Reading binary

Practical Assignment 1

Informatics 1 for Biomedical Engineers

Graz University of Technology

Abstract

The analysis of data over time is a common task in many fields. In most cases any entry in a time series is marked with a timestamp. Short biomedical sensor data on the other hand are commonly encoded as continuous stream. Here the value of every sensor is recorded once every frame. The analyst can rely on a fixed interval for the encoded data (1000 frames per second in our case). In the end the result is stored as a binary file. Reading and understanding data encoded in such a basic format is thus a vital skill for anyone working in the field of biomedical science.

The PTB Diagnostic ECG Database - `https://www.physionet.org/physiobank/database/ptbdb/`

1. Tasks

In this assignment the student's task is to prepare real-world data for future use. Biomedical sensor data is frequently encoded in binary format. This keeps the file size down (recordings quickly become huge) but does not make the data easy to process. Thus if we want to work with the data in Python we first need to transform it into a usable format.

The data provided for this course are from the The PTB Diagnostic ECG Database[1].

1.1. Download a data package (0 pt)

Download URL: `https://www.physionet.org/physiobank/database/ptbdb/` Since we use real world data you can download the direct source.

Feel free to implement your program with any patient data set. We will be testing your program with *patient001*, specifically, the *s0010_re* dataset. We have uploaded this set to our github. You can also pull it from there![2]

Whilst this may not be interesting right now playing with various datasets may yield interesting results in future tasks!

In your package you will find 2 files.

$<$ *dataset_index* $>$ *.dat* contains the binary sensor information. This is the file you will be working with. 12 common sensor leads are encoded frame by frame. Meaning you will get the first 2 bytes for each sensor (frame 0) before the second frame starts once again with lead 0 (i).

$<$ *dataset_index* $>$ *.hea* is the header file describing how to read the data. Details are listed in section 1.3

$<$ *dataset_index* $>$ *.hea* . These are sensor data for the 3 Frank lead ECGs. We will not be using them this course! The file is added purely for the sake of completeness!

[1] `https://www.physionet.org/physiobank/database/ptbdb/`

[2] `https://github.com/pkasper/info1-bm/tree/master/assignments/data`

1.2. Open the binary input file (2 pt)

Open the input data file (*.dat*) you downloaded. Whilst python usually takes care of the file management please make sure your program explicitly closes the file handle when you are done reading. This can be done by either calling *.close()* or via the use of a *with* statement. Remember that these are binary files so you want to pass the "b" flag when opening! **IMPORTANT:** Your submission should use the command line parameters and expect any data files in the root directory (same directory as your python source file).

1.3. Parse the data into the data structure (7pt)

In order to be able to read the input data you will have to look at your header file (.hea). For the task at hand only the first line is important.

Header file example

```
s0010_re 15 1000 38400
s0010_re.dat 16 2000 16 0 -489 -8337 0 i
s0010_re.dat 16 2000 16 0 -458 -16369 0 ii
s0010_re.dat 16 2000 16 0 31 6829 0 iii
s0010_re.dat 16 2000 16 0 474 4582 0 avr
s0010_re.dat 16 2000 16 0 -260 11687 0 avl
s0010_re.dat 16 2000 16 0 -214 -16657 0 avf
s0010_re.dat 16 2000 16 0 -88 -12469 0 v1
s0010_re.dat 16 2000 16 0 -241 5636 0 v2
s0010_re.dat 16 2000 16 0 -112 -14299 0 v3
s0010_re.dat 16 2000 16 0 212 -17916 0 v4
s0010_re.dat 16 2000 16 0 393 -6668 0 v5
s0010_re.dat 16 2000 16 0 390 -17545 0 v6
s0010_re.xyz 16 2000 16 0 -3 -13009 0 vx
s0010_re.xyz 16 2000 16 0 120 7109 0 vy
s0010_re.xyz 16 2000 16 0 -18 -1992 0 vz
```

The first line states $s0010\_re$ 15 1000 38400. The first number is your patient number used to identify the files. The second number is the amount of leads present in the data. 1000 frames are recorded per second and 38400 frames were recorded in total ($38.4 seconds$). The last value may be different in your data. Since we use real word samples not all ECGs are of the same length.

The next lines show the individual leads and the order how they are stored in the file. The pre-defined data-structure is ordered in the same way so you do not need to worry about that. The first value marks the file the data is in, the second shows the size of each recorded value. 16 means you

are dealing with 16-bit (2-byte) entries. Thus, you want to read 2 bytes per value. (2-byte numbers are often referred to as *short*) Everything else should be ignored for this course! Please note that we will only be working with the 12 conventional leads which are all stored in the .dat file and note the 3 Frank leads (stored in the .xyz). Thus, you only have 12 values in your data structure! Please copy the following data structure into your code. (Naturally, this will not be subject to plagiarism checks). Note: Depending on your reader you may have to manually fix the whitespaces and indentation!

