University of Glasgow | School of Computing Science

Machine Learning the Premier League

Victor Semerdjiev

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

September 2014

Abstract

The MCFC Analytics 2011-12 Premier League Season Full Dataset contains a statistical record of all match events that have occurred during the 2011-12 Premier League season. Machine learning is a sub-field of artificial intelligence (AI) concerned with the engineering of computer systems that can learn from data. The goal of this project was to develop an application that would enable non-experts in statistical analysis to apply powerful machine learning algorithms for the analysis of this and other similar datasets.

The K-Means clustering algorithm, SVM classification algorithm, and an Interquartile Range outlier detection filters have been included in the application. Methods for evaluating the performance of each of these algorithms have been implemented. The results of the included algorithms can be visualised on interactive 3-d scatter plots.

The application stores its datasets in an embedded database, which enables their efficient filtering using SQL. The application is state-based and multithreaded, which facilitates the pleasant and efficient interaction with its user interface. The application's usability was tested with users, and the received feedback was favourable.

Acknowledgements

I would like to extend my heartfelt gratitude to my supervisors Dr. Simon Rogers and Mr. Ala' Al-Afeef. I am grateful for the advice and guidance they have provided me with throughout the time of this project.

In addition, I would like to thank my family, without whose support I would not be where I am today.

Contents

Chapter 1

Introduction

In August 2012, as a part of the MCFC Analytics project, the Manchester City Football Club (MCFC) and the sports data analysis company OptaPro jointly released two datasets containing data about the 2011-12 Premier League season to the general public. (Perhaps coincidentally, MCFC was also the champion of the Premier League that season.) One of the aforementioned datasets, named the "Full Dataset", contains a statistical record of all match events that have occurred during the 2011-12 Premier League season. The other dataset, named the "Advanced Dataset", contains information about not only what match events have occurred in every match of the 2011-12 Premier League season, but also about when, as well as where on the football pitch, each of these events happened.

Machine learning is a sub-field of artificial intelligence (AI) concerned with the engineering of computer systems that can learn from data. Some machine learning algorithms are particularly useful for performing various statistical analyses on large and complex datasets. In many cases there are several machine learning algorithms capable of performing the same type of task, which allows for the convenient categorical grouping of machine learning algorithms. For example, there are machine learning algorithms for regression, classification, clustering, anomaly detection, recommender systems, etc., just to name a few. Machine learning algorithms can also be categorized based on the type of data (i.e., labeled, or unlabeled data) that they operate on (supervised learning, and unsupervised learning, respectively).

Recent technological advancements (such as the advent and proliferation of powerful mobile devices) have made data collection easier than ever before. This has led to the accumulation of countless (and often massive) datasets. Much knowledge, wisdom, and intelligence could be derived from the datasets that are currently available, which in turn could serve to assist humans in making smarter and more educated decisions (decisions that are based on factual data). For example, insights gained from the MCFC Analytics datasets could potentially help inform the managers of the teams in the Premier League and help drive the strategies of their teams. The main issue with the MCFC Analytics datasets is that they are very complex, which makes it difficult for someone who is not an expert in advanced statistical analysis to derive interesting and meaningful information from them.

The goal of this project was to create an application that would provide football enthusiasts (most of whom do not possess the pre-requisite technical knowledge and skills to perform advanced statistical analyses on their own) with the opportunity to easily run powerful machine learning algorithms in order to analyse the Full Dataset (although the produced application also supports other datasets). Some of the most popular machine learning algorithms for various tasks such as clustering, classification, and anomaly detection have been included in the produced application. This provides users of the application with the opportunity to perform a range of machine learning tasks and to be exposed to a variety of machine learning algorithms. The application was designed so that it would also familiarise its users with basic machine learning theory, as well as so that it would teach them how to effectively operate the provided machine learning algorithms.

There are several qualities of the produced application that differentiate it from alternative applications for machine learning analysis. Firstly, it has an embedded database, which already contains the Full Dataset, and to which new datasets could easily be added. This allows for increased flexibility in filtering and manipulating datasets due to the power of SQL, which is used to query the database. Some other applications for machine learning analysis also provide the functionality of querying a database, however they usually require that the database be developed separately and in advance (in contrast to this application, which allows dynamic database development). Secondly, the produced application assumes absolutely no prior knowledge of Machine Learning from its users. At the time of writing of this dissertation, no alternative full-featured applications for machine learning analysis that are similar in this regard had been identified. Thirdly, the produced application supports automatic model selection for the K-Means clustering algorithm (through running the K-Means algorithm multiple times and returning the model that had the least within cluster sum of squared errors). (This feature is also present in Java-ML, but it is not present in Weka.) Lastly, the produced application provides an advanced way (interactive 3-D scatter plots) of visualising the results of machine learning algorithms – a feature that is not typically present in conventional machine learning applications and environments. (Matlab and Octave also support interactive 3-D scatter plots, but they are environments for general numerical computation, and not machine learning applications per se).

Chapter 2

Background Survey

2.1 Sports Analytics

The world of modern-day professional sports is an immensely competitive one. Professional athletes and teams are constantly looking for ways to improve their performance in an attempt to gain an edge over the competition. Given the amount of money on the line, winning is of utmost importance for the financial viability and success of professional athletes and teams. Thus, there has been an increase in the use of science and technology in professional sports in recent years. Intelligence obtained through sports analytics can be used to shape the strategies of professional athletes and teams. Nowadays, there are companies that specialize in sports data analysis (e.g., OptaPro the company behind the MCFC Analytics project). Also, it is not uncommon for Premier League teams to make use of the services of professional data analysts (as is evident from the partnership between MCFC and OptaPro). Last but not least, sports analytics is popular in a betting context.

2.2 The MCFC Analytics 2011-12 Premier League Season Full Dataset

Released to the general public in August 2012, the Full Dataset is a product of the joint collaboration of MCFC and the sports data analysis company OptaPro. It contains a statistical record "of every event for every player in every game of the 2011-12 Premier League season" [11]. The dataset is originally stored in Microsoft Excels .xls format, and consists of 10369 rows (training examples) and 236 columns (features) of data. The complexity and size of the Full Dataset make it difficult to infer deep insights from it without performing some sort of advanced statistical analysis.

Each row of the Full Dataset represents all match events, related to a single player, in a single match. For example, a single row of the Full Dataset could tell us that on the 23rd of October, 2011, Wayne Rooney played for 90 minutes in a match against Manchester City, had 2 shots on target, 0 goals, 71 ball touches, etc. Individual rows of the Full Dataset can be combined in order to obtain a different set of data (e.g., instead of analysing what a single player accomplished in a single match, we could analyse what a player accomplished over the entire season, or what a team accomplished over the entire season, etc.).

2.3 Previous Work on the MCFC Analytics 2011-12 Premier League Season Full Dataset

The MCFC Analytics datasets were released to the general public with the idea that regular people might discover interesting patterns in the data. Hence, most of the previous work on the Full Dataset has been done outside of academia, due to which it is rather informal. Much of the previous work on the Full Dataset has appeared on various personal blogs and websites.

