

Appariement de questions/réponses

Félix Jamet

April 6, 2018

Contents

1	Structures de données et algorithmes (semeval_struct.py)	2
1.1	Imports	2
1.2	Arbres	2
1.2.1	Transformation en profondeur	2
1.2.2	Transformation sur N niveaux	3
1.2.3	Transformation générique	3
1.3	Tris	4
1.3.1	Dictionnaire	4
1.3.2	Tri naturel	4
2	Fonctionnalités de support (semeval_util.py)	5
2.1	Imports	5
2.2	Sauvegarde et chargement d'objets sur le disque	5
3	Traitement du langage naturel (semeval_taln.py)	6
3.1	Imports	6
3.2	Analyse des questions par un modèle	6
3.3	Pondération de termes	7
3.3.1	<i>Term Frequency</i>	7
3.3.2	<i>Inverse Document Frequency</i>	7
3.3.3	<i>Term Frequency - Inverse Document Frequency</i>	8
4	Évaluation des approches (semeval_executable.py)	10
4.1	Imports	10
4.2	Scores	10
4.2.1	Écriture des fichiers de prédiction	10
4.3	Dimensions orthogonales d'une approche	11
4.3.1	Modèle de langage	11
4.3.2	Corpus	11
4.3.3	Extraction de contenu	12
4.3.4	Filtrage des mots	12
4.3.5	Construction des sacs de mots	12
4.3.6	Création des approches	13
4.4	Exécution des approches	14
5	Résultats	16

1 Structures de données et algorithmes (semeval_struct.py)

1.1 Imports

```
1 import re
2 import operator
```

1.2 Arbres

1.2.1 Transformation en profondeur

Il s'agit ici d'appliquer récursivement une fonction à chacune des feuilles d'un arbre.

```
1 def transformtree_deep(func, tree):
2     """Transform a tree by applying a function to its leaves.
3
4     Parameters
5     -----
6     func : function
7         The function that will transform each leaf.
8
9     tree : recursive dict
10        The tree to transform.
11
12     Returns
13     -----
14     out : recursive dict
15        The transformed tree.
16     """
17     return {
18         key: transformtree_deep(func, value)
19         if isinstance(value, dict)
20         else func(value)
21         for key, value in tree.items()
22     }
```

1.2.2 Transformation sur N niveaux

L'inconvénient de `transformtree_deep` est qu'elle parcourt l'arbre entièrement, tandis qu'il serait parfois utile de ne parcourir que les N premiers niveaux.

```
1 def transformtree_n(func, tree, n):
2     """Transform a tree up to a maximum depth by applying a function to its leaves.
3
4     Once the maximum depth is reached, the function is applied, even if it is not a leaf.
5
6     Parameters
7     -----
8     func : function
9         The function that will transform each leaf, as well as the final depth.
10
11     tree : recursive dict
12         The tree to transform.
13
14     n : int
15         The maximum depth
16
17     Returns
18     -----
19     out : recursive dict
20         The transformed tree.
21     """
22     if n > 0:
23         return {
24             key: transformtree_n(func, value, n - 1)
25             if isinstance(value, dict)
26             else func(value)
27             for key, value in tree.items()
28         }
29     else:
30         return { key: func(value) for key, value in tree.items() }
```

1.2.3 Transformation générique

Une sémantique intéressante pour la transformation d'arbre est de considérer que transformer un arbre jusqu'au niveau -1 revient à le transformer sans limite de profondeur.

```
1 def transformtree(func, tree, n=-1):
2     if n < 0:
3         return transformtree_deep(func, tree)
4     else:
5         return transformtree_n(func, tree, n)
```

1.3 Tris

1.3.1 Dictionnaire

Par défaut, le tri sur les entrées d'un dictionnaire se fait sur les clés. On propose la fonction `sorted_items` pour faire un tri sur les valeurs.

```
1 def sorted_items(dictionary, key=operator.itemgetter(1), reverse=False):
2     return sorted(dictionary.items(), key=key, reverse=reverse)
```

