**SPARK**

Sprint 1

#use 3.12 python version

Analytics Module

OBJECTIVE: Develop a real-time analytics system that:

* Scrapes and ingests real-time global event data [Damien – In Progress]
* Maps global events to vendor locations
* Categorizes events (e.g. natural disasters, cyber threats) [Krittika]
* Visualizes insights/event-vendor mappings via dashboards and maps
* Triggers alerts/actions based on severity (i.e. rate events by impact)

KEY FEATURES:

1. Real-Time Event Scraping [Damien]
2. Event Ingestion (receive parsed event data from scraper, clean, optionally store versions)
3. Location Extraction
   1. NLP-based location parsing from event text
   2. Normalize and geocode to [lat, long] for map overlays
   3. Tools: spaCy, geopy, Google Maps API, GeoNames
4. Event Categorization
   1. Classify events into categories (e.g. natural disasters, political unrest, cyber threats, health outbreaks, etc.)
   2. Methods: Rule-based, Zero-shot NLP, or Topic Modelling
5. Vendor-Event Impact Mapping
   1. Cross-reference event locations with vendor site locations
   2. Output: Vendor(s) affected by each event
6. Severity Scoring/ Power Scaling of events
   1. Estimate event impact level using keywords, news frequency, (optionally) sentiment, etc.
   2. Scores used to prioritize/suppress alerts
7. Event-based trigger system
   1. Trigger webhooks or messages when events meet threshold conditions
   2. Inputs: Event category, severity, vendor mapping
8. Dashboard and Mapping Interface
   1. Render global map of events and vendors
   2. Event filters by category/severity
   3. Timeline/real-time update view

**✅ Sprint 1 Priority Tasks for You**

1. Collaborate to access sample scraped events
2. Extract and normalize locations
3. Classify events by category
4. Store processed data as JSON/ Azure Cosmos DB for map/dashboard use [Storage Layer]

A screenshot of a black screen

AI-generated content may be incorrect.

A screenshot of a white and black text

Description automatically generated

#### **Sub-section of Scraper: Event Categorization**

Overview of steps:=

1. **Input**: title, text, and/or summary from each scraped event.
2. **Preprocessing**: Clean and normalize text (remove HTML, fix encoding like â€™, etc.).
3. **Feature Extraction**: Optionally use keywords, TF-IDF vectors, embeddings, or language models.
4. **Categorization Approaches** (choose one to start with):
   * 🔹 **Rule-based**: Simple keyword dictionary (e.g., {'earthquake': 'Natural Disaster'})
   * 🔹 **Zero-shot classification** (via models like facebook/bart-large-mnli)
   * 🔹 **Topic modeling** (LDA, BERTopic)
   * 🔹 **Fine-tuned classifier** (if you label enough samples later)
5. **Output**: Add a new column: event\_category.

Current suggested Optimal Approach:

1. **Data Preprocessing**
   1. Text Normalization: Clean raw text by fixing encoding issues, removing HTML tags, special characters and handling case sensitivity
   2. Tokenization & Lemmatization: Break down text into meaningful tokens and normalize words to their base form using libraries like spaCy or nltk
2. **Embedding-based Feature Extraction**
   1. Pre-Trained Transformers: Use pre-trained transformer model (e.g. BERT, RoBERTa, or DistilBERT) to create embeddings for event text to capture context and semantic relationships
   2. Can fine-tune models for specific categories if have labelled data, or use zero-shot classification setup
3. **Zero-Shot Classification** (no need labelling)
   1. Zero-shot Learning: use zero-short classification via pre-trained models like facebook/bart-large-mnli (avail. on Hugging Face). To classify events into categories without labelled training data. Just need to define categories (e.g. Natural Disaster, Cybersecurity, Economic) and model will assign probability score for each category
   2. #Note: for fine-tuning, have a labelled dataset (training samples with known event categories), so model specializes for specific domain
4. Categorization Pipeline
   1. After text processed, categorize event using zero-shot classification or fine-tuned transformer models
   2. Store categorized event in Cosmos DB (optional), or can store in JSON

Pros of approach:

1. Scalability – pre-trained transformers like BERT allows model to understand diverse topics and capture semantic meaning better than simple keyword-based or traditional classifiers.
2. Zero-shot Classification: don’t need labelled dataset to start.
3. Model can be extended and fine-tuned later.
4. Cosmos Db ensures categorization results are stored efficiently for fast access.

Implementation:

1. Fetching of data:
   1. Used custom CSV parsing when reading CSV file (sep=’|’, quotechar=’”’) since separator in scraped CSV file is “|”.
2. Pre-processing:
   1. Checked null value percentages by column.
   2. Standardization of column names (lower case, no whitespace, separated by \_)
   3. Timestamp conversion - Converted UTC datetime column to datetime object.
   4. Converted keywords column to actual list.
   5. Removal of duplicates by URL and title.

