8CC00_clusteringAndClassification

Rebecca Kuepper

Apr 17, 2021

CONTENTS:

CHAPTER

ONE

SOURCE FILES

1.1 Clustering module

Python scipt for clustering of data by means of k-means, evaluation with silhouette scores and HCS clustering.

clustering.**HCS**(*graph: dict*, *originalEdges: list*, *nrIt: int = 10*, *clusters=[]*) → list
Highly connected subgraph clustering, using Karger cut.

Parameters

- **graph** – graph dict on which to perform HCS clustering.
- **originalEdges** – edges of the graph.
- **nrIt** – nr of iterations that the kargercut should be performed to assume the minimum cut is reached.
- **clusters** – parameter to remember previous clusters during recursion. Should be the empty list when called for the first time.

Returns list containing the clusters in graph dict format.

clustering.**calculateCentroids**(*clusteredData: list*, *dim: int*) → list
Recalculate the new centroid based on the averages in the old cluster configuration.

Parameters

- **clusteredData** – list containing the old clusters.
- **dim** – required dimensions for the centroids.

Returns List containing new centroids on the position of the average of the cluster.

clustering.**contractEdge**(*graph: dict*, *v: str*, *w: str*) → None
Edgecontraction. Create one supernode from two nodes.

Warning: The original graph is overwritten.

Parameters

- **graph** – dict of graph in which edges need to be contracted.
- **v** – first node to be merged into supernode.
- **w** – second node to be merged into supernode.

clustering.**createEdges**(*c: float*, *edgesdict: dict*) → list
Evaluate whether the correlation of an edge is above c and return a list of edges that does.

Parameters

- **c** – threshold value for correlation coefficient of edge

- **edgesdict** – dictionary containing all possible edges and their correlation coefficients.

Returns List like [(node1, node2), (node1, node3)] where nodes are strings.

clustering.**highlyConnected**(*graph: dict*, *mincut: int*) → bool
Decide whether graph is highly connected.

Parameters

- **graph** – dict of graph to decide on.
- **mincut** – minimum number of cuts.

Returns minimumcut > nrNodes/2

clustering.**kMeans**(*data: list*, *k: int*, *distMethod: str*, *maxit: int*) → list
k-means algorithm, using the distMethod to calculate the distance between data points.

Parameters

- **data** – list of int, float, list or tuple values for datapoints.
- **k** – integer to decide the number of centroids.
- **distMethod** – Method by which the distance between data points needs to be chosen.
- **maxit** – Maximum number of iterations

Returns a list of sets per cluster.

clustering.**karger2subgraph**(*supernodesgraph: dict*, *originalEdges: list*) → tuple
Create subgraphs from the resulting supernodes graph after kargercut.

Parameters

- **supernodesgraph** – the two supernodes that remain after Karger cut.
- **originalEdges** – list of edges from the original graph (before the cut).

Returns tuple containing two subgraphs in dict format.

clustering.**kargerMinCut**(*g: dict*) → tuple
Perform a kargercut in a graph.

Parameters **g** – dict of graph in which karger cut needs to be done.

Returns minimum nr of edges that need to be cut, graph that remains

clustering.**nodepairFraction**(*overallCorrelations: dict*, *c: float*) → float
Calculate the fraction of node pairs that have an absolute value of their correlation coefficient of at least c.

Parameters

- **overallCorrelations** – dictionary containing node pairs and their correlations (can be calculated with the function overallCorrelationcoefficients)
- **c** – Threshold for fraction.

Returns fraction of number of node pairs that is above threshold.

clustering.**overallCorrelationcoefficients**(*data: collections.abc.Iterable*, *names: list*) → dict
Create a dict in which for each node pair the correlation coefficient is calculated.

Parameters

- **data** – iterable containing coordinates for each datapoint.
- **names** – isterable containing respective names for each datapoint in data.

Returns Dict like {(node1, node2): correlationcoefficient} where the nodes are strings and the correlation coefficient is a float.

clustering.**silhouetteScore**(*clusteredData: list*) → tuple

Calculate silhouette score for clustered data. Function returns tuple containing on the first index the silhouette score for the clustering and in the second index a dict containing the silhouettes for all datapoints.

