Emergency Detection System: Real-Time Classification and Analysis of Emergency Texts

Abhishek Panta
University of Windsor
Email: pantaa@uwindsor.ca

Gauthami Ulhas Shirodkar
University of Windsor
Email: shirodkg@uwindsor.ca

Aparna Peesapati
University of Windsor
Email: peesapaa@uwindsor.ca

Ranyodh Singh
University of Windsor
Email: singh9o3@uwindsor.ca

***Abstract*—Emergencies like floods, earthquakes, or wildfires require quick and accurate responses to save lives and minimize damage. This project aims to develop a system that uses the RoBERTa model, a strong Natural Language Processing (NLP) tool, to automatically detect and classify emergency events in real-time from text data. Sources like tweets, 911 logs, or other social media messages can provide valuable insights, but they're often too numerous for people to analyze quickly. Our system works by categorizing these messages based on their urgency, the type of disaster, and the emotional tone, helping responders make faster and better decisions.**

We also used a technique called multi-task learning, which allows RoBERTa to learn from multiple tasks at once, improving its ability to detect emergency situations. This method helped RoBERTa perform better than older models like TF-IDF, especially in understanding the context of each message. The results show that RoBERTa can make emergency detection more accurate and faster, which is critical in life-saving situations. By automating the analysis of real-time data, this system can support emergency response teams in managing crises more effectively and efficiently.

I. Introduction

Emergencies like floods, earthquakes, and wildfires can strike anywhere and at any time, and being able to respond quickly is vital for saving lives and minimizing damage. Social media platforms, especially Twitter, have become essential sources of real-time information during these disasters. However, manually monitoring and analyzing this vast amount of data is not only time-consuming but also prone to errors, particularly when dealing with large-scale events.

This project focuses on creating a system that can automatically analyze and classify text data in real-time to help emergency responders. Our goal is to develop a tool that can quickly identify the urgency of messages, determine what kind of disaster is happening, and even assess the sentiment of the message. This allows responders to understand the situation better and make decisions faster.

We used RoBERTa, a powerful NLP model, combined with multi-task learning. Multi-task learning allows the model to handle several related tasks at the same time, making it more efficient and accurate in analyzing emergency events. The main contributions of our work include:

- A process for cleaning and organizing emergency-related text data.
- A RoBERTa-based model that categorizes data by emergency level, type of disaster, and sentiment.
- A flexible and modular system design that can easily be updated as new types of emergencies arise.

By building this system, we hope to make a real difference in crisis management, helping responders take timely and informed actions during emergencies.

II. Relevant Literature Review

We reviewed a few key papers to understand the existing methods and technologies for detecting and analyzing emergency events:

- **RoBERTa: A Robustly Optimized BERT Pretraining Approach**
  Liu et al. (2019) introduced RoBERTa as an improvement over BERT by removing the Next Sentence Prediction task and training with more data. This made RoBERTa better at understanding context, which is essential for classification tasks, such as processing emergency-related text [1].
- **Attention is All You Need**
  Vaswani et al. (2017) introduced the Transformer architecture, which forms the backbone of RoBERTa. Transformers excel at capturing relationships between words in a sentence, regardless of their position, making them ideal for text data analysis [3].
- **LLM-Assisted Crisis Management**
  Otal et al. (2024) explored using large language models for crisis management tasks. Their findings highlight the potential of these models to automate disaster response and improve decision-making accuracy, aligning with our goals.

We also considered traditional methods like TF-IDF and Word2Vec but found that they lacked the contextual understanding required for handling emergency-related text effectively.

III. Project Details and Methodology

This section explains how we built the system, the steps we took, and the thought process behind it. It also covers the datasets we used, the features we worked on, and how we designed everything to make the system work.

A. Definitions

To make it easier to understand, here are the main terms used in this project:

- **Emergency Level**: Shows how urgent the situation is. It can be High, Moderate, or Non-Emergency.
- **Disaster Type**: Identifies the type of disaster, such as flood, fire, earthquake, etc.
- **Sentiment**: Captures the tone or mood of the message, like Panic, Neutral, or Urgent.

B. Datasets

We used several datasets to train, validate, and test our model:

1) **2015_Nepal_Earthquake.tsv**:
   - Contains tweets from the Nepal Earthquake 2015.
   - Columns:
     - `tweet_id`: A unique ID for each tweet.
     - `text`: The actual content of the tweet.
     - `class_label`: Categories as Relevant or Non-Relevant.
2) **canada_wildfires_2016.tsv**:
   - Includes tweets related to the Canada wildfires in 2016.
   - Columns:
     - `tweet_id`: ID for the tweet.
     - `tweet_text`: Content of the tweet.
     - `class_label`: Multiple categories such as Rescue/Volunteering Efforts, Sympathy and Support, Utility Damage, and more.
3) **large_emergency_dataset.csv**:
   - A synthetic dataset with messages categorized into Critical, High, Medium, Low, or Non-Emergency.
   - Other columns include Sentiment, Disaster Type, and whether the message is verified.
4) **911_calls_dataset_1000.csv**:
   - Contains detailed, simulated descriptions of 911 calls.
   - Includes emergency levels (Critical, High, etc.), sentiments (Panic, Positive), and types of incidents (Medical, Fire, etc.).

