UNIVERSIDAD REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR

DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2016/2017

Trabajo Fin de Grado

Interaction Concepts and Data Structures for the Visual Data Classification in Projections of nD Data

Autor: Pablo Pérez Martínez

Tutores: Manuel Rubio Sánchez, Dirk Joachim Lehmann

Abstract

Data classification is one of the main fields Machine Learning. The aim of this project is developing an extension to a program of data visualization so that the user or expert can classify the data based on their knowledge.

It is part of a data visualization application (developed by Dr Dirk Joachim Lehmann) that projects multidimensional data into two dimensions (like points). From this projection it is offered to separate them by the user. To do this, the user can utilize the following options:

- Drawing: The user can draw class-separating (ellipses, rectangles, lines, curves, polylines and polygons) to separate the points you see on the screen.
- Manipulating: The user is allowed to manipulate these separators by either erasing, or repainting, moving or filtering the results.

Once the data has been labeled, the user has two more possibilities:

- Store classification: The user will save the classification in a text file in .csv format that assigns to each data, the class to which it belongs to. This data can be used later if necessary.
- Re-use the labeling for a later classification: The user may utilize labeling in certain moments to classify data of the same number of variables. This could be used so that having a reliable model they can classify new data from a database with the same data type.

Keywords

Multidimensional projection, halfspaces, classification, labelling, visualization, high dimensional data, class-separating objects, Star Coordinates.

Resumen

En el aprendizaje automático (“Machine learning”), uno de los campos principales es la clasificación de datos. Este proyecto consiste en desarrollar una extensión a un programa de visualización de datos de manera que el usuario o experto pueda clasificar los datos a partir de su conocimiento.

El punto de partida es una aplicación de visualización de datos desarrollada por el doctor Dirk Joachim Lehmann que proyecta datos multidimensionales en forma de puntos en dos dimensiones. A partir de esta proyección se ofrece separarlos al gusto del usuario. Para ello el usuario puede hacer uso de las siguientes opciones:

- Dibujar: Se permiten dibujar separadores (elipses, rectángulos, líneas, curvas, polilíneas y polígonos) para separar los puntos que se observan en la pantalla.
- Manipular: Se permite al usuario manipular estos separadores ya sea borrando, repintando, moviendo o filtrando los resultados.

Una vez los datos han sido clasificados el usuario podrá elegir una de las siguientes opciones:

- Almacenar la clasificación: El usuario guardará la clasificación en un archivo de texto en formato .csv que asigna a cada dato la clase a la que pertenece. Estos datos se podrán utilizar posteriormente para lo que sea necesario.
- Reutilizar un etiquetado para una posterior clasificación.

El usuario puede utilizar un etiquetado de un momento determinado para clasificar datos del mismo número de variables. Esto tendría la utilidad de, teniendo un modelo fiable, poder clasificar nuevos datos de otra base de datos siguiendo los mismos patrones

Palabras clave

Proyecciones multidimensionales, subespacios, clasificación, etiquetado, visualización, datos de varias dimensiones, objetos separadores, Star Coordinates.

Index

1. Introduction

In this era of internet and big data, data analysis, visualization and classification are some of the usual topics.
A few decades ago, data classification was a manual and costly process. With the advent of machine learning and data classification techniques, the performance of this process was improved. In addition, the data visualization allowed having a simpler and more familiar access to the manipulation and the analysis of data. Especially when users have a lot of data.

There are many solutions for data classification problem, and its number is being increased. One of the data classification options is to rely on an expert's knowledge of data visualization. This solution involves three phases:

- Find appropriate projections to separate the data best.
- Interactively, separate the data to assign them an appropriate label.
- Based on this separation, describe a description rule how further data could be classified automatically.

Firstly, the users find a good visualization that represents groups of data. This can be done using 2D projections to display multidimensional data in a recognizable way. For example, we can see the display shown in Figure 1.1. These groups will highlight a specific property of the dataset that the user prioritized when doing the analysis.

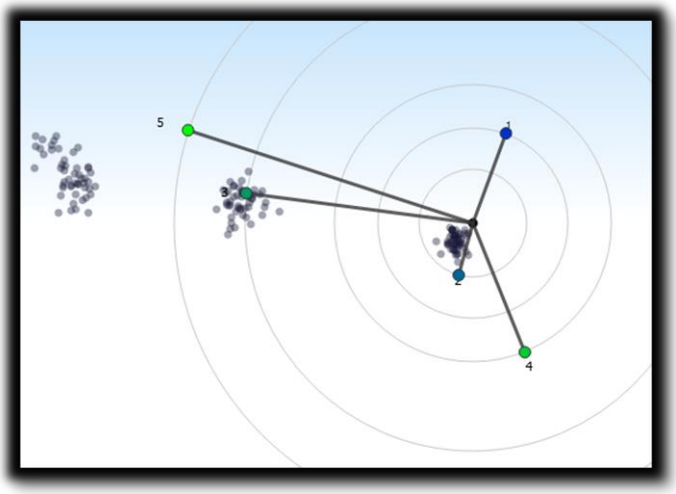


Figure 1.1: Example of significant projection.

Secondly, the domain expert should be able to set those points belonged to the same class. For that objective is important to have an interactive visual interface. For example it would be easy to understand that users could encircle points that, from their point of view, belong to the same class. (Figure 1.2)

Finally, to finish this process, one must take the knowledge of the expert and apply it by finding a rule to classify new points or data.

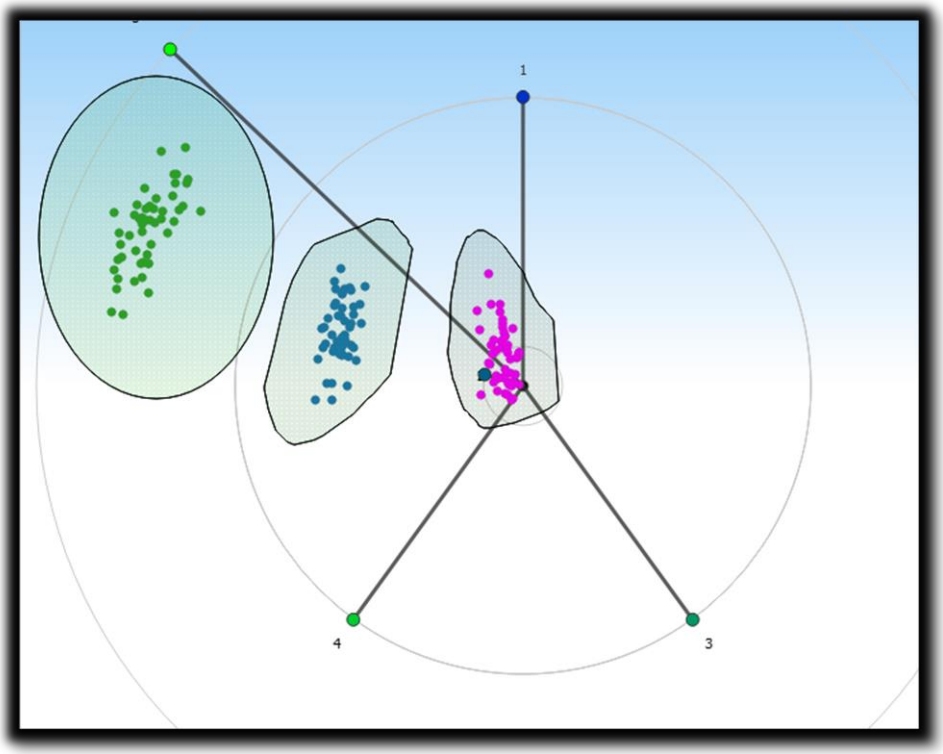


Figure 1.2: Encircled points of each class.

This Bachelor's thesis is presented as part of a bigger project developed by Dr Dirk Joachim Lehmann.

In particular this software projects data (data stored in files for instance in *.csv*). Based on matrix operations it does a projection converting (thanks to a projection operation) a N x M matrix in a M x 2 matrix which numbers represent the position of the point.

This software allows to perfectly finding projections where the data are separated when evaluating their characteristics.

At this point it is not possible to label data and classify data so it would be positive to implement an extension to the program for this purpose.
Starting to make this extension is the main aim of this Bachelor's thesis, focusing on the second step (labelling or classifying interactively data).
To start the project the student is provided with a document containing the initial objectives as well as hints on how to do it.
The most relevant contents of this document are included below:

Subtasks for this Thesis:
In this section, the subtasks of this work are explained and summarized within a work schedule.
Get ready (1 October- 20 November): Setup the Developer System and become familiar with C++

- At first, the developer system for the framework MPVID02 has to be set up (Figure 1.3). For this, a developer system description can be found in the project management framework Trello.
- Basic knowledge about C++ is mandatory. The bachelor-candidate should become familiar with C++ via tutorials and private study before starting to implement the interaction tasks for visual classification into the framework.

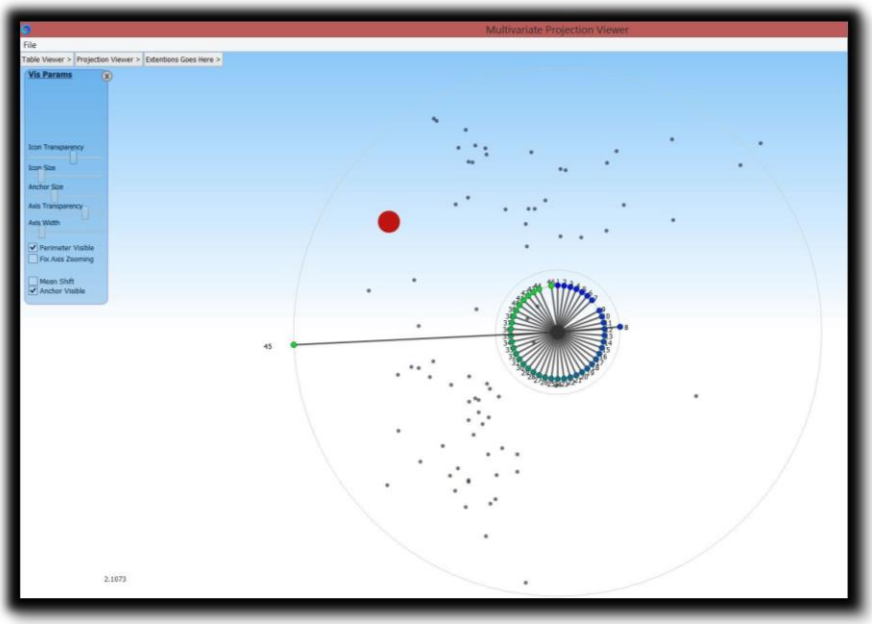


Figure 1.3: Initial situation.

Interaction Implementation (21 November -21 January):

- Implementation of class-separating interaction schemes, as lines, curves, polygons etc.
- Class Assignment: All data elements that share the same half-space belong to the same class.
- Implementation of functions for editing and deleting of formally drawn class-separating objects (Which would need an update in Class Assignment due to the changes introduced might have changed halfspaces).

Save/Load-Function Implementation (21 January- 10 February):

Implementation of Save function in order to save the interactively conducted classification on storage. It is desired to store:

- the classification result.
- the underlying projection matrix A on that this classification is built on.

The final UI might look like similar to Figure 1.4, where a toolbox provides the different interaction schemes.

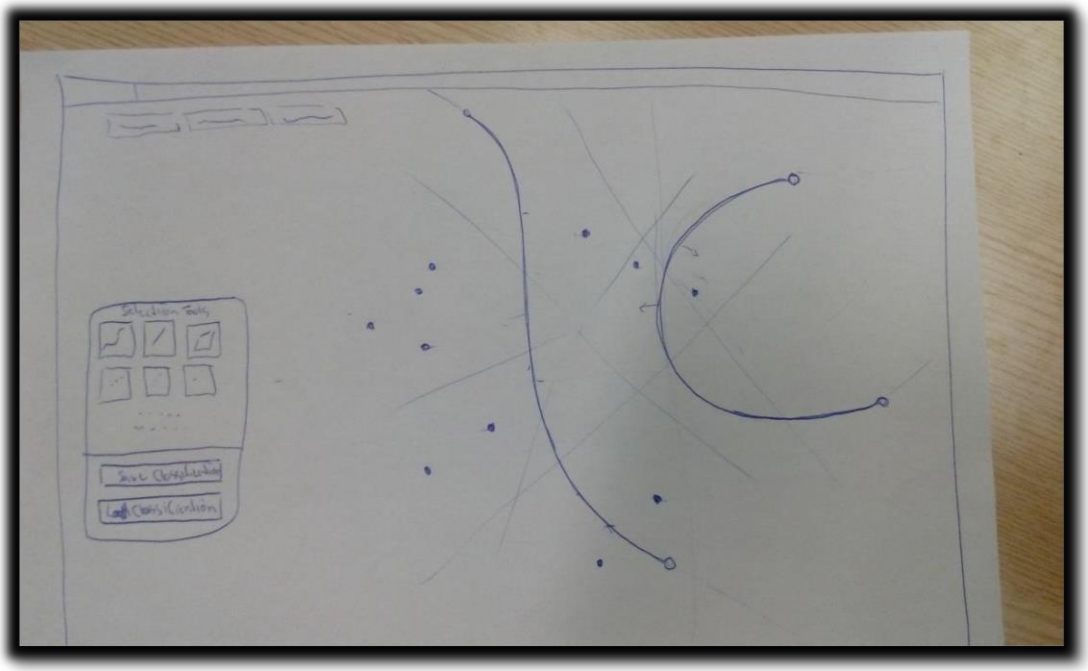


Figure 1.4: Desired application.

Write Bachelor's thesis (10 February - 25 March):

It should be written in English and it has to comply strictly with regulations of TFG.
This subtasks and schedule will guide the development of this extension. As a medium-term project there are changes in needs as well as suggestions not foreseen in advance. Below is the most important suggestion set during the time of development:

Implementation of new classifying methods based on a previous labelling (done manually):

It is needed to store the classification result, design an algorithm which classifies data based on the classification result and display this new classification on the screen.

For this it has been decided to use two approaches: the Euclidean classification (distance to centroids), and Voronoi regions classification (nearest neighbour). This would be detailed throughout the document.

2. Objectives, state of the art and methodology

In this chapter there is a description of the research question, the state of the art and the methodology used in this Bachelor's thesis.

