QPrism

Release 0.4.0

Ramzi Halabi, Zixiong Lin, Rahavi Selvarajan, Jana Kabrit, Abhish

Oct 06, 2022

CONTENTS

The **QPrism** Python package serves as a quality assessment toolbox for data collected using sensors in smartphones and wearables (eg. accelerometer, gyroscope, audio and video). The package leverages digital signal and image processing techniques along with machine learning algorithms to assess the quality of sensor data covering data availability, interpretability, noise contamination and consistency. QPrism is completely data-driven, requiring no a priori data assumptions or application-specific parameter tuning to generate a comprehensive data quality report.

Check out the *Usage* section for further information, including the *Installation* of the project.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Implementation

This page detailed the implementation for DQM metrics we are using in the sensor module as a supplementary material. For the simpler definition of each metrics, please refer to the *Glossary* page.

IRLR

The ratio of the number of records having two or more data points over the total number of records, calculated as: $IRLR = \frac{n(len>1)}{N}$, where $n(len > 1)$ is the number of records having a length of 2 points and above, and N is the total number of records. IRLR can be used to investigate a commonly observed quality issue that arises when the data collection app creates and uploads a record file that includes no data points or a single uninterpretable point.

SCR

The ratio of available channels per record by the typical number of channels for each sensor, calculated as: $SCR = \frac{SCC}{ESCC}$, $ESCC = mode(SCC)$, where SCC is the sensor channel count, $ESCC$ is the expected sensor channel count calculated as the most frequent channel count per sensor. SCR allows the detection of missing channels (e.g., acceleration across the z-axis) in case of sensor defect or partial data upload.

MDR

The ratio of missing data points over the total number of record points for all sensor types. The MDR metric measures the level of discontinuity manifested through skipped data points due to undersampling, calculated as: $MDR = \frac{MPC}{TPC}$, where MPC is the missing point count, and TPC is the total point count. $MPC+ = \frac{t(n+1)-t(n)}{\hat{t_s}} - 1, if : t(n+1) - t(n) \geq 2\hat{t_s}$, and $\hat{t_s} = mode(t_s)$. Where $t(n)$ is the timestamp of point n, t_s is the sampling interval, and

VDR The ratio of non-NaN data points over the total number of data points. The VDR metric measures the level of invalid data points in the records, calculated as: $VDR = \frac{VPC}{TPC}$, where VPC is the valid point count, and TPC is the total point count. $VPC+ = 1$, if there is a non-NaN data point in the record. VDR allows the detection of NaN data points.

SNR A measure of the amplitude or power ratio of the signal component, i.e., desired data by the noise component. SNR describes the level of data variability around the mean absolute value, approximating the proportion of noise contamination with respect to the desired data of interest, calculated as: $SNR = 20log\frac{S}{N} = 20log\frac{MAV(x)}{\sigma_x}$, where S is the desired data (signal) approximated as the MAV (mean absolute value) of data record x, and N is the undesired data (noise) approximated as the standard deviation of data record x. This metric gives insight into the noise level rather than its type, providing feedback on the required level of data denoising.

APD

The ratio of anomalous points by the total number of recorded points, calculated as: $APD = \frac{APC}{TPC}$, $APC+ = 1, if : DS(x[n]) > \overline{DS(x)} + 3\sigma_{DS(x)}$, where APC is the anomalous point count, and TPC is the total point count. The APC is incremented by 1 whenever the decision score (DS) of a data sample $x[n]$ exceeds

the set threshold being a z-score of 3. The APD metric allows for detecting and counting outliers in sensor data on a point-by-point basis, resulting in a density estimate of anomalies. It is computed using an unsupervised machine learning technique termed feature bagging method followed by outlier score thresholding to count anomalous points.