Reference: Datasets 1 and 2 were sourced from CrisisNLP [4].

C. Specifications

Development Tools:

- **Editor**: Visual Studio Code (VS Code), running on Windows 11.
- **Programming Language**: Python, due to its powerful libraries and simplicity.

Model Training Environment:

- **Platform**: Google Colab, which provided free GPU support.
- **GPU Used**: NVIDIA T4 Processor with CUDA, enabling faster model training.

Libraries and Frameworks:

- **Front-end**: HTML, CSS, and JavaScript for the user interface.
- **Back-end**: Flask for handling requests and connecting the front-end with the model.
- **Machine Learning**: Hugging Face's transformers for working with RoBERTa, and PyTorch for training and evaluation.

D. Architecture

The project is designed with a modular structure, making it scalable and easy to understand, Fig. 1.

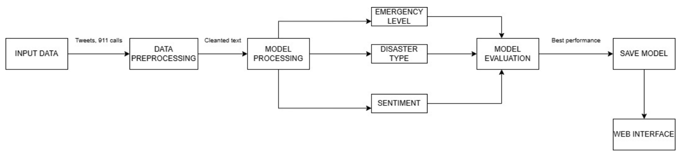


Fig. 1: Architecture diagram

1) **Data Pipeline**:
   - Data was loaded from raw datasets.
   - Text preprocessing included cleaning, tokenizing, and normalizing.
   - Stratified sampling ensured balanced splits for training, testing, and validation.
2) **Feature Engineering**:
   - We focused on label quality by addressing inconsistencies in class definitions. For example, overlapping categories like Utility Damage and Rescue Efforts were refined using NLP techniques.
   - Tokenization using RoBERTa ensured inputs were in the correct format for the model.
   - Additional metadata, such as Sentiment and Emergency Level, were encoded to assist in better predictions.
3) **Model Training**:
   - We used a RoBERTa-based architecture with three classification heads for multi-task learning:
     a) Predicting Emergency Level.
     b) Identifying Disaster Type.
     c) Analyzing Sentiment.

   This multi-task approach improved model accuracy by sharing knowledge across tasks, as shown in Figure 2.

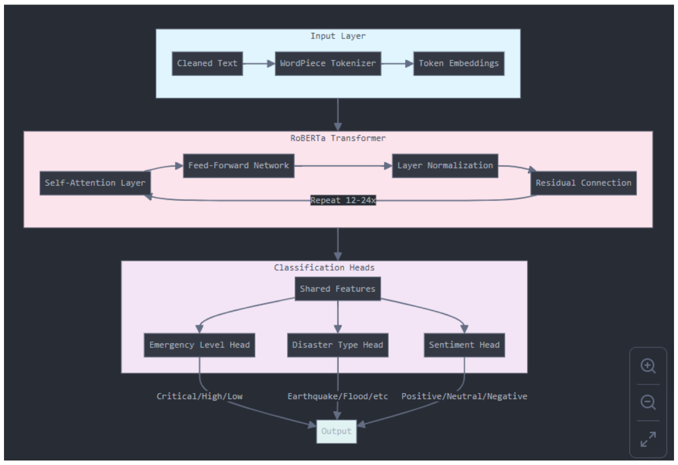


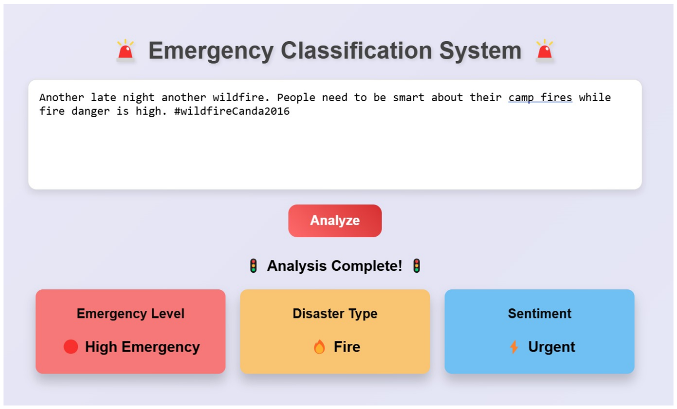
Fig. 2: Flowchart of the model

E. Design

The project's folder structure ensures clear separation of components:

