Brendan Magdamo, Brendan Manning, Stephanie Scheerer
Projects in Data Science
5 April, 2021

3.3 - Model Report

Introduction

Phase 2 was set aside as the primary time to train our model. One might recall that in Phase 1, "We initially created a baseline model that did not involve any real Natural Language Processing and instead used a count vectorizer to represent the Tweets. This gave us a baseline F1 score of between 0.73 and 0.77 depending on where the data was split."[1] In this phase, we switched to a Simple Transformers ClassificationModel with inputs from a pre-trained RoBERTa model called "roberta-base". This is the base version of the RoBERTa model that is case sensitive. This improved our performance up to approximately 0.81 on the testing dataset we set aside and 0.81857 in our first submission to Kaggle.

We believe that we are, for all intents and purposes, finished training the model. The only thing left to do is include the tuned hyperparameters in this model. For Phase 3, we simply have to feed in a new set of data to the existing model in Brendan Magdamo's notebook on GitHub. This model will use the hyperparameters decided by the tuning job that is running as this document gets written. Our new data which was collected during Phase 2 will serve as the test dataset that will be used for the final model predictions and the model dashboard.

Which feature engineering techniques were used? Justify your choice and describe the working of the techniques in the context of your project goals.

String Sanitization

We began our model training by doing the basics. We removed urls, html, and punctuation from the input tweets. Initially we considered keeping some of the punctuation with the reasoning that the hashtag symbol may have been a useful indicator. However, it was ultimately determined that this negatively impacted model results so all punctuation was removed. We also lowercased the text. We concluded that no meaningful information would be found in the capitalization of different words because randomly collected text data, particularly Twitter data, is not often subject to traditional grammatical conventions. Hence, the pre-trained RoBERTa model may take into account a word capitalization that is not actually grammatically correct and therefore may convey an alternate meaning to the one the Twitter user intended. Finally, and after much deliberation, we chose to preserve emojis in the text. In PD 3.2, we documented how exploratory

[1] See PD 2.1A

data analysis revealed that the relative distributions of common emojis like 😂, 😭, and 🔥 were different between tweets for real and "false alarm" disaster tweets. Therefore, they were deemed to have some predictive power in the model output and so we chose to include them. Initially we thought that we may have needed to create an encoder for this. However, modern language models have built-in emoji parsing capabilities, so there was no need on our part to create some sort of word representation or encoding, as the model could read and understood the UTF encoding for the different emojis.

Abbreviations

Rather than ask the model to learn two ways of expressing the same sentiment (the words and the abbreviation), we replaced a hand-curated dictionary of common abbreviations with the words themselves. We arrived at the list by searching for commonly used abbreviations online. We found a Python dictionary that had a plethora of terms that people would be likely to use in an online setting. This dictionary of abbreviations may be less useful for other types of text data, but it worked very well with our Twitter data. The replacement of abbreviations, in conjunction with also removing all non-ASCII characters, increased our F1 score from 0.81857 all the way to 0.83512. This was a significant improvement so we would recommend utilizing the same Python dictionary for similar tasks that involve Twitter text data as it strongly impacted the outcome.

Back Translation

Back Translation can be an effective way to reduce overfitting. It's similar to how in image-based learning one might grow the size of their database by including flipped, filtered, or otherwise processed versions of the source images. However in the context of textual data, the idea is to translate back and forth between different languages. This is done by translating the input strings from English to some other language and then back into English. The resulting translation will be slightly different than the original, giving us a little bit of extra information that can be useful for creating the model and improving performance.

In our case, we used the Helsinki-NLP module to translate to French and then back again. While our algorithm worked for small batches, it eventually led to a GPU memory error when applied to the full dataset. Alternatively, we used Excel's builtin translate functionality to translate train.csv and test.csv, then concatenated the translated files with the originals. Unfortunately, the quality of the translations provided by Excel's translate functions were poor and our F1 score in Kaggle decreased from 0.83 to 0.825. If we had more time, we would have found a solution to the Helsinki-NLP model memory problem. We also would have treated the language as a sort of "hyperparameter" and tried it again with (a) other Romance languages similar to English (like Spanish, Italian, etc.) and (b) languages very unlike English (Latin, Chinese, Arabic, etc). Unfortunately, due to time constraints and the decision to spend the majority of Phase 3 working on the creation of the final dashboard to demonstrate the model's usefulness, the back translation was discarded.

