Protein Zernike Patches

Created by Emily Baum in collaboration with the Bilodeau Group

UVA Department of Chemical Engineering

Last Updated 4/18/2024

Table of Contents

get_patch_coords.py

This set of functions can be used to generate a set of patches from a protein surface, format surface features, and prepare the patches to be represented by 3D Zernike descriptors.

- **read_ply(file_name, n_edges=3)**

  This function is used to read in a protein.ply file and extract the points and edges information of the protein surface. The function is designed with the output of the EDTSurf program in mind, but any ply file should work.

  """

  Parameters
  ----
  file_name : file.ply

  Protein Connolly surface file in .ply format (output file from EDTsurf)

  n_edges : int, optional

  Number of edges on each face of the mesh. The default is 3. (triangular mesh)

  Returns
  ----
  vertices : numpy array of float

  Points on the Connolly surface (x,y,z) and color code (r,g,b) from EDTsurf.
  Recommended color based on the nearest atom.

  faces : numpy array of float

  Information used by ply files to create a triangular mesh.
  [n_edges=3, point_index_1:n_edges, r, g, b]

  """

- **make_surface_patches(surface_points, n_patches=1, patch_radius=8, features=True, atom_centers=None)**

  This function is used to generate surface patches from the protein surface point coordinate and feature information. It can create any number of random unique patches

of a specified radius, up to the number of surface points. If you have already calculated custom features (for example - electrostatics or a hydropathy scale) at each point on the surface, append the values of the features to surface_points before running this function. If you want to include both the atom_center points and features, you will need to run the function with features = False and append the features to the patch_dict output.

"""

Parameters
---
surface_points : numpy array of float

Coordinates, color code, (and opt. custom features) of protein surface points.
Recommended to use vertices output from read_ply().

n_patches : int, optional

Number of unique protein surface patches to generate. The default is 1.

patch_radius : float, optional

The radius of the generated protein patch in Angstroms. The default is 8.

features: bool, optional

Include pre-defined features along with surface coordinates. The default is True.

atom_centers : file.pdb, optional

The protein file from which to include atom center information.
This may be desired to distinguish a patch surface from its inverse.
Using this with features = True may cause issues downstream unless features
for atom center coordinates are added manually. The default is None.

Returns
---
patch_dict : dict of numpy arrays

The dictionary contains the coordinates of all surface points (and opt
atom centers) within patch_radius of a random surface point.
Dictionary entries are named by the centerpoint_index of each patch.
If features = True, each dictionary entry will contain an array of coordinates
and an array of features.

"""

- **get_atom_centers(file_name)**

  This function is called by make_surface_patches() when atom_centers is specified with a pdb file name. It uses the biopython package to find the atom coordinates of the specified protein structure, removing any other molecules. If you intend to create a protein patch with surface depth, this is a good place to start. Modification would be needed to fill in the space between the surface and the atom centers.

  """

  Parameters
  ----
  file_name : file.pdb

  The protein file from which to extract atom center information.

  Returns
  ----
  atom_centers : numpy array of float

  The coordinates (x,y,z) of each atom in the protein structure (including hydrogens). Water molecules and heteroatoms (such as ions and small molecules) are removed.

  """

- **scale_patch(patch_dict, patch_radius=8, features=True)**

  This function is critical for representing a protein surface patch with Zernike descriptors. 3D Zernike polynomials are defined only within a unit sphere (r <= 1), so patches need to be scaled to a radius of 1 to avoid cutoff. This function takes in the patch dictionary from make_surface_patches() and scales each patch in the dictionary.

  """

  Parameters
  ----
  patch_dict : dict of numpy arrays

  The dictionary contains the coordinates of all surface points (and opt atom centers) within patch_radius of a random surface point. Dictionary entries are named by the centerpoint_index of each patch.

patch_radius : float, optional

The radius of the generated protein patch in Angstroms. The default is 8.

features: bool, optional

Indicate whether patch_dict was created with predefined features along with surface coordinates. The default is True.

Returns
---
scaled_dict : dict of numpy arrays

The dictionary contains the scaled coordinates of all surface points (and opt atom centers) within patch_radius of a random surface point. Dictionary entries are named by the centerpoint_index of each patch. Coordinates must be scaled into a unit sphere for the 3D Zernike function.