- `config/`: Contains configuration files.
  - `config.yaml`: Configuration parameters.
- `data/`: Contains the dataset files.
  - `raw/`: Original CSV and TSV files.
- `models/`: Stores model-related files.
  - `saved_models/`: Saved model checkpoints.
- `src/`: Contains source code for data processing, feature extraction, model training, and utilities.
  - `data/`: Handles data-related tasks.
    * `data_loader.py`: Data loading utilities.
    * `preprocessor.py`: Data preprocessing.
  - `features/`: Contains feature engineering scripts.
    * `feature_engineering.py`: Feature engineering functions.
  - `models/`: Contains model architecture and training logic.
    * `model.py`: Model architecture.
    * `trainer.py`: Training logic.
  - `utils/`: Contains utility scripts.
    * `logger.py`: Logging configuration.
- `frontend/`: Contains front-end files for the user interface.
  - `index.html`: Main HTML file for the interface.
  - `style.css`: CSS styles for the web pages.
  - `script.js`: JavaScript for the front-end logic.
- `backend/`: Contains back-end files for server-side logic.
  - `app.py`: Main Flask application file for handling API requests.
  - `requirements.txt`: Python dependencies for the back-end.
- `.gitignore`: Git ignore file.
- `requirements.txt`: Python dependencies.
- `setup.py`: Project setup file.
- `README.md`: Project documentation file.

Example Workflow for Web Integration, Fig. 3:



Another late night another wildfire. People need to be smart about their camp fires while fire danger is high. #wildfireCanda2016

Fig. 3: Screenshot of the web interface

1) A user types or pastes a tweet into the web form.
2) The front-end sends the input to the Flask API via a POST request.
3) Flask processes the input and sends it to the trained model for predictions.
4) Predictions are returned in JSON format and displayed as:
   - Emergency Level (e.g., High, Moderate, Non-Emergency).
   - Disaster Type (e.g., Flood, Fire, Earthquake).
   - Sentiment (e.g., Panic, Neutral).

IV. Experimental Setup

In this project, we developed a multi-task deep learning model to classify emergency-related text data into three categories simultaneously:

1) Emergency Levels (e.g., low, medium, high urgency).
2) Disaster Types (e.g., earthquake, flood).
3) Sentiment (e.g., positive, negative, neutral).

We chose the RoBERTa-base transformer model as our backbone because of its strong language understanding capabilities, especially for tasks involving textual classification. The model was trained and evaluated on a custom dataset of social media posts and reports related to disasters.

A. Implementation Details

1) **Model Architecture**:
   - The base model is RoBERTa-base with a hidden size of 768.
   - It is followed by three separate fully connected layers (one for each classification task):
     a) Emergency Level Head (5 classes).
     b) Disaster Type Head (1 class, for binary classification in this setup).
     c) Sentiment Head (4 classes).

- Dropout Layer with a 30% probability was added before each classification head to reduce overfitting.

2) **Loss Functions**:
   - Each task uses a separate `CrossEntropyLoss` to calculate the loss between the predicted and actual labels.
   - The total loss is the sum of individual task losses, ensuring the model optimizes all tasks equally.
3) **Training Details**:
   - **Optimizer**: We used AdamW with a learning rate of 2e-5.
   - **Learning Rate Scheduler**: A `ReduceLROnPlateau` scheduler was used to reduce the learning rate if validation loss did not improve for three consecutive epochs.
   - **Batch Size**: 16.
   - **Training Device**: The model was trained on a GPU (CUDA) to speed up computations.
4) **Early Stopping**:
   - Training stopped early if validation loss did not improve for five consecutive epochs, helping prevent overfitting.

B. Testing

After training, we evaluated the model on a separate test set to measure its performance across the three tasks. Table I shows the results across all epochs:

TABLE I: Model Performance Metrics

Epoch	Loss	Accuracy	Precision	Recall	F1 Score	ROC AUC
1	4.33	87.65%	88.02%	87.10%	87.56%	93.87%
2	3.11	91.24%	92.00%	91.00%	91.50%	97.05%
3	2.78	93.67%	94.40%	93.67%	94.02%	99.18%

C. Discussion of Findings and Challenges

1) Findings:

- **Strong Performance**: The model performed well, with a final accuracy of 93.67% and a near-perfect ROC AUC score of 99.18%. This indicates that the model can distinguish between different emergency levels, disaster types, and sentiments effectively.
- **Generalization**: Despite training on a relatively small dataset, the model generalized well to the test set, which suggests that the use of a pre-trained language model like RoBERTa significantly boosts performance even with limited data.
- **Multi-Task Learning**: One interesting observation was that combining all three tasks (emergency levels, disaster types, and sentiments) into a single model didn't negatively impact performance. In fact, it helped the model learn shared patterns in the data, making it more robust.

