IntelligentFactChecker

Akshit Bhatia, Nalanda Chakrabarti and Muhammad Faizan Batra

Universität Paderborn

1 Problem Definition

Build a corpus-driven fact-checking engine, which returns a confidence value between -1 (fact is false) and +1 (fact is true) given a fact from DBpedia.

2 Approach

1. **Finding Subject, object and Predicate in the sentence**

   (a) **Find Predicates**: Based on the training data, we selected certain predicates which are used in majority. As per the infobox on wikipedia, we know that all the data are arranged based on some keywords. For example, for wife, the infobox uses keyword: 'spouse'. So, we look out for those words in a sentence and remove it from that sentence. Thus, our new sentence would contain a subject and an object along with certain stopping words. The words that we remove, we store them in another variable called **predicate**.
   Different Predicates: We can have predicates of two types. A 'One word' predicate or 'two worded' predicates. For example: spouse, author, starring etc are one word predicate. While birth place, death place, prime minister etc are 'two worded' predicates. While searching 'two worded' predicates, to do it efficiently, we searched only for the word 'place' in the sentence and then we selected the word that is indexed one word ahead of the word 'place'. That word would tell us the predicate that has to be used. i.e if it is birth place in the sentence, then the word "birth", would let us know that the predicate belongs to "born".
   These are the predicates that we have used: born, died, spouse, starring, innovation place, award, team, author, prime minister.

   (b) **Separate Subject from Object**: On observing the training data, we could see that majority of sentences has **is** in the sentence. Then there were some sentences with **'s**. And at last there are sentences which has neither **is** nor **'s**. Using this we were able to determine how to separate 'subject' from 'object' in the sentence. We didn't remove any stop words because all of those stop words are important too.
      i. **Sentences with "'s" in them:** The word with "**'s**" is the subject that we searched on Wikipedia. We replaced **'s** with **zzz** for those

words. Please note that "'s" can also occur in the sentences with "is" in them. So this step is only to recognise 'Subject' and it aids the pre-processing of the sentences that has 'is' in them, because, without 's, we take the words before 'is' as Subject.

ii. **Sentences with "is" in them:** The sentences with **'is'** in them, can be split upon the 'is'. So the words before 'is' belongs to the *'Subject'* that would be searched on internet while others would belong to the *'Object'*. But if the sentence also contained "'s",(which we replaced with "zzz"), then the word marked with "'s"("zzz" in the code) is selected as the 'Subject'.
Note: We selected words with "'s" as subject because, in one of the example in training file, a noun has no references in the Wikipedia but the noun marked with 's has a page in Wikipedia.

iii. **Neither "is" nor "'s":** For the sentences that have neither **"is"** nor **"'s"**, for example, Camp Rock stars Nick Jonas, in them, we remove the predicate and simply divide the sentence into 2 parts.

(c) **Remove unnecessary symbols** : We remove all the unnecessary symbols like brackets, full-stop or comma from the sentence. This step is done after replacing the "'s" with "zzz".

(d) **Searching through Internet** : After performing the above three steps, we are left with a tuple with Subject, Object and a Predicate. For searching, we use **Wikipedia Infobox**. The class *FetchOnline.java* is responsible for doing this. The next section describes this approach in details.

(e) **Checking the Reality of Facts:** We used **Levenshtein Distance** to compare two objects. We used this, as there is a possibility that on internet, the word is not in its stemmed form but with us, we may have the root word. The Levenshtein distance, will calculate the number of changes to make both words similar. So it was the best approach.
We also thought of using the Jaccard Similarity, but that will require Shingling of tokens, and that will be waste of efficiency for the program. As we know that amount of data will involve maximum of 5 words on each side(usually), so using tfidf and cosine metrics would have also been a misadventure. Thus Levenshtein distance was best choice.

2. **Searching through Wikipedia**

(a) **Library used** : To access Wikipedia, we used JSOUP as it is easier to use and there is no need to import any API framework. To access information, Wikipedia has provided several endpoints that can be used by the developers. Following URL is used as an endpoint.
URL: https://en.wikipedia.org/w/index.php?search=<XXX> with XXX being the keyword, we can search information related to this keyword. From this information, we searched the elements that contain the word

"Infobox".

(b) **Differentiating between Subject and Object** : As described in Section 1, we have mentioned how are we differentiating between 'subject' and 'object'. The variable in the code **QueryToSearch** represents the Subject. And the variable **QueryToMatch** is the Object.

(c) **Calculate the edit distance** : Now that we have the Wikipedia result for our predicate, we compare that information against the object that we have. We determine the length of object, and then for every sentence searched from Wikipedia, we compare as many words from that sentence as there are in the object. As mentioned earlier, Levenshtein Distance metric is used here.

