NLP 2016/2017 - Homework 3

Marco Favorito 1609890

June 12, 2017

1 Introduction

N.B.: the report is a bit longer than 4 pages (consider the Appendix as a plus, if you want you can ignore it); I hope this will not be a problem.

1.1 Problem explanation

The problem belongs to the topic of ”Information Extraction”, a well-known task in NLP and MR. The assigned task consists in the extraction of triples, from a text corpus of knowledge (e.g. Wikipedia pages), in the form (p_1, r, p_2) where p_1 and p_2 are concepts (possibly disambiguated from a lexicalized semantic network like BabelNet) and r is a relation semantically suited for p_1 and p_2. In this specific case, the target relations are given as input, and the task is to find them in semantically annotated Wikipedia articles. A parallel task consists in the generation of question-answer pairs from the extracted triples, after the designing of question pattern for each target relation.

1.2 Proposed solution: high-level description

In order to solve this problem, I drew a lot of inspiration from some of the key concepts of the three suggested papers: [1], [2] and [3]. My approach can be summarized in the following points:

1. Define the target relations and, for each of them, define also examples of ”relational phrase” that best represent possible occurrences of the relation in the corpus (call them *seeds*, as described in the introduction of [2]).
2. For each encountered sentence, build its syntactic-semantic graph (as described in [1]), combining both the output of the syntactic parser and the semantic analyzer.
3. Compute all the shortest paths between every node with a concept. Then, filter out paths according to some criteria, such as ”contains at least one verb” or ”has subject and object”.

4. For every path, compute a similarity measure with every seed phrase. If the path filter those path which do not reach a certain similarity threshold, reject those relation instances (inspired by [3]).

For the question-answer pairs: as suggested by you, I designed question patterns for open answers and other ones for boolean answers, the latter type populated both with positive and negative cases. For their creation, I used heuristics such as employing search engines for searching occurrences of questions related to the target relation; or simply my knowledge of the language. I tried to make them the most generic as possible, in order to make them to fit with almost any relation instance.

2 Information Extraction

The IE task is quite hard: you have to deal with a lot of difficulties, from a wide range of factors.

Support and text format First of all, the format and the support of the text from which you intend to extract informations. The provided dataset, for instance, is very big in terms of disk space, and the only possibility to do all in local is to read it already compressed. However, reading from secondary storage costs a lot of execution time. Another solution might be to crawl the Wikipedia articles via web, but you have to deal with other format issues. In our case, the text format is decent, even if occasionally happens that an unexpected text formatting or whatever such as different encoding or typos can affect the stability and correctness of the program.

Syntactic and semantic parsers precision The precision of lower building block of the majority of IE solutions, i.e. syntactic parsers, semantic analyzers/annotations. If one of them fails considerably (e.g. a sentence badly syntactic-parsed or wrong semantic annotations) can yield a totally wrong output. The possible English statements are so many and so ambiguous, and the scenario above described turns out to happen quite often, hence reducing the performance of the procedure.
In my solution I used spaCy[1] for the syntactic parsing and the computation of similarity, for which it employs Word2vec. For semantic analysis, I used only the annotations provided in the dataset: in order to improvement system performances, it might be helpful to use an online semantic analyzer (e.g. Babelfy) that gives more detailed and updated informations.

Scalability and generalization If one want to find instances of a specific relation, he might be tempted to use relation-based rules, based on the fact that generally relations occurs in similar contexts, and then enumerate as much as possible these contexts and retrieve instances with high precision. This approach, however, is poorly scalable, not

[1]spaCy is a free open-source library featuring state-of-the-art speed and accuracy and a powerful Python API. Site: https://spacy.io/

exportable to other languages and agnostic to unseen patterns and language evolutions. I retain that my approach satisfies scalability and easy translation, since the only input required, beside the corpus, is only the seeds for the target relations; moreover, it deals with unseen patterns by employing the similiarity measure provided by spaCy. Indeed, the seeds are *semantically* compared with the candidate relation instances.

