Training and Fine-Tuning Transformer-Encoder Models for a Fact-Checking System

Fri3PM_Group3

Abstract

This project developed an automated fact-checking system to verify climate science claims using a corpus of evidence. The system performs two main tasks: evidence retrieval and claim classification. Evidence retrieval identifies relevant evidence from a large knowledge source, while claim classification categorizes claims into SUPPORTS, REFUTES, NOT_ENOUGH_INFO, or DISPUTED. Preprocessing steps included lowercasing, lemmatization, stop word removal, and applying TF-IDF and cosine similarity. We used transformer architecture for the models, incorporating text and position embeddings. Our results show effective claim verification, though challenges remain in generality and data imbalance. Future improvements could explore alternative preprocessing and refined model architectures.

1 Introduction

1.1 Background

The field of Natural Language Processing (NLP) has seen significant advancements in recent years, particularly in the area of automated fact-checking. Automated fact-checking is crucial for addressing the increasing volume of misinformation, especially in domains such as climate science where unverified statements can distort public opinion and policy decisions. The challenge lies in developing systems that can accurately verify claims against reliable sources of evidence, a task that involves complex processes of evidence retrieval and claim classification.

1.2 Project Aims

The primary goal of this project is to develop an automated fact-checking system that can accurately verify claims using a given corpus of evidence. The system is designed to perform two main tasks:

Evidence Retrieval: Given a claim, the system must search through a large knowledge source (a corpus of evidence passages) and identify the most relevant pieces of evidence.

Claim Classification: Once the relevant evidence is retrieved, the system must classify the claim based on this evidence. The classification should fall into one of four categories: SUPPORTS, REFUTES, NOT_ENOUGH_INFO, or DISPUTED.

2 Literature Review

The application of Fact-Checking systems, evidence retrieval, and claim verification offers valuable insights for developing an effective automated fact-checking system.

For instance, Andreas Vlachos and James Thorne(Thorne and Vlachos, 2018) discuss key methodologies and challenges in verifying claims using large evidence corpora. They highlight the importance of natural language inference and stance detection in building effective systems.

Further, the detection of check-worthy claims is a crucial precursor to evidence retrieval and claim classification. Israa Jaradat and colleagues(Jaradat et al., 2018) emphasize the importance of identifying claims worth fact-checking. Their system, ClaimRank, focuses on preprocessing steps necessary for effective evidence retrieval and claim classification, showcasing its performance across multiple datasets.

Additionally, the role of context in fact-checking cannot be understated. Shaden Shaar and Firoj Alam(Shaar et al., 2021) introduce methods for detecting previously fact-checked claims by incorporating contextual information, underscoring the complexity involved in claim verification. Their research highlights the necessity of contextual analysis in ensuring the accuracy of automated fact-checking systems, as the same claim might have different truth values depending on its context.

3 Data Preprocessing

Data preprocessing is the necessary approach that any language processing must have, the purpose of data preprocessing is to break compositional documents into individual components for understanding the language. As the model training side, the dataset with normative data preprocessing could obviously improve the model performance. Detailly it might reduce the training time cost and increase the prediction accuracy. In this project, the major approaches of data preprocessing are lowercase words, lemmatization, removing stop words, Term Frequency Inverse Document Frequency (TF-IDF), Cosine Similarity, and data reconstruction.

3.1 Simple Data Preprocessing

The stopwords and lemmatizer are both from NLTK (Natural Language Toolkit). NLTK is a free, open-source library for language processing, it provides easy-to-use interfaces for many corpora and lexical resources which include stopwords and lemmatizers used in this project(Bird et al., 2024).

Lemmatization is the sequence that is widely used for data preprocessing in machine learning and NLP (natural language processing). In natural language processing lemmatization techniques transform and identify words into lemmas which are base or root forms of the words. Compared to stemming which applies simpler rules to chop off prefixes or suffixes, lemmatization is more popular since it brings context to the words and does the morphological analysis of the words(Balakrishnan and Lloyd-Yemoh, 2020). In the project, the Lemmatization is done by "stem.wordnet.WordNetLemmatizer()" from NLTK.