1.3.2 Tri naturel

Il n'y a pas de support natif dans python pour trier des éléments dans l'ordre naturel. La fonction `natural_sort_key` permet d'atteindre ce résultat, lorsque utilisée comme le paramètre `key` d'une fonction de tri.

```
1 def natural_sort_key(key):
2     """Transform a key in order to achieve natural sort.
3     from https://blog.codinghorror.com/sorting-for-humans-natural-sort-order/
4     """
5     def convert(text):
6         return int(text) if text.isdigit() else text
7     return [convert(c) for c in re.split('([0-9]+)', key)]
```

2 Fonctionnalités de support (semeval_util.py)

2.1 Imports

```
1 import pickle
```

2.2 Sauvegarde et chargement d'objets sur le disque

On utilise pickle pour sauvegarder et charger des objets qu'il serait long de reconstruire à chaque execution d'un script.

```
1 def save_object(obj, filename):  
2     pickle.dump(obj, open(filename, 'wb'))  
3  
4 def load_object(filename):  
5     return pickle.load(open(filename, 'rb'))
```

3 Traitement du langage naturel (semeval_taln.py)

3.1 Imports

```
1 import os.path
2 import math
3 from itertools import chain
4 from collections import Counter
5 from semeval_xml import get_semeval_id, get_related_threads, xmlextract
6 from semeval_util import save_object, load_object
```

La classe Counter est une sous-classe de dict, permettant de compter les occurrences d'une clé. Elle est ici utilisée pour représenter des sacs de mots.

3.2 Analyse des questions par un modèle

Étant donné que les questions sont organisées de manière hiérarchiques, à savoir N questions originales, chacune accompagnée de 10 questions reliées, elles vont être représentées par un arbre.

Chaque question est passée dans un modèle de langage, produisant ainsi un document.

La fonction `make_document_tree` permet de construire l'arbre des documents, à partir :

- des questions originales (`original_questions`),
- d'un modèle de langage (`model`),
- d'une fonction d'extraction de contenu (`content_extractor`)

```
1 def make_document_tree(original_questions, model, content_extractor):
2     result = {}
3     for org in original_questions:
4         orgid = get_semeval_id(org)
5         result[orgid] = {
6             get_semeval_id(rel): model(content_extractor(rel))
7             for rel in get_related_threads(org)
8         }
9         result[orgid]['org'] = model(content_extractor(org))
10    return result
```

L'analyse d'une phrase par un modèle étant une opération possiblement coûteuse, les documents liés aux questions vont être sauvegardées sur le disque afin de ne pas avoir à refaire tous les calculs à chaque fois.

```
1 def make_or_load_document_tree(xml_source, saved_path, model, content_extractor,
2     ↪ verbose=False):
3     if os.path.isfile(saved_path):
4         if verbose:
5             print('Loading document tree from', saved_path)
```

```

5         result = load_object(saved_path)
6         return result
7     else:
8         if verbose:
9             print('Creating document tree. This might take a while...')
10
11        extractor = xmlextract(xml_source)
12        result = make_document_tree(
13            extractor.get_org_elements(), model, content_extractor)
14
15        if verbose:
16            print('Saving document tree to', saved_path)
17        save_object(result, saved_path)
18
19        return result

```

3.3 Pondération de termes

TF (*Term Frequency*) et IDF (*Inverse Document Frequency*) sont des mesures permettant de pondérer des termes selon leur importance dans un corpus.

Les documents sont ici manipulés comme des sacs de mots, implémentés ici sous forme de compteurs.

3.3.1 *Term Frequency*

La TF d'un terme correspond à sa fréquence d'apparition dans l'ensemble des documents.

$$TF(terme, document) = \frac{occurences(terme, document)}{taille(document)}$$

où la taille d'un document correspond au nombre de termes qu'il contient.