3 Triple Mapping

As I stated before, every sentence is syntactically and semantically parsed. In the sentence there must be at least one subject[2]. Then, over the shortest paths, there are applied some filtering rules:

- the path has at least one verb;
- the path has concepts in the ends;
- the first token is a subject and the last one is an object[3]

It is worth to mention two tricks used in the implementation. The first is: take only the first sentences (let's say, the first five) from a Wikipedia article. This makes the search more accurate, since those sentences are more likely to be defined "definitional knowledge", and so better suitable for our purposes[4]. The second one is: take the concept of the Wikipedia page from the first sentence and, in the following sentences, replace the subject concept with the concept of the page. I heuristically checked that, in the vast majority of the cases, this approach yields good sentences, and allow us to exploit a lot of sentences that otherwise would be "wasted".
Now it is time to map the extracted triples. As I mentioned above, I used the similarity function provided by spaCy APIs: I look for any seed that has a similarity score more than a threshold value[5] for the shortest path. If the new relational phrase is compliant with the seeds and the similarity measure of a certain relation, I add the new relational phrase to the set of seeds of that relation, such that it contributes to searching for other similar relation instances. In order to prevent semantic drift, the similarity computed with the new seeds is multiplied by the similarity with which they joined in the seeds set, such that the threshold increases implicitly and we avoid relational phrases too distant from the original seeds.

4 Q/A generation

The question-answer pairs are generated dynamically, processing the relation instances in micro-batch fashion, online with the search. When a fixed-size buffer containing relation

[2]For the program, a token in the graph is a subject if the dependency tag is either "nsubj" or "nsubjpass"
[3]I.e.: the POS tag is in ["pobj", "cobj", "dobj", "attr"]
[4]According to the Wikipedia guidelines, an article should begin with a short declarative sentence, defining what (or who) is the subject and why it is notable [1].
[5]Defined as input, or set at default value 0.9

instances for a given relation is filled, the Q/A pairs generation starts automatically, iterating over the buffered relation instances and the provided question patterns. For each question pattern, we replace the left placeholder ”X” with the left concept mention in the question string and the right concept still in the question string if the question pattern is binary, otherwise we report it in the answer. For each binary question pattern I instantiate a positive and a negative Q/A pair by picking, respectively, the correct right concept and a right concept picked uniformly at random among the other right concepts.

5 Discussion and Conclusion

I designed the proposed solution in order to make it the most simple and abstract as possible, thus easily scalable and extensible to other relations and to other languages. The only requirement is to define a set of *seeds*, i.e. examples of relational phrases for the target relation. A similarity threshold for each set of seeds can be defined for tuning the search: the higher, the better the precision, but also the lower the recall; however, this component is fundamental for a good approach, since it achieve seed generalization and improves the search. Errors in the yielded relation instances are mainly due to bad semantic annotations and fails of the syntactic parser.
Although my approach is quite straightforward at high-level, I had to cope with a lot of implementation details that I cannot describe in 3/4 pages. I tried to produce adequate documentation for the code. I suggest you to start from `src/README`.
In Table 1 there are some numbers about the delivery. Notice that those number are computed from a ”biased” sample of the corpus: firstly I scanned every document of the corpus looking for some key words correlated to the target relations; then, in order to retrieve instances for a certain relation, I analyzed only documents previously filtered (i.e. containing at least one key word). It is an heuristics that helped me a lot for validate and test the program (especially for the rarest relations, such as ”smell” and ”taste”), and obviously it has not affected performances on the whole corpus. Analyze the whole corpus is too expensive in terms of time.

5.1 Further improvements

Nothing is perfect, and surely my work will not be the first. Here I tried to summarize pros and cons, looking for further improvements.

- Until now, the user has to manually define seeds for the target relation; a great progress can be made if it is implemented an automatic way to retrieve those seeds only from key words or relation name, for example from a structured knowledge source (e.g. Wikidata) and example sentences.
- Similarity threshold might be estimated and validated in an automatic fashion using some hypernym analyzer for filter out, from a sample, relation instances with infrequent categories.

