Physionet.org WFDB Records: Quick Guide (Python)

Francesco Rosnati

August 9, 2024

Database source:
Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

Contents

1 How to access the Data

For this guide, I have chosen to download the entire MIT-BIH Arrhythmia Database from PhysioNet. This dataset provides access to a comprehensive collection of ECG signals that are widely used in the research and analysis of cardiac arrhythmias.

While the WFDB Python library offers functions to directly access and download specific records from PhysioNet, such as using the `wfdb.io.get_record()` function, I decided to download the full dataset for offline use. This allows for more flexible data processing and ensures that all files are locally available.

The dataset was downloaded from the following URL:

https://physionet.org/content/mitdb/1.0.0/

Python Version Compatibility:

As of the date noted in the document title, the latest Python version was found to have compatibility issues when loading *annotations*. However, Python version **3.8** has been confirmed to work reliably without such issues.

In the sections that follow, I will demonstrate how to load and process these records using the WFDB Python library, focusing on signals and annotations separately. Note that while we will be converting signal data to Pandas DataFrames for ease of manipulation and analysis, the annotations are not ported to DataFrames in this guide.

2 Loading Signals and Annotations

To work with WFDB records in Python, you typically need to load both the signal data and any associated annotations. The WFDB library provides two main functions for this purpose: `rdrecord()` for loading signals and `rdann()` for loading annotations.

Here's how you can load a signal and its annotations:

```python
# Load the signal and annotations
record = wfdb.rdrecord('path/to/record')
annotation = wfdb.rdann('path/to/record', 'atr')
```

This code snippet demonstrates how to load the signal data and annotations from a record. The following subsections provide a detailed explanation of each function.

2.1 Using the `rdrecord()` Function

2.1.1 Overview of `rdrecord()`

The `rdrecord()` function is a key tool in the WFDB Python library, designed to read and assemble data from WFDB records. Each record typically consists of multiple files: a `.dat` file containing raw signal data, a `.hea` file containing metadata, and possibly an `.atr` file with annotations.

Function Overview When you call `rdrecord()`, it reads the signal data from the `.dat` file and interprets it using the metadata in the `.hea` file. This allows the function to convert the raw data into meaningful units, such as millivolts for ECG signals. Additionally, the function can read specific portions of the data, select particular channels, and handle records stored at varying sampling rates. However, `rdrecord()` only deals with the signal data and does not read or associate any annotations from the `.atr` file. The function operates on one record at a time, specified by the path provided to the `record_name` argument.

2.1.2 Arguments of `rdrecord()`

Below is an explanation of each argument that can be passed to the `rdrecord()` function:

```python
def rdrecord(
    record_name,
    sampfrom=0,
    sampto=None,
    channels=None,
    physical=True,
    pn_dir=None,
    m2s=True,
    smooth_frames=True,
    ignore_skew=False,
    return_res=64,
    force_channels=True,
    channel_names=None,
    warn_empty=False,
)
```

- `record_name`: (Required) The full path to the WFDB record, including the base name of the files but without the extension. For example, if your files are named `100.dat` and `100.hea` and are located in `/path/to/records/`, you would use `record_name='/path/to/records/100'`.
- `sampfrom=0`: Specifies the starting sample index from which to begin reading the data. The default value is 0, meaning reading starts from the beginning of the record.
- `sampto=None`: Specifies the ending sample index at which to stop reading the data. If set to `None`, the function reads to the end of the record.
- `channels=None`: A list of channel indices to read. If set to `None`, all channels are read.
- `physical=True`: Determines whether the signal is returned in physical units (e.g., millivolts) or in raw digital samples. If `True`, the signal is returned in physical units; if `False`, it is returned in digital samples.
- `pn_dir=None`: Specifies the PhysioNet directory from which to download the record if it is not available locally. This is useful for accessing datasets hosted on PhysioNet.
- `m2s=True`: Converts multi-frequency signals to a single frequency by resampling to the highest frequency in the record.
- `smooth_frames=True`: Smooths out differences in sample rates between channels by averaging samples. This is relevant for records where different channels are sampled at different rates.
- `ignore_skew=False`: If set to `False`, the function will correct any skew (time offset) between channels. If set to `True`, skew is ignored.
- `return_res=64`: Specifies the bit resolution of the returned signals. The default value is 64 bits, but it can be set to 32 or 16 bits per sample to reduce memory usage.
- `force_channels=True`: Ensures that the returned signal array contains all requested channels, even if some are missing (those missing will be filled with zeros).
- `channel_names=None`: Specifies the names of the channels to be read. If set to `None`, the default channel names from the record are used.

