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 currently a 1st year master's student at the University of Padova, having previously worked as a Data Scientist at BTG Pactual.

2. What motivated you to compete in this challenge?

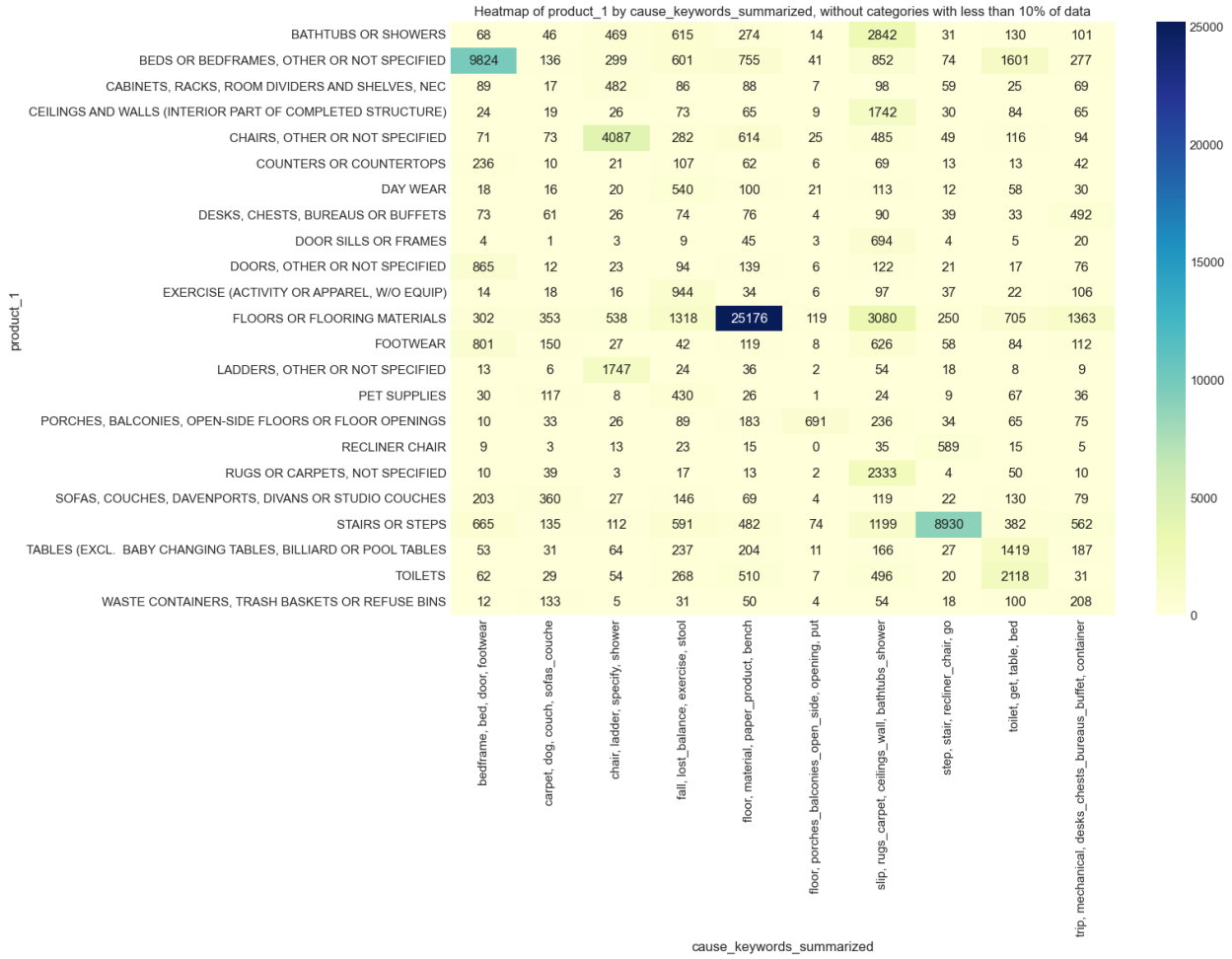
My strong interest in learning more about LLMs and how they can be used in conjunction with tabular data motivated me to compete in this challenge. Having no previous applied experience with LLMs I used this project as a way to learn more about how these models can be applied to real-world scenarios. Also knowing older adults that have experienced serious falls motivated me more to contribute to this competition in order to potentially reduce the occurrence of falls.

