National University of Computer and Emerging Sciences

Artificial Intelligence (HCI)

Project – Sentence Validation

NAME:	Areeba Nasir
ROLL NUMBER:	i17-0052
DEGREE PROGRAM:	BSCS
SECTION:	C
DATE OF SUBMISSION:	December 9, 2020
SUBMITTED TO:	Mr. Umair Arshad

Introduction:

Sentence Validation, as the name suggests, is a Natural Language Processing (NLP) task where the input features consist of two pieces of text. The aim is to choose from two natural language statements with similar wordings which one makes sense and which one does not make sense. For example,

Sent0: The bed is in the dog.
Sent1: The dog is in his bed.

Sent0 is against the common sense. So, the proposed model should be able to tell which sentence amongst the two is against common sense.

Proposed Approach:

For this project I used multiple approaches from machine learning models to deep learning models. "Other Approaches" contains all the approaches that I tried but didn't give a good accuracy score. "Final Approach" describes the approach that is finally used for sentence validation.

- **Other Approaches:**
  - **Random Forest:**
    Random forest was implemented for sentence validation with 200 iterations. But this machine learning model did not give a good accuracy.
  - **Sequential Deep learning model:**
    Sequential deep learning model was used to find the sentence against common sense. In this model, following parametes were used:
    - Count vectorizer was used for tokenization and conversion into numeric data along with other data preprocessing.
    - "relu" activation function was used at each layer.
    - Sigmoid function was used at output layer.
    - Epoch value was set to 20
    - Number of dense layers was 4.

    The accuracy of this model was 50% which was not good.
  - **Multichannel Neural Network:**
    Multichannel neural network was used with the following hyperparameters:
    - The model contains 4 channels and each channel has an embedding layer, 1D conv, max pooling layer and relu activation function.
    - First layer of model has L2 Regulizer to stop the model from overfitting on train dataset

    The accuracy of this model was 58%

- **Final Approach**

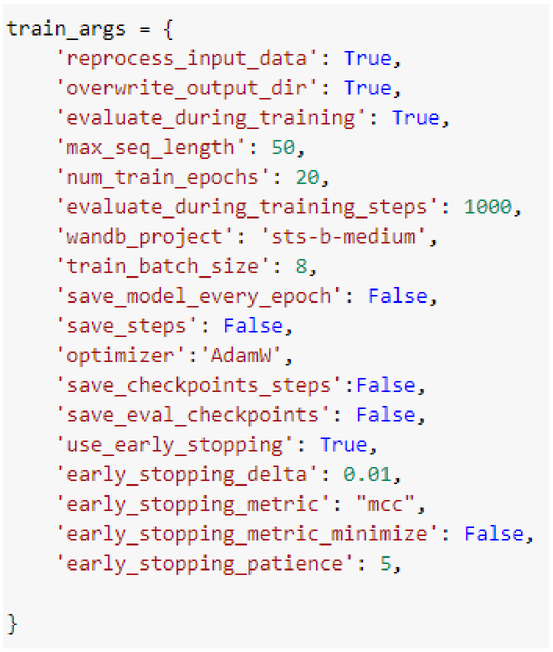
  The final approach that is used in the project is "Simple Transformers: **ROBERTa**". Considering the unprecedented success of ROBERTa and other Transformer models in many NLP tasks, I used transformer model to solve sentence validation problem.
  I performed following steps in this approach:

  1. Prepare the data:

  Simple Transformers expects a Pandas Dataframe. In our probem it will accept 3 columns; text_a, text_b, and labels. Any additional columns will be ignored. The name format of columns should be exactly the same. So, the column names are renamed as "sent0": text_a, "sent1": text_b, "answer: labels".

  2. Configuration Options:

  There are many configuration options available in Simple Transformers to help fine-tune a model. The list of hyper-parameters I used is as follows:



```
train_args = {
    'reprocess_input_data': True,
    'overwrite_output_dir': True,
    'evaluate_during_training': True,
    'max_seq_length': 50,
    'num_train_epochs': 20,
    'evaluate_during_training_steps': 1000,
    'wandb_project': 'sts-b-medium',
    'train_batch_size': 8,
    'save_model_every_epoch': False,
    'save_steps': False,
    'optimizer':'AdamW',
    'save_checkpoints_steps':False,
    'save_eval_checkpoints': False,
    'use_early_stopping': True,
    'early_stopping_delta': 0.01,
    'early_stopping_metric': "mcc",
    'early_stopping_metric_minimize': False,
    'early_stopping_patience': 5,

}
```

  The model is trained with **epoch 20**, and **batch size 8**. Moreover, it uses **Early Stopping** to stop training when early stopping metric does not improve on the basis of early_stopping_patience, and early_stopping_delta. The model is evaluated during every 1000 training steps with the help of validation dataset. "AdamW" is used to optimize the model.

