An R Package for Affinity Propagation Clustering

Ulrich Bodenhofer and Andreas Kothmeier

Institute of Bioinformatics, Johannes Kepler University Linz
Altenberger Str. 69, 4040 Linz, Austria
apcluster@bioinf.jku.at

Version 1.0.1, March 2, 2010

Scope and Purpose of this Document

This document is a user manual for the R package apcluster. It is only meant as a gentle introduction into how to use the basic functions implemented in this package. Not all features of the R package are described in full detail. Such details can be obtained from the documentation enclosed in the R package. Further note the following: (1) this is neither an introduction to affinity propagation nor to clustering in general; (2) this is not an introduction to R. If you lack the background for understanding this manual, you first have to read introductory literature on these subjects.

Contents

1 Introduction

Affinity propagation (AP) is a relatively new clustering algorithm that has been introduced by Brendan J. Frey and Delbert Dueck [1].[1] The authors themselves describe affinity propagation as follows:[2]

> "*An algorithm that identifies exemplars among data points and forms clusters of data points around these exemplars. It operates by simultaneously considering all data point as potential exemplars and exchanging messages between data points until a good set of exemplars and clusters emerges.*"

AP has been applied in various fields recently, among which bioinformatics is becoming increasingly important. Frey and Dueck have made their algorithm available as Matlab code.[1] Matlab, however, is relatively uncommon in bioinformatics. Instead, the statistical computing platform R has become a widely accepted standard in this field. In order to leverage affinity propagation for bioinformatics applications, we have implemented affinity propagation as an R package. Note, however, that the given package is in no way restricted to bioinformatics applications. It is as generally applicable as Frey's and Dueck's original Matlab code.[1]

2 Installation

2.1 Installation via CRAN

The R package `apcluster` (current version: 1.0.1) is part of the *Comprehensive R Archive Network (CRAN)*[3]. The simplest way to install the package, therefore, is to enter the following command into your R session:

```
> install.packages("apcluster")
```

2.2 Manual installation

If, for what reason ever, you prefer to install the package manually, download the package file suitable for your computer system and copy it to your harddisk. Open the package's page at CRAN[4] and the proceed as follows.

Manual installation under Windows

1. Download `apcluster_1.0.1.zip` and save it to your harddisk
2. Open the R GUI and select the menu entry

[1] `http://www.psi.toronto.edu/affinitypropagation/`

[2] quoted from `http://www.psi.toronto.edu/affinitypropagation/faq.html#def`

[3] `http://cran.r-project.org/`

[4] `http://cran.r-project.org/web/packages/apcluster/index.html`

```
Packages | Install package(s) from local zip files...
```

In the file dialog that opens, go to the folder where you placed `apcluster_1.0.1.zip` and select this file. The package should be installed now.

Manual installation under Linux/UNIX/MacOS

1. Download `apcluster_1.0.1.tar.gz` and save it to your harddisk.
2. Open a shell window and change to the directory where you put `apcluster_1.0.1.tar.gz`. Enter

   ```
   R CMD INSTALL apcluster_1.0.1.tar.gz
   ```

   to install the package.

2.3 Compatibility Issues

Both the Windows and the Linux/UNIX/MacOS version of the package have been built under R 2.10.1, but have been tested with earlier R versions too. Apart from installation warnings that the package has been built with a more recent version of R, it should work without severe problems on R versions $\geq$2.6.1.

3 Getting Started

To load the package, enter the following in your R session:

```
> library(apcluster)
```

If this command terminates without any error message or warning, you can be sure that the package has been installed successfully. If so, the package is ready for use now and you can start clustering your data with affinity propagation.

The package includes both a user manual (this document) and a reference manual (help pages for each function). To view the user manual, enter

```
> vignette("APCluster-Manual")
```

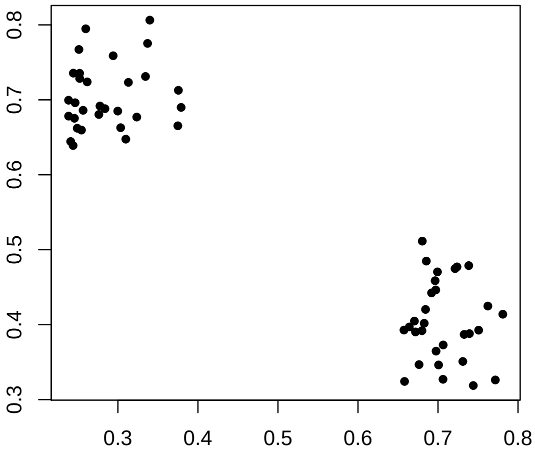
Help pages can be viewed using the `help` command. It is recommended to start with

```
> help(apcluster)
```

