COMP90042 Project 2019: Automatic Fact Verification

Zhengyu Chen 991678 Shuyu Yuan 985441

May 2019

Contents

1 Abstract

This fact verification system aims to build a neural network to verify claims by extracting relating information from raw texts (namely, wiki files) to distinguish whether it supports or refutes the claim. We proposed Lstm network to treat with the problem of fact verification. Compared with the recurrent neural network, Lstm has already been proved that it has a better performance when dealing with the long sequence sentences. We employed the neural network and trained this network for our fact verification system. The experimental studies that Lstm has the ability to judge the relationship between claims and evidence among the labels of support, refutes and not enough info. Moreover, we also preprocessed the dataset for training and dataset for testing in order to simplify other procedures and improve the efficiency of the system.

2 Introduction

In our daily life, we may hear of some facts and other words, some of them are right without a doubt. However, there are still some facts which are out of our common knowledge. Such contents exist throughout the whole network, and the amount are too huge to be verified manually. In this case, it is apparent that we need an auto-fact-verification system. In this report, we are going to build a fact-verification system, which has the ability to identify whether the fact is right or wrong based on the information of wiki docs. This system will support the fact with the output of SUPPORT label and refute the fact with the REFUTES label. However, sometimes when the wiki docs are not enough to verify the fact, this system will say not enough info with the label of NOT ENOUGH INFO. This whole system is built based on the wiki docs. We have got totally 109 texts storing the Wikipedia entries with each of the text file containing over 8000 entries. These wiki entries are our evidence that will support or refute the fact. For each entry, the entity is clearly placed at the first of the entry, then all the content of this entity are split into sentences and marked with the number of the sentence. Except for the situation of the not enough info, the result will be placed with the label of either SUPPORTS or REFUTES followed by the evidence, which are one or more sentences retrieved from the wiki docs. When the system cannot find sufficient evidence front the wiki docs to support or refute the fact, it will label the fact as NOT ENOUGH INFO.

This system is divided into three parts. The first one is preprocessing the dataset for training and testing, the second one training our system with the training preprocessed dataset and the last one the to verify the test facts through our system.

Data Preprocessing Because the structure of the training data and the test data are totally different, they should be preprocessed separately, which is complicated. For the training data, as all the evidence of each fact are clearly

marked with the entity name and the sentence ID, it is quite straightforward to retrieve the evidence from the wiki files. When it comes to the testing data, we can search the wiki docs so as to get related evidence that is likely to contain the proof of supporting or refuting the facts. Of course, we have tried some libraries to do information retrieval, like Whoosh, Pylucene, etc, but it seems they are not as easy to use as using the built-in hash function in the programming language, which is Dictionary in Python.

Training

After having preprocessed the training dataset, we use the dataset to train our neural network, we choose the LSTM as our main neural network structure because LSTM is based on the recursive neural network, which has already been proved that is more suitable for natural language processing, and LSTM has a better performance when the sequence are quite long. The objective of this training is to enable this system to specify the relationship between the claim sentence and the evidence.

Fact Verification

With the complement of the training, the preprocessed dataset for testing is ready to be input to this system. This system will label the fact among SUPPORT, REFUTES and NOT ENOUGH INFO. In this report, we explain the work process of our system as well as the justification of choosing them. Also, performance and result analysis will be included in this report. We are going to explain some of the other ways we have tried but not good enough. According to the analysis, we will also provide some advice for improvements.

3 Background

3.1 Dataset

The dataset is derived from the FEVER challenge, which is an ongoing research competition. It' s an available dataset for verification against textual sources. FEVER means Fact Extraction and VERification. It consists of 185,445 claims generated by altering sentences extracted from Wikipedia and subsequently verified without knowledge of the sentence they were derived from. The claims are classified as SUPPORTED, REFUTED or NOT ENOUGH INFO by annotators achieving 0.6841 in Fleiss . For the first two classes, the annotators also recorded the sentence(s) forming the necessary evidence for their judgment. To characterize the challenge of the dataset presented, they develop a pipeline approach and compare it to suitably designed oracles. The best accuracy achieved by the office team on labelling a claim accompanied by the correct evidence is 31.87%, while if ignoring the evidence they achieve 50.91%. Thus they believe that FEVER is a challenging testbed that will help stimulate progress on claim verification against textual sources. [2]

