DataVilij™

Software Requirements Specification

DataVilij™

Author: Ritwik Banerjee
Professaur Inc. ™

Based on the IEEE Std 830™-1998 (R2009) document format

Table of Contents

1. Introduction

Given the increasing importance of data-driven artificial intelligence (AI) in many aspects of computer science, visualizing how AI algorithms work is becoming increasingly important. Java is among the most important programming languages used to implement these algorithms, but it lacks standard data visualization libraries (unlike some other languages such as Python). Moreover, all existing libraries are meant to show us the final output of the data science algorithms. They are not designed for visualizing the changes that happen *while* the algorithms are running and updating the data. In other words, the visualization libraries do not help us see *how* these algorithms learn from the data.

DataViLiJ (Data Visualization Library in Java) will be a desktop application that will allow users to select an algorithm (from a set of standard AI algorithms) and dynamically show the user what changes, and how.

1.1 Purpose

The purpose of this document is to specify how the DataViLiJ application should look and operate. This document serves as an agreement among all parties and as a reference for how the data visualization application should ultimately be constructed. Upon reading of this document, one should clearly understand the visual aesthetics and functionalities of application's user interface as well as understand the way AI algorithms are incorporated.

1.2 Intended Audience

The intended audience for this document is the development team, including instructors, software designer, and software developers.

1.3 Product Scope

The goal of this project is for students and beginning professionals in AI to have a visual understanding of the inner workings of the fundamental algorithms. AI is a vast field, and this project is limited to the visualization of two types of algorithms that "learn" from data. These two types are called **clustering** and **classification**. The design and development of these algorithms is outside the scope of the project, and the assumption is that such algorithms will already be developed independently, and their output will comply with the data format specified in this document. DataViLiJ serves simply as a visualization tool for how those algorithms work. Both clustering and classification are, in theory, not limited to a fixed number of labels for the data, but this project will be limited to at most four labels for clustering algorithms, and exactly two labels for classification algorithms. Further, the design and development of this project will also assume that the data is 2-dimensional. As such, 3D visualization is currently beyond the scope of DataViLiJ.

As for the GUI interactions, touch screen capabilities are not within the scope of this application.

1.3 Definitions, acronyms, and abbreviations

1) **Algorithm:** In this document, the term 'algorithm' will be used to denote an AI algorithm that can "learn" from some data and assign each data point a label.
2) **Clustering:** A type of AI algorithm that learns to assign labels to instances based purely on the spatial distribution of the data points.
3) **Classification:** A type of AI algorithm that learns to assign new labels to instances based on how older instances were labeled. These algorithms calculate geometric objects that divide the *x*-*y* plane into parts. E.g., if the geometric object is a circle, the two parts are the *inside* and the *outside* of that circle; if the geometric object is a straight-line, then again, there two parts, one on each side of the line.

4) **Framework:** An abstraction in which software providing generic functionality for a broad and common need can be selectively refined by additional user-written code, thus enabling the development of specific applications, or even additional frameworks. In an object-oriented environment, a framework consists of interfaces and abstract and concrete classes.
5) **Graphical User Interface (GUI):** An interface that allows users to interact with the application through visual indicators and controls. A GUI has a less intense learning curve for the user, compared to text-based command line interfaces. Typical controls and indicators include buttons, menus, check boxes, dialogs, etc.
6) **IEEE:** Institute of Electrical and Electronics Engineers, is a professional association founded in 1963. Its objectives are the educational and technical advancement of electrical and electronic engineering, telecommunications, computer engineering and allied disciplines.
7) **Instance:** A 2-dimensional data point comprising a *x*-value and a *y*-value. An instance always has a name, which serves as its unique identifier, but it may be labeled or unlabeled.
8) **Software Design Description (SDD):** A written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the project.
9) **Software Requirements Specification (SRS):** A description of a software system to be developed. It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide. This document, for example, is a SRS.
10) **Unified Modeling Language (UML):** A general-purpose, developmental modeling language to provide a standard way to visualize the design of a system.
11) **Use Case Diagram:** A UML format that represents the user's interaction with the system and shows the relationship between the user and the different *use cases* in which the user is involved.
12) **User:** Someone who interacts with the DataViLiJ application via its GUI.
13) **User Interface (UI):** See *Graphical User Interface (GUI)*.

1.4 References

[1] IEEE Software Engineering Standards Committee. "IEEE Recommended Practice for Software Requirements Specifications." In *IEEE Std. 830-1998*, pp. 1-40, 1998.

[2] IEEE Software Engineering Standards Committee. "IEEE Standard for Information Technology – Systems Design – Software Design Descriptions." In *IEEE STD 1016-2009*, pp.1-35, July 20 2009

[3] Rumbaugh, James, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, The*. Pearson Higher Education, 2004.