Which modeling algorithms were used? Justify your choice and describe the working of the algorithms in the context of your project goals.

Modeling Algorithms

We used Simple Transformers for all of our Phase 2 model training because of its simple interface. It also included an easy way to ensure training with CUDA support (by passing `use_cuda=True` to the ClassificationArgs constructor). We wanted to try using both RoBERTa and DistilBERT as we spent a fair amount of time researching the BERT model and knew it was state of the art in terms of NLP. Based upon our research and our experiences in the process of working to get a BERT-based model running, we decided to go with the RoBERTa and DistiliBERT variations of BERT.

- RoBERTa was appealing because it "removes BERT's next-sentence pretraining objective," which is helpful because tweets are more like single sentences[2] than long paragraphs. This model was run in Google Colab because difficulties arose with setting up a container in Temple's HPC GPU cluster.
- DistilBERT is a simpler form of BERT and "has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark" (see source). We were curious how big the tradeoff between training time and performance would be with this model. In the initial stages of the project, the DistilBERT model was taking a long time to run even in Temple's HPC GPU cluster, so in order to speed up feature extraction, the algorithm was ran once with the feature weights saved to a text file so that more time could be spent examining hyperparameter optimization. Ultimately though, the tradeoff between performance and training time for DistiliBERT wasn't too heavily explored as the RoBERTa model, which used Google Colab, was selected for final use instead of DistilliBERT so the challenges with running it on Temple's GPU resources were able to be avoided.

Brendan Magdamo created the RoBERTa model and Brendan Manning created a DistilBERT model. Through a lengthy process of adjustments, the RoBERTa model eventually achieved an F1 score of 0.835 while the DistilBERT one achieved an F1 score of approximately 0.78. Accordingly, the RoBERTa model was chosen for use in the final phase of the project and will be used for the creation of the final dashboard illustrating the model's performance.

[2] This is not to say that tweets are "sentences" in the usual terms, of course!

