GenTex

Release 0.1.2

GenTex developers

Oct 04, 2019

CONTENTS:

CHAPTER

ONE

INTRODUCTION

GenTex stands for General Texture analysis.

This package provides a suite of routines that combines standard texture analysis methods based on GLCM and entropy/statistical complexity analysis methods.

CHAPTER

TWO

WHAT IS THIS PACKAGE FOR?

GenTex provides a number of the standard algorithms required for generating complexity/texture measure estimates from multimodal imaging data. These include:

1. Generation of multidimensional feature spaces from multimodal 'image' data (i.e. multiple 'co-registered' 1,2,3, or 4 dimensional data sets, e.g. multiple 'co-registered' time series, multimodal image data, space/time data..) via the use of a set of image templates, including:

- single voxels
- linear sequences in cardinal directions (ref.)
- notches in cardinal directions (ref.)
- light cones in cardinal directions and 45 degree angles (ref.)

2. Clustering methods for generating discrete ('grey') levels from the constructed feature space (the levels are then typically mapped to the original image space at the anchor points of the templates)

3. Building co-occurrence matrices from a discrete level 'image' or a pair of discrete level 'images', where the discrete level 'images' are typically generated via feature space clustering of the original multimodal data sets (time series, images, space/time data. . .)

4. Estimation of various complexity/texture measures from the co-occurrence matrices. (Haralick measures and epsilon machine related quantities) such as:

- CM Entropy
- EM Entropy
- Statistical Complexity
- Energy Uniformity
- Maximum Probability
- Contrast
- Inverse Difference Moment
- Correlation
- Probability of Run Length
- Epsilon Machine Run Length
- Run Length Asymmetry
- Homogeneity
- Cluster Tendency
- Multifractal Spectrum Energy Range

- Multifractal Spectrum Entropy Range

CHAPTER

THREE

QUICKSTART

3.1 Installation

```
pip install gentex
```

3.2 Getting started

Here is a dummy example:

First, we create some dummy ndarray (but could be any nd dataset, from an image for example) and a mask.

```
import numpy as np
import gentex

C = np.random.randint(3, size=[5, 5, 5])
maskC = np.ones([5, 5, 5])
```

Then, we compute the GLCM from this array along 2 different offsets

```
offset3 = [[1, 1, 1], [-1, -1, -1]]
cm = gentex.comat.comat_mult(C, maskC, offset3, levels=3)
```

GenTex supports many different type of offsets (rect, conic, angle, distance, etc.). Refer to `gentex.template` for the one available.

Finally, we get a sample of possible statistics extracted from the GLCM

```
texm = ['CM Entropy',
        'EM Entropy',
        'Statistical Complexity',
        'Energy Uniformity',
        'Maximum Probability']
mytex = gentex.texmeas.Texmeas(cm)
for meas in texm:
    mytex.calc_measure(meas)
    print(f'{meas} = {mytex.val}')

CM Entropy = 3.129436250609541
EM Entropy = 1.5849625007211563
Statistical Complexity = 1.584962500721156
Energy Uniformity = 0.117431640625
Maximum Probability = 0.1484375
```

Refer to `gentex.texmeas` for the measures available.

CHAPTER

FOUR

GENTEX PACKAGE

Table of Contents

4.1 gentex.comat

The gentex or general texture analysis package provides a suite of routines that combine standard texture analysis methods and entropy/statistical complexity analysis methods to provide a number of the standard algorithms required for generating complexity/texture measure estimates from multimodal imaging data. These include:

1. Generation of multidimensional feature spaces from multimodal 'image' data (i.e. multiple 'co-registered' 1,2,3, or 4 dimensional data sets, e.g. multiple 'co-registered' time series, multimodal image data, space/time data..) via the use of a set of image templates, including:
   a) single voxels
   b) linear sequences in cardinal directions (ref.)
   c) notches in cardinal directions (ref.)
   d) light cones in cardinal directions and 45 degree angles (ref.)
2. Clustering methods for generating disrete ('grey') levels from the constructed feature space (the levels are then typically mapped to the original image space at the anchor points of the templates)
3. Building co-occurrence matrices from a discrete level 'image' or a pair of discrete level 'images', where the discrete level 'images' are typically generated via feature space clustering of the original multimodal data sets (time series, images, space/time data. . .)
4. Estimation of various complexity/texture measures from the co-occurrence matrices.

gentex.comat.**cmad**(*images*, *masks*, *distance*, *angles*, *levels*)
Uses the comat or comat_2T functions to generate co-occurence matrices at the specified anlge(s) and distance(s) provided, which is more in the spirit of the original Haralick papers on texture analysis. So far this only makes sense for 2 and 3 dimensions (well 4 might make sense but. . .)

