Construe

Construe is a knowledge-based abductive framework for time series interpretation. It provides a knowledge representation model and a set of algorithms for the interpretation of temporal information, implementing a hypothesize-and-test cycle guided by an attentional mechanism. The framework is fully described in the following paper:

[1]: T. Teijeiro and P. Félix: *On the adoption of abductive reasoning for time series interpretation*, Artificial Intelligence, 2018, vol. 262, p. 163-188. DOI:10.1016/j.artint.2018.06.005.

In this repository you will find the complete implementation of the data model and the algorithms, as well as a knowledge base for the interpretation of multi-lead electrocardiogram (ECG) signals, from the basic waveforms (P, QRS, T) to complex rhythm patterns (Atrial fibrillation, Bigeminy, Trigeminy, Ventricular flutter/fibrillation, etc.). In addition, we provide some utility scripts to reproduce the interpretation of all the ECG strips shown in paper [1], and to allow the interpretation of any ECG record in the MIT-BIH format with a command-line interface very similar to that of theWFDB applications.

Additionally, the repository includes an algorithm forautomatic heartbeat classification on ECG signals, described in the paper:

[2]: T. Teijeiro, P. Félix, J.Presedo and D. Castro: *Heartbeat classification using abstract features from the abductive interpretation of the ECG*, IEEE journal of biomedical and health informatics, 2018, vol. 22, no 2, p. 409-420. DOI: 10.1109/JBHI.2016.2631247 .

The *Construe* algorithm is also the basis for the arrhythmia classification method described in the following papers:

[3]: T. Teijeiro, C.A. García, D. Castro and P. Félix: *Arrhythmia Classification from the Abductive Interpretation of Short Single-Lead ECG Records*, Computing in Cardiology, 2017, vol. 44, p. 1-4.DOI: 10.22489/CinC.2017.166-054.

[4]: T. Teijeiro, C.A. García, D. Castro and P. Félix: *Abductive reasoning as the basis to reproduce expert criteria in ECG Atrial Fibrillation identification*. Physiological Measurement, 39(8), 084006. DOI: 10.1088/1361-6579/aad7e4

This method won the First Prize in the Physionet/Computing in Cardiology Challenge 2017, providing the best results in Atrial Fibrillation detection among the 75 participating teams.

Installation

This project is implemented in pure python 3, so no installation is required. However, the core algorithms have strong dependencies with the following python packages:

1. sortedcontainers
2. numpy
3. python-dateutil

In addition, the knowledge base for ECG interpretation depends on the following packages:

4. scipy
5. scikit-learn
6. PyWavelets

As optional dependencies to support the interactive visualization of the interpretation results and the interpretations tree and to run the demo examples, the following packages are also needed:

7. matplotlib
8. networkx
9. pygraphviz and graphviz

Finally, to read ECG signal records it is necessary to have access to a proper installation of theWFDB software package.

To make easier the installation of Python dependencies, we recommend the Anaconda or Miniconda Python distributions. Alternatively, you can install them using pip with the following command:

```
~$ pip install -r requirements.txt
```

Once all the dependencies are satisfied, it is enough to download the project sources and execute the proper python or bash scripts, as explained below. Please note that all our tests are performed on Linux environments, so unexpected issues may arise on Windows or OS-X environments. Please let us know if this is the case.

Getting started

Construe as a tool for ECG analysis

Along with the general data model for knowledge description and the interpretation algorithms, a comprehensive knowledge base for ECG signal interpretation is provided with the framework, so the software can be directly used as a tool for ECG analysis in multiple abstraction levels.

Interpreting external ECG records. The construe-ecg tool:

Any ECG record in MIT-BIH format can be interpreted with the *Construe* algorithm. This is done via the construe_ecg.py script, which is intended to be used as a production command-line tool that performs background interpretations of full ECG records (or sections). The result is a set of annotations in the MIT format. This tool tries to follow the WFDB Applications command-line interface. The usage of the construe-ecg application is as follows:

```
usage: construe_ecg.py [-h] -r record [-a ann] [-o oann]
                       [--level {conduction,rhythm}] [--exclude-pwaves]
                       [--exclude-twaves] [-f init] [-t stop] [-l length]
                       [--overl OVERL] [--tfactor TFACTOR] [-d min_delay]
                       [-D max_delay] [--time-limit TIME_LIMIT] [-k K] [-v]
                       [--no-merge]

Interprets a MIT-BIH ECG record in multiple abstraction levels, generating as
a result a set of annotations encoding the observation hypotheses.

optional arguments:
  -h, --help            show this help message and exit
  -r record             Name of the record to be processed
  -a ann                Annotator containing the initial evidence. If not
                        provided, the gqrs application is used.
  -o oann               Save annotations as annotator oann (default: iqrs)
  --level {conduction,rhythm}
                        Highest abstraction level used in the interpretation.
                        Using the "conduction" level produces just a wave
                        delineation for each QRS annotation in the initial
                        evidence, while the "rhythm" level also includes a
                        rhythm interpretation of the full signal, but at the
                        expense of a higher computational cost in several
                        orders of magnitude.
  --exclude-pwaves      Avoids searching for P-waves. Default:False
  --exclude-twaves      Avoids searching for T-waves. It also implies
                        --exclude-pwaves Default:False
  -f init               Begin the interpretation at the "init" time, in
                        samples
  -t stop               Stop the interpretation at the "stop" time, in samples
  -l length             Length in samples of each independently interpreted
                        fragment. It has to be multiple of 256. Default:23040
                        if the abstraction level is "rhythm", and 640000 if
                        the abstraction level is "conduction".
  --overl OVERL         Length in samples of the overlapping between
                        consecutive fragments, to prevent loss of information.
                        If the selected abstraction level is "conduction",
                        this parameter is ignored. Default: 1080.
  --tfactor TFACTOR     Time factor to control the speed of the input signal.
                        For example, if tfactor = 2.0 two seconds of new
                        signal are added to the signal buffer each real
                        second. A value of 1.0 simulates real-time online
                        interpretation. If the selected abstraction level is
                        "conduction", this parameter is ignored. Default: 1e20
```

```
  -d min_delay          Minimum delay in samples between the acquisition time
                        and the last interpretation time. If the selected
                        abstraction level is "conduction", this parameter is
                        ignored. Default: 2560
  -D max_delay          Maximum delay in seconds that the interpretation can
                        be without moving forward. If this threshold is
                        exceeded, the searching process is pruned. If the
                        selected abstraction level is "conduction", this
                        parameter is ignored. Default: 20.0
  --time-limit TIME_LIMIT
                        Interpretation time limit *for each fragment*.If the
                        interpretation time exceeds this number of seconds,
                        the interpretation finishes immediately, moving to the
                        next fragment. If the selected abstraction level is
                        "conduction", this parameter is ignored. Default:
                        Infinity
  -k K                  Exploration factor. It is the number of
                        interpretations expanded in each searching cycle.
                        Default: 12. If the selected abstraction level is
                        "conduction", this parameter is ignored.
  -v                    Verbose mode. The algorithm will print to standard
                        output the fragment being interpreted.
  --no-merge            Avoids the use of a branch-merging strategy for
                        interpretation exploration. If the selected
                        abstraction level is "conduction", this parameter is
                        ignored.
```

Some common usage examples

Perform a full interpretation of record `100` from the MIT-BIH Arrhythmia Database (the output will be stored in the `100.iqrs` annotation file):

```
$ python construe_ecg.py -r 100
```

Perform a delineation of the selected heartbeats in the `.man` annotation file for the record `sel30` from the QT database, and store the result in the `sel30.pqt` file.

```
$ python construe_ecg.py -r sel30 -a man -o pqt --level conduction
```

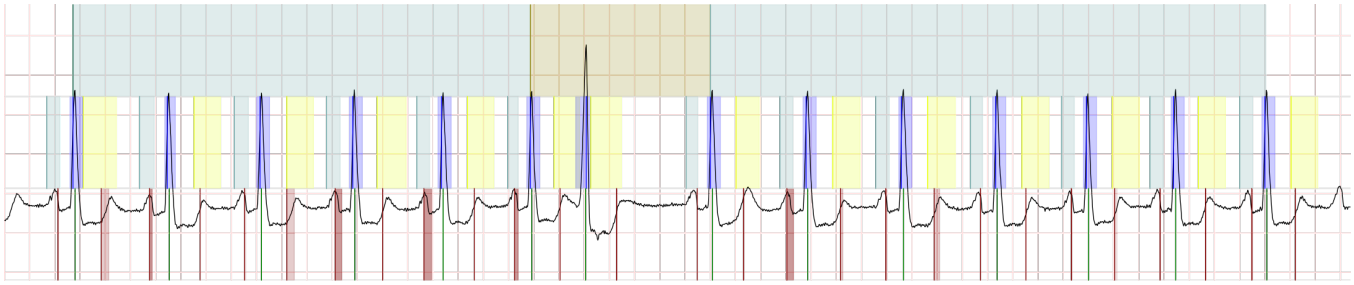
The same than before, but avoiding P-Wave delineation (only includes QRS complexes and T-waves):

```
$ python construe_ecg.py -r sel30 -a man -o pqt --level conduction --exclude-pwaves
```