2.1 Research question

Data classification is a traditional task in the field of machine learning. This assignment consists on assigning different classes and labelling data according to certain discrimination properties.
In the case of visual data classification the recognition of visual patterns by the user could be used to classify the data.

The process of classifying data according to a domain expert knowledge has three phases. The first involve finding a good projection where the expert can identify differences between data. The second one consists on giving the user the chance to separate data according to what was previously observed. The last phase would be to classify data based on the labelling that has been made deducting a description rule.

Dr. Dirk Joachim Lehmann's software covers the first phase allowing the interaction with data. The users are able to change the projection in order to find the best projection which separates data. This let the users find data behaviour patterns. For example: given a database of diseases, recognise how many people are likely that they are infected.

Once data are visually separated, saving and classifying is required. As this software does not allow this, it is beneficial to develop this functionality.
This Bachelor's thesis focuses on the second phase leaving a slightly introduction to the third.

One possible solution to this problem would involve allowing the following aspects:

- Painting class-separating objects.
- Manipulating this objects.
- Labelling the data in each projection to assign them an appropriate class.
- Saving a classification.
- Classify new data based on a previous labelling.

Painting class-separating objects

To paint class-separating it is needed to choose which different drawings are the best to separate the projected points. Being formally drawn class-separating objects, it is logic to use most functionality of drawing programs as MS Paint (which is well known and easy to use).

The basic drawing objects in MS Paint are as follows (Figure 2.1): Line, curve, ellipse (or circle), rectangle (or square), round rectangle and polygon. Others like arrows, clouds, or hearts do not seem to be so relevant for this purpose.

Figure 2.1: MS Paint drawing buttons.

Other class-separating object that could be useful is a polyline.

Eventually, apart from these classifiers, it is extremely useful and precise to draw a free form polygon without clicking every single point on the screen (just moving the mouse). This will let the user draw a precise region quickly.

Giving a familiar environment would help with the usability of the program. The benefit of basing the painting operations on well-known software would be that this new one would be user-friendly and easy to use.

Manipulating these objects

To edit, manipulate or delete objects is mandatory in order to correct the mistakes made by the user. These operations would improve accuracy when it comes to sorting data. Furthermore, this would reduce the time in case of making a mistake by not having to start from scratch.

Having drawn these objects, the question arises as to how they should be manipulated in case of wanting to change the position or size (the actual situation). Three operations are needed: move, edit (repaint) and delete.

Move operation:

The move operation would make able to drag an object and drop it where it is wanted. In case of being one over another there should be any other possibility to move the one which is below.

Edit operation:

It could happen that the drawn object is not as precise as the user would want. It may be smaller or bigger so there should be the possibility to change the dimensions or to repaint it.

Delete operation:

It is needed to delete one object or all of them. This is a normal operation so it must be very easy.

In addition to these operations there could be a filtrating operation that leaves a single class.

Being user-friendly an easy to understand would be important as while painting objects. That is why buttons ought to be familiar (for example as in Figure 2.2).

Figure 2.2: Descriptive buttons of moving, repainting and deleting (respectively).

Labelling the data in each projection to assign them an appropriate class

To classify the projected data points depending on which region or subspace these are located is the most important thing on this project. This should be made when a new class-separating object is painted or when one is edited or deleted.

One of these regions is defined as a part of the Cartesian plane enclosed by classifiers. All the points that share the same half-space belong to the same class.

As the simplest example, if there is only one line placed on the screen, there could be two different classes (Figure 2.3). However if all the data are projected to the same side of the line there would be only one class (Figure 2.4).

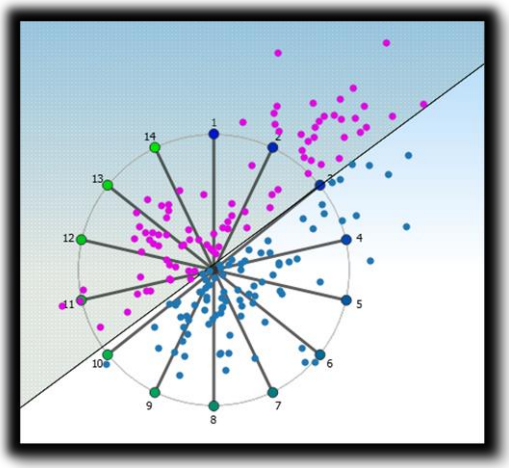


Figure 2.3: One line, two classes.

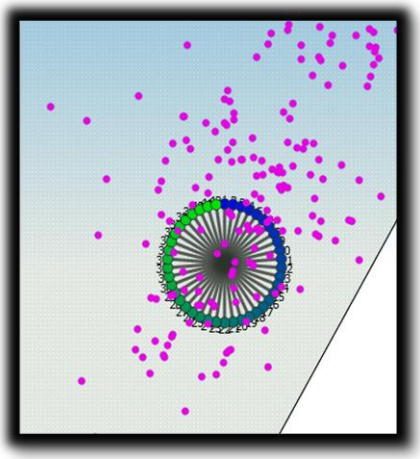


Figure 2.4: One line, one class.

The idea is to have as many different classes as different half-spaces are on the screen excluding the ones which do not contain any projected point.

For example in (Figure 2.5) there are some regions that do not contain any points so there are only six groups of points instead of the number of regions (twelve).

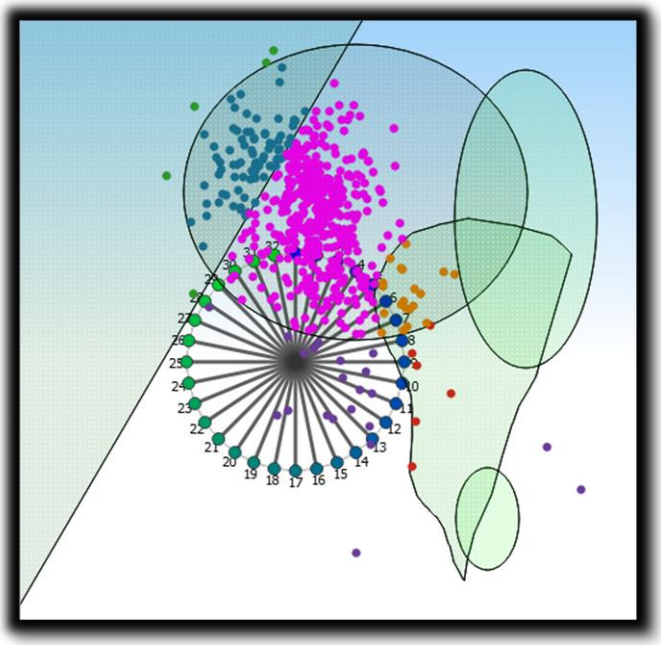


Figure 2.5: Classification.

The program should re-label when any edition operation is made. When the user moves a class-separating object some points might change their class. The same applies in case of deleting and editing.

This classification must be stored in a data structure which allows to know to which class each point belong.

Saving a classification

To save a classification means that after painting the objects and got the different classes the user can save it. If there is a classification stored, it would be possible to share it, to contrast with others… Another benefit is that the user could reload a classification to change it if needed.

Classify new data based on a previous labelling

One of the aims of data classification is to classify automatically data. This is a machine learning task that it is introduced in this section.

When a user gets a classification for a projection it should be useful to classify other data based on that classification that could be in any other projection.

When talking about classifying large amount of data time is limited, so if computers can get a good classification mechanically it would help the process.

There are lots of different techniques to sort data, in this Bachelor's thesis there are two introduced:

- Euclidean classification: this means to get some representative points and reclassify data depending on which of this point is projected closer. This would be a good approach when data are separated and classes are well differenced.
- Voronoi based (Nearest neighbour) classification: this approach means to get points of the original classification and reclassify new points based on which is closer. In Voronoi diagrams each point has associated a region within a point will be closer to it. (Figure 2.6)

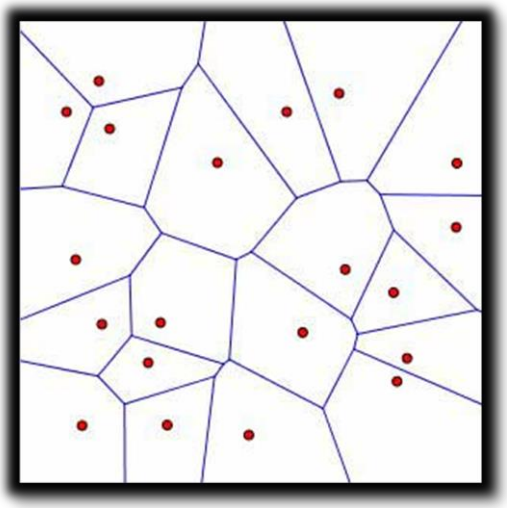


Figure 2.6: Voronoi diagram.

After all of this, the aim would be to get a rule and get a mathematical function that classifies data.

2.2 State of the art

In this section there is an introduction to similar topics.

2.2.1 Data visualization

Data visualization consists in the transformation and subsequent representation of scientific data and abstract concepts in images. A general discussion on visualization methods is found in [5, 10, 26, 34].

Some of the most important contributions of the visualization are the following:

- The density of information per area in an image is significantly greater than that of a text.
- It makes it possible to recognize data behavior patterns
- Represent data of several dimensions or variables at the same time.
- It enables people to interact directly with the data.
- It facilitates the understanding of complicated concepts. (It allows for greater understanding and clarity.)

There are lots of different visualizations to solve different issues. These are some common data visualizations:

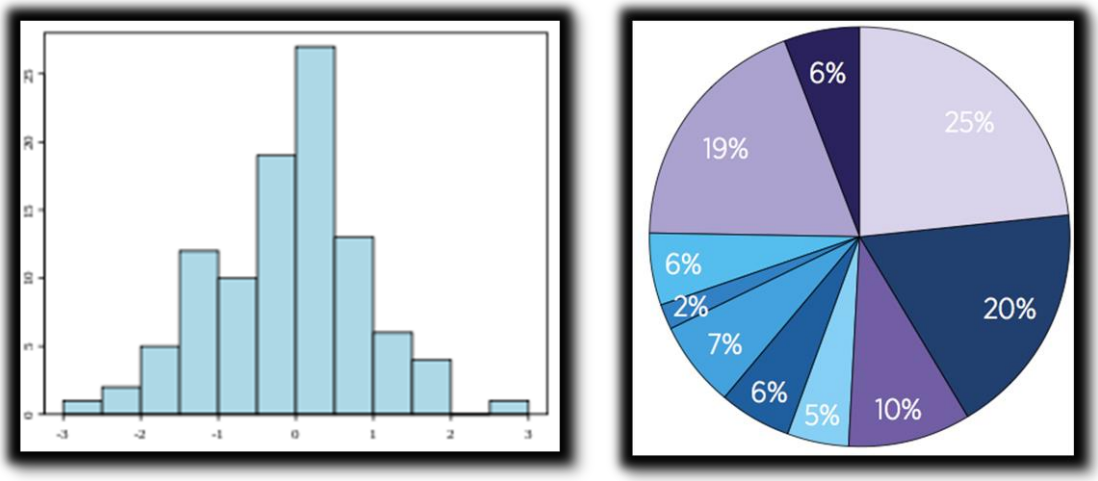


Figure 2.7: Bar Chart and Pie Chart.

Multivariate visualizations transform the search for relationships between variables into a 2-D pattern recognition problem. This simplifies the analysis of the data and, because most of the data has a dimension greater than two, are the most useful visualizations.

Some common multivariable visualizations are: Star Coordinates (Figure 2.10) [18], Parallel Coordinates (Figure 2.8) [16, 34] and Chernoff Faces (Figure 2.9)[33].

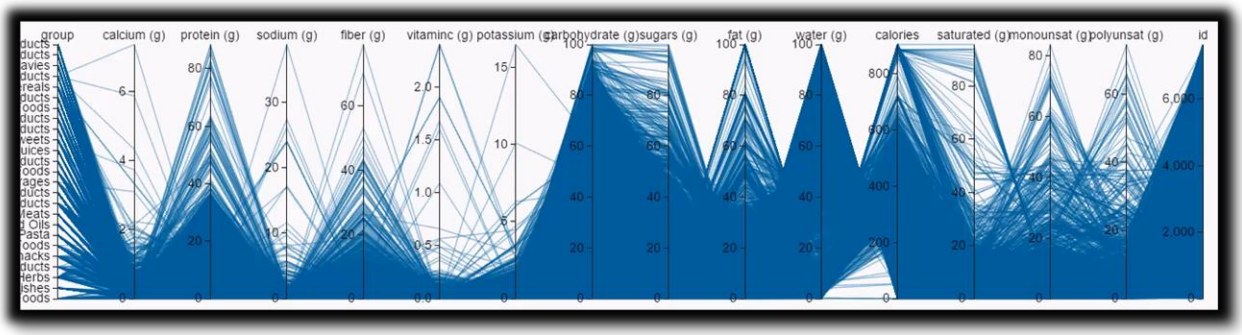


Figure 2.8: Parallel Coordinates.

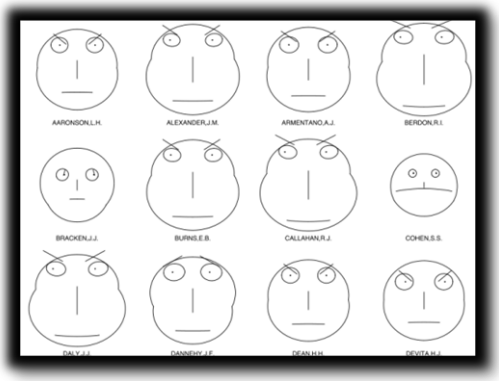


Figure 2.9: Chernoff Faces.

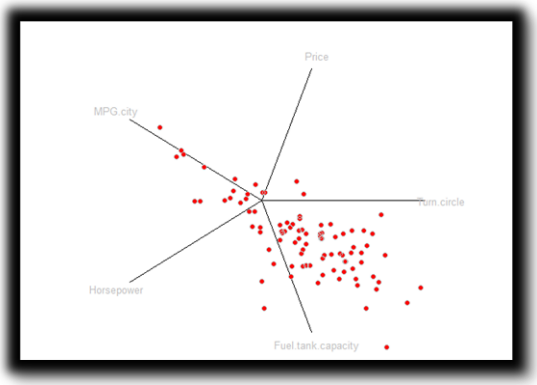


Figure 2.10: Star Coordinates.

There are a considerable number of tools to use and implement new data visualizations. With any graphics library (such as QT or D3.js) they can be implemented.

