Disaster Message Classification and Response System

Group Name: The Tokenizers
Team Members: Ryan Hsieh, Salvador Robles, Collin Wolfhope

1. Introduction:

Recent flooding events in Texas have shown the importance and the big challenge in disaster response. Emergency authorities are often overwhelmed by the huge volume of different types of reports such as social media posts. These messages contain vital information for efficiently and accurately creating a response to a disaster event, but they often are unstructured and difficult to process as well as being very slow and tedious for relevant authorities to classify the disaster type for each message.

The Disaster Response Messages dataset from Hugging Face contains tens of thousands of real disaster text messages, like earthquakes, floods, and humanitarian crisis reports. Each message has multiple binary labels corresponding to needs such as food, shelter, water, or aid. Using NLP and ML classification techniques we can create a system that uses this dataset as training data to automatically predict which resources a given disaster message is requesting. Note that our model supports multi-label classification as a single message can match to multiple different labels. For example, “My house got destroyed, we need somewhere to stay and we need to eat.” which corresponds to both food and shelter related.

Our system will focus on four critical labels (water, food, aid related, and shelter) and will be trained using different vectorizer approaches and ML algorithms. We also experimented with a Retrieval Augmented Generation (RAG) system to answer domain questions such as “Where are the current flood events happening?” by searching through the classified messages and generating a response using an LLM. This will help to potentially route messages to the appropriate humanitarian response emergency teams so that affected communities could have a swifter response. Even small improvements in routing speed can make a huge difference during natural disasters.

2. Data Description:

2.1 Dataset

Our data comes from the Disaster Response Messages dataset from Hugging Face, collected from multiple disaster events including earthquakes, tsunamis, flood, hurricanes, and

humanitarian crises. More specifically, it includes events like: earthquake in Haiti in 2010, floods in Pakistan in 2010, etc. This dataset contains more than 26,000 multilingual messages across the training, testing, and validation set. Each data point includes a message in English (which is our main input), the original message, and 36 possible disaster categories overall, for example, if the message is related to shelter or not.

Train	**Test**	**Validation**
21046	2629	2573

Table 1: Train/Test/Validation split sizes

For this project, we narrowed our focus and selected four disaster labels (Water, Food, Shelter, Aid Related) which in our belief represent different high-impact categories that are critical for immediate response. Since we are training a single model for each label, we can easily expand this project to cover all labels, but due to time and computational constraints we will stick with these four as a proof of concept.

2.2 Label Description

As said before, we have selected these categories: Aid Related, Water, Food, and Shelter. Table 2 below summarizes the meaning of each label, why it is important in disaster response, and how many positive examples appear in our dataset for each of them.

Label	**Meaning**	**Importance**	**# Positive Examples**
Aid Related	Captures general requests for help, support, or assistance.	This is the broadest label and often serves as an indicator that information has actionable information.	8685
Water	Describes a lack of clean water or issues with water supply.	Access to safe water is critical for survival and sanitation.	1321
Food	Messages mentioning hunger or food shortages.	Ensuring food access is a core part of emergency relief logistics	2329
Shelter	Messages about destroyed homes, unsafe living conditions	Identifies people experiencing homelessness or displacement.	1878

Table 2: Description of selected labels.

These label counts clearly show some class imbalance, with Aid Related being the category that appears more often, while Water related accounts for nearly one fourth of the most representative class. Having labels with fewer positive examples can impact and be challenging for machine learning models.

To better illustrate how these labels appear in real disaster messages, we show two real examples from the dataset along with their assigned categories. This highlights the multi-label nature of the problem, where individuals communicate multiple urgent needs at the same time.

Message	**Labels**
Let’s do it together, need food in Delma 75, in didine area.	Water and Aid Related
We live in La Pleine. Since Tuesday we've been sleeping on the street. Hunger is killing us.	Shelter, Food, and Aid Related

Table 3: Examples of Message and Labels in the dataset.

2.2 Preprocessing & Feature Engineering

For text preprocessing, messages were lowercase and punctuation and duplicated whitespace was removed. After simple cleaning, we focused on different ways to represent our data going from text to a numerical representation that will be able to be used as an input to a Machine Learning model. We are treating this vectorization process as one of the main components of our experimentation.

We experimented with two Bag of Words vectorizers: Count Vectorizer and TF-IDF, each with three different vocabulary sizes (1000, 3000, and 5000 maximum features). The idea here is to examine how feature dimensionality affects model performance and if the increase of dimension is worth the computational effort. We also compared these traditional vectorizers with transformer embeddings from three different models: all-MiniLM-L6-v2, all-mpnet-base-v2, and BAAI/bge-large-en-v1.5. Transformer embeddings capture semantic meaning in messages, which is important due to messages describing the same emergency can use different vocabulary.

Comparison between the performance of our feature engineering pipelines is one of our goals, we want to determine which representation works best for disaster-related message classification.

3. Methodology:

3.0 Quick Overview

- **Exploratory Data Analysis (EDA)**
- **Data Preprocessing**
    - Count Vectorizer (1000, 3000, 5000)
    - TF-IDF Vectorizer (1000, 3000, 5000)

    - Latent Dirichlet Allocation (LDA) (200 Topics) + TF-IDF (1000, 3000, 5000)
        - Topic Modeling
    - BERT Embeddings (BAAIs)
- **Model Implementation**
    - One Vs Rest for Embedding splits
    - Machine Learning algorithms
        - Logistic Regression
        - Linear SVM
        - SGD Classifier
        - Multinomial NB
        - MLP Classifier (Neural Network)
            - Encoder-decoder models
                - BERT + Fine Tuning
- **Evaluation**
    - F1, Precision, Recall
    - Macro F1, Micro F1
    - Hamming Loss
- **RAG**
    - Retrieval of relevant messages
    - LLM Summarization and actionability

3.1 Task & Approach

We will define our task as a Multi-Label Binary Classification problem, where we have to optimize for the performance across all our different labels which are binary (either Yes or No). Each of our four labels is treated as an independent binary decision, the model predicts either Yes or No for each label separately. Because all labels represent essential humanitarian needs, we consider them equally important, and our goal is to build machine learning models that improve performance across all categories without sacrificing one label to improve another.

To achieve this, we train one independent classifier per label and evaluate each classifier on its own performance (One vs Rest Classifier does exactly this). This approach fits perfectly on the nature of our Multi-Label Classification problem, where we allow a model to learn specific patterns relevant to its own category.

3.2 Classification Algorithms

To build our Multi-Label Binary Classification system, we created a variety of experiments by testing different machine learning algorithms that are good for text classification, this will serve as very good strong baselines next to our Fine-Tuning approach (more in next section). Each algorithm was paired with every single one of our feature engineering methods (Count Vectorizer, TF-IDF, and Transformer Embeddings). This resulted in a wide variety of model configurations and comparing them allowed us to see clearly what machine learning and feature engineering method works best.

For the following machine learning algorithms we used a One vs Rest classification strategy, where we train a separate binary classifier for each label. This approach fits well with our problem and dataset since our labels represent semantically different needs and would need to find different linguistic patterns. This also ensures a consistent training framework across all experiments.

3.2.1 Logistic Regression

Logistic Regression is a strong baseline in binary classification, and it works great with high dimensional text classification tasks, especially when it is combined with Count Vectorizers and TF-IDF representations. It learns its decision boundaries through maximum likelihood estimation and by applying a sigmoid function to produce probability estimates for each class.

3.2.2 Linear SVM

Linear Support Vector Machines perform relatively well on text data and high-dimensional vector spaces. It tries to find the most optimal hyperplane that separates the two classes, by maximizing the decision margin. Compared to Logistic Regression, it tends to be more robust to outliers due to its support vector decision boundary logic.

3.2.3 SGD Classifier

The SGD Classifier trains a linear model using stochastic gradient descent, updating weights based on mini-batches rather than the full dataset which makes it more efficient on big feature spaces. It can approximate algorithms such as Linear SVM while giving faster training speed.

3.2.4 Multinomial Naive Bayes

This is a classic generative model that also does well in text classification tasks, this algorithm models the probability of observing word counts given each class. The core of this algorithm is the Bayes' theorem with the assumption (naive) that words (our features) are conditionally independent given the class label. This works well with Count Vectorizer and TF-IDF representations, but due to the nature of Transformer Embeddings where the embeddings can be negative values we won't be applying this algorithm to those feature engineering methods.