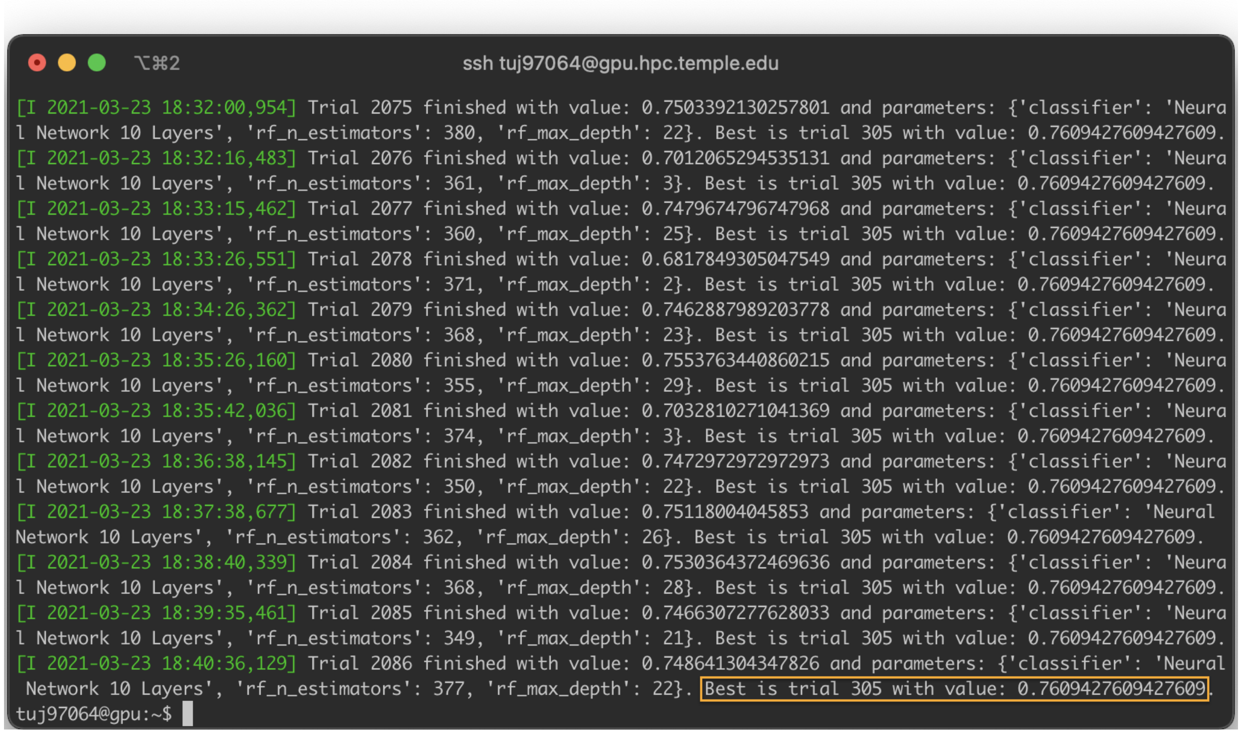
Hyperparameter Tuning

We used Optuna and Weights and Biases hyperparameter tuning to improve our model's performance and outcome.

When using wandb for hyperparameter tuning, we create our *ClassificationArgs* with `wandb_project="DisasterTweets"` so that we can track the progress of the wandb sweep online. Optuna hyperparameter tuning does not require this additional syntax.

Using Optuna, we noticed that hyperparameters tended to converge to the optimal values very quickly. When tuning a 10-layer neural network,[3] the optimal value was found after 305/10,000 iterations. Similarly when tuning a Logistic Regression model, it converged after 218 and 585 iterations in two separate trials. The maximum F1 score produced from these trials on Brendan Manning's models (which were different from Brendan Magadmo's and *not* the models ultimately chosen for Phase 3) were 0.761 and 0.773 respectively. What this *did* show us is that while the tested model types all performed similarly, Linear Regression was indeed a slightly better choice. See figures 1 and 2.

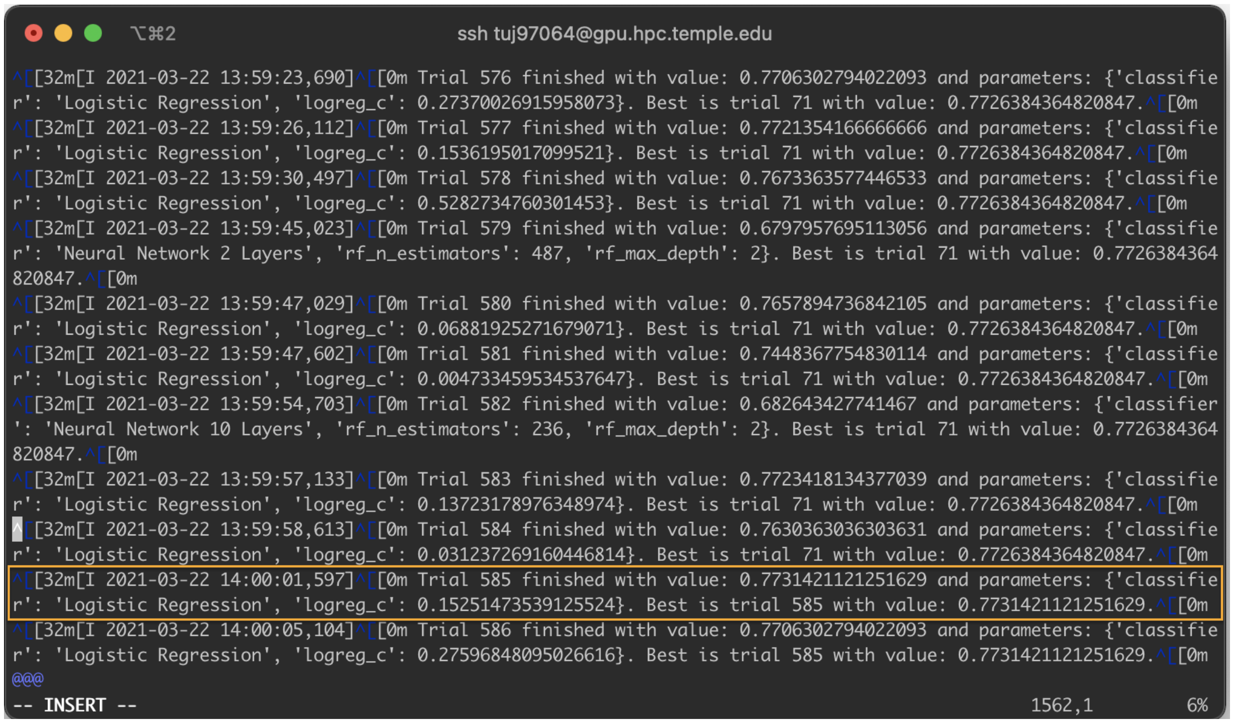
Figure1:



```
[I 2021-03-23 18:32:00,954] Trial 2075 finished with value: 0.7503392130257801 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 380, 'rf_max_depth': 22}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:32:16,483] Trial 2076 finished with value: 0.7012065294535131 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 361, 'rf_max_depth': 3}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:33:15,462] Trial 2077 finished with value: 0.7479674796747968 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 360, 'rf_max_depth': 25}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:33:26,551] Trial 2078 finished with value: 