DVHA MLC Analyzer

Release 0.2.3

Dan Cutright

Jan 27, 2021

CONTENTS:

CHAPTER

ONE

DVHA MLC ANALYZER

Batch analyze DICOM-RT Plan files to calculate complexity scores

DVH Analytics (DVHA) is a software application for building a local database of radiation oncology treatment planning data. It imports data from DICOM-RT files (i.e., plan, dose, and structure), creates a SQL database, provides customizable plots, and provides tools for generating linear, multi-variable, and machine learning regressions.

DVHA-MLCA is a stand-alone command-line script to batch analyze DICOM-RT Plans using the MLC Analyzer code from DVHA.

Complexity score based on: Younge KC, Matuszak MM, Moran JM, McShan DL, Fraass BA, Roberts DA. Penalization of aperture complexity in inversely planned volumetric modulated arc therapy. Med Phys. 2012;39(11):7160–70.

1.1 Installation

To install via pip:

```
$ pip install dvha-mlca
```

If you've installed via pip or setup.py, launch from your terminal with:

```
$ mlca <init-scanning-directory>
```

If you've cloned the project, but did not run the setup.py installer, launch DVHA-MLCA with:

```
$ python mlca/main.py <init-scanning-directory>
```

1.2 Command line usage

```
usage: mlca [-h] [-of OUTPUT_FILE] [-xw COMPLEXITY_WEIGHT_X]
            [-yw COMPLEXITY_WEIGHT_Y] [-xs MAX_FIELD_SIZE_X]
            [-ys MAX_FIELD_SIZE_Y] [-ver] [-v] [-n PROCESSES]
            [init_dir]

Command line DVHA MLC Analyzer

positional arguments:
  init_dir              Directory containing DICOM-RT Plan files
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -of OUTPUT_FILE, --output-file OUTPUT_FILE
                        Output will be saved as
                        dvha_mlca_<version>_results_<time-stamp>.csv by
                        default.
  -xw COMPLEXITY_WEIGHT_X, --x-weight COMPLEXITY_WEIGHT_X
                        Complexity coefficient for x-dimension: default = 1.0
  -yw COMPLEXITY_WEIGHT_Y, --y-weight COMPLEXITY_WEIGHT_Y
                        Complexity coefficient for y-dimension: default = 1.0
  -xs MAX_FIELD_SIZE_X, --x-max-field-size MAX_FIELD_SIZE_X
                        Maximum field size in the x-dimension: default = 400.0
                        (mm)
  -ys MAX_FIELD_SIZE_Y, --y-max-field-size MAX_FIELD_SIZE_Y
                        Maximum field size in the y-dimension: default = 400.0
                        (mm)
  -ver, --version       Print the DVHA-MLCA version
  -v, --verbose         Print final results and plan summaries as they are
                        analyzed
  -n PROCESSES, --processes PROCESSES
                        Enable multiprocessing, set number of parallel
                        processes
```

For example:

```
$ mlca "C:\PatientDicom" -n 8
Directory: C:\PatientDicom
Begin file tree scan ...
File tree scan complete
Searching for DICOM-RT Plan files ...
     100%|| 9087/9087 [00:59<00:00, 153.52it/s]
1650 DICOM-RT Plan file(s) found
Analyzing 1650 file(s) ...
      10%|                           | 169/1650 [02:02<13:35,  1.82it/s]
```

1.3 Dependencies

- Python >3.5
- Pydicom
- NumPy
- Shapely
- tqdm

1.4 Support

If you like DVHA-MLCA and would like to support our mission, all we ask is that you cite us if we helped your publication, or help the DVHA community by submitting bugs, issues, feature requests, or solutions on the issues page.

1.5 Cite

DOI: https://doi.org/10.1002/acm2.12401 Cutright D, Gopalakrishnan M, Roy A, Panchal A, and Mittal BB. "DVH Analytics: A DVH database for clinicians and researchers." Journal of Applied Clinical Medical Physics 19.5 (2018): 413-427.