2.2.2 Data classification

The problem of data classification consists in sorting data in different classes and labelling them. It has numerous applications in a wide variety of mining applications. Some overviews on data classification may be found in [15, 19, 25, 35].

In [21] there is an important dissertation of this issue where given a set of training data points along with associated training labels it is needed to determine the class label for an unlabelled test instance.

Below are some common techniques in Data Classification:

Feature Selection Methods

Probabilistic Methods

Decision Trees [28, 29]

Rule-Based Methods

Instance-Based Learning

SVM Classifiers

Neural Networks [6, 7]

A different way of enhancing the classification process is to use human supervision in order to improve the effectiveness of the classification process. Two forms of human feedback are common, active learning and visual learning.

Active Learning: Most classification algorithms assume that the learner is a passive recipient of the data set, which is then used to create the training model. Thus, the data collection phase is cleanly separated out from modelling, and is generally not addressed in the context of model construction. However, data collection is costly, and is often the (cost) bottleneck for many classification algorithms [8, 31].

In active learning, the goal is to collect more labels during the learning process in order to improve the effectiveness of the classification process at a low cost. Therefore, the learning process and data collection process are tightly integrated with one another and enhance each other. Typically, the classification is performed in an interactive way with the learner providing well-chosen examples to the user, for which the user may then provide labels.

Visual Learning: While active learning collects examples from the user, visual learning takes the help of the user in the classification process in either creating the training model or using the model for classification of a particular test instance. [2, 13, 20, 32]

This help can be received by learner in two ways:

• Visual feedback in construction of training models: the feedback of the user may be utilized in constructing the best training model. Since the user may often have important domain knowledge, this visual feedback may often result in more effective models.

• Diagnostic classification of individual test instances: the feedback is provided by the user during classification of test instances, rather than during the process of construction of the model. The goal of this method is different, in that it enables a better understanding of the causality of a test instance belonging to a particular class.

2.2.3 Similar projects

Star-class:

Star-Class is an interactive visual classification tool that permits clients to visualize multi-dimensional information by projecting points on 2-D using Star Coordinates [11] and allows users to interact with the display to classify data.

Classification operations in a data-mining task are usually performed using decision trees where one class is assigned to each leaf of the decision tree. The popularity of decision trees is due to fast simplicity [3], construction [18] and easy conversion to SQL statements.

Most classification systems are designed for minimal user intervention. However, in [4] is a dissertation to increase user involvement in the classification process, with three important reasons: (1) the use of powerful human pattern recognition capabilities in decision tree construction, (2) the increase in confidence, and (3) the possibility of incorporating domain knowledge to the algorithm.

For instance, the client can see whether certain classes are obviously characterized while different classes have much cover with each other and whether objects in a certain class spread over large spaces or form small clusters [17].

When the user reaches a good projection, the user specifies regions by “painting” over the visualization with the mouse. StarClass allows the user to control the size of the paint-brush for more efficient user interaction. An “Eraser” operation is also provided. Each region is shown with a background with a unique color.

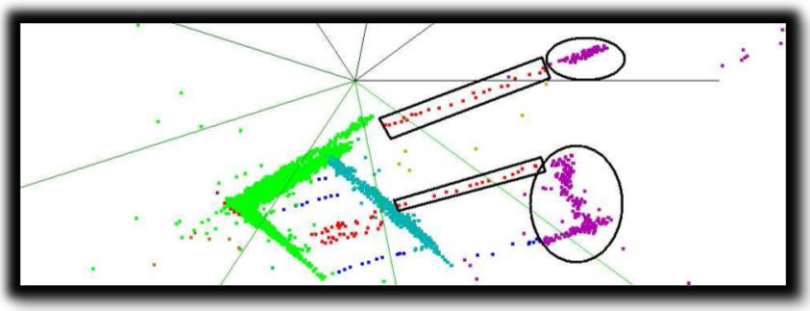


Figure 2.11: Star Class.

2.2.4 Choosing colours

Choosing colours for the different classes that are well differenced is difficult.
The selection of colours must be based on the number of data classes, the nature of their data and the end-use environment [14].
The problem that concerns us involves using a qualitative difference between classes due to the nature of the data. For this task, it is needed to choose the number of data classes (it starts with 20 different possibilities but it can repeat colours in case it is needed). The end-use environment would be the computer screen.

In [14] a tool that solves a certain number of cases when choosing colours **(www.colorbrewer.org)** is proposed**.** Referring to qualitative colour schemes it is marked that qualitative colour schemes (Figure 2.12) rely on differences in hue to create a colour scheme that does not imply order and implies just a difference in kind. Since there is no conceptual ranking in nominal data it is inappropriate to imply order when depicting these data with colour (for example, by using a light-to-dark single-hue sequence). Qualitative schemes work best when hue is varied and saturation and lightness are kept or nearly constant.

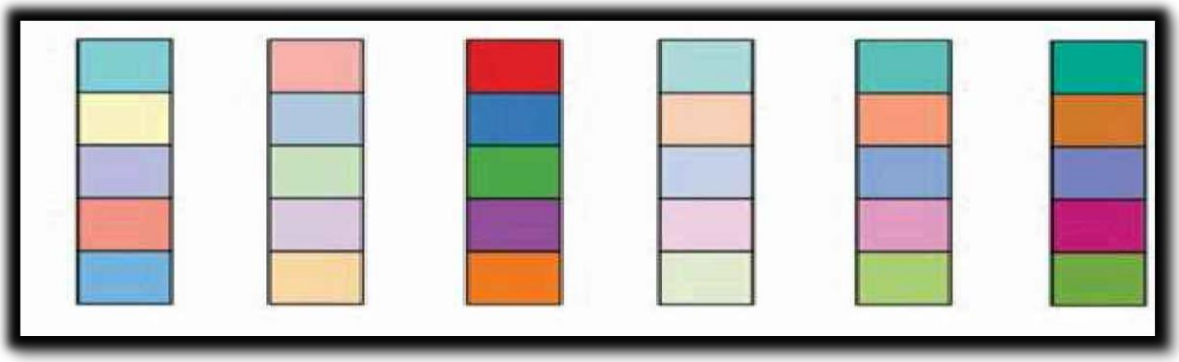


Figure 2.12: Qualitative color scheme.

There is a maximum of twelve colours for qualitative colour schemes in this tool. Those colours would be part of the extension colours. In case it is needed other colours are added.

2.2.5 Voronoi diagrams

Voronoi diagrams are a representation that given some points partitions the plane into regions, called Voronoi regions. This regions are delimitated by points where the point inside is the closest among all the points. (Figure 2.13).That regions let classify based on Euclidean distance.

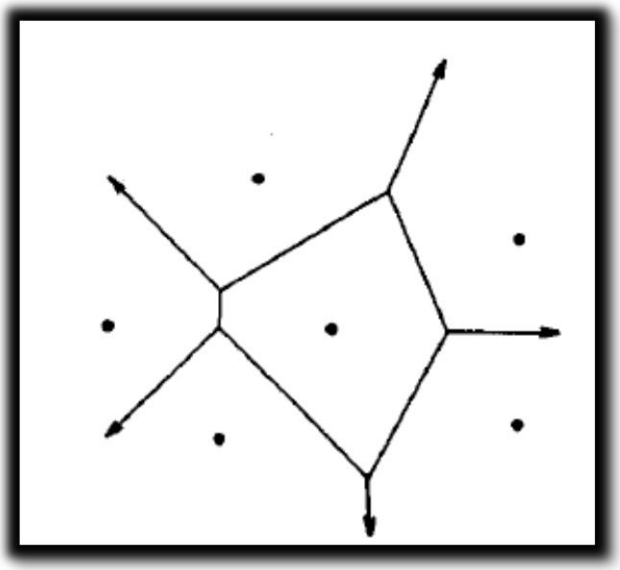


Figure 2.13: Voronoi diagram.

Steven Fortune (1987) introduces a geometric transformation that allows Voronoi diagrams to be computed using a sweepline technique O(n log n).

2.3 Methodology

This region contains the tools used as well as the methods used to achieve the objectives

Tools

The program of data visualisation developed by Dr. Dirk Joachim Lehmann uses some technologies to make this extension most of them have been used:

C++: C ++ is a programming language designed by Bjarne Stroustrup in the 1980's. It is an extension to the programming language C in order to allow the manipulation of objects. From the point of view of object-oriented languages, C ++ is a hybrid language.

As Qt uses C ++ programming language it is desirable to code in this language. Also this structure as an object-oriented language makes easy to understand the code.

In this software it is used to link the frontend programmed with qml (part of Qt) to the rest of the code.

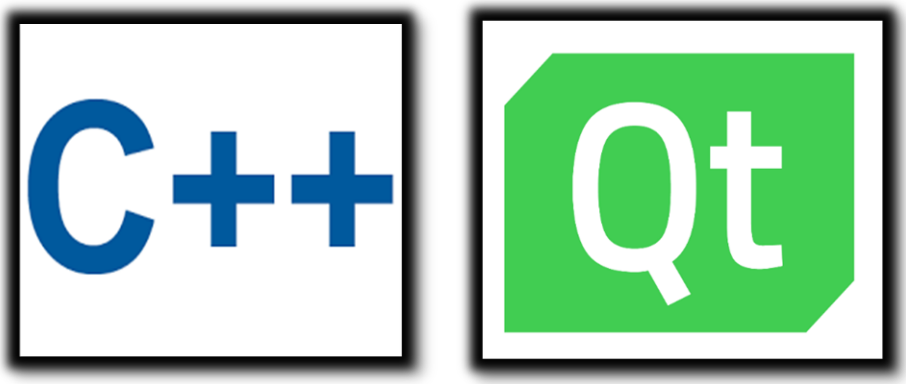


Figure 2.14: C++ and Qt.

QT: Qt [27] is an object-oriented multiplatform framework widely used to develop software using graphical user interfaces. Qt uses the C ++ programming language natively, in addition it can be used in several other programming languages through bindings, for example in Python.

In this project version 4.7 is used to continue with what was used in the initial software.

Qt Quick (that is used in this program) includes a declarative scripting language called QML. This provides a way of building highly dynamic user interfaces.

To display the menus QML is used, and to connect the frontend with the backend Qt and C++.

Others:

Mingw: MinGW (Minimalist GNU for Windows), formerly known as MinGW32, is an implementation of the GCC compilers for the Win32 platform

It is the compiler used to compile the code in C ++ and QtQuick (Qt + QML).

Git: Git is a version of control software designed by Linus Torvalds, thinking about the efficiency and reliability of maintaining application versions when they have a large number of source files. For more information, see [12].

Figure 2.15: Git.

Trello: Trello is a project management software with web interface. It uses the Kanban system to register activities with virtual cards, organize tasks, add lists, attach files, tag events, add comments and share boards. The figure shows an image of a site that shows how accessible and correct it is for project management:

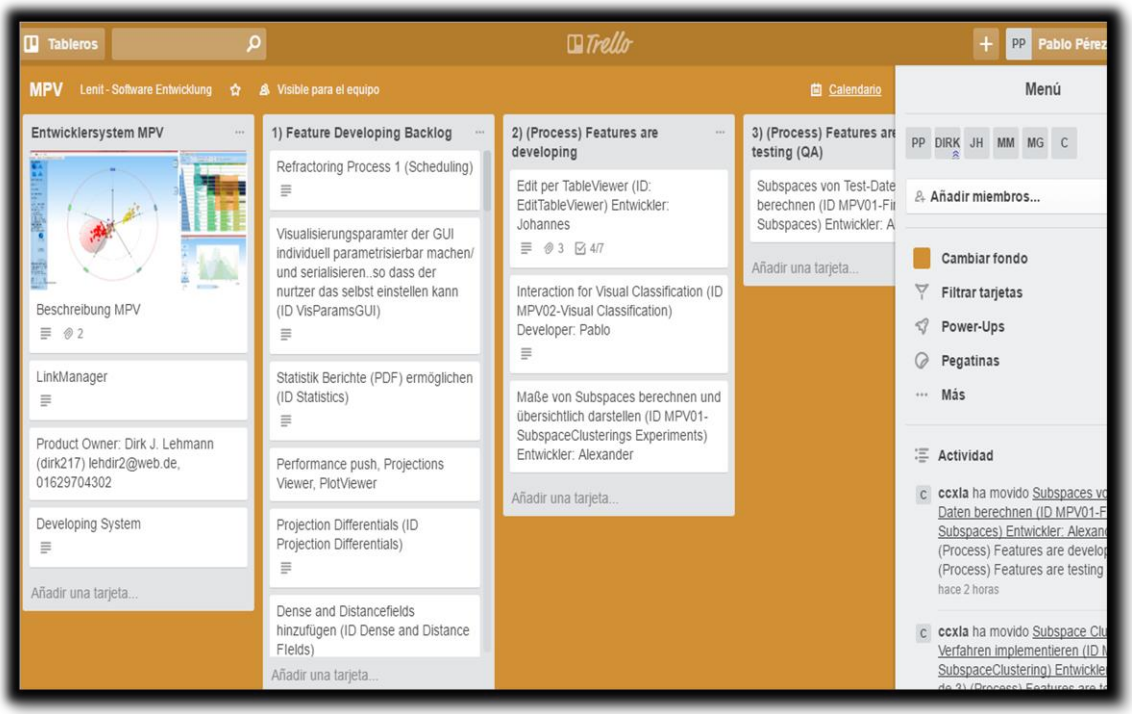


Figure 2.16: Trello example.

In particular, in the different sections the following has been done using the techniques previously explained:

Painting class-separating objects

The methodology used in drawing interaction must be similar to that kind of programs:

In case of lines when the user clicks for the first time the start point is set, then when they move the mouse they can see the line until they release the mouse. In the case of this program that line (between the start point and the release point) is useless, so it is needed to make it longer (until the end of the plane).

While drawing curves happens roughly the same. The user must click three times (the first two to introduce the line and the third to set the curve), after that it is needed to elongate.

Drawing rectangles and ellipses is quite similar. The user sets the first point and the object is bigger or smaller depending on the distance of the release point (while they are moving the mouse, the screen must show how this object is changing in order to let the user know what they are drawing).

Polylines and polygon must be a sequence of clicks. In the case of polylines the first two must set a line to the end of the plane (also the last two). To finish a polygon or a polyline MS Paint uses double click, so to continue the usability criteria of familiarity this must be the way to end this kind of drawn class-separating objects.