Affinity propagation does not require the data samples to be of any specific kind or structure. AP only requires a *similarity matrix*, i.e., given l data samples, this is an $l \times l$ real-valued matrix S, in which an entry S_{ij} corresponds to a value measuring how similar sample i is to sample j. AP does not require these values to be in a specific range. Values can be positive or negative. AP does not even require the similarity matrix to be symmetric (although, in most applications, it will be symmetric anyway). A value of $-\infty$ is interpreted as "absolute dissimilarity". The higher a value, the more similar two samples are considered.

To get a first impression, let us create a random data set in $\mathbb{R}^2$ as the union of two "Gaussian clouds":

```
> cl1 <- cbind(rnorm(30, 0.3, 0.05), rnorm(30, 0.7, 0.04))
> cl2 <- cbind(rnorm(30, 0.7, 0.04), rnorm(30, 0.4, 0.05))
> x <- rbind(cl1, cl2)
> plot(x, xlab = "", ylab = "", pch = 19, cex = 0.8)
```



Now we have to create a similarity matrix. The package `apcluster` offers several different ways for doing that (see Section 5 below). Let us start with the default similarity measure used in the papers of Frey and Dueck — negative squared distances:

```
> s <- negDistMat(x, r = 2)
```

We are now ready to run affinity propagation:

```
> apres <- apcluster(s)
```

The function `apcluster` creates an object belonging to the S4 class `APResult` that is defined by the present package. To get detailed information on which data are stored in such objects, enter

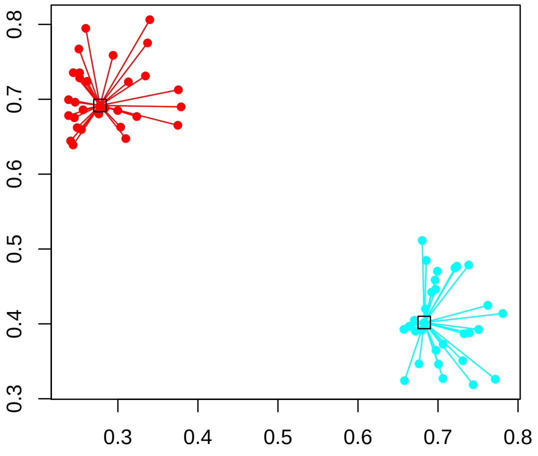
```
> help(APResult)
```

The simplest thing we can do with the output is to enter the name of the object (which implicitly calls `show`) to get a summary of the clustering result:

```
> apres

APResult object

Number of samples     =  60
Number of iterations =  132
Input preference      =  -0.1555249
Sum of similarities  =  -0.2556224
Sum of preferences    =  -0.3110499
Net similarity        =  -0.5666723
Number of exemplars  =  2

Exemplars:
   28 32
Clusters:
   Cluster 1, exemplar 28:
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
      27 28 29 30
   Cluster 2, exemplar 32:
      31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
      54 55 56 57 58 59 60
```

For two-dimensional data sets, `apcluster` allows for plotting the original data set along with a clustering result:

```
> plot(apres, x)
```



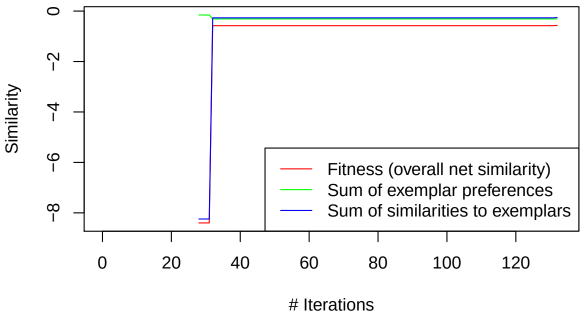
In this plot, each color corresponds to one cluster. The exemplar of each cluster is marked by a box and all cluster members are connected to their exemplars with lines.