CHAPTER

TWO

USAGE

2.1 Plan Summary

A simple example is below. Check out the module references for details about every method and property available.

```
>>> from mlca.mlc_analyzer import Plan
>>> plan = Plan('rtplan.dcm')
>>> plan
```

Output will look like:

```
Patient Name:         ANON11264
Patient MRN:          ANON11264
Study Instance UID:   2.16.840.1.114362.1.6.7.4.17517.7693757184.462857971.1073.8123
TPS:                  ADAC Pinnacle3
Plan name:            ANON
# of Fx Group(s):     3
Plan MUs:             776.1, 659.2, 512.3
Beam Count(s):        9, 9, 8
Control Point(s):     140, 120, 100
Complexity Score(s): 1.274, 1.290, 1.127
```

2.2 Beam Data

You can access specific beam information. Some examples are shown below.

```
>>> beam = plan.fx_group[0].beam[0]
>>> beam.area
    [16172.0, 16172.0, 13398.0, 13398.0, 428.0, 428.0, 787.0, 787.0, 582.0,
    582.0, 1728.0, 1728.0, 3479.0, 3479.0, 502.0, 502.0, 8587.0, 8587.0]
>>> beam.gantry_angle
    [200.0]
>>> beam.meter_set
    90.199996948242
>>> beam.younge_complexity_scores
    array([0.0081077 , 0.        , 0.00878864, 0.        , 0.01600782,
           0.        , 0.02859258, 0.        , 0.02102578, 0.        ,
           0.01606121, 0.        , 0.01505108, 0.        , 0.01389648,
           0.        , 0.0075811 , 0.        ])
```

CHAPTER

THREE

DVHA-STATS

3.1 MLC Analyzer

Tools for analyzing beam and control point information from DICOM files Hierarchy of classes: Plan -> FxGroup -> Beam -> ControlPoint

class mlca.mlc_analyzer.**Beam**(*beam_dataset*, *meter_set*, *ignore_zero_mu_cp=False*, ***kwargs*)
Bases: object

Collect beam information from a beam in a beam sequence of a pydicom RT Plan dataset. Automatically parses control point data with ControlPoint class

Parameters

- **beam_dataset** (*Dataset*) – element of a BeamSequence (300A,00B0)
- **meter_set** (*int, float*) – the monitor units for beam_dataset
- **ignore_zero_mu_cp** (*bool*) – If True, skip over zero MU control points (e.g., as in Step-N-Shoot beams)

property aperture
Get aperture shapely object for every control point

Returns ControlPoint.aperture for each control point

Return type list

property area
Get aperture area for every control point

Returns ControlPoint.area for each control point

Return type list

property collimator_angle
Collimator angles for each control point

Returns A list of float values, defining collimator angles for each control point

Return type list

property couch_angle
Couch angles for each control point

Returns A list of float values, defining couch angles for each control point

Return type list

property cp_count
Get the number of control points in this beam

Returns Length of ControlPointSequence (300A,0111)

Return type int

property cp_mu
MU for each control point

Returns Numpy array of control point MUs

Return type np.ndarray

property cp_seq
Get the control points

Returns ControlPointSequence (300A,0111)

Return type Dataset

property cum_mu
Cumulative monitor units for each control point

Returns A list of float values representing the cumulative MU

Return type list

property gantry_angle
Gantry angles for each control point

Returns A list of float values, defining gantry angles for each control point

Return type list

property jaws
Jaw positions for each control point

Returns Each element is a ControlPoint.jaws dict

Return type list

property leaf_boundaries
Get the leaf boundaries

Returns LeafPositionBoundaries (300A,00BE)

Return type DataSet

property mlc_borders
Get the MLC border for each control point

Returns A list of `ControlPoint.mlc_borders` describing the boundaries of each leaf

Return type list

property name
Get the Beam name

Returns In order of priority, BeamDescription (300A,00C3), BeamName (300A,00C2), or "Unknown"

Return type str

property perimeter
Get aperture perimeter for every control point

Returns ControlPoint.perimeter for each control point

Return type list

property perimeter_x
Get x-component of aperture perimeter for every control point

Returns ControlPoint.perimeter_x for each control point

Return type list

property perimeter_y
Get y-component of aperture perimeter for every control point

Returns ControlPoint.perimeter_y for each control point

Return type list

property younge_complexity_scores
Complexity score based on Younge et al

Returns Younge complexity scores for each control point

Return type np.ndarray

class mlca.mlc_analyzer.**ControlPoint**(*cp_elem*, *leaf_boundaries*, ***kwargs*)
Bases: object