RLC The inverse of the coefficient of variation of record length variability across data records, such that record length is the number of points per record. RLC is a measure of uniformity between the lengths of different data records across sensors, calculated as: $RLC = 2(1 - sig(\frac{\sigma_{RL}}{\overline{RL}}))$ where $sig(x)$ is the sigmoid function, σ_{RL} is the standard deviation of record lengths, and $\overline{RL}$ is the mean record length. The higher RLC is, the less the burden of data segmentation and preprocessing is before the analysis.

SRC The ratio of mode sampling rate by the total number of sampling rate, calculated as: $SRC = \frac{MSC}{SC}$, $MSC+ = 1$, if there is a sampling rate equals to mode samplint rate, and SC is the total sampling rate count. SRC provides a measure of the stability of data discretization, hence the level of sampling continuity and uniformity. The higher SRC is, the less the risk of information loss is.

VRC The inverse of the coefficient of variation of record data value min-max range variability across sensors with the min-max range describing the difference between the maximum and minimum value of a record. VRC is the degree of stability of the data value range across sensors, calculated as: $VRC = 2(1 - sig(\frac{\sigma_{VR}}{\overline{VR}}))$, where $sig(x)$ is the sigmoid function, σ_{VR} is the standard deviation of value ranges, and $\overline{VR}$ is the mean value range. While value range variability is expected in real-world settings, high data consistency indicates low range variability, hence higher reproducibility and sensor output stability.

1.2 Usage

1.2.1 Installation

To use QPrism, the installation can be done with pip. Since pip does not resolve the dependencies' version efficiently, please install QPrism with the following steps.

```
$ python3 -m pip install --upgrade pip
```

```
$ pip install -r https://raw.githubusercontent.com/aid4mh/QPrism/main/requirements.txt
```

```
$ pip install --no-deps QPrism
```

1.2.2 Examples

We have provided detailed demo notebooks for each submodule.

Sensor

Demo for QA a single record can be accessed here.

Demo for QA multiple records in a directory can be accessed here.

We have also provided demos that can validate the integrity of our DQM metrics, those demos can be accessed in the test folder at here, any notebook with the name <DQM_name>_testing.ipynb is the validation demo.

We have a code template that the user can use with minor modifications to the input data.

Note that the input dataframe should only contains numerical values. If there are non-numerical values, either a timestamp or a text entry, please refer here for an example of solution.

Video

Demo for QA video(s) can be accessed here.

Audio

Demo for QA audio(s) can be accessed here.

1.3 API

Welcome to the API page for QPrism. This page is an overall class and function reference, please refer to the individual class and function page for further details. For reference on concepts repeated across the API, see *Glossary*.

1.3.1 Video

Video_DQM

This is the class for computing the video DQM.

class QPrism.Video.DQM.**Video_DQM**

Bases: object

artifacts_ratio(*path: str*)

This function gets how much of the video that contains video artifacts

Returns the ratio of the video that contains artifacts (motion blur, too grainy, static) if the input path is a file. Returns a dict mathing the input videos to their artifacts percentage if the input path is a folder.

path
[str] path to a folder or a file

bit_rate(*path: str*)

Get the bit rate of the video

Returns the bit rate for input video or each video in the input folder.

path
[str] path to a folder or a file

`creation_time`(*path: str*)

This function gets the creation time of a video.

Returns the creation time of the video if the input path is a file. Returns a dict matching the input videos to their creation time if the input path is a folder.

path
[str] path to a folder or a file

`duration`(*path: str*)

This function gives the length for both a folder of audios or a single audio file.

Returns the length of the input audio if the input path is a file. Returns a dict matching the input audio files to their length if the input path is a folder.

path
[str] A path to a folder or a file.

`format`(*path: str*)

Get the format of the video This functions calls the video_format function and gives the format for both a folder of videos or a single video file.

path : path to a folder or a file

dict (incase of folder) str (incase of file)

dict : {'video_file1' : '.mp4', ... 'video_filen' : '.mp4'} str : '.mp4'

`framerate`(*path: str*)

This function get the framerate of the video.

Returns the frame rate of the video if the input path is a file. Returns a dict matching the input videos to their frame rate if the input path is a folder.

path
[str] path to a folder or a file

`illumination`(*path: str*)

This function gives the brightness of the video.