- Also the number of first sentences of Wikipedia article is a tunable parameter: it can be set dynamically during the analysis, for example it might be increased if the ratio between retrieved sentences and analyzed pages goes under a certain threshold.
- My efforts have been dedicated to make a generic approach, trying to keep the design simple and to employ the lowest number of resources, also constrained to the provided dataset and annotations. Maybe the employing of a search engine for retrieve documents by queries suited for the target relations and the use of an online semantic analyzer with more informations about the disambiguated text (e.g.hyponymy, synset, disambiguation score) might inspire new approaches and developments, hopefully with better results.

Relation	#Instances	#Q/A pairs	Sim. threshold
place	2263	8272	0.93
generalization	66619	305305	0.92
material	5759	21072	0.98
smell	75	330	0.91
taste	22	110	0.92

Table 1: Here some statistics about the extracted triples and the question/answer pairs divided by relation. Similarity thresholds for each relation are set experimentally.

Appendix A

Designed seeds

The format for the seed file, as described in `src/README`, is the following:

```
relationName1:seed1,seed2,...,seedN[_similarityThreshold];
relationName2:seed1,seed2,...,seedN[_similarityThreshold];
...
```

Now I report the set of seeds used for the assigned relation:

```
place:located in,can be found in,placed in_0.93;
material:made out of,made of,made from_0.98;
generalization:is,is a,is an,is a kind of,is an example of_0.92;
smell:
    smells like,
    smell like,
    have an odor,
    has an odor similar to,
    has an odor suggestive of,
    has a distinctive smell of,
    smells similar to,
    bears a smell of,
    has an odor suggestive of,
    emit odor similar to,
    has smell reminiscent of_0.91;
taste:
    tastes like,
    taste like,
    has a taste similar to,
    has similar taste to,
    taste and smell like_0.92;
```

Notice: the parser strips new lines, tabs and space, both left and right, for every expression.

Learned relational phrases

In this section I list by relation some examples of learned relational phrases, the relative similarity score and the original seed, during the analysis:

- generalization:
  - `is one of`, learned from `is a kind of`, with score 0.942742136649
  - `is example of`, learned from `is an example of`, with score 0.966127138644
  - `be kind of`, learned from `is a kind of`, with score 0.928349554723

- place:
  - `located in Town in`, learned from `located in`, with score 0.946531268056
  - `be in`, learned from `can be found in`, with score 0.935142774674
  - `located in vicinity of`, learned from `located in`, with score 0.931529674734
  - `situated in Chiniot in`, learned from `located in`, with score 0.936133710621
- smell:
  - `has odor similar to` learned from `has an odor similar to`, with score 0.969229893713
  - `is liquid with odor similar to that of`, learned from `has an odor similar to`, with score 0.92144254913
  - `has odor reminiscent of`, learned from `has smell reminiscent of`, with score 0.941676403815
  - `has smell of`, learned from `has a distinctive smell of`, with score 0.918841982886
  - `taste smell like`, learned from `smell like`, with score 0.950404015742
- taste:
  - `has taste like` learned from `taste like`, with score 0.930043061233
  - `has taste akin to`, learned from `has a taste similar to`, with score 0.916158656076
  - `taste more like`, learned from `taste like`, with score 0.943789646138

References

[1] Claudio Delli Bovi, Luca Telesca, and Roberto Navigli. "Large-Scale Information Extraction from Textual Definitions through Deep Syntactic and Semantic Analysis". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 529–543. ISSN: 2307-387X. URL: `https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/660`.

[2] Mike Mintz et al. "Distant Supervision for Relation Extraction Without Labeled Data". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*. ACL '09. Suntec, Singapore: Association for Computational Linguistics, 2009, pp. 1003–1011. ISBN: 978-1-932432-46-6. URL: `http://dl.acm.org/citation.cfm?id=1690219.1690287`.

[3] Andrea Moro and Roberto Navigli. "Integrating Syntactic and Semantic Analysis into the Open Information Extraction Paradigm". In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI '13. Beijing, China: AAAI Press, 2013, pp. 2148–2154. ISBN: 978-1-57735-633-2. URL: `http://dl.acm.org/citation.cfm?id=2540128.2540437`.