COMP0084:Information Retrieval and Data Mining CourseWork

18101589
ucabyo0@ucl.ac.uk

ABSTRACT

KEYWORDS

Data mining, Information retrieval

1 INTRODUCTION

1.1 Aim

An increasing amount of online misinformation has motivated research in automated fact checking. Your task in this project is to develop information retrieval and data mining methods to assess the veracity of a claim. Specifically, the automated fact checking project consists of three steps: relevant document retrieval, evidence sentence selection, and claim veracity prediction.
In this coursework, we need to finish following tasks:

- 1. Completing the basic word frequency statistics and verify the zipf‘s law.
- 2. Using TF-IDF vector to represent the text, and compute the cosine similarity.
- 3. Building query-likelyhood model with Laplace Smoothing, Jelinek-Mercer Smoothing and Dirichlet Smoothing.
- 4. Using one of the implemented information retrieval methods, based on the claim, extract the five most similar documents about the claim, then extract the five most similar sentences in the document, and select one word embedding method to represent the claim and sentence and implement the Logistic Regression and analyse its performance.
- 5. Using the labelled data in the development subset to analyze the sentence retrieval performance of your model with recall, precision and F1 metrics.
- 6. Building a neural network based model to assess the truthfulness of a claim in the training subset.
- 7. Do a literature review regarding fact checking and misinformation detec- tion, identify pros and cons of existing models/methods and provide critical analysis.
- 8. Propose ways to improve the machine learning models you have implemented in task6, and analyse why.

1.2 Data Description

We will be using the publicly available Fact Extraction and Verification (FEVER) dataset [1]. This dataset consists of a collection of documents (wiki-pages), a labeled training subset (train.jsonl), a labeled development subset (dev.jsonl) and a reserved testing subset (test.jsonl).
The number of documents(wiki-pages) is 5395867 and divided into 109 files, and this dataset consist 185,445 claims which manually verified against Wikipedia pages and classified as Supported, Refuted and NotEnoughInfo. For the first two classes, there is a combination of sentences forming the necessary evidence supporting or refuting the claim. Labeled training subset (train.jsonl) include 145449 claims, labeled development subset (dev.jsonl) includes 19998 claims, and and a reserved testing subset (test.jsonl) includes 19998 claims.
In training subset, each line include four fields: "id","verifiable","label" ,"evidence". The value of the "evidence" field is a list of relevant sentences in the format of [_, _, wiki document ID, sentence ID].

[1] http://fever.ai/resources.html

2 PREPROCESSING

Before starting each task of this course, we need to preprocess the text. we got the tokens by tokenisation, remove stopword and punctuation, and word lemmatization.

2.1 Tokenisation

I split each sentence into tokens, I used nltk library [2] to implement it. for example, if I have a sentence like "I like an apple", after tokenisation, I will got ["I","like","an,"apple"].

2.2 Remove Stopword and Punctuation

In English, including a lot of words similar to a/an/the, these words have no practical meaning. We generally reduce the structural feature noise by removing punctuation and redundant stopword. At the same time, remove the stop word and punctuation could save storage and balance the word frequency distribution. I implement this using NLTK library[2]

2.3 Lemmatization

After the word tokenisation, we get the tokens. But in these tokens, we will find that "drive" and "driving" are not the same token. In general, we have two ways to solve this problem, 1. Stemming: extract the stem or root of the word form (not necessarily able to express complete semantics) 2. Lemmatization: restore a language vocabulary of any kind to a general form (can express complete semantics). The lemmatization is mainly applied to text mining and natural language processing for more fine-grained and more accurate text analysis and expression. [2]
So I use the lemmatization to preprocessing the text, I implement it using the NLTK library [2]

3 TASK

3.1 Task1: Text Statistics

3.1.1 Task Description. Count frequencies of every term in the collection of documents (5 marks), plot the curve of term frequencies and verify Zipf‘s Law (5) marks. Report the values of the parameters for Zipf‘s law for this corpus. (10 marks in total)

3.1.2 Solution. As we know, Zipf‘s law is an empirical law formulated using mathematical statistics. Zipf’s law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table. So word number n