There must be a menu where icons describe the objects the users are going to draw.
Similar to this:

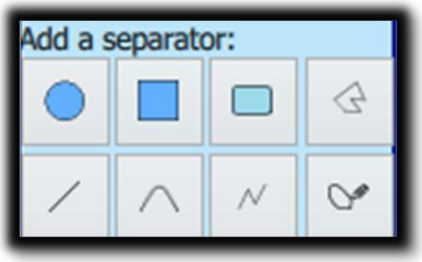


Figure 2.17: Example of separators menu.

Manipulating these objects

The methodology used for editing, manipulating or deleting class-separating objects should be easy to use and user-friendly:

In case of moving there should be the possibility to drag and drop objects. When the mouse is over an object the user should know that and there must be the possibility to drag it by clicking the mouse. The program should calculate the position of the object while it is moving and finally, when it is dropped, reclassify the data.

Deleting could be one of the most used operations so it should be easy to delete a class-separating object. This could be that when users are over one, they can remove it by clicking the right button of the mouse.

Repainting is a not as common operation. When an object is not the desired size, the user should be able to change it.

There ought to be a menu from where all this operations can be done in case the user cannot access to be above a class-separating object. The users could know which one is the separator that they want to change so a menu (figure 2.18) with representative buttons (moving, repainting and deleting) is mandatory.

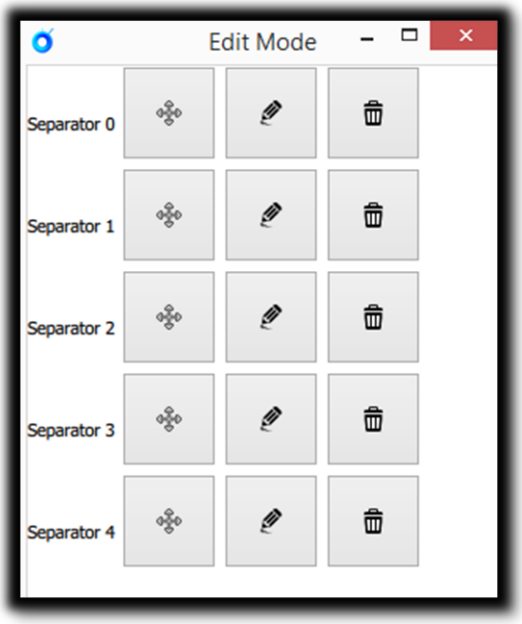


Figure 2.18: Editing menu.

Labelling the data in this projection to assign them an appropriate class

The methodology used for classifying data consists on giving each half-space a class label:

This classification is stored in a Data Structure. In this case it is a list that assigns a point its corresponding class. For example if point 0's class should be class 3, in the list the first entry should be 3.

The process to classify data (detailed in implementation) is as follows:

For each point it is checked if it is in one of the known subspaces. If it is in one it is labelled as belonging to that class. On the contrary if it is not contained in any, the half-space to which belongs is found.

The way to find a half-space is to get the smallest subspace (in the moment that it no longer cuts with other separating objects).

Saving a classification

The methodology used for saving a data classification involves creating files that store this data:
The procedure is to open a file dialog in which users will enter the name of the file where the data will be saved.

The file type would be .txt and there is saved the properties of each point comma the class it belongs. (Figure 2.19)

Figure 2.19: Saved classification.

Classify new data based on a previous labelling

The process of classifying data based on a previous classification has two steps:

-The first step is to set the previous classification (save).

-The second one is to classify data (or new data).

In nearest neighbour approach when the previous classification is stored, representative points are calculated as the media of all the points that are part of a class. After that, when trying to reclassify, each point is labelled with the class of the representative point that is closer. (Figure 2.20)

In Voronoi approach all points are stored and which class they belong to. The way to reclassify data is to label them with the class of the point in which Voronoi region they are.
These calculations will be detailed in the implementation section (as all introduced in methodology).

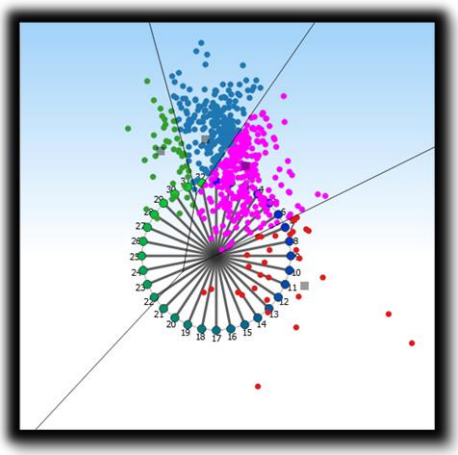


Figure 2.20: Euclidean classification.

3. Computer description

In this section the whole application is described, from the initial situation to the complete development of the extension.

3.1 Initial Situation

In this section there is a description about part of the software designed by Dr. Dirk Joachim Lehmann. This software has been the basis on which the Bachelor's thesis has been performed.

The following are the folders that have been used for the development of the task:

Data: data folders contain files .data, .json and .names. These are used to load data for later projection.

Source/headers: the files this folder contains are part of the C++ language. Each class in C++ has a header file (.h) and a source file (.cpp).

Resource: in this folder there are resources as images (icons used in the project) and other files like main.qml . This file contains in qml (QT Quick MarkUp Language) the visual information of the program (simple UI elements, such as buttons, etc).

Tools: here there are shortcuts to build, start or build and start the program.

Below are the classes where the code is implemented as well as others needed to complete all tasks:

Implementation:

- IFile/IClassification.cpp = Backend: Here, required data structures shall be implemented
- UI/VisualClassification.cpp = Frontend: Here, the interaction shall be implemented (mouse events are caught here, and rendering stuff can be implemented, within the paint() method).

Needed:

- Controller/GlobalControllerObject and Controller/controller = Connect frontend with backend: The access from the frontend to elements of the backend shall be organized via the globally available IController_GlobalControllerObject.
- IMatrix/IMatrix.cpp = data are organised as matrix.
- UI/renderfield.cpp = Frontend: Here, the projection is implemented. (And also the interaction with the axes). (Figure 3.1)
- UI/tableview.cpp = Frontend: Here, the table viewer is implemented.(Figure 3.2)
- IComputationalGeometry/IVoronoi.cpp = needed to draw voronoi diagrams.

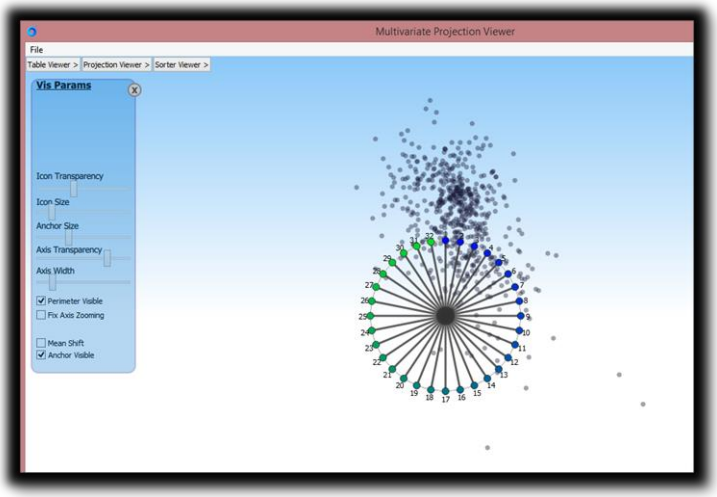


Figure 3.1: Projection.

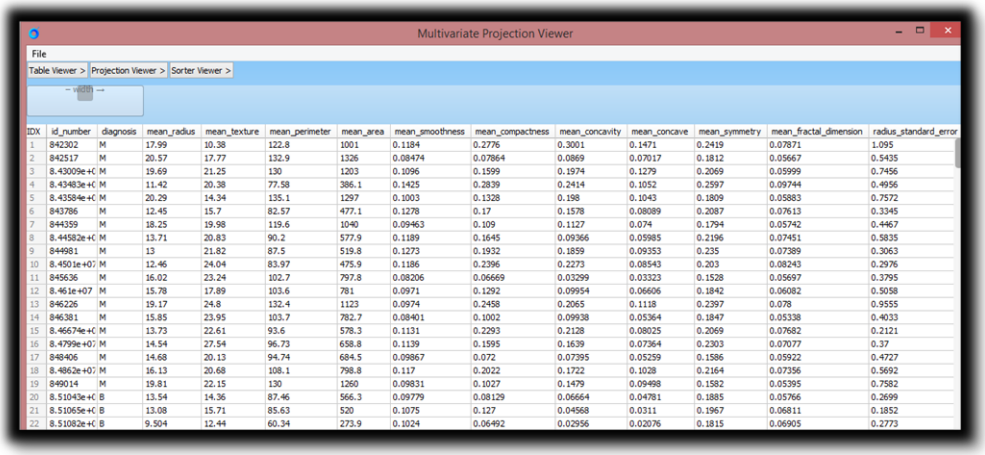


Figure 3.2: Table view.

The figure below includes the class diagram of the part of the program that concerns us:

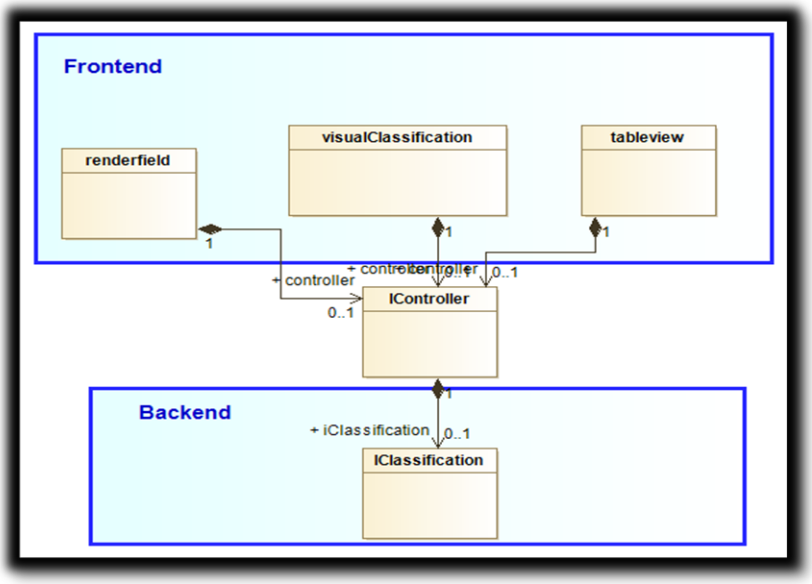


Figure 3.3: Class diagram (Frontend- Backend).

3.2 Requirements

In this section the marked objectives will be transformed into requirements. These requirements will guide the design and implementation of the application.

Functional Requirements:

R1. The user must be able to add the class-separating objects at any time while classifying.

R2. The user must be able to change the position of the objects.

R3. The user must be able to delete the separators.

R4. The user must be able to visualize the results of the classification.

R5. The user must be able to save the classification.

R6. The user must be able to save any current classification in order to use it later for other classifications.

R7. The user must be able to load other data to classify them using a previous version of classes.

R8. The user must be able to access the saved classifications after the application is closed.

Non Functional Requirements:

RN1. The application must have a UI easy to learn and use.

RN2. The visualization of data must have differentiable colours.

RN3. The manipulating operations must minimize the user errors.

RN4. The program must classify data in real time.

RN5. The application must have different ways to interact.

RN6. The model must be intuitive and user-friendly.

RN7. The user should know when is over a separator.

3.3 Analysis

When analysing the questions of the problem we can establish a series of needs that will require the extension of the program.

In the software section where it is developed, a menu must appear in which the different options of the project are separated. There will be several sections: introduce separators, save, reclassify...

In the first place will be necessary the possibility of introducing on screen different separators whether they be lines, curves, circles or ellipses, squares or rectangles, etc.

The separators that have been considered most opportune are the following:

- Lines
- Curves
- Polylines
- Polygons
- Ellipses
- Rectangles

They are simple, well-known and easy to recognize in the menu (Figure 2.17)

Subsequently, once the different separators have been introduced, the data must be automatically classified so that the elements that share the same subspace are classified as part of the same class. The data once classified will be assigned a different color based on the class to which they belong. Each of the different spaces completely

enclosed between lines of separators will be a subspace. This classification must be entered and stored in a simple and correct data structure.

Once you have entered these values (class-separating objects) the user may have the possibility to edit them. To do this, it will be necessary to enable editing and erasing functions. It is positive to create a separate menu in which different editing options can be made as well as being able to do them directly by placing the mouse over the objects.

The main interactive functions to be implemented are the following:

- Change position: moving the classification element is necessary because it may not be exactly where users wanted to have drawn. To do this the interaction should be natural, drag the element by clicking the mouse to the position where the users want to leave the object.
- Repaint: In the event of a serious drawing error, repaint the element. It will be eliminated and it will be allowed to draw one of the same type (for example if it was a polyline it will be allowed to draw a polyline)
- Delete one of the delimiters.
- Delete all objects at the same time.
- Filtrate.

Another option that the software must offer is to save the classification that users have done manually by introducing the different classifiers. To do this a button should open a dialog in which the name of the file in which the users want to save the data is entered. Once done this will automatically save the original data by adding the class to which they belong. In addition the position of the points that have been drawn will be stored in another file so that we can reconstruct the image that is saved.
In this section of the menu also a list of the colors is contributed so that we can know to which class each color of the points refers. And also there is an option to modify projections.

As an extension of the project, there is an option to classify new points based on the Voronoi diagram of the classified points or based on distance to representative points.
This section of the menu should have two steps (Figure 3.4):

-Save the current classification

-Reclassify (if one classification is stored)

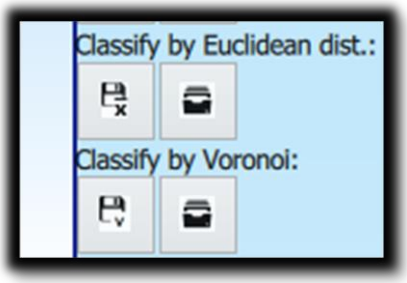


Figure 3.4: Buttons to reclassify (two steps).