Suppose we want to have better insight into what the algorithm did in each iteration. For this purpose, we can supply the option `details=TRUE` to `apcluster`:

```
> apres <- apcluster(s, details = TRUE)
```

This option tells the algorithm to keep a detailed log about its progress. For example, this allows us to plot the three performance measures that AP uses internally for each iteration:

```
> plot(apres)
```



These performance measures are:

1. Sum of exemplar preferences
2. Sum of similarities of exemplars to their cluster members
3. Net fitness: sum of the two former

For details, the user is referred to the original affinity propagation paper [1] and the supplementary material published on the affinity propagation Web page.[1] We see from the above plot that the algorithm has not made any change for the last 100 (of 132!) iterations. AP, through its parameter `convits`, allows to control for how long AP waits for a change until it terminates (the default is `convits=100`). If the user has the feeling that AP will probably converge quicker on his/her data set, a lower value can be used:

```
> apres <- apcluster(s, convits = 15, details = TRUE)
> apres

APResult object

Number of samples    =  60
Number of iterations =  47
```

```
Input preference     =  -0.1555249
Sum of similarities  =  -0.2556224
Sum of preferences   =  -0.3110499
Net similarity       =  -0.5666723
Number of exemplars  =  2

Exemplars:
   28 32
Clusters:
   Cluster 1, exemplar 28:
      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
      27 28 29 30
   Cluster 2, exemplar 32:
      31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
      54 55 56 57 58 59 60
```

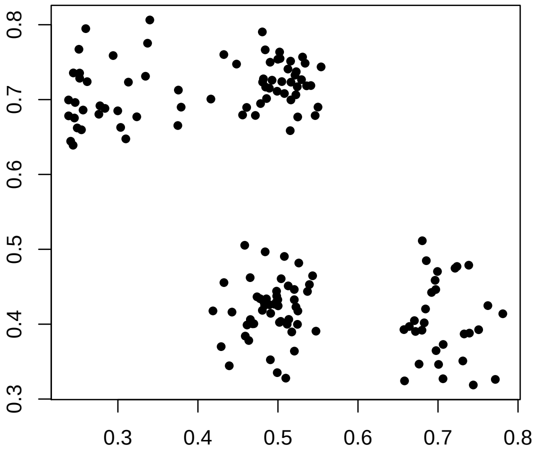
4 Adjusting Input Preferences

Apart from the similarity itself, the most important input parameter of AP is the so-called *input preference* which can be interpreted as the tendency of a data sample to become an exemplar (see [1] and supplementary material on the AP homepage[1] for a more detailed explanation). This input preference can either be chosen individually for each data sample or it can be a single value shared among all data samples. Input preferences largely determine the number of clusters, in other words, how fine- or coarse-grained the clustering result will be.

The input preferences one can specify for AP are roughly in the same range as the similarity values, but they do not have a straightforward interpretation. Frey and Dueck have introduced the following rule of thumb: "*The shared value could be the median of the input similarities (resulting in a moderate number of clusters) or their minimum (resulting in a small number of clusters).*" [1]

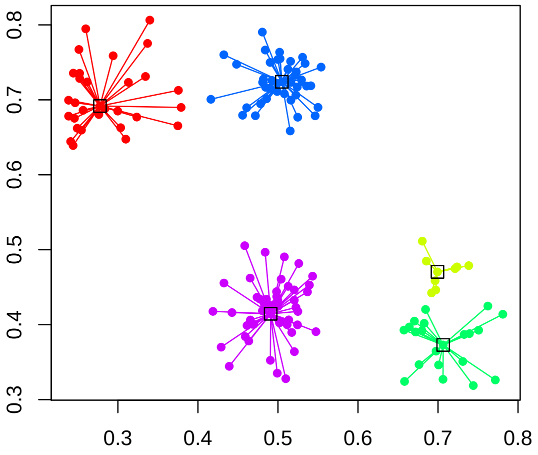
Our AP implementation uses the median rule by default if the user does not supply a custom value for the input preferences. In order to provide the user with a knob that is — at least to some extent — interpretable, the function `apcluster` has a new argument `q` that allows to set the input preference to a certain quantile of the input similarities: resulting in the median for `q=0.5` and in the minimum for `q=0`. As an example, let us add two more "clouds" to the data set from above:

```
> cl3 <- cbind(rnorm(20, 0.5, 0.03), rnorm(20, 0.72, 0.03))
> cl4 <- cbind(rnorm(25, 0.5, 0.03), rnorm(25, 0.42, 0.04))
> x <- rbind(x, cl3, cl4)
> s <- negDistMat(x, r = 2)
> plot(x, xlab = "", ylab = "", pch = 19, cex = 0.8)
```