3. Model Creation:

There are many pre-trained models available like BERT, XLNet, XLM, and RoBERTa but I used ROBERTa model. ROBERTa is a revolutionary self-supervised pretraining technique that learns to predict intentionally hidden sections of text.
The method ClassificationModel() creates the model and I passed the following parameters in this method:
model_type: robert, model_name: roberta-base, num_labels: 2, args: train_args

4. Model Training:

The next step is the model training. Simple Transformers will automatically run and log the results from any function passed into the training method as keyword arguments. The method train_model takes training data frame, validation data frame, and any additional parameters such as pearson_corr, spearman_cor. After running th train_model command, it will start training the model on the given dataset.

Results:

Evaluation is the step to get the results from the model so that we can know how much better the model is.

5. Model Predictions:

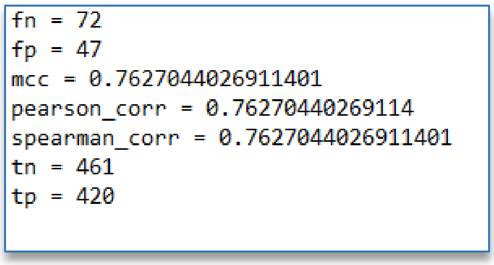
The method eval_model is used to get the results from the trained model. It takes test dataset in the form of list of lists as a parameter and perform predictions on it. In case of our problem, the format of test dataset passed to this method is as follows:

[[sent0 , sent1] , [sent0 , sent1], [sent0, sent1], ……….]

This method returns three values:

1. result: It is in the form of a dictionary. It will include any metric functions passed into the eval_model method.
2. model_outputs: It contains the probabilities of each class for all examples in the test dataset.
3. wrong_predictions: It contains a list of input features of each incorrect predictions.

With the hyperparameters specified, I got the following result:



```
fn = 72
fp = 47
mcc = 0.7627044026911401
pearson_corr = 0.76270440269114
spearman_corr = 0.7627044026911401
tn = 461
tp = 420
```

The accuracy of the model on test dataset is **88.1**.

Graph:

The visualization of the model is done with the help of "**Wandb**". Each time the model script is run instrumented with **wandb**, the hyperparameters and output metrics of the model are saved to the wandb.

At training phase:

Figure 1 shows the training loss of the model over the steps.

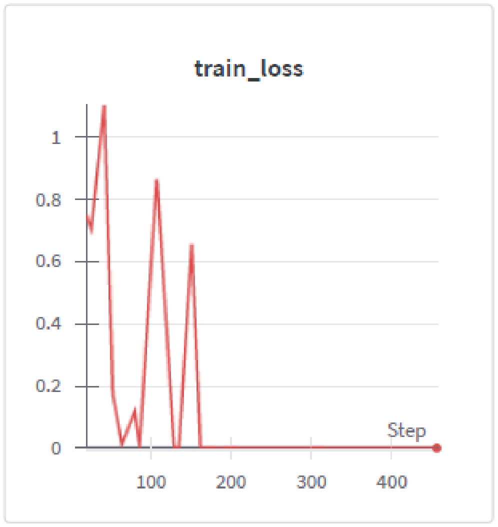


Figure 1

Figure 2 and 3 shows the Spearman rank-order correlation and Pearson product moment correlation of the model.

