Coursework 1: Question classification Report

Yan Liu 10443939
Yifan Yang 10581851
Panjian Huang 10436976
Jiasen Tian 10456489

Abstract

This paper reports the details of Text Mining Coursework1: Question classification. It shows how to pre-process data, how to split the dataset into training and development subset, then how to build three models: Bag-of-words (Bow) model, BiLSTM model, and BiLSTM ensemble model. For each model above, you can choose two kinds of word embeddings: randomly initialize word embeddings and Glove pre-trained word embeddings. It also shows how to train and evaluate the model by providing accuracy and confusion matrix.

1 Introduction

Question classification is a task in Natural Language Processing that requires the system to provide a concise class of answer to Natural Language questions [4]. The goal or problem of this coursework is to categorize questions into different semantic classes that impose constraints on potential answers so that they can be utilized in later stages of the question answering process. For example, when considering the question: What was the first domesticated bird, hope the system classifies this question as having answer type animal.

We develop three models with classifier. Through experiments, we found that the Bow model can not achieve high accuracy, and its performance is around 70%. While the performance of the Bilstm model or Bilstm ensemble model can reach about 80%.

2 Approaches

2.1 Sentences to Vectors

This coursework has two kinds of word embeddings: randomly initializing word embeddings and pre-trained word embeddings.

Randomly initializing word embeddings: Firstly, building a vocabulary (word to index dictionary), in which each lowercase unique word from the preprocessed dataset is assigned an ID. Then add word **#UKN#** into vocabulary list to stand for words that are not in the vocabulary. Next,using $embedding = nn.Embedding()$ in PyTorch to randomly initialize word vectors. Therefore, given a sentence which has removed stop words, we found all the words with their indexes in the sentence and the vectors that they correspond to. We add up the vectors for all the words and divide by the number of words in the sentence. Finally, the given sentence can be represented as vectors.

Pre-trained word embeddings: Firstly, pruning Glove pre-trained embeddings, which means removing the words that do not appear in the dataset. Following that, we built a new vocabulary based on Glove, allowed us to gather the weights of each word in GloVe and use the function $torch.nn.Embedding.from\_pretrained()$ to obtain vector that the word corresponds to. Finally, the given sentence can be represented as vectors.

2.2 Build Models

There are three models used in this system: Bag-of-Words model, BiLSTM model, BiLSTM ensemble model. All of them are able to support both Randomly word embeddings and Glove pre-trained word embeddings.

The Bow classifier [3] comprises a Bow sentence transferring operation and simply feed-forward neural network. For the Bow sentence transferring operation, the tensors of words that appear in the object sentence will be calculated. The mean of word vectors will be regarded as the sentence vector, then the sentence vector and its label will be sent into the feed-forward neural network as a training sample. The loss function is $nn.NLLLoss()$ which could calculate the negative log-likelihood loss. Obtaining log-probabilities in a neural net-

work is easily achieved by adding a $LogSoftmax$ layer in the last layer of FNN.

Regarding the Bi-LSTM model [2], it takes word embeddings as inputs, then outputs hidden states with hidden dimensionality. The linear layer maps from hidden state space to tag space. Next, the model inputs the sentence which contains preprocessing words to Bi-LSTM and outputs a sentence vector which is a simple stack of the results of the forward RNN and the reverse RNN. Finally, the $LogSoftmax$ layer outputs predicted values of all the words in a sentence and their mean as the predicted probabilities of all labels.

The third model is an ensemble model consisting of five sub-models which are all Bi-LSTM. Each Bi-LSTM is trained by different training sets generated by bagging over the original training set. In addition, the final predicted label for each test question is decided by the vote result of five sub-models.

3 Experiments

3.1 Experiment Set-up

The used data: The used data contains the TREC dataset [1] called train5500.txt, test.txt and glove.small.txt. The dataset file contains 5452 questions as instances and answer types as their labels. We then split the dataset file into 10 portions, where 9 portions are training set (4906 sentences) and 1 portion is development set (546 sentences). The development set is used to do early stopping, which means if the test error is found to increase on the development set for 2 consecutive epochs, the training process will be stopped and the parameters of the network are taken as final ones. The test.txt is used as a testing dataset to evaluate the performance of the models. The glove.small.txt contains the pre-train weights of all words which are produced by Glove.

Pre-processing steps: There are several preprocessing steps in the experiment. Firstly, all the questions with labels are loaded from the given TREC dataset file which will be separated into labels and instances, then sentence instance will be split into single words and punctuations. Second, some of the stop words in each sentences will be removed. Finally, the sentences will be translated into a set of word list which only retains meaningful and useful words in all sentences.

Performance metric: The confusion matrix (in analyses section) and F-score are used in this experiment to measure model performance.

3.2 Results

The performances of the Bow model with randomly word embedding (accuracy is 0.662) or pretrain word embedding (accuracy is 0.756) is shown respectively as following:

```
D:\Text_Mining\final_report\cw1\src>python question_classifier.py
test -config ../data/bow.config
Using randomly initialize word embeddings
[[2. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 7. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [1. 0. 1. ... 0. 0. 0.]
 [0. 2. 0. ... 0. 0. 1.]]
correct_rate: 0.662
F score: 0.4050514017781793
D:\Text_Mining\final_report\cw1\src>python question_classifier.py
test -config ../data/bow.config
Using pre-trained word embeddings
[[2. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 8. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 3. 0. 0.]
 [1. 0. 0. ... 0. 2. 0.]
 [0. 0. 0. ... 0. 0. 5.]]
correct_rate: 0.756
F score: 0.5223126467432124
```