Plutôt que de calculer la TF d'un terme dans un document à chaque fois que nécessaire, la TF de tous les termes d'un document est stockée dans un dictionnaire.

```

1 def term_frequencies(bag):
2     documentlen = sum(bag.values())
3     return {
4         term: occurrences / documentlen
5         for term, occurrences in bag.items()
6     }

```

3.3.2 *Inverse Document Frequency*

L'IDF d'un terme est proportionnelle à l'inverse du nombre de documents dans lesquels il apparaît. Elle se base sur la DF (*Document Frequency*), correspondant au nombre de document dans lesquels un terme apparaît.

$$DF(terme, corpus) = ||\{doc / doc \in corpus \wedge terme \in doc\}||$$

$$IDF(terme, corpus) = \log \left(\frac{taille(corpus)}{DF(terme, corpus)} \right)$$

De la même manière que pour la TF, l'IDF de tous les termes du corpus est stockée dans un dictionnaire.


```

1 def document_frequencies(corpus):
2     result = Counter()
3     for document in corpus:
4         result.update({term for term in document})
5     return result
6
7
8 def inverse_document_frequencies(corpus, DF=None):
9     if DF == None:
10         DF = document_frequencies(corpus)
11     return {term: math.log2(len(corpus)/docfreq)
12             for term, docfreq in DF.items()}

```

3.3.3 Term Frequency - Inverse Document Frequency

La TF-IDF d'un terme correspond à une combinaison de sa TF et de son IDF :

$$TF\text{-}IDF(\text{terme}, \text{document}, \text{corpus}) = TF(\text{terme}, \text{document}) * IDF(\text{terme}, \text{corpus})$$

La TF-IDF d'un terme est implémentée comme une fonction utilisant des dictionnaires TF et IDF passés en paramètres.

```

1 def tf_idf(term, termfreq, inversedocfreq, out_of_corpus_value):
2     """Term Frequency - Inverse Document Frequency of a term using dictionaries.
3
4     If the term is not in the inverse document frequency dictionary, this function will
→ use the argument out_of_corpus_value.
5
6     Parameters
7     -----
8     term : str
9         The term.
10
11     termfreq : dict
12         The term frequencies of the document.
13
14     inversedocfreq : dict
15         The inverse document frequencies of the corpus.
16
17     Returns
18     -----
19     out : float
20         The TF-IDF value of the term.
21     """
22     if term not in termfreq:
23         return 0
24     if term not in inversedocfreq:
25         return out_of_corpus_value
26
27     return termfreq[term] * inversedocfreq[term]

```

1. Sacs de mots Le score TF-IDF d'un sac de mots correspond à la somme des valeurs TF-IDF de ses éléments :

$$\text{score}_{TF\text{-}IDF}(\text{sac}, \text{document}, \text{corpus}) = \sum_{\text{terme} \in \text{sac}} TF\text{-}IDF(\text{terme}, \text{document}, \text{corpus})$$

```

1 def tf_idf_bow(bag, termfreq, inversedocfreq, out_of_corpus_value):
2     return sum(tf_idf(term, termfreq, inversedocfreq, out_of_corpus_value) *
3               ↪ occurrences
3               for term, occurrences in bag.items())

```

2. Similarité de documents Le score de similarité entre deux documents correspond au score TF-IDF du sac de mots qu'ils forment.

$$\text{similarité}(doc_a, doc_b, corpus) = \text{score}_{TF-IDF}(SAC_{doc_a} \cap SAC_{doc_b}, SAC_{doc_a} \cup SAC_{doc_b}, corpus)$$

Où SAC_{doc} est le sac de mots de doc .

Les documents doc_a et doc_b sont considérées comme ayant un rôle symétriques, c'est pourquoi le sac de mots envoyé à score_{TF-IDF} est leur union.

```

1 def tf_idf_bow_scorer(bag_maker, doca, docb, inversedocfreqs, out_of_corpus_value):
2     бага = bag_maker(doca)
3     bagb = bag_maker(docb)
4     intersection = бага & bagb
5     termfreq = term_frequencies(baga + bagb)
6     return tf_idf_bow(intersection, termfreq, inversedocfreqs, out_of_corpus_value)

```

Où `bag_maker` est une fonction retournant un sac de mots.