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

The main idea of my approach is to develop an information extraction pipeline to aid in obtaining more specific details about what happened in the fall events. Inspired by TabLLM I created a custom prompt in a human-readable format for each data point, combining information contained on tabular variables and event narratives (which are "translated" so that technical terms are easier to understand). Six questions, each related to a different characteristic of the fall (cause, diagnosis, etc.), are appended to the prompts and fed to a Text2Text Generation LLM that extracts information by answering the questions. Finally, I train one LDA model for each kind of question to model the topics of the answers, creating a granular way of analyzing and recovering falls given their different characteristics.

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

Since my approach is more related to information extraction there aren't visualizations related to specific insights. The following heatmap however provides a good demonstration of how this information can be used to potentially generate insights:



The Y axis is a categorical variable that describes the main product involved with the fall whereas the X axis shows the topics created from the question "what caused the patient to fall?". Visualizations like this can be used to evaluate if the topics make sense. It also demonstrates how the combination of the extracted textual information from the narratives with the tabular variables allow for a more granular understanding of the events.

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

```
def translate_narrative(text):
    # lowercase everything
    text = text.lower()

    # unglue DX
    regex_dx = r"([^\W]*(dx)[^\W]*)"
    text = re.sub(regex_dx, r". dx: ", text)

    # remove age and sex identifications
```

```
    ## regex to capture age and sex (not perfect but captures almost all of the cases)
    regex_age_sex = r"(\d+)\s*?(yof|yf|yo\s*female|yo\s*f|yom|ym|yo\s*male|yo\s*m)"
    age_sex_match = re.search(regex_age_sex, text)

    ## format age and sex
    if age_sex_match:
        age = age_sex_match.group(1)
        sex = age_sex_match.group(2)

        # probably not best practice but it works with this data
        if "f" in sex:
            #text = text.replace(age_sex_match.group(0), f"{age} years old female")
            text = text.replace(age_sex_match.group(0), f"patient")
        elif "m" in sex:
            #text = text.replace(age_sex_match.group(0), f"{age} years old male")
            text = text.replace(age_sex_match.group(0), f"patient")

    # translate medical terms
    for term, replacement in medical_terms.items():
        if term == "@" or term == ">>" or term == "&" or term == "***":
            pattern = fr"({re.escape(term)})"
            text = re.sub(pattern, f" {replacement} ", text) # force spaces around
replacement

        else:
            pattern = fr"(?<!-)\b({re.escape(term)})\b(?!-)"
            text = re.sub(pattern, replacement, text)

    return text
```

This function "translates" the narratives into non-technical terms using the `medical_terms` dictionary. I found that for my approach this is essential in order to get better topics in my model. This is especially true due to the fact that there are many terms that mean the same thing but are written in different ways, e.g. "clinical diagnosis", "dx", ">>".

```
def make_big_narrative(row):
    age = row["age"]
    sex = process_nan(row["sex"]).lower()

    diagnosis = process_nan(row["diagnosis"]).lower()
    other_diagnosis = process_nan(row["other_diagnosis"]).lower()
    body_part = process_nan(row["body_part"]).lower()
```

```
    diagnosis_2 = process_nan(row["diagnosis_2"]).lower()
    other_diagnosis_2 = process_nan(row["other_diagnosis_2"]).lower()
    body_part_2 = process_nan(row["body_part_2"]).lower()

    product_1 = process_nan(row["product_1"]).lower()
    product_2 = process_nan(row["product_2"]).lower()
    product_3 = process_nan(row["product_3"]).lower()

    narrative_processed = process_nan(row["narrative_processed"])

    if diagnosis_2 != "":
        return f"The patient has suffered a fall. The patient is a {age} years old
{sex}." + \
        f"\nThe things involved in the accident were: {product_1}, {product_2},
{product_3}." + \
        f"\nThe patient reports {diagnosis} and {other_diagnosis} on its {body_part}.
The patient also reports {diagnosis_2} and {other_diagnosis_2} on its {body_part_2}."
+ \
        f"\nThis is the description of the incident: {narrative_processed}"

    if other_diagnosis != "":
        return f"The patient has suffered a fall. The patient is a {age} years old
{sex}." + \
        f"\nThe things involved in the accident were: {product_1}, {product_2},
{product_3}." + \
        f"\nThe patient reports {diagnosis} and {other_diagnosis} on its {body_part}."
+ \
        f"\nThis is the description of the incident: {narrative_processed}"

    else:
        return f"The patient has suffered a fall. The patient is a {age} years old
{sex}." + \
        f"\nThe things involved in the accident were: {product_1}, {product_2},
{product_3}." + \
        f"\nThe patient reports {diagnosis} on its {body_part}." + \
        f"\nThis is the description of the incident: {narrative_processed}"
```