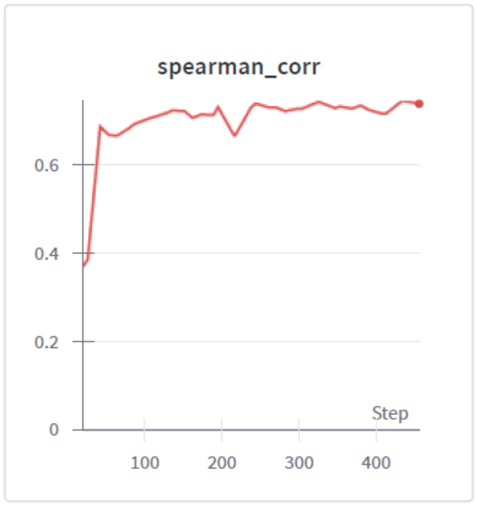


Figure 2

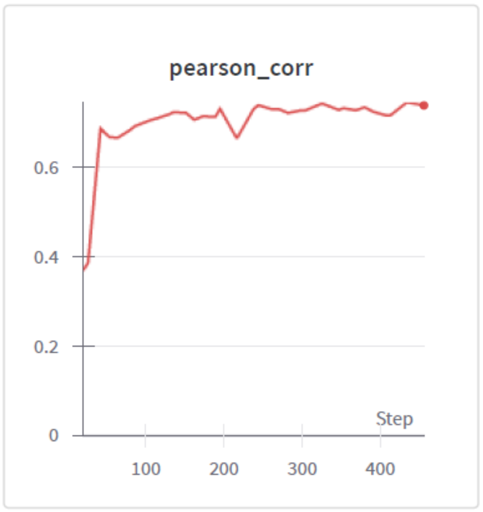


Figure 3

At prediction phase:

In figure 4, the left pie chart shows the amount of real sent0 and sent1as invalid (against common sense).

The right pie chart shows the amount of predicted sent0 and sent1 as invalid (against common sense).

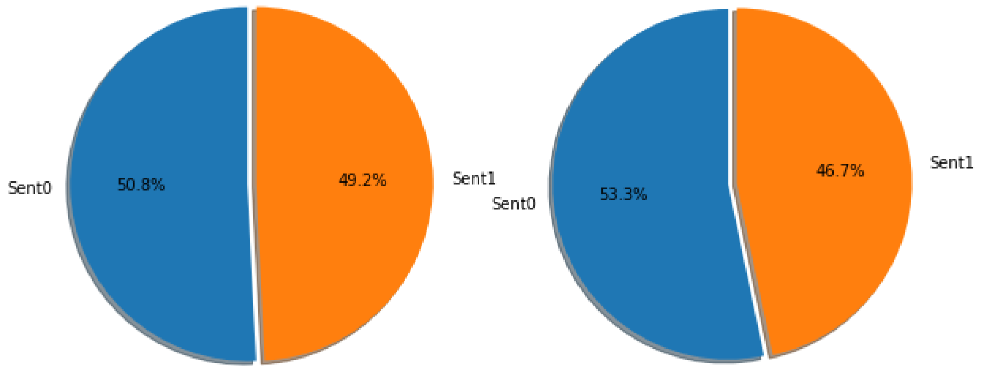


Figure 4

Figure 5 shows the ROC curve of the model.

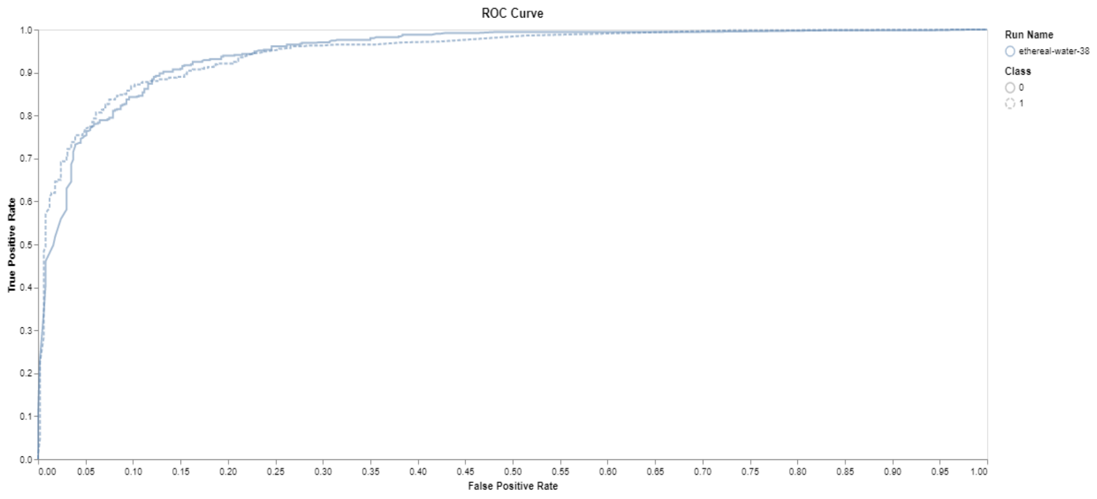


Figure 5

Figure 6 shows the confusion matrix of the model. As we can observe, the number of sent 0 predicted as sent 1 is greater than the number of sent 1 predicted as sent 0.