[2] https://www.nltk.org/
[2] https://www.nltk.org/

has a frequency proportional to 1/n.
In this task, I traverse all documents (wikipages) and calculate the frequency of each word. For the convenience of query word and its frequency, I use python's dict data structure to save the frequency of each word. We can keep stopword when verifying Zipf's law, because stopword is often the highest word frequency. the Zipf's law can be write in

$$P(r) = \frac{C}{r^{\alpha}} \tag{1}$$

then I am looking for the logarithm of the equation, I got

$$\log P(\gamma) = \log C - \alpha * \log r \tag{2}$$

I used the Least squares method to calculate parameter, I use the sklearn library [3] to implement it here. and the parameter $\alpha = -1.28$ and $C = -0.76$

I use the dot figure to plot the Zipf's law, and the coordinates are the natural logarithm of the rank and frequency. the results show in Figure 1

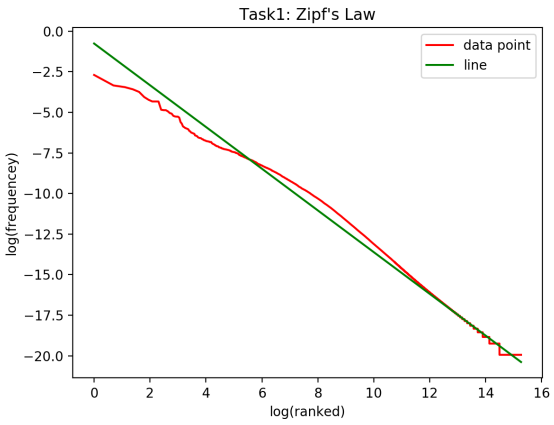

Figure 1: Task1

3.2 Task2: Vector Space Document retrieval

3.2.1 Task description. Extract TF-IDF representations of the 10 claims and all the documents respectively based on the document collection. The goal of this representation to later compute the cosine similarity between the document and the claims. Hence, for computational efficiency, you are allowed to represent the documents only based on the words that would have an effect on the cosine similarity computation (5 marks). Given a claim, compute its cosine similarity with each document and return the document ID (the "id" field in the wiki-page) of the five most similar documents for that claim (5 marks). (10 marks in total)

3.2.2 Solution. TF-IDF is short for Term Frequency-Inverse Document Frequency.
Term frequency $tf(t,d)$ is the number of times that term t occurs in document d. we use $f_{t,d}$ denote the the raw count term t in document. Term frequency calculation formula is

$$tf(t,d) = \frac{f_{t,d}}{(numberofwordsind)} \tag{3}$$

Inverse document frequency is a measure of how much information the word provides. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$idf(t,D) = log\frac{N}{|d \in D : t \in d| + 1} \tag{4}$$

with N total number of documents in the corpus $N = |D|$
$|\{d \in D : t \in d\}|$: number of documents where the term t appears (i.e., $\text{tf}(t,d) \neq 0$. If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to $|\{d \in D : t \in d\} + 1|$.
Thus, the TF-IDF's caluculation formula is

$$tfidf(t,d,D) = tf(t,d) * idf(t,D) \tag{5}$$

In this task, we got the tokens after tokenisation, then remove the stopword and punctuation, and do the lemmatization to got final tokens. the total number of Wikipages is N.
Then, I calculated the term frequency of each of the 10 claims corresponding to the claim. these 10 claims id are [75397, 150448, 214861, 156709, 129629, 33078, 6744, 226034, 40190, 76253]. donated $tf_{claimid}(t,d)$
Next, I traversed through each document in each Wikipage and calculate the term frequency $tf_{documentid}(t,d)$ for each word in each document, at same time, we calculate the $idf(t,D)$ of each term.
then we could calcuate the tf-idf of claim is

$$tfidf_{claimid} = tf_{claimid}(t,d) * idf(t,D) \tag{6}$$

Also we could calcuate the tf-idf of claim is

$$tfidf_{documentid} = tf_{documentid}(t,d) * idf(t,D) \tag{7}$$

I got two vector $tfidf_{documentid}$ and $tfidf_{claimid}$ after calcuated the tf-idf. Then I consider calculating the cosine similarity of the two vectors. the formual of cosine similarity is

$$cosine\_similarity = cos(\theta) = \frac{A \cdot B}{||A||||B||} \tag{8}$$

A, B are two vectors.
I traversed the tf-idf of all the documents for each of the 10 claims, and then calculate the corresponding cosine similarity, and then get the five documents with the highest similarity. the result shown in attachment csv.file.
and I show the tf-idf vector of 10 claim in Table 1

