DSGA 1019 Final Project Report

Yilong Chen, Yaozhong Huang, Cewen Zhu, Ning Yang, Chuqin Wen, Rui Hu
{yc6292 yh2563 cz2767 ny675 cw4155 rh3553}@nyu.edu
New York University

May 2023

1 Introduction

With the growing popularity of smartphones and social media, platforms such as Twitter became popular channels to communicate emergencies. The public can quickly report incidents they observe in real-time, which has prompted various organizations, such as disaster relief agencies and news agencies, to systematically monitor Twitter for the newest events. This makes it meaningful to create machine learning models to classify disaster-related tweets accurately and efficiently.

In this report, we explore the use of Natural Language Processing (NLP) and advanced Python techniques to preprocess and classify tweets accurately. To accomplish this task, we divided the process into three main steps: preprocessing, model development, and optimization.

First, we preprocessed the tweets using techniques such as tokenization, removal of unnecessary words (e.g. emojis, numbers, tags, mentions and URLs) and lemmatization. The preprocessed tweets were then fed into our machine learning models for training and evaluation.

As our baseline, we developed an SVM model using bag-of-words vectorization, which achieved an accuracy of 76.66%. We then fitted a BERT model to our preprocessed tweets using a 3-layer neural network with sigmoid activation functions. Using BERT we achieved an accuracy of 83.79%, which is a 10% improvement over our baseline.

To further improve the efficiency of our methods, we optimized our code by employing techniques such as loop reduction, Cython, avoidance of function calls, and multiprocessing. As a result, we were able to reduce preprocessing time by 49.36% and SVM training time by 99.91%, significantly improving the efficiency of our implementation.

2 Data Preprocessing

During the preprocessing stage, we removed 110 duplicate and 18 contradictory tweets. We then performed tokenization on each tweet, converting them into lists of words to help us recognize unnecessary information. To ensure optimal model performance, we eliminated digits, emojis, punctuations, stopwords, websites, hashtags, and mentions, aiming to maintain the purest possible input. The final step involved lemmatization, which transformed words to their root form, such as converting "walking" to "walk".

3 Models

We built two models: BERT Large Model (uncased) and SVM.

3.1 BERT Large Model (uncased)

To build our machine learning model, we utilized the pretrained tokenizer and model from BERT large model (uncased) as our base. BERT, or Bidirectional Encoder Representations from Transformers, is a powerful language representation model that performs excellently on various natural language processing tasks.

3.1.1 Transfer Learning

Transfer learning was implemented using the pretrained BERT large model (uncased). We fine-tuned it to adapt to our specific task of detecting real disaster tweets. We designed a 3-layer sequential neural network with a Sigmoid activation function for the binary classification task. The neural network takes the output from the BERT model and processes it through the layers to generate a final probability of the tweet is about a real disaster.

To prevent overfitting, we applied the dropout method to our neural network layers, which is a regularization technique that helps improve the model's generalization by randomly disabling a proportion of neurons during training. After hyperparameter tuning, a dropout rate of 50% gives the best validation performance and was applied to the final model.

3.2 SVM

We built our SVM from scratch. We used the Python Counters class to implement the bag-of-words for word vectorization. For the naive SVM, we chose the Pegasos algorithm with stochastic sub-gradient descent. For the improved SVM, We found that the update equation

$$w_{t+1} = (1 - \eta_t \lambda) w_t + \eta_t y_j x_j$$

in the Pegasos algorithm is equivalent to the following update step:

$$s_{t+1} = (1 - \eta_t \lambda) s_t$$

$$W_{t+1} = W_t + (\eta_t y_j x_j)/s_{t+1}$$

After hyperparameter tuning, we achieved an accuracy of 76.66% with an optimal lambda value of 0.0078.

4 Optimizations

We employed optimization techniques on preprocessing, BERT Large Model, and SVM.

4.1 Preprocessing Optimization

In the preprocessing part, we mainly did four optimizations. First, we used the Map function instead of for loops. In addition, we decreased function calls. Based on the previous two strategies, we employed two methods separately to optimize our program. We first used cython by declaring cython variable types, and functions because some for loops are hard to be converted to Map, and cython is very efficient on improving for loops. We also used multiprocessing techniques to improve the running time.

4.1.1 Results

1. *Map function instead of for loops:* Running time changes from 23.3 s to 16 s. Improved **31.33%**.

2. *Decrease Function Calls:* Running time changes from 16 s to 14.1 s. Improved **11.88%**.

3. *Cython:* Running time changes from 14.1 s to 13 s. Improved **7.80%**.

4. *Multiprocessing:* Running time changes from 14.1 s to 11.8 s. Improved **16.31%**.

Overall, multiprocessing based on the first two optimizations performed the best. Running time changes from 23.3 s to 11.8 s. Improved **49.36%**

4.2 BERT Large Model Optimization

The model was trained for 7,000 epochs. The number of epochs was determined through experimentation to achieve optimal model performance.

Our model was trained on a GPU, with the host part running on the CPU to set up the parameters and data for the computation. The GPU was utilized to run multiple kernels to perform the actual computation, significantly accelerating the training process. Additionally, we used batch processing to further speed up the training process.

4.2.1 Results

Our model achieved an accuracy of 0.83787 on the disaster tweet detection task, placing it in the top 5% among 1,200+ participants in the competition. This demonstrates the effectiveness of our method, which combines the power of a BERT large model (uncased) and transfer learning using a well-designed neural network architecture.

4.3 SVM Optimization

We focus on two aspects of optimization: loop reduction and Cython-style coding.

4.3.1 Loop Reduction

In the original Python code, the `dotProduct` function uses nested loops, which we have reduced to a single loop with Python's built-in `sum()` function and dictionary comprehension. This simplification leads to potential performance improvements.

For the `pegasos_advanced` function, we employ the `cycle()` function to create a generator that cycles through data points in $\mathbf{X}$ and $\mathbf{y}$. The `zip()` function then combines iterators for input features and labels during stochastic gradient descent, resulting in efficient memory usage and cleaner code.

4.3.2 Cython-style Coding

The Cython implementation introduces type annotations for efficient C code generation and defines Cython functions using the `cdef` and `cpdef` keywords. These optimizations result in reduced overhead and improved performance.

4.3.3 In-place updates

The optimized increment function updates dictionaries in place, reducing memory overhead and contributing to loop reduction by removing the need to loop through the dictionary again to create a copy.

4.3.4 Results

These optimizations enhance the performance and efficiency of the SVM algorithm, making it more suitable for large-scale problems and datasets.

1. *Loop Reduction* Running time changes from 5314 s to 10.9 s. Improved **99.79%**.

2. *Cython-style Coding* Running time changes from 10.9 s to 4.39 s. Improved **59.72%**.

Overall, the running time changes from 5314 s to 4.39 s. Improved **99.91%**

5 Conclusions

To summarize, our best-performing model for classifying true disaster events on Twitter is BERT, which achieved an impressive testing accuracy of 83.79%. Additionally, we implemented a simple SVM model from scratch using Pegasos algorithm as our baseline model, which achieved a stable testing accuracy of 78.7% on the same task. We were able to significantly optimize our data preprocessing pipeline, resulting in a 49.36% improvement in runtime. Similarly, our optimizations for the SVM model resulted in a remarkable 99.91% improvement in runtime.