"""

Zernike_Polynomials.py

This set of functions takes a set of protein surface features and scaled coordinates and transforms them into a compact, information-dense, rotationally invariant descriptor fit for use in neural networks.

- **zernike_polynomials(voxels, N=20)**

  This function calculates the value of Zernike polynomials at a set of coordinates within a unit sphere. Zernike polynomials are a set of functions orthogonal on a unit sphere; Zernike polynomials are both continuous and non-redundant, which makes them robust for storing protein patch features. Relevant equations are shown below and discussed in greater detail at https://doi.org/10.1002/prot.22030.

$$Z_{nl}^{m}(\mathbf{x}) = \sum_{v=0}^{k} q_{kl}^{v} |\mathbf{x}|^{2v} e_{l}^{m}(\mathbf{x})$$

$$\text{with } -l < m < l,\ 0 \leq l \leq n, \text{ and } (n-l) \text{ even where } 2k = n - l$$

$$q_{kl}^{v} = \frac{(-1)^k}{2^{2k}} \sqrt{\frac{2l+4k+3}{3}} \binom{2k}{k} (-1)^v \frac{\binom{k}{v}\binom{2(k+l+v)+1}{2k}}{\binom{k+l+v}{k}}$$

$$e_l^m(\mathbf{x}) \equiv r^l Y_l^m(\vartheta, \varphi) = r^l c_l^m \left(\frac{ix-y}{2}\right)^m z^{l-m} \times \sum_{\mu=0}^{\left\lfloor \frac{l-m}{2} \right\rfloor} \binom{l}{\mu} \binom{1-\mu}{m+\mu} \left(-\frac{x^2+y^2}{4z^2}\right)^\mu$$

where c_l^m are normalization factors:

$$c_l^m = c_l^{-m} = \frac{\sqrt{(2l+1)(l+m)!(l-m)!}}{l!}$$

"""

Parameters
---
voxels : numpy array of float

The coordinates of the points at which to calculate the value of the 3D Zernike function. A voxelized grid of evenly spaced points is recommended so that the polynomials are defined across the entire unit sphere.

N : int, optional

Order of approximation of the 3D Zernike function. Higher order provides higher resolution but increases computational time. Choosing an order too large may lead to overfitting. Must be consistent across polynomials, moments, and descriptors. The default is 20.

Returns

Z : numpy array of float

Calculates the value of the 3D Zernike function at a given set of coordinates within a unit sphere. Equation numbers follow: https://doi.org/10.1002/prot.22030.

"""

- **voxelize(scaled_patch, gridpoints=20, feature_array=None)**

This function creates a grid of equally spaced points spanning from -1 to 1 in each direction. Of these, only points within a unit sphere are retained, because outside points would be unmappable by Zernike polynomials. A mapping of the surface patch to the voxelized sphere is returned along with the voxel center coordinates. If an array of features is specified, a mapping of features to the voxel grid will also be returned. It is recommended for gridpoints to be specified as an even number, because with the above equations, Zernike polynomials are undefined at [0,0,0].

"""

Parameters
----
scaled_patch : numpy array of float

The scaled coordinates of all surface points (and opt atom centers) within patch_radius of a random surface point.

gridpoints : int, optional

The number of voxels in each dimension to construct a unit grid (gridpoints**3). Only the voxels within a unit sphere are retained. The default is 20.

feature_array: numpy array of float, optional

An array of features to map to the voxelized grid. The default is None.

Returns
----
voxels : list of float

The coordinates of each voxel centerpoint.

voxel_surf: numpy array of int

The mapping of the surface patch to the voxel grid. The values in the list are 1 if the voxel contains part of the surface, or 0 if it does not.

voxel_feat: numpy array of float, optional

The values of the features defined in feature_array for each voxel. Empty voxels are given a default value of 0, and voxels containing multiple surface points are assigned an average value.

"""

- **voxel_surface(scaled_patch, voxels, gridpoints)**

This function is called by voxelize() in order to map the surface patch into the voxelized grid.