3.3 Task3: Probabilistic Document Retrieval

3.3.1 Task description. Establish a query-likelihood unigram language model based on the document collection, and return the five most similar documents for each one of the 10 claims (5 marks). Implement and apply Laplace Smoothing, Jelinek-Mercer Smoothing and Dirichlet Smoothing to the query-likelihood language model, return the five most similar documents for the 10 claims (5 marks). (10 marks in total)

[3] https://scikit-learn.org/stable/

claim id	tf-idf vector
40190	[0.0042, 0.0051, 7.4e-06, 2.8e-06, 1.2e-05, 2.2e-05, 0.0004, 0.00014, 2.16e-05]
129629	[0.00029, 2.451e-06, 1.762e-05, 0.00027, 0.000141, 5.856e-06, 9.510e-05, 6.150e-06, 0.000125, 6.262e-06]
75397	[0.0119, 0.0717, 2.618e-05, 0.000135, 9.314e-05, 1.218e-05]
76253	[8.074e-05, 2.0123e-05, 0.00193, 1.855e-05]
226034	[0.0180, 5.0261e-05, 4.034e-05, 0.000117, 0.000106]
156709	[0.00796155308069784, 0.16147024841790308, 0.0018225918692673364]
6744	[6.880834449626552e-05, 0.005520350373261644, 0.00046763436249905565, 1.3490611095355653e-05]
150448	[6.018e-05, 0.0075, 0.000161, 0.000336]
33078	[7.56e-05, 0.000318, 1.0652e-05, 1.365e-05, 1.0605e-05, 0.000966, 7.1982e-05]
214861	[6.89e-06, 1.44e-05, 1.01e-05, 3.38e-05, 3.28e-05, 7.67e-05, 4.96e-06, 3.50e-05, 5.19e-05, 2.8e-05, 2.277e-05, 1.2e-04, 3.37e-06, 9.94e-05, 6.43e-05]

Table 1: TF-IDF vector of 10 claims

3.3.2 Solution. The language model is the model used to calculate the probability of a sentence.
At first, we build the unigram language model. we define w_i as the word in a sentence, because I want to build the unigram language model. the probability of a sentence is

$$P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i) \tag{9}$$

Then I consider the querylikelihood query model. The query Q contains $q_1, q_2, q_3, ..., qk$ and the document using the language model M_D Rank documents by the probability that the query could be generated by the document model by the formal:

$$P(Q|M_D) = P(q_1...q_k|M_D) = \prod_{i=1}^{k} P(q_i|M_D) \tag{10}$$

In this task, we need to implement three smoothing method to the query-likelihood language model.

- Laplace Smoothing: Laplace Smoothing also known add one smoothing, If number of occurrences of words in document D are $(m_1, m_2, ..., m_{|V|})$ with $\sum m_i = N$, where $|V|$ is the number of unique words in the entire collection (vocabulary size), the probability after a laplace smoothing.

$$p_{Laplace}(m_1) = \frac{m_1 + 1}{|D| + |V|} \tag{11}$$

- Jelinek-Mercer Smoothing: introduced a λ which independent of document, query. the probability after Jelinek-Mercer Smoothing is

$$p\,(q_i|D) = (1 - \lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|V|} \tag{12}$$

  where $f_{q_i,D}$ is the frequency of q_i in document D, and the c_{q_i} is the number of q_i. In this task we set the $\lambda = 0.5$
- Dirichlet Smoothing: Dirichlet Smoothing make smoothing depend on sample size the probablity after Dirichlet Smoothing is

$$p\,(q_i|D) = \frac{f_{q_i,D} + \mu\frac{c_{qi}}{|V|}}{|D| + \mu} \tag{13}$$

  and μ is a constant.

