# CSE 354 Assignment 3 Report Spring 2022

JeongYoon Lee (114133199)

April 26, 2022

## Part 1. Model Implementation

1. Problem 1 (Initialize the Model Class)

   I used AutoTokenizer.from_pretrained() for tokenizer and AutoModelForSequenceClassification.from_pretrained() for model which is already imported. I put the model_name for both function, and also passed number of classes through the model's second arguments.

2. Problem 2 (Initialize the Dataloader Class)

   I used encode_plus to get the token ids values for each review. I use arguments options with "truncation = True" and "max_length = 512" to get the 512 tokens. It will give first 512 tokens if it is longer than 512. And then appended to tokens list.

   For the label, we can find in label_dict list with label index. And then appended to labels list.

3. Problem 3 (Training Function)

   3-(a)

   For the set_training_parameters, I made conditional filter with the name of training_type. If the training_type is "frozen_embeddings", freeze embeddings layer, if the training type is "top_2_training", freeze layer 0,1,2,3 and embeddings layer. And if training type is "top_4_training", freeze embeddings layer and layer 0 and layer 1. Finally for the "all_training", not freezing anything. For the freezing, I used layer.requires_grad = False.

   3-(b)

   For the train function, send reviews and labels through self.model, and get loss value from that result. And then do propagate the loss backwards to the model and update optimizer's parameters with zero_grad(), backward(), and step().

   Also, we can find the precision, recall and f1 value from the get_performance_matrics function using the result from the model. We need to use cpu since gpu doesn't do numpy computation.

   3-(c)

   For the validation part, it has same process with 3-(b), but except loss backwards and update optimizer. It is not needed for this part. And Testing as well.

4. Problem 4 (Test Function)

   For the Test Function, the code is same with validation. So just past the reviews and labels using cpu, and get the loss from that result. And then get the precision, recall and f1 from the get_performance_metrics function.

## Part 2. Precision, Recall, F1 scores (Testing)

Experiment 5 : test_loss: 0.2712 test_precision: 0.8627 test_recall: 0.8980 test_f1: 0.8675
→ Frozen embeddings

Experiment 6 : test_loss: 0.2880 test_precision: 0.8523 test_recall: 0.9275 test_f1: 0.8804
→ top_2_training

Experiment 7 : test_loss: 0.2863 test_precision: 0.8557 test_recall: 0.9313 test_f1: 0.8850
→ top_4_training

Experiment 8 : test_loss: 0.2595 test_precision: 0.8755 test_recall: 0.9067 test_f1: 0.8841
→ all_training

| | Loss | Precision | Recall | F1 |
|---|---|---|---|---|
| Experiment 5 | 0.2712 | 0.8627 | 0.8980 | 0.8675 |
| Experiment 6 | 0.2880 | 0.8523 | 0.9275 | 0.8804 |
| Experiment 7 | 0.2863 | 0.8557 | 0.9313 | 0.8850 |
| Experiment 8 | 0.2595 | 0.8755 | 0.9067 | 0.8841 |

## Part 3. Analysis explaining

From my experiment result, the experiment 8 has the highest accuracy which loss value is 0.2595, and also has high precision and f1 value. The more layers we used for training, the probability that model has overfitting will increase. However, each cases' training accuracy and validation accuracy is almost 98%, we cannot surely decide this is overfitting. But since the experiment 8 has the best result which is the test result with training all the layers (means that it has smallest training data), I think this doesn't have much overfitting problem.

But as I run this code several times, this result differ from all the trials. And since each experiments' result don't have big difference between each values, we can try other new tuning to find their relationship and find the optimum model.