Previous work on the Full Dataset encompasses diverse areas such as descriptive statistical analyses (e.g., correlation analyses, etc.) and data visualizations, among others. For example, one interesting project on the dataset performs an analysis of how teams win (i.e., what are the correlations between various team characteristics and winning) [22]. Another notable project on the Full Dataset analyses how fouls turn into cards [21].

In terms of previous machine learning work performed on the Full Dataset, there has been one completed thesis project [20], in which the author attempts to predict match outcomes in a betting context, by using a multinomial logistic regression algorithm and experimenting with different feature representations. The author of this project managed to produce winning betting strategies over bookmakers [20]. It is important to note that the "Full Dataset" is relatively recent, so it is possible that there are other ongoing projects on it.

2.4 Existing Related Software

Few machine learning applications for data analysis have been designed specifically for users who are new to machine learning. This section contains an assessment of the strengths and weaknesses of two such applications. (These are the only such applications that I have managed to identify.)

2.4.1 Microsoft Azure Machine Learning

Microsoft Azure Machine Learning is a cloud-based, software as a service (SaaS) machine learning web application. It has a particular focus on predictive analytics and aims to make machine learning more accessible to a much wider audience [12]. The main strength of Microsoft Azure Machine Learning is that by leveraging the simplicity and the wide reach of the cloud it offers easy access to powerful data analytics and machine learning tools to a broad audience. Being offered by one of the biggest and most reputable IT companies in the world, the service is also seemingly well supported and rapidly evolving (as indicated by its recent progress). The main weaknesses of Microsoft Azure Machine Learning are that it requires paid subscriptions for full access to its capabilities, and that (as of September 2014) it does not offer as many machine learning services as some alternative applications and libraries for machine learning analysis do (e.g., Weka).

2.4.2 Teaching Aid

Teaching Aid is an application that is used to assist in teaching the fundamentals of machine learning [25]. For example, it can be used to demonstrate how changing the parameters of machine learning algorithms affects their output. The main strength of Teaching Aid is that it can provide students who use it with a comprehensive low-level understanding of how many machine learning algorithms operate. The main limitation of Teaching Aid is that it cannot be used to effectively analyse big datasets (such as the Full Dataset). (It was designed as a teaching tool, and not as a full-fledged application for machine learning analysis.)

2.5 Statement of Problem

The absence of free and comprehensive applications for machine learning analysis (of big datasets) that are designed specifically for users who are new to machine learning revealed the need for creating such an application. The application should be centred around providing machine learning tools for the analysis of the Full Dataset, but at the same time it would need to be universal enough to support other datasets as well. An analysis of the problem of engineering such an application led to the discovery that the two main components of such an application would be:

1. A machine learning component
2. A visualisation component

The first component refers to the machine learning algorithms and services that the application would support. **The second component refers to** the methods and libraries that the application would use to visualise the results of machine learning results (i.e., the clustering assignments in a clustering algorithm, the classification accuracy of a classifier in a classification algorithm, and the identified outliers in an outlier detection algorithm).

2.6 Review of Machine Learning Components

Having identified what the main components of the application would be, the next step was to survey external machine learning and visualisation components which could be incorporated in the application. This section contains an examination of the strengths and weaknesses of several machine learning components.

2.6.1 Matlab and Octave

Matlab is “a high-level language and interactive environment for numerical computation, visualization, and programming, [which can be used to] analyse data, develop algorithms, and create models and applications” [10]. Matlab is incredibly powerful and versatile; however, it is not easy-to-use (especially for beginners). Effectively implementing machine learning algorithms in Matlab usually requires prior machine learning knowledge, as well as skills in Matlab development. Another limitation of Matlab is that it is proprietary commercial software, licenses for which are quite expensive (which makes it unaffordable for many people).

GNU Octave “is a high-level interpreted language, primarily intended for numerical computations” [4]. It is very similar to Matlab, and it supports most of the capabilities and functionalities provided by Matlab. Unlike Matlab, GNU Octave is available free of charge (under the terms of the GNU General Public License). One major limitation of GNU Octave is that it does not provide a Graphical User Interface (GUI). Hence, it must be used through its command line interface, which makes it even less intuitive to use than Matlab.

It has to be noted that both Matlab and Octave have outstanding plotting and visualisation capabilities. Creating beautiful scatter plots, histograms, contour plots, line plots, etc. in Matlab and Octave is often as easy as calling a function. Given how important plotting and visualisation are to machine learning partially explains why Matlab and Octave are such popular choices for teaching machine learning and for implementing machine learning algorithms. Another notable advantage of Matlab and Octave is that it is usually easier to implement a machine learning algorithm in them than it is to implement the same algorithm in another high-level programming language. (For this reason, many machine learning practitioners often implement learning algorithms in Matlab/Octave first, and only later, once their Matlab/Octave implementations are tested and proven to be correct, they try to port them to other programming languages.)

There are methods for integrating code written in other high-level programming languages (such as Java) with Matlab and GNU Octave. Theoretically, this should allow the software provided by both Matlab and GNU Octave to be incorporated in other applications. Using Matlab code in external applications poses licensing issues (since Matlab is not open-source software, unlike GNU Octave).

2.6.2 Weka and Java-ML

Weka is "a collection of machine learning algorithms for data mining tasks" [7]. The algorithms provided by Weka can either be applied directly to a dataset (there is a standalone Weka application, which has a GUI), or they can be called from another Java program [7]. Weka contains tools for data pre-processing, classification, clustering, and visualization, among others [7]. Java-ML is another Java library which provides a collection of implementations of various machine learning algorithms [1]. Both Weka and Java-ML are open-source projects, so the source codes of both libraries are freely available, and may be modified and redistributed. Unlike Weka, there is no version of Java-ML as a standalone application (thus Java-ML has no GUI). Compared to Weka, Java-ML provides a somewhat more limited collection of machine learning algorithms and services, and in general Weka appears to be the more extensive and advanced machine learning library of the two.

The strengths of Weka lie in the unified machine learning environment that it presents, the extensive collection of efficient implementations of machine learning algorithms that it provides and supports, and the numerous useful evaluation services for machine learning algorithms that it has on offer. Another advantage of the Weka machine learning library (which was deemed to be particularly pertinent to the produced application) is that Weka has native support for fetching data instances from a relational database. This is a distinguishing feature, because it allows datasets that are stored in databases to be efficiently queried and filtered with SQL.

However, Weka also has some drawbacks. One limitation of Weka is that by default it only uses simple 2-D scatter plots to visualise the results of its machine learning algorithms. While a reasonable default, this method is not optimal, as alternative techniques for visualising data are quite capable of producing more sophisticated visualisations. Another limitation of Weka is that its only implementation of the K-Means clustering algorithm (SimpleKMeans) does not support automatic model selection by automatically running the K-Means clustering algorithm multiple times and returning the model of the algorithm run that achieved the best results (In K-Means, different random initialisations of the cluster centroids could yield much different algorithm results). Application developers can overcome these limitations of Weka by extending Weka's default functionality.