Returns the brightness of the video if the input path is a file. Returns a dict matching the input videos to their frame rate if the input path is a folder.

path
[str] path to a folder or a file

`object_detection`(*path: str*, *modelname: str*)

This function detect all the objects present in the video.

Returns the detected objects in the video if the input path is a file. Returns a dict matching the input videos to their detected objects.

path
[str] path to a folder or a file

`resolution`(*path: str*)

This function returns the resolution of a video or the videos in given folder

path
[str] path to a folder or a file

save_csv(*path: str*, *output_path: str*, *modelname='yolov5s'*)

Creates a csv file consists of all the video metrics.

Saves a csv file to the input path with all video quality metrics

path
[str] path to single video file or folder

output_path
[str] path to the output csv file

modelname
[str] yolov5 model name for object detection

1.3.2 Audio

Audio_DQM

This is the class for computing the audio DQM.

class QPrism.Audio.DQM.**Audio_DQM**

Bases: object

RMS(*path: str*)

Compute the RMS value of the audio. This can be used to distinguish audios that are louder from each other.

Returns the root mean squared value of the audio file(s). Returns a dict matching the input audio files to their RMS value if the input path is a folder.

path
[str] A path to a folder or a file.

observation_duration(*path: str*)

Get the length of the audio

This functions calls the audio_length function and gives the length for both a folder of audios or a single audio file.

Returns the length of the input audio if the input path is a file. Returns a dict matching the input audio files to their length if the input path is a folder.

path
[str] A path to a folder or a file.

sampling_rate(*path: str*)

Get the sample rate of the audio

This functions calls the sample_rate function and gives the bit rate for both a folder of audios or a single audio file.

Returns the sample rate of the input audio if the input path is a file. Returns a dict matching the input audio files to their sample rate if the input path is a folder.

path
[str] A path to a folder or a file.

save_csv(*path: str*, *output_path: str*)

Creates a csv file consists of all the audio metrics.

Saves the csv file to output_path.

path
[str] A path to single audio file or folder

output_path
[str] path to the output csv file

voice_classification(*path: str*)

Get a list of all the sounds in an audio

This function identifies all the prominent sounds inside the audio

Returns a list of prominent sounds in the input audio if the input path is a file. Returns a dict matching the input audio files to their list of prominent sounds if the input path is a folder.

path
[str] A path to a folder or a file.

1.3.3 Sensor

DQM_single_record

This is the class for computing a single record DQM.

class QPrism.Sensor.DQM.**DQM_single_record**

Bases: object

A class represents the data quality matrix.

compute_DQM()

Compute the DQM for input record under current metrics configuration. Must call set_input_path before calling this method.

get_APD()

Return the APD score for given input data as a str. APD must be included in the DQM class.

get_DQM()

Return the computed DQM as a list. compute_DQM must be called before this method.

get_IRLR()

Return the IRLR score for given input data as str.

get_MDR()

Return the MDR score for given input data as a str. MDR must be included in the DQM class.

get_SNR()

Return the SNR score for given input data as a str. SNR must be included in the DQM class.

get_SRC()

Return the SRC score for given input data as a str. SRC must be included in the DQM class.

get_VDR()

Return the VDR score for given input data as a str. VDR must be included in the DQM class.

`get_anomaly_index()`

Return the index (row number) of the anamoly points for the input record. APD must be included in the DQM class.

`get_fields()`

Return a list represents the current included metrics in DQM. compute_DQM must be called before this method.

