Corpus-Driven Fact Checking Engine

Nihal Yadalam Murali Kumar, Rahul Gururaja Rao, and Shivam Sharma

University of Paderborn
nihal@mail.uni-paderborn.de
rahulrao@mail.uni-paderborn.de
sshivam@mail.uni-paderborn.de

1 Problem Statement

Given a TSV file with facts from DBpedia, build a corpus-driven fact-checking engine which generates a TTL file with truth values between -1.0 (fact is false) and +1.0 (fact is true) for all the given facts.

2 Approach

Our approach involves three major steps:

- To separate subject, predicate and object in the form of a Triplet *(s,p,o)* from the fact statement.
- To retrieve the related data of input's subject.
- To check the correctness of the fact-based information extracted from the data.

We have explained our initial and final approaches for each step in the following sub sections and the challenges that we faced during implementation.

2.1 Initial Approach

1. Finding the subject, predicate, and object triplet from the input fact statement:
   - For extracting the *(s,p,o)* triplet from the input fact statement, we used Stanford's CoreNLP library.
   - The main attraction of CoreNLP is the pipeline (Figure 1) that it uses.
   - In CoreNLP library, we used Open Information Extraction (OpenIE) annotator specifically, which extracts the relation triplets in the form (subject, relation, object) from the statement. For Example, for the input statement *Albert Einstein's birthplace is Ulm*, OpenIE forms the triplet by identifying the subject, object and relation in the statement as shown in Figure 2.
   - This will give us the following triplets (Subject, Predicate, Object):
     - Albert Einstein, have, birthplace
     - Albert Einstein's birthplace, be, Ulm

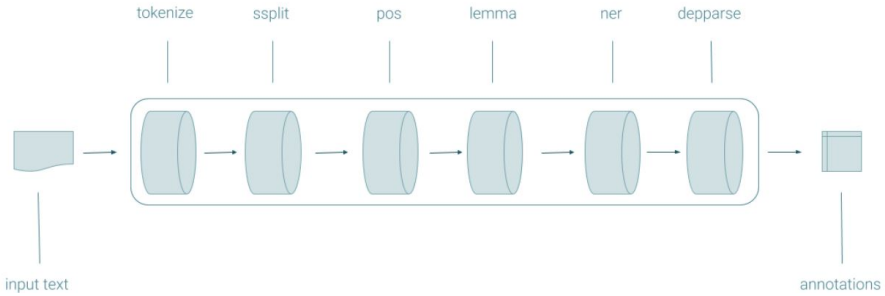

Fig. 1. Pipeline of CoreNLP Library
(`https://stanfordnlp.github.io/CoreNLP/`)

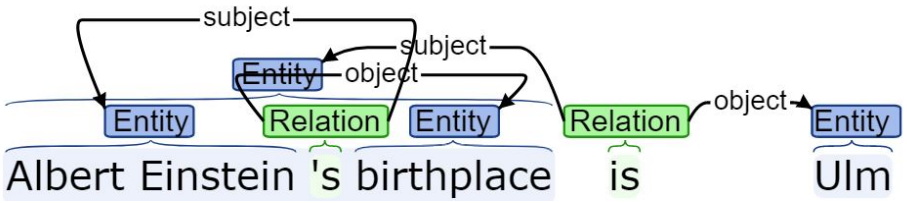

Fig. 2. Visualization of the OpenIE annotator
(`https://corenlp.run/`)

Therefore, the triplet is **(Albert Einstein, born, Ulm)**.

2. Retrieve the related data based on an extracted subject from the fact statement:
   - For this, we decided to use the DBpedia database API.
   - For our example of input statement: *Albert Einstein's birthplace is Ulm*, we use the URL `http://dbpedia.org/data/Albert_Einstein.json` to get all the data related to Albert Einstein
   - Our initial idea was to extract the information in a "*key*" : "*value*" pair of the important points in the returned JSON format.
3. Check the correctness of the fact statement
   - Based on DBpedia's extracted information, we check the *predicate* from the triples with against the *key* values of the corresponding subject.
   - If the *key* matches with the *predicate* then we check for the corresponding *value* with the *object* from the triples.
   - If they match, the value returned will be +1.0, meaning that the input fact is true. Otherwise, it will return −1.0, meaning that the fact is false.

