NLP Final Group 5

Question 1

> **Briefly explain your choice to do evidence sentence extraction and compare the**
> **results of choosing different parameters(if has) in your method.(15pt.)**

RNN

In our method, there are 3 main steps to extract the evidence sentence for RNN model training.

Step 1

Remove sentences in articles that contains “Link:, *, JavaScript, Twitter, ___, [IMG], \” because sentences with these words usually don’t contain useful information.

Step 2

Discard sentences in articles that contains less than 10 words in order to focus on sentences with more information. You can set the parameter of the length of the sentence. In our experience, 10 is a better choice cause we don’t want to discard too much information, while choosing smaller parameter will lead to longer data processing time.

Step 3

Count the occurrences of vocabularies and put the top 2000 vocabs with highest counts into a set. We only keep vocab that occurrences more than 3 times, so the set might be smaller than 2000. We’ve tried to keep vocab with different occurences such as 2 and 4. However, choosing 2 will keep a lot of redundant words that doesn’t help training; choosing 4 discards too many word and thus the set couldn’t provide enough information. This is a bag-of-word like process. However, instead of creating vectors, we just simply convert the set into a string by concating the vocabs in the set with a seperate space. Finally, insert the claimant and claim in front of the string.

After completing these steps, we write the evidence string and the rating into a csv file for model training, and this is the end of our evidence sentence extraction of RNN model.

BERT

Our goal in evidence sentence extraction is to transform the data into following format:

```
[
  {
    article : concatenate related evidence sentences into single string
    question : given claim
    options : [true, partial, false]
    answer : chosen options index
  }
]
```

The main idea of this format is the treat each fact checking problem as a multiple choice problem.

(Similar to HW3)

How we concatenate the evidence sentences ?

In each given article file, it contains a list of sentences. According to the given `train.json` (or `valid.json`, `test.json`), we can find article related to the claim. For each related article, we will go through all the sentences and decide whether the sentence is related to the claim or not.

The sentence is said to be *related to the claim* if any of the claim's vocabulary is contained in the sentence.

```
def is_related_to_claim(self, claim):
  claim_list = claim.split()
  text_list = self.text.split()

  for c in claim_list:
    for t in text_list:
      if c == t:
        return True
  return False
```

If the sentence is related to the claim, we will complete the following preprocessing. Otherwise, the sentences will be ignored.

- Convert to lower case (Although BERT will do this, we need lower case for later comparison)
- Remove customize weird character
- Remove url string
- Remove extra space
- Remove stopwords (according to `nltk` stopword library)

After collecting all related sentences, we will concatenate them into a single string.

LSTM

For LSTM preprocessing, our goal is to transfer the data into this format :

context	label	rating
merge evidence and claim	[true, partial, false]	given rating

The main idea of this format is to treat this problem as fake news checking problem, which consider evidence and claim as some part of news.

For each evidence article, we want to select the most important one base on the following formula:

1. For each evidence, run through all the sentence. If the sentence contain the vocabulary in claim, add `1` to its importance score.
2. Choose the evidence with the highest score.

After choosing the

Question 2

> **Describe how you implement your sequence model, including your choice of**
> **packages, model architectures, loss functions, hyperparameters (learning rate,**
> **epochs, etc.)etc.(15pt.)**

RNN

For RNN model, we use torchtext and pytorch module to implement it. We selected bidirectional GRU as our model architecture, and used Glove provided in torchtext.vocab for word embedding. We chose SGD as our optimizer, and trained 25 epochs with learning rate 1e-1, and the loss function is CrossEntropyLoss.

BERT

For BERT model fine-tuning, we use the Hugging-Face's transformers library. The structure of the model was similar to that used in a previous assignment (HW3). To prepare the data for BERT, we concatenated the preprocessed articles with the claim and each possible option (”False”, “Possibly True”, “True”) and truncated it to fit the model's maximum input length. The model was trained using a learning rate of 5e-5, Adam as the optimizer, and for 5 epochs.

Question 3

> **All of method you have tried, and compare with your best method.(10pt.)**

RNN

At first, we just filter the article by the method provide in “Question1, RNN” step 1 and 2, then simply use the first 2000 words in the articles as the evidence sentence. This way generates evidence sentence fast but inaccurate, since many sentences are discarded directly. As a result, the model as a poor performance with only 45% f1 score in the test set.

After using the bag-of-word like process in “Question1, RNN” step 3, the f1 score performance improves to 50.8%.

We also tried an experiment just for fun and found some interesting result. We trained the model without giving claimant and claim, only provide the evidence sentence extracted by us, and it still got 40% f1 score in the test set. The model seldom guess “2: true” if we don’t provide claimant and claim.

