Traffic Density Prediction Using NLP & Transformer Models

Chenese A. Gray

The Pennsylvania State University

Dr. Prasenjit Mitra

TABLE OF CONTENTS

ABSTRACT

Traveling in New York City can be a pain, especially when you are not aware of the traffic conditions of your current area. This paper proposes a method to predict traffic density throughout the city of New York using tweets related to traffic in the New York City Metropolitan Area. Data was gathered using SNScrape to collect tweets from the NYC area about traffic. These tweets underwent natural language processing (NLP) and was used to train and test three transformer models including the Text-to-Text Transfer Transformer (T5), Decoding-enhanced Bidirectional Encoder Representation from Transformers (BERT) with Disentangled Attention (DeBERTa), and a Robustly Optimized BERT Pretraining Approach (RoBERTa). Each model yielded an accuracy score of 98.58%, 94.64%, and 97.92%.

Keywords: traffic density, transformers, NLP, tweets, deep learning, classification

INTRODUCTION

According to the United States Census Bureau [1], New York City was the highest populated city in the U.S. in 2020 with 8.38 million residents; more than twice the population of Los Angeles in 2020 (3.973 million) and nearly four times the population of Chicago (2.699 million). Consequently, approximately 1.4 million households in NYC own a vehicle, so two out of every five New Yorkers own a car [3]. Despite the city being home to "one of the largest subway systems in the world" [2], almost half of the city's residents utilize a personal vehicle as their primary mode of transportation. With NYC being the most densely populated area out of all cities in the U.S. with a population greater than one million [11], this high volume of vehicles has in turn caused immense volumes of traffic through the city.

This research proposes using state-of-the-art transformer models to predict the level of traffic density based on commuter tweets.

LITERATURE REVIEW

Wan et al. proposed using BERT to classify whether or not tweets are traffic related or traffic non-related. After traffic non-related tweets were filtered out, a pre-trained BERT Question-and-Answer model was applied to the traffic related tweets to gain more context from the scraped text. They set BERT's hyperparameters – learning rate, batch size, optimization steps, dropout probability, and epochs– as such: 2×10^5 , 24, 4018, 0.1, and 1, respectively. Their model yielded an accuracy score of 98.9%, +2.65% and +5%, respectively of SVM and Naive Bayes.

The next experiment Wan et al., utilizes BERT as a question and answer model. Of the ten thousand tweets fed to the classifier, 137 of them were traffic related. The Q&A model was able to extract traffic information from 118 or 86% of these tweets. The model also yielded 85% F1-score, -4.5% of human performance, which met the authors expectations.

Dabiri & Heaslip utilized deep learning models to classify traffic related tweets and extract their location. The tweets were classified as either non-traffic (NT), traffic incident (TI), or traffic condition and information (TCI), the two latter classifications being traffic related. The authors implemented a Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to perform this classification. They also utilized word2vec and FastText word embedding tools as a substitute for the traditional bag-of-words method.

During experimentation, the authors trained and tested their models using 5-fold validation. They performed hyperparameter tuning then obtained the average results of the 5 folds. They chose to train LSTM as their RNN model and performed experiments using three different models, CNN, LSTM, and CNN + LSTM. Results yielded that all the models performed extremely well regardless of the word embedding tool used. The CNN model performed best out of all the models.

The paper from which this code was adopted was written by Eric & Anna Leon [17]. Eric and Anna used five transformer models, DeBERTa, RoBERTa, T5, GPT, and LongFormer, to classify whether comments made on the internet were toxic or not. The data used for their experimentation were thread conversations acquired from Reddit. As part of their experiments, they trained each model using augmentation, truncation, splitting, oversampling, and various seeds and epochs. Out of all these experiment combinations, they found that DeBERTa performed better than all other models, yielding a final accuracy score of 90.4% when using non-augmented data.

METHODS

Natural Language Processing

Natural language processing (NLP) is a method used that allows computers to understand text in the way that humans do [6]. NLP is the foundation of any text classification experimentation. Considering that this experiment uses tweets as data, the NLP consisted of hashtag, url, and special character removal. As well as making all words lowercase and removing stop words.

RoBERTa

RoBERTa is a pre-training approach built from the transformer model BERT. BERT uses bidirectional self-attention to understand the context of a sequence of words given as an input. Since this paper was published in 2018, it has revolutionized machine learning by providing a state-of-the-art method for NLP tasks [15]. RoBERTa improves on the performance of BERT by training the model longer, over more data with longer sequences, and over more batches. BERT also uses two unsupervised tasks for pre-training: masked language models (MLM) and next sentence prediction (NSP) [4]; RoBERTa removes NSP and "dynamically modifies the MLM used in BERT" [11].