[4] McLaughlin, Brett, Gary Pollice, and David West. *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*. O'Reilly Media, Inc., 2006.

[5] Riehle, Dirk, and Thomas Gross. "Role model based framework design and integration." In *ACM SIGPLAN Notices*, Vol. 33, No. 10, pp. 117-133. ACM, 1998.

1.5 Overview

This software requirements specification (SRS) document will clearly define the operational capabilities of DataViLiJ and its UI functionalities and aesthetics. Note that this document is not a software design description (SDD), and it does not include design components that use UML or specify how to build the appropriate technologies. It is simply an agreement concerning what to build. The remainder of this document consists of two chapters, appendices, and the index. Chapter 2 provides the general factors affecting the application requirements. It also provides the background for these requirements, putting the application in perspective with other related applications such as the algorithms that will be used. Chapter 3 provides the software requirements to a level of detail sufficient to allow the design of the application to satisfy those requirements. This includes the GUI and the core operations. Here, every input into the application and every output from the application,

along with all the functions, are described. The appendices include sample data that illustrates the data format, and background information on the type of algorithms.

2. Overall Description

Imagine you are a traveler who has never tasted a certain exotic fruit *X*, and upon reaching a far-away tropical island, see this fruit for the first time. You don't know how to pick the tasty ones, of course. But based on your experience with other fruits, you know that the color and softness are good indicators of taste. So, you start tasting randomly, and gradually, you figure out what color and softness you want when you go shopping for this fruit. AI algorithms work in a similar way. If the color and softness are somehow both represented in the [0,1] interval, then every instance of *X* can become a 2-dimensional data point. If the instance is known to be tasty, then it can have a label called "tasty" (or "somewhat tasty", "not tasty", etc.). Based on such a representation of the fruits, the algorithms can *learn*. Then, even if you throw a new fruit at the algorithm, it can figure out whether or not it is likely to be tasty. As mentioned in Sec. 1.3, such algorithms are beyond the scope of this project. As far as we are concerned, any such algorithm is *iterative* (the iteration being the machine's equivalent of a human tasting one fruit at a time and getting a better understanding with more experience).

Now imagine you are a student of AI who wants to understand *how* these algorithms learn. For that, it would be helpful to see how these algorithms manipulate the data they are learning from. This is very similar to how we gain a better understanding of, say, a sorting algorithm, when we see a running animation of that algorithm manipulating its input array of data. With this motivation in mind, the rest of this section will the whole system.

2.1 Product Perspective

The complete system will consist of an algorithm component, and the DataViLiJ component which will provide the dynamic visualization of the algorithm's use and manipulation of the data. Since this project will interact with the output of the algorithms, the communication will happen through a dedicated interface. Further, since this is a data-centric application, the data needs to be stored somewhere in a predetermined format that is understood by the algorithm module as well as the visualization module. For this, files or serializable Java objects will be used wherein the data will be stored in the tab-separated data (TSD) format (see Appendix A). The algorithm will read the data from a TSD file and modify some aspects of it as it runs through multiple iterations. The GUI will allow the user to specify how the modifications will be dynamically displayed on a plot. Fig. 1 shows the major components of the overall system and its interconnections.

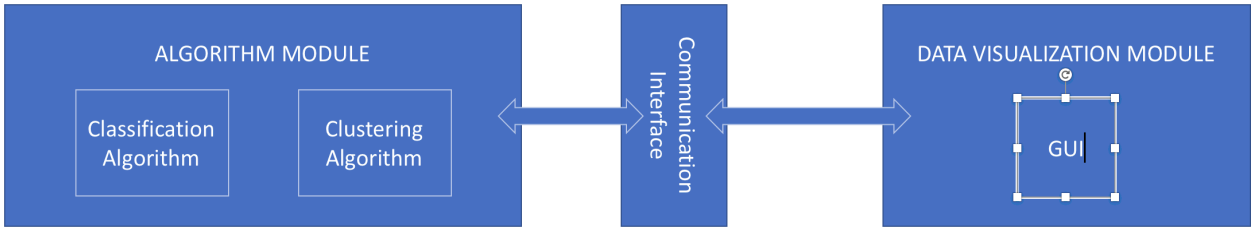


Fig. 1. Block Diagram of Overall System

Note that the algorithm module is not a part of this project, which is restricted to the communication interface and the data visualization module (i.e., DataViLiJ). However, for testing the application, a dummy algorithm should be implemented as part of the application development.

To limit the resource usage by this application, only one algorithm will run on a single data file at any given time. A single data file is expected to be no larger than 2,000 instances.

