Package ‘vudc’

October 12, 2022

Type Package

Title Visualization of Univariate Data for Comparison

Version 1.1

Date 2016-02-20

Author Csaba Farago

Maintainer Csaba Farago <farago@inf.u-szeged.hu>

Description Contains functions for visualization univariate data: ccdplot and qddplot.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2016-02-21 16:30:22

R topics documented:

`vudc-package` *Visualization of Univariate Data for Comparison*

Description

Contains two functions for visualization univariate data: `ccdplot` and `qddplot`.

Details

Package:	vudc
Type:	Package
Version:	1.1
Date:	2016-02-20
License:	GPL (>= 2)

The data visualization functions are the following:

- `ccdplot` - Cumulative Characteristic Diagram
- `qddplot` - Quantile Difference Diagram

The package contains input data for case study, related to four software projects written in Java (`projectdata`).

Author(s)

Csaba Farago <farago@inf.u-szeged.hu>

References

Csaba Farago. Visualization of Univariate Data for Comparison. *Annales Mathematicae et Informaticae*, volume 45, pages 39–53, 2015.

`ccdplot` *Cumulative Characteristic Diagram*

Description

The parameter array (or list of arrays) is sorted non-ascending. Then cumulative values are calculated for every value. E.g. the fifth element of the resulting array will be the sum of the first five elements in the sorted array. The resulting array is illustrated visually, taking the vector indices as x coordinate, the calculated cumulative values as y coordinate, and the points are connected with lines.

The function can handle an array or list of arrays. In the latter case all the values are drawn on the same diagram, along with the union of them (if not stated otherwise); this is called Composite Cumulative Characteristic Diagram. The diagram works with numeric data of any distribution; however, the most spectacular results are expected in case of data of normal distribution with mean of 0.

Usage

```
ccdplot(x, remove.absolute = NA, remove.ratio = NA,
drawcomposite = TRUE, jump = NA,
xlab = "Observations", ylab = "Cumulatives", ...)
```

Arguments

`x`	the array or list of arrays to be illustrated.
`remove.absolute`	if provided, the values greater than this one in absolute are not considered in drawing the diagram. Useful if the dataset contains outliers, as this diagram is sensitive on them.
`remove.ratio`	if provided, the upper and lower ratio is removed before drawing the diagram. E.g. if this value is 0.05, then the lower and upper 5% is omitted. If the parameter `x` is list of arrays, then this is performed globally (i.e. not one by one on each element).
`drawcomposite`	this logical parameter indicates if the union of the provided data should be drawn. The default value is true. This feature is especially useful if the values within arrays are disjoint: it provides an overview how the individual parts fit to the whole.
`jump`	in case of Composite Cumulative Characteristic Diagrams (i.e. if the `x` is a list), the diagram looks better if small gaps between characteristic lines are drawn. The size if this gap is automatically calculated; however, that can be overwritten with this parameter.
`xlab`	title for the x axis, see `plot`.
`ylab`	title for the y axis, see `plot`.
`...`	standard graphic parameters. See `par` for details.

Details

The cumulative characteristic diagram of normal distribution with mean value 0 is an upside-down U shape.

If the right end of the diagram is above the left end, it means the sum of positive values are greater than the absolute value of the sum of negative values, meaning the average is above 0. A long horizontal part in the middle indicates high number of 0 values.

In case of comparing two or more characteristic, if one of them turns to be relatively "taller" than the others (i.e. their heights are more or less the same, but their width significantly differ; in this case considering the narrowest), this means the variance of that subset is greater than variance in the other cases.

Author(s)

Csaba Farago <farago@inf.u-szeged.hu>

Examples

```
# Create cumulative characteristic diagram using 100 random values
# of standard normal distribution.
ccdplot(rnorm(100));

# Create cumulative characteristic diagram using 95 random values
# of standard normal distribution and 5 outliers.
```

```
ccdplot(c(rnorm(95), 11:15));

# Neutralize the bias caused by the outliers,
# removing the values greater than 10 in absolute.
ccdplot(c(rnorm(95), 11:15), remove.absolute=10);

# Neutralize the bias caused by the outliers,
# removing the upper and lower 10%.
ccdplot(c(rnorm(95), 11:15), remove.ratio=0.1);

# Create composite cumulative characteristic diagram of 4 arrays
# (normal distribution, different element numbers, all the mean are 0,
# the standard deviations are different).
ccdplot(list(rnorm(50, 0, 5), rnorm(100), rnorm(150), rnorm(200, 0, 0.2)))

# Set some standard graphic parameters.
ccdplot(rnorm(100), main="Cumulative Characteristic Diagram",
sub="100 random values of standard normal distribution",
ylab="Summed random values", xlab="Sorted occurrences", col="red");
```

