Sentiment Analysis using BERT

This project uses BERT, a state-of-the-art pre-trained model for natural language processing, to perform sentiment analysis on a dataset of customer reviews. The dataset consists of sentences in three different opinion polarities (positive, negative, or neutral).

Prerequisites

To run the code, you will need to create a python environment and install:

a. Python >= 3.9.x
b. pytorch = 1.13.1
c. transformers = 4.22.2
d. datasets = 2.9.0
e. numpy = 1.23.5
f. pandas = 1.5.3

Dataset

The dataset used in this project can be found in the **/data** directory. It consists of two CSV files: **traindata.csv** and **devdata.csv**. Each row in the CSV files contains five elements: **opinion polarity, aspect category, target term, character offsets** and **original sentence**. There are 12 different aspect categories in total:

AMBIENCE#GENERAL
DRINKS#PRICES
DRINKS#QUALITY
DRINKS#STYLE_OPTIONS
FOOD#PRICES
FOOD#QUALITY
FOOD#STYLE_OPTIONS
LOCATION#GENERAL
RESTAURANT#GENERAL
RESTAURANT#MISCELLANEOUS
RESTAURANT#PRICES
SERVICE#GENERAL

Algorithm

Four files can be found in the src doc:

preprocessing.py: This file defines a set of functions and classes to preprocess and prepare a dataset for a sentiment analysis task using PyTorch.

The **preprocessing** function takes in the raw data file and applies label encoding to the polarity column and maps aspect categories to corresponding questions. This function essentially preprocesses the raw data by converting the aspect terms to questions and sentences to replies. [1]

The **max_len_token** function takes in the preprocessed data file and the tokenizer and calculates the maximum token length required for padding/truncating input sequences to a fixed length. This function returns the maximum token length for the dataset.

The **create_dataset** class takes in the preprocessed data file, the tokenizer, and the maximum token length as input and creates a PyTorch dataset. The **__len__** method returns the length of the dataset, and the **__getitem__** method returns a dictionary with the input sentence, the encoded input IDs, attention mask, and the corresponding polarity label. The **encoding** object is created by encoding the sentence and aspect category pair, adding special tokens, and padding/truncating the sequence to a fixed length.

model.py: This file contains the PyTorch model for the text classification task. It defines the neural network architecture that will be used for training and inference.

The model is defined as a class called **SentimentCheck**, which inherits from the **nn.Module** class in PyTorch. The **__init__** method initializes the BertModel from the **bert-base-uncased** pre-trained model in transformers, a dropout layer with a probability of 0.4 and a fully connected layer with **n_classes** output nodes.

The **forward** method is defined to perform a forward pass through the network. It takes in **input_ids** and **attention_mask** as input, which are the encoded input and attention mask produced by the tokenizer during preprocessing. The input is passed through the **BertModel** to obtain the output, which is then passed through the dropout layer and fully connected layer to obtain the final output tensor. [2]

classifier.py: This file contains the **Classifier** class, which is responsible for training the model, making predictions, and evaluating the model's performance. It uses the functions defined in **model.py** and **preprocessing.py** to build and preprocess the data before training and inference.

The class loads training and validation data from files, preprocesses it, encodes it using a pretrained BERT tokenizer, and creates dataloaders for training and validation data. The class defines a BERT-based neural network for sentiment analysis and trains it using backpropagation and a cross-entropy loss function. The class uses the Adam optimizer and a linear learning rate scheduler. The class also uses gradient clipping to avoid exploding gradients. The model's accuracy is evaluated on the validation set, and the best-performing model is returned.

tester.py: This file contains the code for running multiple runs of the **Classifier** on the training, validation, and test sets, and reporting the accuracy for each run. It uses the **argparse** module to parse command line arguments, and the **time** module to measure the execution time.

In summary, the **SentimentCheck** class in the **model.py** file defines the architecture of the BERT-based model. It consists of a pre-trained BERT model followed by a dropout layer and a fully connected layer. The **classifier.py** script trains the model on the training set and evaluates it on the development and test sets. The **tester.py** script runs multiple experiments with different random seeds to obtain a more accurate estimate of the model's performance. Several trials with different combinations of recommended hyperparameters were done before arriving at the final set. [2]

Results

The model executes 5 runs in 892.20 s (178 per run) and achieves accuracy of [86.97, 86.44, 86.7, 87.23, 88.83] on the **devdata** set with an average of 87.23.

Authors

Jinji SHEN (jinji.shen@student-cs.fr)
Mengyu LIANG (mengyu.liang@student-cs.fr)
Nhat Mai NGUYEN (nhatmai.nguyen@student-cs.fr)
Yanjie REN (yanjie.ren@student-cs.fr)

Resources

[1] Li, Y., Xu, L., Liu, X., Li, M., & Zhao, E. (2019). Preparing Pretraining

Data for Contextualized Embeddings with Entity-Aware and Aspect-Aware Preprocessing. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1, Long and Short Papers) (pp. 3449–3458). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1035

[2] 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. Retrieved from https://arxiv.org/pdf/1810.04805.pdf