CS410 – Course Final Project

Project Documentation

Sembian2@illinois.edu

Project Option 4: Competitions – Text Classification Competition

Team Name: Sembian2 (Individual)

Project Installation guide

The project code is completely executed in a google colab environment, please download the ipynb file and upload to google, you can also make a copy directly from the google colab link https://colab.research.google.com/drive/1gzwQJeSNKXuIljOX34z-quQePByhPt75?usp=sharing

```
Download the ipynb and load into google colab ( https://colab.research.google.com/)
and enable the GPU runtime
```

****Project Documentaiton:**** https://github.com/Sembian2-CS410Fall2020/CourseProject/blob/main/sembian2_cs410f2020_project_documentation.pdf

****Project Progress Report:**** https://github.com/Sembian2-CS410Fall2020/CourseProject/blob/main/sembian2_cs410f2020_project_progress_report.pdf

****Project Proposal Document:**** https://github.com/Sembian2-CS410Fall2020/CourseProject/blob/main/sembian2_cs410f2020_project_proposal.pdf

****Project Presentation Video**** (YouTube Link) : https://www.youtube.com/watch?v=cH2tZB5n_8Y

Motivation & Dataset:

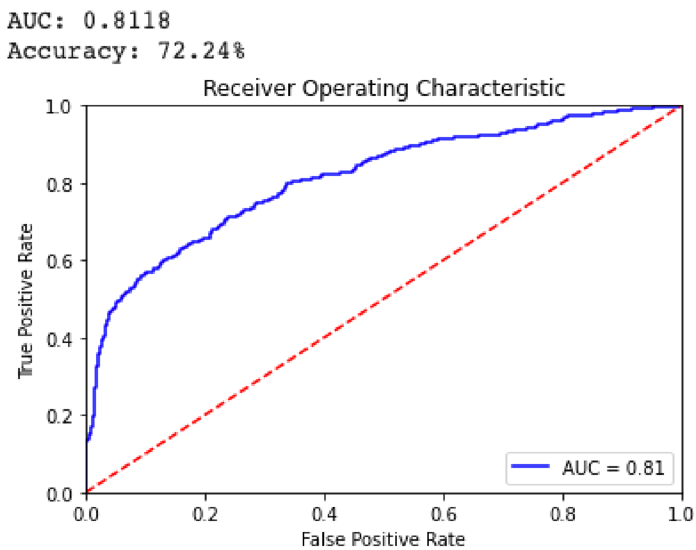
The text classification competition involves a binary classification of tweets with a balanced training including labels indicating SARCASM(0), NOT_SARCASM(1), I used the state-of-the-art Transformers, pytorch libraries with BERT Embeddings. The training dataset had 5000 labelled samples with balanced label distribution of 50 % and the test dataset had 1800 rows with additional id field.

Approach:

The training dataset is loaded into pandas DataFrame and the response column was pre-processed with text cleaning including removing @USER and @URL, expanding shortwords, expanding emoji's, and removing any stopwords using nltk library, removing special characters and punctuations. After pre-processing the response tweets column and label column is split into training and validation I used a .33% validation and .77% training data set.

Classification Methods:

The First approach is to use the Multinomial Naïve Bayes by applying TF-IDF and got a baseline AUC score of .8118 the accuracy was around 72% I used this as a baseline and tried improving the baseline using BERT embeddings and a feed forward neural network.



Text Classification with Transformers in PyTorch: BERT

The transformer-based LM(Language models) has shown promising progress on number of NLP benchmarks. By combining transfer learning methods with large-scale transformer language model is becoming a standard in modern NLP compared to traditional classification approaches. In this final approach to improve the baseline score of 72.24% from the MultinomialNB approach we will attempt to increase the accuracy score by implementing a transformer architecture and fine-tuning of the pre-trained BERT model for classification.

