Masters Pilot: Mimic Dataset Generation Library

Ty Vaughan
March 26, 2018

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Overview

1. Introduction
2. Specifications
3. MDGL Components
4. Conclusion

Introduction

Customer: Dr. Xiaopeng Zhao
Advisor: Dr. Michael W. Berry

Project: Mimic Dataset Generation Library tool

- A tool that can access Mimic and, given a set of specifications, generate a patient dataset.
- Should be easy to setup and use

Introduction - Problem

One of the biggest issues with research is data access, interpretation, and processing.

- Ongoing effort has recently begun to help standardize data interpretation for Mimic [1].
- Their current solution is the Mimic Code Repository

Another issue is the ease of which data can be accessed.

Introduction - Solution

Mimic Dataset Generation Library

- Written in Python 2.7
- Requires the libraries psycopg2 and numpy
- Consists of 6 python scripts

The library is easily expandable with the code existing in the Mimic Code Repository and can quickly generate a patient dataset in ~19-30 minutes.

Introduction - Solution

When this tool is used, it will generate a patient dataset. The resulting dataset will be able to be found in a folder named "`patientfiles [YMD-HMS]`".

- Each patient will have its own data file named "`[subject_id].csv`"
- All lines within the file will follow the format:

```
[HH:MM], [name], [ID], [value]
```

- The first six lines contain static info about the patient at the beginning of the stay: record ID, age, gender, height, ICU type, and weight.

Overview

1. Introduction
2. Specifications
3. MDGL Components
4. Conclusion

Specifications

With the MDGL, a user can create a specifications files that specifies the following information:

- Which ICUs to consider: CCU, SICU, MICU, NICU, CSRU, TSICU
- Patient characteristics: Age, sex, length of ICU stay
- Parameters: Abbreviation, Mimic Ids

Specifications

To the right, we can see an example of the specifications file format:

ICUs

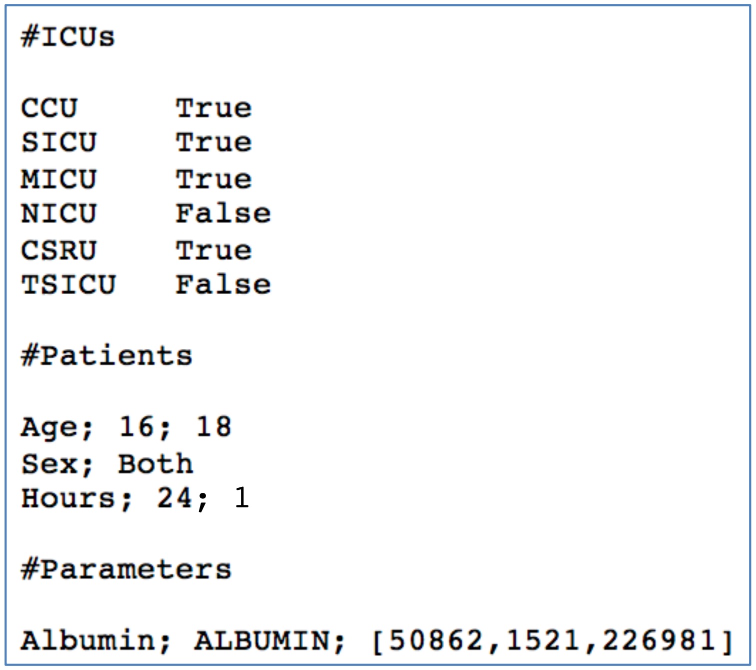
    [ICU_type] [True|False]

Patients

    Age; [min_age]; [max_age]
    Sex; [M|F|Both]
    Hours; [num_hours]; [required]

Parameters

    [Abbr.]; [Description]; [Array of Mimic IDs]



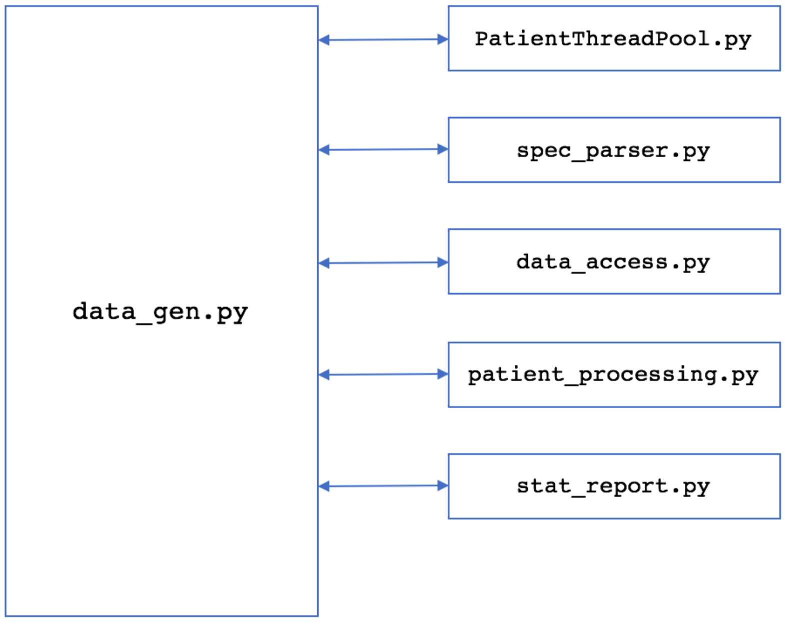
```
#ICUs

CCU      True
SICU     True
MICU     True
NICU     False
CSRU     True
TSICU    False

#Patients

Age; 16; 18
Sex; Both
Hours; 24; 1

#Parameters

Albumin; ALBUMIN; [50862,1521,226981]
```