Collect control point information from a ControlPointSequence in a beam dataset of a pydicom RT Plan dataset

Parameters

- **cp_elem** (*DataElement*) – element of a ControlPointSequence (300A,0111)
- **leaf_boundaries** (*Dataset*) – LeafPositionBoundaries (300A,00BE)

property aperture
This function will return the outline of MLCs within jaws

Returns a shapely object of the complete MLC aperture as one shape (including MLC overlap)

Return type Polygon

property cum_mu
Cumulative MU for this ControlPoint

Returns CumulativeMetersetWeight (300A,0134)

Return type float

property jaws
Get the jaw positions of a control point

Returns jaw positions (or max field size in lieu of a jaw). Keys are 'x_min', 'x_max', 'y_min', 'y_max'

Return type dict

property leaf_type
Get the MLC orientation

Returns Returns 'mlcx', 'mlcy' or None

Return type str, None

property mlc
Get the MLC orientation

Returns Returns 'mlcx', 'mlcy' or None

Return type str, None

property mlc_borders
This function returns the boundaries of each MLC leaf for purposes of displaying a beam's eye view using bokeh's quad() glyph

Returns the boundaries of each leaf within the control point with keys of 'top', 'bottom', 'left', 'right

Return type dict

property perimeter
Perimeter of the aperture

Returns Aperture perimeter

Return type float

property perimeter_x
x-component of the aperture perimeter

Returns x-component of the Aperture perimeter

Return type float

property perimeter_y
y-component of the aperture perimeter

Returns y-component of the Aperture perimeter

Return type float

class mlca.mlc_analyzer.**FxGroup**(*fx_grp_seq*, *plan_beam_sequences*, ***kwargs*)
Bases: object

Collect fraction group information from fraction group and beam sequences of a pydicom RT Plan dataset. Automatically parses beam data with Beam class

Parameters

- **fx_grp_seq** (*Dataset*) – element of FractionGroupSequence (300A,0070)
- **plan_beam_sequences** (*BeamSequence*) – BeamSequence (300A,00B0)

property beam_count
Get the number of beams

Returns Length of FxGroup.beams

Return type int

property beam_mu
Get the monitor units for each beam

Returns Beam.meter_set for each beam

Return type list

property beam_names
Get the beam names

Returns Beam.name for each beam

Return type list

property cp_counts
Get the number of control points for all beams

Returns Beam.cp_count for each beam

Return type list

property fx_mu
Get the number of MU for for this fraction

Returns The sum of FxGroup.beam_mu

Return type float

update_missing_jaws()
In plans with static jaws, jaw positions may not be found in each control point

property younge_complexity_score
Get the Younge complexity score this fraction

Returns The sum of Beam.younge_complexity_scores for all beams

Return type float

class mlca.mlc_analyzer.**Plan**(*rt_plan*, ***kwargs*)
Bases: object

Collect plan information from an RT Plan DICOM file. Automatically parses fraction data with FxGroup class

Parameters **rt_plan** (*str, Dataset*) – file path of a DICOM RT Plan file or a pydicom Dataset

property patient_id
Get the patient id

Returns PatientID (0010,0020)

Return type str

property patient_name
Get the patient name

Returns PatientName (0010,0010)

Return type str

property plan_name
Get the plan name

Returns RTPlanLabel (300A,0002)

Return type str

property sop_instance_uid
Get the patient study instance UID

Returns SOPInstanceUID (0008,0018)

Return type str

property study_instance_uid
Get the patient study instance UID

Returns StudyInstanceUID (0020,000D)

Return type str

property tps
Get the treatment planning system

Returns Manufacturer (0008,0070) and ManufacturerModelName (0008,1090)

Return type str

property younge_complexity_scores
Get the Younge complexity scores for each FxGroup

Returns FxGroup.younge_complexity_score for each fraction group

Return type list

class mlca.mlc_analyzer.**PlanSet**(*file_paths*, *verbose=False*, *processes=1*, ***kwargs*)
Bases: object

Parse DICOM-RT Plan files, analyze MLCs

Parameters

- **file_paths** (*list*) – A list of file paths to DICOM-RT Plan files
- **verbose** (*bool, optional*) – Set to true to print detailed information (ignored if multiprocessing enabled)
- **processes** (*int*) – Number of parallel processes allowed

mlca.mlc_analyzer.**get_options**(*over_rides*)
Get MLC Analyzer options

Parameters **over_rides** (*dict*) – Over rides, keys may be 'max_field_size_x', 'max_field_size_y', 'complexity_weight_x', or 'complexity_weight_y'

Returns Options for field size and complexity weights. Default values are 400 and 1.