2.1.1 System Interfaces

This application will have a single GUI as its user interface. The user should, at no point, need to go beyond this GUI to utilize any functionality provided by the application. Even though the primary goal of DataViLiJ is to dynamically display changing data on a *x*-*y* plot, it should allow users to select the data they want to work on, and the algorithm they want to use. The program may thus use a series of linked windows and their corresponding operations to allow the user to perform the entire process:

(1) Select a tab-separated data file.
(2) Select the algorithm type (classification or clustering).
(3) Select a specific algorithm from the type selected above.
(4) Choose stopping criteria (e.g., the maximum number of iterations for the algorithm chosen).
(5) Interval at which the plot data will be dynamically updated (e.g., every 5 iterations).

Any algorithm used as part of this application will have the following characteristics (see Appendix B):

a. It may change the labels of some or all data points.
b. It will not change the actual (x, y) position of any data point.
c. Classification algorithms will add a single straight line to the data, but clustering algorithms will not.

2.1.2 User Interface

This is a desktop application. It will, therefore, allow users to interact with a mouse and a keyboard. Table 1 below provides the complete list of high-level use cases illustrating how a user will interact with the application.

Use Case	GUI Context	Use Case Title
1	Primary Window	Start Application Use
2	Toolbar	Load Data
3	Toolbar	Create New Data
4	Toolbar	Save Data
5	Primary Window	Select Algorithm Type
6	Primary Window	Select Algorithm
7	Configuration Window	Select Algorithm Running Configuration
8	Primary Window	Running an Algorithm
9	Primary Window	Export Data Visualization as Image
10	Primary Window	Exit Application

Table 1. List of Use Cases

Section 3 provides further details for each use case in Table 1, including the corresponding GUI descriptions (Sec. 3.1) and functional requirements (Sec. 3.2).

Note that each use case must be carefully designed by the use of UML class diagrams and UML sequence diagrams in the SDD document.

2.1.4 Hardware Interfaces

The DataViLiJ application should be deployable and runnable on any desktop device with a keyboard, a mouse, and a screen. This is subject to the limitations of the software interface of such a device. Note that touch screen capabilities are not exploited by this application, but this additional functionality of the screen should not affect the performance or any functionality of this application.

2.1.3 Software Interfaces

The DataViLiJ application is a desktop application developed using Java. Since it is a data visualization application, employing the ViLiJ framework is highly recommended. The software and operating environment required for its development, testing, deployment, and running are:

(1) Java Standard Edition Development Kit: Oracle JDK8u161 or higher
(2) Operating System (one of the following):
    a. Windows 10
    b. Mac OS X 10.11 or higher
    c. Ubuntu 16.04 LTS or higher
(3) Visualization Library in Java: ViLiJ

2.1.4 Communication Interfaces

This application does not depend on any external communication interfaces. The only internal communication interface is the interface between the algorithm and data visualization modules, as shown in Fig. 1. This internal interface is important, but its internal details – i.e., *how* it achieves the communication – is not relevant to the end requirements of this software.

2.1.5 Memory Constraints

The application is limited only by the memory constraints set upon it by the hardware (e.g., the device's RAM) and software (e.g., the memory limitation of the Java Virtual Machine).

2.2 Product Functions

With this desktop application, the user will be able to see *how* an AI algorithm gradually figures out simple patterns in the given data, and thereby 'learn' from that data. The user shall be able to select the data they want to work with, and the algorithm they want to see at work. Further, the user shall also be able to create their own data if they want to.

The running of the algorithm will be highly configurable. The software will allow the user to specify the stopping criteria of the run, the intervals at which the data display will be dynamically updated, and even provide play/pause capabilities for the algorithm run.

At any point of such a run, the user shall be able to export the data visualization as an image.

2.3 User Characteristics

There is only one type of user at whom this application is aimed: undergraduate computer science students with an interest in AI. The data visualization GUI should therefore be extremely user friendly so that the user's attention is not taken away from the core objective of understanding the data and the algorithms. Further, the GUI design should ensure that the aesthetics (e.g., choice of color, icon, tooltip, etc.) make the software use intuitive, and the learning curve is very gentle.

3. Specific Requirements

This section lists all the factors that affect the requirements stated in this SRS. These factors are not constraints on the design, but rather, aspects whose change will require changes in the SRS itself. As such, these are the details the software must adhere to. Further, this SRS includes data-specific and algorithm-specific details in its appendixes; they must therefore be considered as part of the software requirements.

3.1 External Interfaces

This section provides a detailed description of all inputs into and outputs from the system. It also provides a basic prototype of the user interface.

3.1.1 User Interfaces

Use Case 1: Start Application Use

Upon starting the application, the user shall see the main screen, which serves as the primary window of the GUI. Initially, it should only contain the toolbar and an empty area for the data display (i.e., a JavaFX chart or plot). The window size and decorations are to be finalized by the UI designer. The designer should also select appropriate icons and tooltips for each toolbar button. Fig. 2 shows an example of such a window.

