmi ces tâches on peut avoir la classification de texte, l'analyse syntaxique, l'analyse de sentiment, la recherche et la classification d'entités nommées. Pour accomplir ces différentes tâches plusieurs algorithme ont été développés en partant des algorithmes classiques comme du HMM, ou MEMM jusqu'aux algorithme de l'apprentissage machine et du deep learning. La performance de ces algorithmes dépend des types de tâche auxquelles ils sont appliqués.

Dans ce projet de traitement de langage naturel, nous sommes amenés à faire une classification de séquences principalement des entités nommées. Un texte déjà annoté nous a été fourni et l'objectif est d'apprendre ces séquences à partir d'un algorithme de notre choix. Même si nous considérons que les algorithmes comme le HMM ou le MEMM ont une bonne performance en classification de séquence, nous avons choisis les algorithmes de deep learning à cause de leur capacité à bien représenter des données complexes. Les modèles ont été développé avec TensorFlow.

Dans la suite du rapport, nous présenterons les méthodes utilisées et les résultats obtenus.

II. Les données

Pour accomplir cette tâche, nous avons utilisé des données issues de deux domaines : les articles scientifiques et la santé. L'objectif est de déterminer des entités nommées scientifiques. La nomenclature utilisée est celui de BIO à première vue. Dans le but de rendre un peu plus uniforme les étiquètes, nous avons remplacé les B- (Begining) par I- (Intermediaire) par ce qu'on a très peu d'entités de type B. À la suite nous n'avons conservé que le nom des entités pour faire la classification.

1. Donnée de la science

	Entrainement	Validation	Test
Nombre de phrase	2210	324	650

Tableau 1: Nombre de phrases par types de données (Science)

Dans cette donnée, nous avons **6 d'entités nommées** (`I-Generic, I-Material, I-Method, I-Metric, I-OtherScientificTerm, I-Task`)

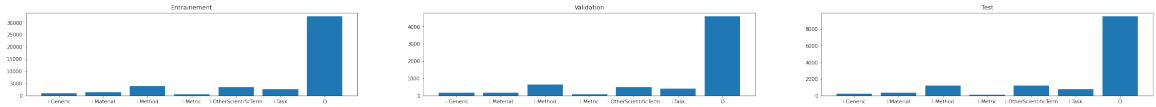


Figure 1: Répartition des Labels (Science)

La figure ci-dessus montre le nombre de chaque entité nommée dans le texte. On peut percevoir que la majorité des mots sont des out of vocabulary, ce qui juste. Ceci pourrait constituer un problème, par qu'on pourrait avoir une donnée non équilibrée. Probablement le fait d'utiliser des séquences et non des bag of words pourrait atténuer cet effet.

Un histogramme de la longueur des phrases se présente comme suit :

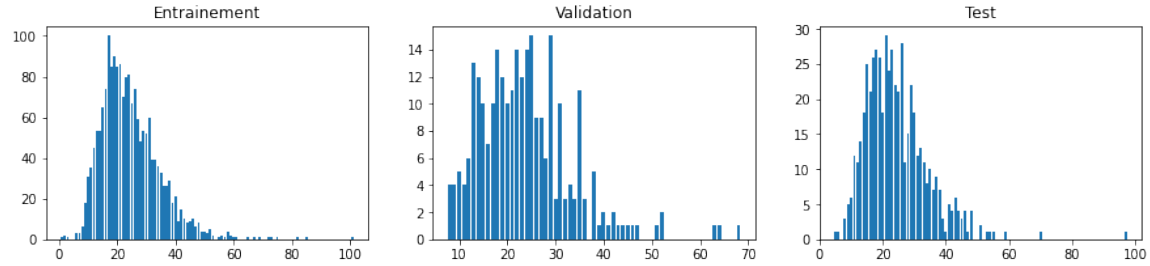


Figure 2: Histogramme de longueur de phrases (Science)

La répartition de la longueur des phrases est assez similaire en entraiment, en validation et en test. Ceci pourrait être un atout pour l'algorithme.