Additionally, stopwords will be removed from the dataset. Stopwords are frequently removed from the dataset for the natural language processing project. The library "nltk.corpus" provides a list of stopwords that contain 179 common words. Removing the stopwords usually draws the model more attention to important words, or content words which obviously improved the accuracy and relevance of NLP tasks(Fan et al., 2023). It is important to notice that before removing the stopwords, it is necessary to lowercase all the words in the data set since stopwords in the NLTK library are all lowercase.

3.2 TF-IDF

Term Frequency Inverse Document Frequency (TF-IDF) is widely used as a standard weighting scheme for information retrieval, it weights the words on the document by term frequency (TF) and the inverse document frequency (IDF) which is represented as the equation 1 where "w" represents the single word in the document and the "d" represents the document(Aizawa, 2003). In this project, the TF-IDF is applied to measure the importance and relevancy of a word in a series or corpus to a text by using TfidfVectorizer from the Scikit-learn library. The TfidfVectorizer provides the transform function to transform documents to the document-term matrix in which the row corresponds to a document, the column represents a word in the vocabulary and each element is the TF-IDF score of that word in the corresponding document(Aizawa, 2003). The equation 1 shows the calculation of the TF-IDF score which involves two terms: Term Frequency (TF) and Inverse Document Frequency (IDF).

The term frequency (equation 2) represents the number of words "w" that exist in document "d" divided by the total number of words in document "d". As term frequency side, if the words appear in the document more frequently it becomes more relevant. In other words, the weight of a word that occurs in a document is simply proportional to the term frequency.(Aizawa, 2003)

The Inverse Document Frequency (equation 3) represents the relevancy of the word which is the log function of the total number of documents divided by the number of documents that contain the word "w". In Inverse Document Frequency, the word that appears in more documents reflects the word has low relevancy and importance in specific documents. (Aizawa, 2003)

$$tf - idf(w,d) = tf(w,d) \cdot idf(w) \tag{1}$$

$$tf(w,d) = \frac{f_{w,d}}{\sum_{t' \in d} f_{t',d}} \tag{2}$$

$$idf(w) = \log\left(\frac{|D|}{df_w}\right) \tag{3}$$

3.3 Data Reconstruction

After the TF-IDF matrix is collected for train, test, dev, and evidence date, the cosine similarity between dataset train, test, dev, with evidence is calculated by function cosine similarity from Scikit-learn library. Cosine Similarity is the distance with dimensions representing features of the data object.

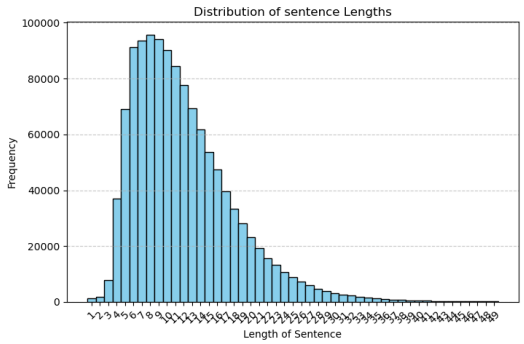

Figure 1: Sentence Length Frequency

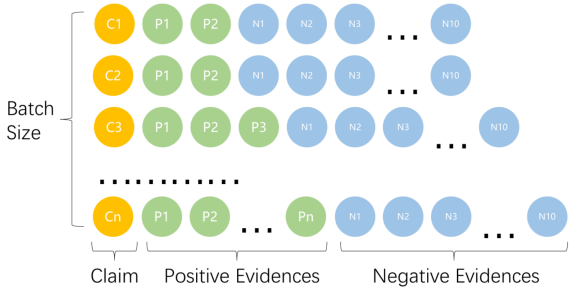

Figure 2: Input data construction

As Equation 4 shows, the cosine similarity is the dot product of vectors A and B divided by the product of magnitudes (length) of the vectors A and B (Pedregosa et al., 2011). Since the TF-IDF matrix sets the L2 norm as default, the sum of squares of vector elements is 1. Therefore, the cosine similarity between two TF-IDF vectors is equivalent to a linear kernel which is the dot product between two vectors (Aizawa, 2003). Then, the Top K negative evidence will be selected by the cosine similarity which the K is determined by the batch size.

