In [1]:
```
import pysparnn as snn
```

In [2]:
```
import pysparnn.matrix_distance
```

In [3]:
```
import sklearn
import sklearn.metrics.pairwise
import scipy.sparse as sparse
import numpy as np


class UserCustomDistance(pysparnn.matrix_distance.MatrixMetricSearch):
    def __init__(self, features, records_data):
        super(UserCustomDistance, self).__init__(features, records_data)
        self.matrix = self.matrix
        self.max_overlap = self.matrix.shape[0]  # for testing purpose

    @staticmethod
    def features_to_matrix(features):
        return features

    @staticmethod
    def vstack(matrix_list):
        return np.vstack(matrix_list)

    def _transform_value(self, v):
        return v

    def user_distance_metric(self, u, v):
        rep = sparse.csr_matrix(np.minimum(u.A, v.A))
        return self.max_overlap - rep.sum()

    def _distance(self, a_matrix):
        return sklearn.metrics.pairwise.pairwise_distances(
            a_matrix, self.matrix, lambda u, v: self.user_distance_metri
c(u, v))
```

In [4]:
```
class SpencerUserCustomDistance(pysparnn.matrix_distance.MatrixMetricSea
rch):
    def __init__(self, features, records_data):
        super(SpencerUserCustomDistance, self).__init__(features, record
s_data)
        self.matrix = self.matrix
        self.max_overlap = self.matrix.shape[0]  # for testing purpose

    @staticmethod
    def features_to_matrix(features):
        return features

    @staticmethod
    def vstack(matrix_list):
        return np.vstack(matrix_list)

    @staticmethod
    def _transform_value(self, v):
        return v


    def _distance(self, a_matrix):
        return np.array(self.max_overlap - self.matrix.dot(a_matrix.tran
spose()).transpose().todense())
```

In [5]:
```
B = sparse.csr_matrix([[0, 0, 1],[0, 1, 1],[1, 1, 1]])
A = sparse.csr_matrix(
[
    [0, 0, 1],
    [0, 1, 1],
    [1, 1, 1]
])
```

In [6]:
```
u_dist = UserCustomDistance(A, range(A.shape[0]))
u_dist._distance(B)
```

Out[6]:
```
array([[ 2.,  2.,  2.],
       [ 2.,  1.,  1.],
       [ 2.,  1.,  0.]])
```

In [7]:
```
su_dist = SpencerUserCustomDistance(A, range(A.shape[0]))
su_dist._distance(B)
```

Out[7]:
```
array([[2, 2, 2],
       [2, 1, 1],
       [2, 1, 0]])
```

In [8]:
```
import scipy
```

In [9]:
```
from scipy.sparse import rand
```

In [10]:
```
from scipy.sparse import csr_matrix
```

In [39]:
```
num_records = 100000
matrix = csr_matrix(rand(num_records, num_records, density=0.0002, forma
t='csr'))
```

In [40]:
```
matrix.sum(axis=1).mean()
```

Out[40]:
```
10.006485936445173
```

In [41]:
```
import pysparnn.cluster_index as ci
```

In [42]:
```
import time
```

In [45]:
```
t0 = time.time()
cp2 = ci.MultiClusterIndex(matrix, np.array(range(num_records)), distanc
e_type=SpencerUserCustomDistance)
t1 = time.time()
(t1 - t0)
```

Out[45]:
```
38.44776201248169
```

In [ ]:
```
t0 = time.time()
cp2 = ci.MultiClusterIndex(matrix, np.array(range(num_records)), distanc
e_type=UserCustomDistance)
# killed after 7 min, this will take a very long time probably
```

In [47]:
```
t1 = time.time()
(t1 - t0)
```

Out[47]:
```
453.0269010066986
```