Natural Language Processing with Disaster Tweets

Introduction

The aim of this project is to carry out binary classification of tweets to establish whether or not the tweet refers to a real disaster. The dataset was taken from the Kaggle competition "Real or Not? NLP with Disaster Tweets." It consists of the tweet texts and a binary label target (1 or 0) indicating the presence or absence of a disaster context. This classification problem is primarily evaluated using accuracy, but precision, recall, and F1 score also add additional value into understanding how the model performs.

Our Method

Data Preprocessing

For the baseline, we implemented a traditional Natural Language Processing (NLP) algorithm based on the procedure of cleaning, normalizing, and preparing the text of the tweet for use in classical machine learning models. This process started with changing all the text to lower case for uniformity and to minimize token sparsity. Subsequently, we used regular expressions to remove non-relevant components such as URLs, Twitter account mentions like @username, hashtags, and all punctuation marks. Then, the words in the cleaned text were tokenized into words through the use of word_tokenize from the NLTK library.

To further reduce noise while semantically emphasizing important words, we applied stop word removal using NLTK's English stop word list. After that, we performed lemmatization with WordNetLemmatizer, which shrinks words into their dictionary base forms, and then stemming with PorterStemmer, which reduces words to their stem form (for example, "connected," "connecting," and "connection" would all be indexed as "connect"). Such normalization allowed us to reduce morphological variation with more consistency. Lastly, we used TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to convert the text into numerical features. This method underscores and amplifies words unique to a document while deemphasizing those frequently used throughout documents. We set the vectorizer to pull out unigrams and bigrams with a limit of 5000 tokens (maxfeatures=5000, ngramrange=(1, 2)) because initial tests displayed balanced effectiveness and good outcomes at this range.

On the second checkpoint, the BERT-based model required more up to date and model-specific preprocessing methods. BERT does not require manual text pre-processing, instead using deep contextual understanding along with pre-trained embeddings. For tokenization, we used BertTokenizer from HuggingFace. The tokenizer splits words into WordPiece tokens, inserts special classification and separation symbols (CLS and SEP), and manages out-of-vocabulary words effectively using subword techniques. Subsequently, each sequence was either padded or truncated to 128 tokens, which enabled all inputs to have the same dimensions for batch training.

The tokenizer created three critical outputs for each tweet: the input IDs which is a set of tokens keyed to integers, attention masks for padding tokens, and token type IDs for differentiating segments in a pair of sentences; although not essential for the single sentence context. These outputs could not be obtained with traditional pipelines such as stopword removal, stemming, or lemmatization.

Reason being, BERT's self-attention sees to learning all semantic associations that exist in the non shrunk text during refining.

Model Architecture and Training

The baseline classifier used a RandomForestClassifier from the Scikit-learn library. As an ensemble technique, it builds a number of decision trees and combines the outputs to improve the generalization of the model while mitigating overfitting. Based on empirical observation, 400 estimators (trees) were set because performance did not improve beyond this. With additional trees, performance slightly declined and the F1 score diminished. The input features were the TF-IDF vectors processed earlier. No further tuning of hyperparameters such as maximum depth and minimum sample split were implemented in this version. Regardless of its simplistic nature, this baseline achieved a score of 0.78363 on the Kaggle public leaderboard, proving that the preprocessing and vectorization pipelines were indeed effective.

In order to make an aggressive change that improves performance, we moved to a transformer-based model by fine-tuning the bert-base-uncased model with HuggingFace Transformes and PyTorch libraries. This model has a stack of transformer layers with multi-head self-attention and feedforward networks that provide rich representations of elements in the input sequence. A binary classification head (a feedforward layer with softmax activation) is added to BERT's final hidden state of the [CLS] token.

We fine-tuned BERT with the following settings:

- Optimizer: AdamW (Adam with weight decay)
- Learning Rate: 2e-5
- Batch Size: 16
- Loss Function: CrossEntropyLoss for binary classification
- Max Sequence Length: 128 tokens
- Epochs Trained: 1 to 4

When training the model, we allocated 90% of the labeled data for training and set aside 10% for validation. We analyzed accuracy for each epoch and noticed performance peaks at epoch 3, with slight reductions in accuracy afterwards likely due to overfitting. The BERT model performed considerably better at capturing the context and syntax of tweets compared to the baseline model which used TF-IDF. This is attributed to attention mechanisms and the use of pre-trained embeddings.

Experimental Results

Baseline Model

The Random Forest classifier model performed exceptionally as a traditional classifier with an accuracy of 0.78363. But it made use of sparse bag-of-words features which limited the model's ability to generalize with text that relies more heavily on context or nuanced meaning.

BERT Model

Epoch	Accuracy
1	0.84216
2	0.83174
3	0.82194
4	0.81887

Performance peaked at Epoch 1, showing the best generalization. Epoch 2 saw a decline, suggesting overfitting.

Discussion and Conclusion

This demonstrates how BERT and other transformer-based models have the edge over traditional models in the domain of natural language processing. This is in conjunction with the baseline Random Forest model that, while performing well owing to proper preprocessing and TF-IDF usage, found it hard to compete against the contextual learning of BERT.

Takeaways:

- For classical models, TF-IDF offers an impressive performance but lacks the ability to capture context.
- Enhancements made to BERT, such as embeddings, allow for a better understanding of the text with less preprocessing work required.
- Using too many epochs leads to overfitting. Early stopping or the use of schedulers can rectify this issue.

Future Work:

- For a boost in accuracy and efficiency, experiment with DistilBERT, RoBERTa, and ALBERT.
- Add early stopping rules or learning rate schedulers to better optimize performance.
- Implement ensemble models or augment the data to increase the model's effectiveness.

References

- *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](https://aclanthology.org/N19-1423/) (Devlin et al., NAACL 2019)Pedregosa, F., et al. (2011).*

- *Transformers: State-of-the-Art Natural Language Processing](https://aclanthology.org/2020.emnlp-demos.6/) (Wolf et al., EMNLP 2020)*
- *Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1810.04805*