CSCI-5832: NLP Final Shared Task

Task 4 Subtask 1: Condescending/Patronizing Language Classification

Authors: Arvind Sreenivas, Dananjay Srinivas, Bhargav Shandilya

Abstract

This report explores different avenues to classify sentences into one of two categories as defined by the first subtask of Semeval task 4 (condescending vs. benign). We have used multiple deep learning architectures to perform this binary classification. For each approach, we describe the architecture, the vectorization process, and the final results that we obtained on our train-test split.

Background and System overview

Dataset used and nature of the data:
For this task we used the data from the practice splits provided to be able to compare our results with others on the leaderboard. The data was biased and needed downsampling which affected the outcome as data was lost in the process.

Computing resources:

- Google Colab GPU

Preprocessing steps, models used, why these models were used:

We imported the data into a DataFrame and one hot-encoded the columns to be used by the models. The data importing process mirrored the standard process followed by https://github.com/Perez-AlmendrosC/dontpatronizeme in their baseline random label assignment Notebook.

Models used:

- **LSTM**
- BERT
- RoBERTa Baseline
- XLNet

Preprocessing Steps:

The following preprocessing steps were used prior to training the LSTM model:

- The training data was downsampled to ensure that the data is not overly biased towards the labels from the higher class.
- Removal of special characters: We use the regex model to remove special characters such as '[/(){}\[\]\|@,;]'
- Removal of numbers, number-letter combinations, and certain symbols such as #, +, _
- Conversion of each paragraph to lowercase
- Removal of stopwords from each paragraph in the dataset: We download a standard set of stopwords obtained from the NLTK stopword corpus. Stopwords do not contribute to contextual information and add redundant information to the dataset.

Tokenization of Words: For the purpose of this project, we have used the tokenizer library provided by Keras. We have used a standard set of filters provided

External tools/libraries used:

- Pandas
- Simpletransformers
- Keras
- Tensorflow
- NLTK
- Numpy
- Sklearn

Experimental setup

Train-test-dev Split:
10% of the data is used for validation, 10% for testing, and 80% for training. The results shown below are based on the randomized train-test split that we performed and not the SemEval train-test split.

Hyperparameter Tuning:
- Number of epochs: 20
- Learning rate: 0.01 (Default)
- Batch Size: 32
- Loss: Categorical Cross-entropy

Evaluation Metrics: (Reporting for most recent LSTM model)
- Accuracy: 0.76
- Precision: 0.46
- Recall: 0.23
- F1: 0.30

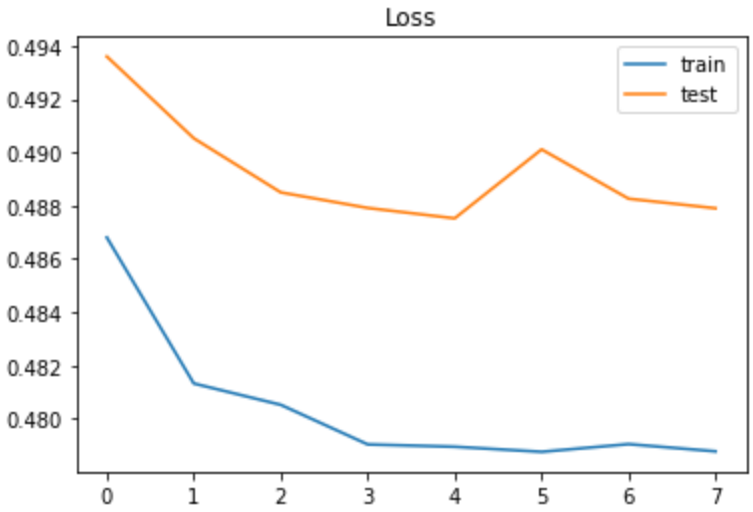
Model Architecture :

```
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_16 (Embedding)     (None, 250, 100)          5000000

 spatial_dropout1d_16 (Spati  (None, 250, 100)         0
 alDropout1D)

 lstm_16 (LSTM)               (None, 50)                30200

 dense_43 (Dense)             (None, 30)                1530

 dropout_8 (Dropout)          (None, 30)                0

 dense_44 (Dense)             (None, 10)                310

 dense_45 (Dense)             (None, 4)                 44

 dense_46 (Dense)             (None, 2)                 10

=================================================================
Total params: 5,032,094
Trainable params: 5,032,094
Non-trainable params: 0
```

Results

	Precision	Recall	Test Accuracy	F1
LSTM	**0.270270**	**0.452261**	**0.831901**	**0.338346**
BERT	0.66	0.09	0.64	0.16
RoBERTa	0.4466	0.6935	0.85	0.5433
XLNet	0.57	0.15	0.71	0.23

Loss is plotted against the number of epochs for the LSTM model as shown below:



- Main quantitative findings: According to the leaderboard on CodaLab, our RoBERTa system is ranked 7th based on the F1 score. (*sreenivasarvind*)

Results_Task1

#	User	Entries	Date of Last Entry	Precision ▲	Recall ▲	F1_Score ▲
1	**tianzhuanJAVA**	4	12/13/21	0.7841 (1)	0.8945 (1)	0.8357 (1)
2	**tushar-gautam**	8	12/13/21	0.6948 (2)	0.5377 (12)	0.6062 (2)
3	**Wanderer**	13	12/10/21	0.6548 (3)	0.5528 (11)	0.5995 (3)
4	valuable	19	12/10/21	0.5777 (6)	0.5980 (9)	0.5877 (4)
5	AliEdalat	9	12/13/21	0.5525 (7)	0.6080 (8)	0.5789 (5)
6	sapa4478	11	12/14/21	0.4533 (9)	0.6834 (7)	0.5451 (6)
7	sreenivasarvind	9	12/11/21	0.4466 (10)	0.6935 (5)	0.5433 (7)
8	sampugh	10	12/01/21	0.4612 (8)	0.5678 (10)	0.5090 (8)
9	BigTomato	7	12/12/21	0.3603 (14)	0.7387 (4)	0.4843 (9)
10	roberta-baseline	1	12/01/21	0.3499 (16)	0.7789 (3)	0.4829 (10)

Conclusion

The RoBERTa model performed the best and gave the best results in terms of precision, recall, and F1 score. However, the base RoBERTa model was used in the example code provided by SemEval, and we, therefore, attempted to build our own LSTM model to perform the task. The performance was notably worse, but the model itself was much lighter and quicker to train than RoBERTa, XLnet, and BERT.