Parameters

images: 1 or 2 element 1d python array of 1-4 dimensional ndarray(s)

of dtype int consisting of an input image(s).

masks: 1 or 2 element 1d python array of 1-4 dimensional ndarray(s) of dtype int consisting of an input mask(s).Determines which voxels to use for building co-occurence matrix

distance: float distance in image to use as offset for calculating co-occurrence matrix

angles: float 1 or 2 element 1d python array of angle(s) to use for direction to voxel in calculating co-occurrence matrix. For 2D images the only angle, angles[0], corresponds to the standard angle from the x-axis in polar co-ordinates. For 3D images, angles[0] corresponds to the angle theta in spherical co-ordinates, i.e. the angle from the z-axis, and angles[1] corresponds to phi, i.e. the angle in the x-y plane.

levels [int] 1 or 2 element 1d python array with number of discrete levels in the image(s) (256 for an 8-bit image but any number of cluster values for general templated images)

Returns

2D ndarray The grey-level co-occurrence histogram. The value P[i,j] is the number of times that gray-level j occurs at the offset specified by distance and thetas, from gray-level i.

gentex.comat.**comat**(*image*, *mask*, *coords*, *levels=255*)

Calculates the co-occurrence histogram of an image given an offset.

Parameters

image: 1-4 dimensional ndarray of dtype int

Input image.

mask: 1-4 dimensional ndarray of dtype int Input mask (same size as image, 0,1 array) Determines which voxels to use for building co-occurence matrix

coords [1D ndarray] coordinate offset array with the appropriate number of dimensions (1-4) for building cooccurence matrix.

levels [int] The input image should contain integers in [0, levels-1], where levels indicate the number of discrete image or grey levels counted (256 for an 8-bit image but any number of cluster values for general templated images)

Returns

2D ndarray The grey-level co-occurrence histogram. The value P[i,j] is the number of times that gray-level j occurs at offset coords from gray-level i.

gentex.comat.**comat_2T**(*image1*, *mask1*, *image2*, *mask2*, *coords*, *levels1=255*, *levels2=255*)

Calculate the co-occurrence histogram from 2 images given an offset.

Parameters

image1: 1-4 dimensional ndarray of dtype int

Input image 1.

mask1: 1-4 dimensional ndarray of dtype int Input mask 1 (same size as image, 0,1 array) Determines which voxels to use for building co-occurence matrix

image2: 1-4 dimensional ndarray of dtype int Input image 2.

mask2: 1-4 dimensional ndarray of dtype int Input mask 2 (same size as image, 0,1 array)

coords [1D ndarray] coordinate offset array with the appropriate number of dimensions (1-4) for building cooccurence matrix.

levels1 : int

levels2 [int] The input images should contain integers in [0, levels(1,2)-1], where levels indicate the number of discrete image or grey levels counted (256 for an 8-bit image but any number of cluster values for general templated images)

Returns

2D ndarray The grey-level co-occurrence histogram. The value P[i,j] is the number of times that gray-level j occurs at offset coords from gray-level i.

gentex.comat.**comat_2T_mult**(*image1*, *mask1*, *image2*, *mask2*, *coordset*, *levels1=255*, *levels2=255*)

Generates and sums co-occurrence histograms from 2 images given a set of offsets.

Parameters

image1: 1-4 dimensional ndarray of dtype int

Input image 1.

mask1: 1-4 dimensional ndarray of dtype int Input mask 1 (same size as image, 0,1 array) Determines which voxels to use for building co-occurence matrix

image2: 1-4 dimensional ndarray of dtype int Input image 2.

mask2: 1-4 dimensional ndarray of dtype int Input mask 2 (same size as image, 0,1 array)

coordset [1D ndarray of coordinate offset sets] Array of coordinate offset arrays with the appropriate number of dimensions (1-4) for building cooccurence matrices.

levels1 : int

levels2 [int] The input images should contain integers in [0, levels(1,2)-1], where levels indicate the number of discrete image or grey levels counted (256 for an 8-bit image but any number of cluster values for general templated images)

Returns

2D ndarray The grey-level co-occurrence histogram. The value P[i,j] is the number of times that gray-level j occurs at offset coords from gray-level i.

gentex.comat.**comat_mult**(*image*, *mask*, *coordset*, *levels=255*)

Generates and sums co-occurrence histograms of an image given a set of offsets.

