1 Problem description

In this assignment, you will develop your own algorithm to *extract* commonsense knowledge (CSK) from general web contents. Although contents from the web are generally noisy, recent researches [1, 2] have shown that using judicious filtering, cleaning and ranking, extracting concept-centric CSK from web contents can be achieved with high precision and recall.

Compared to other methods such as crowdsourcing CSKB construction (e.g., ConceptNet [3], ATOMIC [4]) or symbolic knowledge distillation from GPT-3 [5], extracting CSK from web contents requires less monetary costs. More importantly, as we iterate over several web documents which could contain repeated information, extracting CSK from these contents will also provide frequency of extracted assertions, which is an important signal for assertion ranking.

Your task is to develop an algorithm to extract what food is eaten by a given animal, i.e., extracting the possible objects of the triple (<animal>, eats, ?). For each animal, you will be given a not-so-small set of 500 documents related to that animal. The documents were collected automatically by (i) querying the Bing Search API with the pattern "<animal> animal facts", and (ii) scraping contents using the `newspaper3k` library[1]. As the data was collected automatically, there are uninformative or empty documents (e.g., because of errors occurring during scraping), or irrelevant documents (e.g., documents about "elephant seal" mixed with the "elephant" documents).

Your algorithm should iterate through all documents, find and extract the relevant information. For instance, for the subject "elephant", the following sentences are the ones that contain relevant information:

1. *Elephants are herbivorous and will eat leaves, twigs, fruit, bark, grass and roots.*[2]
2. *Elephants consume grasses, small plants, bushes, fruit, twigs, tree bark, and roots.*[3]
3. *Elephant is a herbivore. Its diet is based on grass, herbs, fruit, bark and leaves.*[4]

You algorithm should not only extract the correct objects, but also produce the frequencies of those objects. You should output a list, where each element is a tuple of the name of the food and its frequency. The output list should be sorted by frequency (from high to low). For example, from the three above example sentences, the ideal output is:

```
[
    ("fruit", 3),
    ("grasses", 3),  # ("grass", 3) is also accepted
    ("bark", 2),
    ("leaves", 2),
    ("roots", 2),
    ("twigs", 2),
    ("bushes", 1),
    ("herbs", 1),
    ("small plants", 1),
    ("tree bark", 1)
]
```

The extracted objects should be either single nouns (e.g., "grasses") or short noun phrases (e.g., "small plants", "tree bark"), but should not be long lists of nouns (e.g., "grass, herbs, fruit, bark and leaves"). Your algorithm should be able to identify singular and plural forms of the same nouns, i.e., "grass" and "grasses" should be grouped. You should also filter out too general objects such as (elephant, eats, food) or (elephant, eats, things), and avoid nonsensical extractions such as (elephant, eats, what) or (elephant, eats, etc.).

Note that you are not allowed to use any existing commonsense knowledge base (e.g., ConceptNet, AscentKB) as they already contain those assertions.

[1] https://github.com/codelucas/newspaper
[2] https://en.wikipedia.org/wiki/Elephant
[3] http://justfunfacts.com/interesting-facts-about-elephants/
[4] https://www.softschools.com/facts/animals/asian_elephant_facts/2310/

2 Evaluation

We will evaluate your output on three metrics: *Precision*@30, *Recall*@30 and *F*1@30, as is the maximal number of objects for a subject in our private test data. Hence, it is important that you rank your extracted objects by frequency as we will only consider the top-30 extractions. The ground truths were collected semi-automatically using OpenIE and manual verification. We will use lemmatized-head-word matching to identify correct answers.

For each animal, the evaluation metrics are given by the following formulae:

$$true\_positive@30 = len(intersection(predictions[:30], ground\_truths))$$

$$Precision@30 = \frac{true\_positive@30}{min(30, len(predictions))}$$

$$Recall@30 = \frac{true\_positive@30}{len(ground\_truths)}$$

$$F1@30 = \frac{2 \times Precision@30 \times Recall@30}{Precision@30 + Recall@30}$$

3 Code skeleton, baseline and suggestions

3.1 Code skeleton

The code skeleton can be found in this GitHub repository[5]. Your algorithm should be implemented in the `your_solution` function in the `solution.py` file. You are free to write your supporting functions, either in the `solution.py` file or new Python files. The inputs of the `your_solution` function are:

1. `animal` (str): the animal;
2. `doc_list` (list[dict[str, str]]): the list of retrieved documents for that animal, each is a dictionary containing 4 keys: `animal`, `url`, `title` and `text`. For example, to access the content of the first document in the list, simply call `doc_list[0]["text"]`.

This function should return a list of tuples, each has two elements: the name of the extracted food (str) and its frequency (int). The list should be sorted by frequency, from high to low. We will only consider the first 30 elements of the list for automated evaluation.

We also provide you with a set of public tests which contain 5 animals (see the descriptions of the GitHub repository for more information). The private tests will have at least 5 other animals.

3.2 Baseline

In the `baseline.py` file, you will find a baseline method implemented by the tutor using the rule-based matching methods of SpaCy[6]. This function searches in the given texts for matches of a simple pattern: "<animal>s* eats* <noun>", then extracts the nouns as results. This simple method gives a micro F1-score of 0.176 on the public test data. Please read the code of this function as well as the description of the GitHub repository for how to run the baseline.