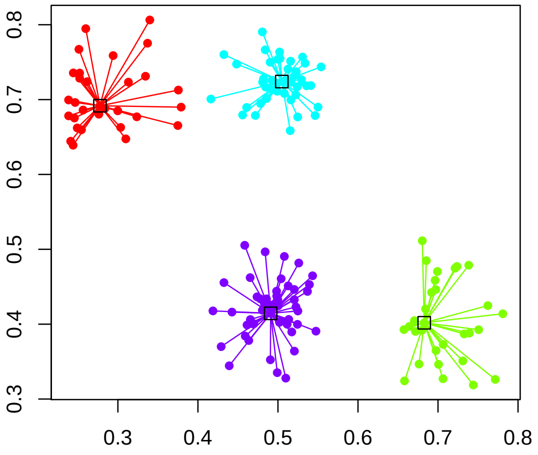
For the default setting, we obtain the following result:

```
> apres <- apcluster(s)
> plot(apres, x)
```



For the minimum of input similarities, we obtain the following result:

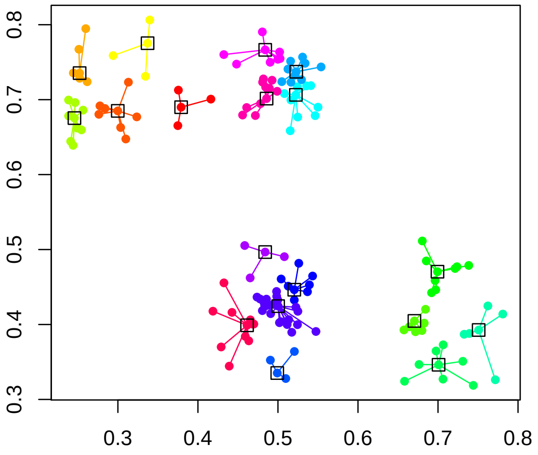
```
> apres <- apcluster(s, q = 0)
> plot(apres, x)
```



So we see that AP is quite robust against a reduction of input preferences in this example which may be caused by the clear separation of the four clusters. If we increase input preferences, however, we can force AP to split the four clusters into smaller sub-clusters:

```
> apres <- apcluster(s, q = 0.8)
```

```
> plot(apres, x)
```



Note that the input preference used by AP can be recovered from the output object (no matter which method to adjust input preferences has been used). On the one hand, the value is printed if the object is displayed (by `show` or by entering the output object's name). On the other hand, the value can be accessed directly via the slot `p`:

```
> apres@p
```

```
[1] -0.008168459
```

The above example with `q=0` demonstrates that setting input preferences to the minimum of input similarities does not necessarily result in a very small number of clusters (like one or two). This is due to the fact that input preferences need not necessarily be exactly in the range of the similarities. To determine a meaningful range, an auxiliary function is available which, in line with Frey's and Dueck's Matlab code,[1] allows to compute a minimum value (for which one or at most two clusters would be obtained) and a maximum value (for which as many clusters as data samples would be obtained):

```
> preferenceRange(s)
```

```
[1] -6.964980e+00 -2.532793e-06
```

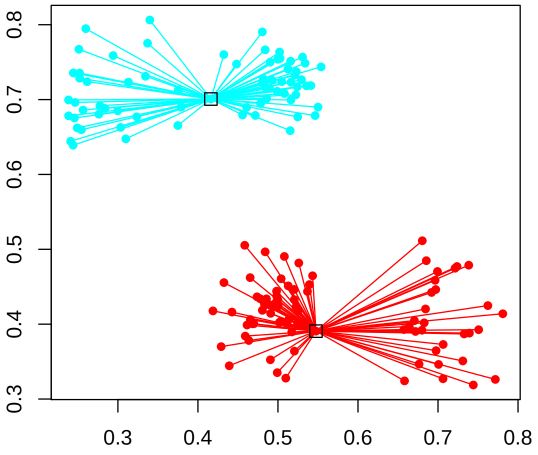
The function returns a two-element vector with the minimum value as first and the maximum value as second entry. The computations are done approximately by default. If one is interested in exact bounds, supply `exact=TRUE` (resulting in longer computation times).

