II. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

   I am a computer-vision/remote-sensing and data-science freelance researcher from Nepal. Amongst other ML competitions, I have been in a prize-winning position for NASA SOHO comet search, NOAA Precipitation Prediction (Rodeo 2), the Spacenet-8 flood detection, and 2019 IEEE GRSS data fusion contest. When I am not trying to keep up with the rapidly evolving and fast-paced AI field or optimizing algorithms I find myself optimizing my performance in trail running or diving.

2. What motivated you to compete in this challenge?

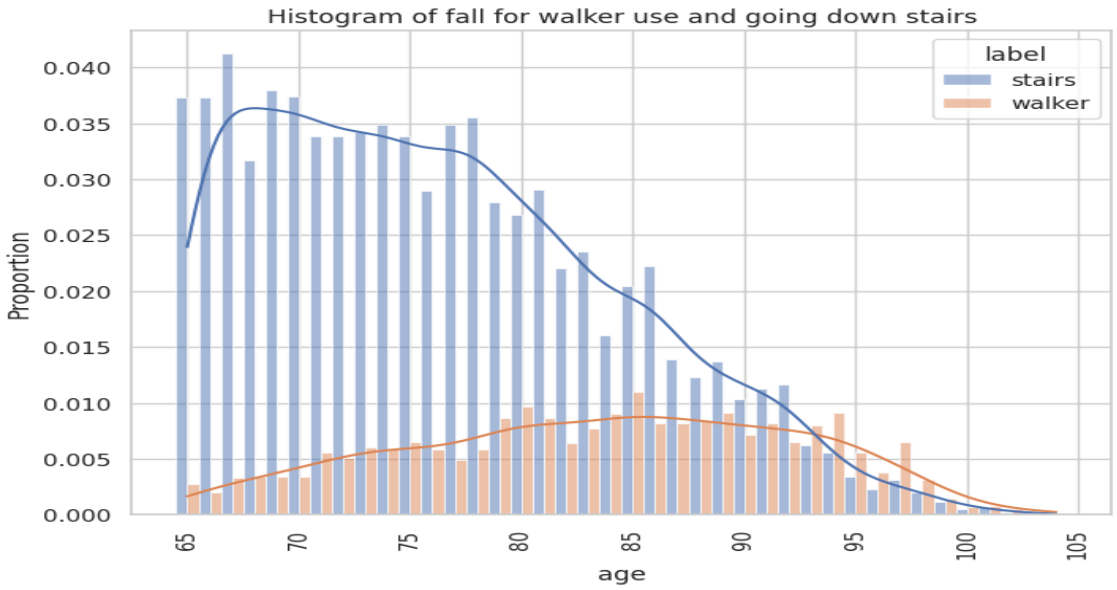
   I enjoy participating in machine learning/data-science challenges and have been doing it for a while. This particular competition gave me the opportunity to use/evaluate recent trends/SOTA in Large Language models, semantic search, knowledge graphs, and vector databases, which was different from my computer-vision background. After exploring the data, specifically extracting insights from the narrative, I became quite intrigued and so went ahead iteratively developing my workflow. I also appreciated the open evaluation nature of this competition and the freedom it gave to come up with creative solutions.

