Fact Checking System for FEVER

Student Name: Jun Wang
Student ID: 1001457
Team: 8395

Student Name: Youshao Xiao
Student ID: 876548
Team: 8395

Abstract

Automatic fact verification is a hot topic in Natural language processing recently. Fact Extraction and VERification as a publicly available dataset for fact-checking, it is very challenging. In this article, we will introduce our automatic fact-checking system for the FEVER challenge set. We used our Title-Based search tree and used the BERT model to build our system and obtained excellent retrieval results and high-performance fact verification. Our system achieved first place in the COMP90042 Project's competition

1 Introduction

Fact Extraction and VERification (FEVER) (Thorne et al., 2018) is a publicly available dataset for automatic fact verification, it needs the system to be able to search for relevant sentences from the provided Wikipedia according to the claims, and then make fact verification on the claims. This includes tasks such as information retrieval, relationship extraction, and classification. So it is very challenging.

In the Web SearchText Analysis competitive project, we need to create a fact verification system for FEVER. Our team 8395 built the system based on our Title-Based Search Tree (TBST) and BERT (Devlin et al., 2018), and we got 86.7% document selection F1, 75.7% sentence selection F1, and 74.4% label accuracy in the final competition. Each item is the first and has reached or even exceeded the results of the real FEVER competition last year. This article will introduce the whole system, which is divided into four parts, document retrieval, sentence selection, evidence selection, and final Recognition. We will describe the pipeline system start with dataset analysis and preprocessing, then explain each part of the system in details, and finally, measure the performance of the system and do the error analysis.

2 Data Preprocessing

FEVER provides four files, training set, development set, unlabelled test set, and Wikipedia pages set. Among them, the Wikipedia text has been preprocessed, and the other three datasets also need to be done pre-processing. First, normalized the Unicode of the claim, convert it from *NFD* into *NFC* format. Then replace the punctuation, ":" to "-COLON-", "(" to "-LRB-", ")" to "-RRB-" etc. Keeping everything consistent will help improve the performance of the system.

3 System

3.1 Document Retrieval

The first step of the whole system is document retrieval, to find the articles that are most relevant to the claims. In order to ensure the performance of the next steps, the number of recalls and articles is used as a measure. There is two method been used in this stage; one is the similarity search, using pyLuence to build a small search engine for wiki documents and search for claims; the other is the Titles-Based Search Tree(TBST) we built ourselves. TBST system has achieved much higher performance than traditional search methods like using either NER or constituency-based or similarity-based method

PyLucence: For the PyLucene, we construct the index using the preprocessed wiki titles and contents. Then we have used the built-in QueryParser to search for wiki titles which match the claim. To capture the pattern that the keywords in the claim may not appear in the wiki titles. We search for the wiki content which matches the claim. The hybrid of both title based and content based searching gives better performance.

Title-Based Tree: To structure the tree, extracted all the titles from the wiki-page, tokenized the titles and lemmatized and lowercased the to-

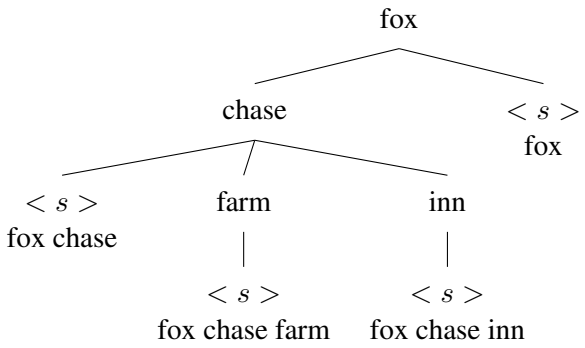

Figure 1: Title-Based Search Tree

kens. Then add the tokens by the order into the title-tree become the nodes. Add a stop sign $< s >$ at the end of the titles and store the lowercase title in the leaf. Figure1 is an example tree of titles. When searching, did the same preprocessing for the claim and then traversed each token of the claim and search in the title tree. When the next word does not exist in the subtree of the previous token, check to see if there is a stop character and if so, add the title contained in it to the potential title. Then match all the possible titles to the claim, filter the titles used the same tokens and keep the longest title of those.

There are a lot of derivative titles for one title, for *Fox*, it has can mean *Fox (Poland)*,*Fox (Finland)* or *Fox* itself, but in the claim, they all represent as 'Fox'. To solve this problem, all the derivative titles have been stored in a dictionary, when a title appears in a claim, all its derivatives are treated as potential titles. There are also case-sensitive questions in the title matching. For example, YouTube is one of the Wikipedia titles, while in the claim it is Youtube. To solve this problem, the case-sensitive and uncase-sensitive match been considered. However, the recall and number of the documents is a trade-off.