Figure 1: Document Retrieval

3.2 Background information

We did our experiment based on the programming language of Python 3.6. For the dataset preprocessing stage, it was completed on our own laptop. However, since the wiki files are so large, it is a little time-consuming. My laptop has 8G memory with 1600 MHz and I5 CPU with 2.4GHz, it took me about half an hour to generate the training file. During the machine learning stage, we did a brief test on our laptop so as to ensure the process was correct and then moved it to the colab and training it with the GPU at the cloud. For the information retrieval of the test file, we did it at a virtual machine at the cloud because it was much more efficient.

4 Data Preprocessing

4.1 Exposition of Technique of Information Retrieval

4.1.1 Training Data Retrieval

For the training data, we use the json library to load it. For each list in the filed 'evidence', the two parameters can be directly treated as two keys in order to retrieve the related sentence in the wiki dictionary. For example, train[75397]['evidence'][1] = ["Nikolaj_Coster-Waldau", '7'], then we use the entity "Nikolaj_Coster-Waldau" and the number 7 as the key to retrieve the sentence wiki_dict["Nikolaj_Coster-Waldau"]['7'], and we get "He then played Detective John Amsterdam in the short-lived Fox television series New Amsterdam -LRB- 2008 -RRB-, as well as appearing as Frank Pike in the 2009 Fox television film Virtuality, originally intended as a pilot .". Note that numbers are uniformly converted to characters to prevent coding problems.

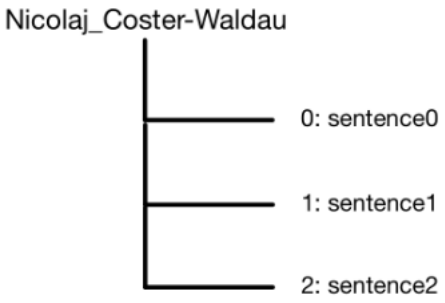

Figure 2: Document Retrieval

However, even we get the keys to locate the the exact evidence, it is still time-consuming to search in all the wiki files. In order to raise the efficiency of data retrieval in the training dataset, we build a dictionary based on all wiki files, the key to this dictionary is the entity, the value corresponding to this key is also a dictionary, in this inner dictionary, the key is the number of the sentence and the value is a certain sentence. The structure of the dictionary are shown below.

With this data structure, the training data retrieval is more efficient. Note that it does work really fast but when moving the whole dictionary to the memory, it is quite time-consuming. But in general, it still saves so much time for training data retrieval.

4.1.2 Training Data Analysis

For the training data, we should first analyze the distribution of (1) the number of "SUPPORTS", "REFUTES", "NOT ENOUGH INFO", (2) the number of evidence in the instances with the label "SUPPORTS" and "REFUTES", (3) the length of all evidence.

It is obvious that the number of instances with the label "SUPPORTS" is far more than the instances with the label "REFUTES" and "NOT ENOUGH INFO", so there may be a risk of overfitting to the label "SUPPORTS", which means we should consider using some methods to prevent overfitting.

For most of the instances with the label "SUPPORTS" or "REFUTES", the number of evidence is equal to or less than 3, and the number of instances having the number of evidence equals to 1 is more than the sum of the numbers of all the other circumstances. We choose to eliminate the the evidences more than three because when we are generating a batch of embedded training data, the longer the evidence is, the more residual space we need, which means other instances which has less than or equal to 3 evidence may need to fill the space