`save_to_file(path: str)`

Save the computed DQM as a csv file to the given output path. Must call compute_DQM before calling this.

path
[str] Path to the output file

`set_APD(included: bool)`

Set whether APD is included as a metric

included
[bool] A bool value to indicate whether APD is included in the result DQM

`set_MDR(included: bool)`

Set whether MDR is included as a metric

included
[bool] A bool value to indicate whether MDR is included in the result DQM

`set_SNR(included: bool)`

Set whether SNR is included as a metric

included
[bool] A bool value to indicate whether SNR is included in the result DQM

`set_SRC(included: bool)`

Set whether SRC is included as a metric

included
[bool] A bool value to indicate whether SRC is included in the result DQM

`set_VDR(included: bool)`

Set whether VDR is included as a metric

included
[bool] A bool value to indicate whether VDR is included in the result DQM

`set_input_data(df: DataFrame)`

Set the path of input record file.

df
[pd.DataFrame] A dataframe represents a single record, all entries in the dataframe should be able to convert to float in order to compute the DQM.

DQM_multiple_record

This is the class for computing the DQM for multiple records. It supports to compute an overall DQM for the whole dataset, and compute a set of individual DQMs for each record in the dataset.

class QPrism.Sensor.DQM.**DQM_multiple_record**

Bases: object

A class represents the data quality matrix for multiple files in one directory.

compute_avg_DQM()

Compute a DQM for the input records under current metrics configuration. The DQM averaged all the metrics that are supported in DQM_single_file. In addition to above metrics, it also computes all the metrics only support DQM_multi_file. The computed DQM consists only one row. If you wish to get the DQM for each individual record, call the compute_individual_DQM method instead. Must call set_input_path before calling this method.

compute_individual_DQM()

Compute a DQM for the input records under current metrics configuration. This computes the DQM for each individual record, and store the DQMs as a list. If you wish to get the averaged DQM with the metrics only support multiple records, call the compute_avg_DQM method instead. Must call set_input_path before calling this method.

get_APD()

Return the APD score for given input data as a str. APD must be included in the DQM class.

get_IRLR()

Return the IRLR score for given input data as a str.

get_MDR()

Return the MDR score for given input data as a str. MDR must be included in the DQM class.

get_RLC()

Return the RLC score for given input data as a str. RLC must be included in the DQM class.

get_SCR()

Return the SCR score for given input data as a str. SCR must be included in the DQM class.

get_SNR()

Return the SNR score for given input data as a str. SNR must be included in the DQM class.

get_SRC()

Return the SRC score for given input data as a str. SRC must be included in the DQM class.

get_VDR()

Return the VDR score for given input data as a str. VDR must be included in the DQM class.

get_VRC()

Return the VRC score for given input data as a str. VRC must be included in the DQM class.

get_avg_DQM()

Return the computed DQM as a list. compute_DQM must be called before this method.

get_avg_fields()

Return a list represents the current included metrics in the multi-file DQM. compute_avg_DQM must be called before this method.

get_individual_DQM()

Return the computed individual DQM as a list. Each element in the list represents a single record. compute_individual DQM must be called before this method.

get_individual_fields()

Return a list represents the current included metrics in the individual file DQM. compute_individual_DQM must be called before this method.

save_avg_DQM_to_file(*path: str*)

Save the computed DQM as a csv file to the given output path. Must call compute_DQM before calling this.

path
[str] Path to the output file

save_individual_DQM_to_file(*path: str*)

Save the computed DQM as a csv file to the given output path. Must call compute_DQM before calling this.

path
[str] Path to the output file

set_APD(*included: bool*)

Set whether APD is included as a metric

included
[bool] A bool value to indicate whether APD is included in the result DQM

set_MDR(*included: bool*)

Set whether MDR is included as a metric

included
[bool] A bool value to indicate whether MDR is included in the result DQM

set_RLC(*included: bool*)

Set whether RLC is included as a metric

included
[bool] A bool value to indicate whether RLC is included in the result DQM

set_SCR(*included: bool*)

Set whether SCR is included as a metric

included
[bool] A bool value to indicate whether SCR is included in the result DQM

set_SNR(*included: bool*)

Set whether SNR is included as a metric

included
[bool] A bool value to indicate whether SNR is included in the result DQM

set_SRC(*included: bool*)

Set whether SRC is included as a metric

included
[bool] A bool value to indicate whether SRC is included in the result DQM

set_VDR(*included: bool*)