Evaluation: Table1 shows the performance of document retrieval. TBST-case is the result that matches precisely using the Title-Based Search Tree, and TBST-uncase is the match result that does not take into account the case sensitive. In contrast to the results of pyLuence, it is clear that the superiority of TBST performance. A very high recall rate can be achieved with a small number of articles. Moreover, because of the tree structure, the search speed is also breakneck. The time complexity of TBST is O(n), where n is the length of the sentence. To run the development set, which contains 5001 instances, it only cost 1 sec.

Method	**Recall(%)**	**Avg Num**
TBST-case	89.40	9.52
TBST-uncse	90.30	28.74
pyLucene k=10	81.55	20
pyLucene k=50	88.12	100
pyLecene k=100	89.92	200

Table 1: Document Retrieval Results

3.2 Sentences Selection

The second step is the sentence selection. This step is the coarse screening based on TF-IDF. The accuracy of using model filtering is high, but it takes a lot of time and resources, it is not able to feed all the sentence of the results of document retrieval into the model. Thus, in this step, the narrow down the scope of the sentence should be done.

TF-IDF Filter: For sentence selections, the goal is to find candidate sentences from the results of document retrieval. To use all the documents found in document retrieval to build a new inverse index. Then calculate the TF -IDF score of between each sentence and claim. There are three different results of document retrieval, TF-IDF, case-title, and uncase-title. First, we do the sentences filter for those three results separately. For the result of pyLuence , the k is took 100, and remove the sentences that did not contain any token in the claim in advance to narrow the scope.

Evaluation: Table2 shows the performance of the sentence selection. Results is the results of the document retrieval. Using the results of the TBST-case for sentence screening, the performance is the best. Although the recall of TBST-case's documents is lower than that of TBST-uncase, but the small number of articles makes the target's information entropy lower, making it easier to find the corresponding sentence. on the contrary, the more the number of documents, the target's information entropy is higher, it will contain more noise affects the final search results. However, because of the case-sensitive problems mentioned in the section, some of the evidence in TBST-case result is empty, which will affect the evidence selection and final recognition. Thus, we combine these three results, merged top 20 of the TBST-case, top 5 of the TBST-uncase and the top 10 of the pyLuence result. It will have 90.16 % recall and 30.52

Results	k	Recall(%)	Avg Num
TBST-case	10	83.62	9.13
	20	86.72	16.00
	30	87.76	21.37
TBST-uncase	10	78.52	9.94
	20	89.92	15.59
	30	84.78	28.80
pyLuence	10	66.07	9.74
	20	71.97	19.04
	30	75.22	27.83
Merged		90.16	30.52

Table 2: Sentence Selection Results

average number of the sentences, which is a good result for next steps.

3.3 Evidences Selection

The next step is evidence selection, which is a fine screening based on the model. Here we choose the BERT and use the BERT pre-trained model. The reason for using BERT is because of its powerful effect, it obtained the state-of-the-art result in many NLP tasks; and the second is because BERT's pre-trained model used Wikipedia as one of the corpus for training, which is very suitable for this task.

Training: We treat the sentence selection as a binary classification task. The input is a claim and a sentence, and the output is positive (is evidence) or negative (not evidence). We extract all the claim and its evidence from the training set to generate the positive training data, and then perform the above search on all the claims, and use the non-evidence sentence in the candidate sentences of the claim as the negative training data. Because when we finally make predictions, the number of negatives will be more than positive, so when generating the training set, if the required evidence is more than three sentences, we also choose the same number of negative. If less than three sentences, We choose Three negative sentences. Finally, we generated approximately 600k+ training data to train our model.

Evaluation: We have a grid search on the threshold of the model; Table3 shows the result. Because negative data is much more than positive, the threshold needs to take a higher value to have better results. However, the evidence of some instances is empty. Use the sentence of the top1 score to fill it. Finally, the threshold was set at 0.999 which had the highest performance of the evidence selection results.

th	Precision(%)	Recall(%)	F1(%)
0.9	63.67	81.25	71.39
0.95	68.47	79.47	73.56
0.99	78.49	74.41	76.39
0.995	80.99	72.90	76.74
0.999	83.36	71.16	76.78

Table 3: Evidence Selection Results

3.4 Recognition

The final step we need to identify the claim, based on the evidence we found, we need to classify the claim into three categories: *SUPPORTS*, *REFUTES* or *NOT ENOUGH INFO*. It is evident that this is a classification task. We still use the BERT model as our classifier, enter the claim and the content of all the evidence, and then classify the claim into the different categories.

