Aspect Based Sentiment analysis

Natural Language Process Project

Libin Cheng 19300740005

The Department of Environmental Science and Engineering

1. Introduction

Sentiment analysis is increasingly viewed as a vital task both from an academic and a commercial standpoint. The majority of current approaches, however, attempt to detect the overall polarity of a sentence, paragraph, or text span, regardless of the entities mentioned (e.g., laptops, restaurants) and their aspects (e.g., battery, screen; food, service). By contrast, this task is concerned with aspect-based sentiment analysis (ABSA), where the goal is to identify the aspects of given target entities and the sentiment expressed towards each aspect. Datasets consisting of customer reviews with human-authored annotations identifying the mentioned aspects of the target entities and the sentiment polarity of each aspect will be provided.

This project is based on subtask 2 of SemEval-2014 Task 4: Aspect Based Sentiment Analysis. We are required to design some methods for sentiment classification specific to an aspect. For a given set of aspect terms within a sentence, determine whether the polarity of each aspect term is positive, negative, neutral or conflict (i.e., both positive and negative). We have two ABSA tasks, one is in the field of restaurant reviews and the other is in the field of laptop reviews. For example: “I loved their fajitas” → {fajitas: positive}; “I hated their fajitas, but their salads were great” → {fajitas: negative, salads: positive}; “The fajitas are their first plate” → {fajitas: neutral}; “The fajitas were great to taste, but not to see” → {fajitas: conflict}.

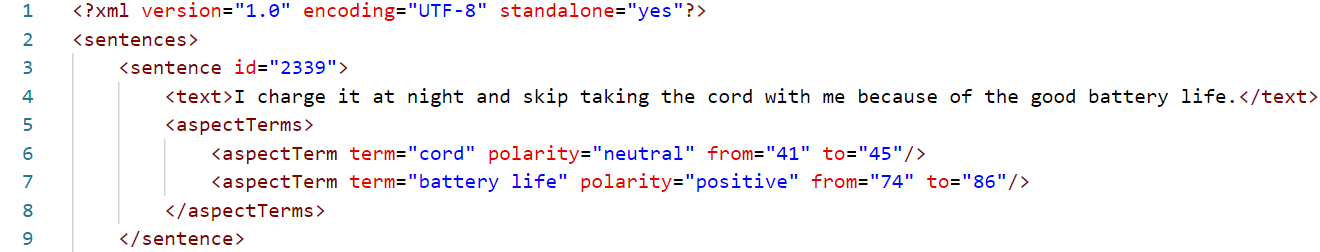
1. Dataset and Format

Two domain-specific datasets for laptops and restaurants, consisting of over 6K sentences with fine-grained aspect-level human annotations have been provided for training.

The dataset of restaurant reviews consists of over 3K English sentences from the restaurant reviews of Ganu et al. (2009). The original dataset of Ganu et al. included annotations for coarse aspect categories and overall sentence polarities; the data provider modified the dataset to include annotations for aspect terms occurring in the sentences, aspect term polarities, and aspect category-specific polarities. Experienced human annotators identified the aspect terms of the sentences and their polarities. Additional restaurant reviews, not in the original dataset of Ganu et al. (2009), are being annotated in the same manner, and they will be used as test data.

The dataset of laptop reviews consists of over 2K English sentences extracted from customer reviews of laptops. Experienced human annotators tagged the aspect terms of the sentences and their polarities. Part of this dataset will be reserved as test data.

The sentences in the datasets are annotated using XML tags. The following example illustrates the format of the annotated sentences of the restaurant’s dataset. The possible values of the polarity field are: “positive”, “negative”, “conflict” and “neutral”. The possible values of the category field are: “food”, “service”, “price”, “ambience”, “anecdotes/miscellaneous”. The format of the laptop’s dataset is the same as in the restaurant datasets, with the only exception that there are no annotations for aspect categories.