Return type dict

3.2 Utilities

Utilities for DVHA-MLCA

mlca.utilities.**create_cmd_parser**()
Get an argument parser for mlca.main

Returns argument parser

Return type argparse.ArgumentParser

mlca.utilities.**flatten_list_of_lists**(*some_list*, *remove_duplicates=False*, *sort=False*)
Convert a list of lists into one list of all values

Parameters

- **some_list** (*list*) – a list such that each element is a list
- **remove_duplicates** (*bool, optional*) – if True, return a unique list, otherwise keep duplicated values
- **sort** (*bool, optional*) – if True, sort the list

Returns A new list containing all values in some_list

Return type list

mlca.utilities.**get_default_output_filename**()
Get the default output file name for mlca.main.process

Returns dvha_mlca_<version>_results_<timestamp>.csv

Return type str

mlca.utilities.**get_dicom_files**(*file_paths*, *modality=None*, *verbose=False*, *processes=1*)
Find all DICOM-RT Plan files in a list of file paths

Parameters

- **file_paths** (*list*) – A list of file paths
- **modality** (*str, optional*) – Specify Modality (0008,0060)
- **verbose** (*bool, optional*) – Print results to terminal
- **processes** (*int*) – Number of processes for multiprocessing.

Returns Absolute file paths to DICOM-RT Plans

Return type list

mlca.utilities.**get_file_paths**(*init_dir*)
Find all files in a directory and sub-directories

Parameters **init_dir** (*str*) – Top-level directory to search for files

Returns Absolute file paths

Return type list

mlca.utilities.**get_xy_path_lengths**(*shapely_object*)
Get the x and y path lengths of a Shapely object

Parameters **shapely_object** (*GeometryCollection, MultiPolygon, Polygon*) – A shapely polygon-like object

Returns Perimeter lengths in the x and y directions

Return type list

mlca.utilities.**is_file_dicom**(*file_path*, *modality=None*, *verbose=False*)

Parameters

- **file_path** (*str*) – File path to potential DICOM file
- **modality** (*str, optional*) – Return False if file is not this Modality (0008,0060)
- **verbose** (*bool, optional*) – Print results to terminal

Returns True if file_path points to a DICOM file, will return False if SOPClassUID (0008,0016) is not found

Return type bool

mlca.utilities.**run_multiprocessing**(*worker*, *queue*, *processes*)
Parallel processing

Parameters

- **worker** (*callable*) – single parameter function to be called on each item in queue
- **queue** (*iterable*) – A list of arguments for worker
- **processes** (*int*) – Number of processes for multiprocessing.Pool

Returns List of returns from worker

Return type list

mlca.utilities.**write_csv**(*file_path*, *rows*, *mode='w'*, *newline=''*)
Create csv.writer, call writerows(rows)

file_path [str] path to file

rows [list, iterable] Items to be written to file_pointer (input for csv.writer.writerows)

mode [str] optional string that specifies the mode in which the file is opened

newline [str] controls how universal newlines mode works. It can be None, ‘’, ‘
‘, ‘ ‘, and ‘ ‘

CHAPTER

FOUR

CREDITS

4.1 Development Lead

- Dan Cutright

CHAPTER

FIVE

HISTORY

5.1 v0.2.3 (2021.01.27)

- Use csv standard library for CSV writing

5.2 v0.2.2 (2021.01.16)

- Multiprocessing support

5.3 v0.2.1 (2021.01.15)

- Fixed issue preventing main run from command line
- Support for large scale analysis, don't store full pydicom datasets
- Added SOPInstanceUID to summary
- Keyboard Interrupt (CTRL+C) during analysis (after file collection) will write the partial results

5.4 v0.2 (2020.12.23)

- Add verbose mode
- dvha-mlca.readthedocs.io
- unit testing

5.5 v0.1rc1 (2020.06.15)

- Initial release

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

R

S

T

U

W

Y