3.2.5 Neural Network

We also tested a Multi-Layer Perceptron, or so called Neural Network with three hidden layers of 512, 256 and 128 units using ReLU as activation function. This architecture allows the system to learn non-linear decision boundaries through composition and thanks to the backpropagation during training. We want to see if Neural Networks can outperform our other linear models.

```
models = {
  "LogisticRegression": OneVsRestClassifier(
    LogisticRegression(
        class_weight='balanced',
        C=1.0,
        random_state=42,
        solver='lbfgs', # small-medium datasets
        max_iter=5000
    )
  ),

  "LinearSVM": OneVsRestClassifier(
    LinearSVC(
        class_weight='balanced',
        C=1.0,
        random_state=42,
        dual=True, # n_samples > n_features
        max_iter=5000
    )
  ),

  "SGDClassifier": OneVsRestClassifier(
    SGDClassifier(
        class_weight='balanced',
        random_state=42,
        max_iter=5000
    )
  ),
```

```
"MultinomialNB": OneVsRestClassifier(
  MultinomialNB(
      alpha=1.0,
      fit_prior=True
  )
),
"NeuralNetwork": OneVsRestClassifier(
  MLPClassifier(
    hidden_layer_sizes=(512, 256, 128),
    activation='relu',
    solver='adam',
    learning_rate_init=0.001,
    batch_size=256,
    max_iter=20,
    shuffle=True,
    random_state=42,
    verbose=False
  )
)
```

Figure 1: List of Hyperparameters used for each ML algorithm.

3.3 Fine-Tuning BERT with Multi-Label Classification

Let us move away from traditional machine learning baselines and look into what has revolutionized natural language processing in recent years. In addition to our classical

classification approaches, we fine-tuned a Transformer based model (BERT-base-uncased) to solve our problem of Multi-Label Classification. Unlike our previous approaches that had vectorizers like TF-IDF, fine-tuning BET allows the model to learn directly from raw text messages. This more of than not results in stronger performance because BERT captures deep semantic relationships within sentences.

A fundamental limitation of vectorization is that they can not generalize to unseen vocabulary with similar meaning. For example a text message containing “drought”, if we never see this word in our training set, then TF-IDF assigns it zero weight across all documents, so there is no signal helping the classifier. But BERT overcomes this limitation through transfer learning, thanks to being trained on massive text corpora. Here, “drought” and “water scarcity” will have a similar semantic meaning.

3.3.1 End-to-End Learning

During our Fine-Tuning process, the raw disaster messages are tokenized using BERT’s tokenizer and then it is passed through the Transformer encoder. Instead of using a single linear classifier (like in our One vs Rest Classifier models) BERT will output a single vector of logits, one for each of our four labels.

We configure BERT such that: “problem_type = multi_label_classification” which will automatically apply a sigmoid activation function to each output unit and optimize the model using Binary Cross Entropy with Logits Loss. This treats each label as an independent binary decision.

```
# 4. Load Model
# problem_type="multi_label_classification" is crucial here!
model = AutoModelForSequenceClassification.from_pretrained(
    model_ckpt,
    num_labels=len(target_cols),
    problem_type="multi_label_classification"
)

# 5. Training Arguments
args = TrainingArguments(
    output_dir="bert_finetuned_disaster",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=learning_rate, # 2e-5
    per_device_train_batch_size=batch_size, # 8
    per_device_eval_batch_size=batch_size, # 8
    num_train_epochs=epochs, # 3
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    report_to="none"
)
```

Figure 2: Fine-Tuning configuration.

3.3.2 Advantages of BERT approach

This approach provides a variety of benefits compared to classical ML approaches, and it outperformed all the classical models across all our main metrics (see Results section for a quantitative comparison).

- We directly learn contextual embeddings without relying on vectorizers such as TF-IDF.
- Fine-Tuning captures subtle semantic clues in short disaster messages, going from urgency information to location details.
- We can train at the same time our four related classes, unlike with our ML approaches in which classifiers are trained independently.

3.4 Metrics

For each label we are going to consider our main metric to be the F1 score, which is the harmonic mean between Precision and Recall. We won't be looking at accuracy at all since it is very misleading due to the class imbalance problem in our data, we could achieve high accuracy simply by predicting everything to be a "No" which is the majority class.

- $TP_{micro} = \sum_{i=1}^{N} TP_i$
- $FP_{micro} = \sum_{i=1}^{N} FP_i$
- $FN_{micro} = \sum_{i=1}^{N} FN_i$
- $Precision_{micro} = \frac{TP_{micro}}{TP_{Micro} + FP_{micro}}$
- $Recall_{micro} = \frac{TP_{micro}}{TP_{Micro} + FN_{micro}}$
- $Micro\ F1 = \frac{2 \times Precision_{micro} \times Recall_{micro}}{Precision_{micro} + Recall_{micro}}$
- $Macro\ F1 = \frac{\sum_{i=1}^{n} F1\ Score_i}{n}$

- $Hamming\ Loss = \frac{number\ of\ misclassified\ labels}{(number\ of\ samples) \times (number\ of\ labels)}$

For our purposes, the primary metrics other than traditional performance metrics such as Precision, Recall, and F1 score that we rely on in this project include Micro F1, Macro F1, and Hamming Loss.

- Micro F1 measures the overall performance of the model by weighting the frequencies of true positives, false positives, and false negatives. It allows for a proportional view of all classes in scenarios where rare classes might not be represented enough in the data to be properly represented by traditional performance metrics
- Macro F1 measures the performance of the model on minority classes by highlighting the disparity between minority and majority labels. It is particularly useful for our purposes because of the imbalance in our source data.
- Hamming Loss measures the proportion of misclassified labels, making it effective in multiclass scenarios for evaluating classifier performance.

We will rely mostly on **Macro F1** as it targets minority classes which is perfect for targeting our class imbalance problem. Note that on our experiment results Hamming Loss and Macro F1 have relatively the same opinion on which of our methods are better, so for simplicity we can stick with Macro F1.

3.4 Retrieval-Augmented Generation (RAG) Prototype

Our RAG prototype serves as an extra layer on top of our classification models to allow coordinators to quickly synthesise report information to assist in data informed decision making. Upon prompting, the system returns information based on actual messages that can be used to scrutinize data/message trends or provide recommendations for the best course of action.

3.4.1 Why RAG in Disaster Response?

Retrieval-Augmented Generation is a framework that empowers and enhances Large Language Models (LLMs) by giving them access to external sources of knowledge, which could be domain-specific information at inference. Rather than just hoping that our LLM has all the answers in its internal knowledge, RAG retrieves documents from a database and provides context to generate a correct answer.

What we envision for our Classification Message System is that there is a RAG system on top of it, adding a second layer after our classifier. After the classification divides the incoming messages into four categories water, food, shelter, and aid related, each humanitarian response team can interact with the subset of messages relevant to their domain.

```
prompt = f"""
You are an intelligent Disaster Response Analyst.
Use the context messages provided below to answer the user's question.
Summarize the specific needs or situations found in the messages.

Context Messages:
{context_str}

Question: {question}

Answer:
"""
```

Figure 3: Prompt used in RAG system..

For example, a team responsible for water distribution may want to ask targeted questions like the following:

- "Are there specific neighborhoods reporting water scarcity?"
- "What locations are the most mentioned in water requests?"
- "Are there safety concerts about available water sources?"

A RAG system will enable us to answer these types of questions. It retrieves the most relevant messages from the water category and then feeds them into an LLM, which then creates a summary of the information and returns actionable information.

3.4.2 Implementation Details of RAG

To implement this retrieval workflow, our pipeline consists of: an embedding based retrieval system and an LLM for summarization. For the retrieval layer, we used the **all-MiniLM-L6-v2** SentenceTransformer to encode every disaster message in our dataset. These embeddings were stored in a FAISS index which is configured with an L2-distance search. Given a query like "What neighborhoods are reporting water scarcity?" we encode the query string (with the same embedding model) and then perform a top-k similarity search to retrieve the most relevant messages.

