AUTOMATIC FACT VERIFICATION SYSTEM

Codalab Team: White&Brown

Himagna Erla
(Student ID: 975172)

Vitaly Yakutenko
(Student ID: 976504)

May 2019

Abstract

Given a collection of claims and Wikipedia corpus the task was to determine if those claims are true/false statements or if there is not enough evidence to make such a decision. The goal was to maximize the accuracy of our responses as well as an F1 score of supporting document and sentence evidence. This collection of claims was obtained from FEVER competition and described in the FEVER paper (Thorne et al. 2018). Train, dev and test sets of claims were provided for building the system. Evidence given in train and dev sets will be referenced as golden evidence in this report.

Methodology

Workflow

The given task was split into 3 stages:

1. Initial candidate documents retrieval using an inverted document-term matrix approach.
2. Ranking search results in order to obtain only relevant sentences to a given claim.
3. Deduce whether a given claim is Supported, Refuted by found sentences or any conclusion cannot be drawn.

Stage 1. Document Retrieval.

In the first stage, the main goal was to retrieve candidate documents that might be relevant to a given claim using a simple and fast method. TF-IDF score using the Vector Space Model is such a method that has been widely adopted by industry. Various implementations are available for public use. In this project, 3 such packages (Pylucene, Whoosh, Xapian) were considered. Their advantages and drawback are listed in the Table 1

Inverted index variations:

Xapian package was chosen for creating an index, and 5 different approaches were used for indexing. In order to compare variations of the index, document recall rate was used for search results. The standard recall rate was used, namely the total number of correctly found documents for all dev set claims divided by the total number of documents known as golden evidence on dev set. Summary of recall rates could be seen in Figure 1.

Comparison and experimental results		
Package	*Advantages-and-drawbacks*	*Result*
pylucene	+ Good documentation. - Requires Java and difficulty to install on MacOS.	Failed to install on MacOS
Whoosh	+ Good documentation. - Index size grows large in volume	It very slow during the search.
Xapian	- Basic documentation - Allows only 1 writer) + Relatively small index size and decent search speed + Allows storing additional document dimensions as the subject, keywords, etc.	chosen for the project

Table 1: Comparison of packages

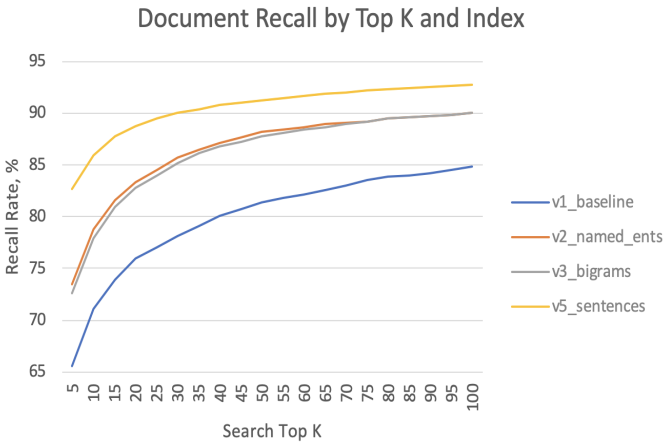

Figure 1: Document Recall Rate for different indices on dev set

Indices description

1. The first index was built based on wiki-pages by joining all sentences related to particular page.

   This approach could yield up to 85% document recall rate on top 100 search results.

   However, for many claims with named entities like film names the golden evidence was missed. That led us to the idea to include named entities in the index.
2. With the second approach, Spacy package was used to extract named entities and noun phrases from the first sentence of wiki-pages, and extracted entities were included in the index as separate tokens.

Search results gave 5% higher recall rate.

The main drawback of the approach was the calculation time of named entities that took 16 hours. Also, some wiki pages did not contain the main entities of the article in the first sentences.

3. To overcome issues from the second approach we used part of speech tagging from nltk package for the whole wiki-page. From POS tagging results we removed everything except nouns, adjectives and cardinals and included nonfiltered items as bigrams in the index.

