Final Project Report

Sadie Crawford and Jonah Kornberg

Executive Summary

The problem we went about solving for our final project was entitled "Real or Not?" and involves classifying tweets as either being about a real disaster or not. This question was posed by a competition on Kaggle.com designed for beginners to natural language processing. The competition supplied 11,000 total datapoints, divided into train and test datasets. Each datapoint contains an ID, text, and optionally a keyword and location. We chose to only consider the text of the tweets in our solution as several examples in the train and test set were lacking the optional entries.

We attempted our solution in a couple iterations. More details can be found in our **Approaches and Tools** section. In summary, we first used basic algorithms such as naïve Bayes and decision tree classifier as found in SciKit learn. Our results were decent, but we wanted to see if we could do better, so we created several neural networks using PyTorch and SimpleTransformers. In conclusion, our best model received a 0.83 F1 score and is currently placed at 212 on the Kaggle leaderboard. We used Google's BERT language model with the transformers package HuggingFace to achieve this result. Other attempts included RoBERTa and LSTM. We will dive further into the runtimes and evaluations of each model in our **Metrics and Evaluations** section.

Overall, we are satisfied with our results and are looking positively at our futures as data science students. We learned a lot in the short time we had and have gained a great appreciation for the tools and resources that are so readily available to help similarly inexperienced students.

Improvements since the last presentation

We spent our last week refining the code to make our results easier to reproduce. Additionally, we tried to create a few more models such as LSTM and RoBERTa using Pytorch, but neither model outperformed our previous record.

Problem Statement, Data, and Baselines

According to the Kaggle competition, the task was: "You are predicting whether a given tweet is about a real disaster or not. If so, predict a 1. If not, predict a 0."[i] We were provided 7,614 tuples for training which each contained an ID number, text, label, and an optional keyword and location. We decided to ignore the keyword and location for the purposes of our analysis as many entries were missing these features. *Figure 1* shows the counts of the most frequent fake and real keywords in the training dataset. It is interesting for observation purposes, as terms like "body bags" and "harm" were frequently not about actual disasters.

```
In [83]: 1 realDf.sort_values('count', ascending=False).head()
```

Out[83]:

	keyword	count	target
138	derailment	39	1
435	wreckage	39	1
303	outbreak	39	1
123	debris	37	1
301	oil%20spill	37	1

```
In [85]: 1 fakeDf.sort_values('count', ascending = False).head()
```

Out[85]:

	keyword	count	target
58	body%20bags	40	0
235	harm	37	0
15	armageddon	37	0
436	wrecked	36	0
337	ruin	36	0

Figure 1: Analysis of the most common keywords for real and fake tweets in our train dataset.

The dataset also included 3,264 test items which included the same columns except for the label. Evaluation on the full dataset had to be done through Kaggle, and aside from one outlier response with a 1.0 F1 score, the upper end of the results was between .8 and .85, so this was our primary baseline goal to reach.

Approaches and Tools

SciKit Learn Classifiers

SKLearn allowed for the usage of multiple, simple classifiers to be used; those utilized include random forest, linear support vector, multinomial Naïve Bayesian, linear regression, and decision tree classifiers. Working in unison with Pandas and Numpy allowed this to be possible with the given data sets. Because the data set originated from Kaggle and was well-formatted, the information received only needed minor manipulations. The csv files were converted to Pandas data frames, which allowed the modeling to take place. Through term frequency-inverse document frequency, the data frame columns of tweets of both the training and testing set were reformatted. Additionally, the important features within tweets were generated to send to the models. With the training reformatted in this manner along with the features, each model was able to be trained. From there, the reformatted tweets within the testing data frame could be labeled with predictions that are saved in a csv format to allow for Kaggle submissions. This all may be viewed in *Figure 2*.