2.6.3 LibSVM and LibLinear

LibSVM and LibLinear are two libraries that provide their own implementations of the SVM classification algorithm. LibLinear is a linear classifier, which provides a linear SVM solver [6]. LibSVM, on the other hand, provides an implementation of the SVM classification algorithm that supports various kernels [3]. LibSVM is the more advanced and extensive library of the two. The reason for this is that some of the kernels for the SVM classification algorithm that are supported by LibSVM (such as a Gaussian kernel) allow it to form complex non-linear classification hypotheses. Such hypotheses are often more accurate than the linear classification hypotheses formed by LibLinear.

The implementations of the SVM classification algorithm provided by LibSVM and LibLinear are quite robust and efficient. Both implementations support multi-class classification (using the one-against-one method) and probability estimates. Both LibSVM and LibLinear are free software, so they can be freely incorporated in other software packages and applications. LibSVM and LibLinear are available for many of the most popular programming languages, including (but not limited to) Java, Matlab, and Python. Although LibSVM and LibLinear are not included in the Weka API by default, they could easily and efficiently be incorporated in it (through the use of Weka wrappers for LibSVM and LibLinear) – the resulting symbiosis greatly enhances the individual capabilities of each of the participating libraries, i.e., LibSVM and LibLinear provide excellent implementations of the SVM algorithm, whereas Weka provides superb machine learning algorithm evaluation services.

One potential drawback of the LibSVM and LibLinear libraries is that they could be perceived as being somewhat bulky, because they offer much more than the absolute minimum required for an SVM implementation. However, in practice that is hardly an issue, because this bulkiness is insignificant by the standards of modern computers. The numerous benefits of LibSVM and LibLinear far outweigh any perceived drawbacks associated with them.

2.7 Review of Visualisation Components

The results of machine learning algorithms are usually visualised using various plots (e.g., scatter plots and contour plots). Due to this reason, our efforts were focused on surveying alternative Java plotting libraries. This section contains an examination of several such libraries.

2.7.1 Plotting Libraries

Due to the visualisation needs of the produced application, an emphasis was placed on examining three aspects of Java plotting libraries:

- Whether they support exporting images of plots in popular image formats.
- Whether they can generate interactive 3-D scatter plots.
- Whether they can generate contour plots.

Edgewall Software, the makers of the GRAL Java graphing library provide an excellent comparison of some of the most popular free Java plotting libraries (GRAL, JFreeChart, jChart2D, Openchart2, Jzy3d, and XChart) [5], which quickly revealed that out of all the open-source Java plotting libraries that they compared:

- all libraries (that participated in the comparison) provide support for exporting images of plots into popular image formats.
- only JFreeChart and Jzy3d support the creation of contour plots.
- only Jzy3d can create interactive 3-D scatter plots.

These findings led to a closer examination of Jzy3d. Jzy3d relies on JOGL to provide support for easily drawing hardware-supported 3D graphics [19]. Performing some tests on Jzy3d revealed that it was incredibly versatile and powerful, albeit perhaps not greatly documented. Another Java plotting library that was examined was Orson Charts – "a 3D chart library for the Java platform that can generate a wide variety of 3D charts for use in client-side (JavaFX and Swing) and server-side applications" [14], however it has a steep price tag (licences for it start at £69).

Chapter 3

Requirements

3.1 Requirements Elicitation

As John Brooks famously proclaimed, “the hardest single part of building a software system is deciding precisely what to build” [2] – a statement which very much holds true in the case of this project. In the nascent stages of the project there was a marked degree of flexibility in terms of what the project could entail. At that time, the only certainty about the content of the project was that it had to involve machine learning and some of the data that was released by the MCFC Analytics project. Most of the requirements were captured and refined through a series of meetings and discussions with my supervisors, Dr. Simon Rogers and Mr. Ala’ Al-Afeef. The initial questions that were being raised were mostly at the macro-level, for example:

- What type of system should be built?
- Who would be the potential users of the system-to-be?

Once enough general questions like these had been answered, the forthcoming ones started to become progressively more specific and design-oriented. Conducting a background survey on existing related software products also added ideas for potential features. It must be noted that the requirements had not been finalised before the start of the implementation phase of the project, and some were added later on. Eventually a set of functional and non-functional requirements (prioritised according to the MoSCoW method) was produced, in addition to a set of well-described use cases. (The requirements documents for the produced application can be located in Appendix A of this document.)

3.2 Requirements Analysis and Overview

Most of the initial requirements revolved around providing users of the application with the ability to apply various useful machine learning algorithms for the analysis of the MCFC Analytics Full Dataset. However, it was determined relatively early on that it would be advantageous if the application was made to be as universal as possible. For this reason, the requirement that the application should support the loading of other datasets (which are of similar format to the MCFC Analytics Full Dataset) was added with a “must have” priority as per the MoSCoW method, as well as its derivative implicit requirement that the user interface of the application should be designed in a way that it supports any dataset and is not heavily biased toward the MCFC Analytics Full Dataset. Once it was observed (during the implementation phase of the project) that adding new algorithms to the application was relatively easy with the help of external libraries which provide machine learning services (such as Weka), a requirement was added to implement a suitable way for visualising algorithm results even in the case of data which is more than three-dimensional (in order to increase the difficulty of the project).

In terms of non-functional requirements, it was requested that the application be designed with the understanding that most of its users would be new to machine learning in mind. For this reason, the application was to be as easy-to-use as possible. Later on in the development process, it was determined that loading datasets taking too long (i.e., much more than a few seconds) was not user friendly, for which reason a target dataset loading time was added to the set of requirements. Of course loading a new dataset also depends on the size of the dataset, so that was taken into account in making the added requirement sensible and reasonable. Also, a requirement was added that the application’s GUI should remain responsive to user interaction at all times (when the application is running).

Chapter 4

Design

With a concrete set of requirements in hand, and having analysed these requirements, the next step was to design the application in a way that it would fully meet its set requirements. A three week period at the start of the project was dedicated exclusively to designing the application. A set of design documents was present at the end of this period, but the application's design developed further throughout the implementation phase of the project. (The final design documents for the produced application can be located in Appendix B of this document.) This chapter provides a discussion on the final design of the application, as well as on how the application's design developed over time. Some additional more low-level analysis of the application's design can be found in the next chapter of this document (chapter 5 – Implementation).

4.1 Overall Software Architecture

It was clear from relatively early on that the Model-View-Controller (MVC) software architectural design pattern could be used for the software architecture of the application. Ultimately, the final software architectural design of the application generally follows the MVC design pattern. In the final design of the application, the machine learning algorithm classes are models, the application's database is another model, three GUI classes (MainView, VisualisationView, and AboutFrame) are views, and there is a dedicated controller class (Controller). (Please refer to Figure 4.1 for a high-level overview of the application's architecture, or to Figure B.1 in Appendix B of this document for a more detailed UML class diagram of the application's architecture.)

There are some classes within the application that cannot be classified as models, views, or controllers. For example, DatasetDatabaseLoader (the class responsible for loading new datasets into the database) is one such class. Despite this, the application's architectural design follows the MVC design pattern as much as possible. Interactive3dScatterPlot, the class that generates interactive 3-D scatter plots (for visualising the results of some machine learning algorithms), is not a view class on its own, but the scatter plots that it generates are embedded in the user interface provided by a view class (VisualisationView) that was developed specifically for this purpose.