The activity diagram below represents this analysis and the way to use the application:

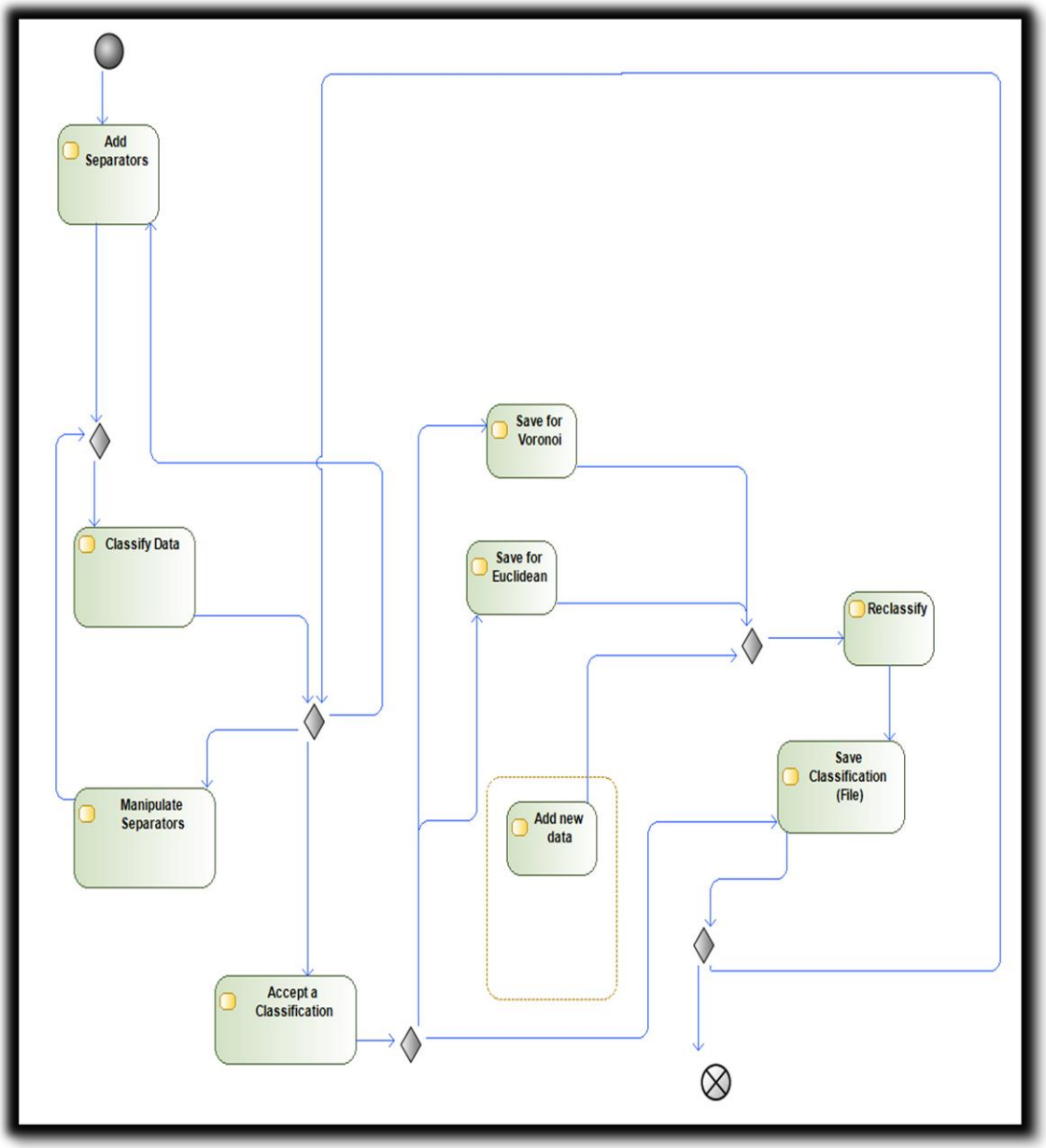


Figure 3.5: Activity diagram.

3.4 Design

For the separators section (Painting class-separating objects and Editing and manipulating this objects) QPainterPath from the Qt library is used. Mouse events are needed to know the points to paint the different objects.

Modifications will have to be made for the different types of classifiers:

- Ellipse: The default function will be used (addEllipse), as in other editing and drawing programs like MS Paint to draw an ellipse are used two points. The first point will fix the starting point of the ellipse and the second point can be moved until it is left to the desired size.
- Rectangles and RoundedRects: The default function will be used (addRect), as in other editing and drawing programs like MS Paint to draw rectangles are used two points. So it is calculated like this:
- Lines: To make lines we will need to close a QPainterPath. A line will be drawn by giving a point (the second will be marked when the mouse is released). As you want the lines to stretch and divide the space into two parts you will need to draw a line going from end to end. To do this, the equations of the lines (figure 3.6) will be used to calculate the points at the ends, keeping the same slope.

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Figure 3.6: Line.

- Polylines: In a similar way to the lines, the first line as well as the last line must be extended. In addition it should be closed eventually depending on how it has started and how it has ended.
- Polygons and drawn Polygons: polygons are a sequence of points. Each new point is joined to the last one. To close these separators a doubleclick event is needed that would join the first point with the last one added.
- Curves: curves would use qpainterpath bezier options. To make it easier for the user it is just the curve what is going to be curve. To elongate a curve tangent is

used. This way some problems as shown in Figure 3.7 are solved. (Green lines would be easier to predict than black while drawing a curve).

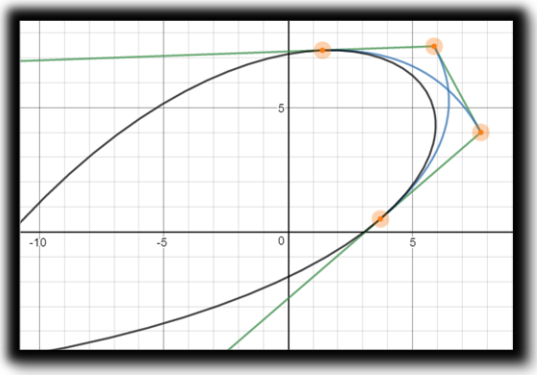


Figure 3.7: Bezier curve.

Editing operations:

Move: To move is needed to know the current position and the next position. For this, it is needed a mouse event (over) and mouse click events which change the position. There must be a mathematical operation that moves the points.

All points of any class-separating objects must be known. When painting the new object it is necessary to calculate the position of the mouse (the distance to the first point of the object), so at the end the mouse ends at the same position over the object.) The intuitive idea is to maintain the distance with the first of the points needed to paint the object and recalculate all the objects based on that distance.

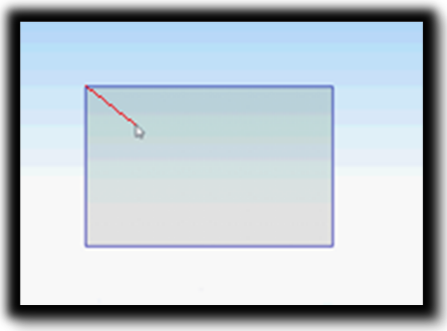


Figure 3.8: Distance mouse-first point.

Repaint: To repaint involves letting paint a new object maintaining the one that is going to be removed. When it is finished the old one is removed.
Delete: To delete an object it should be removed from the data structure that contains them (a list).
Save: The best way to save the classification is preserving their attributes and adding the class. As the original data is a .csv file, the new file would be a .csv where it is appended the class which correspond to the data.
Reclassify: Euclidean classification: The idea is to get a centroid that represents each class. This could be done using the mean. After having this centroids, the points projected must be labelled with the class of the nearest centroid (using Euclidean distance).
Voronoi based classification: The idea is to get every Voronoi region that had the same class. The new points would be classified to the class of the region they are inside.

3.5 Implementation

This section details the implementation of the functionality of this extension.

Drawing class-separating objects

The first subtask is implemented in class UI/visualClassification.cpp (also in UI/visualClassification.h).

In this class painting operations are developed. All this operations ought to be done with the mouse so it is needed to catch these mouse events.

Firstly, when related to drawing class-separating objects, knowing which kind of object is being drawn is mandatory. For this, main.qml has some buttons which are directly connected with some Q_INVOKABLE methods. These methods (rectangle(), roundRectangle(), line(), ellipse(), curve(), polyline(), polygon() and polygonDraw()) set the mode of drawing. If m_mode is -1 it means that the user has not pressed any drawing button.
Secondly, mouse events let the user paint the class-separating object:
mousePressEvent(QMouseEvent *event): when mouse is pressed the position is caught.

mouseMoveEvent(QMouseEvent * event): while mouse is moving it is needed to repaint the class-separators to facilitate drawing.
mouseReleaseEvent(QMouseEvent * event): when a release event occurs the program should act accordingly with what is being drawn.
mouseDoubleClickEvent(QMouseEvent * event): it is used to end polylines or polygons.
After the object is painted m_mode returns to -1.
All separators are saved as a list of Separator (a private class which has the relevant information of the different object).
Depending on what kind of class-separating object it is drawn, it is stored a different QPainterPath,a different QString (for example: "Ellipse","Rect"…) and a different list of points (which length depends on the type).

Manipulating these objects

For editing the user may need to manipulate these classifiers. From main.qml there are some buttons connected to visualClassification.cpp. The methods that are associated to that buttons are moveSeparator(int), repaintSeparator(int), deleteSeparator(int) and cleanSeparators().

- moveSeparator(int): sets edit mode and indicates which classifier is going to be moved. When mouse is released or moved the program catches the event and recalculates the position of the new object in relation to the mouse initial position. (Figure 3.10)
- repaintSeparator(int): sets m_mode to what the object was. For example if it was an ellipse, m_mode is set to ellipse. While the repaint operation, the object that is going to be removed is drawn is red and the new one is drawn in green. After the repaint operation the first object is removed and the new one is added. (Figure 3.9)
- deleteSeparator(int) and cleanSeparators(): deletes the class-separating object referred by the number or all.

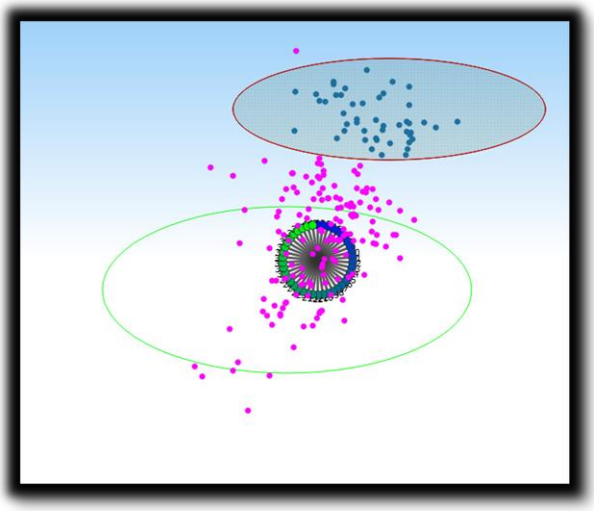


Figure 3.9: Repaint example.

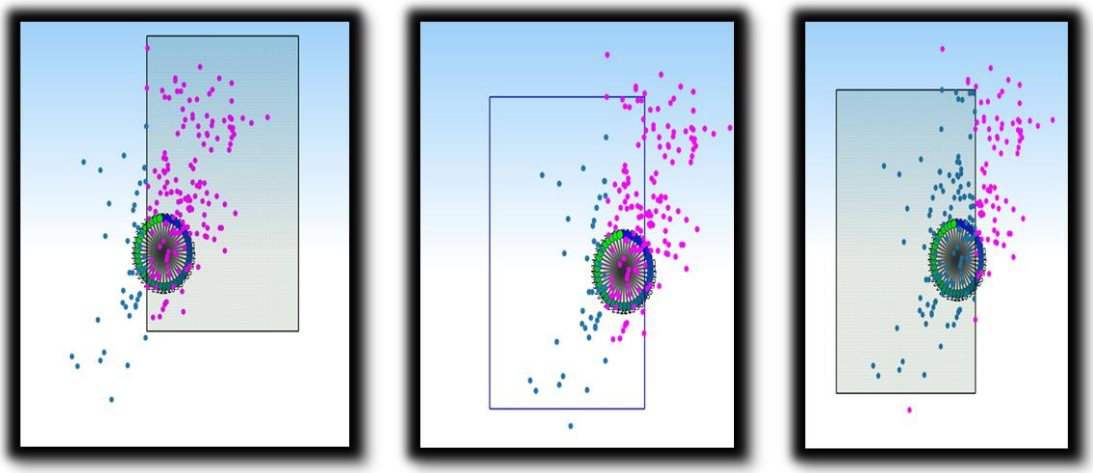


Figure 3.10: Move example.

This manipulating operations can be done while moving the mouse over the class-separating objects. Thanks to hover events the program can notice if the mouse is over a separator. When the user clicks left button the program calls moveSeparator(int) method with the number of the separator the user is over. In case it is right button, it is similar to call deleteSeparator.

The visual interaction for the user is that the class-separating object changes its colour to blue and the number is shown (as it is show in Figure 3.11)

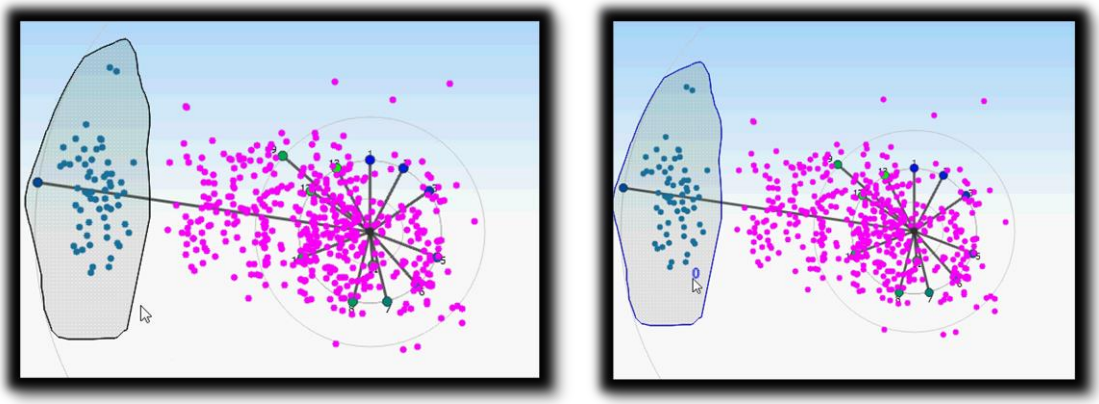


Figure 3.11: Mouse over.

Filtrating operation:

The user can select the class and delete all the other separators. The way to do it is to get the subpath where the user clicks and delete any other paths.