4 Évaluation des approches (semeval_executable.py)

4.1 Imports

```
1  #!/usr/bin/env python3
2  from itertools import product, combinations
3  import spacy
4  from spacy.lang.en.stop_words import STOP_WORDS
5  from semeval_struct import *
6  from semeval_util import *
7  from semeval_xml import get_semeval_content
8  from semeval_taln import *
```

4.2 Scores

Les scores sont stockés dans un arbre construit à partir de l'arbre des documents. `compute_relqs_scores` calcule les scores de similarité d'un noeud de l'arbre des documents, en attribuant à chaque question relié son score obtenu en comparaison avec sa question originale.

```
1  def compute_relqs_scores(orgqnode, scorer):
2      return {relid: scorer(orgqnode['org'], orgqnode[relid])
3          for relid in orgqnode.keys() - {'org'}}
```

`make_score_tree` transforme le premier niveau d'un arbre de documents en lui appliquant `compute_relqs_scores` associé à la fonction de scoring recue en paramètre.

```
1  def make_score_tree(document_tree, scorer):
2      return transformtree(
3          lambda x: compute_relqs_scores(x, scorer),
4          document_tree,
5          0
6      )
```

4.2.1 Écriture des fichiers de prédiction

Semeval fournit un script permettant de noter les performances d'une approche. Ce script prend en entrée un fichier de prédiction dont chaque ligne correspond à une question reliée et est formatée de la manière suivante :

```
orgq_id relq_id 0 score true
```

Les troisième et cinquième colonnes sont sans intérêt pour cette tâche.

Le fichier de prédiction est destiné à être ensuite comparé à un fichier de référence de Semeval, afin d'évaluer les performances du système.

La fonction `write_scores_to_file` permet de générer ce fichier de prédiction. Les résultats sont triés sur le tas, pour correspondre à l'ordre du fichier de références.

```
1 def write_scores_to_file(scores, filename):
2     """Write a semeval score tree to a prediction file.
3
4     Parameters
5     -----
6     scores : dict of dict of float
7         The scores to write.
8
9     filename : str
10        The name of the output file.
11    """
12    linebuffer = [(orgid, relid, str(0), str(score), 'true')
13                  for orgid, relqs in scores.items()
14                  for relid, score in relqs.items()]
15
16    linebuffer.sort(key=lambda x: natural_sort_key(x[1]))
17
18    with open(filename, 'w') as out:
19        out.write('\n'.join(['\t'.join(el) for el in linebuffer]))
```

4.3 Dimensions orthogonales d'une approche

Plusieurs dimensions orthogonales sont envisagées pour appairer des questions. Ces dimensions sont combinées les unes avec les autres, en faisant un produit cartésien, formant ainsi une approche.

4.3.1 Modèle de langage

Un seul modèle de langage est utilisé.

```
1 models = {
2     'spacy_en': spacy.load('en')
3 }
```

4.3.2 Corpus

Les approches sont testées sur les données 2016 et 2017 de Semeval.

```
1 corpuses = {
2     '2016': 'SemEval2016-Task3-CQA-QL-test-input.xml',
3     '2017': 'SemEval2017-task3-English-test-input.xml',
4 }
```

4.3.3 Extraction de contenu

Deux manières d’extraire du contenu sont envisagées. Elles se différencient au niveau de l’extraction du contenu des questions reliées. La première extrait uniquement le sujet et le corps d’une question, tandis que la seconde extrait également les commentaires des questions reliées.

```
1 extractors = {  
2     'questions': get_semeval_content,  
3     # 'questions_with_comments': get_semeval_content_with_relcomments  
4 }
```

Ces fonctions sont fournies dans le fichier `semeval_xml.py`.