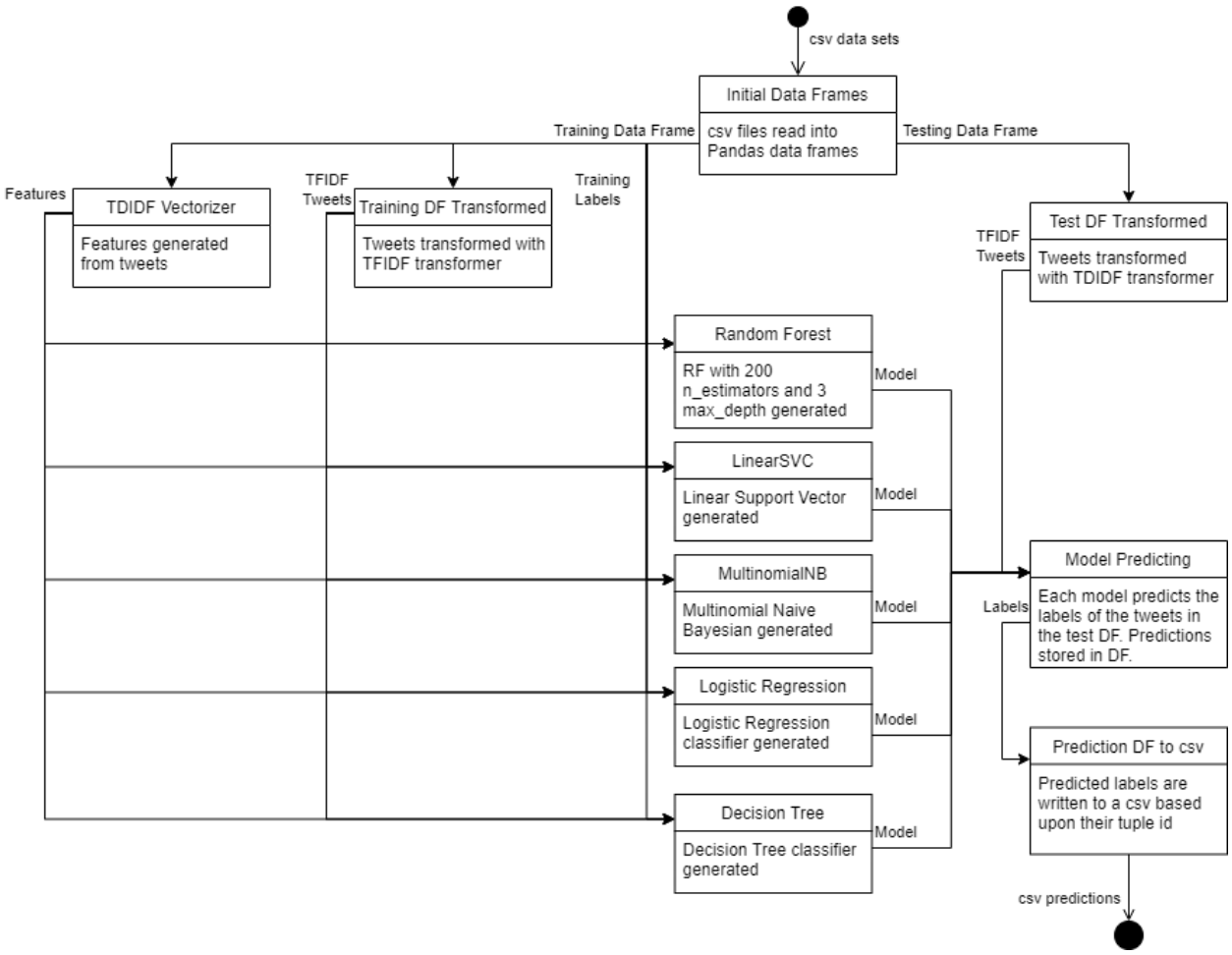

Figure 2: For the SciKit Learn classifiers, the models are generated from features and a reformatted version of the training data. This allows for the Kaggle testing set to be predicted and saved.

BERT Classifier

The BERT classifier was more complicated to successfully create than the previous. The best version of this classifier utilized Google's BERT language model, PyTorch, Pandas, and HuggingFace's transformers. The BERT code is split into 3 sections: pretraining (or preprocessing), model training, and testing. During preprocessing, the unused columns of the original data sets are removed. From the training data set, a validation data set is created to be used in preliminary evaluation of the BERT model. These refined data sets are saved as .csv files to be utilized later. The training and validation data sets are taken from the .csv files and reformatted into BucketIterators that the BERT trains upon. During training, the BERT Model is saved through checkpoints, yielding a .pt file that may be loaded later and avoiding the need for the long training period to reoccur during every new classification runtime. Another BucketIterator is created from the testing .csv file in order that the BERT Model may predict the labels of such data. Once this is done and saved in short term memory within a data frame, the data frame formatted for Kaggle submission with predictions is saved in a .csv file. The

RoBERTa and LSTM classifiers worked very similarly. The only real changes necessary were switching the string tokenizer and encoder from the HuggingFace transformers package.