Many clustering algorithms need to know a pre-defined number of clusters. This is often a major nuisance, since the exact number of clusters is hard to know for non-trivial (in particular, high-dimensional) data sets. AP avoids this problem. If, however, one still wants to require a fixed number of clusters, this has to be accomplished by a search algorithm that adjusts input preferences in order to produce the desired number of clusters in the end. For convenience, this search algorithm is available as a function `apclusterK` (analogous to Frey's and Dueck's Matlab implementation[1]). We can use this function to force AP to produce only two clusters (merging the two pairs of adjacent clouds into one cluster each):

```
> apres <- apclusterK(s, 2)
```

```
Trying p = -0.00696751
   Number of clusters: 18
Trying p = -0.0696523
   Number of clusters: 5
Trying p = -0.6965002
   Number of clusters: 4
Trying p = -3.482491 (bisection step no. 1)
   Number of clusters: 2

Number of clusters: 2  for p =  -3.482491
```

```
> plot(apres, x)
```



5 Similarity Matrices

Apart from the obvious monotonicity “the higher the value, the more similar two samples”, affinity propagation does not make any specific assumption about the similarity measure. Negative squared distances must be used if one wants to minimize squared errors [1]. Apart from that, the choice and implementation of the similarity measure is left to the user.

Our package offers a few more methods to obtain similarity matrices. The choice of the right one (and, consequently, the objective function the algorithm optimizes), still has to be made by the user.

5.1 The function `negDistMat`

The function `negDistMat`, in line with Frey and Dueck, allows to compute negative distances for a given set of real-valued data samples. Above we have used the following:

```
> s <- negDistMat(x, r = 2)
```

This computes a matrix of negative squared distances from the data matrix `x`. The function `negDistMat` is a simple wrapper around the standard function `dist`, hence, it allows for a lot more different similarity measures. The user can make use of all variants implemented in `dist` by using the options `method` (selects a distance measure) and `p` (specifies the exponent for the Minkowski distance, otherwise it is void) that are passed on to `dist`. Presently, `dist` provides the following variants of computing the distance $d(\mathbf{x},\mathbf{y})$ of two data samples $\mathbf{x} = (x_1,\ldots,x_d)$ and $\mathbf{y} = (y_1,\ldots,y_d)$:

Euclidean:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

use `method="euclidean"` or do not specify argument `method` (since this is the default);

Maximum:

$$d(\mathbf{x},\mathbf{y}) = \max_{i=1}^{d} |x_i - y_i|$$

use `method="maximum"`;

Sum of absolute distances / Manhattan:

$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|$$

use `method="manhattan"`;

Canberra:

$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} \frac{|x_i - y_i|}{|x_i + y_i|}$$

summands with zero denominators are not taken into account; use `method="canberra"`;

Minkowski:

$$d(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d}(x_i - y_i)^p\right)^{\frac{1}{p}}$$

use `method="minkowski"` and specify p using the additional argument `p` (default is `p=2`, resulting in the standard Euclidean distance);

We do not consider `method="binary"` here, since it is irrelevant for real-valued data.

The function `negDistMat` takes the distances computed with one of the variants listed above and returns -1 times the r-th power of it, i.e.,

$$s(\mathbf{x},\mathbf{y}) = -d(\mathbf{x},\mathbf{y})^r. \tag{1}$$

The exponent r can be adjusted with the argument `r`. The default is `r=1`, hence, one has to supply `r=2` as in the above example to obtain squared distances.

Here are some examples. We use the corners of the two-dimensional unit square and its middle point $(\frac{1}{2}, \frac{1}{2})$ as sample data:

```
> ex <- matrix(c(0, 0, 1, 0, 0.5, 0.5, 0, 1, 1, 1), 5, 2, byrow = TRUE)
> ex
```

```
     [,1] [,2]
[1,]  0.0  0.0
[2,]  1.0  0.0
[3,]  0.5  0.5
[4,]  0.0  1.0
[5,]  1.0  1.0
```