4.3.4 Filtrage des mots

Les mots d’un sac de mots peuvent être filtrés ou non selon un prédicat.

```
1 def isnotstopword(word):  
2     return word not in STOP_WORDS  
3  
4 filters = {  
5     'gtr2': lambda word: len(word) > 2,  
6     'nostopwords': isnotstopword,  
7     'nofilter': lambda x: True,  
8 }
```

4.3.5 Construction des sacs de mots

Les sacs de mots sont construits à l’aide de deux fonctions. La première est une fonction d’extraction de caractéristique, qui étant donné un token, renvoie la caractéristique désirée de celui-ci. La deuxième est une fonction d’extraction de phrase, qui étant donné un document, renvoie un itérable contenant des mots.

Chaque méthode de construction de sacs de mots utilise ces deux fonctions.

```
1 def extracttext(tok):  
2     return tok.text  
3  
4 def extractlemma(tok):  
5     return tok.lemma_  
6  
7 def extractlabel(ent):  
8     return ent.label_ if hasattr(ent, 'label_') else None  
9  
10 def getentities(doc):  
11     return doc.ents  
12  
13 wordextractors = {  
14     'text': extracttext,  
15     'lemma': extractlemma,  
16     'label': extractlabel,  
17 }  
18  
19 sentenceextractors = {  
20     'entities': getentities,  
21     'document': lambda x: x,  
22 }
```

```

23
24 bowmakers = {
25     'named_entities_text': ('text', 'entities'),
26     'named_entities_label': ('label', 'entities'),
27     'tokens': ('text', 'document'),
28     'lemmas': ('lemma', 'document'),
29 }
30
31 def getbowmakerfunctions(key):
32     return (wordextractors[bowmakers[key][0]], sentenceextractors[bowmakers[key][1]])

```

Les fonctions associées aux éléments de bowmakers sont destinés à être passés à la fonction `createbowmaker`, retournant une fonction permettant de construire un sac de mots selon les modalités voulues.

```

1 def createbowmaker(wordextractor, sentenceextractor, filters):
2     def bowmaker(document):
3         return Counter(
4             list(filter(lambda x: all(f(x) for f in filters),
5                           map(wordextractor, sentenceextractor(document))))
6         )
7
8     return bowmaker

```

4.3.6 Création des approches

Les arbres des documents sont précalculés pour éviter de répéter cette opération coûteuse.

```

1 training_file = 'SemEval2016-Task3-CQA-QL-train-part1.xml '
2
3 training_doctree = make_or_load_document_tree(
4     training_file,
5     'train_2016_part1.pickle',
6     models['spacy_en'],
7     get_semeval_content,
8     verbose=True
9 )
10
11 doctrees = {
12     '_'.join((model, corpus, extractor)): make_or_load_document_tree(
13         corpuses[corpus],
14         '_'.join((model, corpus, extractor)) + '.pickle',
15         models[model],
16         extractors[extractor],
17         verbose=True
18     )
19     for model, corpus, extractor in product(models, corpuses, extractors)
20 }

```

La fonction `nonemptypartitions` est utilisée pour combiner les filtres.

```

1 def nonemptypartitions(iterable):
2     for i in range(1, len(iterable) + 1):
3         for perm in combinations(iterable, i):
4             yield perm
5
6 def join_predicates(iterable_preds):

```

```

7     def joinedlocal(element):
8         for pred in iterable_preds:
9             if not pred(element):
10                return False
11            return True
12        print('joining', *(pred for pred in iterable_preds))
13        return joinedlocal
14
15 filters_partition = list(nonemptypartitions(set(filters) - {'nofilter'}))
16
17 filters_partition.append(('nofilter',))

```

Les approches sont créées en faisant le produit cartésien des dimensions envisagées.

```

1 approches = list(product(doctrees, bowmakers, filters_partition))