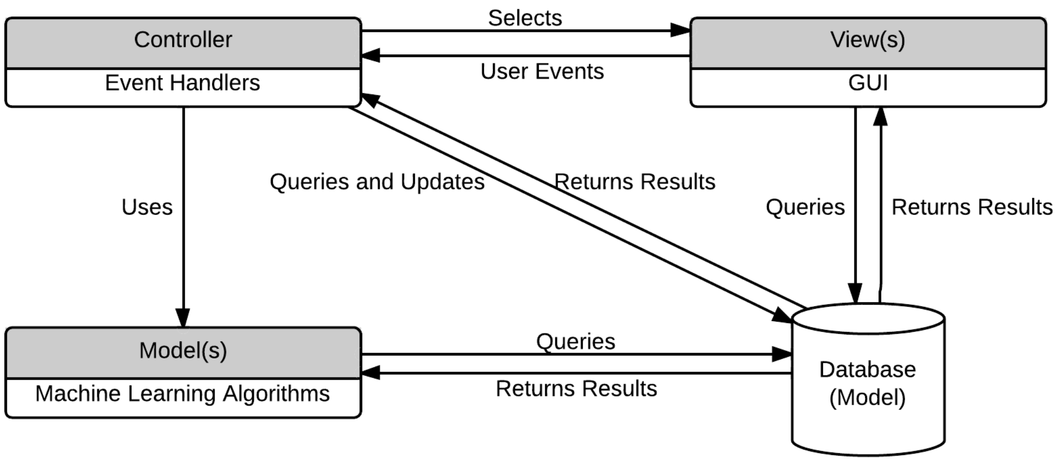


Figure 4.1: Overall software architecture of the application

4.2 Machine Learning Algorithms

The choice of which machine learning algorithms to include in the application was made relatively early on. The following machine learning algorithms have been included in the application:

- K-Means clustering algorithm (The K-Means clustering algorithm was selected among alternative clustering algorithm because of its relative simplicity and effectiveness.)
- Support Vector Machines (SVM) classification algorithm (The SVM classification algorithm was selected among alternative classification algorithms because of its fit to the Full Dataset, with respect to the quantity of training examples and features contained within the Full Dataset.)

In addition, an Interquartile Range outlier detection filter has been included in the application. The particular selection of algorithms included in the application is largely based on the varied functionality they offer.

4.2.1 Choice of Algorithm Implementations

Several alternative machine learning components that could potentially be incorporated in the application were surveyed. (Please refer to section 2.6 of this document for the survey of machine learning components.) Based on the conducted survey, and because of Weka's comprehensiveness, it was decided that the Weka API would form the backbone of the produced application (in terms of machine learning functionality). Hence, where possible, the application uses implementations of algorithms from the Weka API. In addition, (based on the conducted survey) it was decided that the application would use LibSVM's implementation of the SVM classification algorithm, because it is arguably the best tool for this specific task (i.e., SVM classification).

4.2.2 The `algorithms` Package

Because the Weka API supplies most of the application's machine learning functionality, the final design of the `algorithms` package (and of the classes within it) is heavily influenced by its compatibility with the Weka API. This package contains all classes of the application that are related to machine learning algorithms. Most of these classes are wrappers for implementations of machine learning algorithms that have been provided by external libraries. For example, the application uses Weka's implementation of the K-Means clustering algorithm (SimpleKMeans), but also extends its functionality so that it supports automatic model selection through running the K-Means clustering algorithm multiple times and returning the model that has the least within cluster sum of squared errors. The application's outlier detection algorithm (Interquartile Range outlier detection filter) is also provided by the Weka API. (An alternative outlier detection algorithm was implemented as well, but after testing and evaluating it, it was decided not to include it in the final version of the application.)

The application uses the implementation of the SVM classification algorithm that has been provided by the LibSVM library. Although it is possible, the application does not access LibSVM directly, but instead uses the Weka API as an intermediary for accessing LibSVM (through the help of a Weka wrapper for LibSVM). There are two reasons for this design decision. **The first reason is** that it enables some of the machine learning services offered by the Weka API (such as its data pre-processing filters, cross-validation, ROC curves/AUC performance metrics, etc.) to be applied directly to LibSVM's implementation of the SVM classification algorithm. **The second reason is** that having the application access all of the implementations of machine learning algorithms that it would need from the same place (the Weka API) allows for a greater level of uniformity in the design of the application. For example, LibSVM's implementation of the SVM classification algorithm normally requires its input data to be in a different format compared to the input format required by the Weka API. Accessing LibSVM through the Weka API (with the help of a wrapper) solves this issue by enabling LibSVM to handle input data that is in Weka's input format. Hence, all machine learning algorithms that have been included in the produced application require input data that is in the same format.

The `algorithms` package has a hierarchical structure. (Please refer to Figure 4.2 for a high-level overview of the structure of this package, or to Figure B.2 in Appendix B of this document for a detailed UML class diagram of this package.) All subclasses within this package are descendants of the abstract superclass Algorithm. There is a second level of abstraction within the package, at which additional abstract classes (which represent the different types of machine learning algorithms that are supported by the application) extend the Algorithm superclass, while at the same time being parents to classes that represent actual implementations of machine learning algorithms.

The Algorithm Class

The top class Algorithm contains class members that are common to all of its subclasses. It is an abstract class, and as such, no instances of its type can be instantiated. It provides implementations for some methods that its subclasses would have to provide identical implementations for otherwise, such as methods for setting the *instanceQuery* object (the SQL query that is used for requesting dataset instances from the database) and for executing it in order to fetch instances from the database. An instance variable *trainingSet* is used to store the instances that were fetched from the database, and a public accessor method allows convenient access to it from outside classes. All of Algorithm's instance variables have a protected access modifier, so that they would be visible to other classes within the `algorithms` package and to all subclasses of Algorithm, but not to any other classes that are located outside of these boundaries. Algorithm also specifies a few abstract method definitions for core machine learning methods (namely the methods TRAIN, EVALUATE, and SETOPTIONS), thus enforcing a rule that all instantiable subclasses of Algorithm must contain actual implementations for these methods.