---

projectdata *Commit and Maintainability Data*

Description

This dataset contains commit and maintainability data about every available revision of four software projects.

Usage

```
data("projectdata")
```

Format

The outermost structure of the data is a list; each element of the list contains commit-related information about one Java project (number of commits in bracket): `Ant` (6102), `Gremon` (1158), `Struts2` (1749) and `Tomcat` (1292).

Each element of the list is a data frame of same structure. One row of data frame contains information about one commit. There are the following columns for each commit:

- `Revision` (integer): revision number
- `MaintainabilityDiff` (double): number indicating how the maintainability was changed on the effect of that commit
- `A` (integer): number of Java files added
- `U` (integer): number of Java files updated
- `D` (integer): number of Java files deleted
- `Churn` (double): code churn value of the commit
- `Ownership` (double): ownership value of the commit

Details

The Java projects are the following:

- Ant: a command line tool for building Java applications (`http://ant.apache.org`)
- Gremon: a greenhouse work-flow monitoring system (`http://www.gremonsystems.com`)
- Struts2: a framework for creating enterprise-ready Java web applications (`http://struts.apache.org/`)
- Tomcat: an implementation of the Java Servlet and Java Server Pages technologies (`http://tomcat.apache.org`)

The data frame contains information about all the available commits from the master branch, which affects at least one Java file.

We determined the maintainability using the Columbus Quality Model. Positive value means that the actual commit increased the maintainability of the overall source code, and negative value indicates the opposite. The absolute value indicates the magnitude of the change. For details how we calculated these values are described in the referred papers.

The source code and version control information for Ant, Struts 2 and Tomcat are publicly available. The Gremon is an industrial software; neither the source code nor the version control information are public.

Author(s)

Csaba Farago <farago@inf.u-szeged.hu>

References

Tibor Bakota, Peter Hegedus, Peter Kortvelyesi, Rudolf Ferenc, and Tibor Gyimothy. A probabilistic software quality model. In *Proceedings of the 27th International Conference on Software Maintenance (ICSM)*, pages 243-252. IEEE Computer Society, 2011.

Csaba Farago, Peter Hegedus, Adam Zoltan Vegh, and Rudolf Ferenc. Connection Between Version Control Operations and Quality Change of the Source Code. *Acta Cybernetica*, volume 21(4), pages 585-607, 2014.

Csaba Farago, Peter Hegedus, and Rudolf Ferenc. The Impact of Version Control Operations on the Quality Change of the Source Code. In *Proceedings of the 14th International Conference on Computational Science and Its Applications (ICCSA)*, volume 8583 Lecture Notes in Computer Science (LNCS), pages 353-369. Springer International Publishing, 2014.

Csaba Farago. Variance of Source Code Quality Change Caused by Version Control Operations. *Acta Cybernetica*, volume 22(1), pages 35-56, 2015.

Csaba Farago, Peter Hegedus, and Rudolf Ferenc. Code Ownership: Impact on Maintainability. In *Proceedings of the 15th International Conference on Computational Science and Its Applications (ICCSA)*, volume 9159 Lecture Notes in Computer Science (LNCS), pages 3-19. Springer International Publishing, 2015.

Csaba Farago, Peter Hegedus, and Rudolf Ferenc. Cumulative Code Churn: Impact on Maintainability. In *Proceedings of the 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 141-150. IEEE Computer Society, 2015.

