“Understanding Biomedical Timeseries-Data”

Übungsaufgabe 1

Informatik 1 für Biomedical Engineering

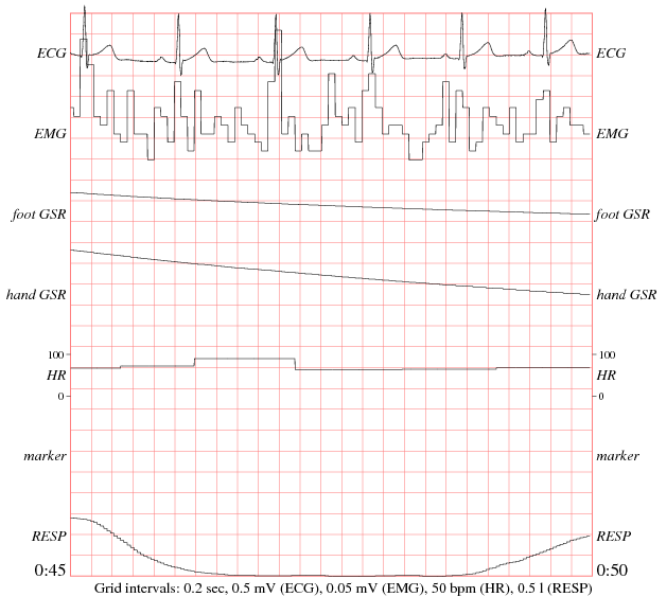
Technische Universität Graz

Abstract

Biomedizinische Zeitserien—so zum Beispiel EKG Aufzeichnungen—sind präzise und oft hochfrequente Serien von Sensorwerten, welche schnell viel Speicher benötigen. Um diese möglichst kompakt, aber dennoch offen zu speichern, wurden eine Reihe von Dateiformaten entwickelt. Eines der verbreitetsten ist WFDB[1]. Dieses trennt die eigentlichen Daten von ihrer Beschreibung.

Um mit derartigen Datensätzen umgehen zu lernen, widmet sich dieses Assignment dem Parsen von so genannten Header (.hea) Dateien im wfdb Format. Diese beinhalten eine Beschreibung dessen, welche Informationen in den Binärdateien kodiert sind und wie diese auszulesen sind.

Im Rahmen dessen lernen Sie, wie Kommandozeilenparameter funktionieren, wie Sie mit Datei Ordnersturkturen umgehen, wie man Dateien liest und Informationen extrahiert, und wie man Python Datenstrukturen wie Listen, Dictionaries und Sets verwendet.



[1] `https://physionet.nlm.nih.gov/physiotools/wpg/wpg.pdf`

1. Tasks

1.1. Datensatz (0pt)

In dieser Vorlesung wird mit realen Daten gearbeitet. Der *drivedb* Datensatz beinhaltet ein Sammlung von Aufzeichnungen gesunder Menschen während dem Abfahren einer vordefinierten Route. Ursprüngliches Ziel war es, automatisch Stress-Zustände des Fahrers zu erkennen.

Laden Sie bitte den Datensatz über eine der angegebenen Ressourcen herunter. Unter Linux kann folgender Befehl verwendet werden:

```
rsync -Cavz physionet.org::drivedb <Ziel-Ordner>
```

Alternativ verwenden Sie das angebotene Archiv und entpacken dies in einen Ordner. Dieser soll über einen relativen Pfad erreichbar sein—am besten als Unterordner im selben Verzeichnis, wie Ihr Code.

Ressourcen:

- `https://physionet.org/physiobank/database/drivedb/`
- Fallback: `https://physionet.nlm.nih.gov/pn3/drivedb/`
- Fertiges Archiv: `https://www.dropbox.com/s/k3p69532r2a9vki/dataset.zip?dl=1`

1.2. Kommandozeilenparameter (1pt)

Ihr Programm soll optional mit einem Kommandozeilenparameter aufgerufen werden. Dieser definiert den Pfad zur `RECORDS` Datei, welche Informationen über die in der Datensammlung enthaltenen Aufzeichnungen liefert.