The two important complimentary concepts in Natural Language Processing:

- Word embeddings
- Language Model

Transformers are used to build the language model and embeddings can be retrieved as the by-product of pretraining. Transformers architecture implements so-called attention mechanism to include an entire sequence as a whole enabling training in parallel when compared to traditional LSTM approaches. The huggingface transformers library has a huge collection of the language models and embeddings and makes it easier for implementing using pytorch in python.

BERT

BERT(Bidirectional Encoder Representations from Transformers) is a mothod of pretraining language representation. BERT does not have a decoder but stacks 24 layer encoders for bert-uncased-large)

```
#Sample code showing the import and instantiation of BERT Model from transformers.
import torch
import torch.nn as nn
from transformers import BertModel
# Instantiate BERT model
self.bert = BertModel.from_pretrained('bert-large-uncased')
```

BERT Tokenizer and Netowrk Architecture

The important limitation of BERT is that the maximum sequence length is 512 tokens, the shorter sentences are added with [PAD] and there is also a [CLS] token for indicating beginning of the sentence and [SEP] token at the end of sentence the tokenized sentence is then encoded using BERT Embeddings the bert-large has 1024 embeddings

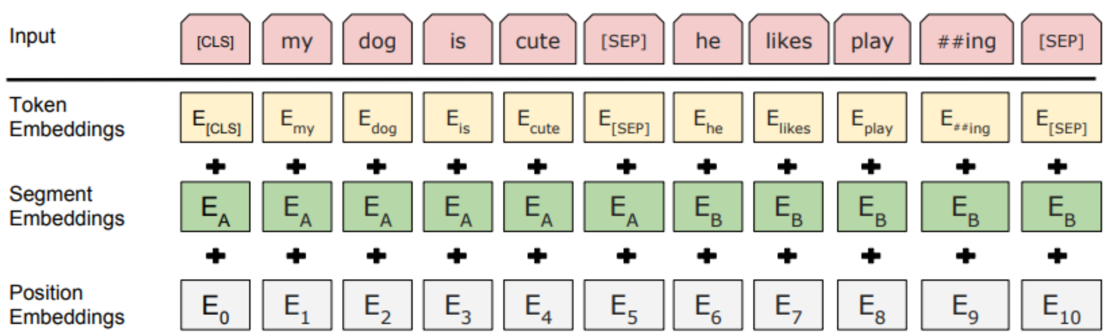
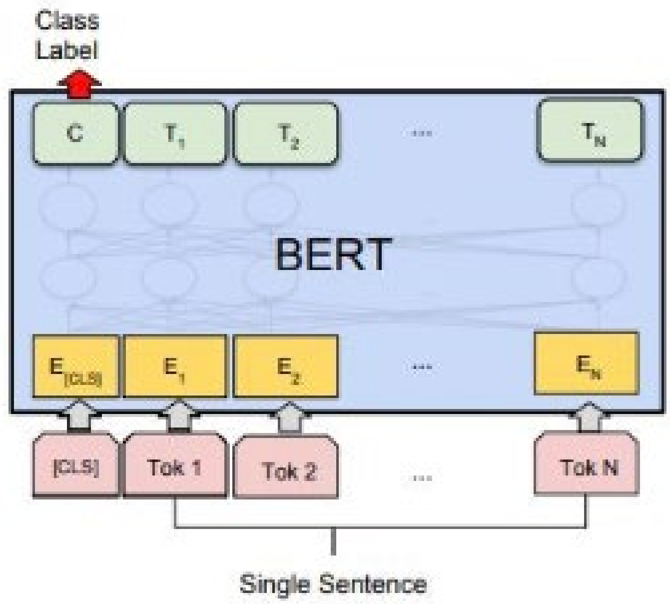


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.