Overview

MDGL Components

As mentioned earlier, the MDGL is comprised of six different component scripts:

1. data_gen.py
2. PatientThreadPool.py
3. spec_parser.py
4. data_access.py
5. patient_processing.py
6. stat_report.py



MDGL: data_gen.py

This script acts as the main component that combines the functionality of the other components. This script is responsible for:

- Establishing connection to the database
- Initializing data structures
- Passing data between components

MDGL: PatientThreadPool.py

This script contains the class, PatientThreadPool, which offers the functionality needed to access the database or perform computations in parallel.

This class offers two public functions: `executeFunc`, which will take in a function to execute and the parameters that should be passed directly into or split evenly between the different executions of that function, and `getResults`, which will return the results from the execution.

MDGL: PatientThreadPool.py

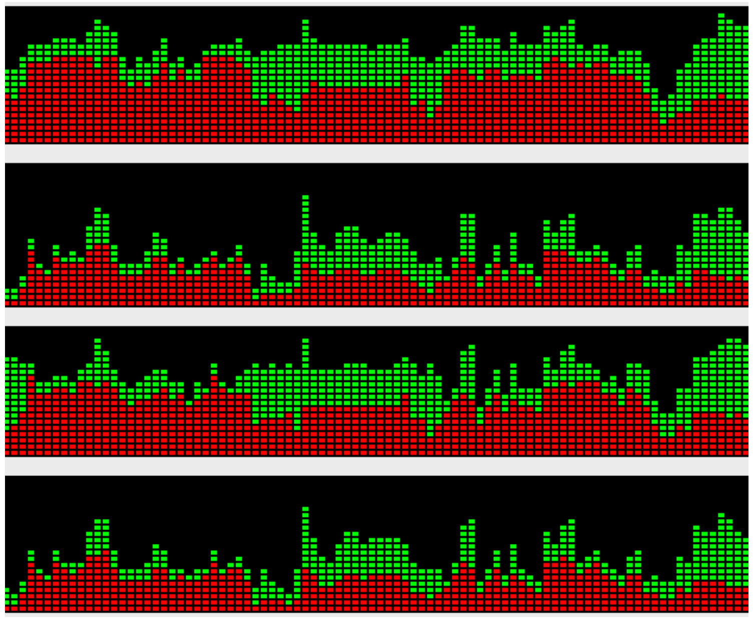
The PatientThreadPool class uses python's Threading library instead of the multiprocessing library for the following reasons:

- When certain modules (such as numpy) are imported, they will change the processor affinity, which can affect the process's ability to run in parallel. This can affect the expandability and usability of the MDGL.
- The multiprocessing.pool() function has difficulty splitting arguments that are very large in size. This considerably hurts runtime.
- The tool is targeted for local execution on a single machine. Multithreaded execution works for this case, and allows for data to be shared among processes for easier data sharing and synchronization.

MDGL: PatientThreadPool.py

Only takes between 20 and 30 minutes on a 2018 MacBook Pro

- 2.3GHz Intel i5 dual-core processor
- 8 GB RAM
- Hyper-threading enabled



Graphs of core usage during runtime for four virtual cores

MDGL: spec_parser.py

This script handles all of the functionality needed to parse information within the specification file. Currently, processing is done in one pass using the section tags (#ICUs, #Patients, #Parameters).

- Returns the information from each tag as a dictionary of information

MDGL: data_access.py

This script contains most of the functionality that will access the Mimic database.

- Patients are gathered using a single query (< 1 sec.)
- All of the more time-consuming database queries are implemented to make use of the PatientThreadPool's functionality
    - Leads to a 69% reduction in runtime when running queries in parallel on 4 virtual cores instead of on one physical core

MDGL: patient_processing.py

This component is responsible for processing all of the selected parameters' measurements. This includes handling erroneous input values, representing different parameters such as mechanical ventilation, or interpreting inequalities or values that do not directly translate to a numeric representation.

- Similar to the functions within `data_access.py`, the main patient processing function is implemented so that it can use the PatientThreadPool class for parallel computation

MDGL: stat_report

This last component generates a report that contains statistical information about the dataset generated. This includes:

- The number of patients
- The parameters selected
- Distributions, patients that had, and the total count of occurrences for all of the parameters

Overview

1. Introduction
2. Specifications
3. MDGL Components
4. Conclusion

Conclusion

Overall, the MDGL makes it very easy to create standardized patient datasets from Mimic that can be easily reproduced, shared, and specified by a user.

For future expansion, this tool can be updated to use existing data processing scripts that exist within the Mimic Code Repository.

Thank You!

Feel free to ask any questions.

THE UNIVERSITY OF TENNESSEE KNOXVILLE

References

[1] Alistair EW Johnson, David J Stone, Leo A Celi, Tom J Pollard; The MIMIC Code Repository: enabling reproducibility in critical care research, *Journal of the American Medical Informatics Association*, Volume 25, Issue 1, 1 January 2018, Pages 32–39, https://doi.org/10.1093/jamia/ocx084