# COMP90042 Project 2019: Automatic Fact Verification

**Codalab Team: ALL_SUPPORTS**

**Minghang Chi <minghangc> (879544) | Yixiang Yang <yixiangy> (791962)**

## 1 Introduction

In this project, a basic Fact Extraction and Verification system is developed that can automatically validate whether a claim is true, false or unverifiable based on the information in a large text corpus. The task requires finding evidence sentences for given claim in the text corpus (wiki corpus here) so that enough supports can be generated to extract the validity of the claim. In our implementation, PyLucene is used for document indexing and relevant information retrieval (Apache Lucene, n.d.), while Bidirectional Encoder Representations from Transformers (BERT) takes the task of evidence generation and validity extraction (Devlin, Chang, Lee, & Toutanova, 2018). Eventually, a Codalab final result achieved 57.5% on label accuracy with 49.1% on document selection F1 and 37.7% on sentence selection F1.

## 2 Methodology and Implementation

In general, This Fact Extraction and Verification system implemented follows the underlying architecture of two core parts. First, an information retrieval helps to index our corpus, which allows fast retrieving the most relevant sentences given a specific claim. Second, feed the claims to be predicted to the trained model and recognize their textual entailment.

### 2.1 Document Indexing

An information retrieval system is the foundation of this whole project. As it enables us to query our claims to be answered in piles of wiki pages and consequently retrieves relevant resources that we need for next processing. Lucene is a completely Java written information retrieval software and currently has been ported to Python named PyLucene. We use Pylucene as a start. Each sentence of the corpus is indexed with its document name and sentence id as an identifier. BM25 (with k1=0.1 and b=0.1) is our choice of scoring formula instead of PyLucene's default fudging TF-IDF. The idea taking each sentence as a single input document rather than the complete document text under the same document name will be talked later.

### 2.2 Document Retrieval

In this module, the system retrieves the top-k (default k value = 10) sentences that best match the query claims. As in last step we use BM25 to construct indexes, BM25 (same parameter as before) is also adopted here. The k1 and b parameter are set both 0.1 respectively (Lu, Stephen, & Andrew, 2005). The value k1 is used to control the sensitivity of term frequency by formula. The value b control the range of what the length of document punished on weights. In a word, in this situation, term frequency is less essential and due to the length of document is merely short text that far less than a normal document. Therefore, k1 and b are set 0.1 respectively.

In case of that some relevant sentence might miss name entities from the claim, the system is able to score the similarity for each sentence measured in both sentence text itself and the document name. Even if a sentence only contains pronouns, still the relevant information can be found.

Traditional method on document retrieval that to find document containing the information of the given claim, and then extract facts in the form of sentences for the verification of the claim (Hanselowski, et al., 2018). In our design, considered time efficiency, we decided indexing sentences before we retrieve them. The reason will be discussed in the later section.

### 2.3 BERT

BERT is a method of pre-training language representations and the pre-trained model on a large text

corpus has been supported. Users can skip the annoying and expense pre-process on any corpus set. Rather than mainstream word embedding methods, word2vec, Glove and ELMo, BERT is used in our implementation. Compared to context-free models word2vec and Glove, BERT performed better on disambiguation as it generate the representation for each word is based on other words in the sentence (Devlin, Chang, Lee, & Toutanova, 2018).
ELMo is built upon in contextual representations but crucially it is unidirectional model. It means each word is only contextualized using the words to its left. While BERT represents a word using both its left and right context (Devlin, Chang, Lee, & Toutanova, 2018).
BERT provides fine-tuning step in the model after the input has been transferred to word embeddings. Just to simplify our task, it is in actual a multi-classification job which for each input sentence, predicts the label, SUPPORTS, REFUTES, or NEI. In the training phase, all of the claims within their evidence are feed into BERT with a pre-trained language model (wiki pages) supported by BERT. All these claims are paired with the evidence one to one, marked its label. For how much evidence it has, there would be the same number of input samples. For example, suppose we have a claim of three evidence and the label of this claim is NEI, then the input sample would be split into three:

*input sample 1: claim, evidence 1, NEI*
*input sample 2: claim, evidence 2, NEI*
*input sample 3: claim, evidence 3, NEI*

The training process is restrained with the computing power if it runs in a single computer. Therefore, there are 1:1:1 for three labels input sample in total number of 30,000 through the training dataset, that is 10,000 for each one.
As for the prediction phase, there are 14,997 claims in total. For each one of them, the system goes to step 2 searching relevant sentences for the claim. If the k value for the information retrieval's retriever is 10, then the input sample fed into BERT is right paired with the claim just like it before, check the following for better understanding:

*input sample 1: claim, relevant sentence 1*
*input sample 2: claim, relevant sentence 2*
⋮
*input sample 10: claim, relevant sentence 10*

Therefore, a total number of 144,970 input examples are to be predicted. Eventually, the model would produce the probabilities of the three labels for each input sample.