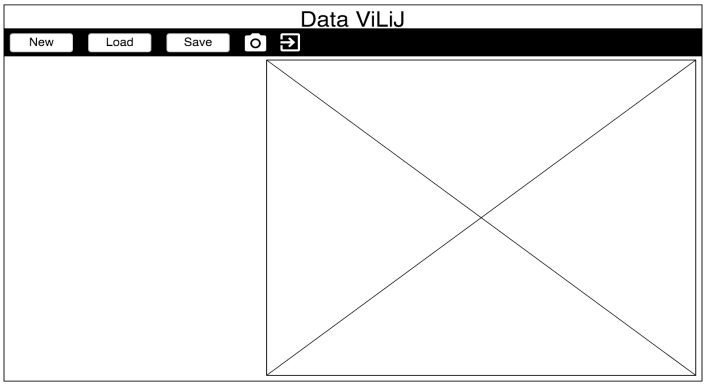


Fig. 2. The primary window at the start of a DataViLiJ session.

Goal in Context	N/A
Precondition(s)	The application has been started and the user is viewing the initial primary window
Scenario	The user is viewing the primary window of the application
Exceptions	N/A
Priority	Critical. Blocks the rest of the software development.
Availability	First Benchmark
Frequency of use	Once per session.
Open Issues	Size, style, and other window decorations to be finalized by the UI designer.

Use Case 2: Load Data

The user shall be able to load valid data from a file in the TSD format by clicking the Load button in the toolbar.

If the data is valid, then the top 10 lines will be shown in a read-only text box. Additionally, the following information about the data will be displayed: (a) the number of instances, (b) the number of labels, (c) the label names, and (d) the source of the data (Fig. 2). If the data is invalid, then an appropriate error dialog shall be shown to the user.

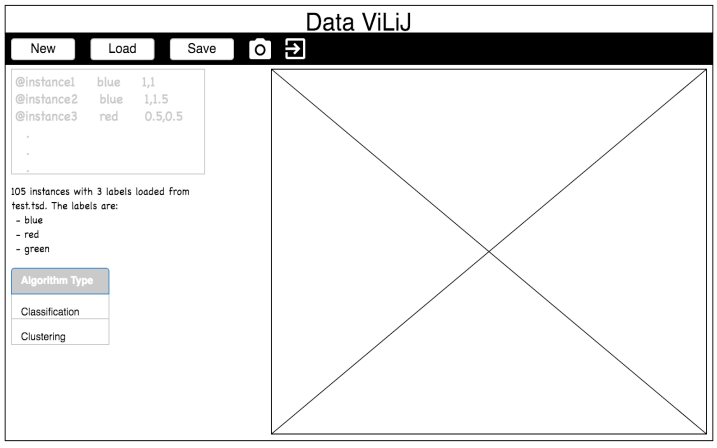


Fig. 2. Primary window after valid data is loaded by the user.

Once the data is loaded, the GUI shall also allow the user to select the type of algorithm to be run. At this point, it should be noted that instances need not always have a label. The **`null`** label name is reserved for this purpose. This label must not be considered a usual label. Further, the type of algorithm is often decided by the type of

data. Classification algorithms require two non-null labels. Therefore, the user shall not be able to choose 'classification' if the loaded data does not meet this criterion.

Goal in Context	Load data
Precondition(s)	The application has been started and the user is viewing the initial primary window
Scenario	The user has clicked the button to load data from a TSD file.
Exceptions	If the data is invalid, the user shall be notified using a dialog with appropriate error message.
Priority	Critical. Blocks the rest of the software development.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	Size, style, and other decorations pertaining to the text area, information display, and selection of algorithm type to be finalized by the UI designer.

Use Case 3: Create New Data

The user shall be able to create new data by clicking on the New button in the toolbar. This use case is similar to the previous, except that the text area will be active to allow the user to type in data in the TSD format. Here, the GUI primary window must provide the user with toggle that tells the program when the creation of new data is finished (e.g., a button that toggles between "Done" and "Edit"). If the data is valid, information about the data must appear just like in Use Case 2. Similarly, the user must also then be able to choose an algorithm type. Additionally, once the user indicates that data creation is finished, the text area must become disabled (and in this example, the control change from "Done" to "Edit").

Goal in Context	Create New data
Precondition(s)	The application has been started and the user is viewing the initial primary window
Scenario	The user has clicked the button to create new data.
Exceptions	If the data created is invalid, the user shall be notified using a dialog with appropriate error message upon indicating that data creation is finished.
Priority	Essential. Must be implemented.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	Size, style, and other decorations pertaining to the text area, information display, data creation toggle control, and selection of algorithm type to be finalized by the UI designer.