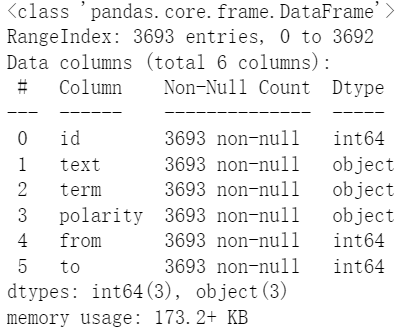
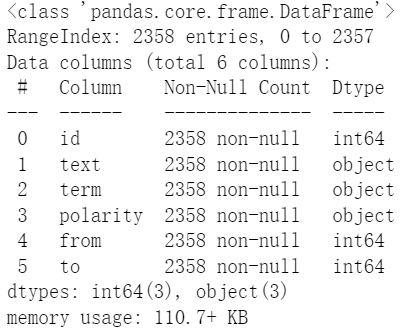


1. Data preprocessing

Firstly, we need to extract useful data from the XML file, which includes the ID of the sentence, the *<text>*, the content of each *<aspectTerm>* in *<aspectTerms>*, polarity, from, and to. Then we need to convert the information of these contents into a CSV file, with each column containing: *id*, *text*, *term*, *polarity*, *from*, and *to*. The dataset can be found in Kaggle. Then, we can further analyze the data and use *etree.ElementTree* in the *XML* library to parse the XML data, obtaining a list of *text\_ List* and a dictionary operation with term as the key and polarity as the value, using *nltk.FreqDist (operation).most\_common(t)* method obtains the most common *t* aspects, which can analyze which aspects the comment is most concerned about. For example, the following figure shows the top 20 areas that comments are most concerned about in the restaurant dataset.



While processing the file, we can see that there are some abnormal space characters in the file, with which we replace the symbols with spaces. So after preprocessing the file, we get the information of CSV file as follows (left is the dataset of restaurants and right is the dataset of laptop).

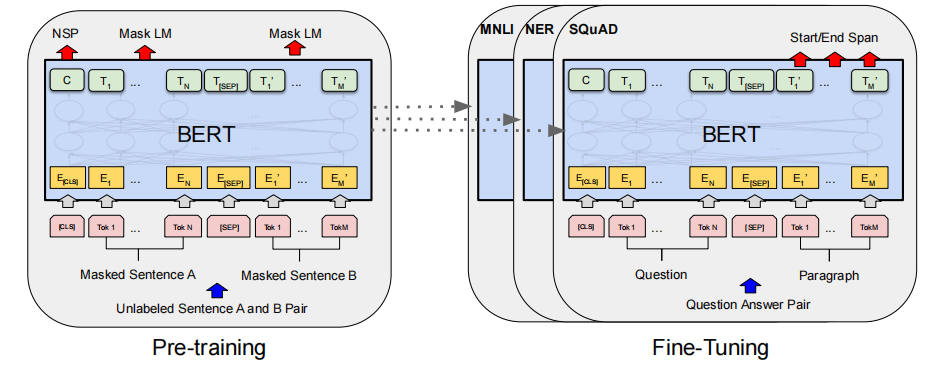
 

Since the test set given by the website does not have polarity, we set 90% of the data set as a training set, and of the remaining 10%, 40% are used as a verification set, and 60% are used as a final test set. The obtained training set, validation set and test set information are shown in the figure below. The right is the restaurant’s dataset and the left is the laptop’s dataset. It represents *df\_train.shape*, *df\_val.shape* and *df\_test.shape*, relatively.

1. Methods

In this project, we use BERT to analyze aspect based sentiment. The full name of BERT is Bidirectional Encoder Representation from Transformers, which is a pre-trained language representation model. It emphasizes that instead of using the traditional one-way language model or shallow splicing of two one-way language models for pre-training, it uses a new masked language model (MLM) so that it can generate deep Bidirectional language representations. This model has the following benefits: 1) Use MLM to pre-train bidirectional Transformers to generate deep bidirectional language representations. 2) After pre-training, we only need to add an additional output layer for fine-tune, and we can achieve state-of-the-art performance in various downstream tasks. No task-specific structural modifications to BERT are required during this process.



In our project, we load the *bert-base-cased* pre-trained model from *BertTokenizer* library and create a corresponding tokenizer object. 'bert-base-cased' is one of the base versions of the BERT model. The base version usually refers to a relatively small model with fewer parameters and computational resource requirements. Compared with other versions (such as large or xlarge), the base version is lighter in model size, but still has good performance.

In order to better process the data, we created a *Reviews* class, which is a subclass of *PyTorch*'s *Dataset* class. This class is used to process data containing information such as reviews, review facets, target tags, etc. This class contains the following parameters: *reviews*, *review\_aspects*, *targets*, *tokenizer* and *max\_len*, which can obtain the corresponding review content and target tags from attributes such as reviews and targets according to the index item, and use the tokenizer object to encode the review content and convert the review content into an input format that the model can accept, and perform necessary padding and truncation operations. The encoding result is stored in the encoding variable in the form of a dictionary. By creating an instance of the *Reviews* class, the raw reviews dataset can be converted into a dataset object suitable for training and evaluating BERT models. This custom dataset can be used with *PyTorch*'s data loaders for bulk loading and processing of data.