Standard Euclidean distance:

```
> negDistMat(ex)
```

```
           1          2          3          4          5
1  0.0000000 -1.0000000 -0.7071068 -1.0000000 -1.4142136
2 -1.0000000  0.0000000 -0.7071068 -1.4142136 -1.0000000
3 -0.7071068 -0.7071068  0.0000000 -0.7071068 -0.7071068
4 -1.0000000 -1.4142136 -0.7071068  0.0000000 -1.0000000
5 -1.4142136 -1.0000000 -0.7071068 -1.0000000  0.0000000
```

Squared Euclidean distance:

```
> negDistMat(ex, r = 2)
```

```
     1    2    3    4    5
1  0.0 -1.0 -0.5 -1.0 -2.0
2 -1.0  0.0 -0.5 -2.0 -1.0
3 -0.5 -0.5  0.0 -0.5 -0.5
4 -1.0 -2.0 -0.5  0.0 -1.0
5 -2.0 -1.0 -0.5 -1.0  0.0
```

Maximum norm-based distance:

```
> negDistMat(ex, method = "maximum")
```

```
     1    2    3    4    5
1  0.0 -1.0 -0.5 -1.0 -1.0
2 -1.0  0.0 -0.5 -1.0 -1.0
3 -0.5 -0.5  0.0 -0.5 -0.5
4 -1.0 -1.0 -0.5  0.0 -1.0
5 -1.0 -1.0 -0.5 -1.0  0.0
```

Sum of absolute distances (aka Manhattan distance):

```
> negDistMat(ex, method = "manhattan")
```

```
   1  2  3  4  5
1  0 -1 -1 -1 -2
2 -1  0 -1 -2 -1
3 -1 -1  0 -1 -1
4 -1 -2 -1  0 -1
5 -2 -1 -1 -1  0
```

Canberra distance:

```
> negDistMat(ex, method = "canberra")
```

```
   1         2          3         4          5
1  0 -2.000000 -2.0000000 -2.000000 -2.0000000
2 -2  0.000000 -1.3333333 -2.000000 -1.0000000
3 -2 -1.333333  0.0000000 -1.333333 -0.6666667
4 -2 -2.000000 -1.3333333  0.000000 -1.0000000
5 -2 -1.000000 -0.6666667 -1.000000  0.0000000
```

Minkowski distance for $p = 3$ (3-norm):

```
> negDistMat(ex, method = "minkowski", p = 3)
```

```
           1          2          3          4          5
1  0.0000000 -1.0000000 -0.6299605 -1.0000000 -1.2599210
2 -1.0000000  0.0000000 -0.6299605 -1.2599210 -1.0000000
3 -0.6299605 -0.6299605  0.0000000 -0.6299605 -0.6299605
4 -1.0000000 -1.2599210 -0.6299605  0.0000000 -1.0000000
5 -1.2599210 -1.0000000 -0.6299605 -1.0000000  0.0000000
```

5.2 Other similarity measures

The package apcluster offers three more functions for creating similarity matrices for real-valued data:

Exponential transformation of distances: the function expSimMat is another wrapper around the standard function dist. The difference is that, instead of the transformation (1), it uses the following transformation:

$$s(\mathbf{x},\mathbf{y}) = \exp\left(-\left(\frac{d(\mathbf{x},\mathbf{y})}{w}\right)^r\right)$$

Here the default is r=2. It is clear that r=2 in conjunction with method="euclidean" results in the well-known *Gaussian kernel / RBF kernel* [2, 3, 4], whereas r=1 in conjunction with method="euclidean" results in the similarity measure that is sometimes called *Laplace kernel* [2, 3]. Both variants (for non-Euclidean distances as well) can also be interpreted as *fuzzy equality/similarity relations* [5].

Linear scaling of distances with truncation: the function linSimMat uses the transformation

$$s(\mathbf{x},\mathbf{y}) = \max\left(1 - \frac{d(\mathbf{x},\mathbf{y})}{w}, 0\right)$$

which is also often interpreted as a *fuzzy equality/similarity relation* [5].

Linear kernel: scalar products can also be interpreted as similarity measures, a view that is often adopted by kernel methods in machine learning. In order to provide the user with this option as well, the function `linKernel` is available. For two data samples $\mathbf{x} = (x_1, \dots, x_d)$ and $\mathbf{y} = (y_1, \dots, y_d)$, it computes the similarity as

$$s(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} x_i \cdot y_i.$$

The function has one additional argument, `normalize` (by default `FALSE`). If `normalize=TRUE`, values are normalized to the range $[-1, +1]$ in the following way:

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\left(\sum_{i=1}^{d} x_i^2\right) \cdot \left(\sum_{i=1}^{d} y_i^2\right)}}$$

Entries for which at least one of the two factors in the denominator is zero are set to zero (however, the user should be aware that this should be avoided anyway).

For the same example data as above, we obtain the following for the RBF kernel:

```
> expSimMat(ex)

          1         2         3         4         5
1 1.0000000 0.3678794 0.6065307 0.3678794 0.1353353
2 0.3678794 1.0000000 0.6065307 0.1353353 0.3678794
3 0.6065307 0.6065307 1.0000000 0.6065307 0.6065307
4 0.3678794 0.1353353 0.6065307 1.0000000 0.3678794
5 0.1353353 0.3678794 0.6065307 0.3678794 1.0000000
```

Laplace kernel:

```
> expSimMat(ex, r = 1)

          1         2         3         4         5
1 1.0000000 0.3678794 0.4930687 0.3678794 0.2431167
2 0.3678794 1.0000000 0.4930687 0.2431167 0.3678794
3 0.4930687 0.4930687 1.0000000 0.4930687 0.4930687
4 0.3678794 0.2431167 0.4930687 1.0000000 0.3678794
5 0.2431167 0.3678794 0.4930687 0.3678794 1.0000000
```

Linear scaling of distances with truncation:

```
> linSimMat(ex, w = 1.2)

          1         2         3         4         5
1 1.0000000 0.1666667 0.4107443 0.1666667 0.0000000
2 0.1666667 1.0000000 0.4107443 0.0000000 0.1666667
3 0.4107443 0.4107443 1.0000000 0.4107443 0.4107443
4 0.1666667 0.0000000 0.4107443 1.0000000 0.1666667
5 0.0000000 0.1666667 0.4107443 0.1666667 1.0000000
```

Linear kernel (we exclude $(0, 0)$):

```
> linKernel(ex[2:5, ])

     [,1] [,2] [,3] [,4]
[1,]  1.0  0.5  0.0    1
[2,]  0.5  0.5  0.5    1
[3,]  0.0  0.5  1.0    1
[4,]  1.0  1.0  1.0    2
```

Normalized linear kernel (we exclude $(0, 0)$):

```
> linKernel(ex[2:5, ], normalize = TRUE)

          [,1]      [,2]      [,3]      [,4]
[1,] 1.0000000 0.7071068 0.0000000 0.7071068
[2,] 0.7071068 1.0000000 0.7071068 1.0000000
[3,] 0.0000000 0.7071068 1.0000000 0.7071068
[4,] 0.7071068 1.0000000 0.7071068 1.0000000
```

6 Known Issues / Future Extensions

- We currently have no implementation that exploits sparsity of similarity matrices. The implementation of *sparse AP* and *leveraged AP* which are available as Matlab code from the AP Web page[1] is left for future extensions of the package. Presently, we only offer a function `sparseToFull` that converts similarity matrices from sparse format into a full $l \times l$ matrix.
- We are aware that our R implementation is often slower than Frey's and Dueck's Matlab code.[1] We have no sufficiently convincing explanation for this at the moment.

7 How to Cite This Package

If you use this package for research that is published later, you are kindly asked to cite it as follows:

> U. Bodenhofer and A. Kothmeier (2010). An R package for affinity propagation clustering. R package version 1.0.1. Institute of Bioinformatics, Johannes Kepler University, Linz, Austria.

Moreover, we insist that, any time you cite the package, you also cite the original paper in which affinity propagation has been introduced [1].

To obtain BibTEX entries of the two references, you can enter the following into your R session:

```
> toBibtex(citation("apcluster"))
```

References

[1] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[2] C. H. FitzGerald, C. A. Micchelli, and A. Pinkus. Functions that preserve families of positive semidefinite matrices. *Linear Alg. Appl.*, 221:83–102, 1995.

[3] C. A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constr. Approx.*, 2:11–22, 1986.

[4] B. Schölkopf and A. J. Smola. *Learning with Kernels*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2002.

[5] B. De Baets and R. Mesiar. Metrics and T-equalities. *J. Math. Anal. Appl.*, 267:531–547, 2002.