0.6817849305047549 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 371, 'rf_max_depth': 2}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:34:26,362] Trial 2079 finished with value: 0.7462887989203778 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 368, 'rf_max_depth': 23}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:35:26,160] Trial 2080 finished with value: 0.7553763440860215 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 355, 'rf_max_depth': 29}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:35:42,036] Trial 2081 finished with value: 0.7032810271041369 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 374, 'rf_max_depth': 3}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:36:38,145] Trial 2082 finished with value: 0.7472972972972973 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 350, 'rf_max_depth': 22}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:37:38,677] Trial 2083 finished with value: 0.75118004045853 and parameters: {'classifier': 'Neural
Network 10 Layers', 'rf_n_estimators': 362, 'rf_max_depth': 26}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:38:40,339] Trial 2084 finished with value: 0.7530364372469636 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 368, 'rf_max_depth': 28}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:39:35,461] Trial 2085 finished with value: 0.7466307277628033 and parameters: {'classifier': 'Neura
l Network 10 Layers', 'rf_n_estimators': 349, 'rf_max_depth': 21}. Best is trial 305 with value: 0.7609427609427609.
[I 2021-03-23 18:40:36,129] Trial 2086 finished with value: 0.748641304347826 and parameters: {'classifier': 'Neural
 Network 10 Layers', 'rf_n_estimators': 377, 'rf_max_depth': 22}. Best is trial 305 with value: 0.7609427609427609.
tuj97064@gpu:~$
```