(b) Single Sentence Classification Tasks:
SST-2, CoLA

While there are multiple approaches I used a custom BertClassifier with a single feedforward neural network with

```
# Specify hidden size of BERT, hidden size of our classifier, and number of labels
        # BERT-Large, Uncased: 24-layer, 1024-hidden, 16-heads, 340M parameters
        D_in, H, D_out = 1024, 50, 2

# Instantiate an one-layer feed-forward classifier
        self.classifier = nn.Sequential(
            nn.Linear(D_in, H),
#https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear
            nn.ReLU(),
#https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html#torch.nn.ReLU
            nn.Linear(H, D_out),
#https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear
           )
```

The final layer out put is passed thru a ReLU activation layer and output dimensions of 2 indicating the 2 labels[SARCASM-0, NOT_SARCASM-1] , the BERT tokenizer is applied on all responses of the training data and map tokens into WordPiece embeddings using the encode_plus function, the following parameters were used for training.

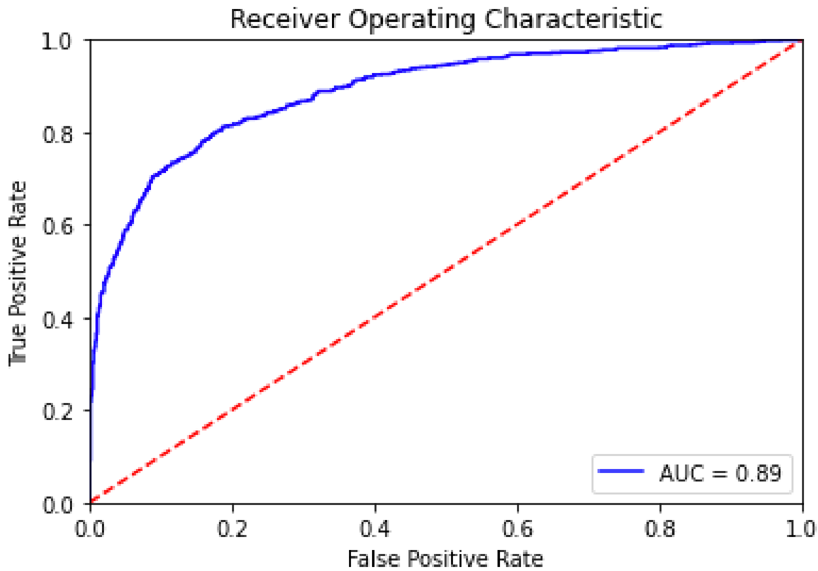
LearningRate	5e-5
Max Sequence Length	89
Batch Size	32

No. of Epochs	4

The model is then trained for 4 epochs and achieved a score of 81.17% on the training set that is almost 10 point increase from the baseline MultiNomialNB model.

```
Start training...

 Epoch  |  Batch  |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
----------------------------------------------------------------------
   1    |   20    |   0.664404   |     -      |     -     |   31.78
   1    |   40    |   0.534449   |     -      |     -     |   30.99
   1    |   60    |   0.533544   |     -      |     -     |   31.70
   1    |   80    |   0.500559   |     -      |     -     |   32.19
   1    |   100   |   0.479444   |     -      |     -     |   32.82
   1    |   104   |   0.415712   |     -      |     -     |   6.14
----------------------------------------------------------------------
   1    |    -    |   0.538812   |  0.444033  |   79.88   |  196.45
----------------------------------------------------------------------


 Epoch  |  Batch  |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
----------------------------------------------------------------------
   2    |   20    |   0.291500   |     -      |     -     |   35.13
   2    |   40    |   0.280435   |     -      |     -     |   33.84
   2    |   60    |   0.300033   |     -      |     -     |   34.14
   2    |   80    |   0.294220   |     -      |     -     |   34.19
   2    |   100   |   0.264788   |     -      |     -     |   34.18
   2    |   104   |   0.178849   |     -      |     -     |   6.32
----------------------------------------------------------------------
   2    |    -    |   0.282156   |  0.582648  |   78.42   |  209.20
----------------------------------------------------------------------


 Epoch  |  Batch  |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
----------------------------------------------------------------------
   3    |   20    |   0.178021   |     -      |     -     |   35.83
   3    |   40    |   0.084245   |     -      |     -     |   34.24
   3    |   60    |   0.126948   |     -      |     -     |   34.21
   3    |   80    |   0.113072   |     -      |     -     |   34.07
   3    |   100   |   0.097395   |     -      |     -     |   34.11
   3    |   104   |   0.030649   |     -      |     -     |   6.31
----------------------------------------------------------------------
   3    |    -    |   0.117088   |  0.745395  |   79.05   |  210.33
----------------------------------------------------------------------


 Epoch  |  Batch  |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
----------------------------------------------------------------------
   4    |   20    |   0.032811   |     -      |     -     |   35.79
   4    |   40    |   0.025885   |     -      |     -     |   34.05
   4    |   60    |   0.056671   |     -      |     -     |   34.21
   4    |   80    |   0.051280   |     -      |     -     |   34.09
   4    |   100   |   0.024765   |     -      |     -     |   34.04
   4    |   104   |   0.079427   |     -      |     -     |   6.33
----------------------------------------------------------------------
   4    |    -    |   0.039798   |  0.846048  |   81.17   |  210.02
----------------------------------------------------------------------
```