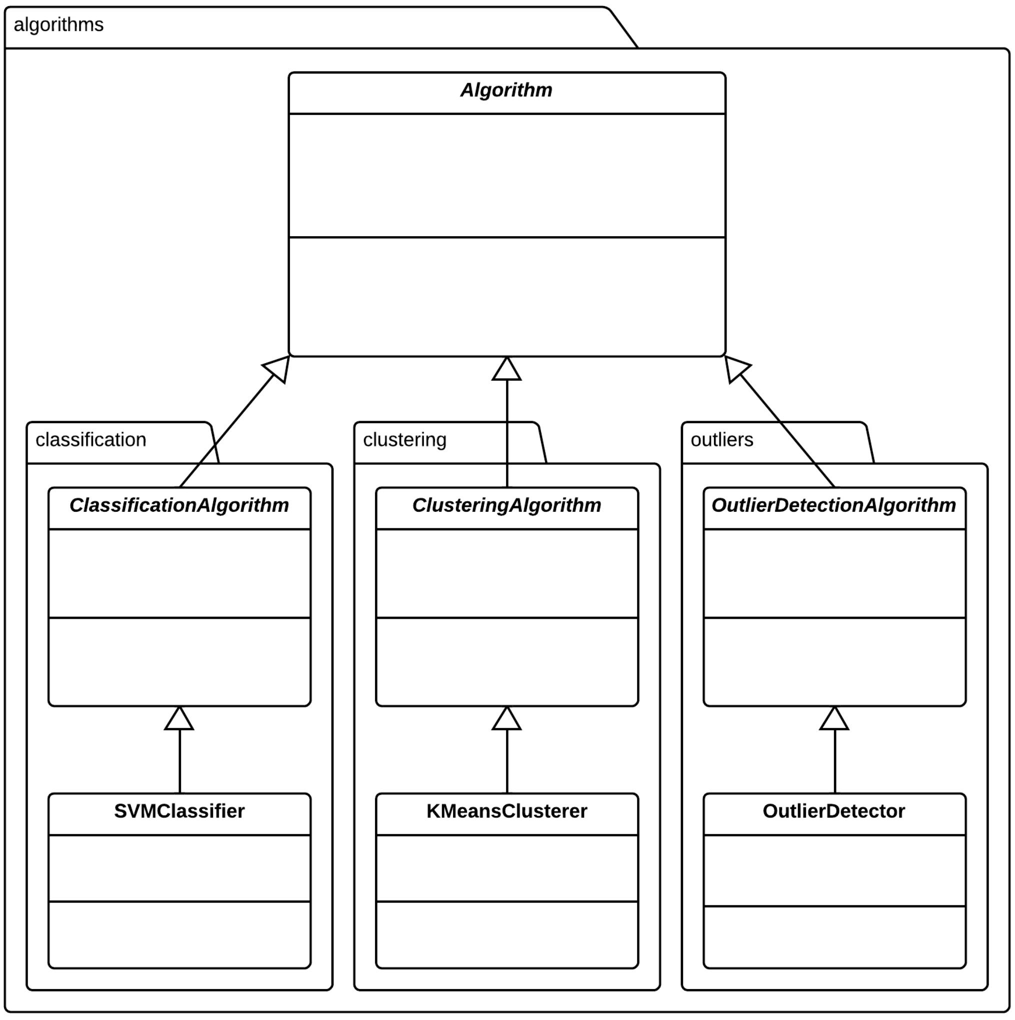


Figure 4.2: Structure of the `algorithms` package

The Abstract Subclasses of Algorithm

At the second level of abstraction within the `algorithms` package, there are the abstract classes ClassificationAlgorithm, ClusteringAlgorithm, and OutlierDetectionAlgorithm (for the different types of machine learning algorithms that have been included in the application), all of which extend the Algorithm superclass. Again, due to these classes being abstract, no objects of their types can be instantiated. The ClassificationAlgorithm class contains class members that are common to all classification algorithms. These include (but are not limited to) references to the Classifier and

Evaluation objects, fields for storing the actual and the predicted class assignments, as well as additional instance variables for holding the reduced training set and the test set (in case the user specifies that the classification algorithm should be trained on a subset of the original training set, and evaluated on the remainder of it). The ClassificationAlgorithm class provides implementations of the following methods:

- SETTARGETLABEL (for setting the target label – the data attribute for the classifier to predict)
- NOMINALISEORDISCRETISEINSTANCES (for nominalising a numeric attribute, usually the class attribute, and for discretising it if it contains more than five unique values, or classes)
- SETEVALUATIONOPTION (for setting the desired method for evaluating a classifier; options include training set, test set, and cross-validation)
- SPLITDATASET (for splitting the training set in two parts of specified proportions, if requested by user)
- TRAIN (for training a classifier)
- EVALUATE (for evaluating a classifier)
- Getters for the *actualClassAssignments*, *predictedClassAssignments*, and *testSet* fields

Likewise, ClusteringAlgorithm contains class members that are common to all clustering algorithms. For example, it provides method implementations of the TRAIN and EVALUATE methods (both of which were defined as abstract in Algorithm). The provided implementations of the aforementioned methods would be inherited by all classes that extend ClusteringAlgorithm. At its present state, the OutlierDetectionAlgorithm class does not make any actual contribution to the functioning of the produced application, but instead exists solely for aesthetic purposes (so that there is a more clearly defined structure within the `algorithms` package, and so that instantiable classes that provide access to implementations of outlier detection algorithms do not have to inherit directly from Algorithm).

Instantiable Classes That Provide Access to Algorithm Implementations

At the lowest level of hierarchy within the `algorithms` package, there are the SVMClassifier, KMeansClusterer, and OutlierDetector classes. Objects of their types can be instantiated, in order to provide access to implementations of different machine learning algorithms. At this hierarchical level of the `algorithms` package, any methods that have been implemented within a class are unique to this class – the most notable example of this being the SETOPTIONS method, for which all of the SVMClassifier, KMeansClusterer, and OutlierDetector classes provide their own implementations.

There is one more class within the `algorithms` package – OutlierEvaluation. An object of its type is returned by the EVALUATE method of the OutlierDetector class. This object consists of class members that provide access to statistical information about the evaluation results of outlier detection algorithms. The OutlierEvaluation class has a somewhat unique purpose within the `algorithms` package. (It is different from the other classes within the package.) This could be conceived as an argument for moving it to a separate package, but this argument was discarded because having a package that contains just a single class is generally considered to be a bad practice in software engineering.

4.3 Plotting Visualisations of Algorithm Results

The main goal for the produced application (in this realm) was that its visualisation capabilities needed to be more advanced than the standard set by Weka (which normally uses 2-D scatter plots for this purpose). Analysis of this issue led to the conclusion that the ability to represent more data dimensions (than Weka) is one potential area for improvement over Weka's visualisation capabilities. In its final design, the produced application has the capability to plot interactive 3-D scatter plots, which is an improvement over Weka's standard plotting and visualisation capabilities.

4.3.1 Choice of a Plotting Library

Experimenting with Weka revealed that its plotting capabilities were somewhat limited. Weka (normally) uses simple 2-D scatter plots for visualising any-dimensional data. It was evaluated that this was an area in which the produced application could make a substantial improvement (compared to Weka). Hence, a requirement was added that the produced application should emulate and extend the plotting capabilities traditionally provided by Weka. With this aim in mind, several external Java plotting libraries were surveyed. (Please refer to subsection 2.7.1 of this document for the survey of Java plotting libraries.)

The following criteria were used in deciding which plotting library to include in the produced application:

- It must provide support for exporting images of plots in the most-popular image formats (i.e., JPEG, PNG, etc.).
- It must be able to generate interactive 3-D scatter plots.
- It should be able to generate contour plots.