"""

Parameters
---
scaled_patch : numpy array of float

The scaled coordinates of all surface points (and opt atom centers) within patch_radius of a random surface point.

voxels : list of float

The coordinates of each voxel centerpoint.

gridpoints : int

The number of voxels in each dimension to construct a unit grid (gridpoints**3). Only the voxels within a unit sphere are retained. The default is 20.

Returns
---
voxel_surf : numpy array of int

The mapping of the surface patch to the voxel grid. The values in the list are 1 if the voxel contains part of the surface, or 0 if it does not.

"""

- **voxel_features(scaled_patch, feature_array, voxels, gridpoints)**

  If feature_array is specified, this function is called by voxelize() in order to map features defined on the surface patch into the voxelized grid.

  """

  Parameters
  ----
  scaled_patch : numpy array of float

  The scaled coordinates of all surface points (and opt atom centers) within patch_radius of a random surface point.

  feature_array: numpy array of float

  An array of features to map to the voxelized grid.

  voxels : list of float

  The coordinates of each voxel centerpoint.

  gridpoints : int

  The number of voxels in each dimension to construct a unit grid (gridpoints**3). Only the voxels within a unit sphere are retained.

  Returns
  ----
  voxel_feat : numpy array of float

  The mapping of the surface patch features to the voxel grid. Empty voxels are assigned a default value of 0. Voxels containing multiple surface points are assigned the average feature value within the voxel.

  """

- **zernike_moments(z_polynomials, voxel_surf, gridpoints=20, N=20)**

  This function calculates the Zernike moments (a product of Zernike polynomials and the protein surface geometry or features). Relevant equations are shown below and discussed in further detail at https://doi.org/10.1145/781606.781639.

$$\Omega_{nl}^{m} := \frac{3}{4\pi} \int_{|\mathbf{x}|\leq 1} f(\mathbf{x}) \overline{Z_{nl}^{m}(\mathbf{x})} d\mathbf{x}.$$

$$\Omega_{nl}^{-m}(\mathbf{x}) = (-1)^{m} \overline{\Omega_{nl}^{m}(\mathbf{x})}.$$

"""

Parameters
---
z_polynomials : numpy array of float

The value of the 3D Zernike function at a given set of coordinates within a unit sphere.

voxel_surf : numpy array of float

The mapping of the surface patch to the voxel grid. The values in the list are 1 if the voxel contains part of the surface, or 0 if it does not. Alternatively, an array of values for mapped features can be used as input.

gridpoints : int, optional

The number of voxels in each dimension to construct a unit grid (gridpoints**3). Only the voxels within a unit sphere are retained. The default is 20.

N : int, optional

Order of approximation of the 3D Zernike function. Higher order provides higher resolution but increases computational time. Choosing an order too large may lead to overfitting. Must be consistent across polynomials, moments, and descriptors. The default is 20.

Returns
---
corrected_moments : numpy array of float

Calculates the 3D Zernike moments, necessary for computation of the Zernike descriptors. Equation numbers follow: https://doi.org/10.1145/781606.781639

"""

- **zernike_descriptor(moments, N=20)**

  This function calculates the array of 3D Zernike descriptors for a protein surface patch. This vector is a compact, information-dense, rotationally invariant descriptor fit for use in neural networks. The Zernike descriptor equation is shown below and discussed in more detail at https://doi.org/10.1002/prot.22030.

$$F_{nl} = \sqrt{\sum_{m=-l}^{m=l} (\Omega_{nl}^{m})^2}$$

  """

  Parameters
  ----
  moments : numpy array of float

    The 3D Zernike moments for a patch voxelized within a unit sphere.
    The moments can contain geometric or feature information.

  N : int, optional

    Order of approximation of the 3D Zernike function. Higher order provides higher resolution but increases computational time. Choosing an order too large may lead to overfitting. Must be consistent across polynomials, moments, and descriptors. The default is 20.

  Returns
  ----
  zernike_descriptor : numpy array of float

    A compact, non-redundant, rotationally invariant descriptor used to represent protein surface patches.
    Equation numbers follow: https://doi.org/10.1002/prot.22030.

  """