Use Case 4: Save Data

If the user chose to create new data, then s/he shall be able to save it by clicking the Save button in the toolbar. This button must remain disabled while there is no data to be saved, which can happen when (a) there is no created data, or (b) there is no change in the created data since it was saved the last time. The data must be saved in a TSD format (e.g., by restricting the extension in the FileChooser).

Goal in Context	Save data
Precondition(s)	The user is viewing the primary window and has created data that remains unsaved.
Scenario	The user has clicked the button to save data.
Exceptions	N/A
Priority	Essential. Must be implemented.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	The default location (if any) for saving the data to be finalized by the developer.

Use Case 5: Select Algorithm Type

Once there is valid data in the application, due either to the loading of valid data or to the creation of new data, the user shall be able to select one of two following types of algorithm: (a) classification, or (b) clustering.

However, based on the limited scope of this project, classification can only be done if there are exactly two labels. Therefore, if this condition is not met, the user must not be able to make this choice. This may be implemented by, say, disabling the 'classification' option. Upon selecting the algorithm type, the primary window shall display a list of algorithms of that type to the user. From this list, the user must be able to select the actual algorithm to run.

Recall that the algorithms are not a part of this project, and upon completion of the project, actual algorithms will be integrated. For testing, however, the developer may need to build a mock classification and a mock clustering algorithm. Examples of two such mock algorithms (one for each type) are provided in Appendix C.

Goal in Context	Select algorithm type from (a) classification, or (b) clustering.
Precondition(s)	The user is viewing the primary window and has loaded/created valid data. If the data was created by the user, then the user has already indicated that the creation process is finished.
Scenario	The user has accessed the control to select the type of algorithm.
Exceptions	N/A
Priority	Essential. Must be implemented.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	The type of control (as well as its size, style, location and other decorations) for the algorithm type selection to be finalized by the UI designer.

Use Case 6: Select Algorithm

Upon selecting the algorithm type, the user shall be able to select an actual algorithm from that category. A prototype of this is shown in Fig. 3. Note that there is a configuration/settings icon next to each choice of the algorithm. This is the control for the algorithm's run configuration, described next in Use Case 7. The actual running of the selected algorithm with the data happens after the user sends the signal to run the algorithm through the appropriate GUI control. In Fig. 3, this control is the 'Run' button, whose action is described in Use Case 8.

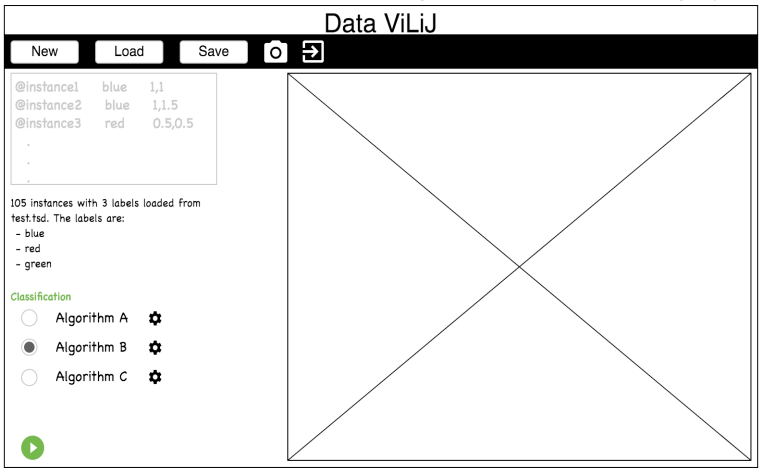


Fig. 3. Primary window after the algorithm selection.

Goal in Context	Select algorithm.
Precondition(s)	The user is viewing the primary window and has selected the type of algorithm to run.
Scenario	The user has accessed the control to select the algorithm.
Exceptions	N/A
Priority	Essential. Must be implemented.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	The type of control (as well as its size, style, location and other decorations) for the algorithm selection to be finalized by the UI designer.

Use Case 7: Select Algorithm Running Configuration

Every algorithm shall be made available with a 'run configuration'. This configuration must be filled in by the user when the algorithm is run the first time. Then onward, this configuration shall be used as the default setting for that particular algorithm. The configuration settings shall be provided by the user on a separate Configuration Window, as shown in Fig. 4. The minimal configurations that must be provided by the user are (a) the number of iterations of the algorithm, (b) the interval, specified in terms of the number of iterations, at which the data will be dynamically updated in the primary window's display area (e.g., every '5' iterations), and (c) whether or not the algorithm should automatically run till its end or wait for the user to indicate that it should continue. The third option is a valuable tool for the user since if the algorithm runs till the very end, the user gets to see an animation of how the data changes throughout the algorithm's run time. The disadvantage, however, is that the user cannot export any *intermediate* image. On the other hand, if the user tells the algorithm to dynamically update the display but wait for the user's indication before continuing, then the overall process is much slower, but the user can spend more time studying the changes and export even those charts that show the intermediate state of the data *during* the algorithm's run.