Examples

```
# Initialization.
library(vudc)
data(projectdata)

#
# Helper functions.
# These are necessary for the drawing and test execution functions.
#

# Frame for drawing four diagrams.
drawFourDiagrams <- function(diagram) {
op <- par(mfrow = c(2,2))
performOperationAllProjects(diagram)
par(op)
}

# Frame for executing an operation on all the available project data.
performOperationAllProjects <- function(operation) {
for (projectName in c("Ant", "Gremon", "Struts2", "Tomcat")) {
operation(projectName)
}
}

# Helper functions for getFourDivisions and categorizeProjectData.

commitCategoryD <- function(actualProjectData) {
actualProjectData$D > 0
}

commitCategoryA <- function(actualProjectData) {
actualProjectData$D == 0 & actualProjectData$A > 0
}

commitCategoryUplus <- function(actualProjectData) {
actualProjectData$D == 0 & actualProjectData$A == 0 & actualProjectData$U >= 2
}

commitCategoryU1 <- function(actualProjectData) {
actualProjectData$D == 0 & actualProjectData$A == 0 & actualProjectData$U == 1
}


# Divides the maintainability difference values into four, based on the related operations.
getFourDivisions <- function(actualProjectData) {
return (list(
actualProjectData$MaintainabilityDiff[commitCategoryD(actualProjectData)],
actualProjectData$MaintainabilityDiff[commitCategoryA(actualProjectData)],
actualProjectData$MaintainabilityDiff[commitCategoryUplus(actualProjectData)],
actualProjectData$MaintainabilityDiff[commitCategoryU1(actualProjectData)]
))
}
```

```
# Extend project data with  operation category (opcat)
# and maintainability change category (changecat).
categorizeProjectData <- function(projectName) {
actualProjectData <- projectdata[[projectName]]

actualProjectData$opcat[commitCategoryD(actualProjectData)] <- 'D';
actualProjectData$opcat[commitCategoryA(actualProjectData)] <- 'A';
actualProjectData$opcat[commitCategoryUplus(actualProjectData)] <- 'U+';
actualProjectData$opcat[commitCategoryU1(actualProjectData)] <- 'U1';

actualProjectData$changecat[actualProjectData$MaintainabilityDiff > 0] <- '+';
actualProjectData$changecat[actualProjectData$MaintainabilityDiff == 0] <- '0';
actualProjectData$changecat[actualProjectData$MaintainabilityDiff < 0] <- '-';

actualProjectData
}

# Removed outliers from project data.
projectDataWithoutOutliers <- function(projectName) {
projectdata[[projectName]][abs(projectdata[[projectName]]$MaintainabilityDiff) < 1000,]
}

# Maintainability change values of commits containing file addition.
maintainabilityDiffsContainingAdd <- function(actualProjectData) {
actualProjectData$MaintainabilityDiff[actualProjectData$A > 0]
}

# Maintainability change values of commits not containing file addition.
maintainabilityDiffsWithoutAdd <- function(actualProjectData) {
actualProjectData$MaintainabilityDiff[actualProjectData$A == 0]
}



#
# Diagram drawing and test execution functions.
# These can be executed independent from each other.
#

# Box plots illustrating maintainability change values.
# This is the motivating example for creating ccdplot.
drawFourDiagrams(
function(projectName) {
boxplot(
getFourDivisions(projectdata[[projectName]]),
main = projectName,
names = c("D", "A", "U+", "U1")
)
}
)

# Box plots illustrating maintainability change values, without outliers.
```

```
drawFourDiagrams(
function(projectName) {
boxplot(
getFourDivisions(projectDataWithoutOutliers(projectName)),
main = projectName,
names = c("D", "A", "U+", "U1")
)
}
)

# Composite cumulative characteristic diagrams illustrating maintainability change values,
# including outliers.
drawFourDiagrams(
function(projectName) {
ccdplot(
getFourDivisions(projectdata[[projectName]]),
main = projectName,
sub = "D, A, U+, U1",
xlab = "Revisions",
ylab = "Accumulated maintainability change"
)
}
)

# Composite cumulative characteristic diagrams illustrating maintainability change values,
# without outliers.
drawFourDiagrams(
function(projectName) {
ccdplot(
