Automated Fact Checking System – Wed2PM Group 2

Xueyi Chen and **Yuqi Deng** and **Meilun Yao** and **Zheyuan Zhao**

Abstract

With the significant impact of climate change and its misinformation increasing, we developed an automated fact-checking system designed to retrieve relevant evidence and classify claims into different classes. The system implements various methods for the tasks. The performance of different models was shown in the result.

1 Introduction

Climate change is a hot topic that has worldwide concern. The misleading statements not only distort public opinion but also influence policy decisions. Therefore, developing an automated fact-checking system becomes a significant problem.

In this report, we propose a system that combines evidence retrieval (task 1) and text classification (task 2). Section 2 described all the approaches we tried. In section 3, the experiment details and evaluation method were shown. Then we gave the result and conclusion in sections 4 and 5.

2 Approach

2.1 Data Preprocessing

Our team imported a variety of tools and resources from the NLTK library, including deactivators, splitters, morphological reducers, and lexical annotators, and downloaded the NLTK's Resource Kit, which includes the 'wordnet', 'stopwords', 'averaged perceptron tagger', and 'punkt'. The text data is split into individual tokens (words) using the NLTK tokenizer. Words are reduced to their base or root form using NLTK's WordNet lemmatizer.

In the evidence data, 'Number of English texts = 1184563', 'total evidence = 1208827', English is in the majority, so in order to simplify the processing, our team chose to remove the non-English words. All capital letters were made lower case, and then the prefixes and suffixes were removed from each word and the words were labeled with their lexical properties.

2.2 Feature Engineering

Thought that the project involves the task of information retrieval, our team uses neural network models, which require capturing complex sequential information and long-distance dependencies. To achieve this, we have experimented with techniques such as BOW, TF-IDF, Word2Vec, and BM25.

2.2.1 Bag of Words (BOW)

Bag of words (BOW) is a simple and powerful text feature extraction, which can transform the text data into numerical feature vectors based on the frequency of words.(Qader et al., 2019) BOW calculates the frequency of each word in the document, and uses these words' frequency as a vector to represent the document.

In our implementation, we use the CountVectorizer function in sklearn library to vectorize all the evidence and claims. This approach is chosen for its simplicity and efficiency in transforming text data into structured vectors, which is crucial for natural language learning tasks.

2.2.2 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a powerful text feature extraction method that evaluates the importance of a word in a document based on statistics by converting text data into numerical feature vectors. This is very suitable for the tasks of information retrieval and document similarity calculation in this project.

In practice, our team used the `TfidfVectorizer` class from Python's scikit-learn package. We created one TF-IDF vectorizer for evidence text and another for claim text, both using IDF to adjust the weights of the words. For the evidence vectorizer, we removed stop words, whereas for the claim vectorizer, we retained stop words to better

capture the language used in claims. The evidence vectorizer was then fitted to the combined set of training claim text and full evidence text, allowing it to learn the words and their importance across all texts. Finally, the training claim text was transformed into TF-IDF feature vectors using the claim vectorizer, which was fitted to the training claim text. This process encompasses both vocabulary learning and feature vectorization.

2.2.3 Word2Vec

Word2Vec is an advanced word embedding technique that represents words as continuous vectors in vector space, capturing semantic relationships and contextual similarities between words. (Rong, 2014)

Word2Vec is based on the idea of learning word vectors from context information. Two main approaches to get this context information are Continuous Bag of Words (CBOW) and Skip-gram.

CBOW predicts the target word based on its context words. In contrast, skip-gram predicts context words based on the target word.

In the implementation, we used Word2Vec function in gensim library. We used all the evidence and train claims as our train data for Word2Vec. The vocabulary is still small, which may not produce a perfect vectorization model.