At last, each sentence is padding with a padding size of 30. As Figure 1 shows, the length(number of words) of most of the sentences has the highest frequency between 3 to 30. In this case, the padding size 30 will cover most of the sentence.

$$CosineSimilarity(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \quad (4)$$

4 Methodology

We divide our system into two steps: a retrieval task and a classification task.

4.1 Retrieval Task

The simplest way to apply a Fact-Checking system is to transform the tasks into a text classification problem. This involves sorting the evidence passages and ranking them based on the likelihood that each passage falls into the relevant category,(Yates et al., 2021) which is positive and negative in our trial. The first application of BERT to text ranking was proposed by Nogueira and Cho, 2019, where they used a BERT model to rerank candidate passages retrieved from an initial keyword search engine, indicating that such architecture can achieve significant performance in the Query-Answer system.

The training corpus of our task contains a mess of evidence passages for candidate selection, making it challenging to train the model effectively and efficiently in practice. The Negative Sampling approach is applied in our experiment as proposed by Mikolov et al., 2013. Specifically, in our implementation, for each claim, we include its relevant evidence passages as positive samples, ensuring the model learns to identify true supporting information. To balance the training process, we select a subset of irrelevant evidence passages as negative samples based on the TF-IDF cosine similarities produced during the data preprocessing phase. This strategy not only helps in reducing the computation burden but also improves the model's ability to distinguish between relevant and irrelevant information. In this scenario, as shown in Figure 2, we eventually generate b(batch size) input clams, denoted by the yellow dots, and m+n evidence instances denoted by green and blue dots, where m is the number of positive evidence given by the training dataset and n is a hyperparameter of the negative evidence passages from sampling. For each training batch, we concatenate the evidence instances into a batch evidence matrix, which simplifies the calculation of vector similarity.

Regarding model selection, our Fact-Checking model consists of an input embedding layer, a positional encoding layer and a transformer encoder block as proposed by Vaswani et al., 2017. Particularly in our system. We use such a neural network architecture for the input clams and another identical model for the evidence passages inspired by Garg et al., 2020. The rationale behind using two models is to allow for independent encoding of the claims and evidence passages, capturing the distinct contextual nuances and semantic features of each, enabling the model to better align and compare the encoded representations of claims and evidence, thereby enhancing the accuracy and ro-

bustness of the similarity calculation and the overall Fact-Checking process. The output of these two models is used to calculate similarity scores later. We experimented with different similarity measures, including Cosine Similarity, Dot product and Manhattan distance, to evaluate their performance.

We use the output from the [CLS] token (the 0th dimension of the transformer output), as inspired by (Devlin et al., 2018), to compute the loss as shown in Equation 5,

$$L = \frac{1}{n}\sum_{i=1}^{n} -log(P_i) \tag{5}$$

where P_i denotes the probability of a ground truth evidence passage and n refers to the total number of positive instances in a single batch. This approach leverages the [CLS] token's representation, which captures the aggregated information of the entire sentence. By minimizing the loss, we aim to increase the probability of the ground truth instances while simultaneously decreasing the probability of the negative instances.

4.2 Classification Task

In terms of the classification task, during the training, models typically learn from perfect context containing ground truth passages, but during inference, they rely on the context generated from the previous retrieval model, which may introduce irrelevant or noisy passages that produce a negative effect on the model's performance. To mitigate this issue, a single input data for the classification model is the combination of the claim vector, ground truth evidence vectors and evidence passages we have retrieved previously via the retrieval model, separated by a special token [sep]. In this case, we simulate the real-world scenario in the training phase where the model must identify relevant information from a mixture of evidence, thereby mitigating exposure bias and enhancing its robustness and accuracy in practical applications. This approach is inspired by Bengio et al., 2015.