The performances of the Bilstm model with randomly word embedding (accuracy is 0.796) or pretrain word embedding (accuracy is 0.812) is shown respectively as following:

```
D:\TextMining\cw1_test\cw1\src>python question_classifier.py
test -config ../data/bilstm.config
Using randomly initialize word embeddings
[[2. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 7. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 3. 0.]
 [0. 0. 0. ... 0. 0. 2.]]
correct_rate: 0.796
F score: 0.5529662539472866
D:\TextMining\cw1_test\cw1\src>python question_classifier.py
test -config ../data/bilstm.config
Using pre-trained word embeddings
[[ 2.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0. 11. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  1.  0.  0.]
 [ 0.  0.  0. ...  0.  3.  0.]
 [ 0.  0.  0. ...  0.  0.  2.]]
correct_rate: 0.812
F score: 0.540175487396764
```

The performances of the Bilstm ensemble model with randomly word embedding (accuracy is 0.798) or pre-train word embedding (accuracy is 0.81) is shown respectively as following:

```
MacBook-Pro:src yifanyang$ python3 question_classifier.py test -
config ../data/bilstm_ensemble.config
Using randomly initialize word embeddings
[[ 2.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0. 11. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  1.  0.  0.]
 [ 0.  0.  0. ...  0.  2.  0.]
 [ 0.  0.  0. ...  0.  0.  2.]]
correct_rate: 0.798
correct_num: 399
F score: 0.5187534186853335
MacBook-Pro:src yifanyang$ python3 question_classifier.py test -
config ../data/bilstm_ensemble.config
Using pre-trained word embeddings
[[ 2.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0. 11. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  1.  0.  0.]
 [ 0.  0.  0. ...  0.  3.  0.]
 [ 0.  0.  0. ...  0.  0.  2.]]
correct_rate: 0.81
correct_num: 405
F score: 0.538325487396764
```

Regarding the Bi-LSTM model, it is obvious that the model with glove pre-train word embedding has higher accuracy than the model with randomly word embedding.

3.3 Ablation study

What if we freeze the pre-trained word embeddings: In the word embedding process, the torch lib provides a pre-trained method called $from\_pretrained()$. If the parameter freeze is set as True, the tensor does not get updated in the learning process, which is equivalent to $embedding.weight.requires\_grad = False$ According to the experiment results, there is no obvious difference in the precision rate. For instance, the results of both Bow models that have different freeze sets are about 75.8% (accuracy) and 0.523 (F-score). However, the training time of both Bow models have a huge difference. When the Bow model whose $from\_pretrained()$ with attribution freeze is true, the training time(3min 25s) is relatively shorter than the Bow model with attribution freeze that is set to false (43min 15s).

Randomly initialized word embeddings VS pre-trained word embeddings: There are two different word embedding methods implemented in this experiment: Randomly initialized word embeddings and pre-trained word embeddings. For randomly initialized word embeddings, each word in the vocabulary list will be transformed to a random tensor that has a default dimension. Each tensor will be represented as a corresponding word vector which will be used in Bow model or BiLSTM model. On the other hand, pre-trained word embeddings using the word tensors are created by GloVe. Concerning models' performance, the pre-trained wording embeddings show better property on speed and accuracy. For example, the Bow model using randomly initialized word embedding receives a precision about 66.2% and the training time is 32min 5s, whereas the model using pre-trained word embeddings gathers a better precision of 75.6%, yet it only took 1min 17s.

3.4 In-depth analyses

What if you use only part of the training set: The size of the training set is controlled by the parameter called validation size. Each model is trained on the training set and tested on the validation size which ranges from 10%, 50%, and 70% of the original training set. Especially for BiLSTM model using randomly initialized word embeddings, its accuracy and F-score are 0.806, 0.561; 0.752, 0.452; 0.686, 0.361 respectively. Since the amount of training data is not enough, these models may not be able to learn the feature distribution of many classes, and the overall performance will be worse.

Which classes are more difficult than the other: Classes like Enty: cremat, HUM: title, ENTY: letter, ENTY: religion, ENTY: product, NUM: volsize, ABBR: abb, ENTY: word, ENTY: symbol, NUM: ord, and ENTY: dismed even have 0% accuracy and ENTY: plant, ENTY: veh, ENTY: food has low accuracy. The details can be viewed in the output.txt after the model has been trained and tested.

Confusion matrix: For this fifty classes multi-classification problem, there is a 50 * 50 confusion matrix whose row is the true label and column is the predicted label. In general, the recall and precision cannot be very high at the same time since the F-score will be influenced by both. Besides, there may be an uneven label distribution problem and the Macro-average is more affected by the small category than the Micro-average. As a result, to evaluate the model more objectively, the Macro-average and F-score evaluations are more reasonable.

What if you use other preprocessing steps: We find that less stop word makes model result better. In the experiment preprocessing process, varieties of stop words were tried to deleted from sentences like 'who', 'what', 'how' and 'where', but the accuracy of the model decreases dramatically, which means those stop words could make a huge influence on sentence meaning therefore they should remain in sentences

4 Conclusions

When the same model uses pretrain word embedding, the training of model is faster and has a better predictions of the types of answers to the test questions. When using the same word embeding, the BiLSTM model and BiLSTM ensemble model performs better and is more accurate than the Bow model.

References

[1] Experimental data for question classification. TREC data set. Xin Li and Dan Roth.

[2] Sequence models and longshort term memory networks. Pytorch tutorial.

[3] Deep learning with pytorch. Pytorch tutorial.

[4] Xin Li and Dan Roth. 2002. Learning question classifiers. In Proceedings of the 19th international conference on Computational linguistics - Volume 1 (COLING '02). Association for Computational Linguistics, USA, 1–7. DOI:https://doi.org/10.3115/1072228.1072378