I use the word frequency file saved before and the word frequency file of the entire corpus(109 wikipages files). We use the claim as the query and each document as the D. We calculate the probability of 10 claims corresponding to each document correspond to different smoothing methods and obtain five documents which take top five probability with each smoothhing methods. the results shown in attachment file.

3.4 Task4: Sentence Relevance

3.4.1 Task description. For a claim in the training subset and the retrieved five documents for this claim (either based on cosine similarity or the query likelihood model), represent the claim and sentences in these documents based on a word embedding method, (such as Word2Vec, GloVe, FastText or ELMo) (5 marks). With these embeddings as input, implement a logistic regression model trained on the training subset (5 marks). Use this model to identify five relevant documents to the claims in the development data and select the five most relevant sentences for a given claim within these documents. Report the performance of your system in this dataset using an evaluation metric you think would fit to this task. Analyze the effect of the learning rate on the model training loss (5 marks). Instead of using Python sklearn or other packages, the implementations of the logistic regression algorithm should be your own. (15 marks in total)

3.4.2 Solution 1: **Word embedding method**
In this task, I use the code from Task3 (Dirichlet Smoothing). At first, we got the top five document correspond to each claim from the overall trainset "train.jsonl", then from these five document, I could found 5 most relevant sentence also use Task3 (Dirichlet Smoothing).
Then I select GloVe to represent the claim and sentence. GloVe's full name is Global Vectors for Word Representation, a word representation tool based on count-based overall statistics. The GloVe works like Word2Vec. The Word2Vec is a "predictive" model that predicts the context of a given word. Glove learns by constructing a co-occurrence matrix (the word * context). The co-occurrence matrix is mainly to calculate the frequency of occurrence of a word in the context. The co-occurrence matrix is a huge matrix, and we get a lower-dimensional representation by decomposition.
I downloaded the pre-trained word vectors from GloVe [4]. In order to reduce the computation resource and time, I select 50 dimension word vectors.

[4] https://nlp.stanford.edu/projects/glove/

3.4.3 Solution 2: **Logistic Regression**

Then I implemented a logistic regression. As we know the logistic function is

$$g(z) = \frac{1}{1+e^{-z}} \tag{14}$$

Then I construct the prediction function

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1+e^{-\theta^T x}} \tag{15}$$

The value of prediction function $h_\theta(x)$ has a special meaning, which indicates the probability that the result takes 1, so the probability of class 1 and category 0 for the input x classification result is:

$$\begin{aligned} P(y=1|x;\theta) &= h_\theta(x) \\ P(y=0|x;\theta) &= 1 - h_\theta(x) \end{aligned} \tag{16}$$

Then I got the cost function.

$$\text{cost}\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta(x)\right) & \text{, if } y = 0 \end{cases} \tag{17}$$

and I use the maximum likelihood estimation to find the J_θ function

$$P(y|x;\theta) = (h_\theta(x))^y \left(1 - h_\theta(x)\right)^{1-y} \tag{18}$$

Likelihood function is:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} P\left(y^{(i)}|x^{(i)};\theta\right) \\ &= \prod_{i=1}^{m} (h_\theta(x^(i)))^{y^(i)} (1 - (h_\theta(x^(i)))^{y^(i)}) \end{aligned} \tag{19}$$

then the (J_θ) function is:

$$\begin{aligned} J(\theta) &= \log L(\theta) \\ &= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \end{aligned} \tag{20}$$

The likelihood function finds the minimum value of $J(\theta)$ using the gradient descent method. According to the gradient descent method, the update process of θ can be obtained:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \quad (j = 0 \ldots n) \tag{21}$$

the step of learning is α, then

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(\mathrm{x}^{(\mathrm{i})}\right)\right) x_j^{(\mathrm{i})} \\ &= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(\mathrm{x}^{(\mathrm{i})}\right) - y^{(i)}\right) x_j^{(\mathrm{i})} \end{aligned} \tag{22}$$

So the θ updated with

$$\begin{aligned} \theta_j &:= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(\mathrm{x}^{(\mathrm{i})}\right) - y^{(i)}\right) x_j^{(\mathrm{i})}, \quad (j = 0 \ldots n) \\ &= \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta\left(\mathrm{x}^{(\mathrm{i})}\right) - y^{(i)}\right) x_j^{(\mathrm{i})}, \quad (j = 0 \ldots n) \end{aligned} \tag{23}$$