Interactive demo examples

All signal strips in [1] are included as interactive examples to make it easier to understand how the interpretation algorithms work. For this, and after installing the optional dependencies described in the [installation](## Installation) section, use the `run_example.sh` script, selecting the figure for which you want to reproduce the interpretation process:

```
./run_example.sh fig4
```



Once the interpretation is finished, the resulting observations are printed to the terminal, and two interactive figures are shown. One plots the ECG signal with all the observations organized into abstraction levels (deflections, waves, and rhythms), and the other shows the interpretations tree explored to find the result. Each node in the tree can be selected to show the observations at a given time point during the interpretation, allowing to reproduce the *abduce*, *deduce*, *subsume* and *predict* reasoning steps [1].

In order to support this kind of interactive analysis in other arbitrary (short) ECG fragments, the fragment_processing.py script is provided. Please note that this tool is conceived just to give insights into the abductive interpretation algorithms and to illustrate the adopted reasoning paradigm, and not as a production tool.

Using *Construe* in other problems and domains

We will be glad if you want to use *Construe* to solve problems different from ECG interpretation, and we will help you to do so. The first step is to understand what is under the hood, and the best reference is [1]. After this, you will have to define the Abstraction Model for your problem, based on the Observable and Abstraction Pattern formalisms. As an example, a high-level description of the ECG abstraction model is available in [2], and its implementation is in the knowledge subdirectory. A tutorial is also available in the project wiki.

Once the domain-specific knowledge base has been defined, the fragment_processing.py module should serve as a basis for the execution of the full hypothesize-and-test cycle with different time series and the new abstraction model.

Repository structure

The source code is structured in the following main modules:

- acquisition: Modules for the acquisition of the raw time series data. Currently it is highly oriented to ECG data in the MIT-BIH format.
- inference: Definition of the interpretation algorithms, including the *construe* algorithm and the reasoning modes (*abduce*, *deduce*, *subsume*, *predict* and *advance*) [1].
- knowledge: Definition of the ECG abstraction model, including *observables* and *abstraction patterns*.
- model: General data model of the framework, including the base class for all *observables* and classes to implement *abstraction grammars* as finite automata.
- utils: Miscellaneous utility modules, including signal processing and plotting routines.

Known issues

- On windows and OS-X systems, the *Dynamic Time Warping* utilities included in the construe.utils.signal_processing.dtw package may not work. These sources are from the discontinued mlpy project, and should be compiled using cython with the following commands:

```
$ cd construe/utils/signal_processing/dtw
$ python3 setup.py build_ext --inplace
```

```
Another possible workaround is to install the *mlpy* package and change the `dtw_std` import in the
`construe/knowledge/abstraction_patterns/segmentation/QRS.py` module.
```

- Abductive interpretation of time-series is NP-Hard [1]. This implementation includes several optimizations to make computations feasible, but still the running times are probably longer than you expect if the selected abstraction level is rhythm. Parameter tuning also helps to increase the interpretation speed (usually at the cost of worse-quality results). Also try the -v flag to get feedback and make the wait less painful ;-).

License

INFORMACIÓN

Investigadores
Paulo Félix Lamas
Tomás Teijeiro Campo

Licenza

Como contribuír

DESCARGAR

Repositorio Gitlab

Descargar de Gitlab

Repositorio Github

PUBLICACIÓNS

On the adoption of abductive reasoning for time series interpretation
Artificial Intelligence, 2018

PROXECTOS DE INVESTIGACIÓN

CARE-U: Un contorno de saúde integrador e ubicuo para a autoxestión da enfermidade crónica