After retrieval, we feed both the query and the retrieved messages into an LLM **Phi-3.5-mini-instruct**. We created the prompt (shown in the above Figure) to behave like a disaster response analyst and summarize the retrieved messages and generate an actionable analysis and answer the given query. This can potentially allow humanitarian response emergency teams to interact with classified messages and then query them and get concise and actionable insights.

4. Results

Like previously mentioned, we are going to test each of our feature engineering methods with every one of the machine learning algorithms. First we are going to show performance of the different created systems using only Classical machine learning algorithms (using macro F1 and hamming loss as main metrics) and then we are going to expand our results by adding our last and best approach which is the Fine-Tuned BERT model.

4.1 Performance across Classical Machine Learning

For the following graphs we will show the performance of the best ML algorithm given a specific feature engineering method. For example, for TF-IDF with max features 1000, the best classifier algorithm was SGD Classifier, so we will only show the Macro F1 of that specific configuration (TF-IDF 1000 + SGD Classifier).

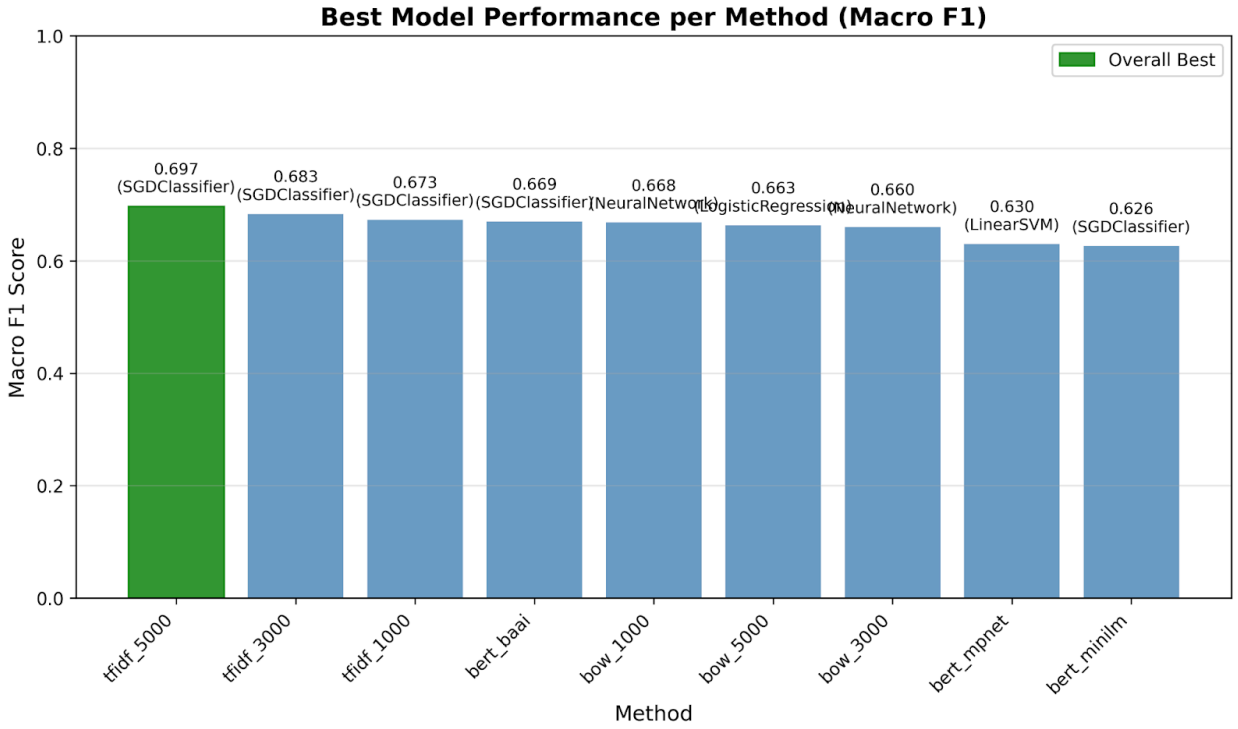


Figure 4: Performance across ML baselines using Macro F1 as the main metric.

In our original testing of our pipeline, we were able to isolate the best methods and models in terms of performance metrics; however we were not satisfied with the numbers that we were getting, so we knew that there would have to be some kind of improvements made to how we were either processing the data or running the models. Traditional performance metrics such as Precision, Recall, and F1 (Harmonic Mean) might not capture the entirety of what was going on without model performances in a multiclass scenario, however it does still provide some insight into the behavior of the model's outputs.

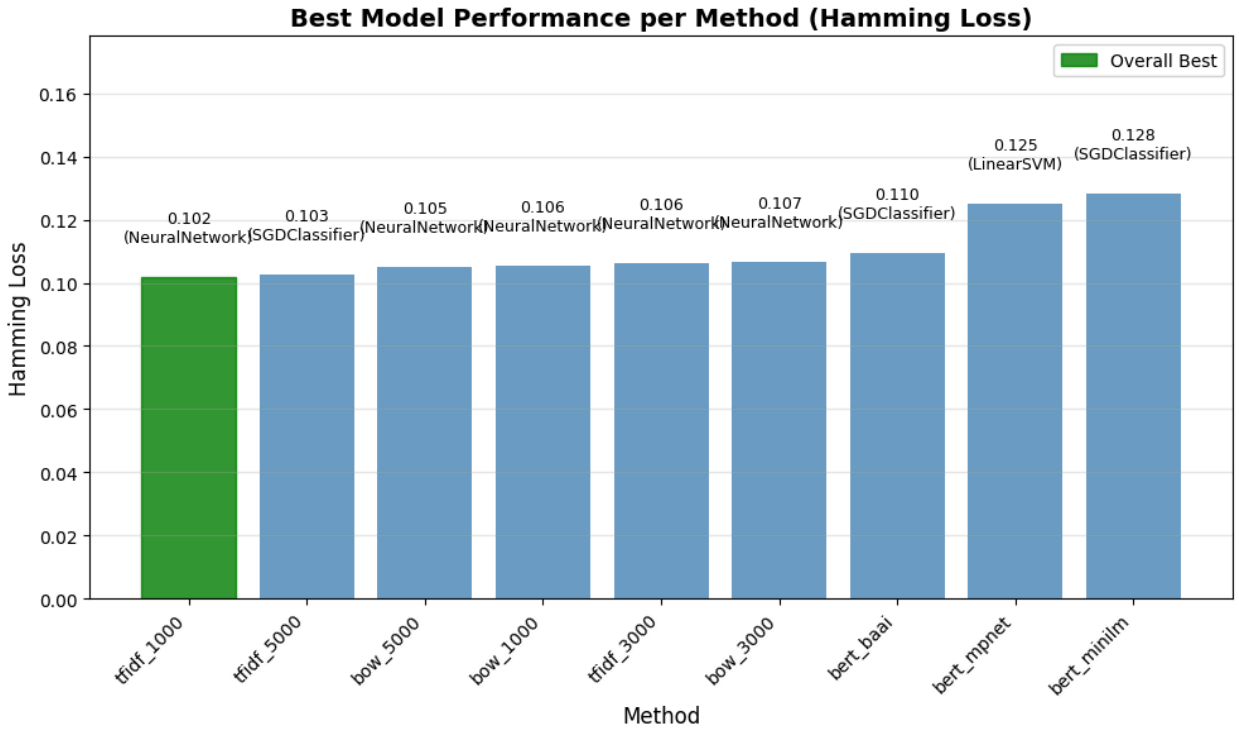


Figure 5: Performance across ML baselines using Hamming Loss as the main metric.

Stacking Hamming Loss (HL) on top of our traditional performance metrics verified that while the models seemed to be performing poorly, there was more going on under-the-hood. HL helped us key into how the models were performing in a more detailed way by evaluating the results of the models in a way that was tailored towards a multiclass environment by taking the proportion of correctly predicted labels into account. These results showed that while we might have had a macro f1 of less than .7, we were still somewhat successful overall in training the models to be able to predict most of the labels (~90%) right in a multiclassification setting.

That being said, this gave us a dilemma as to how best to select the best performing models as traditional performance metrics might seem to indicate to us that the best model by a landslide would be the SGDClassifier, however, based on the Hamming Loss, it could just as well be the (MLPClassifier) Neural Network.