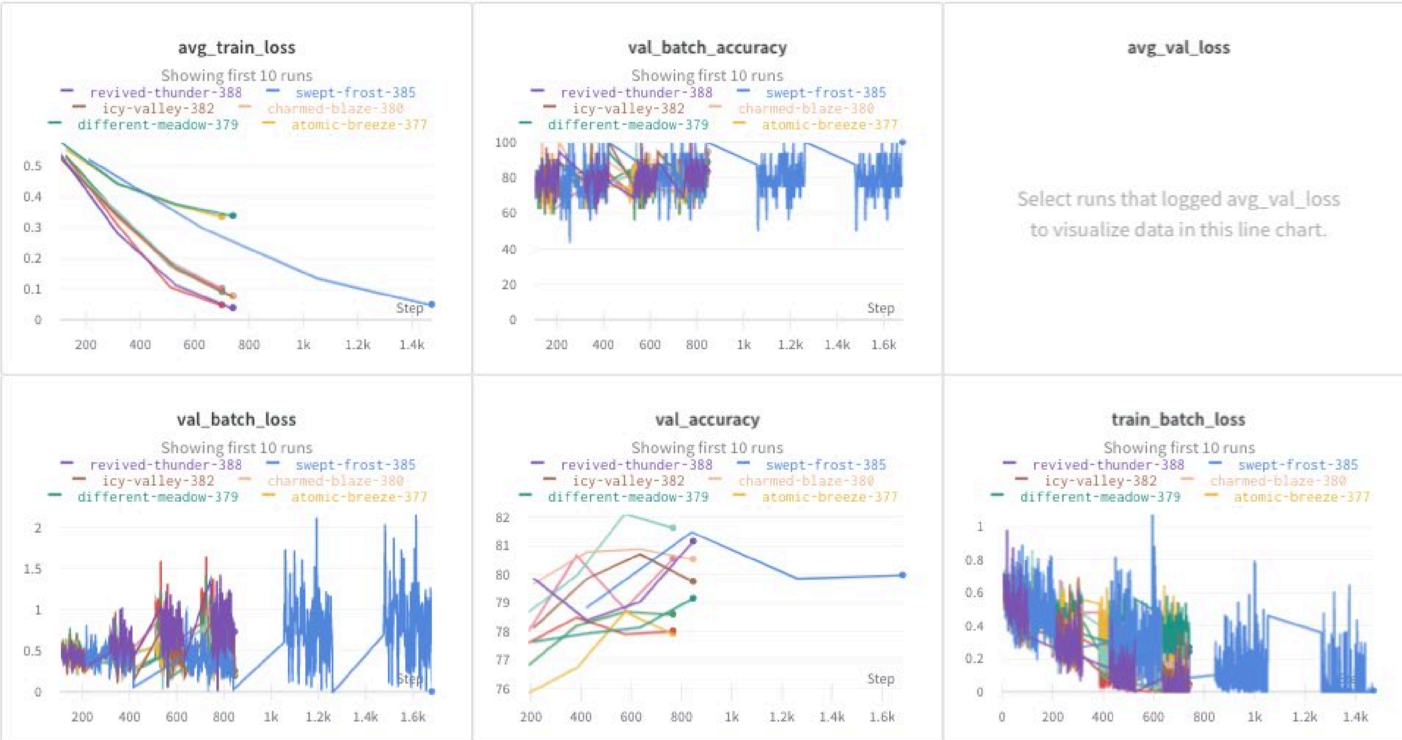
```
AUC: 0.8887
Accuracy: 81.15%
```



For HyperParameter tuning I used wandb.com (weights and Biases) to report out the various runs and compared the best score and the run named revivedpthunder-388 scored the highest and achieved a 81.17 validation accuracy and 75.19 Test Accuracy in the leaderboard

[Report Link to wandb]

Training and Validation Reports HyperParameter Tuning



Run set 12

Name (12 visualized)	train_batch_loss	avg_train_loss	val_accuracy	val_loss	State	Notes	User	Tags
revived-thunder-388	0.2708	0.0398	81.17	0.846	finished	Course ...	balasul	baselin
swept-frost-385	0.007672	0.05055	79.988	0.8483	finished	Course ...	balasul	baselin
icy-valley-382	0.04722	0.07824	79.761	0.6482	finished	Course ...	balasul	baselin
charmed-blaze-380	0.02918	0.07794	80.542	0.6727	finished	Course ...	balasul	baselin
different-meadow-379	0.243	0.3388	79.173	0.4438	finished	Course ...	balasul	baselin
atomic-breeze-377	0.2203	0.3355	77.93	0.4644	finished	Course ...	balasul	baselin
fine-dawn-375	0.01691	0.09295	81.641	0.6235	finished	Course ...	balasul	baselin
fallen-lion-373	0.02623	0.1033	80.566	0.6269	finished	Course ...	balasul	baselin
morning-plasma-371	0.09982	0.09411	78.613	0.7194	finished	Course ...	balasul	baselin
true-star-370	0.06728	0.04901	78.027	0.906	finished	Course ...	balasul	baselin
upbeat-surf-368	0.007758	0.05253	79.59	0.8318	finished	Course ...	balasul	baselin
desert-river-366	0.01703	0.05498	81.348	0.7513	finished	Course ...	balasul	baselin

1-12 of 12

Tables	Training Accuracy
TF-IDF Vectorizer and Multinomial Naïve Bayes	72.24%
Transformers with BERT_large_uncased embeddings	**81.17%**

Test Accuracy of 75.18% - Position 10 on Leaderboard as of 12.02.2020

Data Livelab   Home   Projects   Courses   Manage   Create   Delete Linked Accounts   Log out