2.2.4 BM25 (Best Matching 25)

BM25 (Best Matching 25) is the classic algorithm for calculating the similarity score between a query and a document, widely used for effectively ranking documents by relevance. Here is the scoring function of BM25(Whissell and Clarke, 2011):

$$Score(Q, X_i) = \sum_{j \in Q} \frac{tf_{ij}(k1+1)}{tf_{ij} + k1\left((1-b) + b\frac{dl_i}{avgdl}\right)} \log\left(\frac{n}{n_j}\right), \tag{1}$$

In our implementation we used `BM25Okapi` class from Python's rank-bm25 package to implement BM25. We first picked up the id and its corresponding text from the evidence dataset and the training claims dataset. Then, we tokenised the texts to prepare them for BM25. The tokenisation will split each document into individual words. Next, we initialised individual BM25 models for evidence and cliams using the `BM25Okapi` class. These models can efficiently search for relevant documents based on the text of the query. We then iterated over each claim in the development dataset. During the iteration we tokenise the current claim text. Then we use the BM25 model to find the similarity between the current claim and all the evidences to get the most relevant k evidences, and find the most similar claim to the current claim in the training model and assign its label to the current claim. Finally, we write the output to a JSON file for further evaluation and analysis.

2.2.5 Cosine Similarity

Cosine similarity works well in high-dimensional spaces and provides a measure invariant to vector length, which is very suitable for comparing evidence and claims in this project. Cosine similarity between TF-IDF matrices for claims and evidence is implemented incrementally to avoid memory error. It breaks the vector matrix of claims into smaller chunks. The number of rows to process at a time is set to 50, which balances the running time and the memory usage. It then computes the cosine similarity for each chunk with the vector matrix of evidence. The cosine similarity between the claims and evidence is defined using the dot product of their vectors. Finally, the results are combined to form a complete similarity matrix for further analysis.

2.2.6 Negative Sampling

Negative sampling is a key technique to improve model performance and train models to efficiently handle large amounts of data, especially in information retrieval tasks. In the case of processing more than 12 million mostly irrelevant evidence, negative sampling solves the challenge of computing softmax functions on large vocabularies. Instead of considering all possible negative examples, negative sampling randomly selects a few negative samples for each positive instance, significantly reducing the amount of computation.

How It Works: The function is implemented in 2 modes, train mode and develop mode. In train mode, the function selects evidence candidates from the 201st to the 400th highest cosine similarity scores. This avoids the top 200 highest scores, which are likely the most similar and often positive examples. This increases the diversity of training datasets. In the developing mode, the function selects the top 200 evidence candidates. A loss function formula is used to distinguish the positive and negative samples. This involves maximizing the probability of the positive pair and minimizing the probability of the negative pairs.

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) = -\log \frac{e^{sim(q_i, p_i^+)}}{e^{sim(q_i, p_i^+)} + \sum_{j=1}^{n} e^{sim(q_i, p_{i,j}^-)}}. \quad (2)$$

2.3 Model

2.3.1 Feedforward Neural Network (FFNN)

The Feedforward Neural Network (FFNN) model is widely used in text classification. (Laudani et al., 2015) The architecture in this FFNN model includes an input layer, a fully connected layer, a dropout layer, another fully connected layer, and an output layer with softmax function.

The input dim of the model was the length of the train data. The hidden dim, as a hyper-parameter, was set to 128. The output dim in this task is the number of different labels, which is 4.

In the training process, the model iterated in each epoch with the following steps: Firstly, get the prediction distribution with the input data through the model. Secondly, calculate the error between prediction and true labels with cross-entropy loss function. Then, calculate the gradient and propagate backward through the network. Finally, update the parameters with the gradient to get the smallest loss.

In this way, the FFNN model continuously adjusts its parameters to improve the classification accuracy of climate change claims.

2.3.2 RNN

The recurrent Neural Network (RNN) model is used to implement both tasks. For task 1, the model is designed to distinguish between relevant and irrelevant pairs using cosine similarity and negative sampling techniques. For task 2, labels are encoded into numerical values and converted to one-hot vectors. Both models are trained with a focus on handling class imbalance and preventing overfitting. The use of early stopping and cosine decay learning rates helps in achieving better generalization and stable training.