The method of vector and matrix can be expressed as

$$\theta := \theta - \alpha \cdot \left(\frac{1}{m}\right) \cdot x^T \cdot (g(x \cdot \theta) - y) \tag{24}$$

Thus, we could use above method to implement the Logistic regression. Because this problem is a classification problem, in this task, I use accuracy as metric to measure the model performance. The confusion matrix in Table 2 and the accuracy can be calculated

Predict condition	True condition	
	Positive	Negative
Predicted positive	TP	FP
Predicted negative	FN	TN

Table 2: Confusion Matrix

$$accuracy = \frac{(TP + TN)}{(TP + FP + FN + TN)} \tag{25}$$

In this task, we chose to use a learning rate of 1 for Logistic regression. I use training set (train.jsonl) as trainingset, then I used 10 claims which use in task23 to verify, because the number of 10 claims is too small, we also randomly took a certain number of positive and negative examples from validationset (share_task_dev.jsonl) to verify. In our model, **our validation set's accuracy is 0.80** . In addition, I can plot a loss function graph with a learning rate of **0.1, 1, 10**. The results show in following

Figure 2: Loss function with different learning rate in Logistic Regression

From Figure 2, it can be concluded that the choice of learning rate is very important. If the learning rate is too large, the loss will jump at the beginning, although it will converge at the end, but it should be converged to the local minimum. If the learning rate is too small, the loss convergence slow, consuming time and resources.

3.5 Task5: Relevance Evaluation

3.5.1 Task description. Implement methods to compute recall, precision and F1 metrics (5 marks). Analyze the sentence retrieval performance of your model using the labelled data in the development subset (5 marks). (10 marks in total)

3.5.2 Solution. To calculate Recall Precision F1-score, confusion matrix shown in Table 2, we could compute the Precision, Recall, F1-Score and Macro-F1 by following formula.

$$Precision = \frac{TP}{TP + FP} \tag{26}$$

$$Recall = \frac{TP}{TP + FN} \tag{27}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{28}$$

In this task, we use the model saved from Task4, and processeing the development dataset (share_task_dev.jsonl) and extract related sentence. Then we import the data into the model and we can get the result in Table 3

Accuracy	Precision	Recall	F1-score
0.737	0.738	0.692	0.714

Table 3: Task5: Results

3.6 Task6: Truthfulness of Claims

3.6.1 Task description: Using the relevant sentences in Subtask 4 as your training data and using their corresponding truthfulness labels in the train.jsonl file, build a neural network based model to assess the truthfulness of a claim in the training subset. You may use existing packages like Tensorflow or PyTorch in this subtask. You are expected to propose your own network architecture for the neural network. Report the performance of your system in the labelled development subset using evaluation metrics you have implemented. Furthermore, describe the motivation behind your proposed neural architecture. The marks you get will be based on the quality and novelty of your proposed neural network architecture, as well as its performance. (15 marks in total)

3.6.2 Solution: In this task, I used all training claims (train.jsonl) and their related sentence as input features. If the label is support, it is set as a positive example. If the label is not support, it is a negative example. I did the same for the validation set (share_task_dev.jsonl). We separate the claim and the sentence in the dataset. We first connect the different sentences, pass the claim and the sentence through the word embeding, and the results are concatenated to get an entire input feature. The input feature contains two components. One is claim and the other is sentence.

As we known, Neural network, such as RNN, CNN, performed well on information retrival task. RNN is used to process sequence data, so we choose the RNN to be our basic model. Due to the gradient vanishing problem, some memories may disappear. LSTM is introduced in the model. In particular, we choose Gradient Re- current Unit which is a new variant of RNN and take the same function as LSTM. and [3] shows GRU is more computational efficiency because of its only two-gate mechanism (update gate and reset gate) rather than 3 in LSTM.

Since the length of the sentence feature is too long, certain information is forgotten due to the characteristics of the LSTM. So we use the Bi-LSTM for sentence feature. In papar [13], Bi-LSTM was used for information retrival tasks and effective results were obtained.