Example:

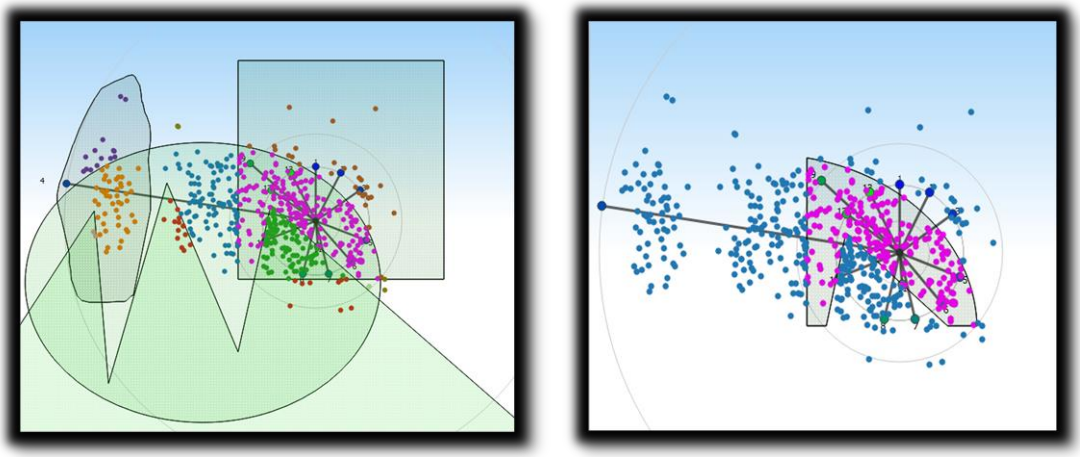


Figure 3.12: Filtrating example.

Labelling the data in this projection to assign them an appropriate class

Classification structures

The data structure implemented is the easiest possible.

It has two lists. There is a list called pathList which has the different half-spaces and a list called pointClass which has the classes associated to each point.

So the first projected point in the projected data matrix would have associated the class of the pointClass.at(0), the second pointClass.at(1) and so on. The class would be associated to the other list. If the point is labelled as class 0 it means that it is contained in the half-space of pathList.at(0).
This structure let access to the class of any point and it is intuitive.
Algorithm:
The algorithm consists of traversing the list of points. If the point is not in one of the already extracted half-spaces, a new subspace (the point's half-space) will be searched and added to the list of half-spaces. If it is in one of the known half-paces, it will be classified in the class which represent that half-space.

The way to get that half-space is to subdivide the plane until it is reached.
Firstly the program gets the top separator that contains the point (it will be the smaller if there are more than one). After that it checks which other separators intersects with it deleting the parts that are not part of the half-space of the point.

Saving a classification

When Clicked the save button, a new file dialog is opened. There the users should write the name where they want to save the current classification. The implementation of this scrolls the initial file and adds a final field with the class to which each record belongs (Figure 3.13). It also saves the points which are projected in that moment (in text files). This points are saved in a file called "nameProjectedPoints" (Figure 3.14).

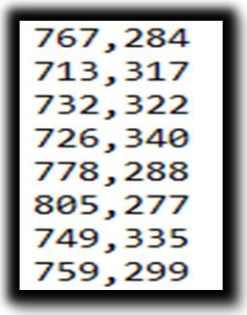


Figure 3.13: Saved file (classification). *Figure 3.14: Saved file (projected points).*

Classification based on a previous visual data labelling

This process has two steps:

- Save the current classification (calculating what is necessary).
- Classify data according to a previous saved classification.

Euclidean:

When save button is pressed, this method calculates the media of each class. Each class would have a representative point.

After this, the user can change the data (maintaining the number of atributes) and re-classify.

The *centroids* projection change if the user changes the projection.

After pressing the classification button the data is classified to the class which represents the nearest point.

This approach is a good approach when the groups of points are together in both the initial display and the display of the new data.

Voronoi based classification:

It is done in two steps, firstly the classification is stored.

The second step, which is in which the new classification is made, has several phases:

- At first it eliminates the points of the previous projection that are projected on the same position.
- With no points in the same projection Voronoi diagram is calculated using Fortune algorithm.
- Once the Voronoi diagram is done, the regions are searched and the points are classified according to the region to which they belong.

4. Validations

4.1 Test

To test the correct operation of the program, the black box tests have been used

Black-box testing is a method of software testing that does not take into account internal structures or workings to examine the functionality. This method of test can be applied virtually to every level of software testing.

Most tests have been done several times with different data and separators.

These are some examples of testing:

Name: Try all classifiers.
Objective: Check that the application does not break when you enter several different classifiers.

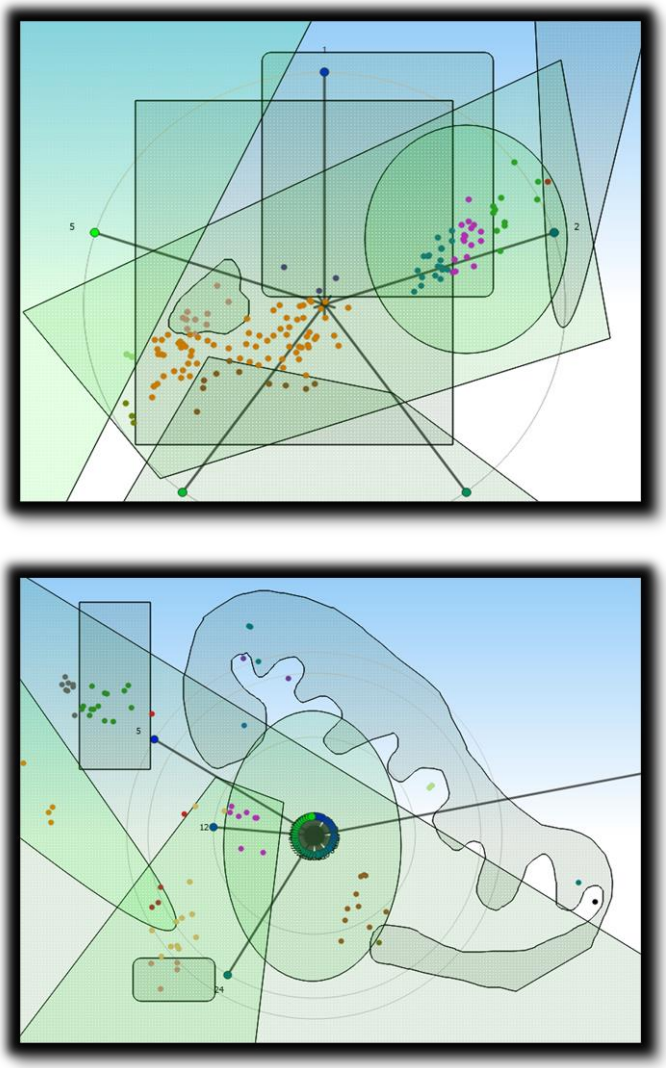


Figure 4.1: Try all classifiers.

Name: Movement
Objective: Check that the application does not break while moving separators and that classifies correctly after moving.

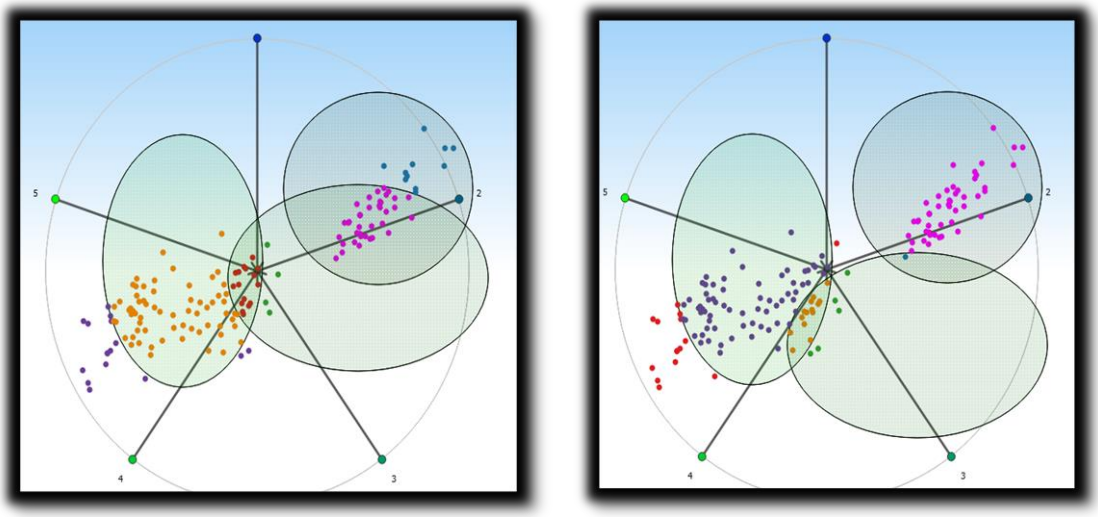
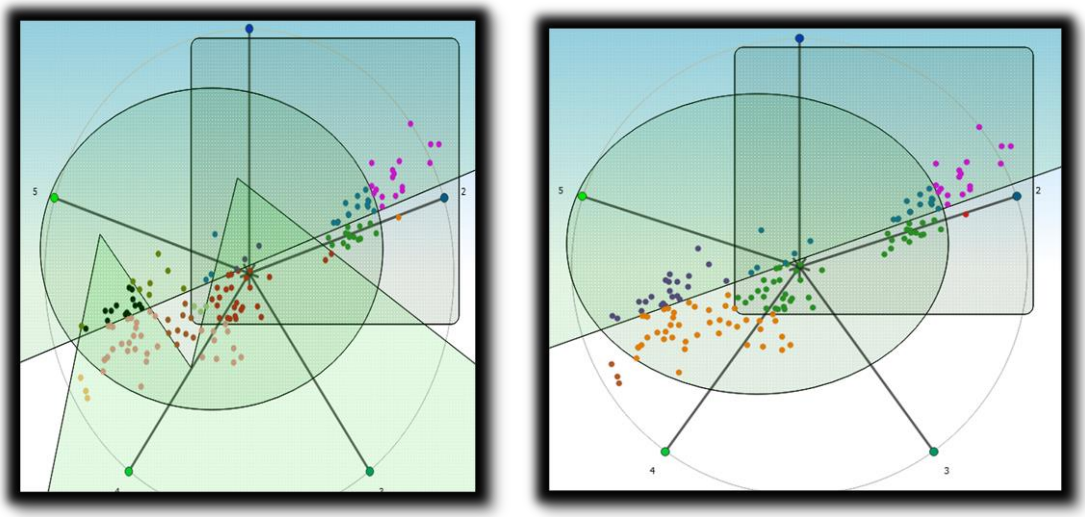


Figure 4.2: Movement.

Name: Delete step by step
Objective: Check that the application does not break while deleting separators and reclassifies properly.



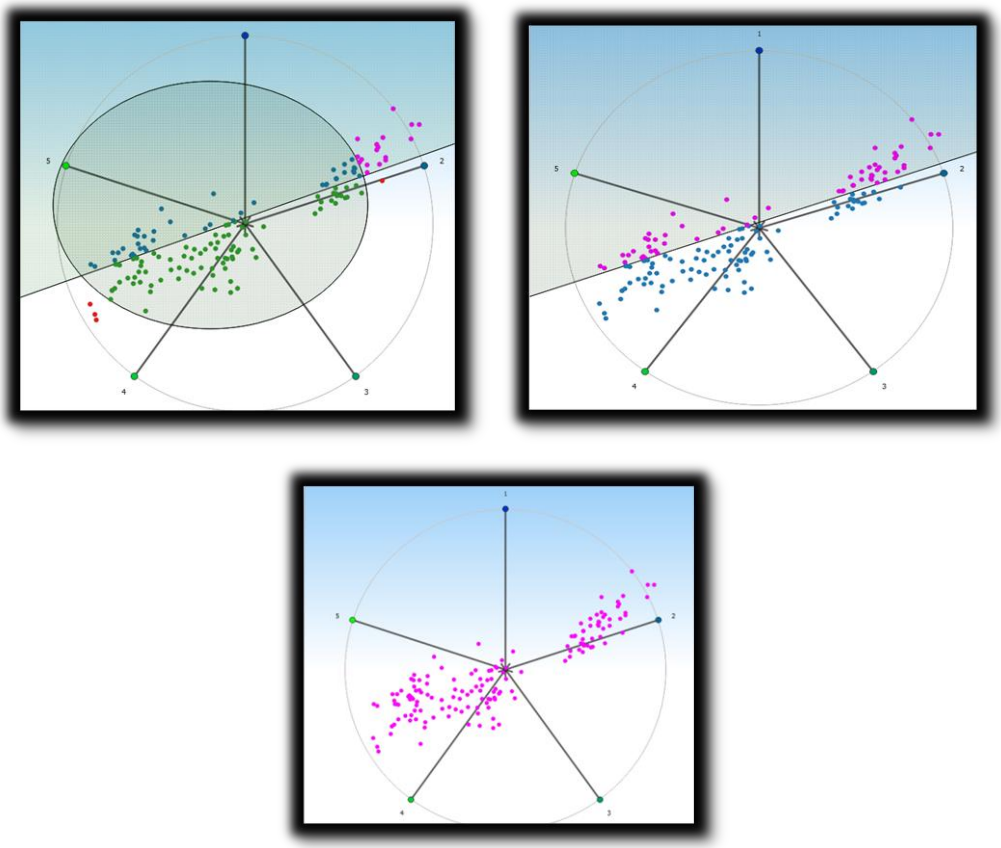


Figure 4.3: Delete step by step.

Name: Different projection axis
Objective: Check that the application works with different projection axis.

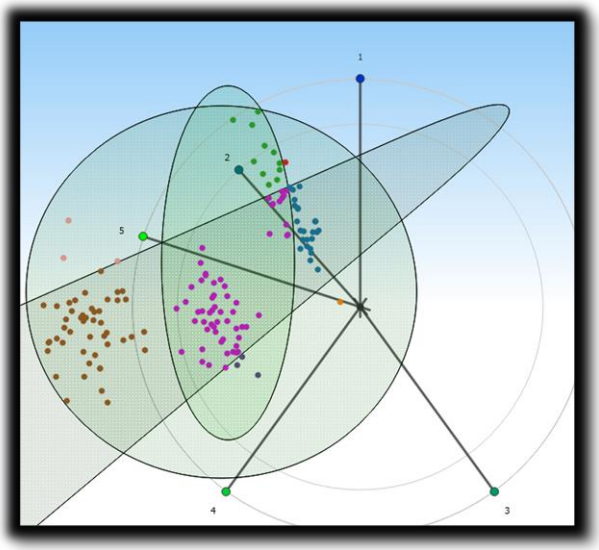


Figure 4.4: Different projection axis.

