AI Course Project Report

Natural Language Processing with Disaster Tweets

Group 4: Lexy Andershock, Gian Fernandez-Aleman, Ethan Miller, Kevin Lam, David Long, and Rylee Petrole

Abstract

Our group participated in the Kaggle competition "Natural Language Processing with Disaster Tweets" to explore NLP applications and machine learning implementations. The baseline model was a DistilBERT pretrained model from KerasNLP. We focused on optimizing hyperparameters, such as the learning rate, batch size, and number of epochs, to improve the model. We did not implement data augmentation, as our dataset was balanced. We also included an early stopping function that monitored the value loss at each epoch. Our advanced model showed slight improvements in accuracy, F1 training score, and average training time, but suffered from overfitting, as seen by the higher F1 training score coupled with a stagnant F1 validation score.

Introduction

The goal of this assignment was to fine-tune a BERT classifier model to detect whether or not a Twitter Tweet is announcing a disaster. The dataset contains text from 10,876 hand-classified tweets. The baseline model, a DistilBERT classifier, is a lightweight version of BERT that runs 60% faster while retaining 95% of BERT's original performance [1]. We measured model performance through F1 score, accuracy, and average training time. We aimed to improve these metrics by optimizing the learning rate, batch size, and number of epochs. We also implemented an early stopping function. The hardware used was a 2024 MacBook Pro with 24 GB of unified memory and 12 CPU cores.

Improvement Ideas & Experiments

We used the paper by Sun et al. [2] to guide our implementation decisions for the advanced model. In this paper, they discovered that a learning rate of 2e-5 or lower is necessary to prevent catastrophic forgetting, a phenomenon in which pre-trained knowledge is erased during the learning of new information. Thus, we choose a slightly larger learning rate of 2e-5 to speed up convergence while overcoming this issue. We also lowered the batch size to 32, which helps with GPU integration because it is a power of two. This batch size was enough to optimize GPU usage without CPU throttling. Since our largest floating-point vector's size was 151, this batch size also fit within our machine's memory limitations. During training, 9 of 12 cores were used, resulting in 75% GPU utilization. We chose not to balance the dataset because it was not severely imbalanced: 57% of the training set were Not Disaster Tweets, and 43% were Disaster Tweets. For the number of epochs, Devlin et al.'s paper [3] found that three epochs were usually sufficient for GLUE (general language understanding evaluation tasks). We implemented an early-stopping function based on observed plateauing in value loss, with a maximum epoch limit of four. Our model stopped after three epochs, which is consistent with the literature. The advanced model saw a slight improvement, with an accuracy score of 0.82899. The F1 score on the training data rose to 0.84, and there were fewer false positives (N=197) and false negatives (N=576), as shown in Figure 1. However, the lack of improvement in the validation set indicates that this model suffers from overfitting. This model took 33.7 minutes to train.

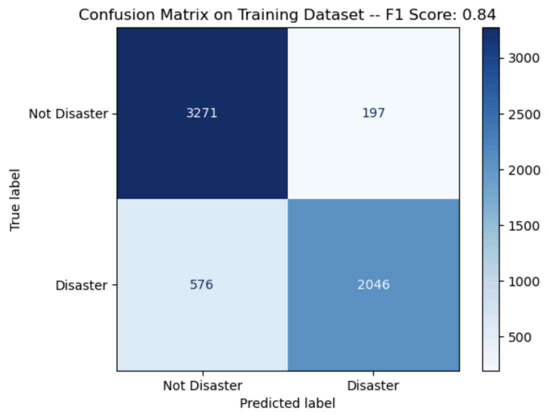

Figure 2 – Confusion Matrix for Advanced Model Training

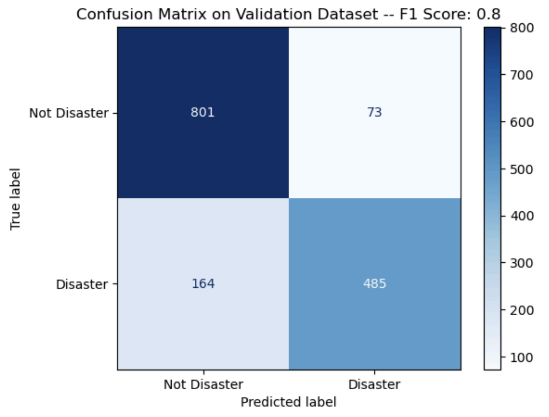

Figure 2 – Confusion Matrix for Advanced Model Validation

Results & Discussion

In the baseline model, the parameters that optimized performance were a large batch size of 100 and a small epoch size of 1, as seen in Table 1. We did not adjust the learning rate from its original value of 1e-5.