Trennen Sie den Pfad (Kommandozeilenparameter) in Ordner und Dateinamen. Sollte der Parameter NICHT verwendet werden, setzen Sie den Pfad zu dem Datensatz auf `DATA/` und den Dateinamen auf `RECORDS`. **Hinweis:** Im Ersten Schritt ist es empfehlenswert, die Datei auch wirklich unter `DATA/` abzulegen.

Sollten der gewünschte Ordner oder die Datei nicht vorhanden sein, geben Sie eine beschreibende Fehlermeldung aus und beenden Sie das Programm mit `exit()`.

Beispielaufruf MIT Parameter:

```
python assignment_1.py DATA/RECORDS
```

1.3. RECORDS Datei und Filechecks (2pt)

Die RECORDS Datei ist eine plaintext Datei (Sie können diese mit einem beliebigen Texteditor öffnen) und enthält die Namen der einzelnen Aufzeichnungen. In jeder Zeile befindet sich ein Name. (**Hinweis:** Unter machen Editoren kann es so aussehen, als würde alles in einer Zeile stehen. Sie können sich aber darauf verlassen, dass die `readlines()` Funktion diese richtig trennt!). Extrahieren Sie alle Namen und überprüfen Sie, ob zu jedem Namen eine *.hea* Datei vorhanden ist. (**Hinweis:** verwenden Sie `strip()` um das Umbrunchszeichen und Whitespaces an den Rändern der Namen automatisch zu entfernen!)

Erstellen Sie für jede Aufzeichnung in *RECORDS* eine Ausgabe nach folgendem Schema: .hea ist vorhanden:

```
Record: <name> ... OK

# Example:
Record: drive01 ... OK
```

.hea ist **NICHT** vorhanden:

```
Record: <name> ... NOT FOUND - skipping

# Example:
Record: drive01 ... NOT FOUND - skipping
```

Für den weiteren Ablauf verwenden Sie ausschließlich die tatsächlich gefundenen Header Dateien.

1.4. Parsen der Header (4pt)

Öffnen Sie nach der Reihe alle gefundenen Header Dateien und extrahieren Sie die Informationen. Eine ausführliche Anleitung finden Sie auf der Webseite von Physionet[2][3]. Im Rahmen dieser Übung reicht aber folgende Kurzfassung:

Informationen sind im Normfall durch Whitespaces getrennt. Die Namen in den `< >` beschreiben immer die Namen des Keys in Ihrem Dictionary.

Erste Zeile:

Record Name <record_name> Name der Aufzeichnung (sollte gleich dem Dateinamen ohne Fileextension sein)

Anzahl der Signale <num_signals> Anzahl der Signale (sollte gleich der Anzahl der restlichen Zeilen in der Datei sein)

Sampling Frequenz <frequency Aufzeichnungsfrequenz. Dies ist eine Gleitkommazahl. Konvertieren Sie das Gelesene mit `.float()`

Anzahl der Frames <num_samples> Wie viele Werte gibt es pro Signal

Restlichen Zeilen (Signal Zeilen):

Dateiname <file_name> Name der Datei, welche die eigentlichen Informationen enthält (sollte pro Header Datei konstant sein).

Format <data_format> Format der gespeicherten Daten. Sollte dieser Eintag ein `x` beinhalten (z.B.: `16x2`) dann ist das Format alles vor dem `x` (Im Beispiel also `16`).

Samples pro Frame <samples_per_frame> 1 wenn nicht definiert. Sonst, der Wert nach dem `x`.

[2] `https://www.physionet.org/physiotools/wag/header-5.htm`

[3] `https://physionet.nlm.nih.gov/physiotools/wag/header-5.htm`

Einheit <unit> Einheit der Signalwerte. Dies ist immer ein String. Versuchen Sie nicht, dies zu konvertierten.

ADC Auflösung <adc_resolution> Präzision des Sensors (für die Übung nicht weiter relevant).

ADC Kalibrierung <adc_zero> 0 Wert eines vergleichbaren Analogen Sensors (für die Übung nicht weiter relevant).

Initialwert <initial_value> Erster Sensorwert.