The key distinction in the performance, for us, seemed to be how the models were assigning weights to their algorithms. Because we know that because of the data imbalance, the models tend towards overpredicting the negative labels, scrutinizing the traditional performance metrics and hamming loss seems to indicate that the SGDClassifier might perform marginally better in precision focused environments, while the neural network would perform better in recall focused environments. For a task like disaster type / aid need classification, perhaps a more recall attuned model such as the Neural Network would be more efficient at handling these types of task because of the need to prioritize catching all possible positive labels at the expense of over predicting.

4.2 Addition of Topic Modeling

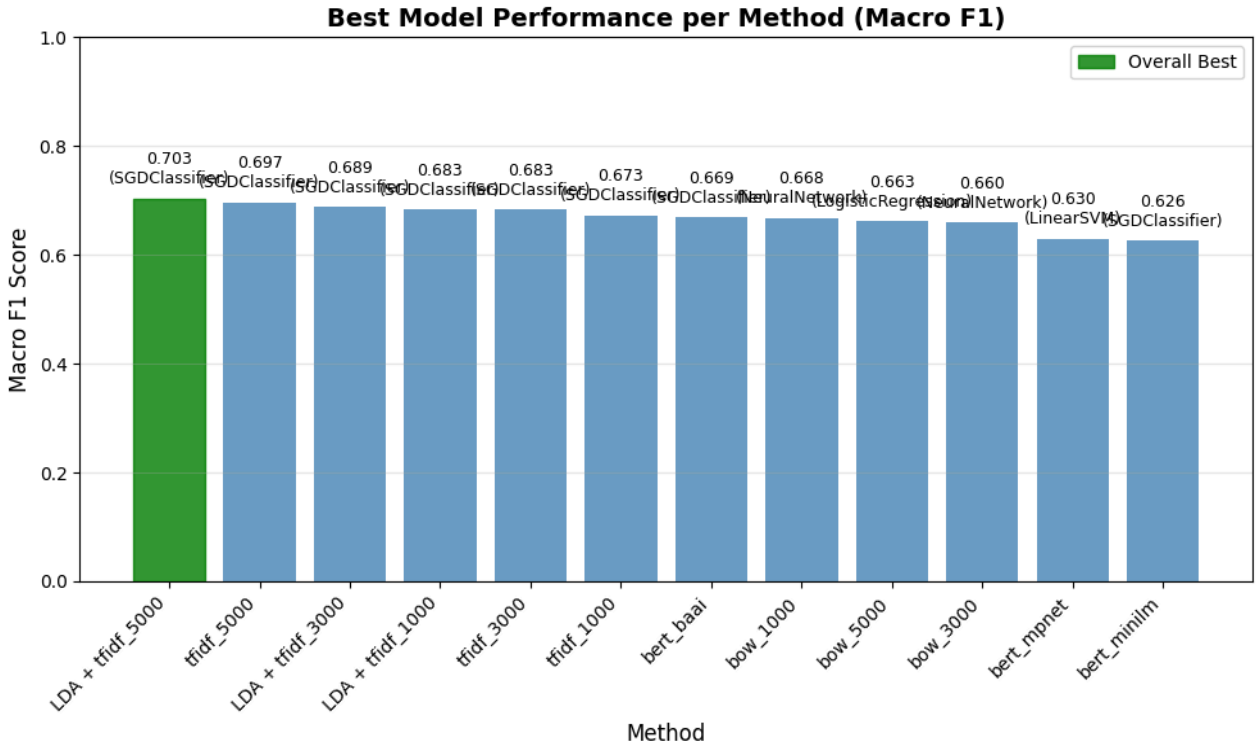


Figure 6: Performance across ML baselines with LDA included using Macro F1 as the main metric.

One solution that we tested for solving the issue of low performance metrics was the addition of a topic model via Latent Dirichlet Allocation (LDA). We varied the number of topics ranging from 50-200 where the greatest performance was gained at 200 topics. Overall the performance gain was rather minimal and suggests that topic modeling might not be providing enough of a solution to address underlying issues such as data imbalance.

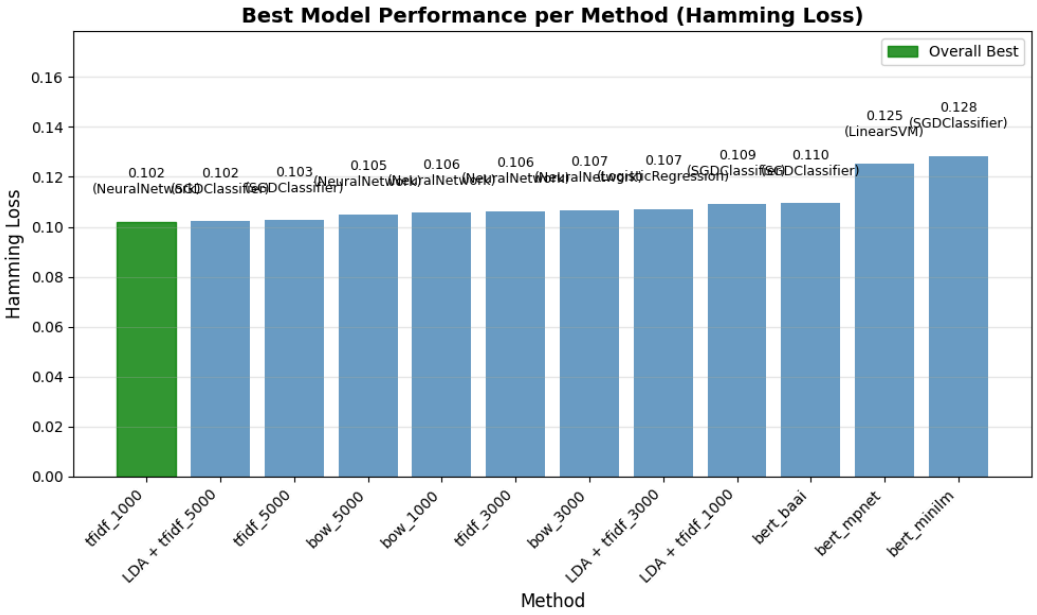


Figure 7: Performance across ML baselines with LDA included using Hamming Loss as the main metric.

This is further indicated by the HL being practically unchanged in terms of performance over prior tests. What this shows is that, while there is a greater change occurring in the Macro F1 score, there is not much changing in terms of the metrics that we care more about, such as recall and mislabeling.

4.3 Fine-Tuned BERT results (Best Approach)

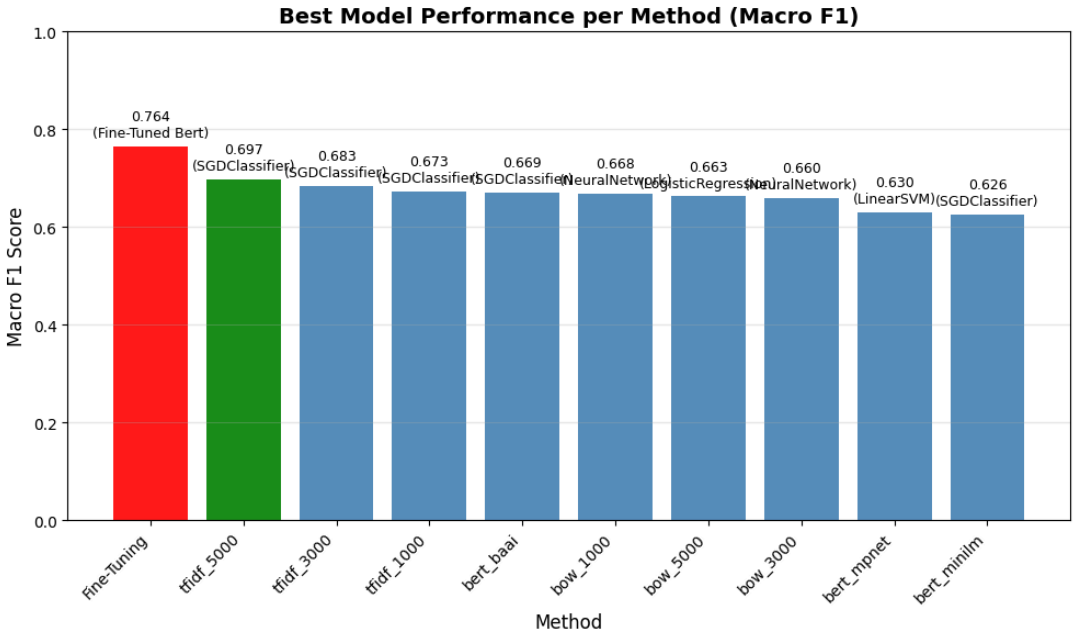


Figure 8: Performance across ML baselines vs Fine-Tuned BERT using Macro F1 as the main metric.