Name: Edit
Objective: Check that the application does not break while editing.

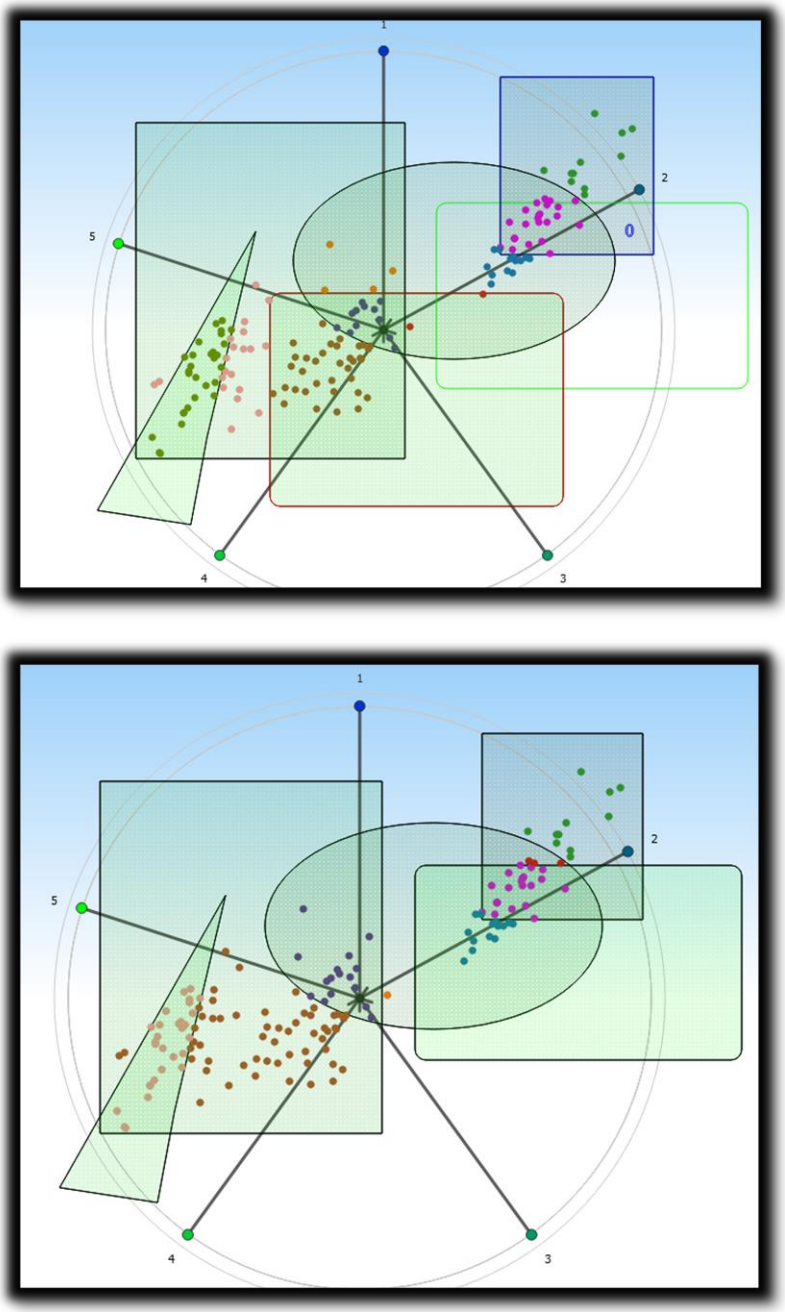


Figure 4.5: Edit.

Name: Large amount of data
Objective: Check that the application does not break working with large amount of data.

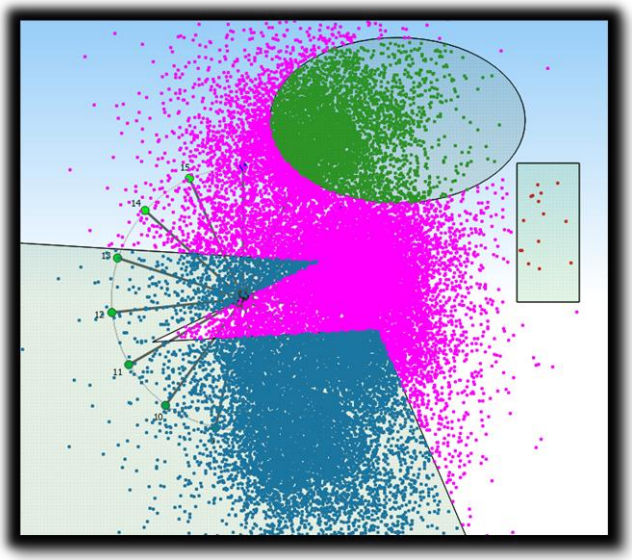


Figure 4.6: Large amount of data.

Name: Save
Objective: Check that the application saves properly.

Name: Classify buttons (empty)
Objective: Check that the application does not break.

Name: Euclidean classification
Objective: Check that the application reclassifies properly.

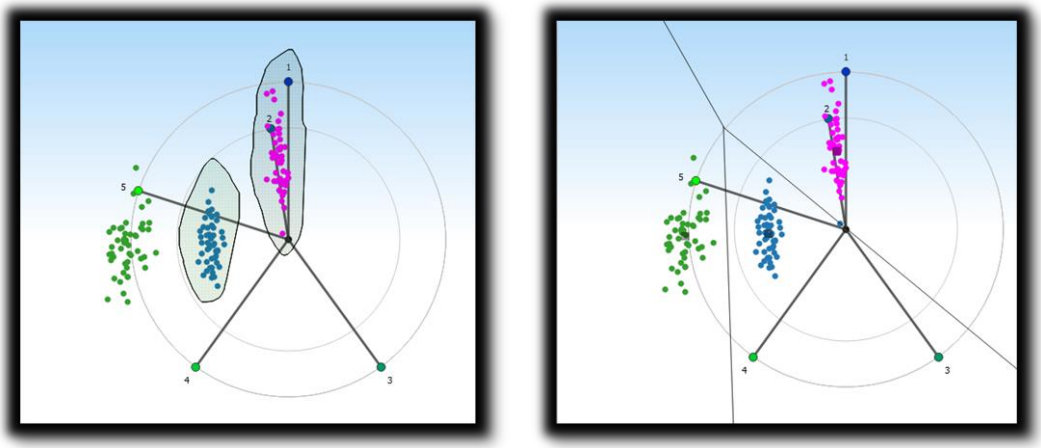


Figure 4.7: Euclidean Classification.

Name: Voronoi classification
Objective: Check that the application reclassifies properly (even with near data).

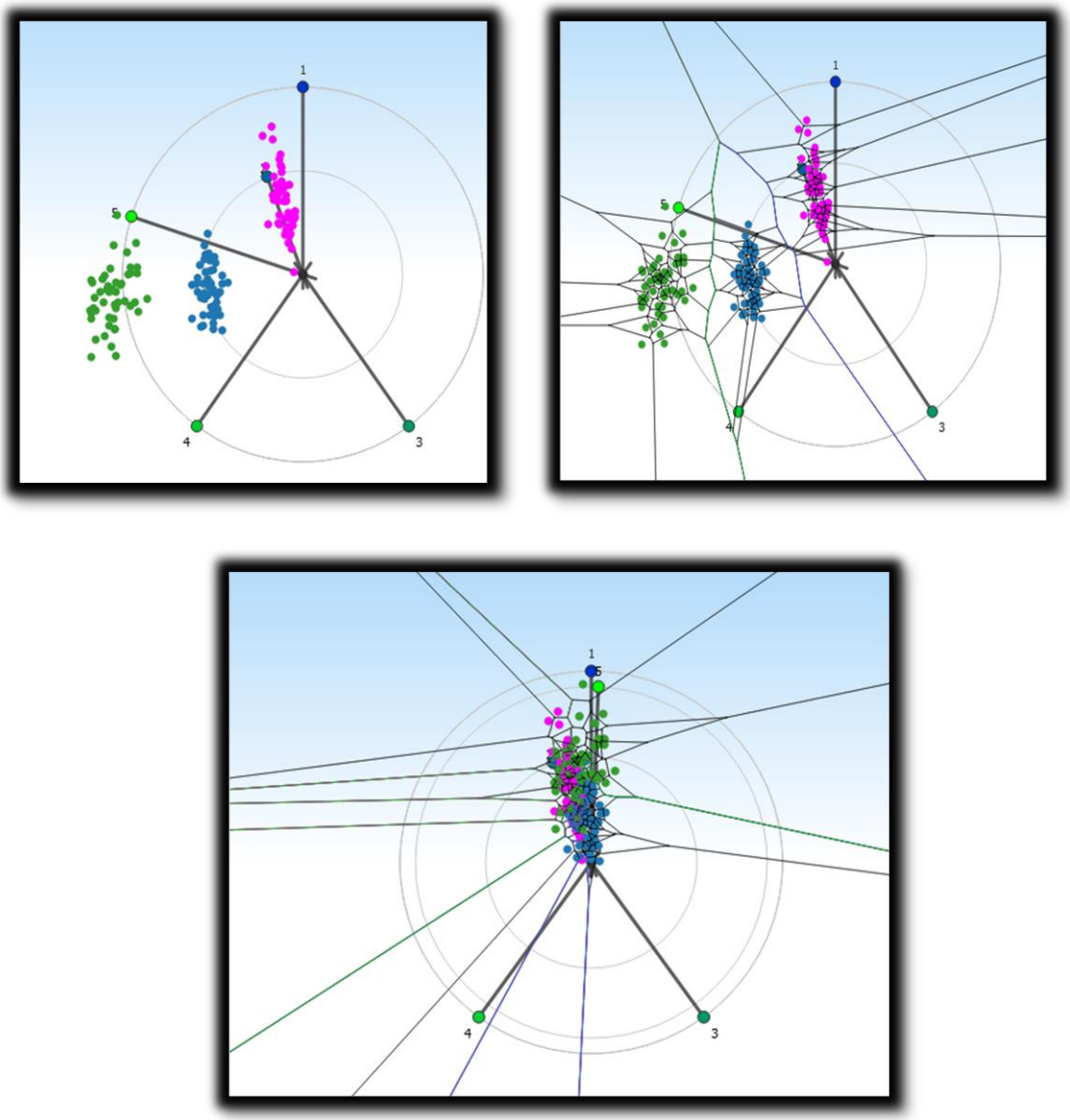


Figure 4.8: Voronoi classification.

Name: Filtering
Objective: Check that the application filtrates properly.

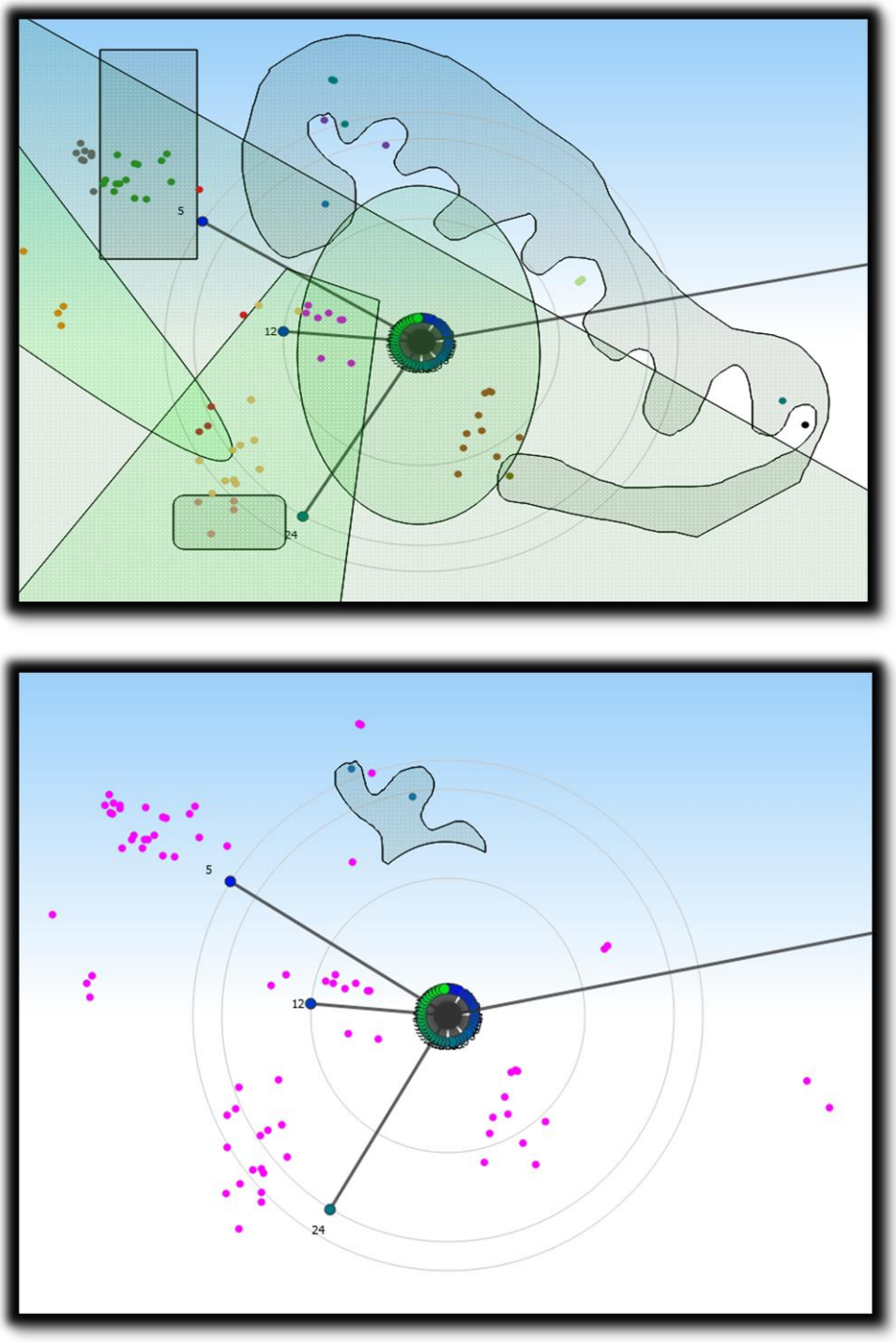


Figure 4.9: Filtering.

