CausalFitbit, an example of DEFT (DAG Extraction From Text)

Robert R. Tucci
tucci@ar-tiste.com

March 5, 2024

Abstract

Previously, we created a GitHub repo called "Mappa_Mundi" that contains a Python app for doing causal DEFT (DAG Extraction from Text). In this white paper, we describe the Python app "CausalFitbit" that applies the Mappa Mundi software to a Fitbit dataset from Kaggle. The CausalFitbit software is available as open source at GitHub.

Previously, we created a GitHub repo called "Mappa_Mundi" (Ref.[3]) that contains a Python app for doing causal DEFT (DAG Extraction from Text).

In this white paper, we describe the Python app "CausalFitbit" that applies the Mappa Mundi software to a Fitbit dataset (Ref. [1]) from Kaggle. The CausalFitbit software is available as open source at the GitHub repo "CausalFitbit" (Ref.[2]).

Note that the Mappa Mundi app uses LLMs to perform 2 distinct jobs:

- doing **sentence splitting** via a fine tuning of BERT called Openie6[1]
- calculating **sentence similarities** via sBERT.

CausalFitbit, on the other hand, doesn't use LLMs at all (even though it uses the same algorithm as Mappa Mundi). How can this be?

- There is no sentence splitting to be done in CausalFitbit, because the sentences are already split; they are simple sentences from the onset.
- In CausalFitbit, we provide an analytical formula for calculating sentence similarity. This will be explained better later on, but basically, what we do is calculate the Z-distance between two "sentences". A sentence in this case is just something like $z = 5$, and the Z-distance between sentences $z = 5$ and $z = 1$ is $|5 - 1| = 4$.

The simple sentences (ssents) considered in the Mappa Mundi repo are **linguistic**, such as "The ball is red". The ssents in CausalFitbit are **symbolic**, such as $z = 5$. In future, we expect that we will be using the Mappa Mundi algorithm with a combination of both linguistic and symbolic ssents.

All necessary CausalFitbit operations are performed by 3 jupyter notebooks which we will describe in detail next.

1. `heartrate_data_thinning.ipynb` **notebook**[2]

   The original dataset is available at Kaggle (Ref. [1]) as a csv (comma separated values) file. The dataset is fairly large ($\sim$ 85MB) because it contains heartbeat (i.e., pulse) data sometimes every 10 seconds or so.

   In this notebook, we use Pandas to average the pulse data over each hour interval. The resulting thinned dataset file is just 160KB.

2. `data_preparation.ipynb` **notebook**

   This notebook generates one csv file per patient (for a total of 33 patients). It stores those files in the folder named `patient_csv_records`

[1]Shameless plug: You could also use my free software SentenceAx. SentenceAx (Ref.[4]) is a full rewrite of Openie6.

[2]Unless otherwise stated, all jupyter notebooks are to be found in the folder entitled `jupyter_notebooks`.

This notebook does a whole bunch of chores, most of which are menial, but some aren't. One non-medial task it does is to add columns for "velocity" features.

Table 1 shows a table illustrating a made-up, pedagogical 2 line dataset. Table 2 shows the same dataset as Table 1, but after adding two more columns, f1Vel and f2Vel, for the velocities of the two features f1 and f2.

id	datetime	f1	f2
1503960366	2016-04-12 00:00:00	3.1	1.8
1503960366	2016-04-13 00:00:00	2.8	2.2

Table 1: Dataset without velocity columns.

id	datetime	f1	f1Vel	f2	f2Vel
1503960366	2016-04-12 00:00:00	3.1	nan	1.8	nan
1503960366	2016-04-13 00:00:00	2.8	(2.8-3.1)/24	2.2	(2.2-1.8)/24

Table 2: Dataset of Table 1 after adding 2 velocity columns f1Vel and f2Vel.

As can be seen from Table 2, to add a velocity column for any feature of a Table, for instance f1, we divide

$\Delta f1 =$ the previous value of f1 minus its current value

by

$\Delta t =$ the previous value of time in hours minus the current value of time.