2. Donnée de santé

	Entrainement	Validation	Test
Nombre de phrase	5424	923	940

Tableau 2: Nombre de phrases par types de données (Disease)

Nous avons un seul type d'entité nommée dans ce corpus (I-Disease). La répartition de mots en fonction des labels se présente comme suit :

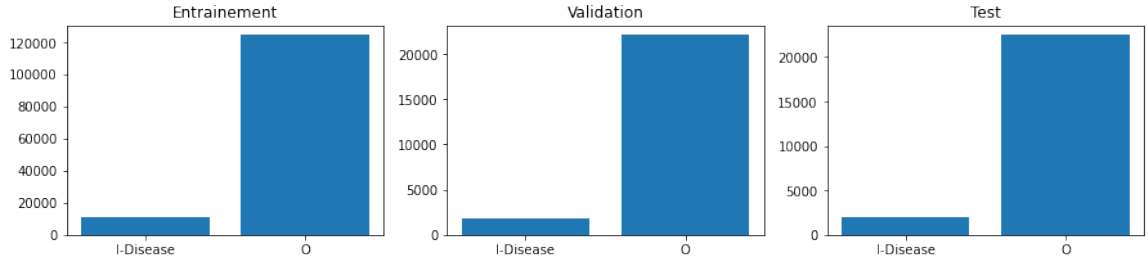


Figure 3: Répartition des Labels (Disease)

Comme dans la donnée de science, les mots out of vocabulary sont prépondérants.

La longueur des différentes séquences des mots se présente comme suit dans la figure suivante :

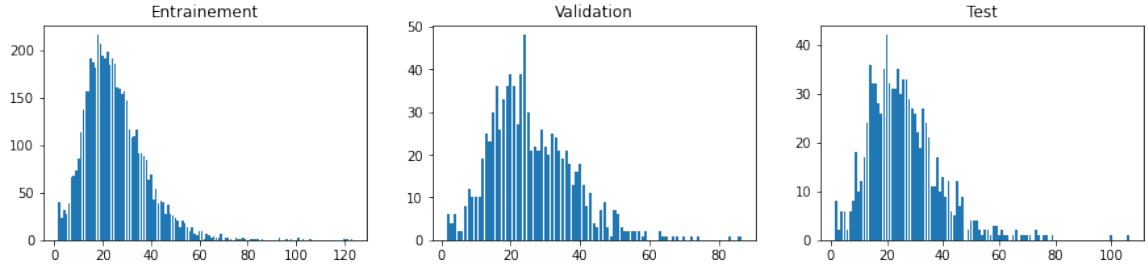


Figure 4: Histogramme de longueur de phrases (Disease)

Les séquences dans les 3 type de données ont une longueur assez similaire.

III. Traitement des données

Nous avons utilisé deux types d'architecture différentes : un modèle avec SpaCy et l'autre du deep learning avec TensorFlow. Le prétraitement des données dans les deux cas est différent. L'objectif du prétraitement est de rendre la donnée utilisable par les modèles. Une analyse du fichier nous permet de savoir qu'après chaque phrase il y a une ligne blanche. À partir de cela, nous pouvons reconstruire les séquences de mots qui constituent une phrase.

1. Prétraitement pour le deep learning

Premièrement, nous avons extrait les données des fichiers textes, en les reconstituant en séquence de mots d'une phrase équivaut à une séquence d'étiquettes. Les séquences de mots sont considérées comme les données et les séquences d'étiquettes comme le libellé des données.

Nous avons ensuite passé les séquences de mots à un tokenizer pour les convertir en séquences de chiffres (index de chaque mot). Cette séquence peut nourrir notre réseau de deep learning.

Dans le but d'avoir des séquences de même taille, nous rembourrer les séquences en prenant comme longueur de séquence la longueur maximale des séquences des données d'entrainement. Le mot de rembourrage est __PADDING__.

```
word_tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token='UNK')
word_tokenizer.fit_on_texts(train_data)
word_index = word_tokenizer.word_index
word_index['__PADDING__'] = 0
index_word = {i:w for w, i in word_index.items()}

x_train = word_tokenizer.texts_to_sequences(train_data)
x_valid = word_tokenizer.texts_to_sequences(valid_data)
x_test = word_tokenizer.texts_to_sequences(test_data)

x_train = pad_sequences(x_train, maxlen=MAX_LEN, padding='post')
x_valid = pad_sequences(x_valid, maxlen=MAX_LEN, padding='post')
x_test = pad_sequences(x_test, maxlen=MAX_LEN, padding='post')
```