Then we define the *PyTorch*'s data loader, which receives the following parameters:

* *df*: DataFrame containing the data, which includes columns such as text, review facet, and target label.
* *tokenizer*: A tokenizer object used to convert text into model input.
* *max\_len*: The maximum text length set for padding and truncating text.
* *batch\_size*: The number of samples for each batch.

By calling the data\_loader function, we can easily create data loader objects for bulk loading and processing of datasets. These data loaders can be used with train, validation, and test steps to efficiently train and evaluate models.

Then we need to use BERT to build a sentiment analysis model. We define a *PyTorch* model class which inherits from *nn.Module*, indicating that this class is a neural network model. It receives a parameter *n\_classes*, which represents the number of categories in the sentiment classification task (four categories in our case). The key implementation here is:

*self.bert = BertModel.from\_pretrained(PRE\_TRAINED\_MODEL\_NAME, output\_hidden\_states=True, output\_attentions=True, return\_dict=False),*

which creates a BERT model object and loads the pre-trained BERT weights. *PRE\_TRAINED\_MODEL\_NAME* is the name of the pre-trained model, which specifies the type of BERT model to use, here is 'bert-base-cased'. *output\_hidden\_states=True* and *output\_attentions=True* means to get all hidden states and attention weights of the BERT model. *return\_dict=False* indicates that the returned result will be in the form of a tuple output by the BERT model. We then define a dropout layer for random inactivation in the output of the model with a dropout probability of 30%. Then we define a linear layer (full connection layer), take the hidden state size of the BERT model as the input feature number, and map it to *n\_classes* output categories. Then we define the forward propagation method of the model class, which receives two parameters: one is the input token ID sequence, which is used to represent the input text; the other is the input attention mask, which is used to indicate which parts are real valid text.

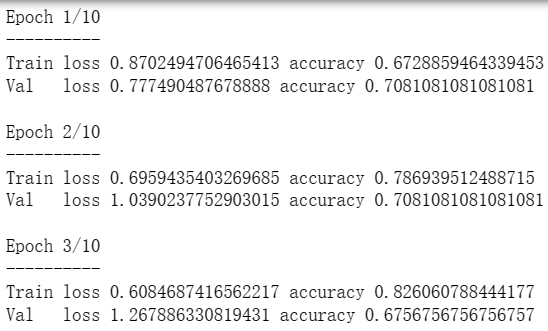
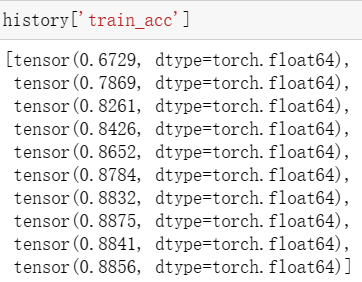
During forward propagation, the input token ID sequence and attention mask are processed through the BERT model. By calling *self.bert(input\_ids=input\_ids, attention\_mask=attention\_mask)*, we can get the output tuple of the BERT model, which contains multiple outputs of the BERT model, including the hidden state of the last layer, pooling output, hidden state sequence and attention weights. However, in this project, we only focus on the pooled output (*pooled\_output*), which is the representation obtained by the BERT model after summarizing the information of the entire input sequence. This means that after the dropout layer is randomly inactivated, the classification prediction is performed through the linear layer (full connection layer), and finally the prediction result is returned.

