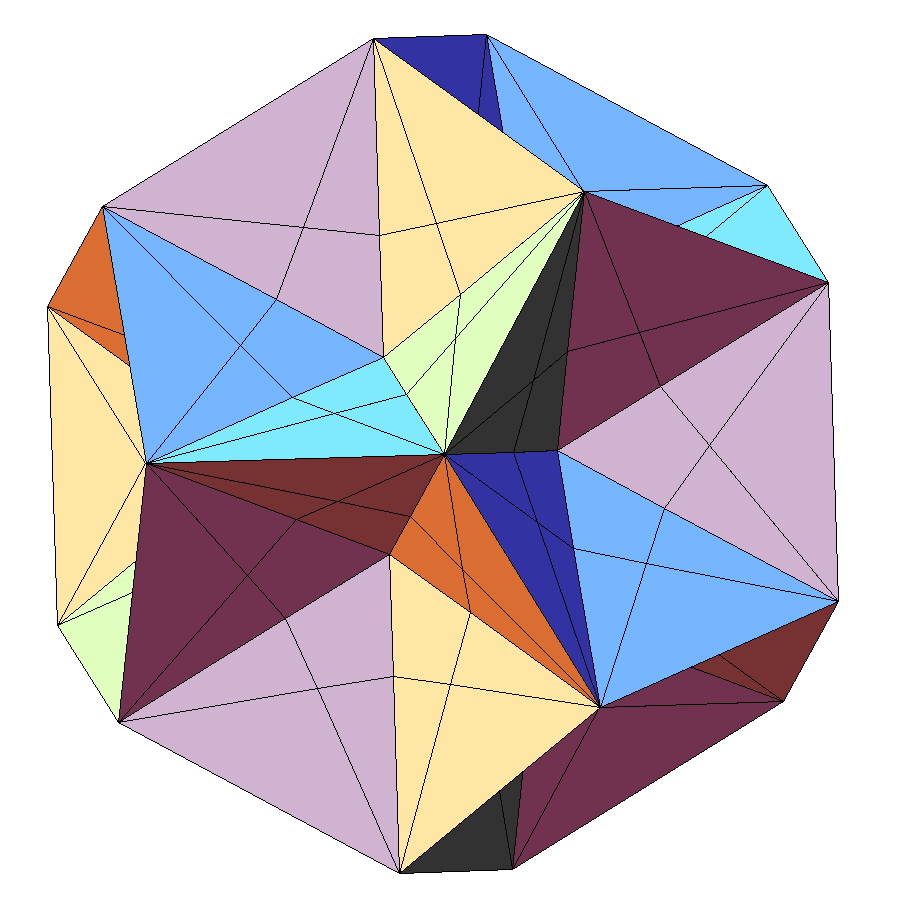
Ranked Data Cluster Analyzation

Programmer’s Guide

Brendan Mikolajczyk, Eric Mustee, Dan Skrodzki



Contents

[Introduction 2](#_Toc406375715)

[Organization and Design 3](#_Toc406375716)

[Where to begin 3](#_Toc406375717)

[How are clusters analyzed anyway? 3](#_Toc406375718)

[Categories of Classes 4](#_Toc406375719)

[The Purpose of Each and Every Class 4](#_Toc406375720)

[Input / Output Files and Formatting 7](#_Toc406375721)

[Troubleshooting 7](#_Toc406375722)

[Future Implementations 7](#_Toc406375723)

# Introduction

This program was primarily developed for Dr. Paige Rinker and her research in the mathematics department at John Carroll University. The purpose of this program is to demonstrate the implementation of Dr. Rinker’s algorithm for finding clusters within a given set of ranked data.

# Organization and Design

The Ranked Data Cluster Analyzer is built upon SWT by the Eclipse foundation. The GUI was designed with Window Builder Pro, a free eclipse plugin available at <http://eclipse.org/windowbuilder/download.php>. The application has separated the mathematic logic behind sorting and the graphical components the user interacts with. These two components are linked through selection listeners, which invoke behaviors when some widget is selected. The GUI components follow a Singleton design pattern so that they may be accessed throughout the program. SelectionListeners closely resemble the command pattern.