getFourDivisions(projectDataWithoutOutliers(projectName)),
main = projectName,
sub = "D, A, U+, U1",
xlab = "Revisions",
ylab = "Accumulated maintainability change"
)
}
)

# Perform Contingency Chi Squared Test on the input data.
performOperationAllProjects (
function(projectName) {
projectDataCategorized <- categorizeProjectData(projectName)
print(paste(projectName, "contingency test", sep = " - "))
chisq.test.result <- chisq.test(table(
projectDataCategorized$opcat,
projectDataCategorized$changecat)
)
print(chisq.test.result)
print(paste("Exact p-value:", chisq.test.result$p.value))
}
)

# Cumulative characteristic diagrams illustrating the maintainability change values
# between commits containing and not containing file additions.
drawFourDiagrams(
function(projectName) {
actualProjectData <- projectdata[[projectName]]
maintainabilityDiffs <- list(
maintainabilityDiffsContainingAdd(actualProjectData),
maintainabilityDiffsWithoutAdd(actualProjectData)
)
ccdplot(
maintainabilityDiffs,
remove.absolute = 1000.0,
main = paste(projectName, "Add", sep = " - "),
sub = "Containing vs. not containing Add",
xlab = "Revisions",
ylab = "Accumulated maintainability change"
)
}
)

# Quantile difference diagrams illustrating the difference of the maintainability change values
# between commits containing and not containing file additions.
drawFourDiagrams(
function(projectName) {
actualProjectData <- projectdata[[projectName]]
qddplot(
maintainabilityDiffsContainingAdd(actualProjectData),
maintainabilityDiffsWithoutAdd(actualProjectData),
main=paste(projectName, "Add", sep = " - "),
sub="Containing vs. not containing Add"
)
}
)

# Testing the difference of the maintainability change values between commits containing
# and not containing file additions, using Wilcoxon-test.
performOperationAllProjects (
function(projectName) {
actualProjectData <- projectdata[[projectName]]
print(paste(projectName, "impact of file addition", sep = " - "))
print(wilcox.test(
maintainabilityDiffsContainingAdd(actualProjectData),
maintainabilityDiffsWithoutAdd(actualProjectData),
alternative = "greater"
))
}
)

# Testing the ratio of the maintainability change variance between commits containing
# and not containing file additions.
performOperationAllProjects (
function(projectName) {
actualProjectData <- projectDataWithoutOutliers(projectName)
print(paste(projectName, "variance of file addition", sep = " - "))
var.test.result <- var.test(
maintainabilityDiffsContainingAdd(actualProjectData),
maintainabilityDiffsWithoutAdd(actualProjectData)
)
print(var.test.result)
print(paste("Exact p-value:", var.test.result$p.value))
}
)

# Quantile difference diagrams illustrating the difference of cumulative code churn values
# between commits of positive vs. negative maintainability change, along with related Wilcox-test.
drawFourDiagrams(
function(projectName) {
actualProjectData <- projectdata[[projectName]]
churnX <- actualProjectData$Churn[actualProjectData$MaintainabilityDiff > 0]
churnY <- actualProjectData$Churn[actualProjectData$MaintainabilityDiff < 0]
print(paste(projectName, "cumulative code churn", sep = " - "))
print(wilcox.test(churnX, churnY, alternative = "less"))
qddplot(
churnX,
churnY,
main=paste(projectName, "Churn", sep = " - ")
)
}
)

# Quantile difference diagrams illustrating the difference of number of contributors
# between commits of positive vs. negative maintainability change, along with related Wilcox-test.
drawFourDiagrams(
function(projectName) {
actualProjectData <- projectdata[[projectName]]
ownershipX <- actualProjectData$Ownership[actualProjectData$MaintainabilityDiff > 0]
ownershipY <- actualProjectData$Ownership[actualProjectData$MaintainabilityDiff < 0]
print(paste(projectName, "code ownership", sep = " - "))
print(wilcox.test(ownershipX, ownershipY, alternative = "less"))
qddplot(
ownershipX,
ownershipY,
differences.rangemin = 0,
main=paste(projectName, "Ownership", sep = " - ")
)
}
)
```