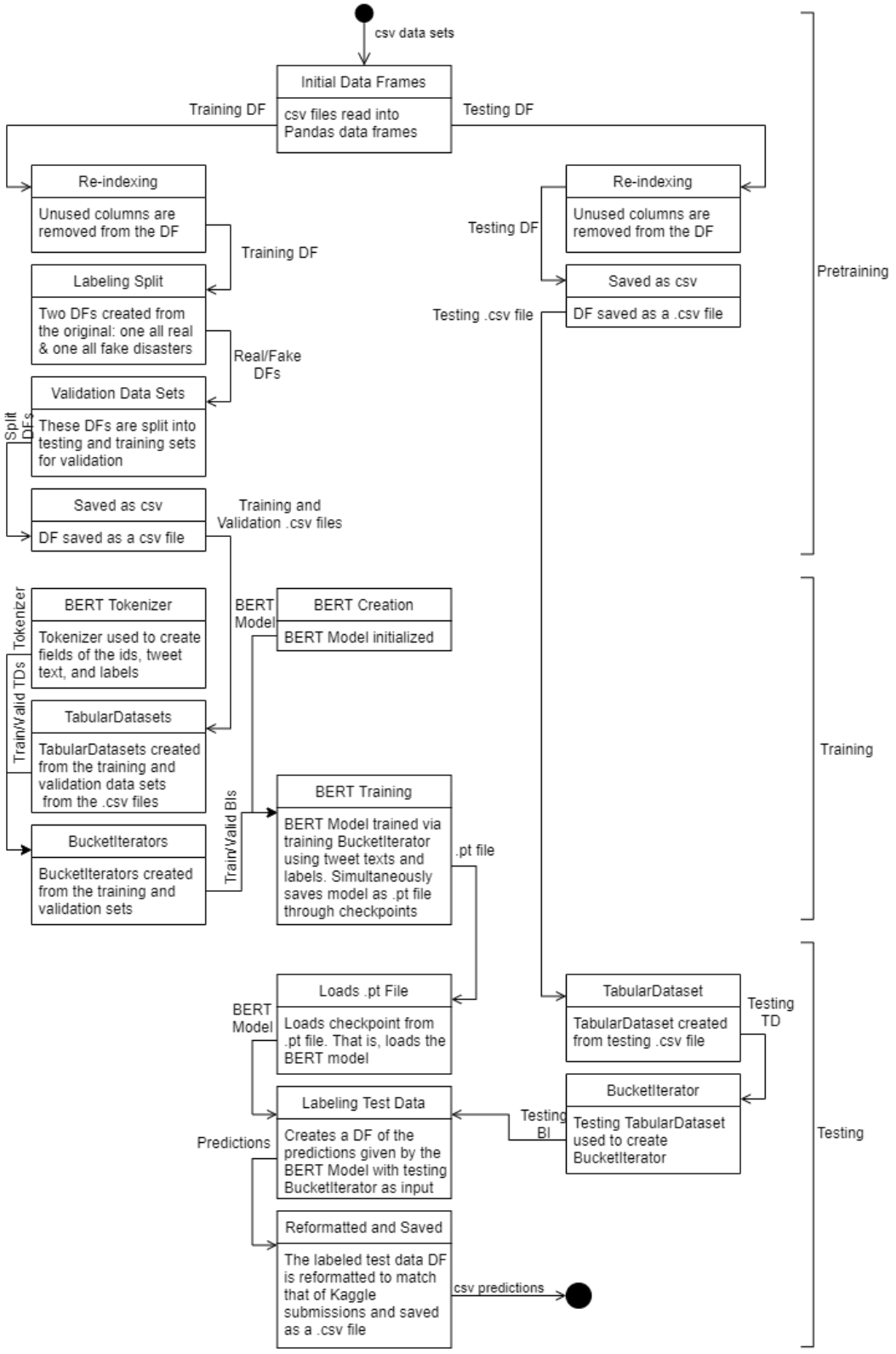

Figure 3: BERT Classifier explanation.

Metrics and Evaluation

Three metrics—accuracy scores, classification runtimes, and F1 scores—were used to identify the success of each classifier. These decisions were made primarily due to Kaggle hiding the test data set's labels. To evaluate the predictions of classifiers, Kaggle utilizes F1 Scores, the harmonic mean of precision and recall. If Kaggle had an infinite submission limit, one could simply submit their model's predictions after each tweak to see whether improvements were made; however, Kaggle has a daily limit of 5 submissions, meaning the team had to create a validation step.

The validation step split the training data set into two, creating training and testing sets. This allows the accuracy of the model trained on a portion of the training data set to be known. The split was done utilizing SKLearn's train_test_split. With enough confidence in the accuracy, the team was able to then submit the predictions to Kaggle to retrieve F1 scores. The F1 scores and classification runtimes are the metrics of true comparison in this instance as they allow for the models to be trained upon the entire training data set whilst comparing them solely on the training set.

Results Analysis

An analysis of the results should take into consideration the different means by which each type of classifier was created. When compared to the deep learning classifiers, the SKLearn classifiers were considerably easier in their construction and relatively more naïve. This is interesting to take into consideration given the difference in F1 scores and runtimes. As will be discussed in the following paragraphs with a couple exceptions, one will see the general pattern: as classification runtime increases so does the F1 scores.