```

4.4 Exécution des approches

Chacune des approches précédemment générées est exécutée et les scores produits sont écrits dans les fichiers correspondants.

```

1 def getpredfilename(doctree, bowmaker, filterspartition):
2     return '_'.join((doctree, bowmaker, *filterspartition, 'scores.pred'))
3
4
5 # inversedocfreqs = transformtree(
6 #     lambda wordextractor: inverse_document_frequencies(
7 #         [[wordextractor(tok) for tok in doc]
8 #          for org in training_doctree.values()
9 #          for doc in org.values()])
10 # ),
11 # wordextractors
12 # )
13
14 inversedocfreqs = {
15     wordex + '_' + sentex: inverse_document_frequencies(
16         [[wordextractors[wordex](tok) for tok in sentenceextractors[sentex](doc)]
17          for org in training_doctree.values()
18          for doc in org.values()])
19     )
20     for wordex, sentex in bowmakers.values()
21 }
22
23 out_of_corpus_value = max(inversedocfreqs['text_document'].values())
24
25 for doctree, bowmaker, filterspartition in approches:
26     wordex, sentex = bowmakers[bowmaker]
27     bowmakerfunc = createbowmaker(wordextractors[wordex], sentenceextractors[sentex],
28                                   [filters[filterkey] for filterkey in filterspartition])
29
30     scores = make_score_tree(
31         doctrees[doctree],
32         lambda a, b: tf_idf_bow_scorer(
33             bowmakerfunc, a, b,

```

```
34         inversedocfreqs[wordex + '_' + sentex], out_of_corpus_value)
35     )
36
37     prediction_file = getpredfilename(doctree, bowmaker, filterspartition)
38     print('writing scores to', prediction_file)
39     write_scores_to_file(scores, prediction_file)
```


5 Résultats

Le script shell suivant est utilisé pour extraire le score MAP d'un fichier de prédiction :

```
1 prediction=$1
2
3 if echo $prediction | grep --quiet "2016"
4 then
5     reference=scorer/SemEval2016-Task3-CQA-QL-test.xml.subtaskB.relevancy
6 else
7     reference=scorer/SemEval2017-Task3-CQA-QL-test.xml.subtaskB.relevancy
8 fi
9
10 python2 scorer/ev.py $reference $prediction | grep "^MAP" | sed 's/ \+;/g' | cut -f 4 -d
    ↪ ';' ;'
```

Année	Sac de mots	Filtres	Score MAP
2016	Lemmes	Mots outils, ≤ 2	0.7573
2016	Lemmes	Pas de filtre	0.7505
2016	Lemmes	≤ 2	0.7503
2016	Lemmes	Mots outils	0.7350
2016	Tokens	≤ 2	0.7329
2016	Tokens	Mots outils, ≤ 2	0.7297
2016	Textes des entités nommées	Mots outils, ≤ 2	0.7283
2016	Textes des entités nommées	Mots outils	0.7281
2016	Textes des entités nommées	≤ 2	0.7205
2016	Textes des entités nommées	Pas de filtre	0.7203
2016	Tokens	Mots outils	0.7099
2016	Tokens	Pas de filtre	0.7060
2016	Étiquettes des entités nommées	Mots outils	0.6738
2016	Étiquettes des entités nommées	≤ 2	0.6738
2016	Étiquettes des entités nommées	Mots outils, ≤ 2	0.6738
2016	Étiquettes des entités nommées	Pas de filtre	0.6738
2017	Lemmes	≤ 2	0.4503
2017	Lemmes	Mots outils, ≤ 2	0.4451
2017	Lemmes	Pas de filtre	0.4408
2017	Tokens	Mots outils, ≤ 2	0.4372
2017	Tokens	≤ 2	0.4367
2017	Tokens	Mots outils	0.4339
2017	Lemmes	Mots outils	0.4338
2017	Tokens	Pas de filtre	0.4314
2017	Textes des entités nommées	Mots outils, ≤ 2	0.3979
2017	Textes des entités nommées	≤ 2	0.3976
2017	Textes des entités nommées	Mots outils	0.3975
2017	Textes des entités nommées	Pas de filtre	0.3972
2017	Étiquettes des entités nommées	Mots outils	0.3459
2017	Étiquettes des entités nommées	≤ 2	0.3459
2017	Étiquettes des entités nommées	Mots outils, ≤ 2	0.3459
2017	Étiquettes des entités nommées	Pas de filtre	0.3459

Année	Score MAP baseline
2016	0.7475
2017	0.4185