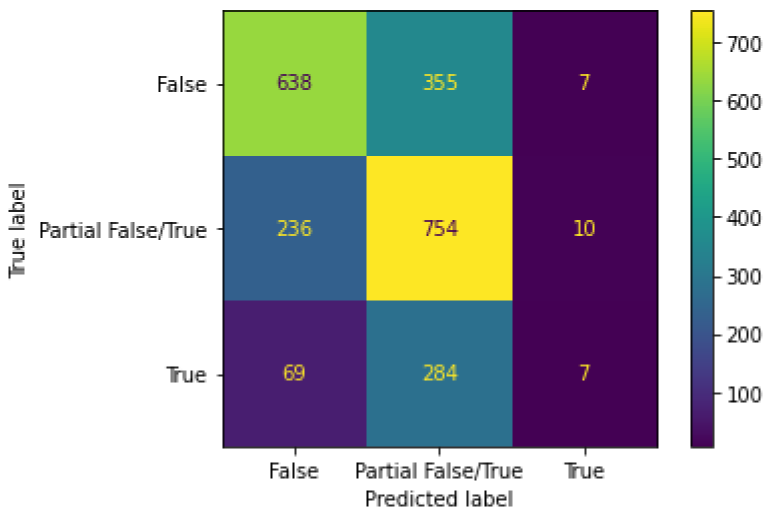
BERT

In our evaluation of BERT, we were disappointed to find that it failed to beat our own RNN model. One potential reason for this outcome is the input sequence maximum length of 512 in BERT, which may not be sufficient to fully capture longer articles. Additionally, while we preprocessed the data using our own method, there is still room for improvement in this area. With further optimization of data preprocessing to feed the model with more relevant information, BERT may have the potential to perform better.

Question 4

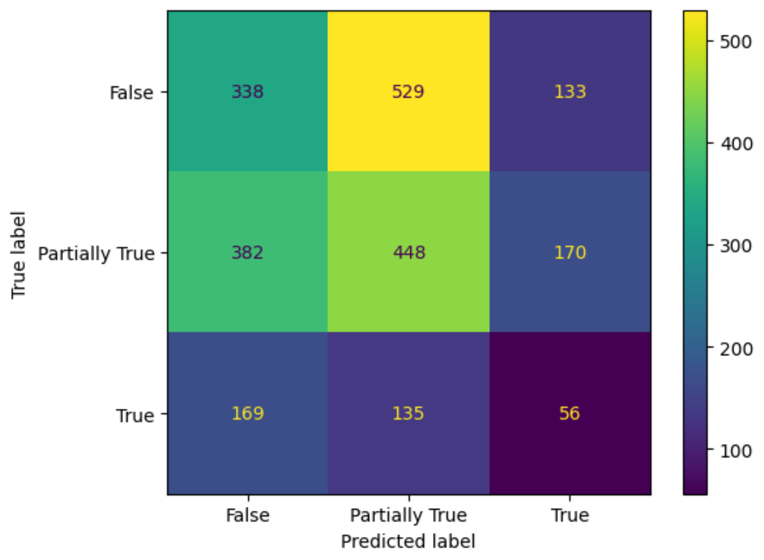
> **Do error analysis. You can use confusion matrix to illustrate the whole model**
> **performance, or try to analyze the dataset’s problem like unbalanced number of**
> **labels could lead what difficulty on training and how to overcome it. Share anything**
> **you observe. (10pt.)**

RNN



Our RNN model has a hard time classifying True and Partial True claim. In order to solve the problem, we decided to pre-train the model by dataset with only True and False claim, and then fine-tune with the whole dataset in the future. Also, we will use data augmentation in order to balance the labels in the dataset.

BERT



How to run our code

RNN

1. Open `Load_data.ipynb` and press run all to generate csv files for training.
2. Open `RNN_Training.ipynb` and press run all to train the model and evaluate the result.

BERT

1. After downloading the zip file `Final_5.zip`, unzip `Final_5.zip`.
2. `cd` into `Final_5` folder.
3. Run the script below to preprocess, train, and inference.

```
#!/bin/sh

pwd # should be in Final_5
BERT_ROOT="BERT"

mkdir models
mkdir polished

# preprocess data
python3 "$BERT_ROOT"/preprocess/main.py;
# move files
mv "$BERT_ROOT"/preprocess/new_train.json "$BERT_ROOT"/polished/train.json
mv "$BERT_ROOT"/preprocess/new_test.json "$BERT_ROOT"/polished/test.json
mv "$BERT_ROOT"/preprocess/new_valid.json "$BERT_ROOT"/polished/valid.json

# train
python3 "$BERT_ROOT"/train.py
```

Location of Dataset

RNN

Put the train.json, valid.json, test.json, and the articles folder in the same folder of the ipynb files.

```
Load_data.ipynb
RNN_Training.ipynb
train.json
```

```
valid.json
test.json
articles/
```

BERT

Put the dataset in the BERT folder

```
- BERT
|- preprocess/
|- train.py
|- articles/
|- train.json
|- test.json
|- valid.json
```