2.2 Challenges in the initial approach

We faced the first challenge in extracting the subject, predicate, and object triplets from the fact sentence using CoreNLP's OpenIE.

- Stanford's CoreNLP OpenIE annotation was not able to classify correct subject and object in many fact statements.
- For getting a triplet for a single fact sentence, OpenIE took a lot of time and CPU RAM.
- As seen in the above example, some of the triplets were highly inaccurate. Thus it was difficult to find a relation between the subject and object.

The second challenge we faced was acquiring the related information returned from the DBpedia's API.

- In our initial approach, we decided to extract the information in a "*key*" : "*value*" pair. However, the output we got was filled with many useless tags and data. Finding and extracting the correct article became very difficult in this situation.
- So, we decided to extract the information from the infobox in the Wikipedia article related to the subject. For our example statement *Albert Einstein's birthplace is Ulm*, the related Wikipedia article URL is `https://en.wikipedia.org/wiki/Albert_Einstein`.
- This approach of extraction from infobox was more undemanding. We decided to get the infobox in a JSON format using the URL `https://en.wikipedia.org/w/api.php?action=query&prop=revisions&rvprop=content&format=json&titles=Albert%20Einstein` (for our example statement, we changed the *titles* parameter in the above URL to Albert Einstein) and used our initial approach of storing the information in "*key*" : "*value*" pair.
- However, upon observing the returned JSON format of the Wikipedia article, we found that the infobox's structure in the page keeps changing based on the category of different subjects. We also tried to get the infobox in an XML format but extracting from XML in a "*key*" : "*value*" format was difficult to generalize for each Wikipedia article.

We faced the final challenge in checking the correctness of the fact statement by comparing the triplets from the input fact statement with the infobox (extracted from the Wikipedia article).

- In some cases, there was no infobox in the Wikipedia article. So we will have to extract the paragraphs from the HTML document of the article and check the triplets within the paragraph.
- In other cases, the predicate that we got from the fact statement might not be the same as in the infobox.

2.3 Final Approach

After having the challenges (as mentioned earlier in our initial approach), we came up with our following final approaches for each challenge:

1. Since we discussed in Section 2.2 that CoreNLP was not suitable for getting the triplets, so we decided to separate the triplet from the fact statement manually. We used JAVA's provided string manipulation for extracting the triplets from the fact statement.
2. Before developing a concrete algorithm, we observed the training data set of facts and found that some sentences have patterns in them. We observed the following combination of the triplet patterns:
   (a) **Subject, Predicate, Object** Example:

Fact ID	Fact Statement
3867375	Robert De Niro's birth place is Greenwich Village.

   Extracted Triplet: Robert De Niro's *(Subject)*, birth place *(Predicate)*, Greenwich Village *(Object)*
   (b) **Object, Predicate, Subject** Example:

Fact ID	Fact Statement
3862445	EBay's subsidiary is Gumtree.

   Extracted Triplet: EBay *(Object)*, subsidiary *(Predicate)*, Gumtree *(Subject)*
   (c) **Object, Subject, Predicate** Example:

Fact ID	Fact Statement
3325169	Nobel Prize in Physiology or Medicine is George Minot's honour.

   Extracted Triplet: Nobel Prize in Physiology or Medicine *(Object)*, George Minot *(Subject)*, honour *(Predicate)*
3. However, we had to define the predicates in such a way that they match with the infobox row field. For example, we take the input statement *Nobel Peace Prize is Kailash Satyarthi's honour.* The extracted predicate *honour* is not a field in the subject's infobox, as shown below.
   Therefore, we had to set a *Wikipedia predicate* (Awards) to match the infobox field.
4. Initially, we investigate whether the input fact statement contains a predicate keyword (e.g., birthplace, death place, honour, stars, etc.) in the *getTriplet()* method. Once a suitable predicate keyword matches, the respective object and the subject are extracted from the statement alongwith an assigned *Wikipedia predicate* using string manipulation.

