HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

MSC BIG DATA TECHNOLOGY

6000H - Natural Language Processing

Argument Impact Classification

Wong Chin Hang 20123808
Le Berrigaud Lohan 20753689
Fan Vincent Ka Chun 20754774
TONG Ka Kiu Cathy 20671611

1. Your team name, names & student ids of team members, your rank and scores on the leaderboard, and the display name of your kaggle account.

Team name: group	Names & student ids of team members: Wong Chin Hang 20123808 Le Berrigaud Lohan 20753689 Fan Vincent Ka Chun 20754774 TONG Ka Kiu Cathy 20671611
Rank 30 Score: 0.55606	Display name of your kaggle account: Cathy Tong, fancent, CHIN HANG WONG, LLB0611

2. What algorithms are you using in this project?

We have tested several models, including Naive Bayes, CNN, RNN and BERT + FNN.

In the project, we have got 3 data sets. They are: training set, validating set and testing set. For each data set, we first loaded the csv file and returned a list of ids, a list of texts, a list of context, a list of joined texts which joined texts and contexts together within the same entry, a list of stance labels and a list of labels.

In our project, we have tried Naive Bayes and CNN. For Naïve Bayes, since the training data is imbalanced that it is dominated by “Impactful”, Naive Bayes can handle the imbalance data set since the prior accounts for the label distribution. In Naive Bayes, we have joined texts with contexts within entry. Then we have created a feature matrix from texts which have been tokenized and stemmed. The training feature matrix and labels are used to train Naïve Bayes Classifier.

In CNN, we have tokenized and stemmed the texts. Stop words are then filtered from texts after tokenized and stemmed. We have built a feature dictionary as well as feature matrix for training CNN model. Our CNN model consists of three parts: the word representation part, the convolutional and pooling layer, and the fully connected part. The word representation part is the word embedding layer. The convolution and pooling layer part includes 4 convolutional layers with different kernel sizes, max pooling layers to aggregate extracted features. The fully connected part utilizes a multi-layer perceptron to make classification on output class.

In RNN, we have joined texts and contexts within entries. Then we have tokenized and stemmed the joined texts. Stop words are then filtered from texts after tokenized and stemmed. Stop words are also filtered. We have built a feature dictionary as well as feature matrix for training RNN model. Our RNN model consists of three parts: word representation part, recurrent part, and fully connected part. The word representation part is for word embedding. The recurrent layers include bi-directional recurrent layers to memorize and summarize contextualized word features. The fully connected part utilizes a multi-layer perceptron to make predictions on output classes.

We decided to employ BERT + FNN as our final model. BERT stands for Bidirectional Encoder Representations from Transformers. It makes use of transformers that learns contextual relations between words within a text. In the BERT model, the transformer encoder reads the entire sequence of texts that it learns the context of a word from surroundings, including left and right of the words. It is different from directional models that only read the text input from left-to-right or right-to-left.

In the BERT model, 15% of words in the sequence are randomly chosen to mask. They are replaced by mask tokens. The model then predicts the original value of masked words based on other non-masked words in the sequence. Moreover, in the BERT training process, the model receives sentences in pairs as input and learns to predict whether the second sentence is the next sentence in the original document. During training, half of the inputs are pairs that the second sentence is the subsequent sentence in the original document. The other 50% are sentence pairs randomly. BERT model makes use of both Masked LM and Next Sentence Prediction.

To prepare data as model input, we have initialized bert tokenizer from pretrained version. Then we have removed stop words from texts and contexts. Then we have combined stance labels with both texts and contexts. The resulting texts and contexts are encoded using BERT tokenizer. To save memory consumption during training, we have created PyTorch DataLoader with a custom data member class so that we can use it to feed into the BERT model.

In our model, it makes use of a pretrained model on English that includes masked language modeling. After the BERT based model, there is a feedforward neural network layer. The second layer will be the classification layer that is used to predict the output classes of impact labels. AdamW will be the optimizer which is an improved version of the Adam optimizer. There is weight decay in AdamW optimizer only after changing the size of parameter-wise step and therefore the weight decay is proportional to the weight itself. It yields better training loss.

3. How do you conduct parameter tuning? Is there any difference between your local validation and online results?

We used an empirical approach for the choice of hyperparameters. We tried to fine tune hyperparameters by comparing F1 scores among them. We did a manual grid search for each of the hyperparameters. The hyperparameters include learning rate, number of layers, number of nodes of each layer, epsilon of Adam, batch size, dropout rate.

The local validation and online results are pretty similar which led us to think that we are not overfitting nor undefitting.

4. How to run your code? Which third-party libraries are you using?

We used Google Colab to run our code because it provides gpu acceleration and it is a convenient platform to share work among teammates.

We used several libraries for different purposes:
1. Numpy, pandas, collections and scipy are employed for general math processing.
2. Pytorch, tensorflow and sklearn are employed for building Naive Bayes, CNN and RNN and provide K-fold cross validation and metrics evaluation.
3. Transformers are employed for Bert model building.
4. Nltk is used for natural language processing.
5. Matplotlib is used for data visualization.
6. The ast module helps Python applications to process trees of the Python abstract syntax grammar.

To run the code, simply run all the cells in the follow sections in the following order:

1. Library Import
2. Data Prep
3. Feature Extraction
4. BERT
5. Output prediction to kaggle

The rest of the notebook code is used for showcasing the methods we tried as mentioned above, including Naive Bayes, CNN and also in depth analysis on the data set. Note that although we used a dataloader, Collab might still run into CUDA insufficient memory error depending on which GPU google assigned. Please factory reset the runtime or connect to a different GPU.