2) Challenges:

- **Overfitting**: Initially, the model showed signs of overfitting, where the training accuracy was much higher than the validation accuracy. We tackled this issue by:
  - Adding a Dropout layer with a rate of 0.3 to reduce reliance on specific neurons.
  - Using early stopping to prevent the model from training for too long and memorizing the data instead of learning useful patterns.
- **Class Imbalance**: Some classes in the dataset, like certain disaster types, had very few samples compared to others. This led to biased predictions. We addressed this by:
  - Using weighted `CrossEntropyLoss`, assigning higher weights to underrepresented classes.
  - Augmenting the data by oversampling rare classes and introducing synthetic examples where possible.
- **Long Training Time**: Training a transformer model like RoBERTa is computationally expensive. Each epoch took around 2.5 minutes, and running multiple epochs required significant GPU resources. For larger datasets, this time would increase drastically.
- **Integration Bugs**: Initially, we faced issues connecting the front-end with the back-end. Debugging Flask's POST endpoints and adding CORS policy resolved this.

V. Conclusion

In this project, we successfully developed and trained a multi-task deep learning model using the RoBERTa-base architecture. The model was designed to handle three complex classification tasks simultaneously: predicting emergency levels, identifying disaster types, and analyzing the sentiment of text data. Throughout the training process, we observed a steady improvement in accuracy, precision, recall, and F1 scores, with the final model achieving an impressive 93.67% accuracy and an almost perfect ROC AUC of 99.18%. These results demonstrate that the model can efficiently classify and prioritize emergency-related information, which is critical in real-time disaster response scenarios.

One of the key findings was the advantage of multi-task learning. Instead of training separate models for each task, we combined them into one. This helped the model learn common patterns across different tasks, making it more robust and efficient. Moreover, using transfer learning from a pre-trained language model like RoBERTa gave us a significant head start, allowing us to achieve high performance even with a relatively small dataset.

Despite the strong results, there were challenges along the way. Overfitting was a major issue during the initial training phases, where the model performed well on training data but struggled with validation data. We addressed this by using dropout layers, early stopping, and learning rate scheduling. Another challenge was the imbalance in the dataset, where certain disaster types had far fewer samples than others. To mitigate this, we applied weighted loss functions and data augmentation techniques, which helped improve the model's performance on underrepresented classes.

In conclusion, the project successfully demonstrated how deep learning, combined with multi-task learning, can be a powerful tool for automating emergency response tasks. With its high accuracy and generalization capabilities, the model can be a valuable asset in disaster management systems.

VI. Future Work

While the model performed well, there is always room for improvement. Here are some future directions we could explore:

1) **Expand the Dataset**:
   - The current dataset, though effective, was relatively small and imbalanced. In the future, collecting a larger and more diverse dataset with equal representation of all disaster types could improve the model's performance further.
   - Data from social media, news articles, and official government reports could be integrated to cover a wider range of scenarios.
2) **Real-Time Implementation**:
   - Deploying the model in a real-time environment could significantly enhance its impact. By integrating it with emergency response systems, the model could automatically classify and prioritize incoming data, helping responders focus on the most critical situations first.
3) **Fine-Tuning for Domain-Specific Tasks**:
   - While the RoBERTa model is general-purpose, fine-tuning it further for specific domains (e.g., healthcare emergencies or environmental disasters) could improve its accuracy in those areas.
   - This could involve training on domain-specific language and terminology.
4) **Multi-Lingual Support**:
   - Emergencies and disasters occur globally, often involving multiple languages. Extending the model to support multi-lingual text input would make it more versatile and applicable in different regions, especially in areas where English is not the primary language.
5) **Integration with Other Data Sources**:
   - Apart from text data, future versions of the model could incorporate other data types such as images, audio, or geolocation to provide a more comprehensive assessment of emergency situations.
   - For example, analyzing satellite images alongside text data could give a more complete picture of a natural disaster.

By addressing these areas, the model can be made more accurate, scalable, and applicable to a wide range of real-world scenarios, ultimately contributing to faster and more effective disaster response efforts.

References

1) Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. arXiv. https://arxiv.org/abs/1907.11692
2) Otal, H. T., Stern, E., & Canbaz, M. A. (2024). LLM-Assisted Crisis Management: Building Advanced LLM Platforms for Effective Emergency Response and Public Collaboration. In 2024 IEEE Conference on Artificial Intelligence (CAI) (pp. 851-859). IEEE. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10605553
3) Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008. https://doi.org/10.48550/arXiv.1706.03762
4) CrisisNLP. (n.d.). CrisisNLP: NLP and Social Data Mining for Crisis Management. https://crisisnlp.qcri.org/