## Where to Begin

We would advise to familiarize yourself with the code by first looking at the *MainGUI* class in the gui package. If you are using eclipse, we highly recommend installing WindowBuilderPro, which was mentioned in the previous section. If the program does not launch, please see the section on Troubleshooting. From the Main GUI, we can see that many of the widgets have attached selection listeners. When a widget is selected or deselected, it calls the added selection listener’s *widgetSelected* method. This is where we would look next: the selection listeners. From there, I would pay particular attention to the *AnalyzeBehavior* class. This class invokes the ClusterAnalyzer class. In turn, this invokes each of the computers to actually analyze the clusters. The computers call upon components like *GFunctionComputer* or *DistanceRanker*. The *DisntanceRanker* class is incredibly important because it allows for a “distance” between two rankings to be generated, which is the main concept behind the entire program. For help on understanding this concept, we recommend seeing Dr. Rinker, or another mathematics professor that understands clustering ranked data (there may not be any other professor), as this is where the low-level, mathematical portions of the program begin, and where the higher-level portions we are currently trying to explain ends.

Another main spot of interest separate from the above is the *RandomDataGenerator* in the gui package. This class calls upon *RandomizeableRankedData* in the main class, which generates data for the program to use.

## How Are Clusters Analyzed Anyway?

Clusters are analyzed in a loop in the cluster analyzer class. The order of events in the loop occur based on dependencies: some computers are very dependent on certain vectors, and some are less dependent. First, we created our vectors in the constructor and fill them with default values. Then, we do an initial call to compute the lambda vector, because it has the least amount of dependencies. After we compute the lambda vector, the QVector is computed, then the CVector is computed. Finally, the sigma vector is computed. The sigma vector computer works by trying to decide between the current cluster center and new cluster centers not very different than the current cluster centers. If no new cluster centers are chosen, the loop stops and the cluster centers are known. If one or more new cluster centers are choosen, the loop recycles from computing the lambda vector.

## Categories of Classes

There are six main packages in the project which categorize the classes: computers, exception, gui, main, selectionListeners, and tests. Below, each package is described:

|  |  |
| --- | --- |
| *Package* | *Contents* |
| computers | Contains all the mathematical code in the project. The low level details of the program go inside of this package. |
| exception | Contains what can go wrong when loading in an input file, such as containing an invalid character, an invalid number, or duplicated numbers. |
| gui | Contains what the user sees, and any related components such as SuperStyledText. |
| main | Contains higher-level components which are call upon lower-level components to do their bidding. For instance, ClusterAnalyzer calls upon all of the computers to do its work. |
| selectionListeners | Contains listeners that are invoked when the user does something on screen. For instance, if the user selects Edit > Copy, the widgetSelected method is called in CopyBehavior. |
| tests | Contains unit tests to make sure nothing breaks when modification occurs. These are not actually used inside the user experience. |

## The Purpose of Each and Every Class

Want to know what each class does each class do and why does it even exist? Below, the purpose of each class is explained, organized by category.

|  |  |
| --- | --- |
| *computers* | |
| Class Name | **Description** |
| *CComputer* | Works on the c-vector, which are used for cluster weightings. |
| *GFunctionComputer* | A sub-component of the QVector computer, does a single mathematical function that becomes easier to use in QVector. |
| *LambdaComputer* | Updates the dispersion-parameters, or otherwise known as the lambda vector. |
| *QVectorComputer* | Updates the Q-Vector, or the assignment probability vector. It tells the likelihood of which cluster a given ranking is most likely to belong to. |
| *SigmaComputer* | Updates the sigma vector, or the actual cluster centers. This is accomplished by taking 1 step at a time. |

|  |  |
| --- | --- |
| *exception* | |
| Class Name | **Description** |
| *DuplicateIntegerException* | The exception is thrown in FileLoader and AddClusterCenter when two integers are attempted to be used in a single ranking. There can be at most one of any integer in a given ranking. |
| *EmptyPiVectorException* | This expcetion is thrown in FileLoader when the user attempts to load a file without any rankings. |
| *InvalidCharacterException* | This exception is thrown in AddDescriptions when the user attempts to input a reserved ~ or \ character. |
| *ZeroInFileException* | This exception is thrown in FileLoader and AddClusterCenter when the user attempts to use a 0 in a ranking. Zero cannot be used as a ranking because there is no distinction between positive zero and negative zero, and rankings must be distinguishable. |

|  |  |
| --- | --- |
| *gui* | |
| Class Name | **Description** |
| *AboutDialog* | This is the dialog that pops up when the user selects Help > About. This was built with WindowBuilderPro. |
| *ErrorDialog* | The error dialog appears when an exception is caught in a lot of places. This was built with WindowBuilderPro |
| *MainGUI* | Perhaps the most important class in our program, MainGUI contains all of the widgets that cause things to happen in the program. It is linked with selection listeners which invoke the lower level behaviors in the program. |
| *RandomDataGenerator* | Generates random data based on certain characteristics to be used for input in the program. |
| *SuperStyledTexet* | A Façade for SWT’s styled text, it allows one to easily format text without having to worry about |
| *Textilizer* | A sub-component to super-styled text which searches for the escape sequences “\\~S” for superscript and “\\~s” for subscript. |

