Fact-checking

Natural Language Processing assignment 2

Amir Hajiabadi Molin Zhang Jialing Li

Academic year: 2021/2022

Introduction

Fake information and news are widely circulated on many websites. The proliferation of fake news in recent years has triggered a number of reactions, most notably the emergence of several manual fact-checking initiatives. Since manual fact-checking is very time-consuming and fully automated fact-checking has credibility issues. Our goal is to avoid wasting time on fact-checked statements and to retrieve the correct information faster.

Downloading dataset

The dataset is about facts taken from Wikipedia documents that have to be verified. In particular, facts could face manual modifications in order to define fake information or to give different formulations of the same concept.

Dataset analysis

The dataset consists of 185,445 claims manually verified against the introductory sections of Wikipedia pages and classified as Supported, Refuted, or Insufficient evidence. We only need the claim to be verified, a set of semantically related statements (evidence set), and fact-checking labels: evidence to support or refute the claim. So we kept the Claim, Evidence, and Label columns and removed other unneeded information.

	Claims	Supports	Refutes
Training dataset	101632	89389	32351
Validation dataset	6458	3611	3554
Testing dataset	6542	3606	3583

Table 1. Distribution of claims, supports and refutes in training, validation and testing dataset

Data preprocessing

We cleaned up the text by removing tags, pronunciations, leading numbers, unnecessary and repetitive symbols. We have also standardized all letters to lowercase and removed stopwords.

Text tokenization

Tokenizers divide strings into lists of substrings. NLTK provides a simpler, regular-expression based tokenizer, which splits text on whitespace and punctuation. We also operated at the level of sentences, using the sentence tokenizer directly. NLTK tokenizers can produce token-spans, represented as tuples of integers having the same semantics as string slices, to support the efficient comparison of tokenizers.

Building and training network

Building network

We primarily defined three models and four connection mechanisms between the “evidence” and “claim” in this experiment. The embedding layer, which is not trainable and simultaneously shares weights, is the first to be applied to the preprocessed claim input and evidence input. Then the 2 features go to the Bidirectional RNN network, noting

that it also shares weights in this period. Next, we defined four distinct concatenations, including “add”, “average”, “concatenate”, and “maximum”, after features are produced by this network. Next, we concatenated a cosine similarity between features from “evidence channel”and “claim channel”. The output is finally obtained via two Dense layers. Here, nodes of the two dense layers are 27 and 13, respectively, and the hidden nodes of the RNN layer are set to 50. The final network parameters are shown in the table below:

	Trainable params	Non-trainable params	Total params
LSTM Network	43,912	1,698,080	1,741,992
RNN_Network	13,612	1,698,080	1,711,692
MLP_Network	601,621	1,697,880	2,299,501

Table 1. Parameters of each network (features merged by “add”)

Training network

We present the combination of two different networks, RNN and LSTM, and two different feature merging methods ('concatenate' and 'maximum'), and the training curves of a total of four models are shown in Figure 1 (the mlp model is not discussed here due to its poor performance). Epochs for training were 15 and batch_size was set to 256. The optimizer here we used was Adam and the learning rate was 0.01.

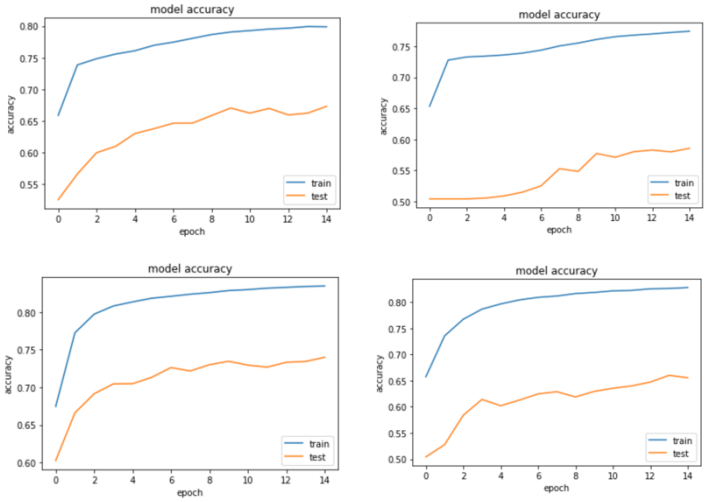

Figure 1. Training curves of 4 models: RNN with ‘concatenate’ (top-left), RNN with ‘maximum’ (top-right), LSTM with ‘concatenate’ (bottom-left), LSTM with ‘maximum’ (bottom-right).

Results and discussion

In the final test set, SampleRNN model with ‘concatenate’, RNN with ‘maximum’, LSTM with ‘concatenate’, and LSTM with ‘maximum’ obtained f1 scores of 0.63, 0.5, 0.71, and 0.59, respectively. LSTM with ‘concatenate’ got the best f1 score all in training (0.83), validation(0.74) and test set (0.71). Finally, there are a few points worth noting. First, in the training curve, we found that the result on the training set is always much higher than that on the validation set. This is related to the splitting dataset, which means that the distributions of the training and validation sets are not the same. However, because it has already been separated, we may use data augmentation to enhance the model's performance. Second, in the test set, for the best model, the precision of category one is 0.84, the recall is 0.54, and the f1 is 0.66; the precision of category two is 0.66, and the recall is 0.90, and the f1 is 0.76. The precision of the model is always higher than the recall, which is also related to the slight imbalance of the data. The number of supports in the training set is more than twice that of the refutes, indicating that we may improve the performance by balancing the data set.