3. High level summary of your approach: what did you do and why?

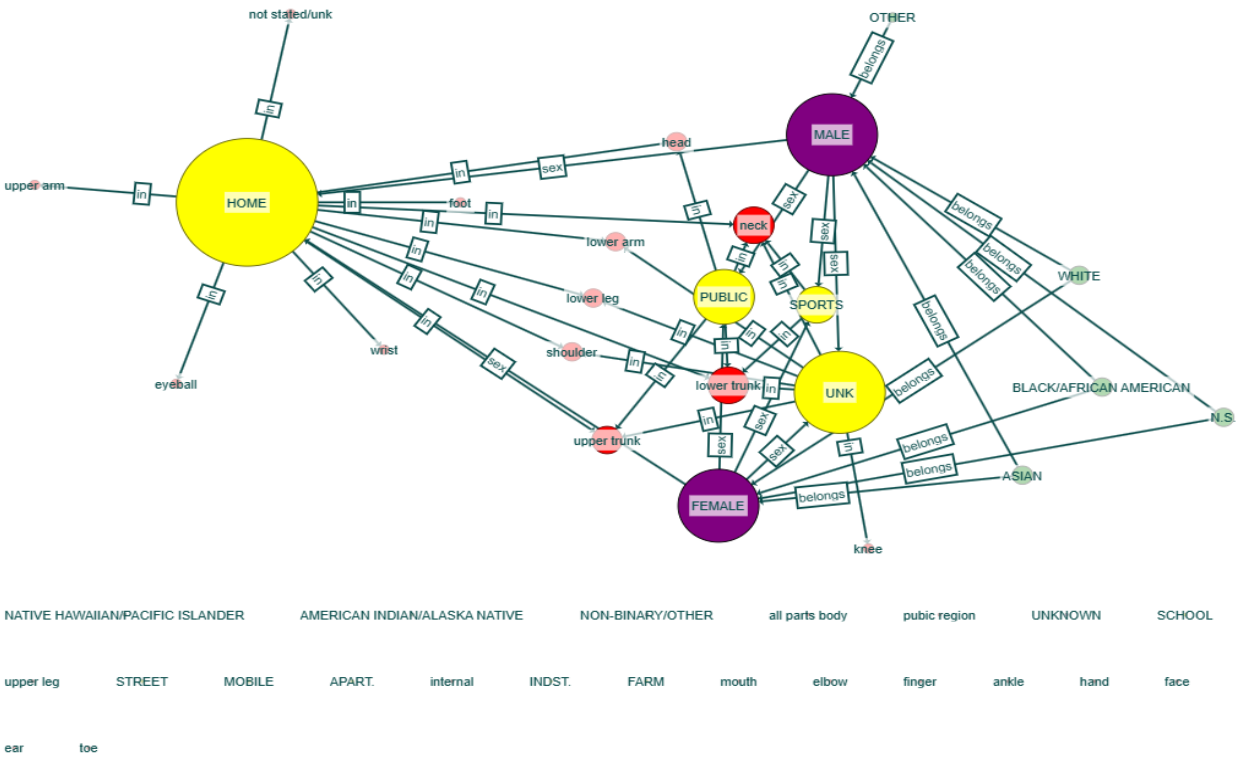
   We leveraged the 'OpenAI's text-embedding-ada-002' embeddings provided by the host to create a vector store in FAISS(Facebook Semantic Search). This allowed us to query the narrative using range search that returned relevant results. A key part of our solution involved meaningful context-aware and narrative(limited character) imposed query construction and understanding/categorizing actions that cause a fall. We also dug into the coding manual to understand how the narrative was structured and how best to extract precipitating events. We compared our solution to samples from OpenAI Chat GPT 3.5 and concluded ours was just as effective. Graphs were constructed for visualization and data exploration.

4. Do you have any useful charts, graphs, or visualizations from the process?

   We were particularly struck by results from two of our data explorations. We compared falls when using a walker to when using a stair. As shown below, the two-differing distribution (gaussian vs long-tailed) tells a story of how age affects the use of walkers and stairs.



By constructing a graph, we found that nerve damage is mostly associated with neck injury.



5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

1. We load narrative embeddings into FAISS and perform a semantic search using range search.

```
# perform range search given query embedings and some threshold value
def extract_query_result(embeds, thres=0.44):
    df = load_primary(primary_path)
    activities = embeds.keys()
    for q in embeds.keys():
        df[q] = [1]*df.shape[0]
    for query_text, embedding_q in embeds.items():
        result = fastIndex.range_search(embedding_q.reshape(1, -1).astype('float32'),
thres)
        df.loc[result[2], query_text]=result[1]

    df_extracted_semantic = df[df[activities].sum(axis=1)!=len(activities)]
    print (df_extracted_semantic.shape, df.shape)
    return df_extracted_semantic
```