Training We trained another BERT model as the final classifier. To do this, we extract all the claims and their evidence from the training set, and find the corresponding content in Wikipedia according to the given evidence, and splicing all the evidence together to become new content. Because some evidence appears in the middle of the article, the subject may be unclear, such as use he or it instead of the original subject. Therefore, when we merge the content of the evidence, we add the title of the article to the front to ensure the integrity of the content. For the claim in the *NOT ENOUGH INFO* category, training set does not provide the evidence, so we used the previous system to find the sentences most relevant to these claims to be evidence of those claims. Finally, we feed the 145k training set claims, their corresponding content, and their labels into the model to train the final classifier.

Evaluation: We chose Bag of words(BOW) and Random Forest classifier to be the baseline. The performance of the recognition result shows on the Table4. The final result of the model achieved high performance, nearly 30% higher than the set baseline.

4 Final Results

In the competition for the test set, our team 8395 achieved first place with 86.7% document selection F1, 75.7% sentence selection F1, and 74.4%

Method	Label Accuracy(%)
BERT-Model	74.77
BOW-DT	44.58

Table 4: Recognition Result

label accuracy. It also shows that the system can achieve the same result for unseen data, and no overfitting occurs. Compared to last year's real FEVER competition results, because of evidence chain, we cannot compare the sentence selection F1, but for the label's accuracy, we have improved the accuracy by nearly 10 % compared to the first place in last year's competition. It shows the high performance of our system.

5 Risk analysis

In this section we present our risk analysis of our final pipeline. The errors mainly comes from four parts.

Document Retrieval: Some instances have second level evidence which the title appears in the evidence rather than claim. For example, one claim is 'Telemundo is an English-language television network,' however, the title of some evidence is "Hispanic and Latino Americans," which has no token appears in the claim. We used pyLucene to find some similar sentence, but the system is still hard to locate this evidence. We may need to consider entity linking and deepen the links in each section to find further secondary evidence.

Sentence selection: The error of the sentence selection comes from two part; One is from TF-IDF filtering, and another is from misclassification of BERT model. Also, there is a trade-off between the False Positive rate and False negative rate. The similarity-based filtering helps to screen a lot of unrelated sentences and reduce the final sentence of each claim to 29 sentences. However, it may also exclude the target sentences which have a low similarity. The BERT based sentence selection model has a few misclassifications. The final pipeline sacrifices 10% of recall rate to improve 20% of the precision rate to achieve the highest F1 score by specifying the threshold to be 0.999.

Recognizing: The errors in the recognizing part come from two sources; One is the lack of the target evidence propagated, it from the document retrieval and sentence selection part. It leads to 10% of inaccuracy in the classification. Another type of error comes from the classification of the model itself. In the development data set, we find that there are 15% instances that model makes the "wrong prediction" even if we provide all the target evidence. There are also a significant number of "misclassification" should be noticed. For example, the claim says that "Magic Johnson was a tap dancer." The development dataset marks it as 'REFUTES' with the evidence while the model marks it as *Not Enough Info*. The fact is that it is correctly marked as *Not Enough Info* if only based on the wiki information without prior information like common knowledge.

Dataset error: There are misspelling error and label annotation errors in the dataset. For the misspelling error, the name "Homer Hickmam" was misspelled as Homer Hickman in the claim "Homer Hickman wrote some historical fiction novels." The other is respect to the annotation errors: the claim 'Mel B released a song on Virgin Records in 2017' is marked as SUPPORTS with the one evidence 'Mel B, Brown began her solo career when she released I want you Back with Missy Elliott on Virgin Records.' It is an annotation error since the evidence actually could not fully support the claim, and the model correctly marks it as *Not Enough Info*.

6 Conclusion

In this article, we introduced our fact verification system for the FEVER dataset. In the system, we used our own title search tree system to do the document retrieval, and then used BERT to carry out our evidence selection and final classification. The system achieved good results. We scored first in the CodaLab's rankings. In the future work, we will consider deepening the links of each part, fine-grain the result of the document retrieval, using the technology of entity linking to find the chain evidence, and generate a more complete training set to train the model.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *NAACL-HLT*.