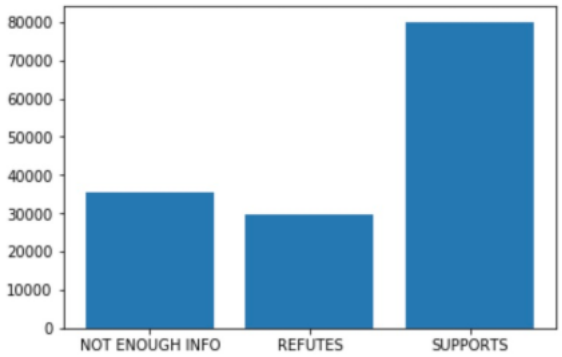

Figure 3: Label Distribution

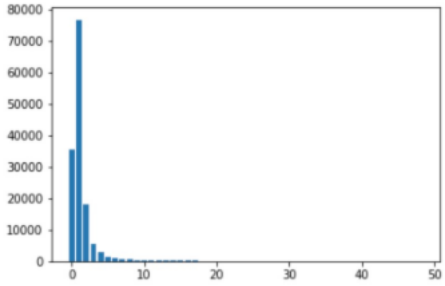

Figure 4: Number of Evidence Distribution

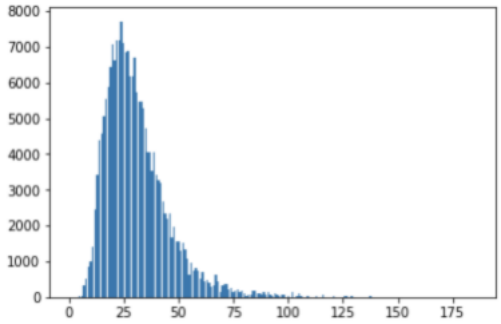

Figure 5: Sentence Length Distribution

with meaningless values and it may cause the problem of training to slow and underfitting.

The distribution of the length of all evidence meets the normal distribution. Again, in order not to fill the space with meaningless values caused by too long sentences, we choose to eliminate 15.7% on the right side, which means we intercept sentences if their length exceeds 85 single words.

4.2 Test Data Retrieval

For the data in the test-unlabelled.json file, it is quite different from data retrieval in the training dataset because the exact evidences are not been given this time. So we divide the data retrieval procedure into three stages.

The first stage is to find the entity in the claim. We employ the pos-tagging technique to find the entity in the claim. As most of the entity are nouns or consist of nouns, so first we get the pos-tags of all the words in the claim, then according to the tags we pick the words of which the pos-tag is NN or begin with NN linked by NN tag. We connect the words with the underscore.

The second stage is to search the relevant sentence from the wiki files based on the entity got from the first stage. We collect all the wiki sentences by add the key name 'evidence' and corresponding value as a list. In this list, the first place is the entity name, the second place is the number of the sentence and the last one is the exact sentence. Structure is display below.

The justification to design such a structure is that this data structure is quite similar with requirement of the final answer.

Next, we are going to pick out the most relevant sentence as our evidence from all the sentences. In the first stage, we search all the entity from the claim and collect all of their wiki sentences, it must be a huge sentence list with so many irrelevant sentences. We plan to use TF-IDF model the rank all the evidences

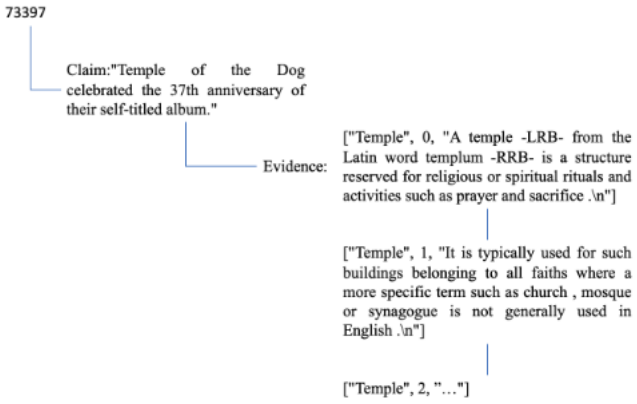

Figure 6: Document Retrieval

and choose the three highest score sentence. Notice that the TF-IDF model is not employed in the whole test dataset, but employed in each certain entry of the test dataset. With the example of the structure diagram of the first stage, we regard the claim as our query and all the sentence of the evidence as our corpus. Each sentence represent for a doc and we calculate the TF-IDF vector of each sentence. Then we get the dot production of each each sentence and the claim. After ranking the dot production, we choose the top three result as the evidence and exclude all the rest sentences.