Parameters

image: 1-4 dimensional ndarray of dtype int

Input image.

mask: 1-4 dimensional ndarray of dtype int Input mask (same size as image, 0,1 array) Determines which voxels to use for building co-occurence matrix

coordset [1D ndarray of coordinate offset sets] array of coordinate offset arrays with the appropriate number of dimensions (1-4) for building cooccurence matrices.

levels [int] The input image should contain integers in [0, levels-1], where levels indicate the number of discrete image or grey levels counted (256 for an 8-bit image but any number of cluster values for general templated images)

Returns

2D ndarray The summed grey-level co-occurrence histogram. The value P[i,j] is the number of times that gray-level j occurs at offset coords from gray-level i summed over all offsets passed to comat_mult.

4.2 gentex.sphere

4.3 gentex.template

class gentex.template.**Template**(*type*, *sizes*, *dimension*, *inclusion*, *handedness=None*, *axbase=None*, *anchoff=None*, *shift=None*)

Class template for generating lists of template voxels

Parameters

type: string Required by constructor. The type of template currently available types are:

- 'RectBox' - rectangular box (1,2,3,4 dimensions) template origin is center of box
- 'RectShell' - shell of rectangular box (1,2,3,4 dimensions) template origin is center of shell
- 'Ellipsoid' - ellispoid (1,2,3,4 dimensions) template origin is center of ellipsoid
- 'EllipsoidShell' - ellipsoidal shell (1,2,3,4 dimensions) template origin is center of shell
- 'Line' - linear template template origin is first point of line
- 'Notch' - notch template template origin is point about which notch is built
- 'Cone' - cone template template origin is start of half cone

sizes: 1D int array (can be empty) Attributes of sizes required for constructing template

dimension: int Dimension of template

inclusion: bool Whether or not to include anchor point (i.e. [0], [0,0],...)

handedness: 1D int array If there are axial asymetries in the template (e.g. Notch) can pass in a vector with +1 for 'right' and -1 for 'left' (default is [1], or [1,1], or...)

axbase: List of ints (each list of length = dimension) Basis vector specifying axis, when appropriate, for direction of template (can be empty) - component lengths will be ignored; only whether the component is zero or nonzero, and the sign will be considered (i.e. only co-ordinate axes and '45 degree' lines will be considered as template axes), so e.g:

```
[1,0] ~ [10,0] ~ x-axis in 2D
[0,1,0] ~ [0,33,0] ~ y-axis in 3D
[1,-1] ~ [30,-20] ~ [108,-1] ~ 135 degree axis in 2
```

if axbase is empty template will pick axes according to following conventions:

- templates requiring single axis specification (e.g. line, notch, cone) will always use positivedirection of first dimension

- templates requiring multiple axis specification, e.g. rectangular parallelipipeds and ellipsoids will choose:
  - largest dimension (e.g. semi-major axis) in positive direction of first dimension
  - next largest dimension (e.g. semi-minor axis) in positive direction of second dimension
  - etc.

anchoff: list of ints Offset of anchor point from template [0,0,0] in template (usually assume [0,0,0])

shift: List of int List of ints to use if you want to shift all points in offset array - useful, e.g. if you want to build cooccurence arrays from offset templates - build one template (set of offsets with no shift and another with an appropriate shift; those can each be passed to the feature space cluster algorithm, then those to the cooccurence matrix builder, and that to the texture measure generator.

4.4 gentex.texmeas

gentex.texmeas package

class gentex.texmeas.**Texmeas**(*comat, measure='Statistical Complexity', coordmom=0, probmom=0, rllen=0, clusmom=0, clusp=0.001, samelev=True, betas=[-20, 20, 40]*)

Class texmeas for generating texture measures from co-occurrence matrix

Parameters

comat: ndarray Non-normalized co-occurrence matrix - chi-squared conditional distribution comparisons require the actual number of counts so don't normalize this before sending in

measure: string Texture measure (default = 'Statistical Complexity'). Choice of:

- 'CM Entropy'
- 'EM Entropy'
- 'Statistical Complexity'
- 'Energy Uniformity'
- 'Maximum Probability'
- 'Contrast'
- 'Inverse Difference Moment'
- 'Correlation'
- 'Probability of Run Length'
- 'Epsilon Machine Run Length'
- 'Run Length Asymmetry'
- 'Homogeneity'
- 'Cluster Tendency'
- 'Multifractal Spectrum Energy Range'
- 'Multifractal Spectrum Entropy Range'

coordmo: int Moment of coordinate differences in co-occurrence matrix needed for calculating 'Contrast' and 'Inverse Difference Moment' (default=0)

probmom: int Moment of individual cooccurence probabilities needed for calculating 'Contrast' and 'Inverse Difference Moment' (default=0)

rllen: int Length of run length used for generating probability of a run length (the higher this probability the larger the constant patches on the scale used for generating the co-occurence matrix) or the epsilon machine run length (default=0)