### 2.4 Label Processing

So far we have 10 relevant sentences to a claim and the predicted label that tells us the validity of the sentence for all claims in the test set. As it has been measured the similarities between claims and each of all their relevant sentences. Arguably, the top one's predicted label is someway aligned with the true label of the claim. It means the predicted label may has the same similarity, of the claim-sentence pair, with the true label. Therefore, for each claim, the system predicts the label as the same as the top relevant sentence has.

## 3 Evaluation

### 3.1 Information Retrieval

***Searching Structure: Document name + all sentences under this document***

For each claim in the devset.json, use BM25 and TF-IDF algorithm to search the relevant top K documents from the given wiki dataset. The result shows in Table 1, through TF-IDF, the document retrieval could coverage 64% correct evidence with k1 = 5; while through BM25 with the default k1 (1.2) and b(0.75) value, the coverage rate achieves 66% with top 5 documents. After changing the k1 (0.1) and b (0.1) value of BM25, the coverage rate could achieve 75% with searching the top 5 documents.

| **Top K documents** | K=3, 4, 5 |
|---|---|
| **Coverage (TF-IDF)** | 58%, 61%, 64% |
| **Coverage (BM25)** | 60%, 64, 66% |
| **Coverage (BM25, k1=0.1, b=0.1)** | 67%, 70%, 75% |

Table 1. Top K documents coverage rate

***Searching Structure: Document name + single sentence in this document***

For each claim in the devset.json, use BM25 and TF-IDF algorithm to search the relevant top K sentences from the given wiki dataset. The result shows in Table 2, through TF-IDF, the sentence retrieval could coverage 58% correct evidence with k = 20; while through BM25 with the default k1 (1.2) and b(0.75) value, the coverage rate achieves 78%. After changing the k1 (0.1) and b (0.1) value of BM25, the coverage rate could achieve 81% with searching the top 5 documents.

| **Top K sentences** | K=10, 20 |
|---|---|
| **Coverage (TF-IDF)** | 58%, 63% |
| **Coverage (BM25)** | 66%, 78% |
| **Coverage (BM25, k1=0.1, b=0.1)** | 74%, 81% |

Table 2. Top K sentence coverage rate

Considering both time consumption and the input file size for the next step, the system finally selects the second searching structure and BM with k1=0.1, b=0.1 as the retrieval setting. The result will return the top 10 sentences as the input of the next step.
Error analysis and enhancement:

- Compare the result with different searching structures, the document name + single sentence is more reasonable since the coverage rate is good and it does not take too much time.
- Compare with different ranking function, the overall performance of BM25 is better than the TF-IDF, it might because BM25 take the short length sentence into consideration, which means that kind of sentence could get more weights.
- Different k1 and b value in BM25 could influence the coverage rate, by tuning these two parameters, a better or more suitable result could be achieved.

## 3.2 Transfer Learning (Deep Learning Model Training)

In this section, the system will train the processing data from train.json based on the different BERT pre-trained models (Devlin, Chang, Lee, & Toutanova, 2018). In order to promote the training process and control the total training time, DataFrame from pandas will be used for transfer the whole dataset into DataFrame format and then store it with the form of binary files. After training the train.json dataset, the system received our own model. Then, the system has evaluated the devset.json dataset, 89% accuracy of the correct label is achieved.
In order to get the most reasonable mode, several changes for the training dataset and model were used in this project. According to Table 3, the last one is the most suitable one since it meets all requirements in the specification and with good performance.

| **Training Dataset**[1] | **Model**[2] | **Parameters**[3] | **Training Time** | **Label Accuracy** |
|---|---|---|---|---|
| All | BASE | TB, EB = 6 | 5 hours | 89% |
| 90,000 | BASE | TB, EB = 6 | 2 hours | 86% |
| 90,000 | LARGE | TB, EB = 6 | 3 hours | 88% |
| 60,000 | LARGE | TB, EB = 6 | 1.3 hours | 87% |
| 30,000 | LARGE | TB, EB = 6 | 34 mins | 84% |

Table 3. Details about the training process

Error analysis and enhancement:

- With the 24 hours requirement, the system only takes 30,000 claims from the training dataset, even the result is not bad, the trained model is not stable for the data which has not occurred in the training dataset.
- For the pre-trained model, the default setting for the training batch size, evaluation batch and prediction batch size are 32, 8, 8 respectively. In term of reducing the training time, the training batch size is changed from 32 to 6, the new evaluation batch size s 6 and the prediction batch size is 6. Thus, the predicted result might vary a lot since it is based on the test data.
- Due to the memory limitation of the Colab, the batch was changed a lot, the higher batch size could get a more stable model. Since there are 10 sentences for each claim, the possible method is reducing the batch size (Devlin, Chang, Lee, & Toutanova, 2018).

