Attempts on an Automatic Fact Verification System Construction

Geng Cheng 790949 , Peijia Long 752767

Abstract

With the consideration of efficiency improvement of fact verification, an automatic verification system is built as required, referencing from FEVER -- it predicts the truth of a set of claims based on the given Wikipedia document collections. As a result, a label correctness of maximum score 46.94 was achieved, and some improvements are feasible against the current system.

1.Introduction

By having an exploration upon how Internet plays an important role in the society nowadays, it could be found that one of the most benefits it offered is the capability and easiness of fetching unknown information or knowledge. Being the largest online multilingual encyclopedia currently, Wikipedia has acted as a solid reference of information with the support of over 11 million document entries the platform empowers [1]. However, due to the great amount of entries and the length of their associated introduction, manual information retrieval and verification is still somehow time consuming and unpleasant, which have led to the increasing possibility of receiving wrong information/knowledge.

The launch of FEVER (Fact Extraction and VERification Shared Task) has largely motivated the evolving of the automatic fact verification system developments. Having provided a dataset containing 185445 claims generating and altering from Wikipedia, participants are required to verify the given claims, and provide a label from ["SUPPORTS", "REFUTES", "NOT ENOUGH INFO"] to each of the claim [2].

Similar to FEVER, in the project, a simpler automatic fact verification system was built. Given a set of Wiki document collection (wiki-pages-text.zip), and a train json dataset with correct label and evidence offered (train.json), the system is expected to retrieve relevant sentences and make justification towards the truth for each claim (i.e. label generation) in the unlabelled dataset (test-unlabelled.json) on the basis of evidences retrieved. The system is designed to be implemented in three stages – Preprocess of documents, Model Training and Predictions. Model Training is a supervised machine learning process with the reference to the training dataset offered, while Predictions refers to the procedure of evidence generation and label justifications upon the unlabelled claim dataset. The implementation process and evaluation outcomes are discussed below.

2. Implementation Process

2.1 Premade Assumptions

With the considerations of convenience and overall consistency, it is assumed that every entry in the provided Wikipedia document collection strictly follows the format of [Document_Identifier, Sentence_Index, Sentence_Content] and has no empty value for each field.

2.2 Used Packages

For the purpose of model training, SciKit-Learn, a simple open-source tool for machine learning is applied in the classification of evidences and its corresponding labels [3]. To be specific, Support Vector Classification (SVC), an implementation on the basis of (Support Vector Machine) SVM library from SciKit-Learn was introduced for the training of the model in particular [3]. In addition, AllenNLP, an opensourced NLP search library, offers various language processing models, which are rather mature and hence were used in the both training and predicting processes.

2.3 Pre-process of Documents

Considering the ineffiency of file searching due to the file size, sentences are indexed. The indexing process consists of two parts with the same tree structure. For every word appeared in the name of at least one document, a list of all document names containing that word is constructed. Also, all sentences of a document is associated to the name of that document.

To construct indices, SHA-1 algorithm was used to hash the key (word for the first part, document name for the second part). The hash value is a hexadecimal string, hence each node in the index tree would have at most 16 children.

Every leaf node holds a collection of key-value pairs. With new pairs being added to the tree, if the amount of data in a leaf node reaches the capacity, it will construct child nodes and distribute the data it holds to them based on the next digit in the hash value.

When searching for a key, its SHA-1 hash value was first computed and search the index tree by layers according to digits in the hash value down to the target leaf node.

2.4 Model Training

The whole model training process could be further categorised into three steps, and they are listed in the table below.