   However, the performance of the index was slightly less than with second approach.

4. It was an experimental index based on sentences from documents found with the second index. This index revealed that among top search results there were "trashy" 2-3 word sentences that could not be used for evaluating claims. This index encouraged us to build sentence index for the whole corpus.

5. In the final approach wiki-corpus sentences with at least 5 tokens and 3 word-tokens were taken as documents and indexed. Additionally, wiki-page name was tokenized and extracted terms were added along with sentence text.

   That allowed to increase document recall further by 2-3% and reach 93% recall for Top 100 results on the dev set.

Analysis of missed golden evidence
With the sentence index there were left only 363 claims on Top100 results with missed golden evidence. Among them for 113 the golden evidence was fully missed. For another 250 cases the index yielded at least 1 document from the golden evidence. In other words, some of the sentences were found, while others were missed.
To further improve index performance a more thorough analysis is required that should be carried out on the level of missed golden evidence sentences.
Possible ways to improve search results might include:

- More thorough study of Xapian dimensions and using weights for named entities during search.
- Topic modelling for wiki-pages and including them into document.

Stage 2. Sentence Selection

Since there could be many relevant sentences returned by the search engine, the goal of this sub-system is to find the potential gold evidence to validate the given claim. The "Gold Evidences" or the best evidences are desired which contain the most relevant information to validate the claim. The main focus of this section was to match our sentence selection process with the gold standard.

Baseline Approach
Our most intuitive and baseline approach was to calculate the Cosine Similarity between each claim and retrieved evidence pair using Spacy package which internally converts the sentences into Wordvec and takes average of the similarity. Then we rank the most similar pairs and take the top-5 as our gold evidence. This, only resulted in Sentence Selection F1 of 19.9 on the given testset (Anuradha, Indukur, and Putta 2017).

Then, we moved on to improve this approach by using a Machine Learning classifier to recognise the similarity patterns. For each claim and evidence pair we calculate the following *features/relationships* to prepare the train and test datasets for the model.
Features
1. Largest Common Sequence: It is the largest sequence of tokens match between the claim and candidate sentence.
2. Cosine Similarity: We remove the stop words in the pairs and calculate the Cosine Similarity.
3. Common Keywords: We extract all the Nouns and Entities and calculate the length of matching keywords.

Sampling and Train Dataset preparation:
For a claim in the given train.json we retrieve the top 2 search results with the second index (index with named entities extracted from the 1st sentence of the wiki page). Retrieved documents are split into sentences and for each claim and sentence pair is formed. For each pair that is golden evidence the label 1 is assigned and remaining pairs are randomly sampled with label 0. Pairs are sampled in a way to ensure training set has balanced classes of positive and negative examples.

Predictions:
For every claim in the test dataset we retrieve the top-k results and calculate the above features for each claim and potential evidence pair. Then for each claim we only pick the sentences which are predicted as 1 as the final evidence for the claim. If there is claim such that all of its top-k results are predicted as 0's then we write its final label as "NOT ENOUGH INFO" and its evidence list is empty.

Comparing Performance of Sentence Selection Models		
Classifier	**Configuration**	**Validation-Accuracy**
Naive Bayes	Flat prior	0.727
Logistic Regression	Regularization C = 0.01 solver = 'lbfgs"	0.742
SVM	loss = 'hinge' regularization = 'L2' C = 0.002	0.739

Using the comparison as shown in the table above the Logistic Model was chosen for the Baseline and we got a score of Document F1 = 42.23 and Sentence F1 = 25.82 on CodaLab.

Our Gold Standard Approach
We soon realised that there is something wrong in our training data set generation as we were getting high accuracy on the validation set but very low Sentence F1 on the actual test set. Then, we devised a new method for sampling to prepare the training dataset for Sentence Selection using the intuition of probability theory. In our Baseline method the sampling method was naive and it did not consider the behaviour of our IR system. So, we define a new sampling method to prepare the dataset to train a sentence selection model as described below.