## 3.3 Label Processing

From the last step, the system received a list of the sentence with the predicted label. The final step is to predict the label for the claim. Different label choosing strategies have applied in this project, Method 1 is taking the highest label frequency from the 10 sentences and return the most frequency one as the claim label; Method 2 is calculating the total accuracy of sentences with the same label, and return the highest score's label as the claim label; Method 3 is returning the label with the most similar sentence which means the highest BM25 value. After that, just returning the sentence with the same label as the predicted evidence.

[1] Ttraining Dataset is the trainset.json data. All represents the whole training data around 240,000 input samples.

[2] BASE model namely, BERT-Base-Uncased model. Large model namely, BERT-Large-Uncased model.
[3] TB = *Train_batch_size*, EB = *Eval_batch_size*

Table 4 shows with Method 3, the system could arrive 60.9 scores label accuracy and 40 scores for the sentence f1 value. The result in the following table is based on the test dataset. For the final competition, the system reaches 57.5 scores for the label accuracy and 37.7 scores for the sentence f1 value.

| Method | Label Accuracy | Sentence F1 |
|---|---|---|
| 1 | 25.97% | 16.8 |
| 2 | 33.33% | 18.9 |
| 3 | 60.9% | 40 |

Table 4. Result for the test dataset

Error analysis and enhancement:

- For Method 1, use the frequency as the filtering operation is not reasonable. It turns out the simple statistic theory is not a good choice for this project as well as Method 2.
- There is another method that did not mention in the previous section, a simple Wrod2Vec model was used in this step. Basically, it build a simple vector with claims and the relevant top 10 sentences, then uses n_similarity to calculate the difference between the claim and each sentence. After that, return the highest scores sentence label as the claim label. However, the result of this method is terrible. Thus, it has not included in Table 4.

### 3.4 Codalab Competition

Figure 1 shows the improvement after applying the different methods.

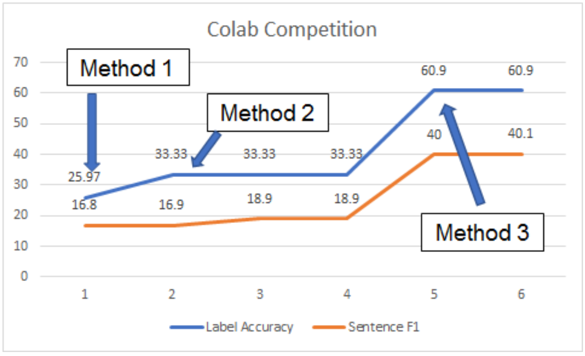

Figure 1. Codalab Competition Rank

## 4 Conclusion

An information retrieval (IR) based automatic fact verification is implemented. The sentence f1 score of 40 and the label accuracy of 60.9 is achieved on the development dataset (devset.json). A Codalab competition with 57.5 scores of label accuracy and 37.7 scores of sentence f1 in the final test. A variety of NLP technologies and tools has been used in this project, such as Lucene, Google Cloud, Colab, TPU, BM25, TF-IDF, Word2Vec and BERT. The total time consumption for the whole project is around 21 hours, which includes 18 hours GPU training time and predicting the claims in the test dataset (equal to 37 minutes TPU training time), 23 minutes for indexing the whole wiki content and 2 hours for transforming the JSON object to input samples (30,000 claims).

## 5 Future Work

- This system is an IR-based question and answer system. A knowledge-based system could be applied in this project (Bajwa, Ali, & Shahzad, 2009).
- Try to tuning more parameters about the BERT pre-train model (google-research, 2019) and find the most suitable batch size and sequence length.
- Explore more NLP technologies and apply these on the dataset (Thorne, Vlachos, Christodoulopoulos, & Mittal, 2018), such as named entity recognition, dependency parsing, and bigram model.

## 6 Reference

Apache Lucene. (n.d.). *Apache Lucune - Welcome to PyLucene*. Retrieved from http://lucene.apache.org/pylucene/

Bajwa, I. S., Ali, S., & Shahzad, M. (2009). Object oriented software modeling using NLP based knowledge extraction. *European Journal of Scientific Research*, 22-23.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805*.

google-research. (2019, 5 29). *google-research/bert*. Retrieved from Github: https://github.com/google-research/bert

Hanselowski, A., Zhang, H., Li, Z., Sorokin, D., Schiller, B., Schulz, C., & Gurevych, I. (2018). UKP-Athene: Multi-Sentence Textual Entailment for Claim Verification. *arXiv preprint arXiv:1809.01479*.

Lu, W., Stephen, R., & Andrew, M. (2005). Field-weighted XML retrieval based on BM25. *International Workshop of the Initiative for the Evaluation of XML Retrieval.*, 161-171.

Thorne, J., Vlachos, A., Christodoulopoulos, C., & Mittal, A. (2018). FEVER: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*.