**Shared Task: SemEval22 PCL Classification (Task 4 Subtask 1)**

**Group 10**

**Members:**

Ragul Venkataraman Ravisankar,

Rahul Kumar Mohan Kumar

Aravindakumar Vijayasri Mohan Kumar,

**Problem Definition:**

The given task is to perform a binary classification of a given paragraph into either patronizing and condescending language (PCL)or non-PCL. We have used the Bert sequence classification approach to solve this problem.

**Data Input and Preprocessing:**

The SemEval github repository provides a module called DontPatronizeMe which helps in fetching the input data. We load the module first and then load the data. The data is in tsv format so we make use of pandas to get the data frames and do the initial pre-processing. After initial data inspection we split the data into texts and labels.

**Dataset Split:**

The given data is first split into train and validation sets using stratified (80:20) split so that the ratio of both class labels are the same.

**Oversampling Minority Class:**

In the train split, there was a high imbalance between labels 0 and 1. Label 0 the majority class (7581) and label 1 the minority class(794). So, we oversampled the minority class (label 1) using RandomOverSampler.It is also done to reduce quantization noise[1]. After oversampling, we get 15162 data points (7581 Class 0 and 7581 Class 1).

**Validation Set:**

The number of label testing samples are 2094 among which 1895 samples are for label 0 and 199 samples for label 1.

**Tokenizer:**

We make use of the hugging face transformers package to import a pretrained Bert tokenizer. For the classification task, a single vector representing the whole input sentence is needed to be fed to a classifier. In BERT, the decision is that the hidden state of the first token is taken to represent the whole sentence[2].

**Word Segmentation:**

After loading the Bert tokenizer, we tokenize the original sentences and encode them to use for training our model. We also add head and tail tokens by default to the encoded data. After the data is divided into words, we store it in a list. Two additional tokens are added manually to the input sequence. [CLS] is added to the beginning of the input text to represent the beginning of the whole sentence. [SEP] is added at the end of the input text to inform the model where the first sentence ends and the second sentence begins. We then again encode the data to return it in PyTorch tensors format. The data is added to the list and the attention mask of the text is added to a new attention\_mask list.

**Training and Validation Data Loaders:**

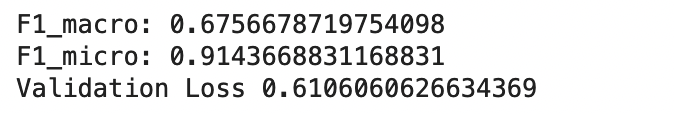
We first start by initializing the batch size to 32 and creating a data loader for the training and validation datasets. The data loader consists of the dataset, a sampler (random for training and sequential for validation) and the batch size.

**Fine Tuning Bert:**

Next, we load the BertForSequenceClassification, pre trained bert model and a linear classification layer and initialize the model with parameters (“bert-case-uncased“, num\_labels=2, output\_attention=False, output\_hidden\_states=False). We then initialize cuda to run the model in gpu. An AdamW[3] optimizer is added to optimize weight decay and learning rate separately so that changing the learning rate will not change the optimal weight decay. Next, the epochs are initialized to 5 and a learning rate scheduler[4] is created to adjust the learning rate during training. We then start the training in batches of 40 and calculate the average training loss and training time for each epoch.

**Validation Results:**

Once training is complete we run the validation and calculate the F1\_macro and F1\_micro along with validation loss and validation time. The validation results are as follows:

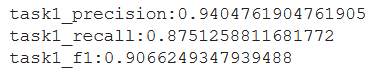


**Generate predictions on Train Data:**

The predictions for the task are generated on the training data and stored to a file. Along with this the gold labels are also generated and stored. These will be used to evaluate the performance of the model.

**Evaluation and Results:**

The official github repo provides an evaluation.py file which is run against the predicted labels. The evaluation.py file calculates the precision, recall and f1 score against the gold labels. Upon evaluation the results obtained are as follows:



**References:**

[1] Camino, R., Hammerschmidt, C. and State, R., 2020. Minority Class Oversampling for Tabular Data with Deep Generative Models. *arXiv preprint arXiv:2005.03773*.

[2] <https://albertauyeung.github.io/2020/06/19/bert-tokenization.html/>

[3] <https://www.iprally.com/news/recent-improvements-to-the-adam-optimizer>

[4]<https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>