# clustering

index
c:\users\geral\documents\matlab\hw06\clustering.py

Author: Gerald Baulig

## Modules

kernel numpy

## Classes

builtins.object

Spectral

class **Spectral**(builtins.object)

```
Spectral(X=None, gamma=1, epsilon=0, mode='default')

Spectral(X=None, gamma=1, epsilon=0, mode='default')
Container class for spectral clustering.
Computes the (W)eights, (D)egrees, (L)aplacian Matrix,
the eigenvalues and eigenvectors of a given dataset (X).
```

Methods defined here:

**__init__**(self, X=None, gamma=1, epsilon=0, mode='default')

```
Initialize self.  See help(type(self)) for accurate signature.
```

**cluster**(self, K, mode='select')

```
cluster(K, mode='select') -> kmeans generator
Initialize and returns a KMeans generator.
(See kmeans)

Args:
    K: Number of cluster centers
    mode: The initialization mode (See init_kmeans)
Returns:
    Generator of KMeans, yields:
        Y: The labels
        means: The cluster centers
        delta: The update delta
        step: The update step counter
```

**set**(self, X=None, gamma=None, epsilon=None, mode=None)

```
set(X=None, gamma=None, epsilon=None, mode=None)
Use this function to change one or several properties.
Recomputes the depending properties as required.

Args:
    X: Set the dataset - updates the whole spectral information
    gamma: Set gamma - updates W, D, L, eigval and eigvec
    epsilon: Set epsilon - updates D, L, eigval and eigvec
    mode: Set the mode - updates L, eigval and eigvec
```

---

Data descriptors defined here:

**D**

```
The degree matrix.
```

**L**
The laplacian matrix.

**N**
Number of datapoints.

**W**
The weight matrix.

**X**
The inserted dataset.

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**d**
The dimension of the dataset.

**eigval**
The eigenvalues extracted from L.

**eigvec**
The eigenvectors extracted from L.

**epsilon**
Epsilon for knn mode.

**gamma**
The gamma value for the RBF kernel.

**mode**
The KMeans initialization mode.

## Functions

**dbscan**(X, points, radius)

```
dbscan(X, points, radius) -> yields Y, x, step
Simple DBScan clustering with one path following agent.
Takes long computation time, because almost every point needs to be checked.
Otherwise it could not complete the pathfinding.
The utilization of the radius is very low.

Args:
    X: The dataset.
    points: Minimal number of points around core-points.
    radius: The scan radius.
Yields:
    Y: The labels.
    x: The current scan point.
    step: The step counter.
```

**init_kmeans**(X, K, mode='mean')

```
init_kmeans(X, K, mode='free') -> means
Initialize K cluster means on dataset X.

Args:
    X: The dataset.
    K: The number of cluster means.
    mode: The mode how to initialize the cluster means.
```

```
            mean = All means start (almost) at the mean of the dataset.
                Pro: The result is determenistic.
                Con: Needs more iterations.
            uniform = Uniform random distributed.
                Pro: May work well on uniform distributed datasets.
                Con: Ks may get lost in huge data gaps.
            normal = Normal random distributed.
                Pro: May work well on normal distributed datasets.
                Con: Ks may get lost in huge data gaps.
            select = Selects random points of the dataset.
                Pro: Makes sure that each K has at least one point.
                Con: Not determenistic like all the other random approaches.
            kmeans++ = Selects a random point and rearranges the
                probabillities of selecting the next point according
                to the distance.
                Pro: May have an appropriate distributed of Ks.
                Con: Great effort for a negligible improvment.
    Returns:
        means: The cluster means.
```

**kernel_trick**(gram, C)

```
    kernel_trick(gram, C) -> W
    Computes a Weight-Matrix of unsimilarity via "Kernel-Trick".

    Args:
        gram: The Gram-Matrix of a Kernel-Function.
        C: An Association-Matrix with cluster assigments per column.
```

**kmeans**(X, means, epsilon=0.0, max_it=1000, is_kernel=False)

```
    kmeans(X, means, epsilon=0.0, max_it=1000, isKernel=False) -
    > yields Y,  means, delta, step
    A generator for simple KMeans clustering.

    Usage:
        See demo_kmeans.py
    Args:
        X: The data N-by-d, where
            N=num datapoints
            d=dimensions
        means: The cluster centers.
        epsilon: Convergence threshold.
            (default=0)
        max_it:
        isKernel:
    Yields:
        Y: The labels or cluster association.
        means: The updated cluster centers.
        delta: The distance to the update step.
        step: The iteration step.
```

**square_mag**(u, v)

```
    square_mag(a, b) -> squared magnitude
    Calculates the row-wise squared magnitude.
    This function is preferably used for distance comparisons,
    because taking a square root has a high computation time.
        np.sum((a-b)**2, axis=1)

    Args:
        u: numpy.ndarray
        v: numpy.ndarray
    Returns:
        The row-wise squared magnitude of a and b.
```

## Data

**KMEANS_INIT_MODES** = ('mean', 'select', 'uniform', 'normal', 'kmeans++')

**LAPLACIAN_MODES** = ('default', 'shi', 'jordan')