DeBERTa

DeBERTa is a transformer language model that improves the performance of both BERT and RoBERTa by using "two novel techniques: a disentangled attention mechanism and an enhanced mask decoder." Rather than word attention being represented using one vector of the sum of the word and position embedding, it splits these two embedding into two numerical vectors. In addition, DeBERTa places importance on the absolute position of words in a sequence when considering the language modeling process [7].

Text-to-Text Transfer Transformer

Text-to-text transfer transformer (T5) is a transfer learning model that allows string outputs rather than just label output like the other models used in this research. Because of this diverse output format, T5 can be utilized for classification problems as well as "machine translation, document summarization, and question answering" [13].

Random Over Sampling

Random Over Sampling (ROS) is a technique used to fix the problem of imbalanced data. It does so by drawing random samples from the minority classes of

data with replacement. The technique creates these random samples until the instances of each class of data are equal.

EXPERIMENTATION

Data Collection

The data used for this experimentation was acquired from Twitter's API using snscrape (SNS). [9] SNS is used to scrape data from social networking platforms such as Twitter, Facebook, Reddit, etc. SNS acquires the scraped data using keywords and filters provided by the user.

I gathered my data by filtering for the 1000 most recent tweets in NYC that contained traffic related words including "traffic", "crash", "lane", "incident", etc. I then began to remove any tweets that were unrelated to traffic and found that only 20% of the scraped tweets were actually traffic related. However, I noticed that a few profiles in the tweet data were designated traffic accounts. Therefore, I performed another scrape of 5000 tweets using these designated traffic account usernames as keywords and combined them with the tweets from the first scrape. I then labeled all the gathered tweets 0, 1, or 2, representing low traffic, moderate traffic, and heavy traffic. The final data set consisted of 5032 rows and 5 columns labeled "Datetime", "Text", "Label", "Username", and "UserLocation".

Data Preparation & Exploratory Data Analysis (EDA)

To prepare the data for usage, I began by removing the unnecessary columns: "Username" and "UserLocation". Then, I performed various NLP tasks on the tweets. *Figure 1* depicts the raw tweets data. To clean the data, I tokenized the data in the "text" column and stored them under a column labeled "tokenized". Next, I made all the tokenized data lowercase and stripped them of any non-alphanumeric characters. Finally, I performed lemmatization and removed all stopwords. The product of these NLP tasks can be observed in *Figure 2*.

As part of my EDA, I visualized the word frequencies and label distributions of the data. I created a word cloud from the tokenized tweets as strings. As depicted in *Figure 3*, some of the most frequent words and phrases in the data included "traffic", "lane blocked", "stopped", "back", "right lane", and "left lane". Next, I created histograms to display the label distribution of the data before and after performing ROS. Taking a look at *Figure 4*, we can observe the label distribution as 43%, 40% and 17% representing moderate traffic, heavy traffic, and low traffic, respectively. In *Figure 5*, we see an equal distribution of class labels amongst the data, each class containing 2030 rows.

Model Training

The first step in training my transformers is to verify that the data is in a format that the models can understand. I did that by first splitting the tweets into training, testing, and validation data, 70%, 15% and 15%, respectively. Then, I put each of these data sets into a model encoder function using each model's built in tokenizer function. This model encoder returns the data as three arrays, input id, attention mask, and token type id. Simply put, these are

numerical vectors that describe the words in the data by their context and position in the sentence.

The models and tokenizers used in the following experiments are pre-trained from Hugging Face. The original setup of the models was adopted from [5] such that the model consists of five layers. The input layer has three nodes, representing each of the three numerical vectors mentioned above. The next layer is a hidden layer with 256 units, or 256 nodes. The third layer is used to adjust the dimensions of the input data. The fourth layer is a dropout layer, which is given a ratio of the hidden layer to dropout in each training run, in this case, the ratio is 0.1. This layer helps to prevent overfitting the model. Finally, the output layer consists of three nodes, each one representing an output label.

This initial setup of this model did yield excellent results, however, I did run some experiments to test the models' performance even more. The first experiment I conducted was ROS. As mentioned above, I utilized the imblearn package in Python to randomly sample with replacement from the minority class of the tweets. I conducted this experiment to test whether the model would perform better with equally distributed labels and slightly more data. The next experiment I conducted was hyperparameter tuning. I made use of keras tuner and the Tensorflow plugin *TensorBoard.* These tools aided me in testing different values for the models' dropout ratio, dense layer units, and optimizer method. Finally, I used ROS to train the model using the best hyperparameters found in the tuning experiment. The results can be observed in the next section.