I designed my network. As shown in the following figure, I first input the claim feature into a GRU, at the same time, I input the sentence feature into a two-layer bi-LSTM. Because the claim feature is relatively short, only GRU should be sufficient. We set the maximum length of the sentence feature to 500 and the maximum length of the claim feature to 50.

When training the model, we can use the learning rate of dynamic attenuation to avoid the model staying at the local optimum. We first iterate the model multiple times with the default learning rate (0.001), retain the model with the highest verification accuracy, and then load the previous model, the learning rate drops to 0.0001, continue to train the model, and retain the model with the highest verification accuracy; Load the previous step optimal model, remove the regularization strategy (dropout, etc.), adjust the learning rate to 0.00001, and train to the optimal. I randomly divide the training set into a training set and a verification set, and perform multiple iterations (15 times) on the training set. Each iteration retains a model, and each iteration calculates the precision, recall, f1-score of the verification set. I get the optimal model through the largest f1-score, and then load the test set into the optimal model to get the accuracy of the test set.

3.6.3 Results. The network graph show in Figure 3.

The Accuracy in validation set(share_task_dev.jsonl) is 0.717

3.7 Task7

3.7.1 Task description: Literature Review. Do a literature review regarding fact checking and misinformation detection, identify pros and cons of existing models/methods and provide critical analysis. Explain what you think the drawback of each of these models are (if any). (10 marks in total)

3.7.2 Literature Review 1: **Fact checking and misinformation detection**

We review many literature about misinformation detection and fact checking. [4] proposed a Context-Aware Approach to do misinformation checking, and [16] proposed a hybrid approach that combines simple heuristics with supervised machine learning to identify claims made in political debates and speeches, and provide a mechanism to rank them in terms of their 'check-worthiness'. and [5] design a system for misinformation checking with Neural Check-Worthiness Ranking.

3.7.3 Literature Review 2: **Word Embedding**

Then I consider the different method of word embedding. At first, [8] propose a word embedding method called word2vec, which pays attention to local corpus. Although word2vec is also a language model, it focuses on the word vector itself, so many optimizations have been made to improve computational efficiency. [6] propose Fasttext which use word2vec's skip-gram model, the main difference is reflected in the features. Word2vec uses word as the most basic unit, that is, predicting other words in its context through the central word. The subword model uses the letter n-gram as the unit, and the value of n is 3 6. Thus each vocabulary can be represented as a string of letters n-gram, and the embedding of a word is the sum of all its n-grams. In this way, our training also changes from the embedding prediction target word with the central word to the n-gram embedding prediction target word with the central word.