Model Architecture

1. Embedding Layer: Converts input tokens (words) into dense vectors of fixed size (200 dimensions for task 1, 50 dimensions for task 2). 2. First RNN Layer with 400 units for task 1 and 100 units for task 2, configured to return sequences. 3. Second RNN Layer with 400 units for task 1 and 100 units for task 2. 4. Dropout Layer with a dropout rate of 0.1 prevents overfitting by randomly setting a portion of input units to zero during training. 5. Dense Layer: Task 1 sets a fully connected layer with 400 units and ReLU activation. Task 2 sets a fully connected layer with 100 units and tanh activation. This layer helps in learning complex features from the RNN outputs. 6. Output Layer with a single unit and sigmoid activation for task 1, which outputs the probability of a claim-evidence pair being relevant. Task 2 sets a dense layer with 4 units (one for each class) and softmax activation, which outputs the probabilities of each class.

For task 1, the binary cross-entropy loss is used to measure the error between predicted and true labels as it becomes a binary classification after processing negative sampling. For task 2, categorical cross-entropy loss is used as it expects to output data into 4 categories. Adam optimizer with a cosine decay learning rate is used for both tasks. This helps to dynamically adjust the learning rate during training. For task 1 accuracy, precision, and recall are used to evaluate the model's performance as the final evaluation F1-score based on both precision and recall. For task 2, only accuracy is used.

Training Process

Batch Size: For task 1, 1000 samples per gradient update. For task 2, 500 samples per gradient update. Due to the large amount of data of evidence, class weights are calculated to handle class imbalance, for task 1 assigning more weight to the positive class (relevant pairs) and for task 2 more weight is assigned to underrepresented classes. Early Stopping: A callback that stops training when the validation loss stops improving for three continuous epochs and restores the best weights. Checkpoint: A callback to save the best model based on validation loss.

The RNN model is powerful for sequential data processing, however, there are some limitations to its use in this project. The maximum sequence length of 230 tokens used in this project could still be too long for the SimpleRNN layers to effectively capture dependencies across the entire sequence. The performance of RNNs can be sensitive to the length of input sequences. Padding sequences to a fixed maximum length (230 or 350 tokens in this project) can introduce noise and reduce the model's effectiveness. SimpleRNN layers are likely to struggle to capture long-term dependencies in the data even with 400 units of hidden dimensions. The model may not perform well in tasks that require understanding remote contexts, such as complex claim-evidence relationships that

contain multiple sentences. The prediction result of retrieving evidence is extremely low is highly due to this.

2.3.3 LSTM

LSTM has more forget gates, input gates and output gates to control the flow and storage of information as compared to RNN, which relies only on the hidden state to preserve information. Therefore, LSTM is not as prone to gradient vanishing.

In our LSTM model design, we created a sequential model using `Sequential`. We added an embedding layer with a vocabulary size of 10,000, an embedding dimension of 200, and an input sequence length of 500. Next, we added an LSTM layer with 400 units, setting both dropout and recurrent dropout to 0.2. We then included a fully connected layer with a sigmoid activation function, with the number of output cells equal to the number of labeled categories. The model was compiled using categorical crossentropy as the loss function, adam as the optimizer, and accuracy as the evaluation metric. For training, we split the data into training and validation sets with an 8:2 ratio and addressed category imbalance by applying category weights. We designed the weights based on the ratio of the number of samples in each category relative to the total number of samples to balance the category imbalance problem so that a few categories get more attention in training.

3 Experiments

3.1 Select top k

We used cosine similarity combined with TF-IDF and BM25, and evaluated the performance with different top-K values. As the value of top k does not affect its accuracy, we need to get the best performing top k value depending on the F-score. å From Figure 2 we observe that the TF-IDF reaches the peak of F1-score which is 0.1051 at top 2. After that all the F-score decreases with the increase in the value of k. And the BM25 peaked at top 3, with its F-score (0.06628) slightly higher than top 2's (0.06617) by a value of 0.00011. So we chose a k value of 3 for BM25 and top 2 for TF-IDF.