Iterating off of some of our earlier conclusions about model performance, we can see that a more complex and powerful combination of embeddings and model, such as BERT embeddings paired with a encoder-decoder (neural network based architecture) model, performs better than our original data splits and ensemble selection of models. The fine tuned BERT model is able to handle the rare cases with much greater efficacy and as such the precision and recall scores during the validation and testing increased considerably, which was apparent in the jump in the score of the Macro F1.

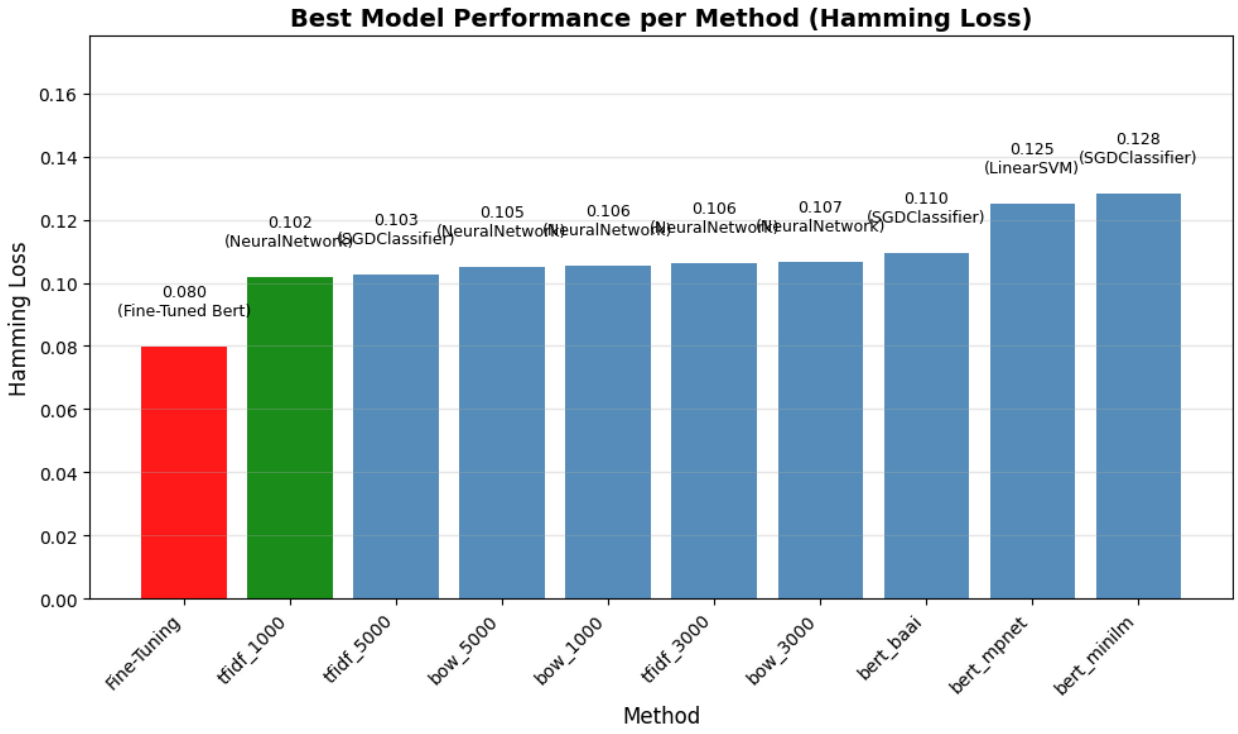


Figure 9: Performance across ML baselines vs Fine-Tuned BERT using Hamming Loss as the main metric.

Not only did we get an increase in the Macro F1, we also saw in Figure 9 a huge drop in the Hamming Loss. Given the struggles with data imbalance, it's no surprise why. The prior issue that we had with the ensemble models was that they struggled with predicting the rare classes and as such had lower recall scores (high FN) than were ideal. The fine tuned BERT model, on the other hand, was able to cut the prior scores by 20% demonstrating a dramatic decrease in the amount of misclassified labels, an indication that the model was able to handle the imbalance better.

4.4 RAG Qualitative Results

In this section we will show a few examples of our RAG system working with queries across our four different categories. The following shows how a RAG system can be helpful to disaster response teams, by retrieving relevant information and getting an actionable summary.

Query: Are there specific locations or neighborhoods mentioned in requests for water?

Found 7 relevant messages.

--- Retrieved Context (Samples) ---

1. where can we find water at petionville please ?

2. needs a more precise address information Seeking water to drink

3. Where can we find the water at PetionVille?

AI Analysis:

Yes, the context messages do indicate concerns regarding the safety and quality of the water being distributed. One message specifically advises residents in affected areas to avoid water from a contaminated river. This suggests that there is a risk of unsafe water, which could be due to contamination. The mention of a need for more precise address information

Query: What areas are reporting the most critical food shortages?

Found 7 relevant messages.

--- Retrieved Context (Samples) ---

1. Affected areas experience severe food shortages caused by obstructed access, drinking-water shortages due to unusable wells, lack of accommodation, communications and other factors.

2. There is no food distribution in the area where i live?

3. where is food being distributed today? i am a disaster victim in Sovona?

AI Analysis:

Sovona is the location where there is a concern about the current food distribution efforts. This concern is expressed through questions asking about the whereabouts of food distribution and how it

Querying: What specific medical supplies are hospitals or clinics requesting?...

Found 7 relevant messages.

--- Retrieved Context (Samples) ---

1. The health authorities have requested the health cluster group to provide medical supplies such as basic antibiotics, Oral Rehydration Salts, normal saline, dextrose saline, bandages, gauze, plaster and syringes.

2. I would also be happy to help hand out medical supplies if needed .

3. We're asking for water, medical supply, food

AI Analysis:

The hospitals or clinics are requesting specific medical supplies such as basic antibiotics, Oral Rehydration Salts, normal saline, dextrose saline, bandages, gauze, plaster, syringes, broad spectrum antibiotics, antiseptics, pain relief medication for trauma injuries, and anti-malarial drugs. Additionally, there is a need for sterilization equipment that can function without electricity, funds for outreach services, water purifiers, and surgical appliances and devices for amputees.

5. Conclusion:

In this project we showed how you can process messages from disaster events using various methods and use the generated features from this process to train an ensemble of different classification models ranging from regression models to neural networks.

Over the course of many iterations, improvements, and corrections, we have generated a set of performance metrics and visuals that can show the success of our data to model pipeline. In the end, the fine-tuned BERT model used in addition with the bert-base-uncased encoder-decoder model was the most effective solution for solving our classification task outperforming our ML baselines (ML models with vectorizer approach). With the BERT + fine tuning pipeline, we are able to produce predictions with a very high degree of reliability, where during our testing we were able to predict most multiclass scenarios with a hamming loss of approximately 0.08, which indicates that we are able to properly have the model distinguish between our labels.

The work that we present in this project is especially relevant given recent flooding disaster events throughout the state of Texas. Systems like the one we've designed here have potential to drastically speed up the amount of time that it takes for on-the-ground coordinators to sort through data. In disaster response scenarios, where every second could cost lives, this kind of system has the potential to provide meaningful improvement to existing operations.

Future work can build upon the pipeline that we've proposed and built in this project to expand the amount of labels being worked with, improving underlying data reliability issues, and introduce new datatypes as well as models better designed to handle those data types in order to provide a more informed set of predictions that can serve to better inform disaster coordinators.

6. Appendix: Q & A presentation discussions

What architecture would you design so that one task (e.g., disaster detection) can explicitly help another (e.g., downstream category classification)?