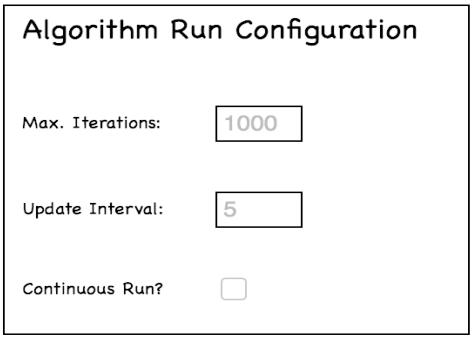


Fig. 4. An algorithm's run configuration

Note that different algorithm types may require different parameters for their run configuration. Appendix B lists these parameters for clustering and classification algorithms.

Goal in Context	Configure how the algorithm runs in DataViLiJ.
Precondition(s)	From the primary window, the user has successfully selected an algorithm to run.
Scenario	The user has accessed the control to select the algorithm.
Exceptions	None. If any configuration is invalid, the software shall handle it gracefully by designing robust methods in the back-end. The values decided in such cases must be reflected in the GUI (e.g., if the user chooses the update interval to be -5, it is reset to 1; and this reset is shown in the GUI).
Priority	Essential. Must be implemented.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	The type of control (as well as its size, style, location and other decorations) for the algorithm selection to be finalized by the UI designer.

Use Case 8: Running an Algorithm

If an algorithm is selected and its run configuration has been provided, then the user shall be able to run this algorithm on the available data. Fig. 3 shows a 'Run' button as a possible GUI control for this action.

If the run configuration has disabled continuous run (i.e., the user must tell the algorithm to resume after every update interval), then the GUI control must make this obvious (e.g., by toggling between enable/disable of the run button). If the run configuration has enabled continuous run, then the GUI control to run the algorithm must remain disabled until the current run is finished.

A Note on Data Modification

Note that even though these algorithms may often manipulate the data, this must not be reflected in the original data that was provided to them. That is, the user must be able to retain the original data for, say, future runs of this or another algorithm.

- *Only the display in the primary window shall be updated based on what the chosen algorithm does.*

Goal in Context	Run the chosen algorithm.
Precondition(s)	From the primary window, the user has successfully selected an algorithm to run.
Scenario	The user has accessed the control to run the algorithm.
Exceptions	If the run configuration does not exist (e.g., when the user is running the chosen algorithm for the very first time), a dialog must appear and ask the user to finish setting up the run configuration before performing the current action.
Priority	Essential. Must be implemented.
Availability	First Benchmark
Frequency of use	Multiple per session.
Open Issues	The type of control (as well as its size, style, location and other decorations) for the running of the algorithm to be finalized by the UI designer.

Use Case 9: Export Data Visualization as Image

The screenshot/camera icon on the toolbar shall remain disabled as long as the chart/plot display is empty. However, since this application allows algorithms to run in a manner where it stops every few iterations (i.e., when the 'continuous run' is disabled from the run configuration), it shall be possible for the user to export the display during these *pauses* of the algorithm. While the algorithm is running (i.e., it is not paused), the export option must remain disabled irrespective of the run configuration's choice of the continuous run parameter.

The export action shall only export the visualization as an image, and not anything else in the GUI.

Goal in Context	Export the data visualization as image.
Precondition(s)	The user has successfully been able to run an algorithm.
Scenario	The user has clicked the toolbar icon to export the current display as an image.
Exceptions	None. If the display is empty or the algorithm is currently running, the toolbar control must remain disabled.
Priority	Essential. Must be implemented.
Availability	Second Benchmark
Frequency of use	Multiple per session.
Open Issues	The type of control (as well as its size, style, location and other decorations) for the export, as well as the quality and file type parameters for the exported image, are to be decided by the UI designer.

Use Case 10: Exit Application

The user shall be able to exit the DataViLiJ application at any time, including while an algorithm is running. However, if there is any unsaved data or an algorithm has not finished running, an appropriate warning dialog shall be provided to the user. Note that there are two different types of exceptional conditions here depending on whether or not there is (a) unsaved data, and (b) an algorithm currently running. These two exceptions are included in the table below.