4.3 Error analysis

There are also some problems and shortcomings in it. For the first stage, we employ the pos-tagging technique to retrieve the entity in the claim. However, there may be some special entities, which don't meet the requirement of the filter. In this case, these entities are more likely to be split, which influence the performance of the final verification. For example, the claim is "Temple of the Dog celebrated the 37th anniversary of their self-titled album.". The entity should be Temple of the Dog while our system can only gets the temple and the dog.

4.4 Information Retrieval Conclusion

We choose different techniques to deal with the information retrieval in training dataset and the testing dataset basically because the testing dataset doesn't

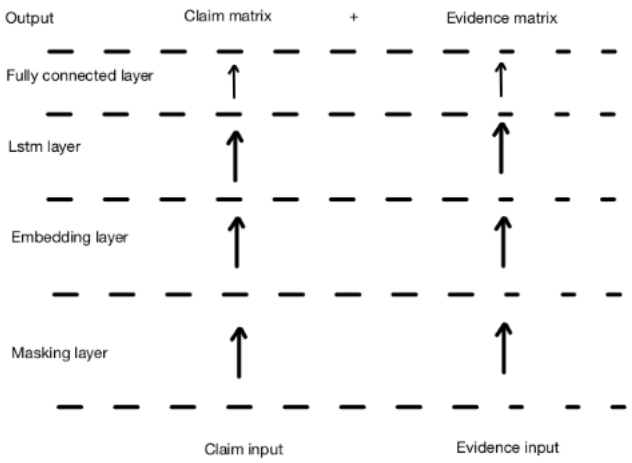

Figure 7: Document Retrieval

provide the certain evidence. Another determination is that evidences should be no more than three sentences according to the analysis of the training dataset.

5 Training

5.1 Model

5.1.1 Structure

We placed a masking layer at the bottom of the neural network. Since we would put the whole sentence in this network, the length of each entry is likely to be different with others'. So we use the masking layer to add zeros at the bottom of each vector until the length of the vector equals the longest length vector in this batch. We put the embedding layer on the masking layer, which is responsible for doing embedding for each masked vector. Then we placed the Lstm layer over the embedding layer. The Lstm layer gets the embedded matrix and sent the output to the fully connected layer. As the output from the Lstm is a 3-dimensional matrix, the fully connected layer projects it back to the 2-dimensional matrix, in the shape of batch size times number of labels, which is three in this report. Particularly, we separated the claim and the evidence while inputting them into the model. The claim sentences and the evidence sentences will do training separately and being added after the fully connected layer. We simply add to matrix together, the size of the final matrix is the batch size times three, which is the number of the label. The structure of the neural network are below:

5.1.2 Model processing

With the structure of the model, first we are going to reading the training data. We extracted the useful part of the training dataset, got the claim sentences and the evidence sentences. For each claim, we formed a dictionary to indicate which sentence is the claim and which are the evidences and then attached with the label. We also have some training entries which are labelled as "NOT ENOUGH INFO". In this case, there is no relevant evidence attached. In order to train our model to identify the irrelevant relationship, we place the former evidence sentence as the evidence in this situation. Besides the problem of not enough info, some claim may has too many evidences to prove it is true or not. In this condition, we will restrict the number of the evidence not over than three. The justification is based on the analysis of the training dataset. Since most of the claims only need not more than three evidences to support, we choose the three as the limitation(Claims of which the evidence sentences are equal or less than 3 dominate over 60% of the whole dataset). The reason why we are essential to set this limitation is quite obvious. Suppose one claim needs over 10 sentences as its evidence, in this batch of this claim, the metric will be quite sparse because other shorter vectors are masked to reach the same length of the longest one. As the metric is sparse, the training performance will not be good enough.

5.1.3 HyperParameter

Hyperparameter are listed in a table below.

batch size	epoch	optimizer	learning rate
32	100	SGD	0.1

We set our batch size as 32 because of the BasicIterator has the default batch size of 32. For epoch we set it as 50 times. We choose the SGD as our optimizer and make the learning rate equal to 0.1. Hyperparameters will not be changed further because we will adjust in other places in order to improve the performance of the system.