While it's not something that we directly tackle within the scope of our project, one potential architecture that could layer tasks is starting with a classification model making a prediction on the "_related" labels. There are a number of labels in the dataset, such as infrastructure, weather, etc., that have to do with larger categories. In this theoretical architecture, you could start by first classifying whether a message is a certain kind of message (whether it is related to; e.g., infrastructure) and then use that to select a more fine tuned model that is trained on more specific types of messaging. For example, if the message is predicted as "infrastructure_related", then it would then trigger a more fine tuned model related to infrastructure (that includes the vocabulary necessary to understand more domain-specific terminology) that would make predictions about what kind of infrastructure the message is discussing (labels: transport, buildings, electricity, hospitals, etc.). Ultimately, this is still a similar type of classification problem as the original

focus of the project, which is to send a large volume of report data into a classification funnel and output the reports to the correct destination / the proper coordinator / specialist who needs to view them during a disaster event without having to have someone manually inspect each report.

Why did you choose Hamming Loss as a core metric (p.8)? How does it complement Macro/Micro F1 in your multi-label setting?

Hamming Loss measures the fraction of incorrect individual labels out of the total number of labels. In a multi-label setting, where a single message can have multiple tags (e.g., [1, 0, 1, 0]), standard accuracy is too harsh (it requires a perfect match for all tags). Hamming Loss is more granular and forgiving, giving "partial credit" for the correctly predicted tags within a message. Hamming Loss provides a holistic error rate that accounts for both False Positives (predicting a tag when it's not there) and False Negatives (missing a tag). It serves as a "sanity check" to ensure that while optimizing for F1 (finding positives), the model isn't spamming incorrect labels just to increase recall.

On what basis do you think BERT underperformed in some settings (e.g., imbalance, insufficient fine-tuning epochs, domain shift)?

Generally speaking, the BERT + fine tuning ended up being our highest performing model in terms of performance metrics, however, there were some downsides to actually using the model. The biggest constraint was that the BERT embeddings and encoder-decoder model took substantially longer than the other embedders and models to compute, train, and evaluate. Additionally, without the downstream model fine tuning process, we saw that the embeddings usually underperformed (on our ensemble models) compared to some of the richer whole-document-context embedding methods such as tf-idf in terms of performance metrics on classification. Additionally, even though the BERT + fine tuned model performed better than the rest of our embedding + ensemble models, it also similarly struggled with recall because of the high number of entries causing data imbalance. As with the rest of our models, the high number of false entries in the dataset (for each label) meant that the BERT model has a propensity of over-predicting on the false entries (FN).

Could you provide more detailed dataset statistics i.e., label frequencies, average message length, multilingual distribution to contextualize model behavior?

From the following list we selected 4 labels to train our models on (**bolded**):

1. related: 20316
2. **aid_related: 10878**
3. weather_related: 7304
4. direct_report: 5081
5. request: 4480
6. other_aid: 3448
7. **food: 2930**
8. earthquake: 2455
9. storm: 2448
10. **shelter: 2319**
11. floods: 2158
12. medical_help: 2087
13. infrastructure_related: 1705
14. **water: 1674**
15. other_weather: 1376
16. buildings: 1335
17. medical_products: 1314
18. transport: 1203
19. death: 1196
20. other_infrastructure: 1151
21. refugees: 876
22. military: 860
23. search_and_rescue: 724
24. money: 604
25. electricity: 534
26. cold: 530
27. security: 471
28. clothing: 406
29. aid_centers: 309
30. missing_people: 299
31. hospitals: 283
32. fire: 282
33. tools: 159
34. shops: 120
35. offer: 119

The average message (by word) length for the selected columns is listed as follows:

1. The average length of **aid_related** messages is **28.51** words per message.
2. The average length of **water** messages is **39.17** words per message.
3. The average length of **food** messages is **32.63** words per message.
4. The average length of **shelter** messages is **36.34** words per message.

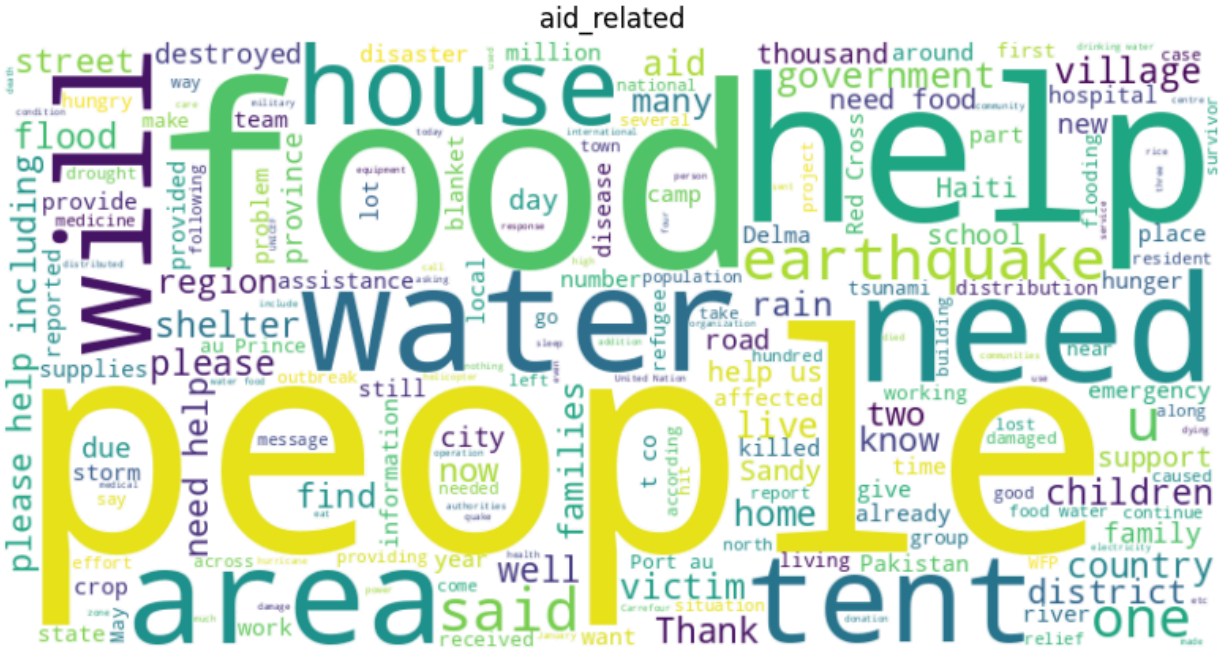
We determined that utilizing the multilingual column to inform model decision making would be unwise because there is an unknown quantity of different languages included in the “original” (language message) column, and would necessitate scrutinizing the entries to identify every present language before we could reasonably work with that data. The reason is that without prior knowledge of every language you would potentially be working with, there is no way to reliably tokenize and embed, because those kinds of functions often rely on knowing what language they’re working with because of vocabulary restraints. Instead we decided to rely on the provided english column “message”, which is the “original” column but already translated into English.

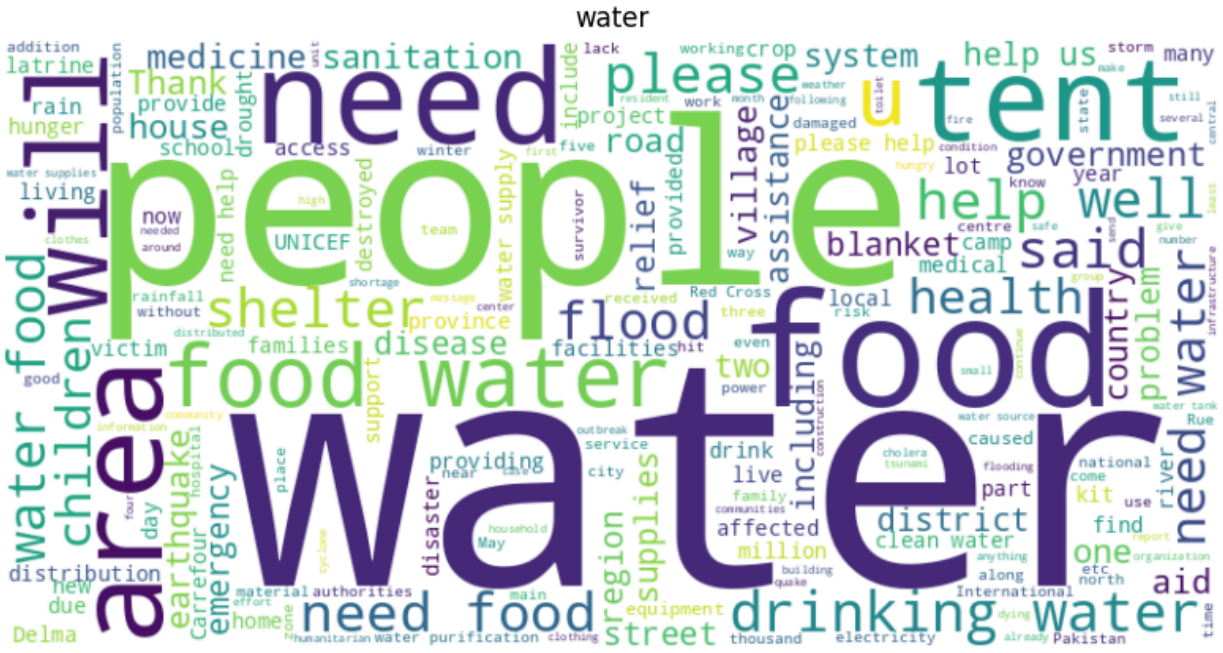
Can you expand on the RAG process (p.15)? Specifically, how do you ensure retrieval relevance before Phi-3.5 generates summaries?