Steps	1st Trial	2nd Trial
0	Named Entity Recognition on the claim	Search for all matching document names
1	Fetch relevant sentences from entities	Fetch relevant sentences from document names
2	Vector generation for each evidence candidate	Vector generation for each evidence candidate
3	Label prediction and comparisons with true label and evidences	Label prediction and comparisons with true label and evidences

Table 1 Steps in Model Training

2.4.1 Reflections on 1st Trial

It could be seen from the Table 1, Step 0 was discarded throughout the development process, which refers to the process of NER upon a claim with the help of SpaCy[4]. Before the start of implementation, it is assumed by the group that almost every claim is generated around at least one proper noun, which could often be recognized as a Named Entity (NE) by the SpaCy language model and recorded as a Document_Identifier in the Wiki document collection at the same time. However, during the procedure of development, it was found that a larger amount of claims do not consist of any NE than expected, making the "en_core_web_sm" model no longer feasible and constraining document searching as well. Moreover, although it is not much related to the overall behaviour of the system, the NER model is not case-sensitive and hence will not be able to extract lowercased NE like "google" out without training on purpose.

2.4.2 Explanations on 2nd Trial

Step0

Ordinary search was used instead. Since the document names are indexed, all words could be considered as a search input except stop words. For each document name retrieved, the claim and document name were tokenised to two lists of words. Then a simple check was made on whether the word list of document name is a subsequence of the word list of claim. If a document name appears to be a substring of another document name (e.g. "Football" and "National Football League"), the longer one was selected since the shorter one is very likely to be only a part of a full entity. This trivial approach works surprisingly well and is a lot faster than applying NER.

Step1

With the use of entities extracted, Document_Identifiers matching the entities were collected and their corresponding Sentence_Content were returned as candidate evidences. It is worth noting that as for retrieval methods, instead of "Exact Match", "Contains" was used previously. However, when encountering general entities like "America", too many relevant documents were returned, extremely increasing time consumptions for following steps. Therefore, the rule of "Exact Match" was applied for cutting the number of processed documents.

Step2

Textual Entailment (TE) is a model for assessing if a sentence is available to imply another, and the the output of a prediction consists of a list of probabilities corresponding to "entailment", "contradiction", and "neutral" [5]. Therefore, the result from TE could be considered as a 3-Dimensional vector behaving as [p("entailment"), p("contradiction"), p("neutral")]. Each relevant sentence getting from Step1 was processed into TE against its corresponding claim and produced a 3D vector in the vector space.

Step3

Since the three probabilities add up to 1, the actual degrees of freedom is 2 and only first two probabilities were kept. As there are 3 classes ("SUPPORTS","REFUTES","NOT ENOUGH INFO"), support vector machines was introduced to classify the 2D vectors. Here, the svm module from scikit-learn was used to train our SVM, using the TE probabilities as training vectors and given labels as target class labels.

2.5 Predicting

After successful training of the model, it is now time to predict the unlabelled claim in the test-unlabelled.json. Similarly, the whole stage could be divided into 4 steps as shown below.

Steps	1st Trial	2nd Trial
0	Named Entity Recognition on the claim	Search for all matching document names
1	Fetch relevant sentences from entities	Fetch relevant sentences from document names
2	Predictions for each candidate evidence	Vector generation for each evidence candidate
3	Label justification and evidence selection	Label selection from filtered evidence

Table 2 Steps in Prediction

2.5.1 Reflections on 1st Trial

With the consideration of time limitations, existing predictor from TE in AllenNLP was used for direct predition without the need of training model, and "entailments" corresponds to "SUPPORTS" in labels, "REFUTES" was classified as "contradiction" and "NOT ENOUGH INFO" was classified as "neutral". Each claim needs to be predicted with each of its potential evidence sentence, and a threshold probability value of 0.5 is set for the selection of evidences and corresponding labelling.

2.5.2 Explanations on 2nd Trial

Similar to the training stage, the previous steps all remained the same except for the last step, which has a subtle difference and is described below.

Step3

Similarly, only the first 2 probabilities from each TE result were taken. Then the trained model could be used to predict the label of the test vectors.

3. Evaluation Result

3.1 Overall Performance

As discussed, several attempts were made with the use of different implementation methods, and the performance results are shown in the figure below.