Sampling and Dataset Preparation:
We use sentence-based index and we sample 5000 claims

from train.json to ensure that distribution of claims' labels is balanced. The we run search and pull the top k results regardless of the fact it's golden evidence or not. Resulting pairs of claims and sentences are assigned label 1 if they match golden evidence, 0 otherwise. So, we end with blocks of size k with 0's and 1's for each claim in the train.json. This method ensures that the probability of our IR retrieving a gold evidence in the top-k results is preserved and we assume that it would behave in the same way on the test set. In addition to this, we considered the fact that our Machine Learning Classifiers do not consider the context or meaning of potential gold evidence to make the prediction. After some research on the current state-of-the-art work on Sentence selection, we realized that BERT the brainchild of Google is a tool we can leverage without having to build a complicated model.

Using and Fine Tuning BERT:

We used the pretrained weights of BERT-Base, Uncased: 12-layer, 768-hidden, 110M parameters and generated the training dataset using a Stratified sample from train.json of size 5000 in the above mentioned method resulting in 18k training examples. The BERT-Base was fine tuned using the structure method as described in the (Wang et al. 2018)on the MRPC template (i.e, format the data like the MRPC dataset). We chose a sequence length of 128 and batch size of 32 on our prepared train dataset. BERT method gave us a validation accuracy of 0.915 for Sentence Selection. On the CodaLab Ongoing Evaluation scoreboard it achieved our best score of 77.5 Document F1 and 68.9 Sentence F1.

As in our analysis below we test the performance of BERT on given devset using Score.py script by varying the top-k results retrieved from our index. Finally, we inferred that top-13 gave the best performance and it was our choice of top for the final submission.

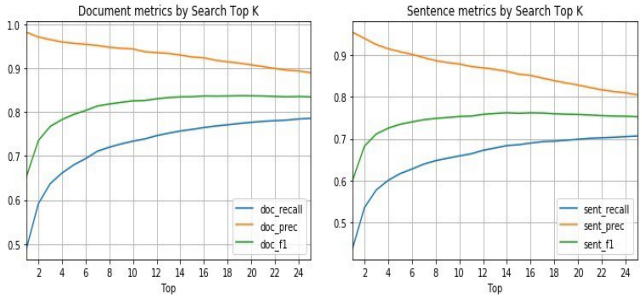

Figure 2: Performance of BERT on Devset.json to choose Top-k

Stage 3. Final Label Prediction:

Baseline Approach

Our Baseline system was a simple Multi Layered Perceptron Classifier with three hidden layers which takes relationships between the claim and evidence as input and outputs two probabilities for "SUPPORTS" and "REFUTES" labels. These relationships include *Jaccard's Similarity*, *Cosine Similarity*, *Length of Common Keywords* and *Longest Common Sequence*. The performance of the Baseline and other models is compared in the table 2.

Decomposable Attention Model:

Using the basic idea as described in (Parikh et al. 2016) we slightly modified it to suit our requirement of label predictions. We use SpaCy to tokenize the input text and construct a semantic vector for each of the token. Also, if a token has no semantic vector available it is assigned one randomly generated OOV vector(Parikh et al, 2016). We use the GloVe vectors from spaCy as weights in our embedding layer, and util functions for these tasks are borrowed from SpaCy source codeSpacy n.d.

Model Components:

1. Align: The claim and evidence are passed through the embedding layer before each going through a Bidirectional LSTM separately, which acts like an attention like mechanism in our architecture. This is the 'Decomposable' part and the dot product of the output of the embedded layer is normalised vertically and horizontally to obtain soft alignment structures.

2. Compare: Each of the phrase is compared to its aligned phrased while it passes through the TimeDistributed layer which is the higher dimension strength of association between them.

3. Combine: The resulting claim-¿ evidence and evidence-¿claim tensors which associate between them are aggregated into one.