Prüfsumme <checksum> Prüfsumme aller Werte des Sensors für eventuelle Integritätschecks.

Blockgröße <block_size> Nicht relevant (sollte immer 0 sein)

Sensor Name Name des Signals. Dies ist alles nach der Blockgröße und kann Leerzeichen beinhalten!

Erstellen Sie ein Dictionary für jede Header Datei, welches die gelesenen Informationen Speichert. Ein genaues Beispiel finden Sie in Appendix A. Halten Sie sich an die definierten Key Namen. Das *signals* Feld soll selbst ein eigenes Dictionary für die einzelnen Signale sein. Verwenden Sie hier die Namen der Signale (z.B.: "ecg", oder "foot gsr") als Key.

Sie brauchen den Inhalt der Datei nicht zu verifizieren (z.B.: prüfen, ob die Signalanzahl der ersten Zeile stimmt usw).

Wichtig: Beim Lesen einer Datei liefert Python (sofern nicht anders exakt definiert) Strings zurück. Konvertieren Sie alle Zahlen zu Integern oder Floats. Konvertieren Sie alle Strings zu Kleinbuchstaben (`.lower()`) um Probleme zu vermeiden! Verwenden Sie auch die hier definierten Namen für ihre Keys im Dictionary!

Erstellen Sie ein Dictionary, welches die Namen der Records (z.B.: "drive01") als Keys, und die zugehörigen Dictionaries (die vorhin pro Header erstellten) als Werte hat.

Hinweis: Auch diese Dateien sind im plaintext Format. Sie können sie also problemlos mit jedem Texteditor öffnen.

1.5. Speichern als Pickle (1pt)

Verwenden Sie die `pickle.dump()` Funktion[4] um das vorhin erstellte Dictionary zu speichern. Der Name soll *ass_1.p* lauten.

Hinweis: pickle speichert in einem Binärformat. Offen Sie daher die Datei, welche die `dump()` Funktion verwenden soll, mit den korrekten Parametern (*"wb"*, 'w' für Schreiben und 'b' für Binary)

1.6. Zusammenfassung der Inhalte (2pt)

Fassen Sie die Informationen, die Sie extrahiert haben anhand der Signale zusammen. Listen Sie, welche Signale Sie gefunden haben, in wie vielen Records diese vorhanden sind und eine Auflistung jener Records, wo die jeweiligen Signale fehlen. Ihr output soll exakt dem hier gelisteten Beispiel entsprechen.

[4] `https://docs.python.org/3.6/library/pickle.html#usage`

Zeichnen Sie eine Box aus Rauten (`#` um diese Ausgabe, und leiten Sie die Auflistung mit `SIGNALS:` ein.Sortieren Sie sowohl die Sensornamen als auch die Records nach Name. Die Whitespaces bauen sich wie folgt auf: 1 vor SIGNALS, 3 vor jedem Signalnamen. Das padding zwischen Signalnamen und der Anzahl der Records soll 3 vom längsten Signalnamen sein. (z.B.: *foot gsr* benötigt 3 und *ecg* benötigt 8). Die Anzahl der Ziffern bei records soll auf 2 gepadded sein (also auch 2 Spaces benötigen, wenn die Zahl nur 1 Stelle hat).

Wichtig: Nicht die Signalnamen hardcoden (direkt in den Code schreiben).

Exakte Lösung: (Beim vollen Datensatz)

```
########################################################################
# SIGNALS:
#   ecg        18 record(s), missing:
#   emg        15 record(s), missing: drive02, drive03, drive04
#   foot gsr   18 record(s), missing:
#   hand gsr   16 record(s), missing: drive02, drive13
#   hr         16 record(s), missing: drive03, drive14
#   marker     16 record(s), missing: drive01, drive03
#   resp       18 record(s), missing:
########################################################################
```

Hinweis: Verwenden Sie Format Strings[5] um die Ausgabe leichter zu machen. Die 72 `#` Zeichen können einfach mit `print("#"*72)` geschrieben werden.