Not all of the features outlined by the selection criteria above ended up being implemented in the produced application, but all of the aforementioned criteria were taken into account when choosing a plotting library for inclusion in the produced application. Ultimately, it was decided that out of all surveyed Java plotting libraries, Jzy3d was the one that best satisfied the application's plotting needs (as set out by the requirements above). Thus, Jzy3d was selected as the Java plotting library to be included in the application.

4.3.2 The `visualisers` Package

This package contains classes that are responsible for providing various visualisation services to the application. For example, classes within this package can generate interactive scatter plots (which can be used to plot the results of many machine learning algorithms) and ROC curves (which can be used to evaluate the performance of classifiers in bi-class classification problems). (Please refer to Figure B.3 in Appendix B of this document for a UML class diagram of this package.)

The Interactive3dScatterPlot Class

Instances of this class render interactive three-dimensional scatter plots in order to provide visualisations of the results of machine learning algorithms. (Such visualisations are only generated on user request.) The Interactive3dScatterPlot class relies on the Jzy3d library to achieve its objectives. It creates a list of pickable points (points that can keep track of mouse clicks on themselves), and stores a reference to the memory address of each created point. Each point represents a data instance. The coordinates of each point (on the Cartesian X, Y, and Z axes) signify the position of the data instance that a given point represents on three dimensions of the data. A point's colour indicates the class assignment of the data instance that it represents (with points that have the same colour representing data instances that belong to the same class). Each pickable point is assigned an event listener and an ID, so that the application can monitor mouse clicks on the pickable points.

The Interactive3dScatterPlot class also has the public method UPDATEPOINTS, which deletes all previously created points and creates a new set of pickable points with appropriate coordinates (with respect to the data instances that the new points represent). During this process, all event listeners that belonged to the old points are also deleted, and each newly created point is assigned its own event listener and ID. This method is used every time a user of the application requests to change one of the data dimensions (attributes) that is measured on the Cartesian X, Y, or Z axes.

The InstanceInfoFrame Class

This class extends JFrame, and is responsible for displaying descriptive information about the data instances that were picked by a user on the 3-D scatter plot that is used for visualising the results of machine learning algorithms. Its main component is a text area, and it provides a public method for setting the text content of this text area. Every time a user picks (clicks on) at least one data instance on the plot, one of two things would happen:

(a) if no InstanceInfoFrame object is currently available, the PickablePointsScatter3D object instantiates one and sets its text area to contain information describing the picked instances.

(b) if an InstanceInfoFrame object is already available, the PickablePointsScatter3D object updates its text area to contain information describing the last picked instances.

The ROCcurvePlotter Class

This class generates ROC curves of the performances of classifiers at different classification threshold levels. The generated ROC curves and the frames that contain them are produced by the Weka API. (The code within this class that is used to achieve this functionality, i.e., the code in the body of the PLOTROCCURVE method, has been copied in its entirety from [26].) The application only generates ROC curves on user request, and users of the application are only given the opportunity to request the generation of ROC curves in cases of bi-class classification problems (because ROC curves are only applicable in such cases). (This happens very rarely when the Full Dataset is being analysed, due to it posing very few bi-class classification problems.)

4.4 Graphical User Interface

Two alternative approaches to the design of the application's GUI were considered. One was for the application to have a single-state GUI, in which all components of the GUI are visible at all times. The other one was for the application to have a step-by-step multi-state GUI, in which the choices made by users of the application determine what content would be displayed on the GUI next. The first option was ruled out as being non-user-friendly, due to the GUI of the application having to be too cluttered. A step-by-step multi-state GUI was thought to be easier to use, which was a very important factor in deciding which GUI approach to follow (considering that most users of the application would likely be new to machine learning). Having chosen the type of GUI for the application (a step-by-step multi-state GUI), the next step was to actually design the application's GUI.

Several Java GUI frameworks were considered for use in the produced application – namely JavaFX, Java Swing, and Java AWT. The conducted evaluation of the three Java GUI frameworks revealed that JavaFX is the most sophisticated and advanced one among the three, that Java AWT is the most primitive one among them, and that Java Swing is sufficient for building GUIs of high quality. Java AWT was quickly ruled out as being the weakest alternative. In the end, it was decided that the GUI of the produced application would comprise of components from the Java Swing library. The main reason for this choice was that the developer of the application already had knowledge of and experience with Java Swing, whereas he would have had yet to learn JavaFX, which would have drastically increased the risk associated with the project (given its relatively short time frame).

4.4.1 The `gui` Package

This package contains classes that are responsible for the appearance of the application's GUI and for managing user interaction with the application's GUI. (Please refer to Figure B.4 in Appendix B of this document for a UML class diagram of this package.)

The Controller Class

This class is the logic unit of the application. It manages interaction between the application and its users and controls the program flow. It is perhaps the most important class of the application. It primarily consists of event handlers for various events that could be triggered by users of the application through their interactions with the GUI of the application. The Controller class also keeps track of and controls the application's state with its *state* instance variable. Every time the state of the application changes, the Controller class asks the application's GUI to refresh itself with respect to the new state that the application is in. The Controller class also instantiates and controls Algorithm objects – feeding them data instances (that have been previously fetched from the database), setting their options (parameters), ordering to train them, and ordering evaluations of their performances. A final responsibility of the Controller class is to control the generation of AboutFrame objects. (You can read more about the AboutFrame class in the last paragraph of this subsection.)

The MainView Class

This class consist mainly of Java Swing components, and is responsible for the appearance of the application's GUI. Although some details of the design of the GUI changed during the implementation phase of the project, the general layout of the GUI remained remarkably consistent throughout the project. (Please refer to Figure 4.3 for a wireframe of the layout of the application's main view.) The main goal for the design of the GUI was that it had to be simple to use, and the general layout of the GUI was designed with this goal in mind. The following steps were taken to ensure the high usability of the application's GUI:

1. The GUI components are grouped on the main frame based on their functionality.
2. The GUI components are well spaced out (in order to ensure that the GUI is not cluttered).
3. Most GUI components show tooltips on mouse-hover (in order to provide users of the application with advice on how to effectively use the program).

The main frame of the user interface is divided into four areas. At the very top, there is the menu area. The application's menu is fixed and does not change throughout the course of a single run of the application. Underneath the menu, there is a dedicated JPanel area (the *topPanel*) that holds a label for displaying information and instructions to the user (the *infoLabel*). The text of the *infoLabel* is updated every time the application changes its state. Underneath the *topPanel*, there is another JPanel area (the *middlePanel*). It is used for holding Java Swing components that allow users of the application to indicate their preferences regarding the program flow and to customise the machine learning algorithms that have been provided by the application. The content of the *middlePanel* is also updated every time the application changes its state. Finally, at the bottom of the main frame, there is a third JPanel area (the *bottomPanel*), which holds a label for displaying the current state of the program (the *currentStateLabel*) and a set of navigation buttons for the following operations:

- Start Over
- Go Back
- Next

The text of the *currentStateLabel* is continuously updated in order to reflect the current state that the application is in. The navigation buttons remain unchanged throughout a single run of the application, with the exception of some navigation buttons getting disabled for the duration of certain states of the application.