(d) **Normalising the edit distance** : The Levenshtein Distance only gives the number of changes required. We used the normalising formula to convert that distance into the error measure in percentage. Thus 1 minus this measure will tell us our confidence that the information is similar to the object and hence we can tell if the fact is true or false.

3 Result

Taking the approach defined above, our model has done a pretty good job. It wasn't a tough program to code and it incorporates an intelligent way to divide the sentence into predicates, object and subject. Based on the test data, our result have been around 70%. Below is the result AUC.

URL: http://swc2017.aksw.org/gerbil/experiment?id=202001190004

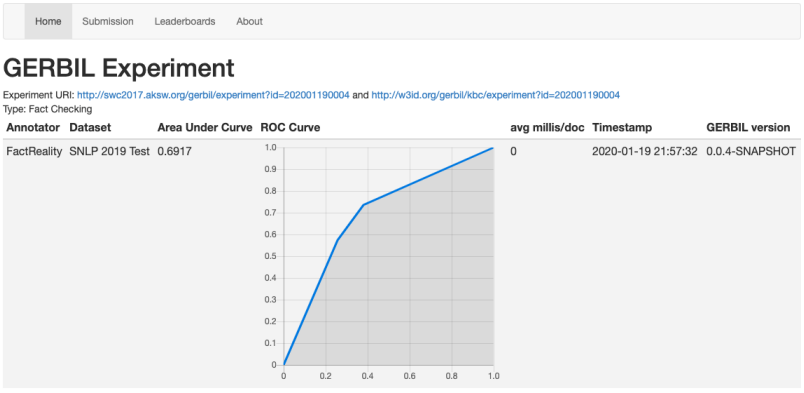

Fig. 1. Area Under Curve - 0.69

4 Pros And Cons of our Approach:

1. **Pros:** The time for processing the test data is reasonably low, that is around 6 minutes for all the facts. The implementation is very simple and it only uses JSOUP as an external library for connecting to the Wikipedia.
   Earlier, we were thinking about using a generic file containing all the verbs in English language. But then that would have involved searching for those verbs in the Fact Sentence. But as per the requirement, which can be recognised from the training file, only selected predicates are needed for the model to work on. That's why, hard-coding those predicates suffices the accuracy of the model.

   Below table represents some of the facts classified properly.

FactId	Fact_Statement
3407450	Rolandas Paksas' office is Lithuania.
3857435	London is Hawley Harvey Crippen's nascence place.
4304857	Bochum is G Data's innovation place.
3610986	Kill Bill stars Seth Gree.

Table 1. Working Example.

2. **Cons:** These are some cases that we have identified where our Model has not performed. We have also mentioned the reason for this bad behaviour.
   (a) **Subject does not matches the Title of the Wikipedia page:** If the Wikipedia page is titled ¡XXX¿ and our query is ¡XXXX¿, then the model would fail. Below is the table of facts representing this scenario.

FactId	**Fact_Statement**	**Wikipedia Title**	**Query**
3236656	Danuel House Jr.'s team is Houston Rockets.	Danuel House	danuel house jr
3215075	Otto Porter Jr.'s team is Chicago Bulls	Otto Porter	otto porter jr

Table 2. Wrong Title Examples

   (b) **Infobox is not present on Wikipedia:** In this case, for the search query, there is no Infobox present. Hence, our model fails here. Below is the table of facts representing this scenario.

FactId	**Fact_Statement**	**Wikipedia Title**	**Query**
3821086	Cole Porter's death place is Santa Monica, California.	Cole Porter	cole porter

Table 3. No Infobox

   (c) **Character Removed during Preprocessing:** During preprocessing, we removed the unwanted characters from the text. Thus, the alphabets with umlaut symbols, ampersand, brackets etc are removed. These characters were important also as Wikipedia contains the title of the page with these characters for some facts.
   Below table represents the scenario.

FactId	Fact_Statement	Wikipedia Title	Query
3413620	Alexis Valdés' spouse is Paulina Gálvez.	Alexis Valdés	alexis valds
3439535	The Great Train Robbery (novel)'s author is Michael Crichton.	The Great Train Robbery (novel)	the great train robbery novel
3826470	Imre Kertész's birth place is Nobel Prize in Literature.	Imre Kertész	imre kertsz
3816067	Opava is Eduard von Böhm-Ermolli's death place.	Eduard von Böhm-Ermolli	eduard von Bhmermolli

Table 4. Character Removed during Preprocessing

(d) **Subject not found on Wikipedia:** For these cases, 'subject' was found to have different spelling as on Wikipedia. Below table represents that scenario

FactId	Fact_Statement	Wikipedia Title	Query
4387619	SNP Schneider-Neureither & Partner AG's foundation place is Ottawa.	SNP Schneider-Neureither & Partner SE	snp schneiderneureither partner ag

Table 5. Subject not present on Wikipedia