COMP90042 Web Search and Text Analysis

Project Report: Automatic Fact Verification System

Xianchen Huang: 858176 Xin Tian: 908811

1. Introduction

In recent years, there are more and more false information and fake news on the Internet, which leads to people being often misled when retrieving the information. However, with the rise of big data, it is time-consuming and lack of resources to manually check whether the information is correct, so automatic fact identification becomes very important.

In this project, our aim is to build a system to verify the given claim is valid or not based on the given datasets. The dataset consists of Wikipedia documents (Thorne et al., 2018), which is used as evidence for looking up and the training dataset is made up with label, evidence and other information. What we should do is to give the claim from the development dataset a correct label as *Supported, Refuted* and *NotEnoughInfo* and the corresponding evidence searched in the Wikipedia docs.

2. Background

In this section, we will introduce the tools we have used to build our system.

2.1. PyLucene

PyLucene (Vajda, 2005) is a python encapsulation of the search engine Lucene, and it is easy to call Lucene's API through python. Lucene is a full-text search module that can be easily embedded into applications. Using PyLucene enables our program to perform a full-text search without having to write additional methods to retrieve information. The version of PyLucene we used is 7.7.1.

2.2. Keras

Keras (Chollet, 2015) is an advanced neural network package that works with Theano and Tensorflow, making it faster to build a neural network.

3. System Clarification

In order to realize the achievement of automatic identification, we turned the verification problem into a classification problem using a deep learning framework to make classification and regression prediction for input claim. In the beginning, we divided the project into several sub-tasks to make it clear and the implementation of each task will be explained below.

3.1. Doc & Sentence Retrieval

3.1.1. Indexing

The first step for information retrieval is to collect all the given Wiki files so we use indexing to specify the index directory TXT files and save the index to the specified directory for search use. For convenience, we specified the format of every index as '[Name][TermName][Line][Content]'and we made one sentence a line. 'Name' indicates the file name, 'TermName' is the page identifier and the page number. 'Line' represents the line at which the sentence appears in the file and 'Content' is the sentence.

3.1.2. Searching

Given the retrieval term, we can get the relative sentences from the index generated earlier. We have two types of search here, one is searching based on the term and another one is searching based on the sentence. The analyzer we used here is 'StandardAnalyzer'.

3.2. Language Processing

To make the training dataset suitable for our system, we have made some preprocessing for the given datasets.

3.2.1. Preprocessing

We have changed the structure for the key-value pairs in the training dataset and then made the preprocessing for every evidence. Firstly, we put all the words in the evidence sentences together to a bag-of-word representation and discards the common stop words in the claim because the stop words have no special meaning and will affect sentence similarity later and then we leave the same semantic words as in the claim. For the claims with 'NOT ENOUGH INFO', which have no evidence, we will select one sentence randomly among the 10 sentences searched by PuLucene based on the claim. In order to look up the same semantic words, we used synsets in NLTK here to find the synonyms of every word. After this operation, we can have a new sentence and then we added padding to the new sentence with a word that does not cause ambiguity. Finally, we can get a new claim and a new evidence sentence.

3.2.2. Feature extraction

In order to get the useful features, we used the new claims and new sentences to use 'Word2Vec' to build a large word vector and then compute the cosine similarity between every pair of the new claim and new sentence. This cosine similarity is one of the features for the following machine learning steps. Another feature is quite similar, which it to use 'Doc2Vec' to build every TXT file to a document vector and compute the similarity of every pair of sentence. The third feature is the length of the claim. We saved the process new claim and the length, label, similarity into a new train file for the further use.

3.3. Modelling

We applied Keras as our tool to build the neural network model. The parameters and code we implemented are as follows, the input is the processed train dataset.

After the training phrase, we can get a differentiated model for classification. The model will output three probabilities which indicate the three labels.

```
model=Sequential([
 Dense(input_dim=3,units=64),
Activation('relu'),
Dropout(0.1),
Dense(units=128),
Activation('relu'),
Dropout(0.1),
Dense(units=256),
Activation('relu'),
Dropout(0.1),
Dense(3),                    #Output three values
Activation('softmax')        #Dimension reduction
])

optimizer=Adadelta(lr=1.0, rho=0.95, epsilon=1e-06)          #The optimizer

# Compile the neural network model
model.compile(loss='mean_squared_error',
          optimizer=optimizer,
          metrics=['accuracy'])

# Train model
print('Training-----------------')
history =  model.fit(train_input,train_output,nb_epoch=500,batch_size=64)
print("Finished training the doc2vec_model")
model.save('vertification.h5')
```