The model employed for classification is identical to the retrieval model, except for a fully connected layer on top of the transformer encoder, followed by a softmax layer for classification purposes. The loss is computed using CrossEntropyLoss. It is worth mentioning that the distribution of class labels in the training set is highly imbalanced, which biases the model towards predicting the majority class. To mitigate this issue, we utilize a weighted loss function based on the distribution of class labels in the training set.

5 Experiments

5.1 Experiments Setup

We conducted our experiments based on a machine running Ubuntu 22.04, equipped with an RTX 4090 GPU with 24GB of graphic memory. The code was implemented using Python 3.10 with PyTorch 2.1.0 and CUDA 12.1.

5.2 Training and Fine-Tuning

The training time for the retrieval model ranges from 20 to 30 minutes, while the classification model takes approximately 10 to 15 minutes.

Our implementation involves a number of hyperparameters, spanning both training parameter settings and model parameter settings. The table 1 shows the training parameter settings we have tried in our coding process, Table 2 describes two different parameter sets of our model. During the experiment, we randomly combined the parameters in the table many times in order to get the best hyperparameter setting.

Tried Parameters	**Values**
Number of negative passages for retrieval model	10,20
Number of candidates for evaluating the retrieval model	5,10
Number of negative predictions for classification model	3,5
Training Batch Size	8,16,32,64,128

Table 1: Training parameter settings

Transformer.Encoder	**Setting 1**	**Setting 2**
Word Embedding Size	256	512
Encoder Hidden Size	256	512
Encoder Attention Head	4	8
Encoder Layer	4	6
Encoder Layer Droupout	0.3	0.4

Table 2: Two sets of model settings

The training process of the model has encountered significant challenges, indicating that the conventional methods previously used for training a

model are no longer effective for this transformer. We manually use "del" in our implementation to delete unused variables to release the GPU memory for the next training step. Eventually, we save the best model which produces the best F1 score for both the retrieval model and the classification model.

6 Results and Discussion

When evaluating and testing the model, we compute the similarity score of the claim and top k candidates (the same k as in the training phase, based on the preprocessing similarity scores), the final result is then selected from the top 5 similarity scores based on the model's outputs. As shown in Table 3, the model setting 2 (refer to Table 2) with the top 10 negative samples based on the preprocessing cosine similarity score outperforms the others in our experiments. The possible reason why a more complex model achieves the best performance could be the fact that it has a greater capacity to capture patterns and relationships within the data. Additionally, a deeper architecture enables it to learn higher-level representations.

Retrival model	**Top K negative**	**F1 score**
Transformer(Setting 1)	10	0.0800
Transformer(Setting 1)	20	0.0782
Transformer(Setting 2)	10	0.0921
Transformer(Setting 2)	20	0.0871

Table 3: F1 score of the retrieval model on development dataset

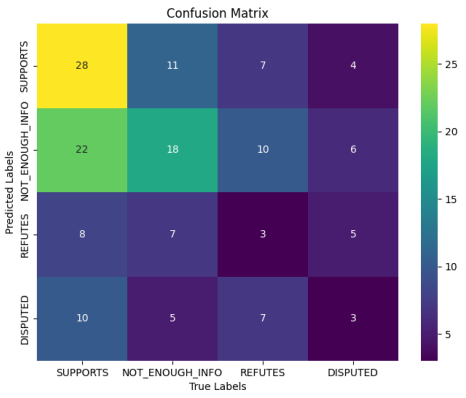

Figure 3: Confusion matrix of the classification model

Regarding the classification model, it is beneficial to use a simple model as in setting 1 (refer to Table 2), otherwise, the model tends to predict the majority class label, even though we used the weighted loss. Figure 3 demonstrates the confusion matrix of the classification model. Although the F1 score on the development set is 0.319, it actually forces the model to learn the diversity of the distribution of input data as we expected. Achieving a relatively high F1 score can be challenging due to the model being trained from scratch and the input data being dominated by the retrieval model and the preprocessing steps.