- `warn_empty=False`: If set to `True`, the function will generate a warning when reading a segment of the record that is outside the range of available data, where data would be zero.

This function provides a high degree of flexibility, allowing you to customize how you read and process physiological signals based on your specific analysis needs.

2.2 Using the `rdann()` Function

2.2.1 Overview of `rdann()`

The `rdann()` function in the WFDB Python library is designed to read annotation files (such as `.atr` files) associated with WFDB records. These annotation files typically contain important markers or events within the physiological signal data, such as the locations of heartbeats, arrhythmias, or other significant occurrences.

Function Overview When you call `rdann()`, it reads the annotations from the specified annotation file (e.g., `.atr`) and returns an object that contains the sample indices where the annotations occur and the corresponding labels or symbols. This function allows you to extract specific annotations over a selected range of samples and provides options to handle annotations in various ways, such as shifting sample indices or summarizing labels. The function operates on one annotation file at a time, specified by the path provided to the `record_name` argument and the `extension` of the annotation file.

2.2.2 Arguments of `rdann()`

Below is an explanation of each argument that can be passed to the `rdann()` function:

```python
def rdann(
    record_name,
    extension,
    sampfrom=0,
    sampto=None,
    shift_samps=False,
    pn_dir=None,
    return_label_elements=["symbol"],
    summarize_labels=False,
)
```

- `record_name`: (Required) The full path to the WFDB record, including the base name of the files but without the extension. For example, if your annotation files are named `100.atr` and are located in `/path/to/records/`, you would use `record_name='/path/to/records/100'`.
- `extension`: (Required) The extension of the annotation file you wish to read. Common extensions include `'atr'` for annotations, but other extensions might be used depending on the dataset.
- `sampfrom=0`: Specifies the starting sample index from which to begin reading annotations. The default value is 0, meaning reading starts from the beginning of the record.
- `sampto=None`: Specifies the ending sample index at which to stop reading annotations. If set to `None`, the function reads to the end of the record.
- `shift_samps=False`: If set to `True`, this option will shift the sample indices so that the first annotation starts at sample 0. This can be useful for aligning annotations with signals when working with specific segments of the data.

- `pn_dir=None`: Specifies the PhysioNet directory from which to download the annotation file if it is not available locally. This is useful for accessing datasets hosted on PhysioNet.
- `return_label_elements=["symbol"]`: Specifies which elements of the annotation labels to return. By default, this is set to `"symbol"`, which returns the annotation symbols. Other options might include `"description"` or `"aux_note"` depending on the available information in the annotation file.
- `summarize_labels=False`: If set to `True`, the function will summarize the annotations by counting the occurrences of each label type instead of returning the full list of annotations. This can be useful for quickly understanding the distribution of annotation types within the data.

3 Working with DataFrames

The WFDB library allows you to convert signal data into a Pandas DataFrame, enabling easy manipulation and analysis. This section outlines how to convert a record to a DataFrame and inspect the resulting data.

3.1 Converting Signal Data to a DataFrame

You can convert a WFDB record to a Pandas DataFrame using the `to_dataframe()` method. This method organizes the signal data into a DataFrame, with each column representing a signal channel and each row representing a sample point.

```
import pandas as pd

# Convert the WFDB record to a DataFrame
df = record.to_dataframe()
```

3.2 Inspecting the DataFrame

After converting the record, you can inspect the DataFrame to understand its structure and content. Pandas provides several methods to achieve this, such as `head()` to view the first few rows and `dtypes` to check the data types of each column.

```
# View the first few rows of the DataFrame
print(df.head())

# Check the data types of each column
print(df.dtypes)
```

Note: In this guide, we only convert the signal data into a Pandas DataFrame. The annotations associated with the WFDB records are not ported to the DataFrame.