2. From the coding manual we found keywords used for precipitating events such as slipped and tripped. We also used ‘s/p’, and ‘after’ also used explicitly as sentence separators in the narrative written as ‘sequence of events’. This was central in our approach to cleaning the narrative to extract a concise description of events.

```
# slipped
sp = {"s'd&f": ["slipped and fell", 'slip and fall', 'slipped'],
        "s/p": "after"}
# tripped
td = {"t'd&f": ["tripped and fell", 'trip and fall', 'tripped']}

sent_first = None
    # 's/p' and 'after are shorthand that the coding manual explicitly recommends to be
used as seperators
    for sent in sentences:
        if 's/p' in sent:
            sent_first = sent.split('s/p')[1]
            break
        if 'after' in sent:
            sent_first = sent.split('after')[1]
            break
```

3. We created a knowledge graph from a small subset(samples belonging to nerve damage) which proved insightful; such as the identification of the neck as a major body part involved during nerve damage.

```
df_do = df[df.diagnosis=='nerve damage']
#print (df_do.shape)
colors=['blue', 'red', 'green', 'yellow', 'purple']
G = nx.DiGraph()
G.add_nodes_from(list(mapping['location'].values()), color='yellow')
G.add_nodes_from(df.body_part.unique().tolist(), color='red')
G.add_nodes_from(list(mapping['race'].values()), color='green')
G.add_nodes_from(list(mapping['sex'].values()), color='purple')

edges_location_body =  []
for item in tqdm(df_do.iterrows(), total=df_do.shape[0]):
      edges_location_body.append((item[1].location , item[1].body_part))

edges_race_sex =  []
for item in tqdm(df_do.iterrows(), total=df_do.shape[0]):
      edges_race_sex.append((item[1].race , item[1].sex))

edges_sex_location =  []
for item in tqdm(df_do.iterrows(), total=df_do.shape[0]):
      edges_sex_location.append((item[1].sex , item[1].location))

G.add_edges_from(edges_location_body, relation='in')
G.add_edges_from(edges_race_sex, relation='belongs')
G.add_edges_from(edges_sex_location, relation='sex')
```

6. Please provide the machine specs and time you used to run your model.

   We report here the notebook runtime of approximately 1 hour, which includes training a random forest(n=10) and a CountVectorizer model. Getting embeddings from clean text, reducing the embedding, and clustering the results (all part of inference) is where compute time is expensive. Also note we pre-loaded the OpenAI ChatGPT3.5 response.

   - CPU (model): AMD Ryzen 5 4500U
   - GPU (model or N/A): N/A
   - Memory (GB): 8GB
   - OS: Windows 11
   - Train duration: N/A
   - Inference duration: Approx 1 hour

7. Anything we should watch out for or be aware of in using your notebook (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

On a clean conda environment with all requirements satisfied as in requirements.txt, we ran the notebook successfully multiple times. Do specify your openai keys. By default the OpenAI ChatGPT response is pre-loaded but you can easily change this to retrieve more responses with the provided prompt.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

   No, but the graph images are saved first and then loaded. The notebook needs to be run so that the graph is rendered again for interactive visualization. The graphs were created using 'yFiles for jupyter' and an external interface opens if you want to export images or do some additional analysis.

9. How did you evaluate the quality of your analysis and insights, if at all?

   We frequently used the 'silhouette score' for evaluating and comparing the cluster embeddings of precipitating events. And also the cosine similarity between embeddings to evaluate different approaches in extracting the precipitating event.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

    We attempted prompt-based extraction of the precipitating event using local LLM (Falcon-7B quantized further to 4-bit) but the process was too slow for us.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

    We took a small subset in making our knowledge graph. In the future, we would not only use larger sets, but also improve the knowledge graph creation process itself. We believe doing so could reveal more actionable insights and identify fringe cases. There are nearly 500 products identified that contributed to a fall. Exploring their impact on fall occurrences, and severity or learning some actionable insights simply will require more time.

12. What simplifications could be made to run your solution faster without sacrificing performance?

    Performance was important to us for iterative development and resulting data exploration, and we think all our methods are optimized. Semantic search with FAISS is blazingly fast. For reducing embedding dimension and clustering, we borrowed some community codes that are also fast. We used CPU only, but some of the operation can greatly benefit from a GPU usage such as extracting embedding from cleaned text or GPU implementations for clustering.