This can all be done with one line of code using the powerful Pandas function `diff()`.

3. `navigating_patient_records.ipynb` **notebook**

id	datetime	f1	f1Vel	f2	f2Vel
1503960366	2016-04-13 00:00:00	2.8	-.0125	2.2	.0166

Table 3: This made-up single line of a dataset would be replaced by the following single line of a patient simp file: `f1= 2.8 &z= .1<SEP>f1Vel= -.01 &z= .3<SEP>f2= 2.2 &z= .1<SEP>f2Vel= .016 &z= .2`

This notebook accomplishes the following 3 tasks.

- **wordifying**

  This step uses the class `PatientSimpRecord`. It generates a simp (simplified sentence) record for each csv patient record in the folder `patient_csv_records` and stores the resulting file in the `patient_simp_record` folder.

  What we call wordifying is illustrated by Table 3.

For any feature (i.e., column) f, let

σ_f = the standard deviation of the column f (calculated with the Pandas function `std()`)

$< f >$= the average of the column f (calculated with the Pandas function `mean()`)

Then

$$z(f) = \frac{f - < f >}{\sigma_f} \tag{1}$$

Wordifying means we replace a segment like `f1=2.8` by a segment like `f1=2.8 &z=.1`. The latter looks like a ssent if you read `&z=.1` as "and z equals 0.1.

An important feature of wordifying is that the number of columns in every row need not me the same because in the wordified rows, if there is missing information for a cell, we skip it.

- **DAG atlas**

  This step is practically identical to its counterpart step in the Mappa Mundi repo. In this step, we use the class `DagAtlas` located in the file `mm_DagAtlas`, to construct as Dag atlas, based on the patient files in folder `patient_simp_records`. Note that all files starting with "mm_" come verbatim from the Mappa Mundi app (Ref.[3]).

  In building a DagAtlas, we are confronted with a decision problems: whether to accept or reject a bridge. The **acceptance indicator functions** A_{bridge} for this decision problems is as follows. Let

  $simi()$ = **similarity function**,

  $z(f)$ = defined by Eq.(1)

  R = the **z-radius**,

  $simi^*$ = **similarity threshold**.

  Then

$$simi(s_1, s_2) = \begin{cases} simi^* + 1 & \text{if } |z(s_1) - z(s_2)| < R \\ simi^* - 1 & \text{if } |z(s_1) - z(s_2)| > R \end{cases} \tag{2}$$

$$A_{bridge} = \begin{cases} 1 & \text{if } simi(s_1, s_2) > simi^* \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

  Eq.(3) is also used in Mappa Mundi, but Eq.(2) is new.

- **Visualizing**

  This step is practically identical to its counterpart step in the Mappa Mundi repo. In this step, we use the class `Dag` located in the file `mm_Dag` and graphviz, to draw DAGs.

In visualizing our DAGs, we are confronted with a decision problem: whether to accept or reject an arrow. The **acceptance indicator function** A_{arrow} for this decision problem is as follows.

$$A_{arrow} = \begin{cases} 1 & \text{if } p_{acc} > p^*_{acc}, N_{acc} > N^*_{acc} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Eq.(4) is also used in Mappa Mundi. There you will find the definitions of p_{acc} and N. Only arrows that satisfy $A_{arrow} = 1$ are drawn. The thresholds p^*_{acc} an N^* are set by the user.

References

[1] Kaggle.com. Fitbit fitness tracker data. `https://www.kaggle.com/datasets/arashnic/fitbit`.

[2] Robert R. Tucci. CausalFitbit at github. `https://github.com/rrtucci/CausalFitbit`.

[3] Robert R. Tucci. Mappa Mundi at github. `https://github.com/rrtucci/mappa_mundi`.

[4] Robert R. Tucci. SentenceAx at github. `https://github.com/rrtucci/SentenceAx`.