Goal in Context	Exit the application.
Precondition(s)	The application is currently running.
Scenario	The user has clicked the toolbar icon to exit the application.
Exceptions	1. If there is any unsaved data, the user is shown a dialog with a three-way choice: (a) exit anyway and lose the data, (b) do not exit and return to the application, or (c) exit, but save the data first. 2. If there is an algorithm currently running, the user is shown a dialog with a two-way choice: (a) terminate the algorithm right away and close the application, or (b) return to the application.
Priority	Essential. Must be implemented.
Availability	Second Benchmark
Frequency of use	Multiple per session.
Open Issues	The type of control (as well as its size, style, location and other decorations) for the export, as well as the details of the dialogs and messages for the two exceptional cases are to be decided by the UI designer.

3.2 Functional Requirements

The functional requirements have been included along with the UI requirements in Section 3.1. Throughout the design and development of this software, it must be stressed that every aspect must be robust and adhere to the fundamental concepts of foolproof programming.

3.3 Design Constraints

JavaFX will be used because it effectively leverages each system's available rendering technologies and provides platform independence for personal computers.

3.4 Software System Attributes

As professionals, all members of this project must take this project seriously. We are dedicated to producing robust software that exceeds the expectations of our customers. In order to achieve this level of quality, we should build a product with the following properties in mind:

3.4.1 Reliability

The program should be carefully planned, constructed and tested such that it behaves flawlessly for the end user. Bugs, whether due to the data or the GUI, are unacceptable. In order to minimize these problems, all software will be carefully designed using UML class and sequence diagrams. A **design-to-test** approach should be used for the implementation.

3.4.2 Availability

Upon completion of both the algorithms module and the data visualization module, and the subsequent integration testing, the product will be available for download for no financial cost.

3.4.3 Security

All security mechanisms and concerns will be addressed in future revisions of this product.

3.4.4 Extensibility

More complex AI algorithms will be added in future. These algorithms will remove the restrictions on classification methods leading only to linear geometric objects (i.e., straight lines). The design must therefore consider algorithms generating polynomials of higher degree and their visualization in the GUI.

3.4.5 Testability

The application must be testable with mock algorithms of each type. As such, the design must allow for separate testing of an algorithm under several different run configurations and data inputs.

3.4.6 Maintainability

The block diagram in Fig. 1 must be adhered to. This will allow the easy decoupling of the algorithms from the GUI through a narrow communications interface. Since this software will run both modules simultaneously, the components that require thread-safety must be carefully identified for easy maintenance and testing.

Update mechanisms will be addressed in future revisions of this product.

3.4.7 Portability

This application has requirements that make it portable across all desktop devices that meet the software requirements in Sec. 2.1.3. This is due, in particular, to the use of Java as the programming language, making it completely host-agnostic. Mobile platforms running Java are, however, not currently being targeted.

3.5 Organizing the specific requirements

This application is simple enough that we need not worry about using an alternative arrangement of the content of this document. The specific requirements for this application already fit neatly into the sections listed in the IEEE's recommended SRS format.

4. Supporting Information

This document should serve as a reference for the designers and developers in the future stages of the development process. Since this product involves the use of a specific data format and specialized algorithms, we provide the necessary information in this section in the form of three appendixes.

Appendix A: Tab-Separated Data (TSD) Format

The data provided as input to this software as well as any data saved by the user as a part of its usage must adhere to the tab-separated data format specified in this appendix. The specification are as follows:

1. A file in this format must have the "**`.tsd`**" extension.
2. Each line (including the last line) of such a file must end with '**`\n`**' as the newline character.
3. Each line must consist of exactly three components separated by '**`\t`**'. The individual components are
    a. Instance name, which must start with '**`@`**'
    b. Label name, which may be **`null`**.
    c. Spatial location in the *x*-*y* plane as a pair of comma-separated numeric values. The values must be no more specific than 2 decimal places, and there must not be any whitespace between them.
4. There must not be any empty line before the end of file.
5. There must not be any duplicate instance names. It *is* possible for two separate instances to have the same spatial location, however.

Appendix B: Characteristics of Learning Algorithms

As mentioned before, the algorithms and their implementation, or an understanding of their internal workings, are not within the scope of this software. For the purpose of design and development, it suffices to understand some basic characteristics of these algorithms. There are provided in this appendix.