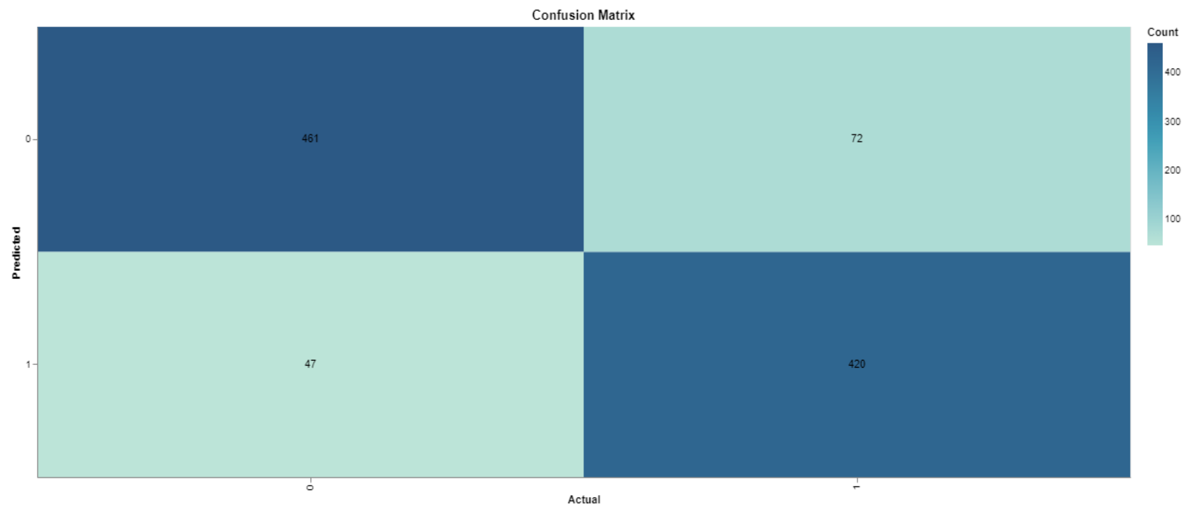


Figure 6

Figure 7 shows the prediction and recall curve of the model.

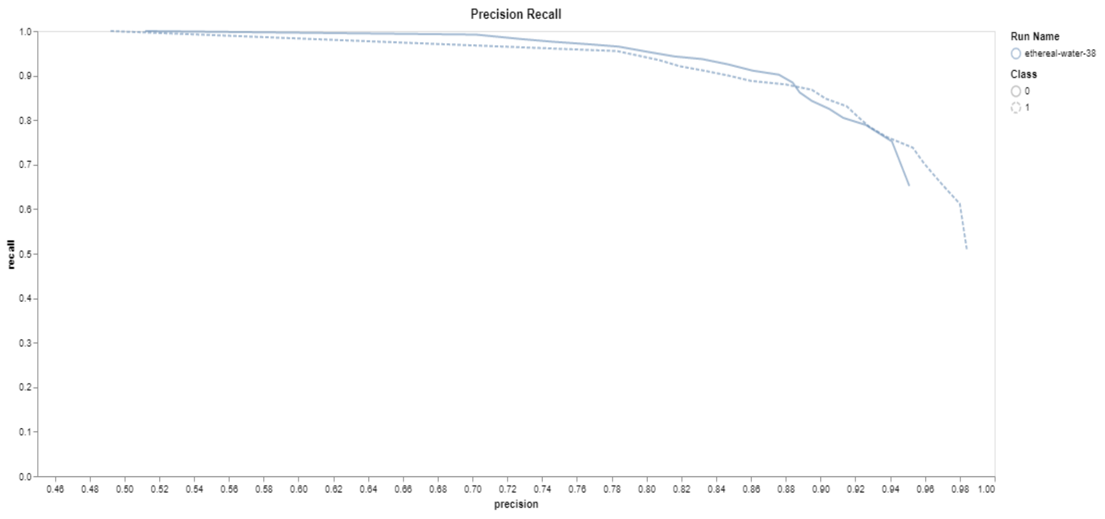


Figure 7