En ce qui concerne les séquences d'étiquette, nous les avions aussi convertis en chiffre en suivant la même procédure avec un tokenizer. Les étiquettes du mot de rembourrage est O (out of vocabulary).

```
[ ] d_tag_tokenizer = Tokenizer(num_words=D_LABEL_SIZE, lower=False, document_count=0)
    d_tag_tokenizer.fit_on_texts(d_train_label)

    d_tag_index = d_tag_tokenizer.word_index
    d_tag_index['__PADDING__'] = 0
    d_index_tag = {i:w for w, i in d_tag_index.items()}

    d_index_tag_wo_padding = dict(d_index_tag)
    d_index_tag_wo_padding[d_tag_index['__PADDING__']] = 'O'


    y_d_train = d_tag_tokenizer.texts_to_sequences(d_train_label)
    y_d_valid = d_tag_tokenizer.texts_to_sequences(d_valid_label)
    y_d_test = d_tag_tokenizer.texts_to_sequences(d_test_label)

    y_d_train = pad_sequences(y_d_train, maxlen=D_MAX_LEN, padding='post')
    y_d_valid = pad_sequences(y_d_valid, maxlen=D_MAX_LEN, padding='post')
    y_d_test = pad_sequences(y_d_test, maxlen=D_MAX_LEN, padding='post')
```

2. Prétraitement pour le module NER de Spacy

Pour entrainer un module NER de SpaCy, les données devraient être sous la forme : ***(phrase, {entities: [(début, fin, étiquette)]}), début*** indexe du premier caractère de l'entité et ***fin*** indexe du dernier caractère de l'entité. L'objectif est de transformé les données en cette forme pour la passer au modèle.

Cette tache a été faite à partir de cette fonction.

```
def load_data(filename):
  '''
  Cette fonction permet de créer un dataset compatible pour entrainer le modèle NER de Spacy
  filename: chemin des données
  '''
  sentence= list()
  labels = set()
  entities = list()
  data = list()
  with open(filename) as fn:
    content = fn.readlines()
  end = 0
  for x in content:
    x = x.strip().split('\t')

    if len(x) !=1:
      end += (len(x[0]) + 1)
      sentence.append(x[0])
      x[1] = x[1].replace('B-', 'I-')
      if x[1] != 'O':
        labels.add(x[1].replace('I-', ''))
        start = end - len(x[0]) - 1
        entities.append((start, end - 1, x[1].replace('I-', '')))

    else:
      sentence = " ".join(sentence)
      data.append([sentence, {'entities' : entities}])
      end = 0
      entities, sentence = list(), list()

  return data, labels
```

IV. Les modèles utilisés pour le deep learning

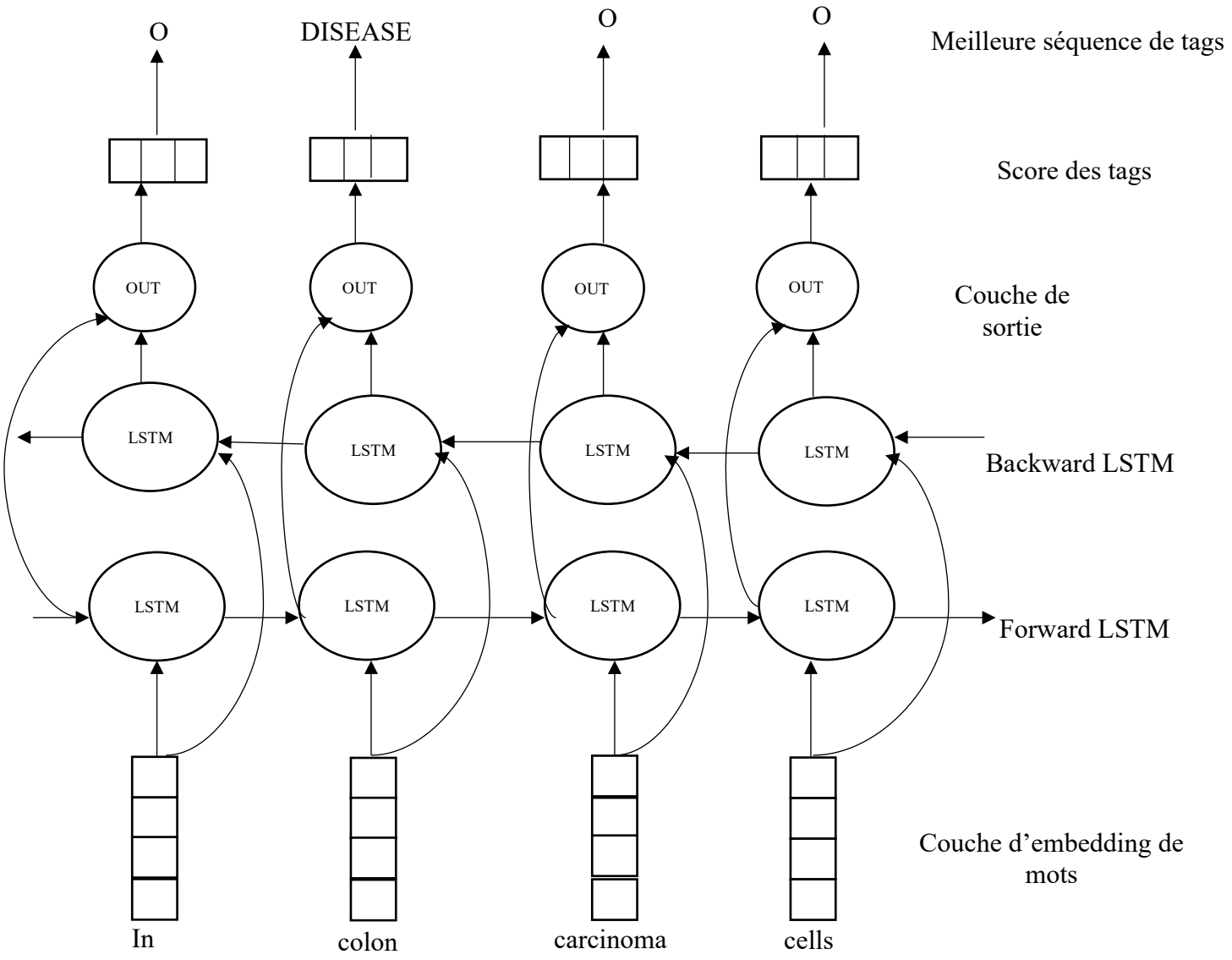
La classification des séquences se faisait au départ avec les algorithmes traditionnels comme du HMM, ou le MEMM. Avec la révolution apportée par le Deep learning, une grande importance a été accordé aux réseaux de neurones récurrents notamment les LSTM pour le traitement de cette tâche. Dans le cadre de notre travail, nous avons utilisé les LSTM.