7 Conclusion

In this project, we developed an automated system to verify climate science statements, performing two primary tasks: evidence retrieval and claim classification. Evidence retrieval finds relevant information from a large knowledge source, while claim classification sorts claims into four categories for evaluation. Our system achieved good results on the development set.

Our data preprocessing steps included lowercasing, lemmatization, removing stop words, and applying TF-IDF and Cosine Similarity, which effectively reduced training time and boosted accuracy. Then we used transformer architecture as our major approach, including text embedding and position embedding. The negative sampling approach is applied for data selection.

However, our project still faces some limitations. The most significant issue is that the retrieval and classification results are highly influenced by TF-IDF similarity scores, which may not always capture the most relevant information. The model's generality and transferability are also concerns, as its performance could decline on other datasets. Data imbalance, where the training set and the evidence corpus are not evenly distributed, also affects classification performance. Furthermore, our processing capabilities are constrained by limited GPU memory.

To improve performance, we can explore alternative preprocessing methods and investigate better retrieval models. Experimenting with architectures such as poly-encoders or hybrid models could improve retrieval accuracy and efficiency by capturing more nuanced relationships between claims and evidence. Additionally, fine-tuning parameters of Scikit-Learn and Pythorch training procedures, such as adjusting the learning rate or model complexity, may also improve performance.

Contributions

Each member's contribution:

Yizhi Liao(1260566) Major contributions to implementing the model which includes both Evidence Retrieval and Claim Classification. Model tuning for the competition.

Jiajin Yang(1338081) Major contributions on data preprocessing, include lemmatization, stop words removal, TF-IDF, and data reconstruction. Helped optimize the hyperparameters.

Chuyuan Xu(1398756) Major contribution on investigating the background of the project and analyze its aims. Find relevant literature review. Helped optimize the hyperparameters.

Kehan Ren (1161074) Helped optimize the hyperparameters. Focused on summarizing the project's limitations and identifying potential improvement.

References

Akiko Aizawa. 2003. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65.

Vimala Balakrishnan and Ethel Lloyd-Yemoh. 2020. Stemming and lemmatization: A comparison of retrieval performances. *Proceedings of SCEI Seoul Conferences*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28.

Steven Bird, Edward Loper, and Ewan Klein. 2024. Natural language toolkit. https://www.nltk.org/. Accessed: 2024-05-25.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yaohou Fan, Chetan Arora, and Christoph Treude. 2023. Stop words for processing software engineering documents: Do they matter? In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, pages 40–47. IEEE.

Siddhant Garg, Thuy Vu, and Alessandro Moschitti. 2020. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7780–7788.

Israa Jaradat, Pepa Gencheva, Alberto Barrón-Cedeño, Lluís Màrquez, and Preslav Nakov. 2018. Claimrank: Detecting check-worthy claims in arabic and english. *arXiv preprint arXiv:1804.07587*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in python/6.8. pairwise metrics, affinities and kernels. https://scikit-learn.org/stable/modules/metrics.html#cosine-similarity. Accessed: 2024-05-26.

Shaden Shaar, Firoj Alam, Giovanni Da San Martino, and Preslav Nakov. 2021. The role of context in detecting previously fact-checked claims. *arXiv preprint arXiv:2104.07423*.

James Thorne and Andreas Vlachos. 2018. Automated fact checking: Task formulations, methods and future directions. *arXiv preprint arXiv:1806.07687*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. 2021. Pretrained transformers for text ranking: Bert and beyond. In *Proceedings of the 14th ACM International Conference on web search and data mining*, pages 1154–1156.