DistanceLearning

Carl Tony Fakhry, Ping Chen, Rahul Kulkarni and Kourosh Zarringhalam

2018-04-01

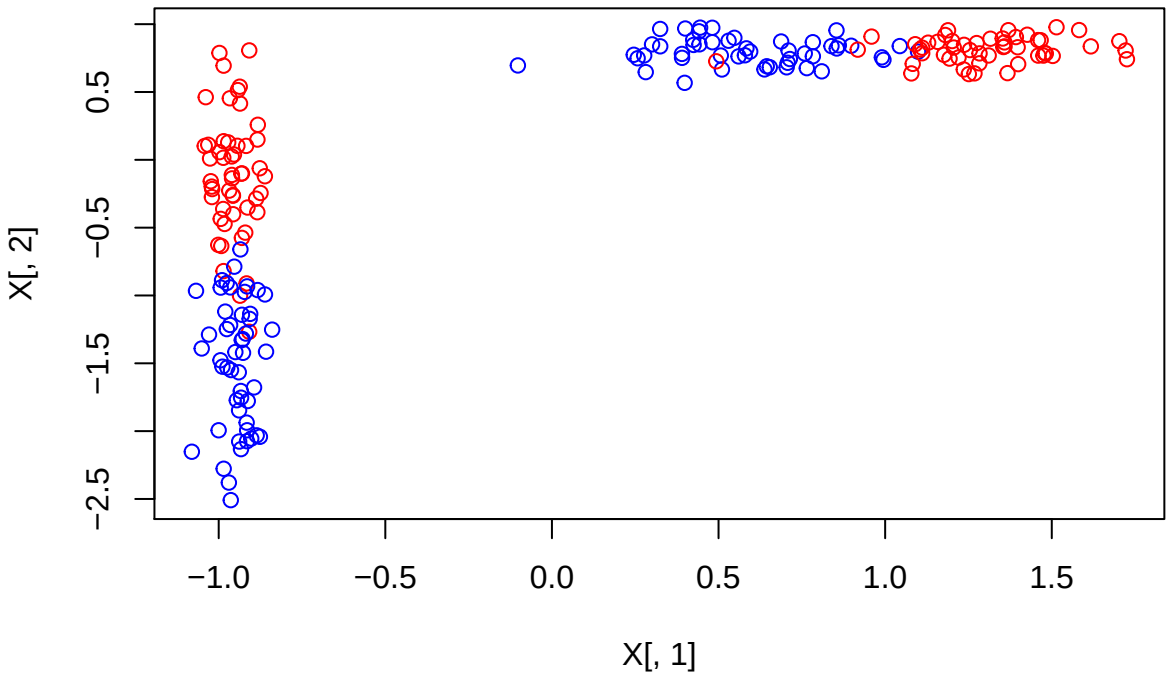
The main objective of distance metric learning is to "optimally" transform the data in order to bring similar points closer to each other while keeping the distance between dissimilar points away from zero. **DistanceLearning** is an R package which implements a comprehensive set (11 methods) of state of the art distance metric learning methods. We will describe our method, Free Energy Metric Learning (FENN) [1], with some mathematical detail, and we will show how it can used with sampled data. For other methods, we will illustrate how they can be called with our package while referring the reader to the original papers for a detailed presentation.

Free Energy Nearest Neighbor

```
library(DistanceLearning)

# Get example data
example_data = system.file("extdata", "example_data.csv", package="DistanceLearning")
df = read.csv(example_data)
Y = as.vector(df$y)
X = as.matrix(df[,2:ncol(df)])

# Plot the data
plot(X[,1], X[,2], col = c("blue","red")[Y+1])
```

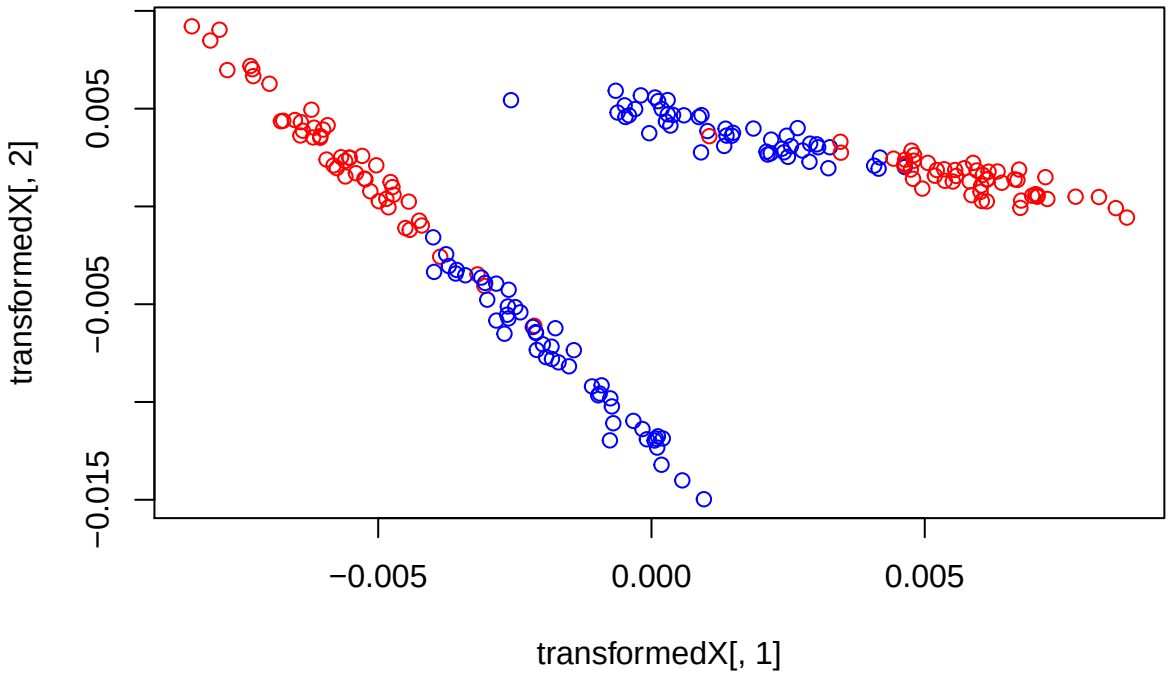


```r
# Learn the FENN metric
result <- FENN(Y, X, method = "CV")

# Get the metric
FENNMetric <- result$FENNTransform

# Get the transformed data using
transformedX <- result$transformedX
# Or using
transformedX <- X %*% FENNMetric

# Plot the transformed data
plot(transformedX[,1], transformedX[,2], col = c("blue","red")[Y+1])
```



`method` is the method for estimating the smoothing parameter This can either be set to `"CV"` or `"fisher.information"`. The default value is `method = "CV"` in which case the smoothing parameter is determined using 10-fold cross-validation. The possible values for the smoothing parameter are generated in a window around the value that maximizes the fisher information. If `method = "fisher.information"` then the smoothing parameter is selected according to the value that maximizes the fisher information. `CV_window` is the window around the value of the smoothing parameter that maximizes the fisher information. The default value is set to `CV_window = NA` in which case the value `CV_window = 4*ncol(X)` is set internally. If `method = "fisher.information"` then the value of this parameter is not considered when computing the FENN metric. `dimension_reduction` is a boolean value on whether to learn the transformation matrix for the optimal dimension of the reduced space. The default value is `dimension_reduction = FALSE`. `threshold` is the threshold cutoff on the sum of free energies when performing dimension reduction. The default value is `threshold = 0.99`. `K` is the number of neighbors to be used for K-NN classification during the CV procedure when computing the optimal tuning parameter. The user is advised to set this value to the number of neighbors that will be used in classification in the transformed space after the FENN metric is applied to the original data. Default value is set to `K = 5`.

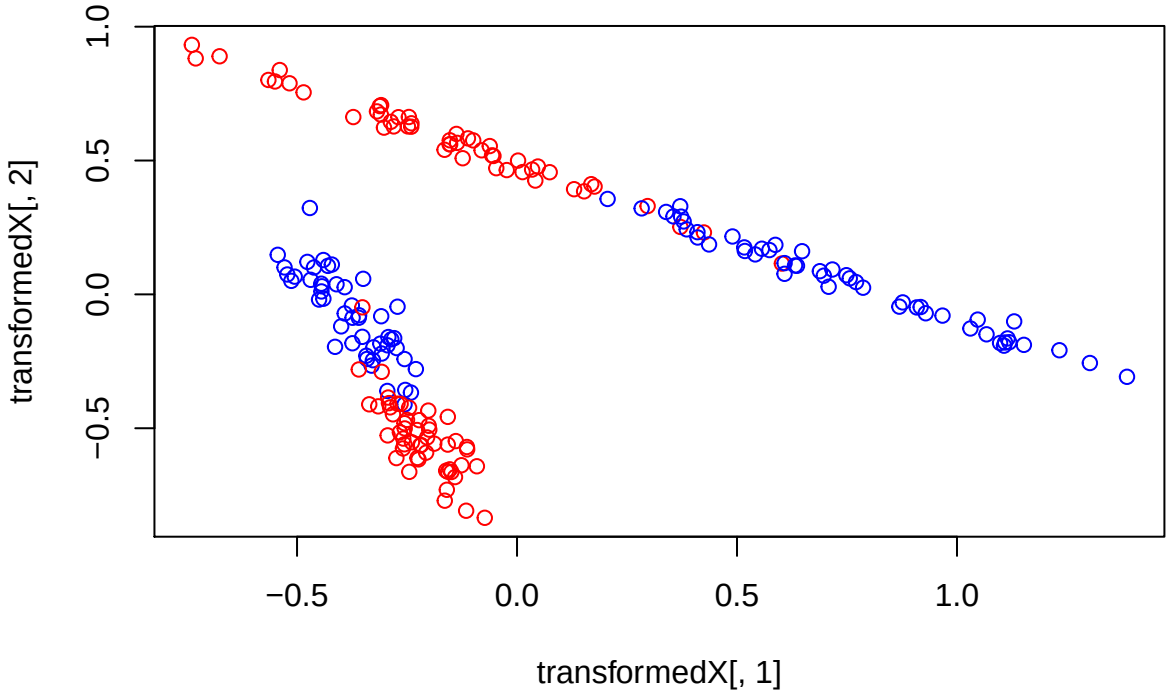
Xing's Method

Xing's method [2] proposes a different solution to the optimization problem in (1). However, Xing's method does not have a closed form solution and requires a gradient based algorithm for solving the optimization problem. Xing's method can be called using the following:

```
# Learn the metric, and get the transformed data
result <- XingMethod(Y, X, S = NULL, D = NULL, learning_rate = 0.1,
                     error = 1e-10, epsilon = 0.01, max_iterations = 1000)
# Get the metric
XingMetric <- result$XingTransform

# Get the transformed data using
transformedX <- result$transformedX
# Or using
transformedX <- X %*% XingMetric

# Plot the transformed data
plot(transformedX[,1], transformedX[,2], col = c("blue","red")[Y+1])
```



The parameters `learning_rate`, `error` and `epsilon` have values strictly between 0 and 1. The `learning_rate` parameter is the learning rate of the gradient based algorithm. `epsilon` is the threshold for convergence of the gradient based algorithm. `error` is a threshold which is used when projecting onto the constraint set. The parameter's default values are those which were set in the original author's code in [2]. In case the data set is large, one may approximate the similarity matrix `S` and the dissimilarity matrix `D`. The default value for both matrices is `NULL` in which case the full similarity and dissimilarity matrices are computed. Otherwise, both `S` and `D` must be of dimension `n x 2` where `n` is the number of sampled pairs for each. The pairs in `S` are indices of data points in the same class whereas the pairs in `D` are indices of data points with a different class label.

DANN and iDANN

DANN and iDANN are local distance metric learning methods which were introduced in [3]. Since both DANN and iDANN are local distance learning method, it does not return a single transformation matrix for the entire dataset. The user has to provide a training data set just as before, but the user also has to provide a new test set for which predictions must be made. DANN can be called in the following manner:

```
X2 = scale(X) #scale X as suggested by the authors in [3]
sample_points <- sample(1:nrow(X2), 40, replace = FALSE)
newX <- X2[sample_points,]
subY <- Y[sample_points]

# Predict class labels for newX
Yhat <- DANN(Y[-sample_points], X2[-sample_points,], newX = newX,
             K = 5, K_M = NULL, epsilon = 1)

# Get the accuracy
Accuracy <- length(which(Yhat == subY))/length(subY)
Accuracy
```

```
## [1] 0.975
```

The parameters `K`, `K_M` and `epsilon` set to the default values as suggested by the authors in [3]. `K_M` is the number of neighbors to be used in learning the DANN metric for each data point in `newX`. The default value is `K_M = NULL` in which case `K_M` will be set to `K_M = max(50, floor(nrow(X)/5))` internally by default. `K` is the number of neighbors to be used for K-NN classification after the DANN metric has been learned. The Default value is set to `K = 5`. Finally, `epsilon` is a tuning parameter. For more details about the parameters we refer the reader to [3]. Similarly iDANN can be called in the following manner:

```
# Predict class labels for newX
Yhat <- iDANN(Y[-sample_points], X2[-sample_points,], newX = newX, n_iterations = 5,
              K = 5, K_M = NULL, epsilon = 1)

# Get the accuracy
Accuracy <- length(which(Yhat == subY))/length(subY)
Accuracy
```

```
## [1] 0.975
```

Since iDANN is just an iterative form of DANN, `n_iterations` is how many iterations of DANN to compute. The default value is `n_iterations = 5`.

ADAMENN and iADAMENN

ADAMENN and iADAMENN are local distance metric learning methods which were introduced in [4]. Similar to DANN, ADAMENN is a local distance learning method and thus it does not return a single transformation matrix for the entire dataset. The user has to provide a training data set just as before, but the user also has to provide a new test set for which predictions must be made. ADAMENN can be called in the following manner:

```r
# Predict class labels for newX
Yhat <- ADAMENN(Y[-sample_points], X2[-sample_points,], newX = newX,
                K = 5, K_0 = NULL, K_1 = 3, K_2 = NULL, L = NULL,
                c = 5)

# Get the accuracy
Accuracy <- length(which(Yhat == subY))/length(subY)
Accuracy
```

```
## [1] 0.975
```

The parameters `K`, `K_0`, `K_1` and `K_2` are set to the default values as suggested by the authors in [4]. `K` is the number of neighbors to be used for K-NN classification after the ADAMENN metric has been learned. The default value is set to `K = 3`. `K_0` is the number of neighbors to be used in learning the local measure of feature relevance for each data point in `newX`. The default value is `K_0 = NULL`, in which case `K_0` will be set to `K_0 = max(floor(0.1*length(Y)), 20)`. `K_1` is the number of neighbors to be used in estimating the posterior class probability of each point in `newX`. The default value is `K_1 = 3`. `K_2` is the number of neighbors to be used in estimating the posterior class probability conditioned on the value of a feature. The default value is `K_2 = NULL`, in which case `K_2` will be set to `K_2 = max(floor(0.15*length(Y)), 2)`. `K_2` must be greater than `K_1`. `L` is the number of neighbors of a data point in `newX` to be used along each feature. The default value is `L = NULL`, in which case `L` will be set to `L = floor(K_2/2)`. `L` must be smaller or equal to `K_2`. `c` is a non-negative tuning paramter which scales the ADAMENN metric. The default value is set to `c = 5`. iADAMENN can be called in the following manner:

```r
# Predict class labels for newX
Yhat <- iADAMENN(Y[-sample_points], X2[-sample_points,], newX = newX, n_iterations = 5,
                K = 5, K_0 = NULL, K_1 = 3, K_2 = NULL, L = NULL, c = 5)

# Get the accuracy
Accuracy <- length(which(Yhat == subY))/length(subY)
Accuracy
```

```
## [1] 0.975
```

Since iADAMENN is just an iterative form of ADAMENN, `n_iterations` how many iterations of ADAMENN to compute. The default value is `n_iterations = 5`.

NCA

NCA is a global distance metric learning method which was introduced in [5]. NCA can be called in the following manner:

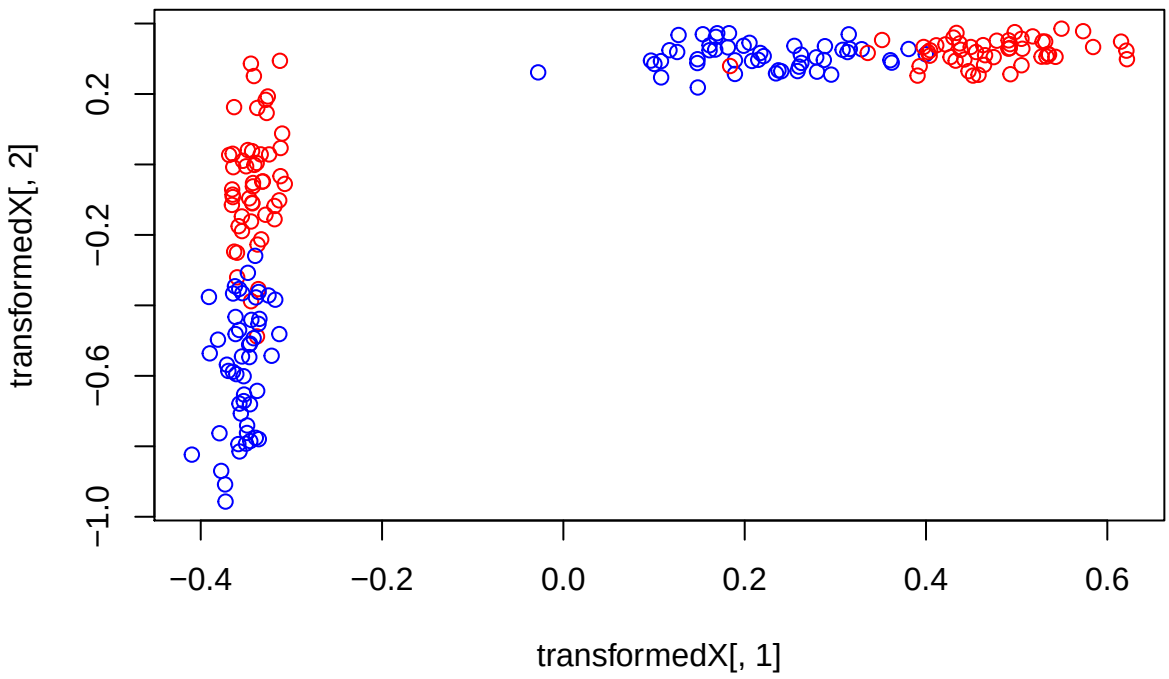
```r
# Learn the metric, and get the transformed data
result <- NCA(Y, X, max_iterations = 100, learning_rate = 0.01)

# Get the metric
NCAMetric <- result$NCATransform

# Get the transformed data using
transformedX <- result$transformedX
# Or using
```

```r
transformedX <- X %*% NCAMetric

# Plot the transformed data
plot(transformedX[,1], transformedX[,2], col = c("blue","red")[Y+1])
```



`max_iterations` is the maximum iterations to use in the solver when learning the transformation matrix. The default value is set to `max_iterations = 100`. `learning_rate` is the learning rate to be used in the solver when learning the transformation matrix. The default value is set to `learning_rate = 0.01`.

LFDA

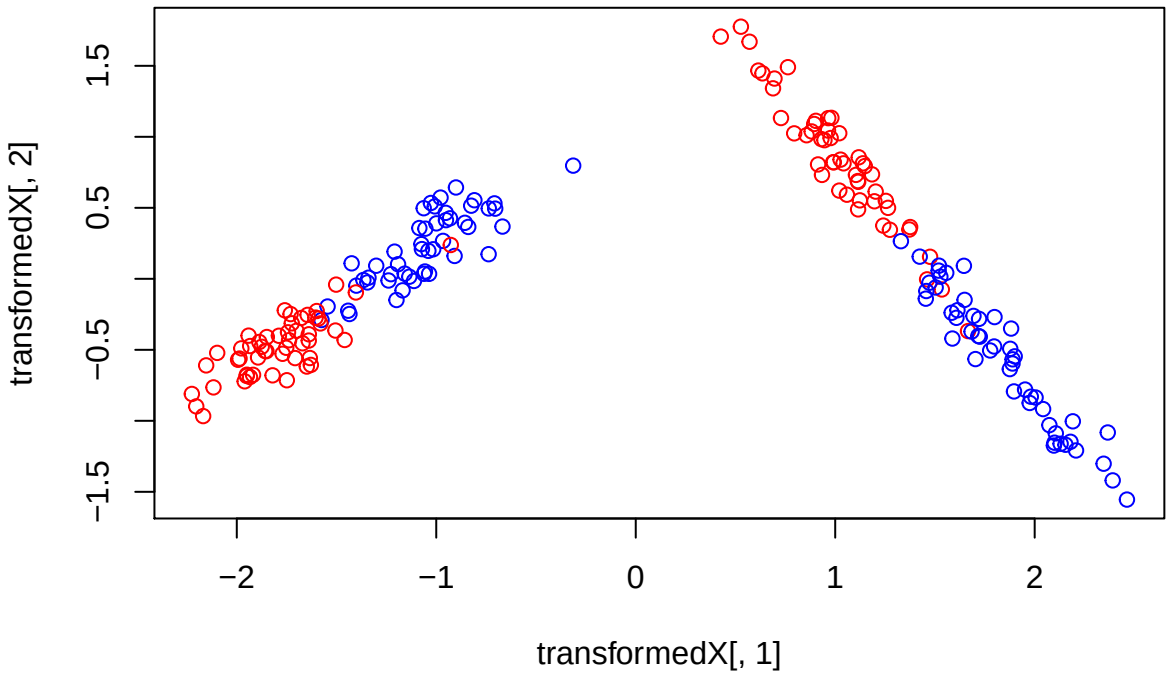
LFDA is a global distance metric learning which were introduced in [6][7]. LFDA can be called in the following manner:

```r
# Learn the metric, and get the transformed data
result <- LFDA(Y, X, metric = "plain", total_dims = NULL)

# Get the metric
LFDAMetric <- result$LFDATransform

# Get the transformed data using
transformedX <- result$transformedX
# Or using
transformedX <- X %*% LFDAMetric

# Plot the transformed data
plot(transformedX[,1], transformedX[,2], col = c("blue","red")[Y+1])
```



The possible options for `metric` are: `plain`, `weighted` or `orthogonalized`. The default value is `metric = plain`. `total_dims` is the number of dimensions on which the data should be projected on after the LFDA metric has been learned. The default value is set `total_dims = NULL` in which case the value is set to`total_dims = ncol(X)`.

RCA

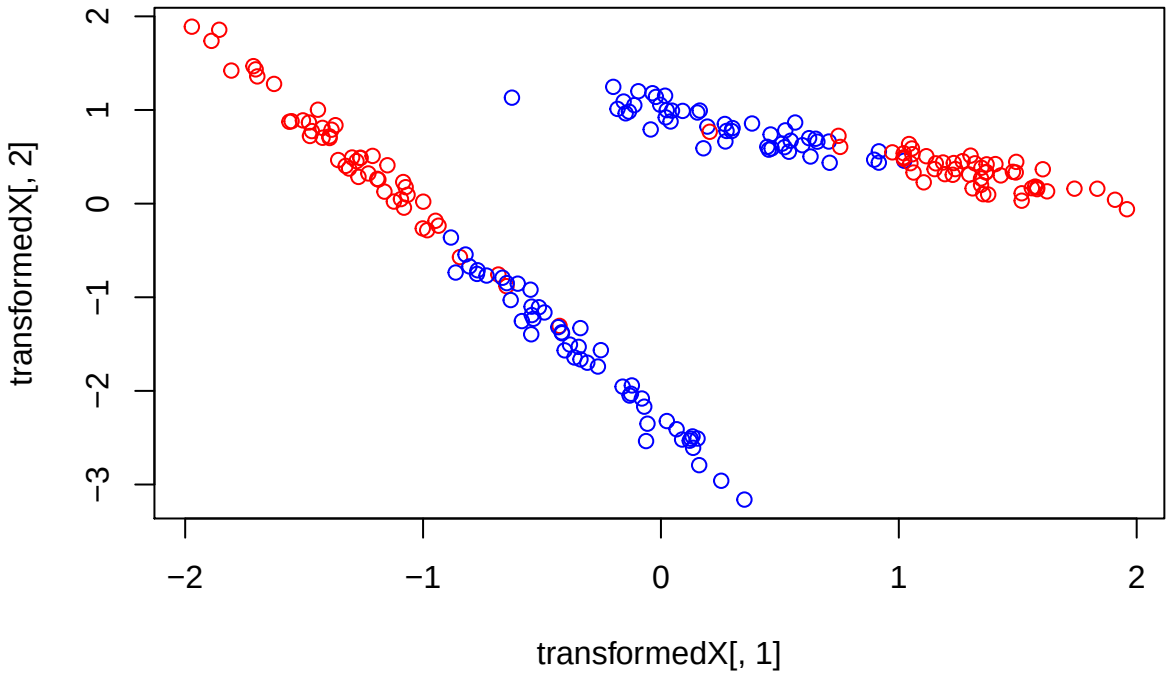
RCA is a global distance metric learning which were introduced in [9][10]. RCA can be called in the following manner:

```
# Learn the metric, and get the transformed data
result <- RCA(X, chunklets = Y, total_dims = NULL)

# Get the metric
RCAMetric <- result$RCATransform

# Get the transformed data using
transformedX <- result$transformedX
# Or using
transformedX <- X %*% RCAMetric

# Plot the transformed data
plot(transformedX[,1], transformedX[,2], col = c("blue","red")[Y+1])
```



`chunklets` is an integer vector indicating to which chunklet each data point belongs to. If a data point does not belong to a chunklet, then its chunklet value should be set to -1 otherwise all chunklets should be represented by a non-negative integer. All data points in the same chunklet must have the same class label. `total_dims` is the number of dimensions on which the data should be projected on after the RCA metric has been learned. The default value is set to `total_dims = NULL` in which case the value `total_dims = ncol(X)` is used as default.

DCA

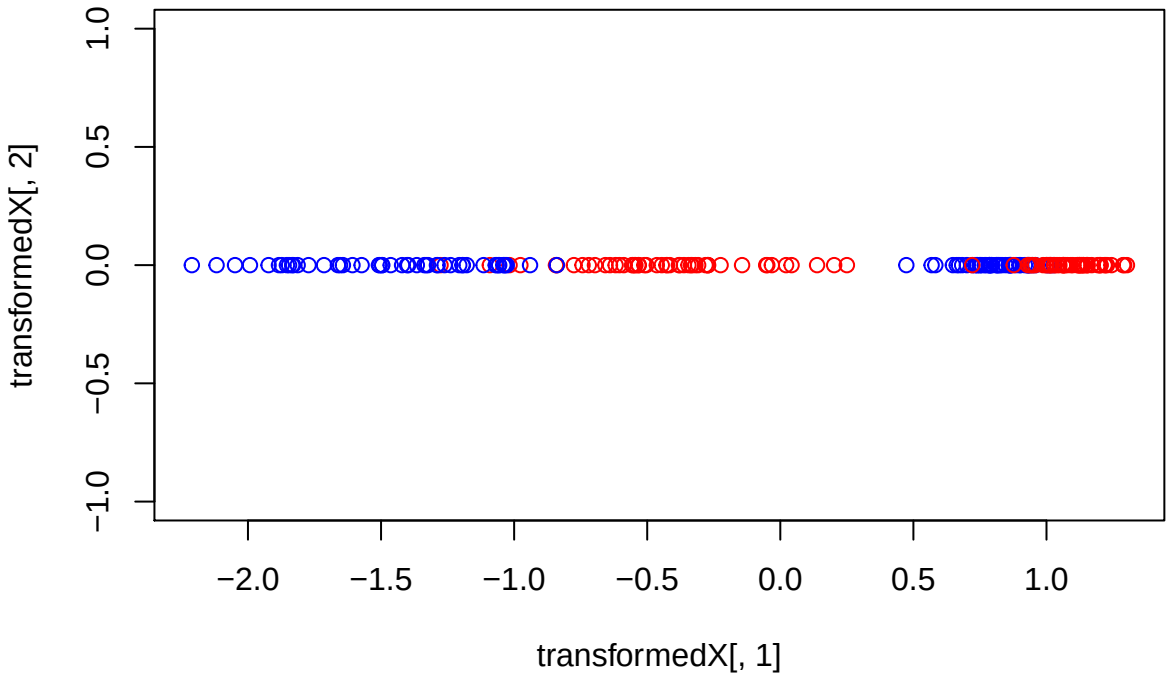
DCA is a global distance metric learning which was introduced in [8]. DCA is similar to RCA, but DCA allows for taking negative constraints into account. DCA can be called in the following manner:

```
# Learn the metric, and get the transformed data
D <- matrix(c(0,1,1,0), nrow = 2, ncol = 2)
result <- DCA(X, chunklets = Y, D = D, total_dims = NULL)

# Get the metric
DCAMetric <- result$DCATransform

# Get the transformed data using
transformedX <- result$transformedX
# Or using
transformedX <- X %*% DCAMetric

# Plot the transformed data
plot(transformedX[,1], transformedX[,2], col = c("blue","red")[Y+1])
```



`chunklets` An integer vector indicating to which chunklet each data point belongs to. If a data point does not belong to a chunklet, then its chunklet value should be set to -1 otherwise all chunklets should be represented by a non-negative integer. All data points in the same chunklet must have the same class label. `D` is a 0-1 symmetric matrix where a 1 at `D[i,j]` indicates that chunklets `i` and `j` share a negative constraint i.e chunklets `i` and `j` do not belong to the same class. The position of the chunklets in the rows and columns of `D` is in increasing order of the integers representing the chunklets. For example, if we have chunklet 1 and chunklet 2, then the first row and first column in `D` will reference chunklet 1, and the second row and second column in `D` will reference chunklet 2. `total_dims` The number of dimensions on which the data should be projected on after the DCA metric has been learned. The default value is set to `total_dims = NULL` in which case the value `total_dims = ncol(X)` is used as default.

Citation

[1] Carl Tony Fakhry, Ping Chen, Rahul Kulkarni and Kourosh Zarringhalam. A Free Energy Based Approach for Distance Metric Learning, Submitted.

[2] Eric P. Xing, Michael I. Jordan, Stuart J Russell, and Andrew Y. Ng. Distance Metric Learning with Application to Clustering with Side-Information. In S. Becker, S. Thrun, and K. Obermayer, editors, Advances in Neural Information Processing Systems 15, pages 521–528. MIT Press, 2003.

[3] T. Hastie, R. Tibshirani, Discriminant adaptive nearest neighbor classification, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (1996) 607–616.

[4] C. Domeniconi, D. Gunopulos, Locally adaptive metric nearest-neighbor classification, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2002) 1281–1285.

[5] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, Advances in Neural Information Processing Systems 17, pages 513–520, Cambridge, MA, 2005. MIT Press.

[6] Masashi Sugiyama. Dimensionality reduction of multimodal labeled data by local Fisher discriminant analysis. Journal of Machine Learning Research, vol.8, 1027–1061.

[7] Masashi Sugiyama. Local Fisher discriminant analysis for supervised dimensionality reduction. In W. W. Cohen and A. Moore (Eds.), Proceedings of 23rd International Conference on Machine Learning (ICML2006),

905–912.

[8] Hoi, S.C.H., Liu, W., Lyu, M.R., Ma, W.Y.: Learning distance metrics with contextual constraints for image retrieval. In: IEEE Conference on Computer Vision and Pattern Recognition.

[9] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning distance functions using equivalence relations. In In Proceedings of the Twentieth International Conference on Machine Learning, pages 11–18.

[10] N. Shental, T. Hertz, D. Weinshall, and M. Pavel. Adjustment learning and relevant component analysis. In Proceedings of the Seventh European Conference on Computer Vision, volume 4, pages 776–792, Copenhagen, Denmark, 2002.