22 APR 24

TEXAS
The University of Texas at Austin

MIMIC LLM

Identifying duplicate chart event measurement types using the Azure OpenAI API.

PHILIP DIGIACOMO
Master of Science in Aritifical Intelligence

Summary of work

- The MIMIC-III dataset for chartevents contains many duplicate measurement types where different values of ITEMID are used to measure the same thing.
- For example the table of ITEMID value counts to the right demonstrates how ITEMID 211 and 220045 both have the same label (Heart Rate)
- We use LLM models including chain of thought and tree of thought methods on Azure's OpenAI API to automatically identify duplicate measurement types. This provides a valuable way to preprocess MIMIC chart events into more concise and useful data.
- The use of MIMIC data was carefully scrutinized to make sure that it is in compliance with physionet's rules. I confirmed via email with Prof Ying that I could use the Azure OpenAI interface with MIMIC data without violating data use agreements.
- The original colab notebook is provided here.
  - https://colab.research.google.com/drive/1y_jDEN_oa_hEP90YaHu_a3jPBRgXboMg?usp=sharing

ITEMID	count	LABEL
220045	2762225	Heart Rate
220210	2737105	Respiratory Rate
220277	2671816	O2 saturation pulseoxymetry
211	2174544	Heart Rate
742	1411239	calprevflg

Chart events data

- We use the MIMIC-III chart events dataset in this tutorial. We read in the first 100 million rows of this dataset (to conserve memory) from a drive location. We only read in certain rows and then save the dataframe as a csv so the redundant rows do not need to be read in. If running for the first time, run the first two commented out cells and comment out the third.
- The head of the resulting dataframe is shown to the right.

	SUBJECT_ID	ITEMID	CHARTTIME	VALUENUM	VALUEUOM	LABEL
0	36	223834	2134-05-12 12:00:00	15.00	L/min	O2 Flow
1	36	223835	2134-05-12 12:00:00	100.00	NaN	Inspired O2 Fraction
2	36	224328	2134-05-12 12:00:00	0.37	NaN	PCA dose
3	36	224329	2134-05-12 12:00:00	6.00	min	PCA lockout (min)
4	36	224330	2134-05-12 12:00:00	2.50	NaN	PCA 1 hour limit

Azure OpenAI set up

- We set up an Azure OpenAI client to get the embeddings associated with each label. Our examples of "Heart Rate" or "Respiratory Rate" are be fed into Azure OpenAI's embedding model and we get vectors associated with each label.
- Note that for my final submission I removed my personal API key.
- After getting the embeddings for each distinct LABEL found, we end up with a series shown to the right.

```
ITEMID
220045    [-0.027896853163838387, 0.001455605262890458, ...
220210    [-0.012819629162549973, 0.011111615225672722, ...
220277    [-0.012289450503885746, 0.007591912988573313, ...
211       [-0.027896853163838387, 0.001455605262890458, ...
742       [-0.008985236287117004, -0.016543535515666008,...
Name: LABEL, dtype: object
```

Nearest neighbors fit of the embeddings

- We fit a nearest neighbors algorithm through sk-learn to get the 5 nearest labels to each given label by measuring the Euclidian distance between their embeddings.
- To the right, we show an example of the nearest neighbors of the first row in the numpy matrix. Its five nearest neighbors are shown. Note that the numpy array's indices are 0,1,2,… and we must convert from this index to ITEMID values and eventually labels.
- The dataframe shown to the right shows how we convert between those values.
- Note that "Heart Rate" appears at ind = 0 and ind = 3 and the nearest neighbors fit has identified 3 in the fit to the right.
- The first nearest neighbor is always identified as the original label itself so we will have to disregard that in future steps.

```
knn.kneighbors([embed_np[0]], return_distance=False)
```

```
array([[  0,   3,   7, 317, 459]])
```

ind	ITEMID	LABEL
0	220045	Heart Rate
1	220210	Respiratory Rate
2	220277	O2 saturation pulseoxymetry
3	211	Heart Rate
4	742	calprevflg

Nearest neighbors in greater detail

- We show more detail to the nearest neighbors fit by showing the labels themselves.
- We can see that some nearest neighbors are true duplicates like “Heart Rhythm” and “Cardiac Rhythm” while some of the other nearest neighbors are not true duplicates.
- For example, code status is a near neighbor with marital status but they are not true duplicates. We will use prompting of GPT-3.5 in future steps to narrow down this list of duplicates to the true matches in a way that incorporates chain of thoughts prompting (first getting embeddings, and then using that output to get duplicates).