[3] Do note that we also tried a Neural Network with fewer than 10 layers, but had less success with that.

Figure 2:

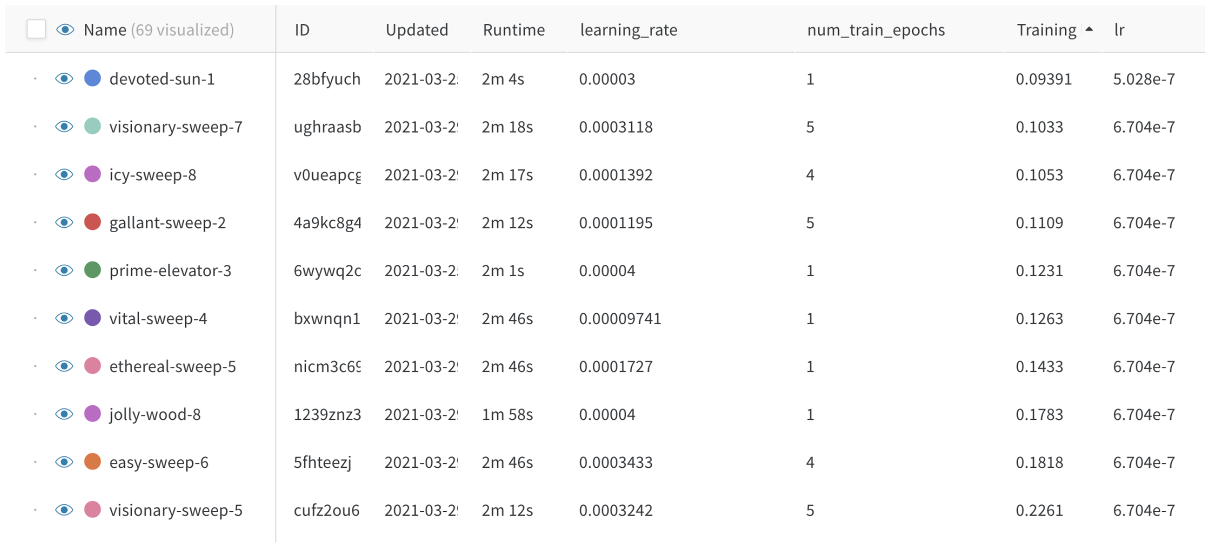


```
^[[32m[I 2021-03-22 13:59:23,690]^[[0m Trial 576 finished with value: 0.7706302794022093 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.27370026915958073}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 13:59:26,112]^[[0m Trial 577 finished with value: 0.7721354166666666 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.1536195017099521}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 13:59:30,497]^[[0m Trial 578 finished with value: 0.7673363577446533 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.5282734760301453}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 13:59:45,023]^[[0m Trial 579 finished with value: 0.6797957695113056 and parameters: {'classifie
r': 'Neural Network 2 Layers', 'rf_n_estimators': 487, 'rf_max_depth': 2}. Best is trial 71 with value: 0.7726384364
820847.^[[0m
^[[32m[I 2021-03-22 13:59:47,029]^[[0m Trial 580 finished with value: 0.7657894736842105 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.06881925271679071}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 13:59:47,602]^[[0m Trial 581 finished with value: 0.7448367754830114 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.004733459534537647}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 13:59:54,703]^[[0m Trial 582 finished with value: 0.682643427741467 and parameters: {'classifier
': 'Neural Network 10 Layers', 'rf_n_estimators': 236, 'rf_max_depth': 2}. Best is trial 71 with value: 0.7726384364
820847.^[[0m
^[[32m[I 2021-03-22 13:59:57,133]^[[0m Trial 583 finished with value: 0.7723418134377039 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.13723178976348974}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 13:59:58,613]^[[0m Trial 584 finished with value: 0.7630363036303631 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.031237269160446814}. Best is trial 71 with value: 0.7726384364820847.^[[0m
^[[32m[I 2021-03-22 14:00:01,597]^[[0m Trial 585 finished with value: 0.7731421121251629 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.15251473539125524}. Best is trial 585 with value: 0.7731421121251629.^[[0m
^[[32m[I 2021-03-22 14:00:05,104]^[[0m Trial 586 finished with value: 0.7706302794022093 and parameters: {'classifie
r': 'Logistic Regression', 'logreg_c': 0.27596848095026616}. Best is trial 585 with value: 0.7731421121251629.^[[0m
@@@
-- INSERT --                                                                                  1562,1          6%
```