The most important method in the MainView class is UPDATEVIEW. It is called from the Controller class with the purpose of instructing the application's MainView object to refresh the application's GUI with respect to the new state that the application is in. The UPDATEVIEW method accepts a String parameter *state* which is used to notify the MainView class of what the new state of the application is, so that the MainView object knows how to update the application's GUI accordingly.

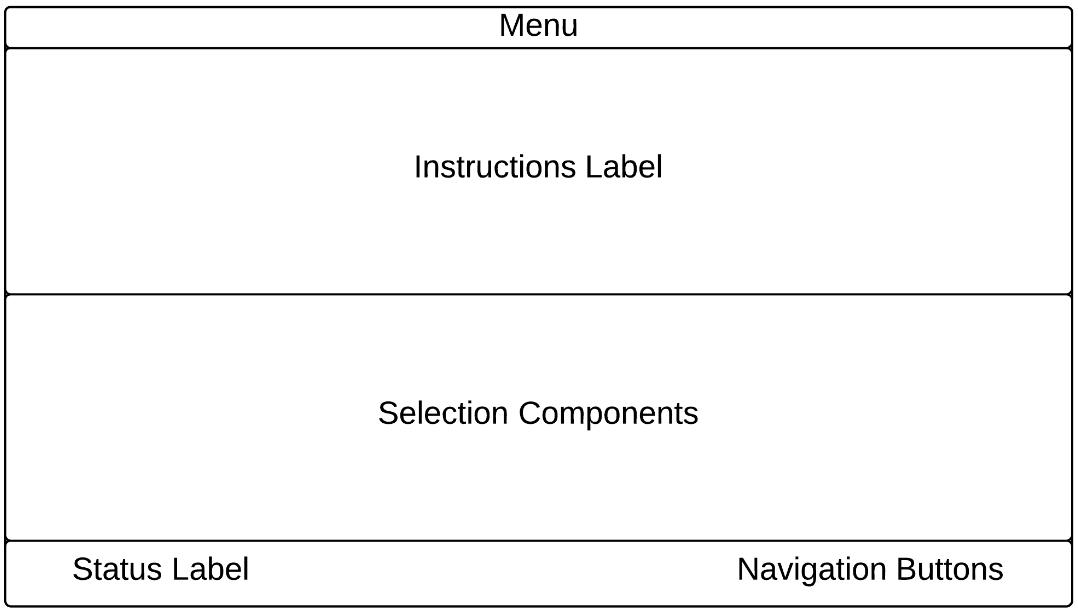


Figure 4.3: Wireframe of the layout of the application's main view

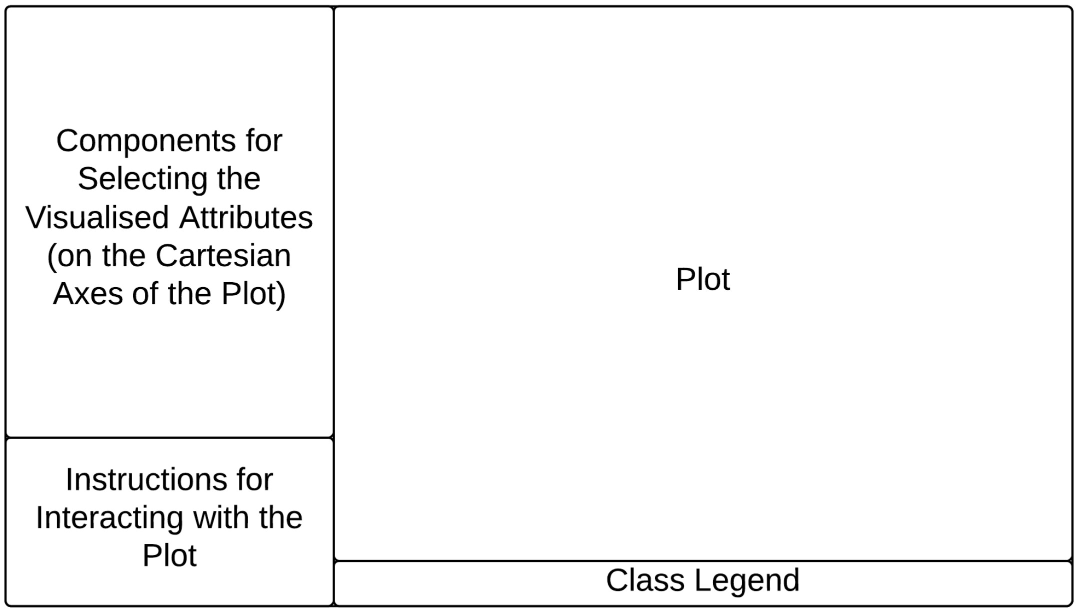

Components for Selecting the Visualised Attributes (on the Cartesian Axes of the Plot)
Plot
Instructions for Interacting with the Plot
Class Legend

Figure 4.4: Wireframe of the layout of the application's visualisation view

The VisualisationView Class

This class defines an additional view that is used to provide an user interface for an Interactive3dScatterPlot object (i.e., an interactive 3-D scatter plot). It is constructed (by the application's Controller object) whenever a user of the application requests to see a visualisation of the results of a machine learning algorithm. Users of the application can request to visualise the results of all of the provided machine learning algorithms, with the exception of classifiers that have been evaluated using cross-validation (because due to the nature of cross-validation, it is not possible to effectively visualise cross-validation misclassification errors on a 3-D scatter plot). The program is designed in a way that only one VisualisationView window (and one interactive 3-D Scatter plot which is embedded in it) can be open at a time.

The layout of a window that is constructed by a visualisationView object is divided into four areas: upper-right, lower-right, lower-left, and upper-left. (Please refer to Figure 4.4 for a wireframe of the layout of the application's visualisation view.) The upper-right area consists of a plot (which is used for visualising the results of a machine learning algorithm) generated by an Interactive3dScatterPlot object, which is embedded in a JPanel (the *RightPanel*). The lower-right area consists of a set of labels that collectively provide a legend of which colour (of the scatter points) corresponds to which class assignment (of the data instances). The lower-left area contains another set of labels that are used to provide instructions to users of the application on how to interact with the 3-D scatter plot visualisation. The upper-left area contains three combo boxes that provide users of the application with the opportunity to change the data attributes that are displayed on the Cartesian X, Y, and Z axes of the 3-D scatter plot visualisation, and a button that allows them to confirm their choice (the *actualisePlotButton*). Once the *actualisePlotButton* is clicked, the 3-D scatter plot visualisation on the upper-right is updated to reflect the user preferences.

The AboutFrame Class

The AboutFrame class is a simple view class that extends JFrame. Its sole purpose is to generate a window that displays descriptive information about the application (on user request). The application's design restricts the number of open AboutFrame windows to one (i.e., no more than one AboutFrame window can be open at a time).

4.5 Database Design and Interaction with the Database

The application stores its datasets in an embedded database. It communicates with its database through a jdbc-driver (which is technically a wrapper for odbc). This methodology allows different RDBMS to be accessed in a universal way from within Java code. Design decisions needed to be made about which RDBMS and library for reading Microsoft Excel .xls files the application should use. The need to use a library for reading Microsoft Excel .xls files arises from the fact that the Full Dataset is originally in .xls format. Hence, the content of the Full Dataset needs to be read into primary memory before it can be inserted in the database. (This process would have to be repeated every time a new dataset that is in .xls format needs to be inserted in the database.)