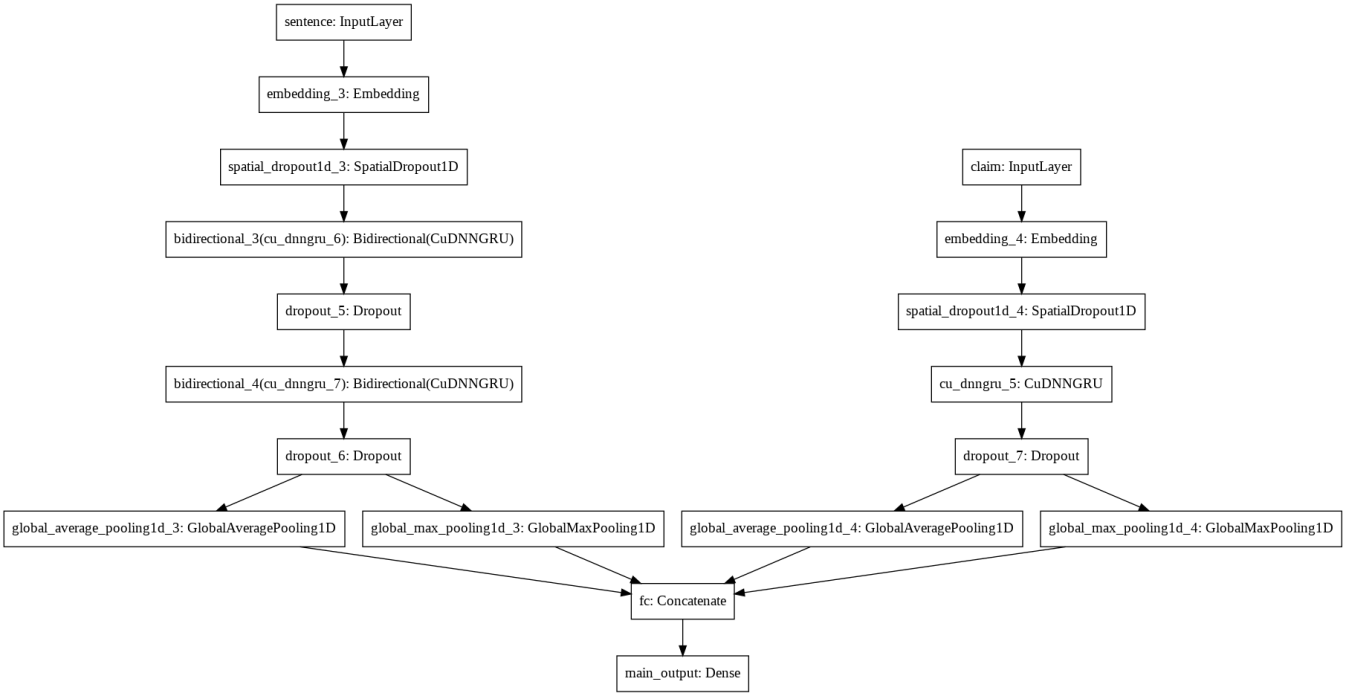

Figure 3: Task6 Network structure

[9] proposed GloVe, and Word2vec only considers the local information of the word, does not take into account the relationship between the word and the word outside the window. GloVe uses the co-occurrence matrix, taking into account the local information and the overall information. There is a problem with word2vec and glow. Words have different meanings in different contexts, and the vector representations of these two model words in different contexts are the same.[10] proposed the ELMo, which learn the complex usage of words through the multi-layer stack LSTM. [12] proposed transformer and proposed a word embedding method called BERT. Bert uses Transformer for feature extraction. The Transformer in bert can be multi-layered and has parallel computing power. The two-way language model of bert uses the encoder part and uses complete sentences.

3.7.4 Literature Review 3: **Model**

As we know, LSTM is a effective model to do the sequence classification problem. [14] show a series of experiments to compare the differences of CNN, LSTM and GRU. [15] utilizes CNN to extract a sequence of higher-level phrase representations, and are fed into a LSTM to obtain the sentence representation. [11] gives the result that applying the two-layer GRU into classifying sentiment of Russian tweets has out-performance among CNN, LSTM, GRU and Bi-GRU.

Besides, In [1], The first job in the NLP to use the attention mechanism. They used the attention mechanism on Neural Network Machine Translation (NMT), which show the attention mechanism is very effective in NLP task. [7] tell us how to use the attention mechanism in RNN and get excellent results.

3.7.5 Conclusion. We can modify the word embeding way to get more word information. Or we can modify the network and add the attention mechanism to the LSTM to make the information more complete.

3.8 Task8

3.8.1 Task description: Propose ways to improve the machine learning models you have implemented. You can either propose new machine learning models, new ways of sampling/using the training data, or propose new neural architectures. You are allowed to use existing libraries/packages for this part. Explain how your proposed method(s) differ from existing work in the literature. The marks you get will be based on the quality and novelty of your proposed methods, as well as the performance of your final model. (20 marks in total)

3.8.2 Solution: After literature review, due to time constraints, I was unable to improve on the Word Embedding method. I am making changes directly on the network structure. We consider that there is a forgotten gate, whether it is Bi-LSTM or GRU. Then the information in the sequence will always be forgotten, then we can effectively solve this problem by adding a attention mechanism. We add the attention mechanism when processing both claims and sentence. This will allow the information to be further extracted. we call this as the claim-sentence Dual attention mechanism.