B.1 Classification Algorithms

These algorithms 'classify' or categorize data into one of two known categories. The categories are the labels provided in the input data. The algorithm *learns* by trying to draw a straight line through the *x*-*y* 2-dimensional plane that separates the two labels as best as it can. Of course, this separation may not always be possible, but the algorithm tries nevertheless. An overly simplistic example where an algorithm is trying to separate two genders based on hair length (*x*-axis) and count (*y*-axis) is shown in Fig. 5 (picture courtesy dataaspirant.com). **The output of a classification algorithm is, thus, a straight line.** *This straight line is what is updated iteratively by the algorithm, and must be shown as part of the dynamically updating visualization in the GUI.* Moreover, a classification algorithm will NOT change

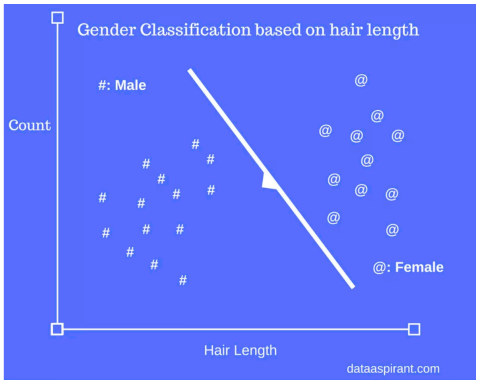


Fig. 5. Output of a classification algorithm

- **the label of any instance provided as part of its input,**
- **the actual (*x*, *y*) position of any instance in its input.**

The run configuration of a classification algorithm should comprise the maximum number of iterations (it may, of course, stop before reaching this upper limit), the update interval, and the continuous run flag. These are shown in Fig. 4.

B.2 Clustering Algorithms

Clustering algorithms *figure out patterns* simply based on how data is distributed in space. These algorithms do not need any labels from the input data. Even if the input data has labeled instances, these algorithms will simply ignore them. They do, however, need to know the total number of labels ahead of time (i.e., before they start running). Therefore, their run configuration will comprise the maximum number of iterations (it may, just like classification algorithms, stop before reaching this upper limit), the update interval, the number of labels, and the continuous run flag. **The output of a clustering algorithm is, thus, the input instances, but with its own labels.** *The instance labels get updated iteratively by the algorithm, and must be shown as part of the dynamically updating visualization in the GUI*. This, along with the importance of providing the number of labels in the run configuration, is illustrated in Fig. 6 (a) and (b) below.

Fig. 6 (a). The input data need not have any labels for a clustering algorithm. Shown here as all instances having the same color. The number of labels decides what the output will look like.

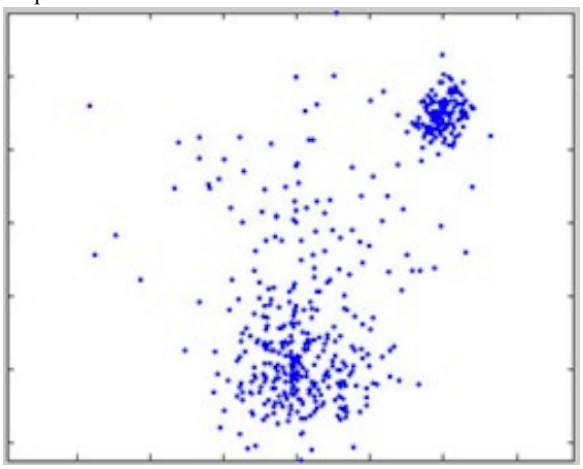
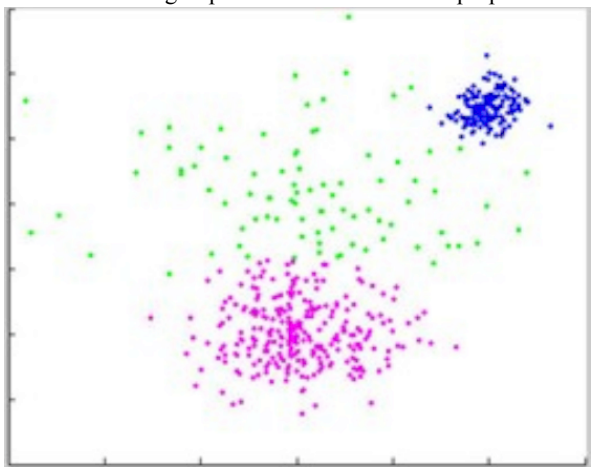


Fig. 6 (b). If the number of labels provided to the algorithm as part of its run configuration is 3, then this is what its output might look like. If the run configuration specified 2 labels, the green instances would get split between the blue and purple ones.



Just like a classification algorithm, **the actual position of any instance will not change**.

Appendix C: Mock Algorithms for Testing

For testing, it suffices to create simple mock algorithms.

For classification algorithm testing, simply writing a test code that creates a random line every iteration is sufficient. To plot such a line in the GUI output, this algorithm can, for example, return a triple `(a, b, c)` of integers to represent the straight-line $ax + by + c = 0$, where a, b, and c are randomly generated using standard Java classes like `java.util.Random`.

For clustering algorithm testing, simply writing a test code that takes as input the number of labels (say, n) along with the data, and randomly assigns one of the labels $\{1, 2, \ldots, n\}$ to each instance.