2. Packages

Folgende Packages sind für dieses Assignment zugelassen und/oder notwendig:

- collections (defaultdicts[6] können hilfreich sein!)
- os
- pickle
- sys

3. Beschränkungen

- Fügen Sie keine nutzlosen Komponenten Ihrer Abgabe hinzu
- Sofern möglich, verwenden Sie eingebaute Funktionen
- Importieren Sie keine extra zusätzlichen packages
- Verwenden Sie keine nicht angeführten Kommandozeilenparameter

[5] `https://docs.python.org/3.6/library/stdtypes.html#str.format`

[6] `https://docs.python.org/3/library/collections.html#collections.defaultdict`

4. Datei Header

All Ihre Quelldateien in Ihrer Abgabe müssen gleich zu Beginn einen Kommentar mit folgenden Informationen enthalten:

- Author: – Ihr Name
- MatNr: – Ihre Matrikelnummer
- Description: – Generelle Beschreibung der Datei
- Comments: – Kommentare, Erklärungen, usw

Bitte einfach den folgenden Code in Ihre Dateien kopieren und den Inhalt anpassen. Je nach PDF-Reader müssen Sie eventuell die Leerzeichen/Einrückungen per Hand anpassen.

Beispielheader:

```
##############################################################################
# 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.
##############################################################################
```

5. Coding Standard

Für diese Lehrveranstaltung orientieren Sie sich am offiziellen PEP 8 Standard[7]. Dieser Beschreibt grundsätzliche Formalitäten im Bezug auf Ihren Code. Folgendes ist besonders zu Beachten:

Sprache. Code schreibt man in Englisch. Im internationen Zeitalter ist es notwendig, dass auch jemand am anderen Ende der Welt verstehen kann, was Sie programmiert haben. Ihr gesamter Quellcode muss daher auf Englisch geschrieben sein. Dies betrifft sowohl die Kommentare also auch Variablennamen und Ähnliches.

Leerzeichen statt Tabulatoren. Python basiert auf Einrückungen, anstatt auf geschwungenen Klammern . Theoretisch gibt es die Möglichkeit, Leerzeichen (spaces) oder Tabulatoren (tabs) zu verwenden. PEP8 schreibt aber, 4 Leerzeichen als Einrückungen vor. Die meisten Python Programmierumgebungen werden automatisch 4 Leerzeichen einfügen, wenn Sie auf die Tabulator-Taste drücken.

Sprechende Namen. Verwenden Sie kurze, aber sprechende Namen für Ihre Variablen, Funktionen, usw. Es muss eindeutig aus dem Namen hervorgehen, was die Aufgabe des Elements ist. Für simple Iterationen kann ein einfaches `i` ausreichend sein, aber dies kann schnell zu Chaos führen. Sollten Sie Variablen haben, die keine Aufgabe haben und nicht verwendet werden, schreibt PEP 8 vor, einen einfachen Unterstrich (_) zu verwenden.

[7] `https://www.python.org/dev/peps/pep-0008/`

72 **Zeichen Zeilenlänge.** In Ihrem Code darf keine Zeile mehr als 72 Zeichen haben. Dies ist Voraussetzung, um den Code ausdrucken, oder auf allen Geräten darstellen zu können. PEP 8 limitiert in manchen Fällen die Länge auf 79 Zeichen und in anderen auf 72. Bitte halten Sie sich in Ihrem Code an die untere Schranke (72). **Hinweis:** Dies inkludiert die Einrückungen mittels Leerzeichen!

6. Automatisierte Tests

Ihr Programm wird mit dem folgenden Befehl ausgeführt:

```
>python assignment_1.py <path_to_records_file>
```

Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist. Beim Ausführen erhält Ihr Programm 2 Minuten Laufzeit. Sollte dieses Limit überschritten werden, gilt ihr Programm als unausführbar. (Die bereitgestellte Laufzeit ist stets sehr großzügig bemessen!)
Sollte Ihre Abgabe einzelne automatische Tests nicht bestehen, wird der Code genauer inspiziert und Punkte anteilshaft vergeben.

