Distribution Sorting Algorithm

December 27, 2020

1 Objective

We will create an alogrithm which sorts data into clusters of similar distributions.

2 Packages

[87]:
```
import numpy as np
from numpy.random import normal, uniform
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from scipy.stats import ks_2samp
```

3 Randomly Generated Data

In practice, this step is not necessary. Image we have an Excel spreadsheet which records various aspects of events and the consequences of the events. We know columns 'prop-1' and 'prop-2' have relavent information regarding the distributions of the corresponding consequences.

[93]:
```
# Create dataframe
data = pd.DataFrame(index = range(1500), columns = ['prop-1', 'prop-2',␣
 ↪'consequence'])

# Create labels
data.loc[0:499, 'prop-1'], data.loc[500:999, 'prop-1'], data.loc[1000:1499,␣
 ↪'prop-1'] = 'A', 'B', 'C'
data.loc[0:249, 'prop-2'], data.loc[250:749, 'prop-2'], data.loc[750:,␣
 ↪'prop-2'] = 1, 2, 3

# Generate random numbers
data.loc[0:499, 'consequence'] = normal(loc = 0, scale = 1, size = 500)
data.loc[500:999, 'consequence'] = uniform(low = -1, high = 1, size = 500)
data.loc[1000:1499, 'consequence'] = normal(loc = 0.25, scale = 1, size = 500)
```

[89]:
```
data.head()
```

[89]:
```
  prop-1 prop-2 consequence
0      A      1    -1.61085
1      A      1   0.0874867
2      A      1    0.222743
3      A      1     1.38703
4      A      1     1.58597
```

[90]:
```
data.tail()
```

[90]:
```
     prop-1 prop-2 consequence
1495      C      3    0.383376
1496      C      3    0.416639
1497      C      3     1.90259
1498      C      3   -0.129283
1499      C      3     2.56419
```

4 Function

We will create our sorting function. The variable 'columns' selects the columns of your data which may have useful information regarding the distribution of the column 'consequence'. The variable 'crit_val' specifies the acceptable risk that disaggregated distributions are, in fact, the same. For example, if crit_val = 0.10, then the algorithm will disaggregate the data if the chance that the data are from the same distribution is less than 10%.

[91]:
```
# Create a function to sort our data into distinct distributions
def sort_data(data, columns, crit_val):

    # Obtain only unique combinations of columns in 'columns'
    results = data[columns].drop_duplicates()

    # Reset index
    results.reset_index(inplace = True, drop = True)

    # Save number of rows
    n = len(results)

    # Record pad for distance; must be > 1
    pad = 2

    # Construct distance matrix
    dist_mat = np.zeros(shape = (n, n))

    # Create boolean function
    boo = lambda k: ((data[columns] == results.loc[k, columns]).prod(axis = 1)).
↪astype(bool)

    for i in range(n):
```

```
        for j in range(i):

            # Distance is pad minus ks-test p-value
            dist_mat[i, j] = pad - ks_2samp(data.loc[boo(i), 'consequence'],␣
↪data.loc[boo(j), 'consequence'])[1]

            # Must be symmetric matrix
            dist_mat[j, i] = dist_mat[i, j]

    # Specify criteria for hierarchical clustering
    clust_alg = AgglomerativeClustering(n_clusters = None,
                                        affinity = 'precomputed',
                                        linkage = 'complete',
                                        distance_threshold = pad - crit_val)

    # Perform hierarchical clustering using the distance matrix dist_mat
    clusters = clust_alg.fit(dist_mat)

    # Record the clusters
    results['cluster'] = clusters.labels_

    return results
```

5 Results

Since we constructed the data ourselves, it is clear the function it is working well.

[94]:
```
sort_data(data, ['prop-1', 'prop-2'], 0.10)
```

[94]:

	prop-1	prop-2	cluster
0	A	1	0
1	A	2	0
2	B	2	1
3	B	3	1
4	C	3	2