Artificial Intelligence II Homework 4

Comments & Model Performance results on Question 1

Pavlos Spanoudakis (sdi1800184)

Basic Execution flow

1. We read the train & validation sets data from the input files, into `DataFrame`'s. The file paths can be modified on the notebook code cell #3. We also check whether all the samples have the expected format without missing values.

2. We create the Train and Validation sets, using `TweetDataset` objects. We tokenize each sample using `BertTokenizer`, for the pretrained `bert-base-uncased` model.

3. We will access the two sets in batches, using `DataLoader` objects.

4. We initialize the Model. Hyperparameters such as number of epochs, batch size, learning rate & Dropout probability can be modified on code cell #5. We use `CrossEntropyLoss` and the `Adam` optimizer.

5. We train the model: We use `lists` to store several performance stats during training, such as Loss and Accuracy on Train and Validation sets after each epoch.
   During each epoch:

   - For each batch given by the train set `BucketIterator`:
     - We make predictions on this batch
     - Extract the predicted labels & calculate the accuracy
     - Calculate & store the batch Loss
     - Perform backpropagation
   - After going through all the batches, we calculate the total accuracy & total Loss for the Train set in the epoch.
   - We perform the same actions on the validation set, this time without performing backpropagation of course.
   - During the final epoch, we store the predicted labels, as well as the model output (on both sets), to use in the final evaluation phase right after.

6. Displaying performance results. After the end of training we display:

   - The Confusion Matrices of the final model predictions on the Train and Validation sets.
   - The F1, Precision and Recall scores on each class, for the final model (on both sets).
     We use `precision_recall_fscore_support` routine from **scikit-learn** to calculate the scores.
   - The ROC Curves for the Validation set predictions.
     We mirror the usage of `roc_curve` from **scikit-learn** for multiple classes, as demonstrated here.
     To create the curves, we apply the `softmax` function to the NN output vector, to convert it to possibility values that add-up to 1, and use the `softmax` output to create the curves.
     `roc_curve` applies generated possibility thresholds to create the curves, therefore if we provided

it with just the predicted labels, it would only apply 3 thresholds to each result, which is insufficient to create useful ROC curves.

Model Architecture

The model has a simple architecture. It is implemented in the `BertTweetClassifier` class:

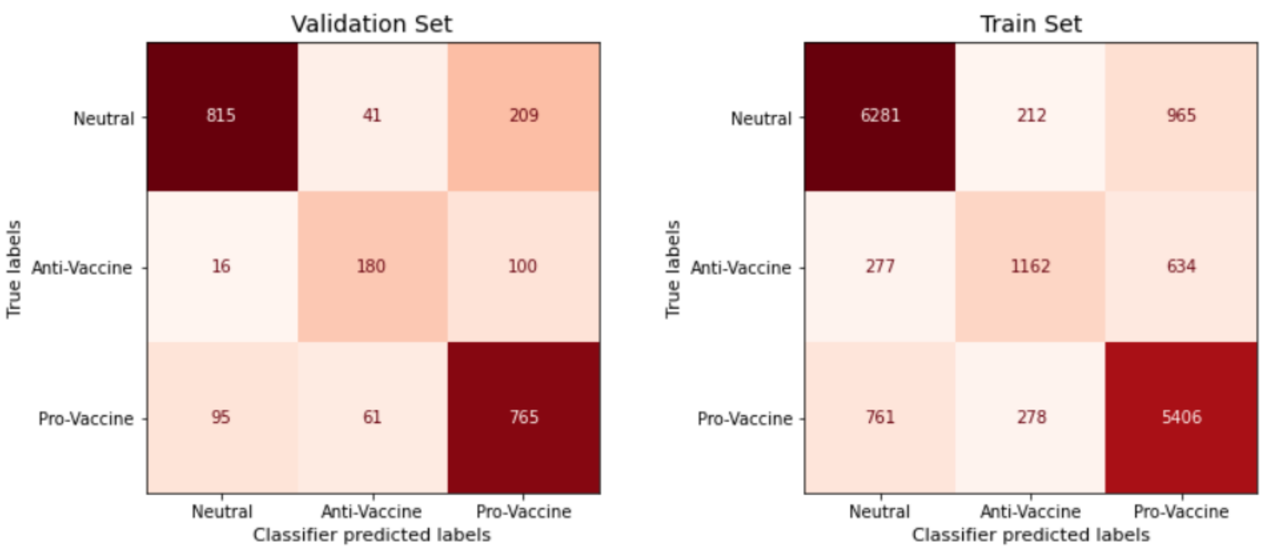
- We start with the `bert-base-uncased` pretrained model, getting the `CLS` size 768 output.
- We apply a dropout layer on the BERT output.
- We then apply a `Linear` layer to produce a size 3 output.
- Finally, we apply a `ReLU` layer and return the result.