The team decided training runtimes were unnecessary to compare given that models may be saved (such as with the .pt files) or trained before heavy, classification computations; therefore, classification runtimes of the testing data set were used to compare the differences between the various classifiers. The majority of the SKLearn classifiers—excluding random forest who had the highest of any classifier—had a runtime of about a second or less. Meanwhile, the deep learning classifiers all had a run time of between 12 and 20 seconds. Still, considering this is the classification of more than 3,000 tweets, this number is relatively small for every classifier. Even at the highest classification runtime, this is slightly over 73 tweets per second.

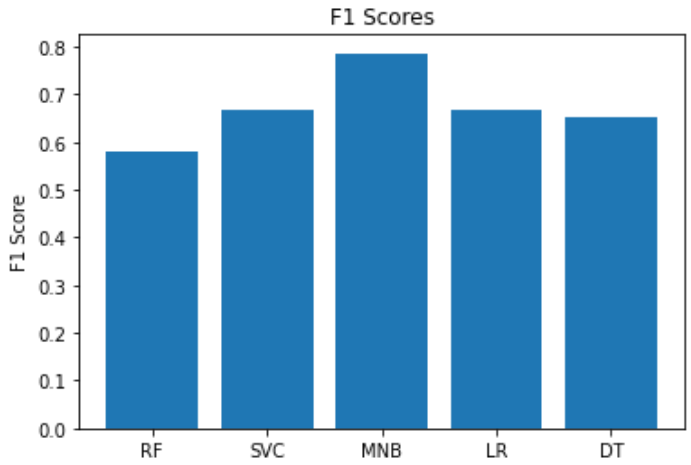

Figure 4: F1 Scores of SKLearn classifiers

As mentioned previously, Kaggle did not disclose the testing set's true labels; this meant relying on Kaggle to evaluate the given classifier. F1 scores were used for this reason. As seen in *Figure 4*, the best SKLearn classifier, which was a multinomial naïve Bayesian reached an F1 score of 0.78577. This did not reach the team's goal of 0.80 to 0.85, which lead to the creation of the next models. As visible in *Figure 5*, three of the remaining models achieved scores within that range. Our best, the BERT model, allowed us to reach 212 on the "Real or Not?" Kaggle leaderboard.

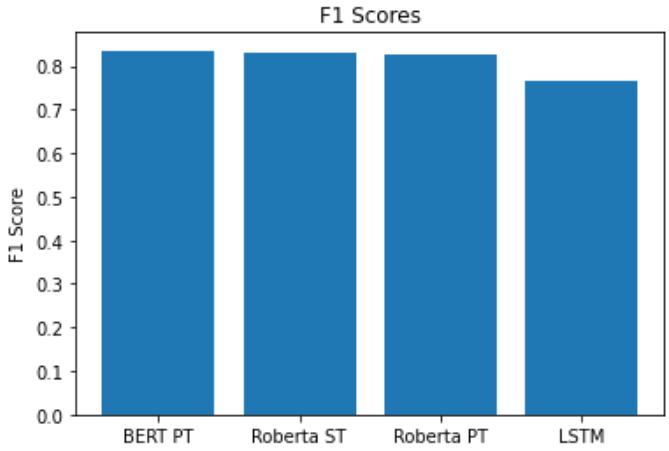

Figure 5: F1 Scores of BERT, Robertas, and LSTM

Challenges and Lessons

Neither team member has had the previous opportunity to work with any sort of machine learning. As such there was a pretty steep learning curve diving into the more complex solutions to our problem. We decided early on that PyTorch would be our facilitator of deep learning based on information found online. The primary issue with PyTorch is that compared to its primary competitor, TensorFlow, it has a much smaller community and resources. We luckily found some very relevant resources and were able to persevere with PyTorch neural networks using some of the most cutting-edge natural language processing techonology. Unfortunately, we

seemed to hit a wall at just above a 0.83 F1 score. While this score met our goal, it was surprising how difficult it was to improve from this baseline. Further improvement to our model will require a much deeper understanding of PyTorch and the associated packages used than we currently have. We became very familiar with transforming the data so that it could fit the formats of the guides we followed. Coming into our assignment, we were expecting an easier and cleaner solution, but we now have a better understanding of the massive complexity of deep learning for even a simple task.

Works Cited

[1] Kaggle, “Real or Not? NLP with Disaster Tweets,” *Kaggle*. [Online]. Available: https://www.kaggle.com/c/nlp-getting-started/

[i] https://www.kaggle.com/c/nlp-getting-started/