ITEMID	count	LABEL	nearest
220045	2762225	Heart Rate	[Heart Rate, Heart Rate, Heart Rhythm, Breath ...
220210	2737105	Respiratory Rate	[Respiratory Rate, Respiratory Rate, Respirato...
220277	2671816	O2 saturation pulseoxymetry	[O2 saturation pulseoxymetry, O2 Saturation Pu...
211	2174544	Heart Rate	[Heart Rate, Heart Rate, Heart Rhythm, Breath ...
742	1411239	calprevflg	[calprevflg, calciferol, flovent, DOPPLER FLAP...
646	1389033	SpO2	[SpO2, SpO2-L, SaO2, pO2, SvO2]
618	1373254	Respiratory Rate	[Respiratory Rate, Respiratory Rate, Respirato...
212	1345660	Heart Rhythm	[Heart Rhythm, Cardiac Rhythm, Heart Rate, Hea...
161	1320180	Ectopy Type	[Ectopy Type, Ectopy Type 2, Ectopy Frequency,...
128	1309585	Code Status	[Code Status, State, CPP, Marital Status, Stat...

GPT 3.5 for duplicate selection

- We prompt GPT 3.5 with our 4 nearest neighbors as measured from the embeddings determined in previous steps and ask it to find identical measurement types. Our exact prompt (with string formatting for the original label and list of possible duplicates) is shown to the right.
- An example of GPT-3.5's output is also shown to the right. We manually give it 5 similar Arterial blood pressure reading types although only a few are identical. GPT's response correctly identifies different wordings of "Systolic arterial blood pressure" without selecting similar yet different items like diastolic BP.

```
Identify any and all
possible measurement
types in the following
  comma separated list
that is identical to
{orig}. Here are the
measurement types:
{L}'''
```

```
[27] dups = ['Arterial BP [Diastolic]', 'Aterial BP Mean', 'Arterial Blood Pressure me
     'Aterial Blood Pressure systolic','Systolic Arterial blood press']
     list_to_prompt('Arterial BP [Systolic]',dups)

     'The possible measurement types that are identical to "Arterial BP [Systolic]" in
     the given list are:\n\n- Systolic Arterial blood press'
```

Table for data pre-processing

- To aid data pre-processing and be able to merge duplicate items in chart events, we create a dataframe that converts an ITEMID to a list of duplicate ITEMID values. Shown to the top right.
- For this to be useful, we must replace duplicate ITEMID values with a single value. We choose to replace all of these with the ITEMID value that is the smallest. We sort the lists of duplicate item IDs so that the first element can be used as a replacement. If the list of duplicate ITEMID values is empty, we insert the original ITEMID itself to allow for better automation. This is shown at the bottom right.

```
ITEMID
220045                                [220045]
220210    [220210, 224690, 224688, 224689]
220277                          [220227, 646]
211               [220045, 212, 3337, 3494]
742                                       []
dtype: object
```

```
0                                 [220045]
1    [220210, 224688, 224689, 224690]
2                          [646, 220227]
3           [212, 3337, 3494, 220045]
4                                 [742]
5                           [834, 3837]
6    [220210, 224688, 224689, 224690]
7                                [3354]
8                                 [161]
9                                 [128]
dtype: object
```

Analysis and summary of work

- The chain of thoughts method was used to split the original problem of finding duplicate items into sub-problems. The first problem involved finding nearest neighbors and the second problem was selecting any nearest neighbors that may be duplicates.
- We could not feed the model all of the ITEM labels and ask it to make groups of duplicates because the list would exceed the max sequence length.
- The tree of thoughts method was used by having the model identify 5 nearest neighbors of labels that may be possible duplicates and then have the model analyze those candidates in further detail to determine duplicates.
- This method could be improved by elaborating on the tree of thoughts by asking the model to come up with duplicate groups with varying degrees of certainty. We could prompt the model to be liberal in its grouping of duplicates or more strict in what it considers a duplicate. Then the different groupings could be used for data pre-processing and evaluated on which was more effective in creating concise and useful data for downstream learning tasks.