Different models performance comparison

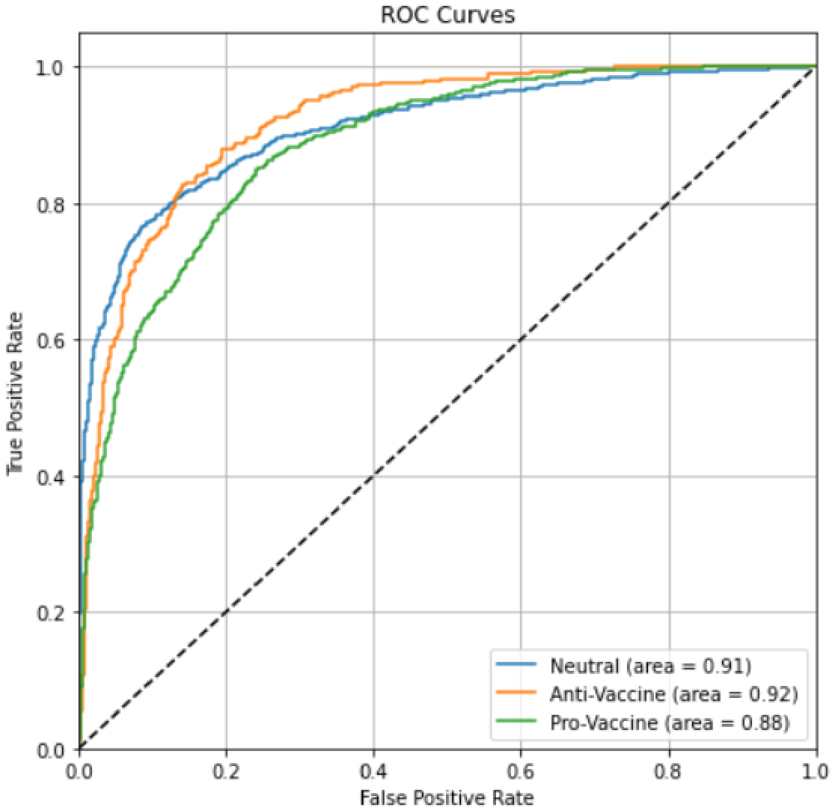
Notes on all models:

- The performance results displayed below have been produced using `SEED = 256` for random generators.
- Unfortunately, GPU-accelerated model results could not be fully reproduced with certainty.
- We use `MAX_LENGTH = 100` in all models.
- The execution of each model takes about 4-5 minutes.

1. This is the preselected model in the interactive notebook. For this model, we use:
    - Learning rate: 3e-5
    - Batch Size: 64
    - # Epochs: 2
    - Dropout probability: 0.2
    - Execution Time: < 20 minutes

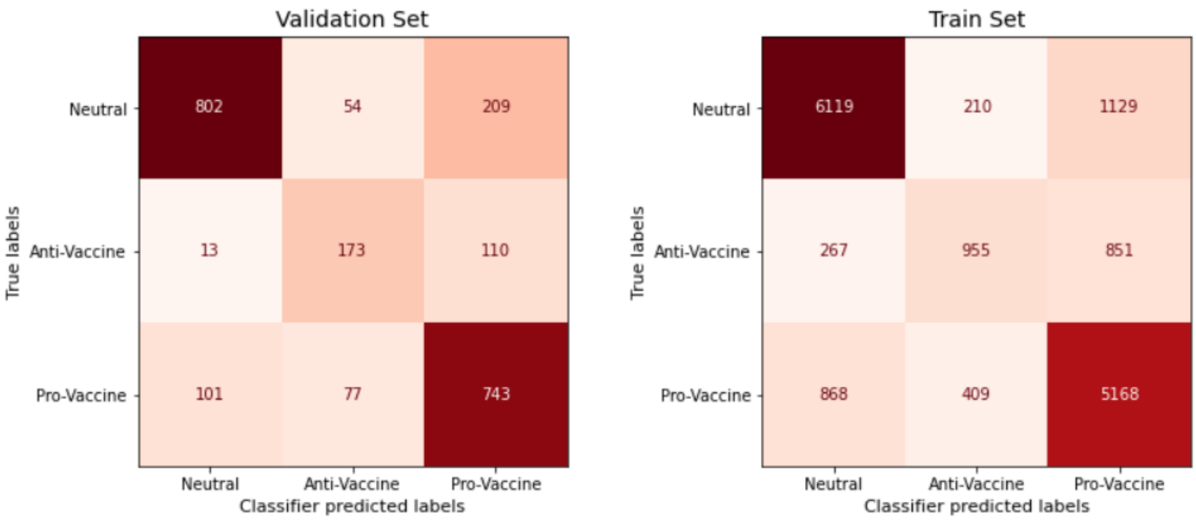


	Train Set			Validation Set		
	Precision	Recall	F1	Precision	Recall	F1
Neutral	0.8422	0.5605	0.8388	0.7653	0.6081	0.8306
Anti-Vaccine	0.8582	0.7034	0.7717	0.8801	0.6383	0.7123
Pro-Vaccine	0.8501	0.6239	0.8039	0.8187	0.6228	0.7669