RESULTS

The three transformer models trained were tested in multiple experiments. The models were compared based on their validation accuracy and loss. The final versions of the models were compared using Accuracy, F1-score, Recall, and Precision. These metrics can be found in Table 1 and Table 2.

In Table 1, we can observe consistently good performance from all the models throughout the experiments. For the first two training sessions, RoBERTa came out on top of its competitors. However, both RoBERTa and DeBERTa ran into trouble during the hyperparameter tuning phase. Taking a look at Figure 6, we can see that the tuning result reported in Table 1 was the only set of hyperparameters RoBERTa performed adequately with. Similar results were observed with the DeBERTa parameter tuning, however, not one of the hyperparameter combinations could score above 50% accuracy. Although one set of hyperparameters did yield a high accuracy with the RoBERTa model, loss was better minimized with the default hyperparameters. From this, I concluded that the default parameters would be best for training the RoBERTa and DeBERTa models.

T5 exhibited consistent high performance throughout the experiments. In Table 1, we see +.9% and +.51% increases in validation accuracy when training with ROS and optimized parameters, respectively. When combining the optimized parameters and ROS, T5 yielded 98.58%

accuracy, 2.11% greater than the default model accuracy; as well as its smallest loss value of .0668.

If we take a deeper look into the models' performance using the confusion matrices in Figure 7, 8, and 9, we can detect slight confusion within all three models when predicting heavy traffic and sometimes moderate traffic. Considering DeBERTa performed the poorest, this model had the most confusion when predicting these labels, particularly heavy traffic. Contrarily, T5 shows the least confusion when predicted these labels. RoBERTa can be found, once again, performing between the two with equal loss amongst both moderate and heavy traffic labels. I also noticed that the least confusing prediction was consistently 0 or low traffic in all the models.

CONCLUSION

This study reveals that pre-trained T5, RoBERTa and DeBERTa can successfully predict traffic density using tweet data. All three models were trained using ROS and various hyperparameter combinations. Based on the experimentation results, ROS can improve models' performance by balancing minority classes within data. Therefore, ROS was used in the final model testing. T5 was built using 192 hidden layers, a .25 dropout rate, and 1e-3 learning rate. The final testing yielded an accuracy score of 98.58%. Both RoBERTa and DeBERTa underwent the final testing stage using 256 hidden layers, a .1 dropout rate, and a 1e-5 learning rate. RoBERTa yielded an accuracy score of 97.92% and DeBERTa yielded 94.64%. Clearly, all models performed well with the task at hand, but T5 performed above the rest. Future considerations and lesson learned can be found in the following section.

DISCUSSION

One thing that I experienced trouble with throughout this study was the runtime in Google Colab notebook. Due to the limited utilized time of the free GPU in all colab notebooks, I often ran into problems like resource exhaustion error, time-expensive training sessions and disconnected runtimes. This issue also led me to play with DeBERTa's batch size. Because DeBERTa was the last model to be trained amongst the three, it was the one that often suffered from resource exhaustion. Increasing its batch size allowed it to complete epochs faster, thus conserving resources. Because of this, in the future I would utilize a Jupyter notebook or a text editor with a terminal for better execution, as well as Tensorflow's built in TPU.

Some other problems I encountered while conducting experiments stemmed from hyperparameter tuning. Initially, the tuning was set to run 3 trials of 2 epochs each. While observing the trials, I noticed that when the "sgd" optimizer was used, the trial yielded poor validation accuracy. I also detected that the number of trials greatly limited the combinations of hyperparameters that could be explored. As a result, I removed "sgd" from the choices of optimizers to be used, and increased the trials and epochs to 5 and 3, respectively.

Finally, when I was analyzing the evaluation metrics from the final testing of the model, I noticed that I had the average method for all the metrics set to "micro". I

found that this was happening because rather than establishing which labels are positive and negative, the “micro” option regards them in a neutral way. This causes false positives and negatives and true positives and negatives to be similar, thus generating similar metrics [14]. This caused all the metrics of each respective model to yield the same value, which raised a red flag. This prompted me to explore the different average methods and the metrics they yielded, leading me to utilize the “macro” average method.