3.3. Classification

To find the accurate label and relative pieces of evidence, we will use the trained model to predict the most possible classification. First, we need to find the possible evidence for the claim in the development dataset. Here we also used the PyLucene to retrieve N top related sentences based on the claim. (We assumed N=3, 5, 8 for comparing the performance in our system). Then we arranged and combined the N retrieved sentences, which means we considering about to combinations. Because every combination is a piece of possible evidence set for the claim, we preprocessed these combinations and the claim the same way we made for the training dataset to get the features of them. After this operation, we can apply the model to classify and finally get several groups of datasets made up with three possibilities (NotEnoughInfo, Supports, Refutes) and we can label every combination of evidence based on the largest possibility After getting all the labels, we can decide the final label by comparing the number of support or refute, if none of them then we can label it as no evidence. When deciding the evidence, we preferred to choose the shortest combination, which has fewer sentences to be the evidence because the fewer sentences there are, the less ambiguity there will be and the more accurate the results will be.

4. Result

For simplicity, we labelled every claim 'NoEnoughInfo' to build the baseline for comparison. Based on the baseline, we have made enhancement several times and each time the figure of the performance is shown in table 1. We have tried 4 models to measure the achievement

and after the comprehensive consideration, we decided the final edition of our system.

Model	Label Accuracy	Sentence F1
Baseline	40%	20%
Model 1	37.11%	26.80%
Model 2	40.72%	25.58%
Model 3	35.79%	13.91%
Final Model	42.32%	25.01%

Table 1: The evaluation of every model

Table 2 shows the performance of our system when using different numbers of sentences as evidence. Recall is used to show the proportion of words with a correct prediction, which is the value of the correct response number over the total number of the misspelled words.

Sentences	Label Accuracy	Sentence F1	Document F1
3	41.62%	24.22%	26.51%
5	42.32%	25.01%	27.33%
8	41.14%	23.44%	25.93%

Table 2

5. Error Analysis

In this section, we will present the error analysis for our system. The errors may be encountered in the information retrieval process and the model classification process and additionally, the missing entity in specific claims. Here we will mention three main errors in these three situations which mainly influence our system:

5.1. Missing Entity

In some certain claims, some corresponding sentences represented as the evidence were absent when doing the sentence retrieval. For example, claim 'Nikolaj Coster-Waldau worked with the Fox Broadcasting Company' has the evidence "Fox_Broadcasting_Company 0" and "Nikolaj_Coster-Waldau 7", however we related sentences we retrieved were "Nikolaj 7", "Coster 63", "Jaime_Lannister 8", "Nukaaka_Coster-Waldau 1", the sentence "Fox_Broadcasting_Company 0" was missing. What we will deal with this error is to discard this kind of sentence during the filtering process.

5.2. Special Characters & Proper Nouns

There are a number of special characters and proper nouns appearing in the sentences, such as parentheses, underline, or some unusual name, company name. For instance, the claim "Jos\u00e9 Mar\u00eda Chac\u00f3n surrendered the island of Trinidad in 1797, one year before his death." This kind of character will make the retrieval system report error so for simplicity we just label these sentences 'NoEnoughInfo'.

5.3. Misclassified

There are quite a number of claim misclassified to the wrong label because of the disability of the model. Our system preferred to classify the claim to 'NoEnoughInfo' rather than the other two label. The reason for this error is that we marked the sentence which causes the system error as 'NoEnoughInfo' by default.

6. Further Development

The model can be refined to do a pre-filter of all input conditions for each claim to find more features. Another solution to improve the system performance is to change the input to the model. Because the similarity cannot reflect the difference obviously, the model input is best to use the vector input and we need to select a more suitable model for this method.

7. Conclusions

In this report, we represented the implementation process of our automatic fact verification system and evaluate the performance of the system by analyzing the results to select the most suitable model. In addition, by analyzing the errors happened in our system, we took some steps to handle these errors to ensure the best performance of the system and based on these operations, we finally made some future deployment, which can make our system more perfect. accurate changes further improvement.

References

Chollet, F. (2015). Keras.

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

Vajda, A. (2005). Pulling java lucene into python: Pylucene. *Retrieved March*, *23*, 2008.