4. Output: The result from the last step is passed step is passed through a dense layer with two outputs and applied with softmax activation. The outputs are probabilities for the 'SUPPORTS' and 'REFUTES' classes as our sentence selection model picks the 'NOT ENOUGH INFO' claims beforehand.

Below is a brief sketch of the Architecture of our Attention Model:

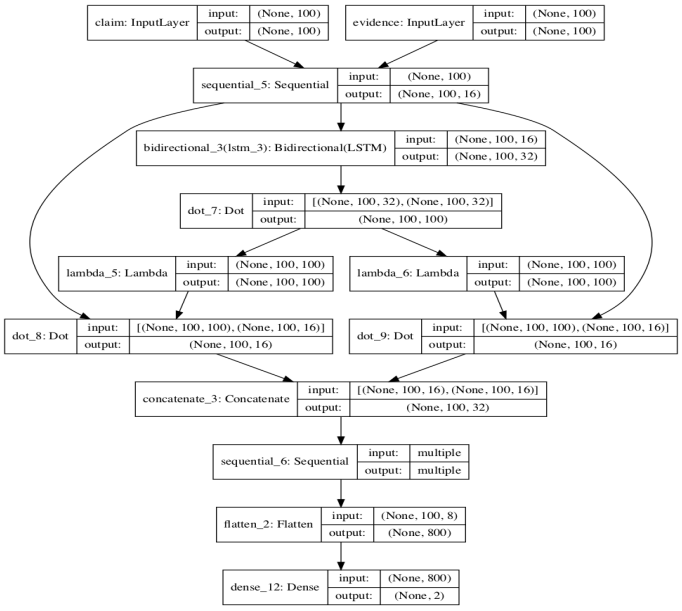

Figure 3: Model

BERT Model for Label Prediction:

Our baseline model was too simple for such complex task and it did not consider any context or meaning of the text. Next, our Decomposable Attention Model architecture was complicated and the hyper-parameters were hard to tune. It gave us a good validation accuracy but failed to match the good accuracy on the final test set, which made us realise that the architecture needs to be refined to suit our classification task of the claim. Thus, we opted for BERT which is pre-trained on millions of Wikipedia words and if we could tune these weights according to our NLP task and requirement it would give us promising re-

sults.(Devlin et al. 2018)

Setup:

As described in (Wang et al. 2018) task of RTE(Recognizing Textual Entailment) we created a parser which would format our training and test set into the RTE dataset format. In the training set we have pairs of claim and all its evidences concatenated, also, if it has many evidences we only consider the first four for faster training. We made sure that our sample from the train.json had equal class distribution.

Running and Tuning:

Using the $run_classifier.py$ script provided by Google we tune the weights of BERT BaseModel on a Machine with one 1 GPU and 2 CPUs with the configuration as described in the table below. In the table below all our Prediction models are compared according to their performance on validation and final test set.

Model	**Configuration**	**Validation-Accuracy**	**Codalab-Accuracy**
Baseline	-3hidden layers -(100,100,3)units -2outputsusing softmax Learning rate=1e-5	0.58	0.38
Attention Model	-Batch size = 76 -Epochs = 12 -Dropout = 0.2 -EarlyStopping on validation loss	0.84	0.48
BERT Tuned	-MaxSequence Len=256 -Batch size=16 -Learning rate=2e-5 -Epochs=5 -Uncased with text lowered	0.93	0.57