APPENDIX

References

[1] Bureau, U. C. (2022, April 7). *American community survey 5-year data (2009-2020)*. Census.Gov. https://www.census.gov/data/developers/data-sets/acs-5year.html

[2] Contributors to Wikimedia projects. (2022, July 10). *History of transportation in New York City*. Wikipedia. https://en.wikipedia.org/wiki/History_of_transportation_in_New_York_City

[3] Dabiri, S., & Heaslip, K. (2018). Developing a Twitter-based traffic event detection model using deep learning architectures. *Expert Systems with Applications*, *118*(15), 425–439. https://doi.org/10.1016/j.eswa.2018.10.017

[4] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018, October 11). *BERT: Pre-training of deep bidirectional transformers for language understanding*. ArXiv.Org. https://arxiv.org/abs/1810.04805

[5] dimasmunoz, "Text Classification with RoBERTa (and TPUs) 🤗," *Kaggle*, Aug. 17, 2020. Accessed: Oct. 06, 2022. [Online]. Available: https://www.kaggle.com/code/dimasmunoz/text-classification-with-roberta-and-tpus/notebook

[6] Education, I. C. (2020, July 2). *What is Natural Language Processing?* IBM. https://www.ibm.com/cloud/learn/natural-language-processing

[7] He, P., Liu, X., Gao, J., & Chen, W. (2020, June 5). *DeBERTa: Decoding-enhanced BERT with disentangled attention*. ArXiv.Org. https://arxiv.org/abs/2006.03654

[8] imbalanced-learn documentation — Version 0.9.1. (2022, May 17). Imblearn. https://imbalanced-learn.org/stable/#

[9] JustAnotherArchivist. (n.d.). *GitHub - JustAnotherArchivist/snscrape: A social networking service scraper in Python*. GitHub. Retrieved October 3, 2022, from https://github.com/JustAnotherArchivist/snscrape

[10] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019, July 26). *RoBERTa: A robustly optimized BERT pretraining approach*. ArXiv.Org. https://arxiv.org/abs/1907.11692

[11] Maciag, M. (2013, October 1). Population density for U.S. cities statistics. *Governing*. https://www.governing.com/archive/population-density-land-area-cities-map.html

[12] NYCEDC. (2018, April 5). *New Yorkers and their cars*. NYCEDC. https://edc.nyc/article/new-yorkers-and-their-cars

[13] Roberts, A., & Raffel, C. (2020, February 24). *Exploring transfer learning with T5: The text-to-text transfer transformer*. Google AI Blog. https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html

[14] RoflcoptrException. (2014, May 22). *What does it imply if accuracy and recall are the same?* Cross Validated. https://stats.stackexchange.com/questions/99694/what-does-it-imply-if-accuracy-and-recall-are-the-same

[15] Vajpayee, S. (2020, August 6). Understanding BERT — (bidirectional encoder representations from transformers). *Towards Data Science*. https://towardsdatascience.com/understanding-bert-bidirectional-encoder-representations-from-transformers-45ee6cd51eef

[16] Wan, X., Ghazzai, H., & Massoud, Y. (2020). *Leveraging personal navigation assistant systems using automated social media traffic reporting*. IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9140144?casa_token=83KRr_1gu2kAAAAA:nnfgZ7qIJb03OYN5aqgoPxhEWeyIzXECLgAp6cNlrotwLjOGqf92jFEZ9GS1NTUX7RR2DGS51w

[17] X, E., & Leon, A. (2021). *Final Report*.

Figures & Tables

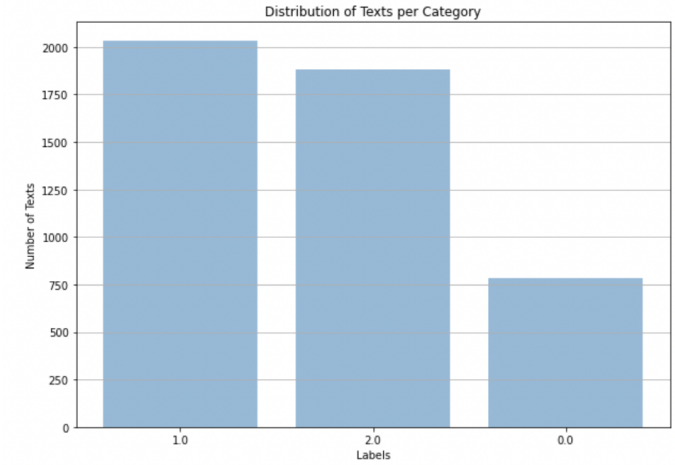
	Datetime	Text	Label
0	2022-08-07 21:15:43+00:00	Accident in #Manhattan on The FDR Drive SB at ...	1.0
1	2022-07-29 10:25:44+00:00	Closed due to Police Activity in #NewYork on W...	2.0
2	2022-07-29 10:25:44+00:00	Closed due to Police Activity in #NewYork on A...	2.0
3	2022-07-29 10:10:43+00:00	Closed due to Police Activity in #NewYork on A...	2.0
4	2022-07-29 09:50:43+00:00	Closed due to Police Activity in #NewYork on A...	2.0