Leaderboard ID: 5f83d14b872c465d24df8b08

Rank	Username	Submission Number	precision	recall	f1	completed
1	zwe	9	0.7470899470899471	0.7844444444444445	0.7653116531165312	1
2	anil4u228	22	0.6988062442607897	0.8455555555555555	0.7652086475615888	1
3	cheny9	2	0.7069943289224953	0.8311111111111111	0.7640449438202248	1
4	ajjain	7	0.7232767232767233	0.8044444444444444	0.7617043661230932	1
5	Artsiom Strok	8	0.6918181818181818	0.8455555555555555	0.7609999999999999	1
6	dheeraj.patta	3	0.6918181818181818	0.8455555555555555	0.7609999999999999	1
7	Zinkoy	73	0.6723549488054608	0.8755555555555555	0.7606177606177605	1
8	zainalh2	22	0.6823843416370107	0.8522222222222222	0.757905138339921	1
9	reckoner	3	0.7382978723404255	0.7711111111111111	0.7543478260869564	1
10	Sembian	8	0.7082514734774067	0.8011111111111111	0.7518248175182481	1
11	Edward Ma	12	0.6872659176029963	0.8155555555555556	0.7459349593495934	1
12	ryotakaki	3	0.7116751269035533	0.7788888888888889	0.7437665782493369	1
13	thecheebo	13	0.7360350492880613	0.7466666666666667	0.7413127413127414	1
14	yeowlong	15	0.6241352805534205	0.9022222222222223	0.7378464334393458	1
15	zy23	32	0.6237471087124132	0.8988888888888888	0.7364588074647246	1
16	wenxif2	94	0.7252155172413793	0.7477777777777778	0.7363238512035012	1
17	samarth.keshari	81	0.6227867590454196	0.8988888888888888	0.7357889949977261	1
18	jzheng5	4	0.7384960718294051	0.7311111111111112	0.7347850362925739	1
19	jsun	11	0.6980942828485457	0.7733333333333333	0.7337901950448075	1
20	jongwoo Jeon	8	0.614403600900225	0.91	0.7335423197492162	1

Using the Transformers, Pytorch and BERT Classification model I was able to beat the baseline score on the leaderboard and improved the score by repeating the training with Hyper Parameter tuning and text pre-processing techniques and achieved a score of 75.18% Test Accuracy, and have no challenges.

References:

- Images for illustration are taken from the original BERT paper (Devlin et al. 2018).
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova,Bert: Pre-training of deep bidirectional transformers for languageunderstanding, 2018.
- Weights and Biases: https://wandb.ai/cayush/bert-finetuning/reports/Sentence-Classification-With-Huggingface-BERT-and-W-B--Vmlldzo4MDMwNA
- The Role of Conversation Context for Sarcasm Detection in Online Interactions: https://arxiv.org/pdf/1707.06226.pdf
- Clark, Kevin, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. "What does bert look at? an analysis of bert's attention." arXiv preprint arXiv:1906.04341 (2019).
- Sileo, Damien. "Understanding bert transformer: Attention isnt all you need." Towards Data Science (2019).
- Weiss, Karl, Taghi M. Khoshgoftaar, and DingDing Wang. "A survey of transfer learning." Journal of Big data 3, no. 1 (2016): 9.
- Biewald, Lukas. "Experiment Tracking with Weights and Biases." (2020).
- Text Tweet Pre processing - https://github.com/digitalepidemiologylab
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) - https://jalammar.github.io/illustrated-bert/
- Pytorch.org - https://pytorch.org/docs/stable/index.html
- BERT Fine-Tuning Tutorial with PyTorch for Text Classification: https://medium.com/@aniruddha.choudhury94/part-2-bert-fine-tuning-tutorial-with-pytorch-for-text-classification-on-the-corpus-of-linguistic-18057ce330e1
- Huggingface transformers library - https://github.com/huggingface/transformers
- BERT for Advance NLP with Transformers in Pytorch - https://www.linkedin.com/pulse/part1-bert-advance-nlp-transformers-pytorch-aniruddha-choudhury/
- Attention Is All You Need; Vaswani et al., 2017. - https://arxiv.org/pdf/1706.03762.pdf
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding; Devlin et al., 2018. - https://arxiv.org/pdf/1810.04805.pdf
- Encoder-Decoder Architecture& Bert Paperon the full research. - https://arxiv.org/pdf/1810.04805.pdf

- Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). [Tackling the poor assumptions of naive bayes text classifiers. In ICML](https://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf) (Vol. 3, pp. 616-623).
- Text classification with transformers in Tensorflow 2: BERT, XLNet - https://atheros.ai/blog/text-classification-with-transformers-in-tensorflow-2