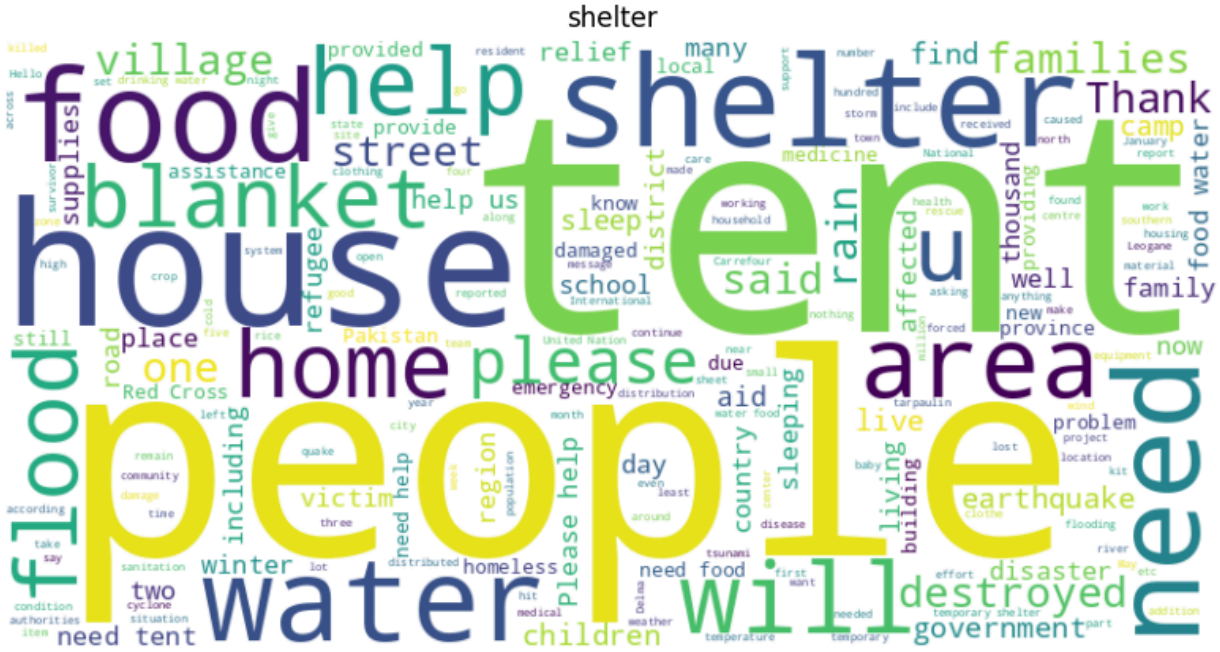
To ensure retrieval relevance within our RAG system, we implemented a semantic vector search pipeline rather than relying on keyword matching. We encoded over 21,000 disaster messages using the all-MiniLM-L6-v2 model and indexed them in a FAISS vector database. When a user submits a query, it is converted into the same vector space, and the system retrieves only the top k messages with the smallest Euclidean (L2) distance. This mathematical proximity ensures that the retrieved context is conceptually aligned with the user's intent, effectively filtering out noise before generation.Once relevance is established, we constrain the Phi-3.5 LLM's output through strict prompt engineering and parameter control. The retrieved messages are injected into a rigid system prompt that instructs the model to act solely as a "Disaster Response Analyst" and base its answers exclusively on the provided context. Furthermore, by setting a low generation temperature of 0.3, we reduce the model's creativity, ensuring the final summary is deterministic, factual, and grounded in the retrieved evidence.

Could you include additional visualizations (e.g., word clouds or topic distributions) to illustrate common disaster themes in the dataset?

Below are the word clouds for the labels that we selected for classification:







How do you envision deployment? Strictly text-based, or multimodal (images, sensor reports)? What is the expected runtime behavior under real emergency load?

The deployment of such a model would be constrained to scenarios dependent on data availability. For the sake of staying relevant to our project, we'll limit our discussions to strictly text-based data implementations; however, the pipeline for accomplishing this form of real-time, low latency data ingest to model pipeline, should in all (from a macro-view) cases, be uniform

across different types of data streams. The obvious limit to any type of classification model, is that it needs a reliable, timely, scrutinizable data stream, and in disaster scenarios, data quality is not guaranteed. For our purposes, ensuring at least a bare minimum of input would require us to build a data collection pipeline, consisting of storing user data, collected from specific, verified (integrity) data sources to then funnel that data into a featurization pipeline. Some examples of what this sort of textual data sources could look like are a text to speech collection of 911 calls or the reports from geolocated social media posts (research tends to show that this doesn't work in practice) or the output from a report app that is designed to aggregate posts from low-bandwidth / service outage zones. Assuming that the data collection process has been established, on a per event basis (as infrastructure is never uniform in disaster scenarios), it then becomes a problem of injecting the data into a feature engineering to model training pipeline, such as the one outlined in this project, and then shipping downstream predictions / calculations to relevant stakeholders (such as quickly processing a large volume of reports so that emergency responders know where to send aid/relief supplies). As mentioned earlier, this pipeline should work with other data types, such as running image data into a computer vision model for a disaster classification task or utilizing local sensors to feed data into a geolocation tuned classifier for risk zones to inform disaster response coordination. While these are interesting ideas that have potential to be expanded on, this project only covers textual data in scope.

Because of the nature of disasters, different contexts necessitate different priorities with what types of models are run. For our model pipeline that we've constructed, the system automatically selects the best performing model from an ensemble of different models. This works well for the scenario where data variability may cause a change in which models outperform others. Generally speaking, our model leans towards having higher recall scores than precision because we wanted to make sure that we were actually catching all of the occurrences of actual positive labels at the expense of overpredicting and raising the number of false positives. The rationale behind this is that you don't want people who need immediate assistance to not get flagged by the classifier as being a certain label, so we over-correct so that we miss as little positive flags (labels) as possible.

Did you ever consider a two-step pipeline: first classify "disaster vs. non-disaster," then assign sub-categories? What could be the implication of this strategy?

Labels for specific disaster occurrences were included in the default dataset; however, upon inspection, the quality of the labeling was not comparable to other labels. Our initial tests on a wider list of labels gave us performance metrics that were undesirable. Scrutinizing the data led us to isolating which labels we were working with, but other labels such as earthquake, fire, storm, floods, weather_related were included as the primary labels concerning disaster event types that were run in our initial tests of the ensemble classification. The reason we left these out is that the overall performance of these labels was hindered by the underlying quality of the data being very poor, in that in some cases, the labels were wrong for a majority of the labels. In order to implement a system that would be able to perform classification of disaster event types, we would have to solve the underlying data issue, i.e., we would have to assign new labels to the text data. This could be performed by doing some form of topic modeling and doing a similarity search (with pre-defined disaster types) to populate the disaster label categories for the different disaster types. However, the obvious drawback of this is that we aren't necessarily sure if this strategy will give us more meaningful data than what we already have. Because this method relies heavily on the underlying quality of the text data, and we've already highlighted limitations in that data quality, there's no guarantee that performing this type of data-preprocessing in order to build a classifier on would be more successful than the base labeling. In the end, the best solution available is to simply manually assign labels, and for a dataset with 26k+ entries, it is simply beyond the scope of what we can reasonably achieve within this project.