In the traditional approach, the attention mechanism acts directly on the global input features. In my network, I apply the attention to the claim feature and the sentence feature separately, which is equivalent to further extracting important information from the claim and sentence features. Because the important information

after the redundancy is removed is the most important. Network structure as shown Figure 4. In this task, I use the trainset and validation set as same as task6. **The accuracy of validation set is 0.734** which is higher than the model in Task6.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Vimala Balakrishnan and Ethel Lloyd-Yemoh. 2014. Stemming and lemmatization: a comparison of retrieval performances. (2014).

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[4] Pepa Gencheva, Preslav Nakov, Lluís Màrquez i Villodre, Alberto Barrón-Cedeño, and Ivan Koychev. 2017. A Context-Aware Approach for Detecting Worth-Checking Claims in Political Debates. In *RANLP*.

[5] Casper Hansen, Christian Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. 2019. Neural Check-Worthiness Ranking with Weak Supervision: Finding Sentences for Fact-Checking. *arXiv preprint arXiv:1903.08404* (2019).

[6] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

[7] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[10] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* (2018).

[11] J Trofimovich. 2016. Comparison of neural network architectures for sentiment analysis of russian tweets. In *Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference Dialogue 2016, RGGU*.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[13] Shuohang Wang and Jing Jiang. 2015. Learning natural language inference with LSTM. *arXiv preprint arXiv:1512.08849* (2015).

[14] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923* (2017).

[15] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. A C-LSTM neural network for text classification. *arXiv preprint arXiv:1511.08630* (2015).

[16] Chaoyuan Zuo, Ayla Karakas, and Ritwik Banerjee. 2018. A Hybrid Recognition System for Check-worthy Claims Using Heuristics and Supervised Learning. In *CLEF*.

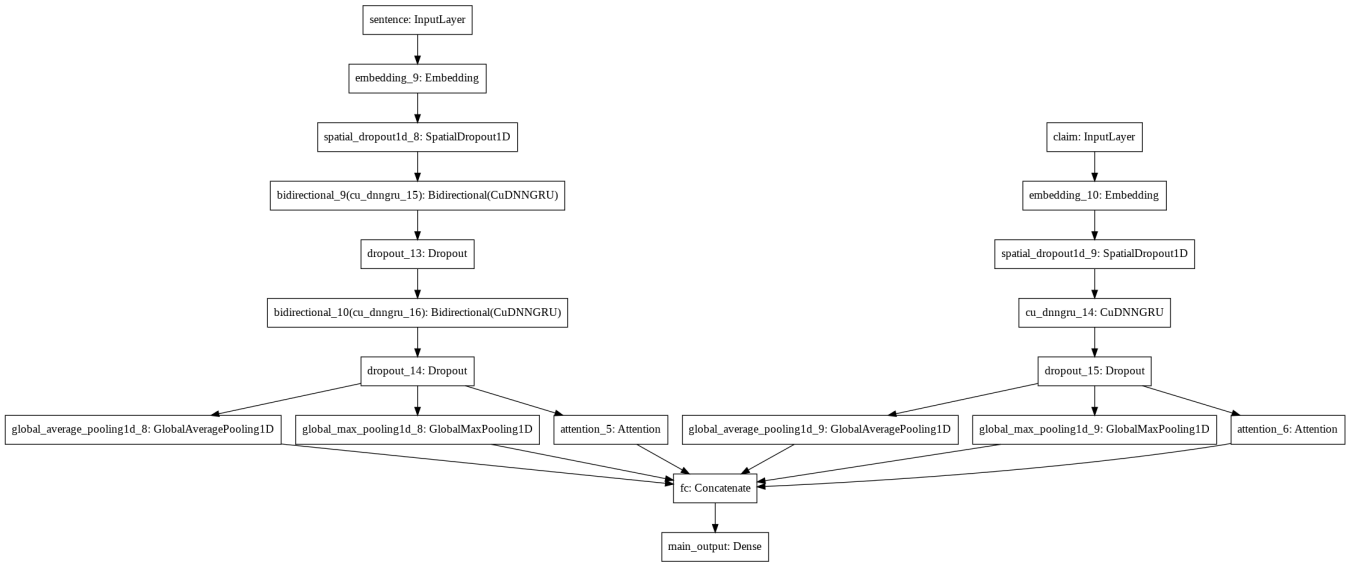

Figure 4: Task8: Network structure