	Batch Size	**Learning Rate**	**Epochs**	**Accuracy**	**F1 Training Score**	**F1 Validation Score**	**Training Time (*avg. sec x epoch*)**
Original Model	100	1e-5	1	0.8069	0.79	0.79	697s x 1
Advanced Model	32	2e-5	3	0.8289	0.84	0.80	675s x 3

Table 1 – Parameter & Accuracy Comparison across the Models

The original model performed moderately well, with an accuracy of .80692, and had a relatively short training time of approximately 11.6 minutes, as seen in Table 1. The model also generalized well, as evidenced by the similar F1 scores between the training and validation sets in Figure 2. The total core usage fluctuated between six and nine, indicating that the system was suffering from CPU throttling due to the large batch size. This model also had a moderate rate of false negatives (N=560) and false positives (N=533), and its accuracy score left room for improvement. Compared to the baseline model, the advanced model showed slight improvements. However, significant tradeoffs in training time were noted, and its F1 score suggests overfitting. We speculate that the BERT classifier model will see minimal improvement in accuracy, even with the implementation of other techniques. Future research should focus on using other pre-trained models to compare their performance with BERT's.

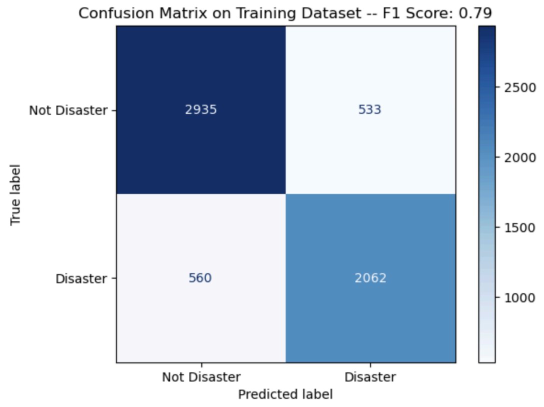

Figure 3 – Confusion Matrix for Original Model Training

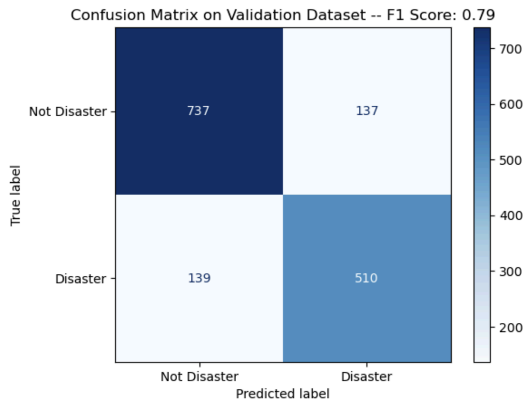

Figure 4 – Confusion Matrices for Original Model Validation

Challenges and Lessons Learned

Our first hurdle was correcting the original model's implementation in Jupyter Notebooks. This took around two hours to complete. Another difficulty was finding previous literature about DistilBERT improvements. Once we found a few papers, however, our project benefited immensely. We recommend conducting a literature review before implementing future project changes. Our biggest bottleneck, though, was the model's slow computation speed. Making changes was difficult because we would often wait 20-30 minutes for the model to re-run, which limited the amount of testing we could feasibly accomplish for this project.

Team Member Contribution

Lexy Andershock contributed to the project by organizing report writing and editing, as well as conducting the literature review. Gian Fernandez-Aleman was responsible for writing and editing the report. Kevin Lam (team leader) contributed by fixing the original Jupyter Notebook implementation on Windows and reviewing the report. Rylee Petrole contributed to model ideas and technical implementation. David Long outlined the project timeline to ensure the project's completion. Ethan Miller coordinated the team's efforts, organized the final project submission, and assisted in writing the project code.

References

[1] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper, and lighter," in *arXiv*, 2019. doi: 10.48550/arXiv.1910.01108.

[2] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to Fine-Tune BERT for Text Classification?," in *Lecture Notes in Computer Science (including subseries Lecture Notes in*

Artificial Intelligence and Lecture Notes in Bioinformatics), 2019. doi: 10.1007/978-3-030-32381-3_16.

[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2019. doi: 10.18653/v1/N19-1423.