Data structure:

```
sensor_data = {0: {"name": "i", "data": []},
               1: {"name": "ii", "data": []},
               2: {"name": "iii", "data": []},
               3: {"name": "avr", "data": []},
               4: {"name": "avl", "data": []},
               5: {"name": "avf", "data": []},
               6: {"name": "v1", "data": []},
               7: {"name": "v2", "data": []},
               8: {"name": "v3", "data": []},
               9: {"name": "v4", "data": []},
               10: {"name": "v5", "data": []},
               11: {"name": "v6", "data": []}
              }
```

For this task it is not essential that you fully understand how this data structure is built up and how it works. Each sensor addressed via its index (0 – 11) and you want to add your frames in the *data*. This can be done as follows:

```
#  Add 1 frame to to sensor 0 ("i")
#  'value' is whatever you read from the binary file
#  for sensor 0 at the current frame

sensor_data[0]['data'].append(value)
```

The individual elements of the data blocks should be normal integers. At the end the length of each of your *data* elements in the sensor_data should should be equal to the total amount of frames you read (38400 in our example case).
Whilst there are other ways to read binary data we suggest using the

struct-package (import struct). Please refer to the documentation [3] how to use the unpack function. (Hint: Check the format characters section in the documentation and you will want to read signed shorts! Also this function always returns a tuple and you will only need the first element.)
The data in these files are stored in a parallel format. This means you get the sensor values for the first frame from all 12 sensors followed by the second frame for, again, all 12 sensors. This is important for your implementation!

1.4. Dump data as pickle (1pt)
As a last part you should please dump the data structure with all its data into a pickle file. For this you want to use *import pickle* and use the *dump()* function. Once again you will want open the output file with the binary flag when you pass it to *dump()*. Your output file should be named "$<dataset\_index>.p$". (In our example case this would be $s0010\_re.p$)
The output should be stored in the same directory as your python script!

2. Additional Notes

Remember, the signal data is stored in the form of 16 bit integers. When you read the data make sure to not read too many bytes as integers are commonly 32 or 64 bit. The *struct* package is meant to deal with these issues. Though, we do not enforce the usage as there are other options to solve this problem without the use of any package.

2.1. Command Line Parameters
Your program should work with command line parameters. Execute it with the command *python assignment_1.py <data_file_name> <num_frames>*. Your submission should expect the files in the same directory as the python source! Remember, *sys.argv* returns strings so you will have to convert the frame number to an integer!

2.2. Output
Make sure the file your script creates has the same identifier as the input (as defined by the command line parameter). Calling your program with *assignment_1.py s0010_re.dat 38400* should create a file called *s0010_re.p*! (Hint: String slicing)

[3] `https://docs.python.org/2/library/struct.html#struct.unpack`

3. File Headers

All python source files have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please feel free to copy and paste the example and change its contents to your data. (Any comments will not be subject for plagiarism checks)! Note: Depending on your reader you may have to manually fix the whitespaces and indentation!
Example:

```
##############################################################################
# Author:      Patrick Kasper
# MatNr:       0730294
# Description: The main file. Assignment only has 1 file...
# Comments:    This is the example comment. I just made it a bit
#              longer so it spans across multiple lines.
##############################################################################
```

4. Allowed packages

The following packages are allowed to use in this assignment:

- sys
- pickle
- struct

5. Restrictions

- Do not add any bloat to your submission
- Use built-in functions where possible (and reasonable)
- Do NOT load extra packages (no imports) except those listed in section 4
- Do NOT require any extra command line arguments!

6. Coding Standard

For this lecture and the practicals we use PEP 8 [4] as a coding standard guideline. Please note that whilst we do not strictly enforce all these rules we will not tolerate messy or unreadable code. Please focus especially on the following three aspects.

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in python 3 (whilst allowing more freedom in python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all use a single underscore (_) for its name.

Max 72 characters. Do not have lines longer than 72 characters. Whilst PEP 8 allows 79 in some cases we ask you to use the lower cap of 72 characters per line. Please note that this includes indentation.

Not following this coding standard will result in a deduction from your achieved points!

7. Self testing

Naturally, you may want to test your program before submitting it. Instructions regarding how to do that can be found on the Wiki [5].

[4] `https://www.python.org/dev/peps/pep-0008/`
[5] `https://palme.iicm.tugraz.at/wiki/Info1BM#Assignment_Self_Tests`

8. Automated Tests

We will test your submissions automatically. The test machines will have the latest anaconda version on the 5$^{\text{th}}$ of October 2016. Your file will be executed with the following command:

```
>python assignment_1.py s0010_re.dat 38400
```

Please make sure your submission follows all the restrictions defined in this document. Your program will have a limited runtime of 2 minutes. If your submission exceeds this threshold then it will be considered non-executable. Please note that this is a very generous amount of time for any of the tasks in this course.
If your submission fails any of the automated tests we will look into your code to ensure the reason was not on our end and to evaluate which parts of your code are correct and awards points accordingly.

We will be testing the following aspects:

- General coding errors (Does the program even run?)
- Do the files contain the correct comment headers?
- Your output with the provided input (*s0010_re*.dat)
- Your output with different input

9. Submission

9.1. Deadline

15^{th} of November 2016 at 23:59.

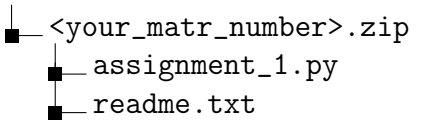
Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will automatically be pushed back for 24 hours.
If for whatever reason you are unable to submit your submission prior to the deadline please contact your tutor BEFORE the deadline.

9.2. Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your python source files please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. This is important to us as immediate feedback so we can adjust the tutor sessions. Please note that all submissions will be anonymised prior to being read. Hence please feel free to be honest (but do stay polite).
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File names and folder structure (see below)
- Comment headers in every source file
- Coding standard upheld
- You do not hard-code the file name and the frame amount!

9.3. Your submission file



Do not submit any additional files. You do not need to submit the pickle file (.p) your script creates!