5.1.4 Model prediction

6 Result

We have tried several approaches to split the sentences in the key "claim" and "evidence", which makes the models have significant differences. At the very beginning, it was not satisfactory at all.

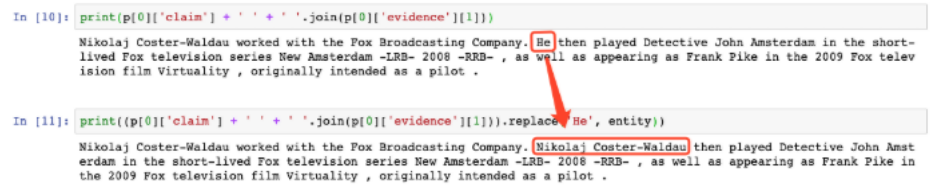

Figure 8: Document Retrieval

Method	Training Data Accuracy	Document Selection	Label Accuracy
Concatenate	56.32%	23.8	33.6
Split	82.71%	58.7	42.2
Split with Entity	92.89%	54.4	45.4

At the first time, we concatenate the sentences in the key "claim" and "evidence" and use this combined sentence as the input to train the model. Then when we are predicting a test case, we also concatenate the sentence in the key "claim" and the sentences we retrieved from the wiki files. However, we have only trained a model with an accuracy of 56.32%, which is too low to accept.

Then we realised that the model will treat the concatenated sentence as one single sentence vector, without relationships of "claim" and "evidence". Then we changed the structure of the model, and we split the two fields, "claim" and "evidence", as two separate inputs of the model. And as a result, we have observed some improvements. The accuracy has been raised to 82.71%.

In order to continue to improve accuracy, we need to make the information inside sentences more accurate. We came up with the idea that to replace the pronoun, like "she" or "he" etc, with the precise entity. For example, for a sentence "Nikolaj Coster-Waldau worked with xxx. He then played xxx", the entity "He" would be replaced with Nikolaj Coster-Waldau" to make the Lstm obtain more information to classify the input. As a result, accuracy has been raised to 92.89%.

The image below shows the accuracy on the training data after each epoch, and this is the result of our training using the final model. It has an accuracy of 92.89

7 Future Work

Since we only used some naive and simple approaches to get the wiki information associated with the entity in the 'claim' field, that is to match the entity with the key strings in all the wiki files and retrieve all subordinates once they are matched, the speed of the searching process is so slow that we have to wait for a long time for this work to be done, and then store what we have retrieved in

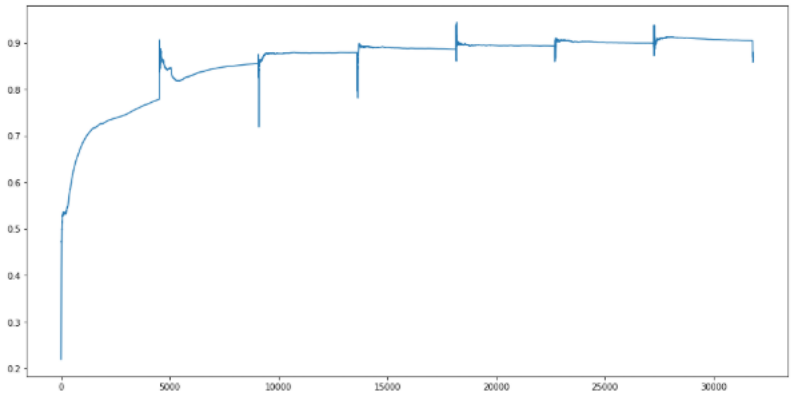

Figure 9: Document Retrieval

a file in order not to waste time on this process again. Besides, the accuracy of the retrieved documents seems not so good even if we have done stemming and lemmatization separately. Maybe some tools or search engines like Pylucene and Whoosh can help.

In the training phrase, although we have added dropout layers inside the model, it seems that not only the accuracy on the training data decreased but also the accuracy on the verification data decreased. And even if Lstms do not suffer from the optimization hurdles that plague simple recurrent networks [1], the structure of our model still needs to be optimized.

References

[1] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[2] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.