4.2 Example of use

An example of use is showed below. It is the description of how this extension should be used.

Firstly, the user loads the data (in this case there are data of iris).

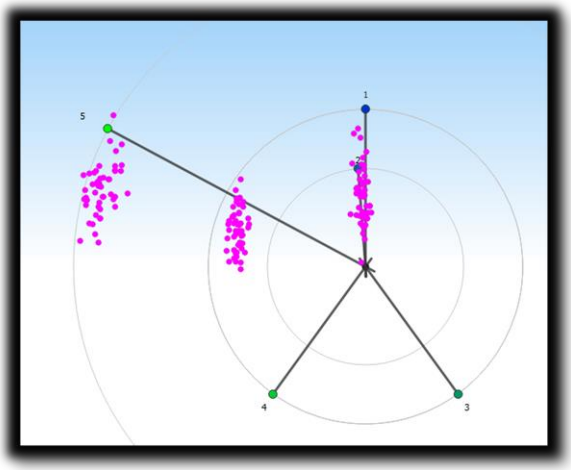


Figure 4.10: Iris data

Secondly, after having separate the data using the projection axis, the user classificate the points using class-separating objects.

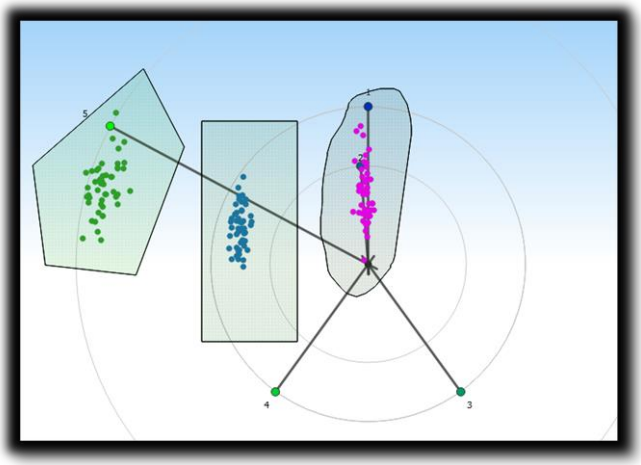


Figure 4.11: Iris data separated.

Then the user saves this structure for a new classification (in this example Voronoi is used).

The second part would be to add new points (a modification of the original data has been use due to the lack of data).

To have better results the user could project the points in a similar projection before reclassifying.

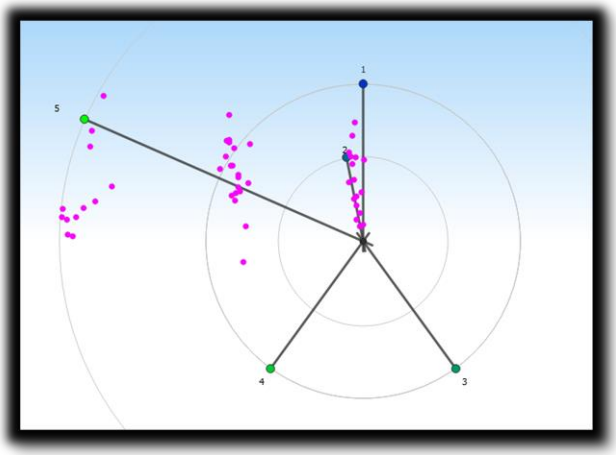


Figure 4.12: New data in a similar projection.

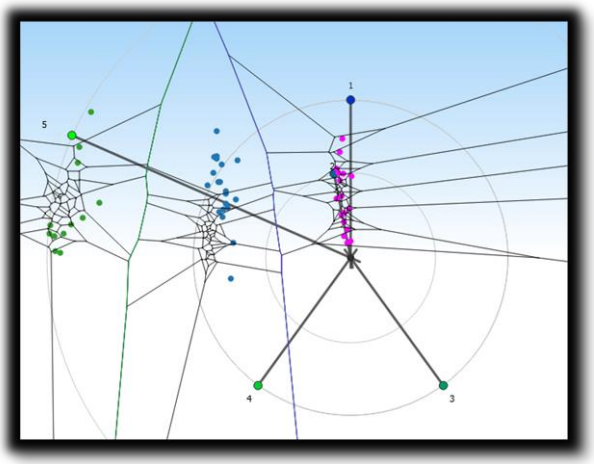
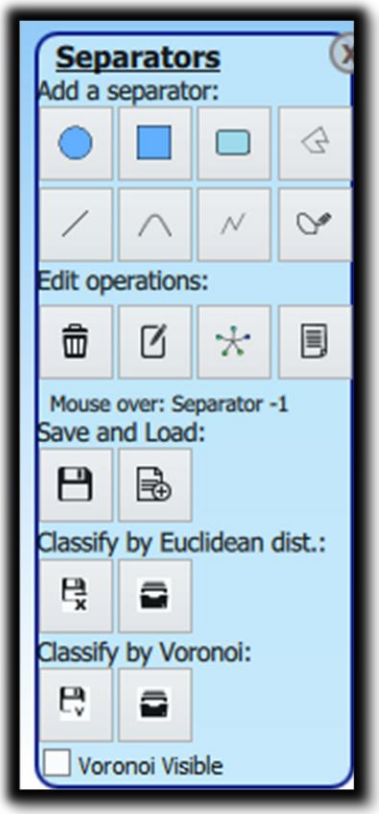


Figure 4.13: New data classified based on Voronoi.

Eventually the user saves this classification.

4.3 Summary of program functionality



There are five sections:

Add a separator:

This section has 8 buttons:

Circle/Ellipse (1): Click and move to resize the ellipse. When mouse is released the ellipse is on the screen at that size and the application classifies data.

Rectangles (2, 3): Click and move to resize.

Polygon (4): Click the points or click and move the mouse for more precision. To finish a double click is needed.

Line (5): Click and move.

Curve (6): Click and move for the first line. Click again or click and move to set the curve.

Polyline (7): Click the points or click and move the mouse for more precision (first line will elongate). To finish a double click is needed.

Freeform Polygon (8): Click and move the mouse. Double click is needed to close the polygon.

Edit operations:

Delete all (1): Delete all the objects.

Edit (2): Present a list of separators where the user can move, repaint or delete.

Projections (3): Click and then you can move the axes. Click again to end the edition of the projections.

List (4): List of colours and classes.

Save and Load:

Save (1): Saves the classification in a file.

Load (2): Loads new data.

Classify by Euclidean dist.:

Save (1): Save the current classification for later use.

Classify (2): Classifies data based on the last classification saved with save (1).

Classify by Voronoi:

Save (1): Save the current classification.

Classify (2): Classifies data based on the last classification saved with save (1).

Checkbox: Enables to see or not the Voronoi diagram.

Shortcuts:

Left Click and drag an object to move it.

Right click over an object to delete it.

Middle click (wheel) to filtrate. (Operation that takes only one class.)

5. Conclusion

5.1 Objectives achieved

In this Bachelor's thesis is described the extension of a software of data visualization. This extension makes able facilitate to classify data using painted shapes to separate data.

In the sample session, which is based on the application, it shows how this program achieves the objective of classifying data from the user's knowledge, with a simple, intuitive and attractive interface for the user.

On the other hand, it has been possible to classify new data based on a previous classification.

5.2 Future work

Subsequent development could consist in:

- Extracting a mathematical function from the classifications produced by the user.
- Export the algorithm to other languages and libraries.
- Improve program usability.

6. Bibliography

[1] ACHTERT, E. KRIEGEL, H., SCHUBERT, E. and ZIMEK, A. (2013), "Interactive Data Mining with 3D-Parallel-Coordinate-Trees", *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. New York, USA.

[2] AGGARWAL, C. (2002), *Towards Effective and Interpretable Data Mining by Visual Interaction* [online], available in https://pdfs.semanticscholar.org/410e/ca0d88fc05431adfd6b5ae0671ec625ee09f.pdf [Date: 2017-05-28]

[3] ANKERST, M., ELSEN, C., ESTER, M. and KRIEGEL, H.-P. (1999), *Visual classification: an interactive approach to decision tree construction,* [online], available in https://pdfs.semanticscholar.org/2619/e07cf15de869687d1367a945500e16f9eca0.pdf [Date: 2017-05-28]

[4] ANKERST, M., KEIM, D. and KRIEGEL, H. (1996), *Circle Segments: A Technique for Visually Exploring Large Multidimensional Data Sets*, [online], available in https://bib.dbvis.de/uploadedFiles/187.pdf [Date: 2017-05-28]

[5] APARICIO, M. and COSTA, C. (2015), "Data visualization" in *Communication Design Quarterly Review,* vol. 3, pp. 7-11.

[6] BHADESHIA H. K. D. H. (1999), *"Neural Networks in Materials Science"* [online], available in https://www.phase-trans.msm.cam.ac.uk/abstracts/neural.review.pdf [Date: 2017-05-28]

[7] BISHOP, C.M. (1995), *Neural Networks for Pattern Recognition* [online], available in http://cs.du.edu/~mitchell/mario_books/Neural_Networks_for_Pattern_Recognition_-_Christopher_Bishop.pdf [Date: 2017-05-28]

[8] COHN, D., ATLAS, L. and LADNER, R. (1994), *Improving generalization with active learning, Machine Learning* [online], available in

http://www.cs.northwestern.edu/~pardo/courses/mmml/papers/active_learning/improving_generalization_with_active_learning_ML94.pdf [Date: 2017-05-28]

[9] DUDA, R., HART, P. and STORK, D. (2001), *Pattern Classification*, Wiley.

[10] FRIEDMAN, V. (2008), *Data Visualization and Infographics* [online], available in https://www.smashingmagazine.com/2008/01/monday-inspiration-data-visualization-and-infographics/ [Date: 2017-05-28]

[11] FUKUNAGA, K. (1990), *Introduction to Statistical Pattern Recognition* [online], available in https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/Pattern_recognition/Introduction%20to%20Statistical%20Pattern%20Recognition%202nd%20Ed%20-%20%20Keinosuke%20Fukunaga.pdf [Date: 2017-05-28]

[12]*Git* [online], available in https://git-scm.com/ [Date: 2017-05-28]

[13] HANSEN, C. and JOHNSON, C. (2004), *Visualization Handbook*, Academic Press, Orlando.

[14] HARROWER, M. and BREWER, C. (2003), *ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps* [online], available in http://www.albany.edu/faculty/fboscoe/papers/harrower2003.pdf [Date: 2017-05-28]

[15] HASTIE, T., TIBSHIRANI, R. and FRIEDMAN, J. (2013), *The Elements of Statistical Learning: Data Mining, Inference and Prediction* [online], available in

https://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf [Date: 2017-05-28]

[16] INSELBERG, A. (1997), *Multidimensional detective* [online], available in http://www.cs.uml.edu/~grinstei/91.510/Papers/inselberg97.pdf [Date: 2017-05-28]

[17] JOHANSSON, S. and JOHANSSON, J. (2009), "Interactive dimensionality reduction through user-defined combinations of quality metrics" *in IEEE Trans Vis Comput Graph,* pp.993-1000

[18] KANDOGAN, E. (2001), "Visualizing multi-dimensional clusters, trends, and outliers using star coordinates", in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining,* pp.107-116

[19] KANDOGAN, E. (2000), "Star Coordinates: A Multi-dimensional Visualization Technique with Uniform Treatment of Dimensions", in *Proceedings of the IEEE Information Visualization Symposium*, Late Breaking Hot Topics, pp. 4–8.

[20] KEIM, D. (2002), Information and visual data mining, *IEEE Transactions on Visualization and Computer Graphics*, pp. 1-8.

[21] KUMAR, V. *Discovery series.* Minnesota, USA, Chapman & Hall/CRC.

[22] LEHMANN, D. J. and THEISEL, H. (2013)," Orthographic Star Coordinates", in *IEEE Transactions on Visualization & Computer Graphics*, pp. 2615-2624

[23] LEHMANN, D. J. and THEISEL, H. (2015), *Optimal Sets of Projections of High-Dimensional Data* [online], available in http://wwwisg.cs.uni-magdeburg.de/visual/files/publications/2016/Lehmann_2015_InfoVis.pdf [Date: 2017-05-28]

[24] LEHMANN, D. J. and THEISEL, H. (2016), *General Projective Maps for Multidimensional Data Projection* [online], available in https://isgwww.cs.uni-magdeburg.de/visual/files/publications/2016/Lehmann_2016_EG.pdf [Date: 2017-05-28]

[25] MITCHELL, T. (1997), *Machine Learning*, McGraw Hill.

[26] POST, F., NIELSON, G. and BONNEAU, G. (2002). *Data Visualization: The State of the Art*, Boston/Dordrecht/London, Kluwer Academic Publishers.

[27] *Qt Documentation* [online], available in http://doc.qt.io/ [Date: 2017-05-28]

[28] QUINLAN, J. (1993*), C4.5: Programs for Machine Learning, Morgan-Kaufmann Publishers* [online], available in https://pdfs.semanticscholar.org/dec4/6fb79cde2823c13dea9a6b604e1dfcb435cd.pdf [Date: 2017-05-28]

[29] QUINLAN, J. (1986), *Induction of decision trees, Machine Learning*, [online], available in http://hunch.net/~coms-4771/quinlan.pdf[Date: 2017-05-28]

[30] RUBIO-SÁNCHEZ, M., RAYA, L., DÍAZ, F. and SÁNCHEZ, A. (2016), "A comparative study between RadViz and Star Coordinates*",* in *IEEE Transactions on Visualization & Computer Graphics.*

[31] SETTLES, B. (2012), *Active Learning*, Morgan and Claypool.

[32] SOUKOP, T. and DAVIDSON, I. (2002), *Visual Data Mining: Techniques and Tools for Data Visualization*, Wiley.

[33] SPINELLI, J. and ZHOU, Y. (2004), *Mapping Quality of Life with Chernoff Faces* [online], available in http://www.dis.uniroma1.it/~santucci/InformationVisualization/Slides/References/MappingQualityLifeChernoffFaces.pdf [Date: 2017-05-28]

[34] TUFTE, E. (1983), *The Visual Display of Quantitative Information*, Cheshire (Connecticut), Graphics Press.

[35] VAPNIK, V. (1995), *The Nature of Statistical Learning Theory* [online], available in http://read.pudn.com/downloads161/ebook/733192/Statistical-Learning-Theory.pdf [Date: 2017-05-28]