Automatic Fact Verification

Dong Jia (960898) Xudong Ma (822009)

Codalab name: Dong Jia

Codalab name: xudongm

Codalab team name: FlyingDutchman

1 Introduction

This report is written for a project regarding automatic fact verification. As a starting point, we are provided with 109 files containing information from Wikipedia. Data inside these files are formatted as “document-title sentence-ID sentence”. On the other hand, a training and a development dataset which include a large amount of labelled claims are also available. Each labelled claim is proved with several evidence sentences. Our task is to retrieve Wikipedia’s information files to pick out sentences to support or refute claims inside a provided test set. Additionally, “NOT ENOUGH INFO” should be labelled to claims which can not find evidence for supporting or refuting. We will discuss about the processes and methods we choose to complete this task during the following parts of the report.

2 Methodology

In this section, we will introduce the whole procedure of automatic fact verification: document retrieval, sentence selection and fact verification.

2.1 Document retrieval

Index building: The Wikipedia articles are structured as document identity, sentence number and sentence text. The first step is to build index for all of them as they are numerous. Firstly, we use Pylucene[1] which is a common used information retrieval tools to construct index. However, Pylucene is relatively complex to use so on the basis of usability we apply lupyne which is a easy-used API of Pylucene.

Pylucene is a full-text search tool, Figure 1 shows the common procedures of it:

indexing and searching. Firstly, lupyne will do text analysis: tokenization, stop words (punctuations and some meaningless words like “in”, “at”) removement, words stemming (removing words inflection and derivation). These can be done by using engine.Indexer() in Lupyne. Next, inverted index will be constructed using key words obtained from last step. And we can also add “field” information indicating the position of each term, for example, title,content or url. “Filed” must be assigned using indexer.set(“Filedname”,engine.Field.Text,stored = True). In this way, we can decide which field to index. Wiki files will be traversed and added to indexer(). Page identity is stored as “filename” field and all sentences belong to this page are joined to a sentence and stored as “text” field. Then we need to use index.add() and index.commit() to finish index construction.

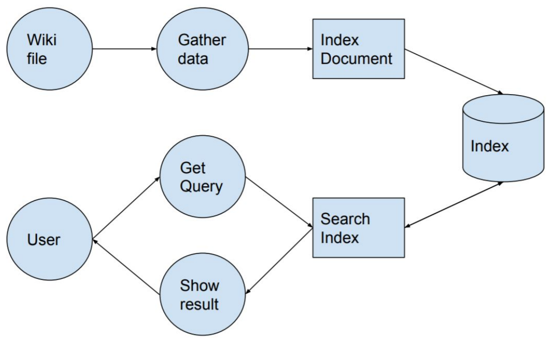

Figure 1 Pylucene workflow

Searching: Now we can search documents. This step need us to match given claims with the content of one or more Wiki file. A claim often include one or more entities. We have observed that claims and their relevant sentences usually have same entities from the training dataset. Namely, we can find entities in claims of test-unlabelled.json and then search wiki’s filename which has same entities. In order to find entities such as song titles, we used AllenNLP[2]. However, not all entities can be found, such as a claim “You Belong with Me is Taylor Swift’ s song”. AllenNLP can only extract “Taylor Swift” while we really need “You Belong with Me”. Therefore, we decide to use pos_tag() of nltk and chose all nouns combined with all terms with a capital letter as entities. “Fieldname” must be joined with entities to form a query string. It decides which field to index such as indexer.search(“filename:”+ query). Program will index “filename” other than “text”. Lupyne will return top k documents (similarities are calculated using tf-idf and lupyne can do it in the background). Figure 2 shows results of different index strategies.

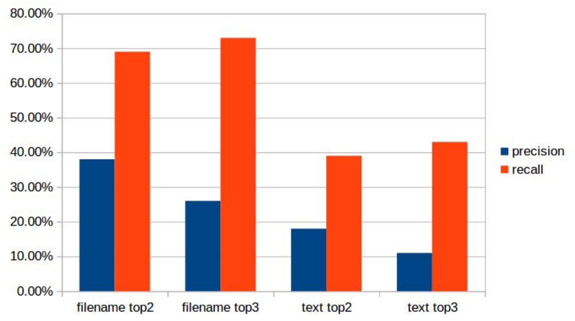

Figure 2 index strategies performances

We also try to index both “filename” and “text”, however, the results are not good, when We return top 3 documents for each of them, the precision value is only 23%

although the recall is 78%. After doing many experiments, we decide to index "filename" and return top 2 documents.