`qddplot` *Quantile Difference Diagram*

Description

Creates quantile difference diagrams. It takes two numerical datasets, sorts them, calculates differences on the same quantiles (i.e. lowest with the lowest, median with the median, 90% with 90% etc.) and displays these differences. It is appropriate for visual illustration of Wilcoxon rank test and F-test of equality of variances.

Usage

```
qddplot(x, y,
remove.ratio = 0.1, differences.range = NA, differences.rangemin = 10,
differences.drawzero = TRUE, quantiles.drawhalf = TRUE,
quantiles.showaxis = TRUE, line.lwd = 5, xlab = "Quantile",
ylab = "Difference", main = "Quantile Differences", ...)
```

Arguments

`x, y`	vectors of numeric values to be compared. It is not necessary to have the same length.
`remove.ratio`	a real number in range [0.0, 0.5). Indicates how much leading and tailing data should not be displayed in order to avoid a bias on the diagram caused by outliers.
`differences.range`	numeric, indicating the value range (i.e. the y axis) to be shown. The default value is NA, indicating the maximum absolute value is considered. This can be overridden with an explicit value.
`differences.rangemin`	numeric, indicating the minimum value of the value range. Used in combination with `differences.range=NA` (the default), else ignored.
`differences.drawzero`	logical, indicating if the y=0 helper line should be displayed.
`quantiles.drawhalf`	logical, indicating if the x=0.5 helper line should be displayed.
`quantiles.showaxis`	logical, indicating if the custom values on the x axis should be displayed.
`line.lwd`	width of the main line.
`xlab`	title for the x axis, see `plot`.
`ylab`	title for the y axis, see `plot`.
`main`	plot title, see `plot`.
`...`	standard graphic parameters. See `par` for details.

Details

The quantile difference diagrams can be used to visualize the results of Wilcoxon rank tests (`wilcox.test`). Wilcoxon rank test basically checks if the values found in one dataset are different (more specifically: higher or lower) than those in the other. If the values in subset x are higher than values in subset y at all quantiles, then the result would look like a line completely on the same part of y coordinate (either completely on the positive or on the negative part).

In most of the cases the diagram resembles on a more or less straight line. The slope indicates the difference between variances (see also `var.test`). The position of the line indicates if the values in one dataset is higher than those in another; being close to 0 at 50%, if they are similar.

Author(s)

Csaba Farago <farago@inf.u-szeged.hu>

Examples

```
# Using default settings with random data.
qddplot(rnorm(100, 30, 50), rnorm(200, 10, 10));

# remove.ratio = 0.0 means the outliers are not removed.
qddplot(rnorm(100, 30, 50), rnorm(200, 10, 10), remove.ratio=0.0);

# remove.ratio = 0.25 means only the middle half is displayed, the upper and lower quantile are not.
qddplot(rnorm(100, 30, 50), rnorm(200, 10, 10), remove.ratio=0.25);

# Illustrating similar and different medians and variances on 4 quantile difference diagrams.
# This is also an illustration for setting custom main title and subtitle.
dataSetA <- seq(-20, 20) + rnorm(41);
dataSetB <- seq(-15, 25) + rnorm(41);
dataSetC <- seq(-40, 40) + rnorm(81);
dataSetD <- seq(-20, 20) + rnorm(41);
op <- par(mfrow=c(2,2));
qddplot(
dataSetA,
dataSetD,
main = "Similar median, similar variance",
sub = "-20...20 vs. -20...20");
qddplot(
dataSetA,
dataSetB,
main = "Different median, similar variance",
sub = "-20...20 vs. -15...25");
qddplot(
dataSetA,
dataSetC,
main = "Similar median, different variance",
sub = "-20...20 vs. -40...40");
qddplot(
dataSetB,
dataSetC,
main = "Different median, different variance",
sub = "-15...25 vs. -40...40");
par(op);

# Change plot style: thicker line in red color.
qddplot(rnorm(100, 30, 50), rnorm(200, 10, 10), line.lwd=10, col="red");
```

```
# Hide axes, helper lines and captions.
qddplot(rnorm(100, 30, 50), rnorm(200, 10, 10), differences.drawzero=FALSE,
quantiles.drawhalf=FALSE, quantiles.showaxis=FALSE, line.lwd=1, yaxt='n',
main="", sub="", xlab="", ylab="");

# Do not consider minimal range.
qddplot(rnorm(100, 1, 1), rnorm(100, 2, 1), differences.rangemin=NA);

# Set an explicit range.
qddplot(rnorm(100, 0, 200), rnorm(100, 0, 200), differences.range=40);
```

Index