3.3 Suggested approach

There are several possible approaches to this problem. Here, we briefly introduce three popular ones that you might consider for your algorithm: pattern-based matching, OpenIE and extractive question answering (or span extraction).

Pattern-based matching is the simplest method which can achieve high precision with carefully-designed patterns and judicious cleaning. However, it is not easy to scale up this method, because for each predicate,

[5] `https://github.com/phongnt570/akbc22-lab07`
[6] `https://spacy.io/usage/rule-based-matching`

one will need to design a different set of patterns. Furthermore, to reach a good recall, one should spend reasonable efforts to write a diverse set of patterns in order to cover a good amount of cases. For example, the only pattern used by the baseline algorithm will not match any of the three example sentences in Section 1. You could look into the provided data and write more patterns to improve the recall of the baseline algorithm.

OpenIE could be used to overcome the shortage of coverage. The advantage of OpenIE is that it does not depend on predefined predicates as it will (ideally) extract all triples mentioned in the input text. However, as existing OpenIE systems have different rules for the extracted triples, one will usually need to post-process the extractions to get desired outputs (e.g., simplifying or splitting the objects). You could use any existing OpenIE methods to solve this assignment. You are more than welcome to use your own OpenIE method from the last assignment.

Extractive QA is the task of extracting answer for a given question from a given context. In other words, given a question and a context, an extractive QA system will answer that question by returning a text span that is directly taken from the context. There are several pretrained QA models that can be found on the HuggingFace website[7]. For example, using the `deepset/roberta-large-squad2` model[8], one can get the result in Figure 1. Since these models extract a text span from the given context, one will often have to post-process it to remove irrelevant information, or to split it into a list of objects. Sometimes, due to the model's capability, it even returns wrong or less-than-enough answers (e.g., try the same input with the `distilbert-base-cased-distilled-squad` model[9]). Another important aspect to consider about this approach is the processing time. Because the models are BERT-like models, it usually takes longer than running the rule-based approaches. If you are going to use this approach, you should pay attention to the trade-off between the processing time and (potential) gains in performance.

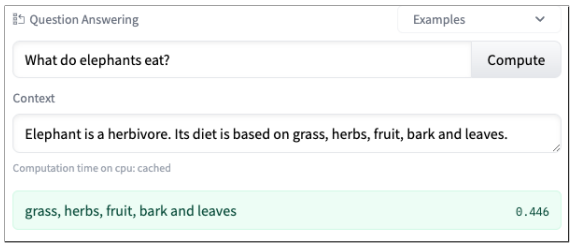

Figure 1: An example using extractive QA.

4 Submission

Please implement your algorithm in the `your_solution` function in the `solution.py` file. Your submission should only contain `solution.py` and other supporting files (e.g., your supporting Python files, model files, etc.) that you have added. You should not include the `public_test` folder or other already provided Python scripts (e.g., `main.py`, `evaluate.py`, etc.).

Please submit all necessary files, which are compressed into a zip file named:
Lab07_MatriculationNumber_Name.zip
to the email address: **akbc-assignments@mpi-inf.mpg.de** with title of the email: **[AKBC]Lab07_MatriculationNumber_Name**

Deadline: 23:59 13.06.2022 (Monday)

[7] `https://huggingface.co/models?pipeline_tag=question-answering&sort=downloads`
[8] `https://huggingface.co/deepset/roberta-large-squad2`
[9] `https://huggingface.co/distilbert-base-cased-distilled-squad`

References

[1] T.-P. Nguyen, S. Razniewski, and G. Weikum, "Advanced semantics for commonsense knowledge extraction," in *Proceedings of the Web Conference 2021*, 2021, pp. 2636–2647. [Online]. Available: https://arxiv.org/pdf/2011.00905.pdf

[2] T.-P. Nguyen, S. Razniewski, J. Romero, and G. Weikum, "Refined commonsense knowledge from large-scale web contents," *arXiv preprint arXiv:2112.04596*, 2021. [Online]. Available: https://arxiv.org/pdf/2112.04596.pdf

[3] R. Speer, J. Chin, and C. Havasi, "Conceptnet 5.5: An open multilingual graph of general knowledge," in *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 4444–4451. [Online]. Available: https://arxiv.org/pdf/1612.03975.pdf

[4] M. Sap, R. Le Bras, E. Allaway, C. Bhagavatula, N. Lourie, H. Rashkin, B. Roof, N. A. Smith, and Y. Choi, "Atomic: An atlas of machine commonsense for if-then reasoning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 3027–3035. [Online]. Available: https://arxiv.org/pdf/1811.00146.pdf

[5] P. West, C. Bhagavatula, J. Hessel, J. D. Hwang, L. Jiang, R. L. Bras, X. Lu, S. Welleck, and Y. Choi, "Symbolic knowledge distillation: from general language models to commonsense models," *arXiv preprint arXiv:2110.07178*, 2021. [Online]. Available: https://arxiv.org/pdf/2110.07178.pdf