This function generates the big prompts ("big narratives") inspired by TabLLM that combine categorical variables with the narratives. While testing different ways of using LLMs to extract information I found that just using the plain narrative would often make models either hallucinate or, if the narrative wasn't very long, repeat the entire narrative as the answer.

```
for i, (case_number, narrative) in tqdm(enumerate(zip(primary["cpsc_case_number"],
primary["big_narrative"])), total=len(primary), position=0, leave=True):
    # append identifier to dataframe
    questions_df["cpsc_case_number"].append(case_number)

    # ask questions
    for context, question in questions.items():
        # create questions and tokenize them
        input_text = narrative + "\n" + question
        input_ids = tokenizer(input_text, return_tensors="pt").input_ids.to("cuda")

        # get answer and remove special tokens
        outputs = model.generate(input_ids, max_new_tokens=20)
        answer = tokenizer.decode(outputs[0]).replace("<pad>", "").replace("</s>", "")

        questions_df[context].append(answer)

    # save dataframe every 2500 steps
    if i % 2500 == 0 and i != 0:
        questions_df_save = pd.DataFrame(questions_df)
        questions_df_save.to_csv("data/questions_primary.csv", index=False)
```

This piece of code iterates over all of the "big narratives" and asks each question to the LLM. It is the main idea of my approach.

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

All of my code is ran with the following specs (except for the LLM inference):

- **CPU (model): Apple M1 Pro 8-core CPU.**
- **GPU (model or N/A): N/A (no GPU was used).**
- **Memory (GB): 16 GB.**
- **OS: macOS Ventura.**
- **Train duration: ~15 minutes to train all LDA models for all answers, ~9 hours for hyperparameter tuning to get the best parameters for each LDA model.**
- **Inference duration: ~7 minutes.**

The LLM inference uses a Kaggle notebook, which has the following specs:

- **CPU (model): Intel(R) Xeon(R) CPU @ 2.00GHz.**
- **GPU (model or N/A): Nvidia P100.**
- **Memory (GB): 30 GB.**
- **OS: Debian Linux.**

- **Train duration: 0 seconds (just used a pretrained model).**
- **Inference duration: ~20 hours (in total, asking six different questions for each data point).**

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.)?

The main quirk is that I ran the LLM inference on another machine with a GPU, but if you have a machine with a strong GPU everything should work out of the box.

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

No.

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

My main way of evaluating the quality of my approach was by comparing certain topics and their categorical variable counterparts (e.g. topics for "what is the full diagnosis?" and the diagnosis variable). I also analyzed the most representative answers for each topic and the coherence score of the LDA models to check if they made sense.

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

The main thing I tried was using a Question-Answering LLM instead of a Text2Text Generation LLM. They are much more efficient and usually have a confidence score attached with the answer, which could prove really useful. However due to the extractive nature of these models (answers can only be parts of the text present in the prompt) sparse narratives proved problematic. I also believe that due to the very specific questions with relatively limited prompts, pre-trained LLMs without any kind of fine-tuning weren't able to capture the ideal answers.

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?

Some next steps I would take would be related to searching for different questions to ask the LLM in order to return a more fine-grained answer. Also related to the textual information extraction, I believe that a Question-Answering model would probably be ideal in the long term due to its efficiency and confidence score: I believe it would be possible to create a dataset and fine-tune an already existing Question-Answering model so that it provides the kinds of answers experts are interested in extracting. Other than that, I believe a more complex topic modeling algorithm (such as BERTopic) would greatly benefit this approach in order to capture the complexity of the data.

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

An easy way to get the solution faster is to ask less questions: even though I asked six different questions some of their answers were very similar (especially the "what" and "how" questions). Also, I believe the code could easily be optimized in order to ask the questions in parallel. One could also focus less on LDA optimization, which took a long time in my code due to the exploration of a variety of different hyperparameters. Finally, I believe that using BERTopic with a GPU would not only create better topics but would also be faster and easier to tune!