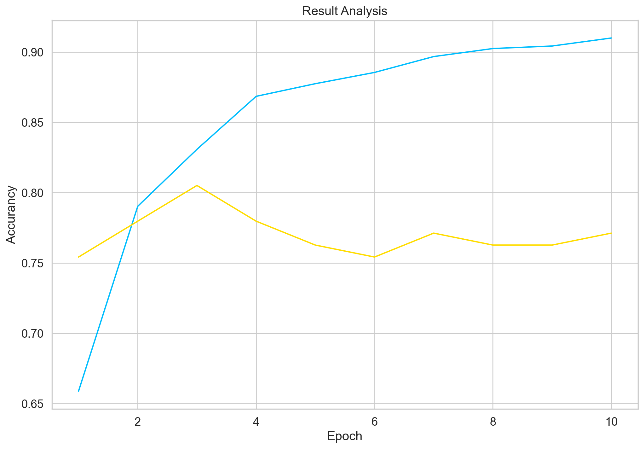
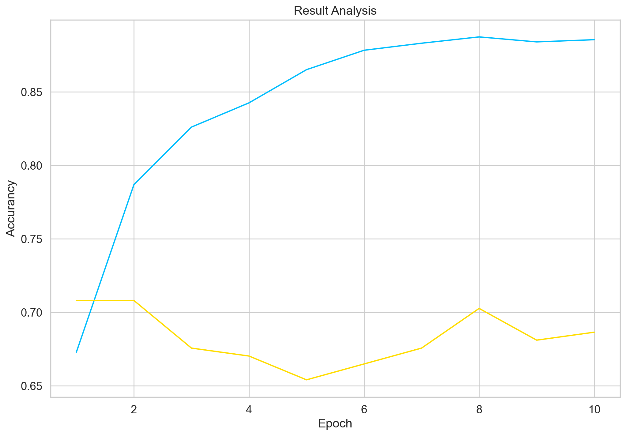
Next, we set some parameters and tools in the training process, including optimizer, learning rate scheduler and loss function. We train for 10 iterations on the entire training dataset. We use the AdamW optimization algorithm, use the parameters of the model (*model.parameters()*) as the parameters of the optimizer, and set the learning rate (*lr=2e-5*) and whether to correct the deviation (*correct\_bias=False*). Next, we get the total number of training steps that need to be performed by multiplying the number of batches in the training data loader by the number of training epochs. Then, we created a learning rate scheduler object with *get\_linear\_schedule\_with\_warmup*. It can automatically adjust the learning rate according to the progress of the training based on the given optimizer, the number of warm-up steps and the total number of training steps. Our loss function chooses the cross-entropy loss function.

Finally, we designed a training loop function, traversed the data loader, obtained a batch of data from the data loader for each iteration, and set the attention mask, used the model for forward propagation, and obtained the predicted output of the model, to obtain the predicted class for each sample by finding the maximum value over the first dimension of the predicted output tensor. We also make use of the forementioned cross-entropy loss function to calculate the loss between the model's predicted output and the target label. Next, we calculate the number of correct predictions by comparing the predicted class and target label for equality and summing the results. We then backpropagate through the computational graph to compute the gradient of the loss with respect to the model parameters. After the loop finishes, calculate the average loss and accuracy for the entire epoch and return these values.

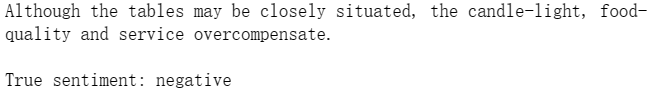
1. Evaluations

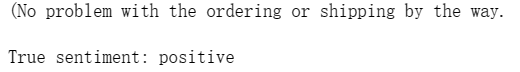
Here is the training process and the training result is as follows (right is the dataset of restaurants and left is the dataset of laptop):

Then we get some sentences and test the model (above is the dataset of restaurants and below is the dataset of laptop):



1. Discussion

In this natural language processing project, we used the BERT language model for sentiment analysis and obtained preliminary results. This model can be used for simple sentiment classification tasks, but it also has the disadvantages of insufficient accuracy (67%-80%) and relatively small number of samples (only two or three thousand pieces of data). Moreover, with the expansion of training times, the accuracy of the test set will decrease slightly. This requires us to further explore the reasons in our future work.

1. References
2. <http://alt.qcri.org/semeval2014/task4/>
3. <https://www.kaggle.com/datasets/charitarth/semeval-2014-task-4-aspectbasedsentimentanalysis>
4. <https://github.com/claragordon/SentimentAspect>
5. <https://github.com/Angohuangg/ABSA-semeval-2014-labtop/tree/main>
6. <https://github.com/Diego999/SemEval-2014-Task-4-ABSA>
7. <https://github.com/1tangerine1day/Aspect-Term-Extraction-and-Analysis/tree/master>
8. [图解BERT模型 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/318495113)
9. [【NLP】Google BERT模型原理详解 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/46652512)
10. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
11. G. Ganu, N. Elhadad, and A. Marian, “Beyond the stars: Improving rating predictions using review text content”. Proceedings of the 12th International Workshop on the Web and Databases, Providence, Rhode Island, 2009.