clusmom: int Moment used for generating cooccurence cluster tendency (default=0)

samelev: bool Whether to treat the rows and columns in the cooccurence matrix as identical 'states' (the methods are very general so this needn't be the case, e.g. different template shapes from different images with different quantization levels could be used to generate the cooccurence matrix which could be of arbitrary shape)

default = True assumes the cooccurrence matrix is square and the rows and columns correspond to the same 'state'

betas: array An array of 3 values, the lower limit, the upper limit and the number of steps to use as the 'inverse temperature' range for estimating the multifractal spectrum from an epsilon machine - getting the range right for an 'arbitrary' epsilon machine is tricky and is expected to be reset over a number of trials before getting a full spectrum estimate. For details on the rationale and algorithm see:

K. Young and J. P. Crutchfield, 'Fluctuation Spectroscopy', Chaos, Solitons, and Fractals 4 (1993) 5-39.

Attributes

emclus: int Number of clusters ('states') found when estimating an epsilon machine from the co-occurrence matrix.

emest: bool Whether or not an epsilon machine has been estimated yet

emmat: float The estimated epsilon machine as a standard Markov process transition matrix.

condo: 2d-array Co-occurrence matrix renormalized as a rowise matrix of conditional probabilites - built as part of epsilon machine estimation

emclasses: list List of which of the values in emclus each row in condo (and hence the cooccurence matrix) belongs to

clusp: float Chisquared p value to use for clustering epsilon machine rows

val: float Value of most recently calculated texture measure

mfsspec: array Array containing the multifractal spectral estimates obtained over the range of 'inverse temperatures' provided in betas

currval: string One of the listed measures method which constitutes the current value in val

Methods

calc_measure(self[, measure, coordmom, ...])	Calculates the appropriate texture measure and puts the value in the class variable val and updates the class variable currval with the passed string

Continued on next page

Table 1 – continued from previous page

est_em(self)	Estimate an epsilon machine from a co-occurrence matrix with #rows = #cols, done implicitly whenever one of the related complexity/entropy measures (EM Entropy, Statistical Complexity, Epsilon Machine Run Length) are calculated.
est_multi_frac_spec(self)	TODO

calc_measure(*self*, *measure='Statistical Complexity'*, *coordmom=0*, *probmom=0*, *rllen=0*, *clusmom=0*, *samelev=True*)

Calculates the appropriate texture measure and puts the value in the class variable val and updates the class variable currval with the passed string

For a discussion of Haralick co-occurrence style texture measures see: R. M. Haralick, 'Statistical and structural approaches to texture'. Proceedings of the IEEE May 1979, 67(5). 786-804.

Parameters

measure: string One of the following measure methods (default = 'Statistical Complexity'):

- 'CM Entropy'
- 'EM Entropy'
- 'Statistical Complexity'
- 'Energy Uniformity'
- 'Maximum Probability'
- 'Contrast'
- 'Inverse Difference Moment'
- 'Correlation'
- 'Probability of Run Length'
- 'Epsilon Machine Run Length'
- 'Run Length Asymmetry'
- 'Homogeneity'
- 'Cluster Tendency'
- 'Multifractal Spectrum Energy Range'
- 'Multifractal Spectrum Entropy Range'

est_em(*self*)

Estimate an epsilon machine from a co-occurrence matrix with #rows = #cols, done implicitly whenever one of the related complexity/entropy measures (EM Entropy, Statistical Complexity, Epsilon Machine Run Length) are calculated.

For info on epsilon machines and the related measures see:

- K. Young, Y. Chen, J. Kornak, G. B. Matson, N. Schuff, 'Summarizing complexity in high dimensions', Phys Rev Lett. (2005) Mar 11;94(9):098701.
- C.R. Shalizi and J. P. Crutchfield, 'Computational Mechanics: Pattern and Prediction, Structure and Simplicity', Journal of Statistical Physics 104 (2001) 819–881.

- K. Young and J. P. Crutchfield, 'Fluctuation Spectroscopy', Chaos, Solitons, and Fractals 4 (1993) 5-39.
- J. P. Crutchfield and K. Young, 'Computation at the Onset of Chaos', in Entropy, Complexity, and Physics of Information, W. Zurek, editor, SFI Studies in the Sciences of Complexity, VIII, Addison-Wesley, Reading, Massachusetts (1990) 223-269.
- C.R. Shalizi and J. P. Crutchfield, 'Computational Mechanics: Pattern and Prediction, Structure and Simplicity', Journal of Statistical Physics 104 (2001) 819–881.

est_multi_frac_spec(*self*)
TODO

PYTHON MODULE INDEX

g

INDEX