Parameters **clusteredData** – list containing sets of data per cluster

Returns (float, dict) where the float is the overall silhouette score for the clustering and the dict contains the silhouette per datapoint like {(coordinate): silhouette}

```
>>> v = [{(1.5, 0.5), (1., 1.5), (0.5, 0.5), (0.5, 2.)}, {(6, 6), (5.5, 6), (6, 5.
↪5)}, {(4.5, 2.), (4., 2.), (3.5, 1.5)}]
>>> x, y = silhouetteScore(v)
0.9352832294102621 {(1.0, 1.5): 0.8928571428571429, (0.5, 0.5): 0.924812030075188,
↪ (1.5, 0.5): 0.8567493112947658, (0.5, 2.0): 0.8986666666666667, (6, 6): 0.
↪9884169884169884, (6, 5.5): 0.9858490566037735, (5.5, 6): 0.9873949579831933,␣
↪(4.5, 2.0): 0.9482758620689655, (3.5, 1.5): 0.9147540983606557, (4.0, 2.0): 0.
↪9550561797752809}
```

clustering.**squaredEuclideanDist**(*u*, *v*) → float

Calculate the Euclidean squared distance between u and v.

Parameters

- **u** – 1D or ND coordinate in int or float, or list or tuple respecitvely.
- **v** – 1D or ND coordinate in int or float, or list or tuple respecitvely.

Returns float of Euclidean squared distance between u and v.

1.2 Classification module

Script for k nearest neighbour classification and trainingsset generation.

classification.**checkLabel**(*datapointname: str*, *label: str*, *labelsdict: dict*) → bool

Checks if assigned label to datapoint is correct according to the information in the labelsdict.

Parameters

- **datapointname** – The node to be checked.
- **label** – The label to be checked
- **labelsdict** – The dict in which all labels for all datapoints are stored

Returns True if label is correct, False if incorrect.

classification.**findName**(*point: list*, *data: list*, *names: list*) → str

Find name that belongs to a datapoint. Note that data and names indices should match.

Parameters

- **point** – list containing coordinate of the point to be named.
- **data** – list containing all points, including above point.
- **names** – list containing respective names of points in data.

Returns name of point

classification.**generateTrainingset**(*fulldataset: collections.abc.Iterable*, *i: int*, *names: list*) → tuple

Generate a leave-one-out trainingsset at index i and return both.

Parameters

- **fulldataset** – the full dataset to be used
- **i** – the index of the point to be left out
- **names** – list of names for the datapoints

Returns tuple containing trainingset, trainingnames, (new point name, new point coordinate)

classification.**nearestNeighbour**(*trainingset: collections.abc.Iterable*, *newDataPoint*, *k: int*, *trainingnames: list*, *labelsdict: dict*, *distMethod: str = 'sqEucl'*) → str

Nearest neighbour algorithm for classification of data.

Parameters

- **trainingset** – dataset that does not contain newDataPoint.
- **newDataPoint** – datapoint to be labelled. May be int, float, list or tuple.
- **k** – nr of neighbours to be considered.
- **labelsdict** – dict containing labels for datapoints in trainingset
- **distMethod** – method of distance calculation to be used. (to be implemented)

Returns label for new datapoint

1.3 Data extraction, transformation and loading module

Extraction, transformation and loading of data.

dataETL.**extractData**(*filename: str*) → tuple

Extract data from file.

Parameters **filename** – name of the file from which data should be extracted.

Returns cellline coordinates, celllinenames, genenames

dataETL.**extractLabels**(*file: str*) → dict

Extract labels from csv file where names are in 2nd column and labels in the 4th and return in dict.

dataETL.**selectData**(*data: list*) → list

Select the data from the 145 cell lines that were assigned to me by email.

1.4 Data processing module

Some basic functions for the processing of data.

dataProcessing.**correlationCoefficient**(*param1: collections.abc.Iterable*, *param2: collections.abc.Iterable*) → float

Calculate the Pearson correlation coefficient of two parameters.

```
>>> correlationCoefficient([1, 2, 3, 5, 8], [0.11, 0.12, 0.13, 0.15, 0.18])
1.0
```

dataProcessing.**covariance**(*param1: list*, *param2: list*) → float
Return the covariance of parameter lists param1 and param2.

Assumption: param1 and param2 contain numbers and are of equal length.

Parameters

- **param1** – List of parameters to be compared.
- **param2** – List of parameters to compare with.

Returns covariance of param1 and param2.

```
>>> covariance([1, 3, 5, 11, 0, 4], [2, 6, 2, 78, 1, 4])
106.4
>>> covariance([1], [1, 2])
Traceback (most recent call last):
    ...
AssertionError: Parameter lists must be of the same length.
```

dataProcessing.**standardDeviation**(*param: list*) → float
Calculate the standard deviation of a list of measurements.

Parameters **param** – list of data of which the std needs to be calculated.

Returns standard deviation of param.

```
>>> standardDeviation([2, 4, 4, 4, 5, 5, 7, 9])
2.0
```

1.5 Graph module

Class for Graphs.

class Graph.**Graph**(*edges: list*)
Bases: object

removeEdge(*node1*, *node2*) → None
Remove edge between node 1 and node 2 from Graph.

removeNode(*node*) → None
Remove node from Graph.

1.6 main module

Main file for clustering and classification assignment.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX