

# Examples of usage of the TSdist package.

Usue Mori, Alexander Mendiburu and  
Jose A. Lozano

In this document we show some examples of usage of the functions in the TSdist package for R.

## 1 Examples of distance calculations between numeric vectors

The distance calculation between times series saved as univariate and numeric vectors can be made with the functions of the type `MethodDistance` of the TSdist package. Of course, `Method` must be substituted by the name of a specific distance measure. For a complete list of distance measures available in the package and the function names of each of them, the reader can access the documentation files. In the following paragraphs we show some examples of usage of these functions.

The `example.series1` and `example.series2` objects are two numeric vectors that represent two different synthetic series which have been generated based on the shapes that define the Two Patterns synthetic database of series (Geurts, 2002) (See Figure 1).

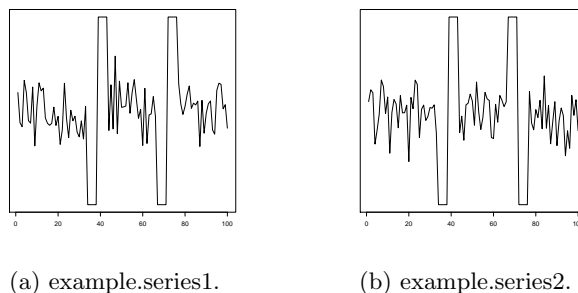
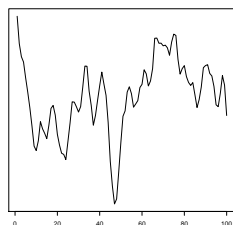


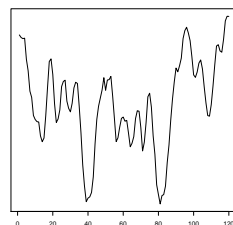
Figure 1: The two example series of the same length included in the TSdist package.

Additionally, `example.series3` and `example.series4` represent two ARMA(3,2)

series of coefficients  $AR=(1, -0.24, 0.1)$  and  $MA=(1, 1.2)$  but with different lengths, 100 and 120, and generated with different random seeds (See Figure 2).



(a) example.series3.



(b) example.series4.

Figure 2: The two example series of different lengths included in the TSdist package.

**Example 1** To calculate the distance between two series such as `example.series1` and `example.series2`:

```
R> DissimDistance(example.series1, example.series2)
```

```
[1] 1.655318
```

```
R> CCorDistance(example.series1, example.series2)
```

```
[1] 1.192903
```

```
R> CorDistance(example.series1, example.series2)
```

```
[1] 1.399347
```

```
R> ARPicDistance(example.series1, example.series2)
```

```
[1] 0.1315731
```

**Example 2** The  $L_p$  distances are a special case, because they can also be called by the wrapper function `LPDistance`:

```
R> ManhattanDistance(example.series1, example.series2)
```

```
[1] 185.1962
```

is equivalent to:

```
R> LPDistance(example.series1, example.series2, method="manhattan")
```

```
[1] 185.1962
```

**Example 3** *Many of the distance measures require the definition of a parameter, which must be included in the call to the corresponding function:*

```
R> EDRDistance(example.series1, example.series2, epsilon=0.1)
```

```
[1] 80
```

```
R> ERPDistance(example.series1, example.series2, g=0)
```

```
[1] 98.29833
```

**Example 4** *Some distance measures can be computed between series of different lengths but in some other cases this is not possible. In these latter cases, the distance can not be computed and the function will return NA and an error message:*

```
R> LCSSDistance(example.series3, example.series4, epsilon=0.1)
```

```
[1] 16
```

```
R> FourierDistance(example.series3, example.series4)
```

```
[1] NA
```

```
Error : The length of the series must be equal
```

**Example 5** *Other errors will be specific to each distance measure and will be determined based on their definition. As in the previous case, when encountering an error, an NA value will be returned and the corresponding error message will be printed. As an example, the window function defined for DTW can not exceed the size of the time series being compared.*

```
R> length(example.series3)
```

```
[1] 100
```

```
R> length(example.series4)
```

```
[1] 120
```

```
R> DTWDistance(example.series3, example.series4, sigma=105)
```

```
[1] NA
```

```
Error : The window size exceeds the length of the first series
```

## 2 Examples of distance calculations between time series objects

The wrapper function called `TSDistances` enables the calculation of distance measures between univariate time series objects of type `ts`, `xts` (Ryan and Ulrich, 2013) and `zoo` (Zeileis and Grothendieck, 2005). This function just takes care of the conversion of data types and then makes use of the chosen `MethodDistance` function. In the following paragraphs we show some examples of usage of this function.

The `zoo.series1` and `zoo.series2` time series included in the package are replicas of the `example.series1` and `example.series2` objects introduced previously but saved in a `zoo` format with a specific time index.

**Example 6** *A basic call to the `TSDistances` function for two series like these is done in the following manner:*

```
R> TSDistances(zoo.series1, zoo.series2, distance="cor")

[1] 1.399347

R> TSDistances(zoo.series1, zoo.series2, distance="dtw", sigma=10)

[1] 123.8757
```

The distance calculation between `ts` or `xts` objects is made in the same manner.

## 3 Examples of distance matrix calculations

In this section, we show how to calculate distance matrices from time series databases.

The `example.database` object included in the package is a matrix that represents a database with 6 ARMA(3,2) series of coefficients AR=(1, -0.24, 0.1) and MA=(1, 1.2) but with different innovations. The 6 series are set in a matrix in a row-wise format. Additionally, the `zoo.database` object included in the package is a multivariate `zoo` object that saves the series of `example.database` but with a specific time index.

**Example 7** *The `TSDatabaseDistances` function calculates the pairwise distance between all the rows in a matrix, so the calculation of the distance matrix can be done easily for the `example.database` object:*

```
R> TSDatabaseDistances(example.database, distance="tquest",
+                       tau=0)
```

```

      series1    series2    series3    series4    series5
series2 0.10257949
series3 0.06673680 0.05260503
series4 0.05891984 0.09107687 0.02323843
series5 0.08248698 0.12273356 0.09874005 0.04785523
series6 0.04706714 0.03049604 0.01984044 0.02876312 0.06191323

```

```
R> TSDatabaseDistances(example.database, distance="euclidean")
```

```

      series1    series2    series3    series4    series5
series2 71.93748
series3 58.67650 60.33323
series4 69.81260 95.93199 60.53757
series5 62.26896 82.10133 69.00826 73.20674
series6 99.88996 118.57977 94.73737 84.40461 108.50799

```

**Example 8** The `zoo.database` object is not a matrix, but the `TSDatabaseDistances` function can also deal with this type of time series databases:

```
R> TSDatabaseDistances(zoo.database, distance="tquest",
+ tau=0)
```

```

      series1    series2    series3    series4    series5
series2 0.10257949
series3 0.06673680 0.05260503
series4 0.05891984 0.09107687 0.02323843
series5 0.08248698 0.12273356 0.09874005 0.04785523
series6 0.04706714 0.03049604 0.01984044 0.02876312 0.06191323

```

```
R> TSDatabaseDistances(zoo.database, distance="euclidean")
```

```

      series1    series2    series3    series4    series5
series2 71.93748
series3 58.67650 60.33323
series4 69.81260 95.93199 60.53757
series5 62.26896 82.10133 69.00826 73.20674
series6 99.88996 118.57977 94.73737 84.40461 108.50799

```

**Example 9** An additional capability of the `TSDatabaseDistances` function is that the distance matrix between two databases can also be calculated. In this case, all the pairwise distances between the series in the first database and the second database are calculated.

For example, if we separate the `example.database` database into two sets and apply `TSDatabaseDistances`, then we obtain the following distance matrix:

```
R> database1<-example.database[1:3,]
R> database2<-example.database[4:6,]
R> TSDatabaseDistances(database1, database2, distance="tquest",
+ tau=mean(example.database))
```

```

      series4    series5    series6
series1 0.05345349 0.08355246 0.04768702
series2 0.07685220 0.12273356 0.03049604
series3 0.02003566 0.09874005 0.01984044

```

```
R> TSDatabaseDistances(database1, database2, distance="euclidean")
```

```

      series4    series5    series6
series1 69.81260 62.26896 99.88996
series2 95.93199 82.10133 118.57977
series3 60.53757 69.00826 94.73737

```

## 4 Use of TSdist to solve classification and clustering problems

The most common usage of time series distance measures is within clustering and classification tasks, and all the measures included in this package can be useful within these two frameworks. For this reason, the **TSdist** package includes two functions (**OneNN** and **KMedoids**) which implement the 1-NN classifier and the K-medoids clustering algorithm, respectively.

Given a pair of train/test time series datasets and the class values of the series in the training set, the **oneNN** function outputs the predicted class values for the test series. Additionally, if the ground truth class values of the series in the testing set are provided by the user, the error obtained in the classification process is also calculated.

For example, suppose we want to classify the series in the **example.database2** database included in **TSdist**, which contains 100 series from 6 very different classes. In order to simulate a typical classification framework, we divide the database into two sets by randomly selecting 30% of the series for training purposes and 70% for testing.

```

R> set.seed(100)
R> trainindex <- sample(1:100, 30, replace=FALSE)
R> train <- example.database2[[1]][trainindex,]
R> test <- example.database2[[1]][-trainindex,]
R> trainclass <- example.database2[[2]][trainindex]
R> testclass <- example.database2[[2]][-trainindex]

```

Then, to apply the 1-NN classifier to the testing set with different distance measures, we use the following commands:

```
R> OneNN(train, trainclass, test, testclass, "euclidean")
```

```

$classes
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 4 4 4 4
[39] 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6

```

```

$error
[1] 0

R> OneNN(train, trainclass, test, testclass, "acf")

$classes
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 4 4 4 3 4 4 3 3 4 4 3 4 3
[39] 3 4 3 3 3 3 4 6 6 5 6 5 5 5 6 6 5 6 6 6 6 5 5 6 5 5 6 5 5 6 5 5

$error
[1] 0.4142857

R> OneNN(train, trainclass, test, testclass, "dtw")

$classes
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4
[39] 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6

$error
[1] 0

```

If we don't provide the class values for the testing data, then only the classification result will be provided, but no error value:

```

R> OneNN(train, trainclass, test, "ar.pic")

[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 5 4 4 3 3 3 3 5 4 5 4 4 4 5 3
[39] 4 5 3 3 4 4 4 5 5 5 6 5 5 3 6 5 5 3 6 6 3 5 6 5 3 5 5 5 6 3 4 3

```

Furthermore, if the selected distance measure requires the definition of any parameters, these should be included at the end of the call:

```

R> OneNN(train, trainclass, test, testclass, "tquest", tau=85)

$classes
[1] 1 3 3 3 2 3 3 3 5 1 2 3 3 3 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 6 4 4
[39] 4 6 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 4 6 6 4 4 6 4 6 6 6 6 6 6 6 6

$error
[1] 0.2571429

```

In the case of clustering, the idea is similar, but in this case we use the `KMedoids` function. Given the data and the number of clusters, this function outputs the clustering result together with the F evaluation measure, (Wagner and Wagner, 2007), if the ground truth clustering is provided by the user.

For example, to cluster the `example.database3` database included in `TSdist` using a variety of distances, first, we save the actual data on one object and the ground truth clustering in another:

```
R> data(example.database3)
R> data <- example.database3[[1]]
R> ground.truth <- example.database3[[2]]
```

Then we apply the KMedoids function as follows:

```
R> KMedoids(data, 5, ground.truth, "dtw")

$clustering
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4
[39] 4 4 3 3 5 3 5 5 3 5 3 5

$F
[1] 0.8933333

R> KMedoids(data, 5, ground.truth, "manhattan")

$clustering
 [1] 1 1 1 2 1 2 3 2 1 2 2 4 1 2 5 1 4 1 4 1 5 2 5 5 5 5 2 4 2 4 3 3 2 3 1 2 3 2
[39] 3 2 5 5 2 5 1 2 5 2 5 2

$F
[1] 0.4907143

R> KMedoids(data, 5, ground.truth, "spec.glk")

$clustering
 [1] 1 1 1 2 3 1 1 3 4 3 4 5 5 5 5 4 5 1 3 2 1 1 4 1 1 1 1 4 1 4 4 5 2 4 5 5 5 4
[39] 5 4 1 1 4 1 4 3 4 4 3 1

$F
[1] 0.4453704
```

As before, if the distance requires the definition of a parameter, we include it at the end of the call:

```
R> KMedoids(data, 5, ground.truth, "edr", epsilon=0.1)

$clustering
 [1] 1 1 1 2 1 2 3 2 1 1 4 2 5 1 2 1 3 5 2 5 1 5 5 2 4 2 5 1 5 3 3 2 3 2 3 3 3 3
[39] 2 3 2 4 4 4 5 1 4 4 4 2

$F
[1] 0.5431746
```

Finally, if the ground truth clustering is not provided, then only the clustering result will be provided, but no F value.

```
R> KMedoids(data, 5, "edr", epsilon=0.1)

 [1] 1 1 1 2 1 2 3 2 1 1 4 2 5 1 2 1 3 5 2 5 1 5 5 2 4 2 5 1 5 3 3 2 3 2 3 3 3 3
[39] 2 3 2 4 4 4 5 1 4 4 4 2
```



## References

- P. Geurts. *Contributions to decision tree induction: bias/variance tradeoff and time series classification*. PhD thesis, University of Liege, Belgium., 2002.
- Jeffrey A. Ryan and Joshua M. Ulrich. xts: eXtensible Time Series, 2013. URL <http://cran.r-project.org/package=xts>.
- Silke Wagner and Dorothea Wagner. Comparing Clusterings - An Overview. Technical Report 2006-04, Universität Karlsruhe (TH), 2007. URL <http://digbib.ubka.uni-karlsruhe.de/volltexte/1000011477>.
- Achim Zeileis and Gabor Grothendieck. zoo: S3 Infrastructure for Regular and Irregular Time Series. *Journal of Statistical Software*, 14(6):1–27, 2005.