|  |  |
| --- | --- |
| *main* | |
| Class Name | **Description** |
| *ClusterAnalyzer* | This is the class that analyzes the input clusters by calling upon the computers. |
| *DistanceRanker* | DistanceRanker is able to get the distance between two given rankings: a partial ranking and a complete ranking, or two complete rankings. |
| *FileLoader* | The file loader is the class that handles the input of processing csv files. |
| *FileType* | Technically not a class, but an enumeration of all the file types supported by the system. |
| *QVector* | A nice wrapper for a 2D array that gives names to its rows in columns for easier access. |
| *RandomizeableRankedData* | A subclass of RankedData. This class can generate new RankedData given a set of parameters and the underlying ranking. |
| *RankedData* | Contains a single ranking. |
| *Settings* | An object that saves the state of the program. It implements Serializable so it can be saved to disk and retrieved later. |

|  |  |  |
| --- | --- | --- |
| *selectionListeners* | | |
| Class Name | **Description** | |
| *AboutDialogBehavior* | Opens the about dialog located at Help > About | |
| *AddClusterCenter* | Adds a cluster center to the random data generator. | |
| *AddDescriptions* | Adds a description to the descriptions list | |
| *AnalyzeBehavior* | Invokes the ClusterAnalyzer object to analyze the pi vector. | |
| *ChangeClusterCenterBehavior* | Changes a cluster center in the random data generator. | |
| *CloseDialogBehavior* | Is used to close error dialogs. | |
| *CopyBehavior* | Copys the selected text to the clipboard. | |
| *ExitBehavior* | Is called when the program exits, and asks the user if they want to save in the case modifications were made. | |
| *HelpMenuBehavior* | Opens the help menu when the user selects Help> View Help. | |
| *ListChangedBehavior* | Is called to modify the controls when the user selects another cluster center in the Random Data Generator. | |
| *OpenFileBehavior* | Opens a file of a given file type present in FileType. | |
| *RandomDataGeneratorStartBehavior* | | Opens the Random Data Generator. |
| *RemoveClusterCenter* | Removes the selected cluster center from the random data generator. | |
| *RemoveDescription* | Removes the selected description from the description list. | |
| *SaveFileBehavior* | Saves a file of a specified FileType present in FileType. | |
| *SelectAllBehavior* | Selects all the text. | |
| *ShowOrHideExpandBar* | Shows or hides the side view with the controls. | |
| *ZoomInBehavior* | Makes the text bigger. | |
| *ZoomOutBehavior* | Makes the text smaller. | |

|  |  |
| --- | --- |
| *tests* | |
| Class Name | **Description** |
| *DistanceTesting* | Tests all the features of the distance ranker. |
| *FileLoadingTesting* | Tests file loading to ensure loading of files is not broken. |
| *GFunctionTesting* | Tests the GFunction computer to ensure its math is not broken. |
| *SigmaTesting* | Tests the sigma computer that is chooses to make progress towards the correct cluster. |

# Input / Output Files and Formatting

Acceptable input files for analysis are .txt files and .csv files. Text files will throw an error if they contain any non-numeric characters, zeroes, or a duplicate integer on a single line. The fileLoader parses individual integers by the following delimiters: commas and spaces. Input files for restoring a session can only be .rnkr files, which are generated when saving session.

All the settings in the settings class are serializable. Object output streams and object input streams are used in reading theses settings. The object output streams save settings objects to a file which holds all the values of these objects. The object input streams accept .rnkr files and read the saved objects to convert back into usable settings. Additionally, you can export the results of an analysis into a .txt file. Essentially, the text in under each result tab is copied and properly formatted into a .txt file.

# Troubleshooting

If upon launching the project and you are greeted with an error, you may be using the wrong SWT jar in your build path. SWT is very picky about your Java Runtime Enviornment – both the operating system (Windows/Mac/Linux) and the architecture (x86 vs x64) of the JRE matter, and the correct jar must be added to the build path, and the incorrect jar must be removed from the build path. We have included the three most popular jars for SWT (Windows x86 and x64, plus Mac x64), and more can be found easily online for alternate operating systems. The project has only officially supported Windows, but it would take a minimal amount of effort for a port to Linux or Mac OS X, as the entire program is written in cross-platform compatible Java, save for the included jars in the build-path.

# Future Implementations

In the future, there is a possibility to implement a visualization aspect to this program. By working with specialized software, it may be possible to simulate a 3-dimensional visual image that portrays the arrangement of the clusters in space. This would be quite a task to take on, and we found it to be of low importance in our current implementation. As far as any other future implementations, that is entirely up to Dr. Rinker and how she decides to use this program.