7. Abgabe

7.1. Deadline

14. November 2017 um 23:59:59

Eine Spätabgabe ist nicht vorgesehen. Ausnahme bilden hier Notfälle. Sollte das Abgabesystem nicht online sein, verlängert sich die Deadline automatisch um 24 Stunden. Sollten Sie, aus diversen Gründen, nicht in der Lage sein, Ihre Abgabe hochzuladen, kontaktieren Sie Ihren Tutor *VOR* der Deadline. (**Hinweis:** Urlaub oder Ähnliches wird nicht als Grund akzeptiert!)

7.2. Hochladen der Abgaben

Assignments werden stets als Archive abgegeben. Erlaubt sind hier die Formate *.zip*, und *.tar.gz*.
Zusätzlich zu ihren Quelldateien, soll Ihre Abgabe auf eine Datei namens `readme.txt` beinhalten. Das Vorhandensein der Datei ist Pflicht, ihr Inhalt aber optional. Sie soll folgenden Inhalt haben: (i) Die Zeit, die Sie benötigt haben, um die Aufgabenstellung zu absolvieren. (ii) Feedback, wo Sie Probleme hatten. Ihr Feedback ermöglicht es, verbreitete Probleme zu erkennen und die Vorlesung und Tutoriumseinheiten entsprechend anzupassen.
Abgaben erfolgen auf der Palme Website. Bitte prüfen Sie vor der Abgabe diese Kriteren:

- Datei- und Ordnerstruktur (siehe unten)
- Kommentarheader in jeder Quelldatei
- Coding Standard

7.3. Struktur der Abgabe

```
assignment_1.zip (or assignment_1.tar.gz)
    assignment_1.py
    readme.txt
```

Bitte fügen Sie den Datensatz **NICHT** Ihrer Abgabe hinzu. Dieser wird automatisch bei den Tests in die entsprechenden Ordner kopiert.

Appendix A. Dictionary Beispiel für `drive01`

Beispiel der Lösung :

```
>>>parsed_records['drive01'] # parsed_records is just an example name
{
 'num_samples': 61499,
 'num_signals': 6,
 'record_name': 'drive01',
 'frequency': 15.5,
 'signals': {
  'ecg': {
   'adc_resolution': 16,
   'adc_zero': 0,
   'block_size': 0,
   'checksum': 9084,
   'data_format': 16,
   'file_name': 'drive01.dat',
   'initial_value': -42,
   'samples_per_frame': 32,
   'unit': '1000'
  },
   'emg': {
   'adc_resolution': 16,
   'adc_zero': 0,
   'block_size': 0,
   'checksum': 11155,
   'data_format': 16,
   'file_name': 'drive01.dat',
   'initial_value': 68,
   'samples_per_frame': 128,
   'unit': '10000'
  },
  'foot_gsr': {
   'adc_resolution': 16,
   'adc_zero': 0,
   'block_size': 0,
   'checksum': -24751,
   'data_format': 16,
   'file_name': 'drive01.dat',
   'initial_value': 2503,
   'samples_per_frame': 2,
   'unit': '1000'
  },
  'hand_gsr': {
   'adc_resolution': 16,
   'adc_zero': 0,
   'block_size': 0,
   'checksum': 20466,
   'data_format': 16,
   'file_name': 'drive01.dat',
   'initial_value': 11149,
   'samples_per_frame': 2,
   'unit': '1000'
  },
  'hr': {
   'adc_resolution': 16,
   'adc_zero': 0,
   'block_size': 0,
   'checksum': 18582,
   'data_format': 16,
   'file_name': 'drive01.dat',
   'initial_value': 84,
   'samples_per_frame': 1,
   'unit': '1/bpm'
  },
  'resp': {
   'adc_resolution': 16,
   'adc_zero': 0,
   'block_size': 0,
   'checksum': -19336,
   'data_format': 16,
   'file_name': 'drive01.dat',
   'initial_value': 5474,
   'samples_per_frame': 1,
   'unit': '500'
  }
 }
}
```

Hinweis: Bei den Keys der Signale (in `signals`) handelt es sich im Marker für Whitespaces und **keine** Underscores (`_`)