The implication of this type of classification task, relating to disaster events is multifaceted. There is a compute bandwidth implication and also a semantic / context prediction implication to a model prediction pipeline. From a purely practical lensview, having a model interpret reports (text data) in real time, would allow for disaster coordinators to isolate geographic regions where different types of disaster events occur. For example, an earthquake can cause downstream disaster events to occur such as tsunamis or infrastructure damage that leads to fires. Having a system that categorizes messages and geolocates those messages to specific areas could assist coordinators in understanding where certain kinds of assistance and resources are needed to address ongoing disaster events. Another potential outcome related to this is that, from a data perspective, if we know that a disaster event is occurring in a certain geographic region that doesn't have certain infrastructure that would require using a more robust model, perhaps you could seamlessly downscale to a lighter model that is more tailored to the local context. In a similar view, there is a possibility that you could perform layered predictions, where disaster type prediction labels inform downstream model predictions by providing additional semantic representations to the input textual data (reports).

7. References:

[1] Dataset: https://huggingface.co/datasets/community-datasets/disaster_response_messages

[2] Phi-3.5 (LLM for RAG): Microsoft. (2024). Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. Microsoft Research. (Used as microsoft/Phi-3.5-mini-instruct for the RAG system).

[3] Scikit Learn: https://scikit-learn.org/stable/api/index.html

[4] Word Cloud: https://www.geeksforgeeks.org/python/generating-word-cloud-python/

[5] Bert-Base-Uncased: https://huggingface.co/google-bert/bert-base-uncased

[6] Latent Dirichlet Allocation: https://gist.github.com/ululh/c3edda2497b8ff9d4f70e63b0c9bd78c

[7] Transformers: https://huggingface.co/docs/transformers/en/index

[8] Faiss Similarity Search: https://faiss.ai/

[9] Shutil: https://docs.python.org/3/library/shutil.html

[10] BERT (Base Model): Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805. (Used for fine-tuning and embeddings).

[11]Sentence-BERT (Embeddings): Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint arXiv:1908.10084. (Used for all-MiniLM-L6-v2 and all-mpnet-base-v2 embeddings).

[12]Latent Dirichlet Allocation (LDA): Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. Journal of machine Learning research, 3(Jan), 993-1022. (Used for topic modeling feature extraction).

[12]RAG (Retrieval-Augmented Generation): Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv preprint arXiv:2005.11401.

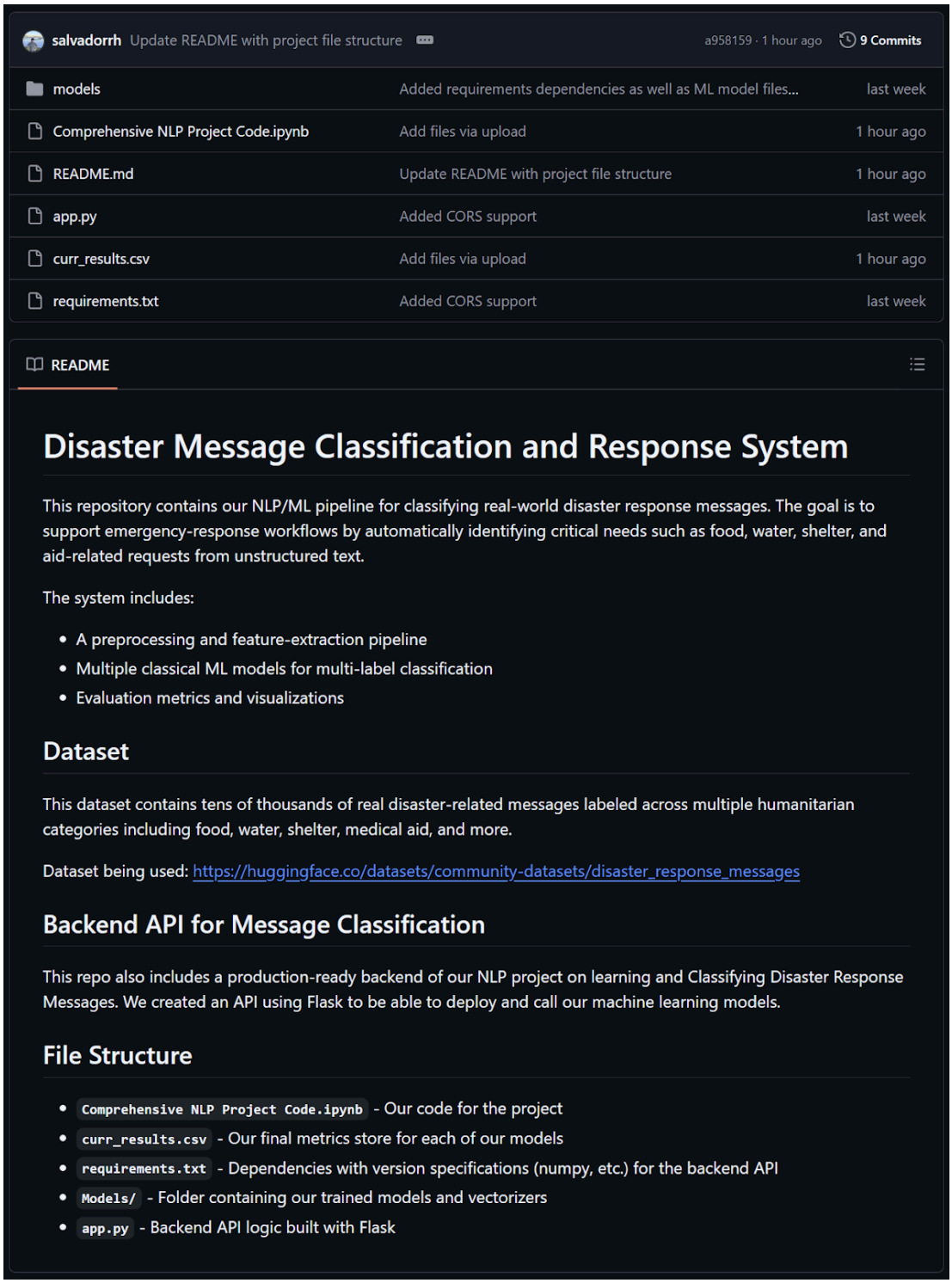
[13] FAISS: Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 7(3), 535-547. (Used for vector retrieval in the RAG pipeline).

[15]SentenceTransformers: Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. (Python framework used for generating dense vector embeddings).

8. Code:

Demo: https://salvadorrobles.com/disaster_classifier.html

Code: https://github.com/salvadorrh/disaster-response-api



salvadorrh Update README with project file structure — a958159 · 1 hour ago — 9 Commits

models	Added requirements dependencies as well as ML model files...	last week
Comprehensive NLP Project Code.ipynb	Add files via upload	1 hour ago
README.md	Update README with project file structure	1 hour ago
app.py	Added CORS support	last week
curr_results.csv	Add files via upload	1 hour ago
requirements.txt	Added CORS support	last week

README

Disaster Message Classification and Response System

This repository contains our NLP/ML pipeline for classifying real-world disaster response messages. The goal is to support emergency-response workflows by automatically identifying critical needs such as food, water, shelter, and aid-related requests from unstructured text.

The system includes:

- A preprocessing and feature-extraction pipeline
- Multiple classical ML models for multi-label classification
- Evaluation metrics and visualizations

Dataset

This dataset contains tens of thousands of real disaster-related messages labeled across multiple humanitarian categories including food, water, shelter, medical aid, and more.

Dataset being used: https://huggingface.co/datasets/community-datasets/disaster_response_messages

Backend API for Message Classification

This repo also includes a production-ready backend of our NLP project on learning and Classifying Disaster Response Messages. We created an API using Flask to be able to deploy and call our machine learning models.

File Structure

- `Comprehensive NLP Project Code.ipynb` - Our code for the project
- `curr_results.csv` - Our final metrics store for each of our models
- `requirements.txt` - Dependencies with version specifications (numpy, etc.) for the backend API
- `Models/` - Folder containing our trained models and vectorizers
- `app.py` - Backend API logic built with Flask