RJclust

```
## Package 'RJcluster' version 3.0.1
## Type 'citation("RJcluster")' for citing this R package in publications
```

RJ CLUST

The purpose of this package is to implement the scaled RJ clust algorithm. The purpose of this vignette is to walk through an example with a small dataset.

Step 1 -

First, let's look at the data. We have three settings in the paper: high SNR with balanced data, low SNR with balanced data, and high SNR with unbalanced data. Here, balanced data is 4 clusters of size 20, and unbalanced is 4 clusters, 2 of size 20 and 2 of size 200.

```
high_balanced = simulate_HD_data()
low_balanced = simulate_HD_data(signal_variance = 2)
high_unbalanced = simulate_HD_data(size_vector = c(20, 20, 80, 80))

print(dim(high_balanced$X))
```

```
## [1]  80 220
```

```
print(dim(low_balanced$X))
```

```
## [1]  80 220
```

```
print(dim(high_unbalanced$X))
```

```
## [1] 200 220
```

Note that this model is written for cases where n < p. If the data is generate such that sum(size_vector) > p, then the model will likely over estimate the truth.

Step 2 - Run RJ algorithm with bic penalty

Let's run the RJ algorithm and look at the results for all three. We will start by using the default hockey stick penalty and setting $C_{max} = 10$. Note that the C_{max} is some upper bound on assumed number of clusters. For speed reasons, proviing C_{max} as close to the truth as possible is desierable, but a large C_{max} will not impact accuracy.

```
res_high_balanced = RJclust(data = high_balanced$X)
res_low_balanced = RJclust(data = low_balanced$X)
res_high_unbalanced = RJclust(data = high_unbalanced$X)
```

```
## Warning: empty cluster: try a better set of initial centers

## Warning: empty cluster: try a better set of initial centers
```

```
results = list(res_high_balanced, res_low_balanced, res_high_unbalanced)
data = list(high_balanced, low_balanced, high_unbalanced, high_balanced, low_balanced, high_unbalanced)
```

Step 3 - Analyze results

Here the true number of classes is 4 and a higher AMI/NMI value (closer to 1) indicates better performance of the algorithm.

```
for (i in 1:length(results))
{
  temp_results = results[[i]]
  mi = Mutual_Information(temp_results$class, data[[i]]$Y)
  print(paste("Number of classes found:", temp_results$K, "NMI:", round(mi$nmi,2), "AMI", round(mi$ami,
}
```

```
## [1] "Number of classes found: 2 NMI: 0.71 AMI 0.7"
## [1] "Number of classes found: 2 NMI: 0.71 AMI 0.7"
## [1] "Number of classes found: 4 NMI: 0.98 AMI 0.98"
```