Born	Kailash Sharma[1][2] 11 January 1954 (age 67) Vidisha, Madhya Pradesh, India (present-day Madhya Pradesh India)
Nationality	Indian
Alma mater	Barkatullah University (B.E., M.E.) Alliance University (Honorary PhD)
Known for	Activism for children's rights and children's education
Spouse(s)	Sumedha Satyarthi
Awards	Nobel Peace Prize (2014) Robert F. Kennedy Human Rights Award (1995)[3]

Fig. 3. Infobox of Wikipedia article on *Kailash Satyarthi*

5. We set the value of an inversion flag to *false* for the cases where the fact statement is structured in the format of subject, predicate, object and *true* for the sentence format object, subject, and predicate. This logic extracts the triplets in the form of *(subject, Wikipedia predicate, object)*.
6. In some cases, our *Wikipedia predicate* did not match in the infobox rows. Therefore, we added an extra supporting predicate named as *Alternative predicate* to improve the accuracy of results obtained in these exceptional cases. For example, the predicate *"Subsidiary"*, has an alternative predicate *"Owner"* and *"Stars"* predicate, has an alternative predicate *"Known for"*.
7. After discussing our challenges regarding getting the related articles (in Section 2.2), we decide to get infobox as a text by parsing the HTML document of the Wikipedia article using the JSOUP library.
8. We use a REST API provided by Wikipedia to fetch a list of alternative URLs that points towards different Wikipedia pages related to the extracted subject.
9. The JSOUP library returns the entire web page information as a *Document* Object. This Document Object is used for fetching the infobox using HTML parsing. Each row in the infobox is stored as a string entry in a list. Also, this Document Object can be utilized to fetch the entire webpage information in paragraphs.
10. If the predicate and the object extracted from the fact statement are not found in the Wikipedia article, the next URL returned by *getAlternativeUrls()* is used for fetching the next document.
11. After getting the predicate, if the object is matched, value $+1.0$ is returned which indicates that the fact is true. Otherwise, value -1.0 is returned, indicating false. However, if the predicate is not matched from any of the

web pages acquired from the Alternate URLs, the fact returns a default value of 0.0, indicating that a relationship cannot be determined between object and subject extracted.

3 Experimental Results

After building the model as defined in Section 2.3, our model has achieved 82% area under the curve for **SNLP 2020 Test** dataset as shown in Figure 4.

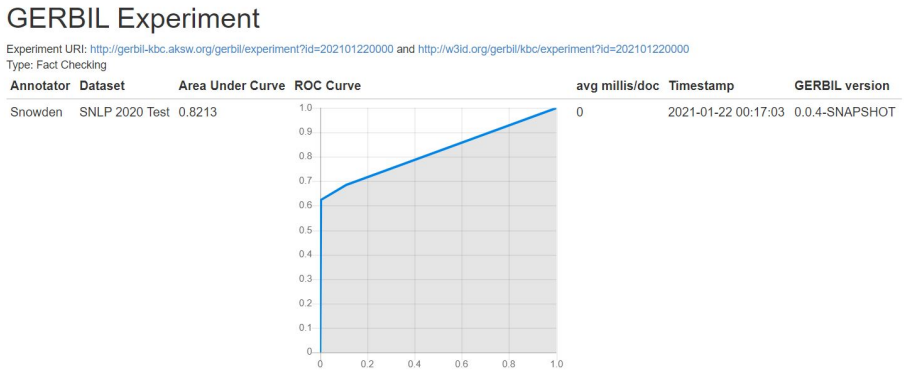

Fig. 4. ROC curve for dataset SNLP 2020 Test

4 Advantages and Disadvantages

– Pros:
  - Since the extraction of Subject, Predicate, and Object has been done manually without using the external libraries like CopeNLP, our algorithm took a lot less time to execute all the facts than the amount of time and memory required by the CoreNLP library took. String manipulation requires far less time compared to these libraries.
  - Accent and language-specific characters are well handled in the preprocessing of the subject before calling the list of Wikipedia URLs. If the extracted subject contains certain characters like the german umlauts (e.g., ä, ü, ö, etc.), the Danish characters (e.g., Ø, etc.) or the Slavic characters (e.g., Ł, etc.), the REST API list will replace these with equivalent English characters.
  - If subject in the extracted triplet is not perfectly produced by the *getTriplet()* method, then *getAlternativeUrls()* method provides the correct list of URLs for any related subject.
  - Our model gives a very less number of false positive results.