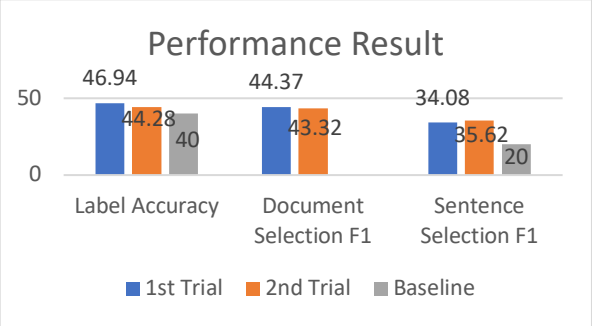

Figure 1 Performance Results

As a result, frustratingly, the self-developed model failed to show a pleasant performance by the time of submission. In addition to that, the system is quite time-consuming and took over 24 hours to finish predictions to the claim dataset. Some possible reasons of the poor performances are analysed below.

3.2 Error Analysis and Reflections

3.2.1 Prediction Model Drawbacks

claim	Potential evidence	1st Trial label	2nd Trial label	True label
A*	B*	NOT ENOUGH INFO	NOT ENOUGH INFO	SUPPORTS

Table 3 Error Example 1

* A = "Temple of the Dog celebrated the 37th anniversary of their self-titled album."

B = "Temple of the Dog was an American rock band that formed in Seattle , Washington in 1990 ."

For both 1st and 2nd trial, TE was used for direct prediction vector generation without training, and it seems that the model itself is weak in

sensing date-related justifications, since they cannot be told simply from constructing attention matrix between two sentences, nor direct cosine/tf-idf similarity calculation. This could possibly be improved by provide training to the model with date-biased data, or build a new one with calculations in consideration, although both of them are quite time-consuming.

3.2.2 Time Consumptions & Inefficient Document Retrieval

Throughout the whole development procedure, index construction and its related document/sentence retrieval was most time-consuming. Besides, current document retrieval schema requires multiple document sentence process for every single claim even with the constraint of 'exact match' – a trade-off with returned Document_Identifier accuracy.

In addition, some mispredictions were caused because of insufficient relevant document/sentence retrieval, and an example is shown below.

claim	Retrieved entity	Predicted Label	True Label
C*	Stomp the Yard	NOT ENOUGH INFO	REFUTES

Table 4 Error Example 2

*C = "Stomp the Yard stars an American actor born in 1992."

Primary reason of this particular failure is insufficient entity recognition in the sentence (i.e. without awareness of finding "actor" in the movie). A possible solution is to layer the candidate entities into primary ones and sub ones, and the document/sentence retrieval rule could be changed into layered search following the priority of entities given, although a speed compromise is probable .

3.2.3 Model Training – Vector Construction

Current vector construction is three-dimensional, which is exactly referenced from TE. However, for a better model behaviour, vectors of higher dimensions are desirable. For instance, types of entities extracted could be considered as a dimension, with different weight considerations on different types – a proper noun could have a larger weight over an adjective. The label accuracy, along with evidence correctness, is expected to raise, following the increase of vector dimensions.

4. Conclusion

The report discusses the implementation process and the performance behaviour reflections upon the developed automatic fact verification system for Wikipedia document collections with the help of AllenNLP and SciKit-Learn. The overall performance is unpleasant due to the time limitations, inappropriate choice of document retrieval schemas and immature behaviour of trained model compared to well-established ones like TE in AllenNLP. To improve the evaluation result in the future, replacement of document storage schema, sentence fetching rule, and a vector construction of higher dimensions are feasible in the future.

References

[1] "Size comparisons", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:Size_comparisons. [Accessed: 28- May- 2019].

[2] J. Thorne, A. Vlachos, C. Christodoulopoulos and A. Mittal, "FEVER: a large-scale dataset for Fact Extraction and VERification", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1803.05355. [Accessed: 28- May- 2019].

[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," MACHINE LEARNING IN PYTHON, p. 6.

[4] "Models & Languages", Spacy.io. [Online]. Available: https://spacy.io/usage/models. [Accessed: 28- May- 2019].

[5] "AllenNLP - Models", Allennlp.org. [Online]. Available: https://allennlp.org/models. [Accessed: 28- May- 2019].