Set whether VDR is included as a metric

included
[bool] A bool value to indicate whether VDR is included in the result DQM

set_VRC(*included: bool*)

Set whether VRC is included as a metric

included
[bool] A bool value to indicate whether VRC is included in the result DQM

set_input_data(*df_list*)

Set the input data list of input file.

df_list
[list of pd.DataFrame] A list of dataframe, each dataframe represents a single record, all entries in the dataframe should be able to convert to float in order to compute the DQM.

helper_function

These are the available data processing helper functions

QPrism.Sensor.data_processing.**timestamp_to_unix**(*df*)

Convert the timestamp in the dataframe to unix format.

df
[pd.DataFrame] A dataframe represents a single record, with the timestamp in the pandas datetime format.

1.4 Glossary

1.4.1 Sensor Module

DQM
Data Quality Metric. The metric generated by the pipeline, including all the quality assessment metrics that are related to user's research.

Correctness
One of the three aspects that the pipeline evaluates. It measures the overall level of the representational and value integrity of the input data via APD and SNR.

Completeness
One of the three aspects that the pipeline evaluates. It measures the overall level of availability and validity of the input data via IRLR, MDR, SCR, and VDR.

Consistency
One of the three aspects that the pipeline evaluates. It measures the overall uniformity of the input data via RLC, SRC, and VRC.

APD
Anomalous Point Density. The ratio of the number of outliers/anomalous data samples over the total number of samples.

IRLR
Interpretable Record Length Ratio. The ratio of the number of interpretable records over the total number of

records. A record is interpretable if it satisfies the following: a. Has more than one row of data. b. Has a non-decreasing timestamp. c. Has non-zero standard deviation on each data channel.

MDR
Missing Data Ratio. The ratio of the number of missing data samples over the total number of samples. The MDR metric measures the level of discontinuity manifested through skipped data points due to irregular sampling.

RLC
Record Length Consistency. The uniformity of data record length across multiple records.

SCR
Sensor Channel Ratio. The ratio of the number of records that has full channels over the total number of records. A record is defined to have full channels if the number of its channels is the same as the mode of the number of channels of given input.

SNR
Signal-to-noise Ratio. The ratio of desired signal amplitude over the noise amplitude. This metric gives insight into the noise level rather than its type, providing feedback on the required level of data denoising.

SRC
Sampling Rate Consistency. The uniformity of the data sampling rate within a record.

VRC
Value Range Consistency. The uniformity of the value range across multiple records.

VDR
Valid Data Ratio. The ratio of the number of non-NaN data points over the total number of data points.

1.4.2 Audio Module

Observation Duration
The length of the audio file(s), in seconds.

Sampling Rate
The number of samples per second of the audio file(s), in Hertz.

Voice Classification
The detected sounds in the audio file(s) using YAMNet, a deep neural network.

RMS
The root mean squared value of the audio file(s). This can be used to distinguish audios that are louder from each other.

1.4.3 Video Module

Duration
The length of the video file(s), in seconds.

Resolution
The number of pixels contained in each frame of the video file(s), in pixels.

Format
The extension of the video file(s).

Bit Rate
The number of bits of the video file(s) per second, in bits/second.

Frame rate
The number of frames of the video file(s) per second , in frames/second.

Illumination
The average pixel magnitude of the RGB channels of the video file(s).

Creation Date
The creation date of the media instead of the actual file. If this part is missing in the metadata, the module will return N/A.

Artifact Ratio
The ratio of distorted frames in the video file(s) by the total number of frames, in fraction.

CHAPTER

TWO

ACKNOWLEDGEMENTS

Supported by CAMH / Krembil Center of Neuroinformatics.

The authors also like to acknowledge Aditi Surendra for designing the module function illustration.

CHAPTER

THREE

CONTRIBUTION

We welcome and encourage project contributions! Please see the here for details.

CHAPTER

FOUR

LICENSE

INDEX

O

R

S

T

V