A screenshot of a computer screen

Description automatically generated

1. EDA – to understand text coverage, time range and keyword/location quality:
   1. Check null values for text and summary fields and drop rows with BOTH text and summary fields as null.
2. Topic Modelling for event categorization:
   1. BERTopic (using title + summary as concatenated input)

Topic Modelling:

* **Unsupervised** technique to automatically discover hidden thematic structure in a large corpus of text. [Automatic clustering of data into topics]

#### Popular methods:

| **Method** | **Description** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **LDA (Latent Dirichlet Allocation)** | Most popular probabilistic topic model. Each document is a mix of topics. | Interpretable, widely supported | Needs tuning, may struggle on short texts |
| **NMF (Non-negative Matrix Factorization)** | Matrix decomposition approach for topics. | Often faster and cleaner on small corpora | Less probabilistic than LDA |
| **BERTopic** | Uses BERT embeddings + clustering (HDBSCAN or UMAP) | Works well on short texts, semantic similarity | Heavier to compute, depends on good embeddings |
| **Top2Vec** | Learns jointly optimized topic and word vectors | No need to predefine topic count | Can be memory intensive |

* Methods to consider for reddit posts:
  + BERTopic
    - Great for real-world texts like news and Reddit posts
    - Uses transformer embeddings for meanings instead of just word co-occurrence.
    - Handles short documents better than LDA.
  + LDA with tuned preprocessing
    - Use TF-IDF or CountVectorizer on cleaned text/summary fields.
    - Not ideal for very short content but still useful with good tokenization.

Possible alternatives to topic modelling:

* Zero-Shot Classification (no training required), e.g using HuggingFace’s transformers.
  + Possible categories provided, algo guesses which fits best, Great for bootstrapping.
* Supervised text classification (better if can label data)
  + Fine-tune model like sklearn + TF-IDF + Logic Regression, BERT (transformers) for text classification [more powerful than rule-based]

BERTopic:

A screen shot of a computer

Description automatically generated

4 key components used in BERTopic:

1. Transformer embedding model
   1. Supports several libraries for encoding text to dense vector embeddings: e.g. Sentence Transformers, Flair, SpaCy, Gensim, USE (from TF Hub) -> Sentence Transformers library provides most extensive library of high-performing sentence embedding models (found on HuggingFace Hub)
   2. Sentence Transformers on HuggingFace Hub:
      1. sentence-transformers/all-MiniLM-L6-v2 : high-performing model that creates 384-dimensional sentence embeddings
2. UMAP dimensionality reduction
3. HDBSCAN clustering
4. Cluster tagging using c-TF-IDF

…

# Note: Coherence score for BERTopic was only ~0.54, and topics classified into were very broad and not very useful, so used openAI for prompting instead to make use of the LLM.

Observations of LLM (OpenAI):

1. Reddit + Google news csv files [Output of scraping] => 359 records => 12.18 seconds



1. Damien’s Suggestion: Instead of scraping the data first, possible to pass in the URLs as part of the prompt and have the scraping + topic modelling done with the prompt?
   1. NO, gpt models cannot browse URLs or load webpages directly UNLESS
      1. Use GPT with browsing enabled (e.g. ChatGPT Pro with browsing or plugins)
   2. Why sending URL in prompt works when using ChatGPT but not when using openAI through python API (openai.chatCompletion)?
      1. ChatGPT uses built-in tool (web) when using ChatGPT with browsing enabled: allows searching, opening and extracting live content from web – similar to a browser plugin
      2. A screenshot of a computer

         AI-generated content may be incorrect.Using GPT-4 via Python API – don’t have access to web browsing/file uploads, so prompt alone is not sufficient to allow scraping
      3. Tested with single URL, results are being returned without an error, BUT wrong info (did not match original article). (Hallucination of details, probably trying to guess a plausible story from the URL itself or previous training data, but no real access. => FAILED ATTEMPT.

A screenshot of a black screen

AI-generated content may be incorrect.

* 1. Conclusion: Can’t perform scraping automatically with openAI API, just by passing in URLs. Need to pass already scraped data into the prompts.

Processed\_data.csv – 1st run, with not very specific location extraction

Processed\_data\_updated.csv – 2nd run, with better location extraction prompt

A screenshot of a computer

AI-generated content may be incorrect.

Findings on location extraction:

* ~64 unknown columns (from random picks, seems like those articles don’t have a specific event and just talking about general things so no location, but check every Unknown value again.
* Some very specific locations not necessary for mapping/geocoding (e.g. specific labs/parks, etc.) -> narrow down to city/region/country in the prompting
* Repetitions like ‘New York City’ vs. ‘New York’ -> doesn’t matter since geocoding will still point to the same latitude/longitude coordinates
* Overly specific locations (e.g. Rikers Island, NASA Jet Propulsion Laboratory) -> refine prompt (e.g. to only extract city/state/country level granularity)
* Other issues/variants in location extraction -> will be handled by geocoding later, ignored temporarily.

Go by verticals (refer to resilinc), make topics more specific (how STL will be affected by it) – how specific? What are the categories in resilinc?

Frontend to be done by 29th May

Mapping:

1. Plotly + Dash (or Streamlit)
   1. Mapping events in real-time
   2. Applying filters
   3. Possibly embedding in a web interface/dashboard
   4. If need very advanced map rendering, then switch to Kepler.gl or deck.gl via Pydeck
   5. Best for: real-time interactivity, dashboards, dynamic filtering, and performance in modern web apps.

A screenshot of a computer

AI-generated content may be incorrect.

1. Folium
   1. Based on Leaflet, so great for base map tiles (OpenStreetMap, satellite, etc.)
   2. Good for **choropleths**, markers, clusters.
   3. BUT not built for real-time updates, lacks dynamic filtering, harder to integrate filters/UI widgets unless embedded in complex Jupyter setups or custom web apps, slower for large datasets
   4. Best for: static or semi-interactive geographic mapping (Leaflet.js-based), simple layers.

* Solution: Explore Plotly (+Dash)