– Cons: Our current model does not work for the following examples:

Sr. No.	Fact Statement
1	Tyson Jolly's team is SMU Mustangs men's basketball.
2	Amitabh Bachchan's award is Padma Shri.
3	Christiano Ronaldo's national team is Portugal.
4	Tyrese Haliburton's team is Sacramento Kings men's basketball.
5	Lionel Messi's national team is Argentina.
6	Semi Ojeleye's team is Boston Celtics.
7	Anushka Sharma spouse is Virat Kohli.
8	James michael tyler stars in Friends.
9	Kevin Garnett's team is Brooklyn Nets men's basketball.
10	Shake Milton's team is Philadelphia 76ers men's basketball.

The main reason behind the model's failure for these examples is that some of the infoboxes are not structured in the way we extracted them using JSOUP. If we try to reformat the infobox for all specific cases then there is a chance of over fitting the data.

5 Flowchart

Figure 5, describes our algorithm's flow and the way it returns the output value by checking the triplets from the fact statement to the information extracted through JSOUP.

6 Future Work

Our algorithm can be updated to cover examples with more specific infobox structure. One possibility is to parse the infobox structure such that the titles in the infobox (with a list underneath) can be identified as a criteria to classify the object. Instead of hard coding the different possibilities or synonyms of a predicate keyword in *alternatePredicate* field, we can use a word net model to find a list of synonyms which can be used as alternate predicate to increase the accuracy of algorithm.

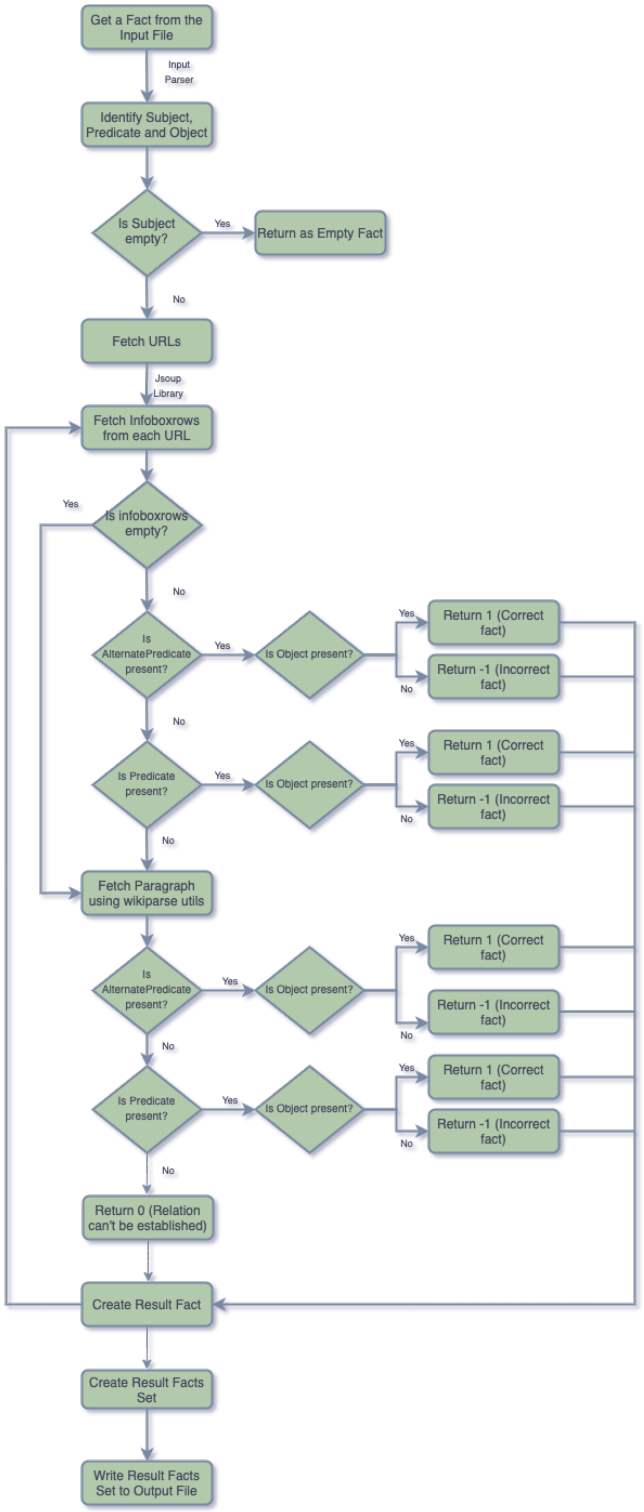

Fig. 5. Flowchart of the algorithm