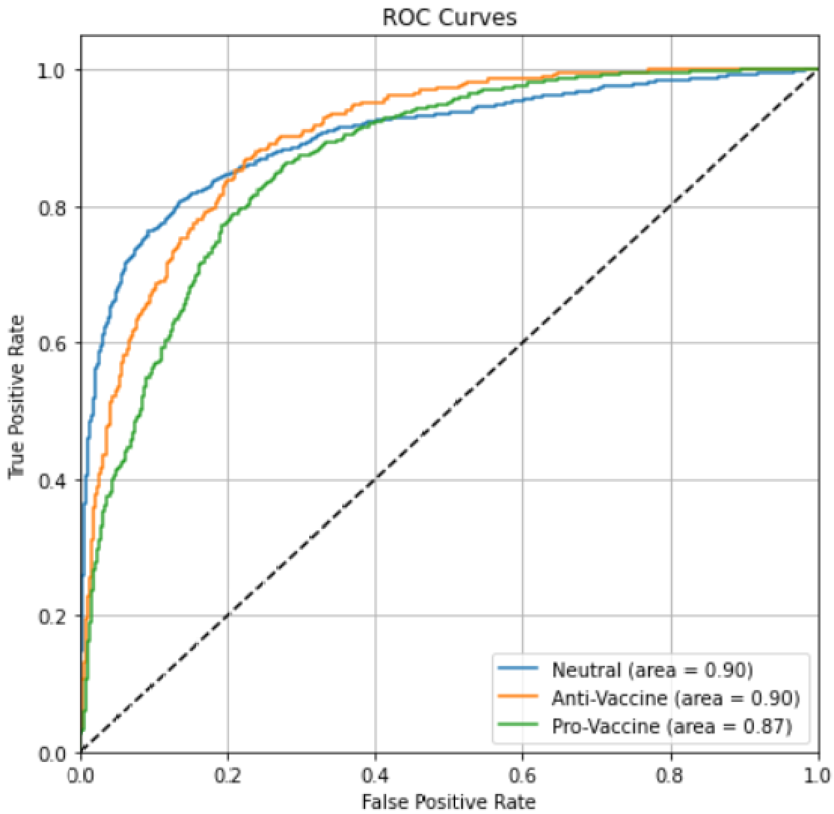


2. In this model, we use:

   - Learning rate: 5e-6
   - Batch Size: 32
   - # Epochs: 3
   - Dropout probability: 0.25
   - Execution Time: < 15 minutes

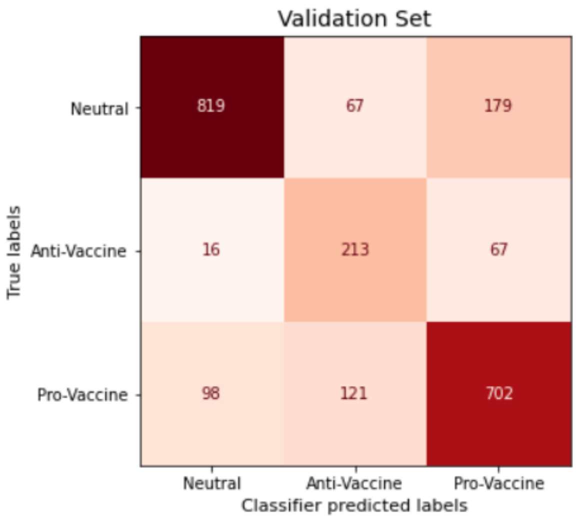


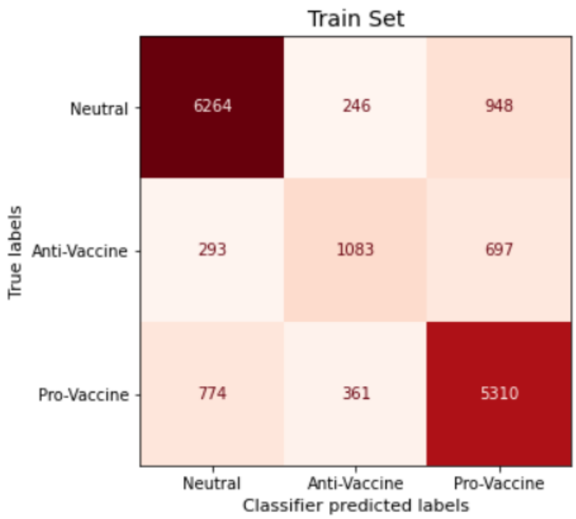
	Train Set			Validation Set		
	Precision	Recall	F1	Precision	Recall	F1
Neutral	0.8205	0.4607	0.8019	0.7531	0.5845	0.8067
Anti-Vaccine	0.8435	0.6067	0.7230	0.8755	0.5691	0.6996
Pro-Vaccine	0.8318	0.5237	0.7604	0.8097	0.5767	0.7494