Figure 1: Pre-NLP Data

```
0    [accident, fdr, drive, sb, st, stopped, traffi...
1    [closed, due, police, activity, west, street, ...
2    [closed, due, police, activity, avenue, americ...
3    [closed, due, police, activity, avenue, americ...
4    [closed, due, police, activity, avenue, americ...
5    [closed, due, police, activity, avenue, americ...
6    [closed, due, police, activity, west, street, ...
7    [overturned, vehicle, two, lane, blocked, onth...
8    [left, lane, blocked, onthefdr, fdr, drive, sb...
9    [overturned, vehicle, two, lane, blocked, onth...
Name: tokenized, dtype: object
```

Figure 2: Post-NLP Data

Figure 3: Tweet Word Cloud




```
Counter({1.0: 2030, 2.0: 1881, 0.0: 784})
```

Figure 4: Label Distribution of Imbalanced Data

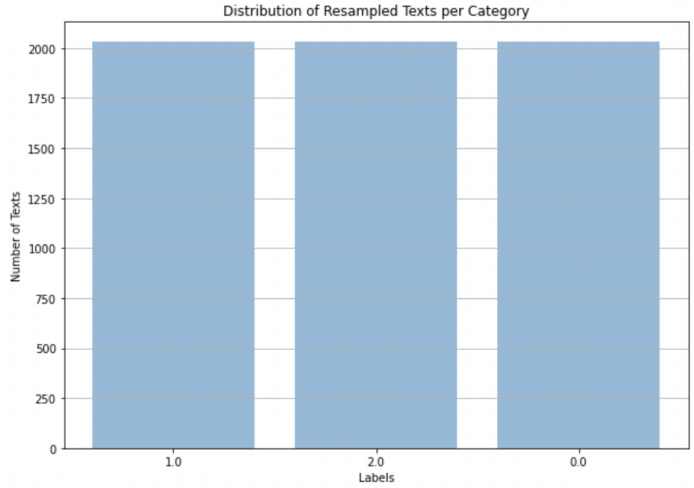


Figure 5: Label Distribution of Balanced Data

Trial ID	Show Metrics	lr	units	dropout	train.epoch_accuracy	validation.epoch_accuracy	train.epoch_loss	validation.epoch_loss	validation.evaluation_accuracy_vs_iterations
0	☐	0.00010000	192.00	0.10000	0.43798	0.42401	1.0271	1.0249	0.42401
1	☐	0.0010000	192.00	0.10000	0.94444	0.96353	0.16919	0.21859	0.96353
2	☐	0.00010000	256.00	0.10000	0.43798	0.42401	1.0270	1.0253	0.42401
3	☐	0.010000	256.00	0.25000	0.43798	0.42401	1.0268	1.0247	0.42401
4	☐	0.00010000	192.00	0.25000	0.40335	0.42401	1.1335	1.0575	0.42401

