SWEMLS: Coursework 3: Implementing inference

Background

Before deploying a system to detect acute kidney injury from blood tests in the production hospital environment, we're going to run one locally against a simulator.

Goals

- Build a system that runs in a single docker container that can detect acute kidney injury from a stream of HL7 messages.

Non-goals

- Build a system that requires more components than a single docker container.

High-level design

The scaffolding for the coursework can be downloaded here. It contains:

- A simple simulator capable of replaying saved HL7 messages over the MLLP protocol, and answering pager requests via HTTP posts.
- A set of sample HL7 messages, representing simulated patient admits, discharges and blood tests results over a period of a year.
- The historial blood test results for patients that could be admitted to the hospital.
- A set of expected AKI events, by patient MRN and time, allowing you to test your simulator.

Detailed design

Simulator

You can run the simulator from the scaffolding with:

```
./simulator.py
```

By default, it will replay the messages from the `messages.mllp` file via MLLP on port 8440, and respond to pager requests via HTTP on port 8441. These defaults can be overridden with flags, use:

```
./simulator.py --help
```

to see the available flags. The simulator is similar to the real hospital environment in that:

- The simulator will send one message at a time, and that message needs to be acknowledged before it will send another.

- If a client does not conform to the MLLP protocol, or sends an invalid acknowledgement, it will close the connection.
- If a client sends a negative acknowledgement to the simulator, it will resend the previous message.

Unlike the real hospital environment:

- The simulator will replay all messages to each new client. This is to allow you to easily rerun your inference code without restarting the simulator. In the real hospital environment, each message will only be sent once.
- After acknowledging a message, the simulator immediately sends the next message, independent of the time that should elapse between messages. In the real hospital environment the messages will be delivered in real time.

MLLP acknowledgements

The simulator requires your code to send acknowledgement messages. After receiving a HL7 message, you should reply with an acknowledgement similar to this:

```
MSH|^~\&|||||20240129093837||ACK|||2.5
MSA|AA
```

In this message, the AA stands for "application accept", and will cause the simulator to send the next message. A value of `AE` or `AR` (for error or reject) will cause the simulator to resend the previous message.

Scaffolding files

The coursework scaffolding contains a number of files:

- `simulator.py`: the simulator described above
- `messages.mllp`: a set of HL7 messages representing the flow of patients in the hospital over a year. The file begins with the historical admit messages for all patients currently in the hospital, to help your code establish the current state of the hospital. The real hospital environment will use a similar strategy when you first connect to their MLLP server.
- `history.csv`: the historial blood test results of all patients in the hospital, along with their mrn. This mimics the `/data/history.csv` file that will be provided in the assessment environment.
- `aki.csv`: blood test results that should trigger a pager request, including the patient's MRN and the blood test result time. You could, if you like, use this to build an integration test for your code. It won't be supplied in the real hospital environment.

Assessment environment

To assess your submission, our automated test system will:

- Checkout the version of your code from gitlab, using the last commit hash submitted to Scientia.
- Build a docker image from your code by running:
  `docker build -t coursework3 .`
  This implies your code will need a `Dockerfile` in its root directory.
- Setup environment variables `MLLP_ADDRESS` and `PAGER_ADDRESS` to tell your code how to connect to the assessment simulator.
- Provide historical blood test results in `/data/history.csv`
- Run your docker image, calculating an f3 score based on its requests to the pager service, and latency metrics based on the time between the last blood test for a patient being sent, and a page request for that patient being received.

Engineering quality

You're now working as a team, and to ensure you can successfully collaborate on the same codebase, you'll need a higher level of engineering quality. All changes should be made under version control via gitlab. Changes pushed to gitlab should have reasonable descriptions that allow others to understand the intent of a change. Changes should be reviewed by another team member before merging.

All functionality should be accompanied by unit tests, and those tests should run quickly. You will likely find it useful to add a slower integration test too.

Your implementation doesn't have to follow the design document submitted for coursework 2, in fact, we'd expect it to evolve during implementation - however, your design document should be kept to-up-date with any changes. It may be reassessed later.

Marking

The marks for this coursework will be broken down as follows:

- 40% for maintaining an f3 score of >0.70
- 20% × clamp((f3 score - 0.70) / (0.95 - 0.70), 0.0, 1.0)
- 20% × 1 / (max(90%ile latency - 3s, 0) + 1)
- 20% assessed engineering quality