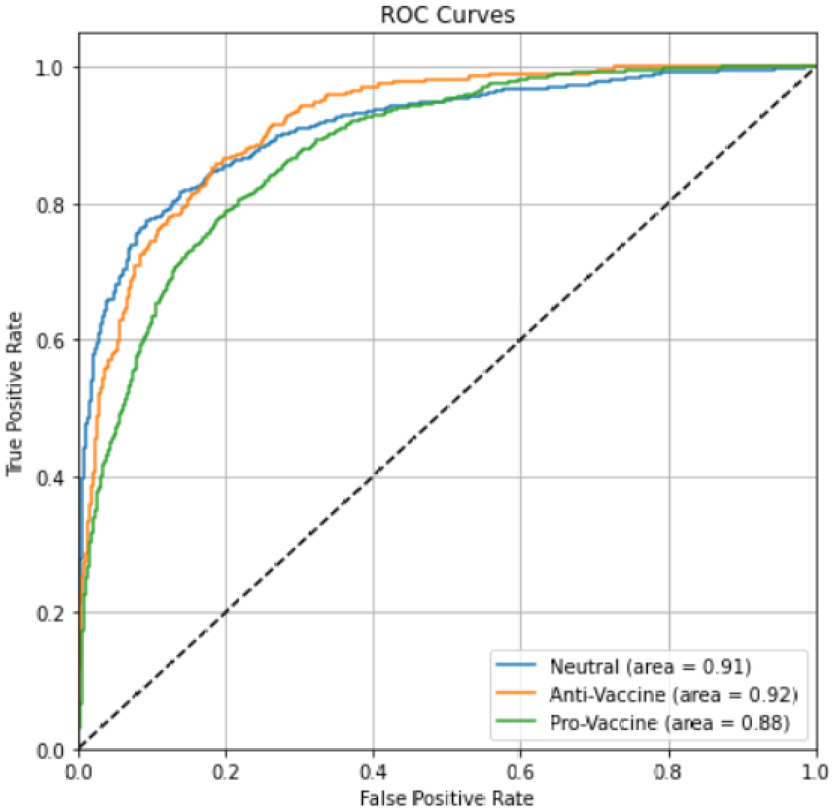
3. In this model, we use:

- Learning rate: 1e-5
- Batch Size: 16
- # Epochs: 2
- Dropout probability: 0.1
- Execution Time: < 20 minutes





	Train Set			Validation Set		
	Precision	Recall	F1	Precision	Recall	F1
Neutral	0.8399	0.5224	0.8239	0.7690	0.7196	0.7622
Anti-Vaccine	0.8545	0.6408	0.7635	0.8778	0.5312	0.7405
Pro-Vaccine	0.8471	0.5756	0.7925	0.8198	0.6112	0.7512



Comments/Observations on the models and their develpoment

- Model hyperparameters are mostly based on/influenced by the BERT authors recommendations:
  - Batch Sizes: 8, 16, 32, 64, 128: Training with Batch Size = 8 caused overfitting in most cases. Training with Batch Size = 128 caused `CUDA: out of memory` errors in Google Colab.
  - Learning Rates: 3e-4, 1e-4, 5e-5, 3e-5: The Learning Rates of the presented models are a result of fine-tuning, tweaking and experimenting with the recommended ones.
  - Number of Epochs: 4 In our case, the experiments showed that 2-3 epochs are more suitable.
- MAX_LENGTH = 100 appeared to be enough for decent results, without increasing the execution time dramatically.
- In comparison with the previous projects models, we notice that BERT model can achieve decent performance even with huge class imbalance regarding "Anti-Vaccine" tweets.
- The performance across all classes is improved, as we can notice in the ROC Curves, where the ROC area for each class in increased from the previous models.

Development

- The notebook has been developed in Google Colab and was based on the Projects 2 & 3 notebooks.

- It has been tested successfully in Google Colab & Kaggle environments, using GPU-accelerated runtimes.
    - **Note**: In Kaggle, Confusion Matrices code crashes because `ConfusionMatrixDisplay.from_predictions` appears to be unavailable.