3.2 Imbalance Data Handling

Due to the imbalance label distribution in the training set, we tried to handle it to avoid the tendency of the dominant class. By addressing imbalanced datasets, we can enhance the model's ability to recognize all classes, thereby improving overall performance.

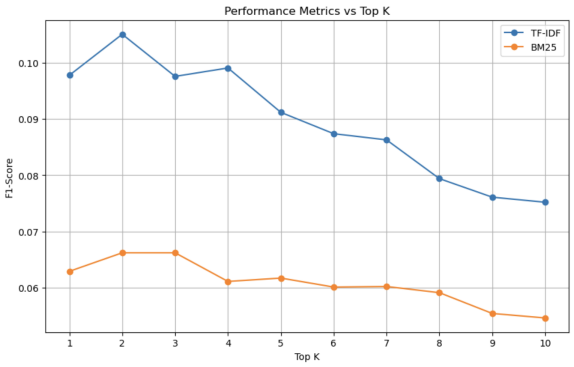

Figure 1: Performance Metrics vs Top K for TF-IDF and BM25

We used SMOTE (Synthetic Minority Oversampling Technique) which generates new synthetic minority samples by interpolating between existing minority samples and their nearest neighbours. (Yi et al., 2022)

Table1 compares the performance of FFNN under original data and balanced data processed by SMOTE. The balanced data is not that helpful in improving the accuracy, this may be due to the simple FFNN can not catching too much data, and the imbalance in the dev set.

Method	F-score	Accuracy	Harmonic Mean
FFNN (original)	**0.0580**	**0.4286**	**0.1021**
FFNN (balance)	0.0580	0.2987	0.0971

Table 1: Performance Metrics in FFNN

3.3 DropOut Parameter and Voting System

Considering that this may be due to the fact that RNNs are prone to gradient vanishing or gradient explosion when dealing with long sequential data, we constructed an LSTM using the same steps in order to better capture the relationships between long sentences. Accuracy is only 0.01753 and Mean of F and A is 0.07672 at the moment, therefore in the part of Task 2, we initialize the label votes dictionary to record the number of votes for each label. The labels based on the two most similar training statements are voted on and the label with the most votes is selected as the final predicted label. This method can effectively combine the label information of similar statements to improve the accuracy of the prediction results

The overall performance was improved, so based on that, we made some comparisons of different drop out ratio.

	precision	recall	f1-score	support
DISPUTED	0.14	0.56	0.22	18
NOT_ENOUGH_INFO	0.20	0.10	0.13	41
REFUTES	0.26	0.41	0.32	27
SUPPORTS	0.52	0.16	0.25	68
accuracy			0.23	154

Figure 2: DropOut=0.1

	precision	recall	f1-score	support
DISPUTED	0.25	0.17	0.20	18
NOT_ENOUGH_INFO	0.26	0.37	0.30	41
REFUTES	0.22	0.44	0.30	27
SUPPORTS	0.50	0.22	0.31	68
accuracy			0.29	154

Figure 3: DropOut=0.2

	precision	recall	f1-score	support
DISPUTED	0.16	0.50	0.25	18
NOT_ENOUGH_INFO	0.24	0.15	0.18	41
REFUTES	0.26	0.41	0.31	27
SUPPORTS	0.55	0.25	0.34	68
accuracy			0.28	154

Figure 4: DropOut=0.3

We tested with dropouts of 0.1, 0.2, and 0.3, and then printed out a Classification Report, and found that performance was best with a dropout of 0.2.

3.4 Evaluation method

To evaluate the performance of different models, we used eval.py to do a quantitative analysis.

Three main evaluative criteria are F1-score of evidence retrieval, the accuracy of text classification, and the harmonic mean of F1-score and accuracy.

The F1-score evaluates the performance in the evidence retrieval task. It combined precision and recall. The claim classification accuracy measures the proportion of claims that are correctly classified by the system. With harmonic mean, the performance of evidence retrieval and claim classification got an overall assessment.