We also tried to optimize hyperparameters using quasi-cloud-based solution Weights and Biases (wandb). From this, we were able to monitor the progress of the tuning job in a convenient web portal with helpful graphs created in real-time. The setup process for this was incredibly time-consuming since wandb's documentation for Simple Transformers was relatively weak. In particular, we noticed that for the ClassificationModel with multiple tuned parameters, there was not any directly comparable example code which made it challenging to figure out the syntax of certain configuration choices. This resulted in several hours of debugging after which, we were able to create a "sweep config" that defined the ranges of the hyperparameters for which we wanted to tune as well as the model object we wanted to use them in and also the performance metric we wanted to maximize (roc_curve)[4]. See figure 3 on the final page for the sweep config we used. We also had trouble getting the sweep to run more than 20 iterations (we couldn't find a parameter in wandb to change this and the trials seem to fail after a certain number of iterations).

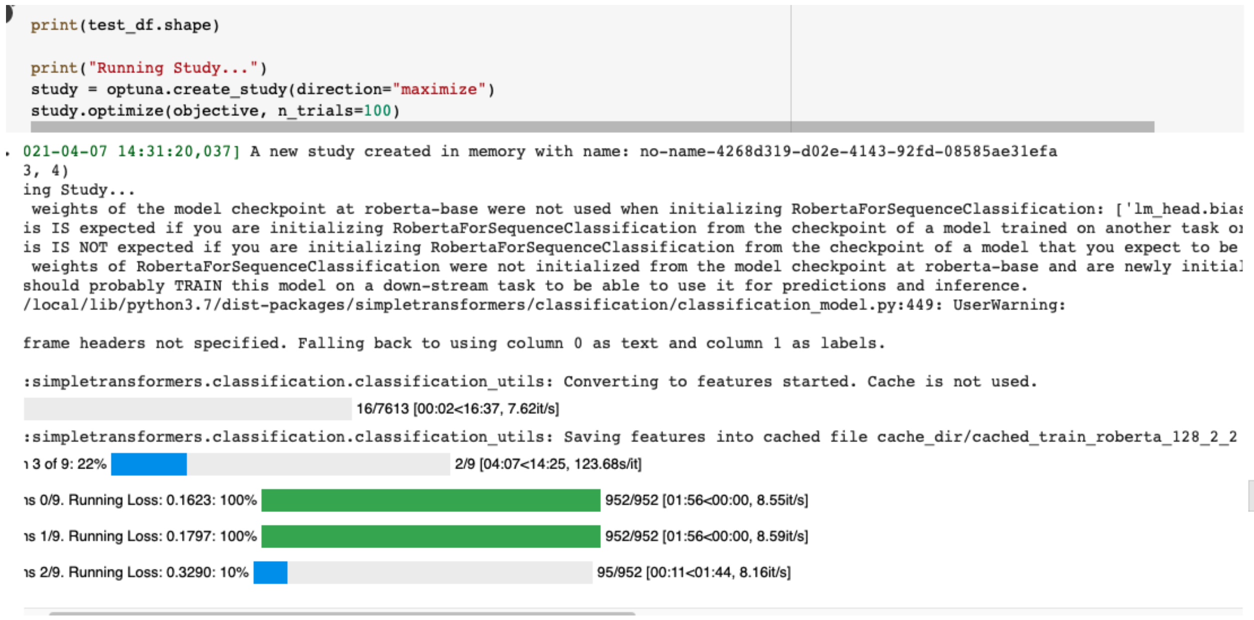
[4] F1 score wasn't available. We figured we would optimize for some other statistic (more representative than training accuracy) and hope for the best. In the end, our final model for Phase 3 won't even be tuned with Weights and Biases anyway, so this shouldn't be considered worrisome.

Figure 3:



Name (69 visualized)	ID	Updated	Runtime	learning_rate	num_train_epochs	Training	lr
devoted-sun-1	28bfyuch	2021-03-2	2m 4s	0.00003	1	0.09391	5.028e-7
visionary-sweep-7	ughraasb	2021-03-2	2m 18s	0.0003118	5	0.1033	6.704e-7
icy-sweep-8	v0ueapcg	2021-03-2	2m 17s	0.0001392	4	0.1053	6.704e-7
gallant-sweep-2	4a9kc8g4	2021-03-2	2m 12s	0.0001195	5	0.1109	6.704e-7
prime-elevator-3	6wywq2c	2021-03-2	2m 1s	0.00004	1	0.1231	6.704e-7
vital-sweep-4	bxwnqn1	2021-03-2	2m 46s	0.00009741	1	0.1263	6.704e-7
ethereal-sweep-5	nicm3c69	2021-03-2	2m 46s	0.0001727	1	0.1433	6.704e-7
jolly-wood-8	1239znz3	2021-03-2	1m 58s	0.00004	1	0.1783	6.704e-7
easy-sweep-6	5fhteezj	2021-03-2	2m 46s	0.0003433	4	0.1818	6.704e-7
visionary-sweep-5	cufz2ou6	2021-03-2	2m 12s	0.0003242	5	0.2261	6.704e-7

The final hyperparameter tuning job is being run as this document is written. We chose Optuna for the final tuning job because it is much simpler than Weights and Biases, meaning we could be sure we were doing it correctly. Optuna is also more industry-standard, so it will be helpful to talk about using it in future job interviews.



```
print(test_df.shape)

print("Running Study...")
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)
```

```
021-04-07 14:31:20,037] A new study created in memory with name: no-name-4268d319-d02e-4143-92fd-08585ae31efa
3, 4)
ing Study...
 weights of the model checkpoint at roberta-base were not used when initializing RobertaForSequenceClassification: ['lm_head.bias
is IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model trained on another task o
is IS NOT expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model that you expect to be
 weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initial
should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/local/lib/python3.7/dist-packages/simpletransformers/classification/classification_model.py:449: UserWarning:

frame headers not specified. Falling back to using column 0 as text and column 1 as labels.

:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
                                          16/7613 [00:02<16:37, 7.62it/s]
:simpletransformers.classification.classification_utils: Saving features into cached file cache_dir/cached_train_roberta_128_2_2
ı 3 of 9: 22%                             2/9 [04:07<14:25, 123.68s/it]
ıs 0/9. Running Loss: 0.1623: 100%        952/952 [01:56<00:00, 8.55it/s]
ıs 1/9. Running Loss: 0.1797: 100%        952/952 [01:56<00:00, 8.59it/s]
ıs 2/9. Running Loss: 0.3290: 10%         95/952 [00:11<01:44, 8.16it/s]
```