2.2 Sentence selection

As mentioned before, two documents are indexed with all of their sentences. In this section, We select some of these sentences which are most possibly related to a claim. We divide the section into two parts: tf-idf to narrow down the candidate sentences set and our label model is also used to filter useless sentences. We use TfidfVectorizer()[3] in sklearn to build a tf-idf matrix and then use cosine similarity to get the similarities between claim and each sentence. Then a threshold is set to filter some sentences with smaller similarities than the threshold. After doing many experiments, the threshold is set as 0.1 finally and we get a result with a sentence recall at 56.96% and document recall at 67.88%.

2.3 Fact verification

Fundamentally, we use a pretrained model from google Bert [4] as the starting point of doing fact verification. We retrain the pretrained model on our dataset. The training is done on sentence level and we would provide each pair of claim and evidence with a label "SUPPORTS", "REFUTES" or "NOT ENOUGH INFO". While, claims labelled with "NOT ENOUGH INFO" inside the training and development sets have no evidences which means there is no data for "NOT ENOUGH INFO" to learn. Therefore, we do a preprocessing on the training and development datasets by picking up the top 4 relevant sentences as the evidences for "NOT ENOUGH INFO" claims based on our sentence selection strategy. Finally, the retrained model is used to do a sentence level prediction on the test dataset. In order to make the prediction more reliable, we will ignore a pair of claim and sentence by labelling it as "NOT ENOUGH INFO" if the maximum classification probability of it is less than 0.5. Finally, those claims which are labelled "NOT ENOUGH INFO" by all its evidences would be labelled as "NOT ENOUGH INFO". Otherwise, claims would be labelled as "SUPPORTS" or "REFUTES" according to which one has more claim-sentence pairs labelled as. However, if "SUPPORTS" and "REFUTES" has a same number of supporting pairs, we will compare the summation of probabilities for claim-evidence pairs from each side and the class with a bigger summation will win the game. As a result, only the evidences of the winner will be output into the final output file.

3 Evaluation

As shown in table 3, our sentence selection (SS) strategy gets acceptable recall value on the development dataset. Besides, our fact verification model is on sentence level so that it can refilter sentences. As a result, the precision and F-score improves a lot after doing labelling compared to doing SS only. Finally, we apply both our sentence selection strategy and labelling model on the test dataset. The performance is also shown in table 3. It is clear that our system performs well both on the development and test dataset. Both our label accuracy and sentence F1 scores exceed not only the baseline of 20% but also the top level of

50%.

Evaluation Metrics	Dev set after SS	Dev set after labelling	Test set after labelling
Label Accuracy	100 %	65.07%	64.77%
Sentence Precision	25.22%	55.36%	55.22%
Sentence Recall	56.96%	51.46%	52.14%
Sentence F1	34.96%	53.34%	53.64%

Table 3: Performance of our system on test and development datasets

4 Error analysis

Our labelling model performs well on the development dataset which turns out a label accuracy at 85.72%. However, it decreases to 64.77% on the test dataset. We believe this is because our sentence selection strategy limits the performance of the labelling model. We find some possible reasons for bad performance of our sentence selection strategy.

Semantics absent: Tf-idf can’t efficiently reflect the importance of words and the distribution of key words. It measures the importance of a word by “word frequency” which is not comprehensive. Therefore, our recall value for sentence selection is only 56.96% which means that there are about 43% evidences are ignored by our sentence selection model.

.

Entity missing: All entities must be extracted to retrieve documents. However, not all entities appear in a claim. E.g. "David Beckham plays football.", and one of it’s evidence: "LA_Galaxy".

Disambiguation: Some document titles may have several disambiguation. We may need to return much more documents to find the intended title which makes sentence selection difficult. E.g. "Catching Fire has a cloud in it.". “Catching Fire” can be a science fiction or a novel.

5 Conclusion and Future Work

In conclusion, our classification model for labelling performs well. However, the sentence selection part needs some improvement. Therefore, if we have more time to do this project, we would like to try a sentence selection strategy based on different parts of speech and import some semantic analysis. On the other hand, the preprocessing and sentence selection part of our system may cost about 5 hours and the fact verification part may cost about 10 hours, therefore the whole process of our system can be finished within 24 hours in a device including only one GPU card, which can meet the project requirement.

Reference

[1] PyLucene http://lucene.apache.org/pylucene/.

[2] AllenNLP https://allennlp.org/

[3] Apt, K. R., & Kozen, D. (1986). Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett., 22(6), 307-309.

[4] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.