Figure 6: RoBERTa HP Tuning Results

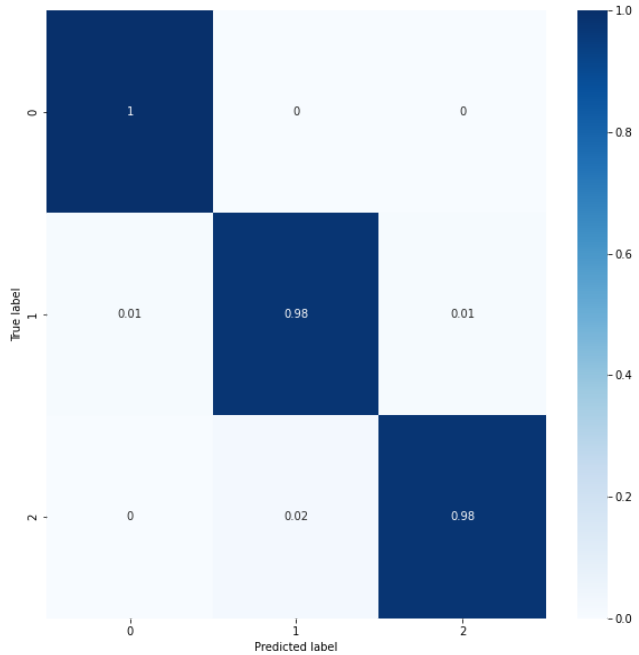


Figure 7: T5 Final Testing Confusion Matrix

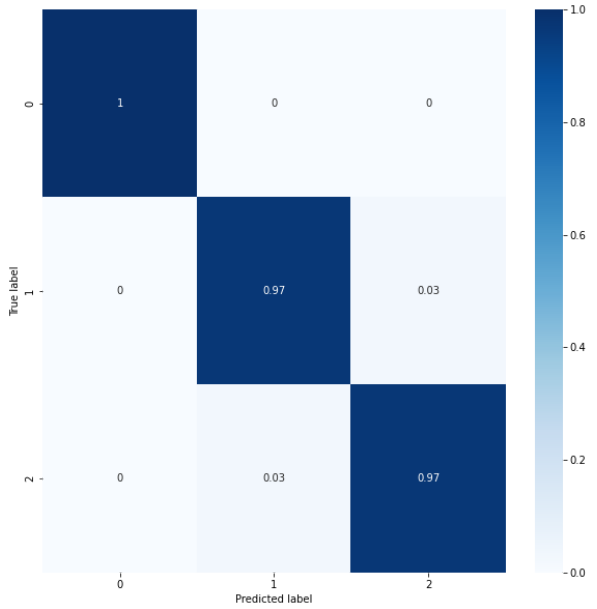


Figure 8: RoBERTa Final Testing Confusion Matrix

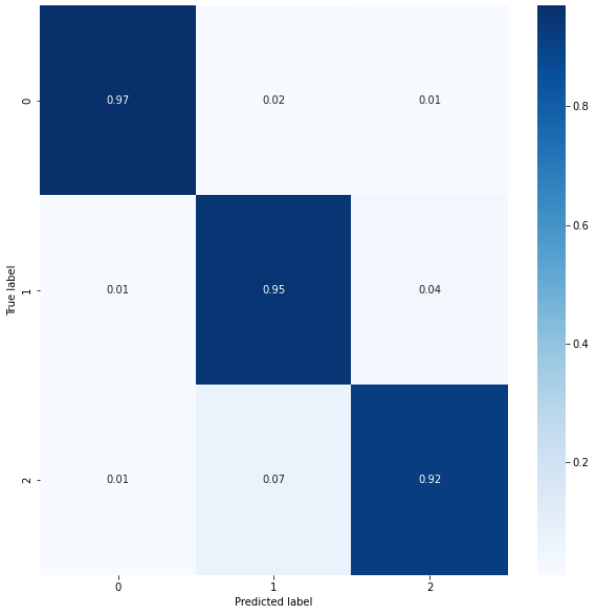


Figure 9: DeBERTa Final Testing Confusion Matrix

	Default Model		*ROS + Default Model*		*HP Model*	
	Valid. Acc	*Valid. Loss*	*Valid. Acc*	*Valid. Loss*	*Valid. Acc*	*Valid. Loss*
T5	*.9645*	*.0789*	*.9737*	*.0747*	*.96960*	*.11829*
RoBERTa	*.9773*	*.0729*	*.9825*	*.0544*	*.96353*	*.21859*
DeBERTA	*.9744*	*.0728*	*.9781*	*.0986*	*<.50*	*>.50*

Table 1: Models' Validation Accuracy and Loss

	Recall	*Precision*	*F1-Score*	*Accuracy*	*Loss*
T5	*.9856*	*.9859*	*.9857*	*.9858*	*.0668*
RoBERTa	*.9789*	*.9789*	*.9789*	*.9792*	*.0488*
DeBERTa	*.9460*	*.9468*	*.9462*	*.9464*	*.1343*

Table 2: Final Testing Evaluation Metrics

Code

https://colab.research.google.com/drive/1LcTkAV9GF1H2gsmvSHLZVCvCVS03-Z--#scrollTo=T9XYQgHDQETq

https://colab.research.google.com/drive/1-cn62Lu2sVfl0dXy_5z5UB3py_FOFLZI

https://colab.research.google.com/drive/18anYK6diJ8e1FPrUbgCVPi8KMhMsjE3w