Le LSTM (long short-term memory) sont est une architecture de réseau neuronal récurrent artificiel utilisée dans le domaine de l'apprentissage en profondeur. Contrairement aux réseaux neuronaux à action directe standard, le LSTM a des connexions de rétroaction[1]. LSTM, permet de garder en mémoire les informations sur les données précédemment traitées, ce qui est très important dans le traitement des données avec les séquences.

LSTM peut-être configuré comme bidirectionnel ou unidirectionnel. Nous avons utilisé des LSTM bidirectionnels dans le but de capturer les informations dans les deux sens de la phrase.

Dans le cadre de notre travail, nous avons testé principalement deux modèles basés sur les LSTM que nous présenterons. L'architecture générale des deux modèles se présente comme le schéma suivant :

[1] Wikipédia (anglais)



Modèle LSTM

Dans le premier modèle, la couche d'embedding de mot a été entrainé sur les données d'entrainement. Dans le second modèle utilisé, nous avons initialisé les poids de la couche d'embedding par le vecteur de poids des mots de vocabulaire dans GloVe. La couche d'embedding n'est pas entrainé dans le second modèle. L'objectif est de savoir l'impact d'un embedding pré-entrainé sur la classification de séquence.

Nous passons les données issues du prétraitement dans le modèle. En sortie, nous avons pour chaque mot de la séquence un score pour chaque type d'étiquette. Le softmax est

appliqué sur ces scores pour déterminer l'étiquette la plus probable. Celle-ci est attribuée au mot. Ainsi se fait l'attribution d'étiquette pour chaque mot de la séquence.

```
# Cette fonction crée le modèle à utiliser
def get_model(vocab, label_size, max_len, embeddings = None):
  model = Sequential()
  if embeddings:
    #charger l'embedding des mots
    num_words = len(vocab)
    embedding_matrix = np.zeros((num_words, 300))

    for word, i in zip(vocab, range(len(vocab))):
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

    model.add(Embedding(len(vocab),300,input_length=max_len,trainable=False))
    model.add(Bidirectional(LSTM(512,return_sequences=True,dropout=0.2)))
    model.add(TimeDistributed(Dense(label_size+1, activation='softmax')))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=["acc"])

  else:
    model.add(Embedding(len(vocab),300,input_length=max_len,trainable=True)) # Couche d'embedding
    model.add(Bidirectional(LSTM(512,return_sequences=True,dropout=0.2))) # Couche de Bidirectionel LSTM

    model.add(TimeDistributed(Dense(label_size+1, activation='softmax'))) # Couche de classification de sequence
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=["acc"])


  return model
```

V. Résultats obtenus

Nous avons entrainé les données. Pour plus de clarté, nous avons rapporté que les résultats sur les données de test. Les courbes d'accuracy et de loss en entrainement et en validation sont dans le notebook que nous avons fourni.