Table 2: Comparing Label Prediction Models Accuracy

Final System Pipeline

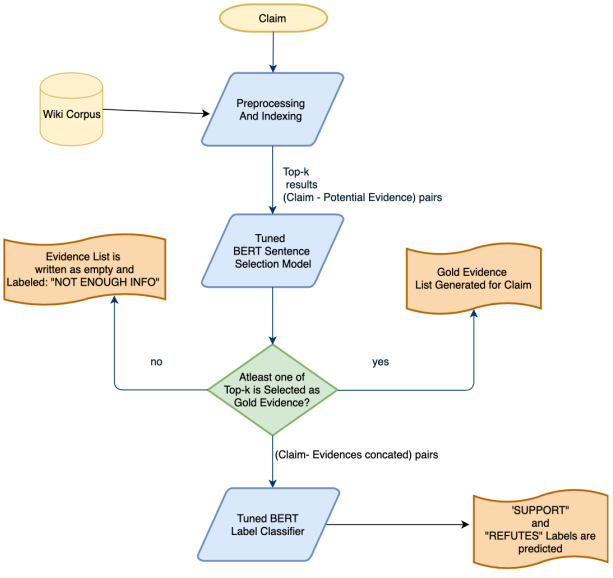

Figure 4: Flow and Pipeline our Final system with our best performing components.

As seen in the above pipeline, our system involves three main components. The Information Retrieval part, the Sentence Selection part and Final Label Predictor. The diagram briefly describes how different components are connected to each other and also how claim is classified into one of the three labels.

Error Analysis And Future Direction

Tuning performance of the document and sentence retrieval stage might yield noticeable growth of performance of the whole system. As mentioned above topic modeling might increase recall rate of the relevant documents. Also, experimenting with indexing and searching parameters might make search more precise.

Our Tuned BERT Model for sentence prediction was trained on a small sample of 5000 items, therefore, it was not able to generalize well on the Final Evaluation scoreboard. Given the computational resources, we would like tune it on the entire train dataset using the BERT-Large pretrained weights which would increase it's performance on the final testset.

Our Decomposable Attention Model has a complex architecture and tuning hyperpameters like the Dropout Rate, Number of Hidden Layers, Maximum Sequence Length and Batch Size was a tedious task. We would like to further tune this modified Attention Model and compare it's performance with BERT as future work.

One of the major issue with Tuned BERT for label prediction was choosing the best Sequence Length. If we concatenate all the evidences of each claim the longest sequence length was thousands of tokens. It would take a very long time to be tuned for such length finding the best set of evidence from list was challenge we could not address in this project.

In addition, for it to predict the labels on examples that it has not seen before we need to tune it on the entire train set on the BERT-Large weights which is computationally expensive. We suspect this must be the primary reason why our score fell in the Final Evaluation scoreboard.

Finally, our system is not scalable and not robust enough to predict unseen examples. We would like to work on tuning our models and conduct future research on best strategies to accurately predict on unseen example and scalability.

Conclusion

Automatic Fact Verification is a combination of Document Processing, Information Retrieval, Information Extraction, Semantic Analysis and Applied Machine Learning. Building an end to end automatic system would involve many components that need to interact with each other and there many strategies that can be implemented to achieve this goal. Finally, the most power tool and method is BERT as it can be generalized to any complex Text Analysis task without having to build complex architectures to predict and classify.

Code Repository

https://github.com/hima950/WSTA_FACT_VERIFICATION

References

Parikh, Ankur P. et al. (2016). "A Decomposable Attention Model for Natural Language Inference". In: *CoRR* abs/1606.01933. arXiv: 1606.01933. URL: http://arxiv.org/abs/1606.01933.

Anuradha, Dr. K., Himaja Indukur, and Tilak Putta (2017). "FEATURE ANALYSIS FOR SEMANTIC TEXTUAL SIMILARITY". In: *semanticscholar*. URL: https://pdfs.semanticscholar.org/19cb/54a5685b77d7e919ab9103c6dbe82e47a278.pdf.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Thorne, James et al. (2018). "FEVER: a Large-scale Dataset for Fact Extraction and VERification". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 809–819. DOI: 10.18653/v1/N18-1074. URL: https://www.aclweb.org/anthology/N18-1074.

Wang, Alex et al. (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pp. 353–355. URL: https://aclanthology.info/papers/W18-5446/w18-5446.

Spacy (n.d.). *Spacy Source Code*. URL: https://github.com/explosion/spaCy.