4 Results

As can be seen from Table 2, the best performance when combined with cosine similarity is the TF-IDF with a harmonic mean of 0.1721. Other methods do not combine well with it, so the values of cosine similarity will be based on the combination with TF-IDF.

Table 3 compares the performance of the different methods. The best performers are still the methods combining TF-IDF and cosine similarity with the highest F-score (0.1051) and harmonic mean (0.1721), and the highest accuracy (0.4740).In contrast RNN performs the worst with an F-score of 0.0066 and a harmonic mean of 0.0131.Other methods such as LSTM and FFNN variants performed moderately well and did not outperform the cosine similarity method.

Method	F-score	Accuracy	Harmonic Mean
BM25	0.0663	0.4091	0.1141
TF-IDF	**0.1051**	**0.4740**	**0.1721**
BOW	0.0744	0.3896	0.1249

Table 2: Performance Metrics for Methods using Cosine Similarity

Method	F-score	Accuracy	Harmonic Mean
RNN	0.0066	0.3961	0.0131
Cosine Similarity	**0.1051**	**0.4740**	**0.1721**
LSTM (Voting System)	0.0491	0.2662	0.0829
FFNN (only claim)	0.0580	0.4286	0.1021
FFNN (with evidence)	0.0580	0.3247	0.0984

Table 3: Performance Metrics for Different Methods

Table 4 compares the performance on the test set. The best performance is FFNN with a 0.644 F1-score and a 0.4545 Accuracy. With the same method of evidence retrieval used in both attempts, FFNN got a better accuracy in classification. The ability of FFNN in classification is greater compared with simply using cosine similarity.

Method	F-score	Accuracy	Harmonic Mean
FFNN	**0.0644**	**0.4545**	**0.1128**
Cosine Similarity	0.0644	0.3498	0.1088

Table 4: Performance Metrics for FFNN and Cosine Similarity in Codalab

5 Conclusion

In this project, we developed an automated fact-checking system, exploring and comparing multiple text similarity computation methods to identify the optimal K-values and method combinations. We also trained RNN, LSTM, and FFNN models. Due to time constraints, we ultimately selecting the FFNN model for the Codalab competition, where our system ranked 28th. The training set consisted of only 1228 samples, and the test set had 154 samples, making the dataset small and prone to overfitting. Future work should focus on improving the architectural design of the FFNN and the decision-making process or incorporating a voting system or experimenting with Bi-LSTM and Bi-RNN designs could further enhance the system's performance.

6 Team contributions

Our team analyze the dataset and discuss the preprocess approach, and then each of us built on that to experiment and tune the different models.

Team Member	Contributions
Xueyi Chen(1174341)	Implementing incremental cosine similarity, negative sampling and RNN model.
Meilun Yao(1076213)	Implementing LSTM model with a voting system.
Zheyuan Zhao (1487244)	Implementing Word2Vec for vectorization, SMOTE for data balance, and FFNN model.
Yuqi Deng (1138222)	Implementing cosine similarity with TF-IDF, BOW and BM25.

Table 5: Team Members and Their Contributions

References

Antonino Laudani, Gabriele Maria Lozito, Francesco Riganti Fulginei, and Alessandro Salvini. 2015. On training efficiency and computational costs of a feed forward neural network: A review. *Computational intelligence and neuroscience*, 2015:83–83.

Wisam A. Qader, Musa M. Ameen, and Bilal I. Ahmed. 2019. An overview of bag of words;importance, implementation, applications, and challenges. In *2019 International Engineering Conference (IEC)*, pages 200–204.

Xin Rong. 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

John S Whissell and Charles LA Clarke. 2011. Improving document clustering using okapi bm25 feature weighting. *Information retrieval*, 14:466–487.

Xinkai Yi, Yingying Xu, Qian Hu, Sujatha Krishnamoorthy, Wei Li, and Zhenzhou Tang. 2022. Asn-smote: a synthetic minority oversampling method with adaptive qualified synthesizer selection. *Complex & Intelligent Systems*, 8(3):2247–2272.