1. Données d'article scientifique

a. Module NER de Spacy

	p	r	f
Material	62.500000	58.666667	60.522696
OtherScientificTerm	58.081705	53.694581	55.802048
Method	67.744681	67.629567	67.687075
Task	66.055046	46.936115	54.878049
Generic	67.078189	63.424125	65.200000
Metric	72.164948	50.000000	59.071730
avg	63.962691	57.524148	60.572805

b. Modèle deep learning sans GloVe

	precision	recall	f1-score	support
Generic	0.64	0.18	0.28	257
Material	0.63	0.06	0.12	375
Method	0.60	0.34	0.43	1177
Metric	0.00	0.00	0.00	140
O	0.77	0.96	0.85	9467
OtherScientificTerm	0.46	0.13	0.21	1218
Task	0.46	0.30	0.36	767
accuracy			0.74	13401
macro avg	0.51	0.28	0.32	13401
weighted avg	0.69	0.74	0.69	13401

c. Modèle deep learning avec GloVe

	precision	recall	f1-score	support
Generic	0.00	0.00	0.00	257
Material	0.00	0.00	0.00	375
Method	0.40	0.00	0.00	1177
Metric	0.00	0.00	0.00	140
O	0.71	1.00	0.83	9467
OtherScientificTerm	0.00	0.00	0.00	1218
Task	0.00	0.00	0.00	767
accuracy			0.71	13401
macro avg	0.16	0.14	0.12	13401
weighted avg	0.53	0.71	0.59	13401

2. Données Médicales

a. Module NER de Spacy

	p	r	f
Disease	89.625668	81.875916	85.575696
avg	89.625668	81.875916	85.575696

b. Modèle deep learning sans GloVe

	precision	recall	f1-score	support
Disease	0.00	0.00	0.00	2047
O	0.92	1.00	0.96	22450
accuracy			0.92	24497
macro avg	0.46	0.50	0.48	24497
weighted avg	0.84	0.92	0.88	24497

c. Modèle deep learning avec GloVe

	precision	recall	f1-score	support
Disease	0.00	0.00	0.00	2047
O	0.92	1.00	0.96	22450
accuracy			0.92	24497
macro avg	0.46	0.50	0.48	24497
weighted avg	0.84	0.92	0.88	24497

Les différents résultats obtenus nous montrent un résultat excellent avec le module NER de SpaCy. Avec les modèles deep learning, nous avons des résultats médiocres surtout avec les données médicales où aucune entité nommée n'a été détecté.

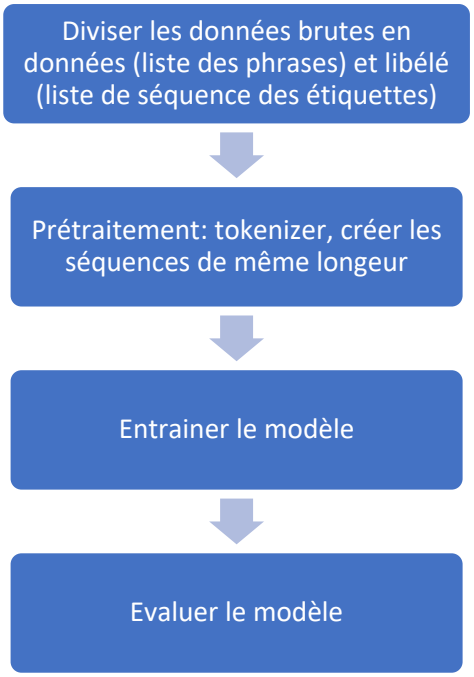
Nous constatons aussi que l'utilisation d'embedding pré-entrainé n'apporte pas une amélioration significative aux résultats.

Nous pouvons aussi dire que le taux d'exactitude n'est pas une bonne métrique pour évaluer nos modèles, par ce que bien que l'exactitude soit élevée (92%), le modèle n'a pu bien classer que les mots hors du vocabulaire.

Nous pouvons aussi augmenter les textes ayant plus d'entités nommées pour pallier cette mauvaise classification.

VI. Conclusion

Ce projet nous a amené à nous familiariser à la classification de séquence. La séquence d'exécution du projet peut être résumée dans le schéma suivant :



Avec les performances obtenues et le temps d'exécution de chaque modèle, le module NER de SpaCy est recommandable. En plus de donner un meilleur résultat, il s'exécute en peu de temps.

VII. Référence

1. https://towardsdatascience.com/named-entity-recognition-ner-meeting-industrys-requirement-by-applying-state-of-the-art-deep-698d2b3b4ede
2. https://aihub.cloud.google.com/u/0/p/products%2F2290fc65-0041-4c87-a898-0289f59aa8ba
3. https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/