Another design decision that had to be made was whether to model and change the format of the data contained within the Full Dataset, or whether to insert the Full Dataset into the application's database as it is originally (i.e., without changing its format or the way it looks). The latter option was chosen because of two reasons. **The first reason is** that this option is more straightforward and easy to implement than the alternative. **The second reason is** that this option could readily be applied to any new dataset that would need to be inserted in the application's database (and not just to the Full Dataset), in contrast to the alternative option.

4.5.1 Choice of a Relational Database Management System

Several alternatives were considered when deciding which Relational Database Management System (RDBMS) to include in the application. MySQL (now owned by Oracle Corporation) is "the world's most popular open source database", being relied on by many of the world's largest organizations including Facebook, Google, and Adobe [17]. Oracle Database is a proprietary database and Oracle Corporation's flagship product [16]. The main strengths of MySQL and Oracle Database are that they are incredibly robust, scalable, sophisticated, and offer extensive functionality. One major limitation of MySQL and Oracle Database is that their complexity makes them somewhat more difficult to use than more lightweight RDBMS such as SQLite and Apache Derby/Java DB.

Java DB is an Oracle supported version of the Apache Derby RDBMS. The strengths of Java DB are that it is full-featured and easy-to-use, it is easily embeddable in applications, and that it is included in the Java Development Kit (JDK) by default [15]. One potential weakness of Java DB is that it does not scale up as well as some of the more sophisticated RDBMS (such as MySQL and Oracle Database) do. However, the aforementioned scalability limitation of Java DB was deemed to be irrelevant to this project and the produced application, because by design each copy of the application has its own copy of the application's database. Java DB was selected as the RDBMS to be used by the application, because it fully satisfied the application's database needs while being arguably the simplest RDBMS (from the ones that were considered) to use.

4.5.2 Choice of a Library for Reading Microsoft Excel .xls Files

The Full Dataset is originally stored in Microsoft Excel's .xls file format. The application is designed so that datasets would be queried from a database, so a method to insert the Full Dataset (and other similar datasets) in the embedded database of the application had to be devised. The option of converting the Full Dataset into a format that Java or the Java DB RDBMS support reading from was considered, however it was decided that while this option would decrease the application developer's workload, having to convert the file formats of the datasets that are to be inserted in the database would place an unnecessary burden on users of the program. An initial survey of the Java API revealed that it contained no classes or methods for reading .xls files, so external libraries that provide the desired functionality had to be considered.

Researching external libraries for interacting with Microsoft documents (and specifically with Microsoft Excel .xls files) revealed that there were several Java libraries that provided such functionality. JExcel (developed by TeamDev) allows its users to "easily display, create, print, read, write or modify [both .xls and .xlsx files]" [23]. JExcel's main strengths are that it is professionally developed and supported, feature-rich, and that it is a comprehensive tool for interacting with Excel spreadsheets [23]. The main drawback of JExcel is that it is a proprietary software that costs upwards of £450. Java Excel API (JExcel API) is a "mature, open source Java API enabling developers to read, write, and modify Excel spreadsheets dynamically" [9]. Judging by its website, and compared to the competition, JExcel API appears to be outdated (it does not support the latest MS Excel file formats) and relatively poorly supported.

Apache POI is yet another Java library for reading from and writing to Microsoft documents – MS Excel, MS Word, MS Powerpoint files, etc., are all supported by Apache POI [24]. Apache POI is an incredibly extensive and comprehensive library for manipulating Microsoft documents. One limitation of Apache POI is that it is not very simple to use, however there is plenty of support for it (in the form of tutorial videos, documentation, etc.). Extending Apache POI, jXLS is a "small and easy-to-use Java library for writing Excel files using XLS templates and reading data from Excel into Java objects using XML configuration" [8]. The main strength of jXLS is that it makes use of smart optimisation techniques which allow it to achieve complex manipulations of Excel files (that would take many lines of code using Apache POI) in just a few lines of code [8]. Apache POI, however, appears to be better supported than jXLS and there are more tutorials available for it. Apache POI was the Java library capable of reading .xls files than ended up being included in the produced application. The main reasons for choosing Apache POI over the alternatives were its robustness and the presence of tutorials explaining how to effectively use it (which was felt would save precious developer time).

4.5.3 The `dbtools` Package

This package contains classes that provide interactive access to the embedded database of the application. (Please refer to Figure B.5 on page 59 for a UML class diagram of this package.)

The DatasetDatabaseLoader Class

This class can generate the application's database (if it does not exist). It also provides everything that is needed to insert a new dataset (that is in Microsoft Excel's .xls format) in the database of the application. Its public method `insertDatasetIntoDatabase` takes a Microsoft Excel file (an .xls dataset file that has been selected by a user of the application) as a parameter. It then reads the content of the .xls dataset file (that it received as a parameter) into primary memory (by using the Apache POI library), while keeping track of the data type of each attribute of the dataset. After that, it creates a new table named after the dataset in the application's database, and separately inserts the dataset's schema and data records, one at a time.

The DatabaseAccess Class

This class provides a series of methods for interacting with the Java DB database. All queries requesting data from the database (within the application) are executed through the private method QUERYDATABASE of DatabaseAccess objects. This method takes a String argument *query* (which specifies an SQL query), establishes a connection with the database, executes the SQL query against the database, and returns the ResultSet object that it received from the database (i.e., the results of the SQL query) to the code that called it. The DatabaseAccess class also defines several public methods (namely GETNAMESOFAVAILABLEDATASETS, GETNAMESOFFIELDSOFTABLE, and GETNAMESOFFULLDATASETINSTANCES), each of which builds a custom-built SQL query statement, executes it against the database (with the help of the QUERYDATABASE method), converts the results of the SQL query into an appropriate format, and sends them back to where the method was called from.

Similarly, all requested updates to the database (within the application) are executed through the private method UPDATEDATABASE of DatabaseAccess objects. This method takes a String argument *update* (which specifies an SQL update statement), establishes a connection with the database, and executes the SQL update statement against the database. There is just one method within the DatabaseAccess class that calls the UPDATEDATABASE method, the public DELETEDATASETFROMDATABASE method, which takes the name of a dataset table in the database (that a user has requested be deleted) as a parameter, builds a respective SQL update statement, and then calls the UPDATEDATABASE method to execute the SQL update statement against the database.

Chapter 5

Implementation

With detailed requirements and design documents in place, the next phase of the project was to code up the project's software product. This chapter provides a discussion on some of the more interesting events that occurred during the implementation phase of the project. The application was written in Java 7, using the Eclipse IDE. It was tested on Microsoft Windows 7. Git and GitHub were used for version control during the development process.

5.1 Development Methodology

Due to the nature of this development project and the pre-set deadline for its completion, it was clear from its onset that its software product needed to be released in a big-bang approach (i.e., all features of the produced application needed to be fully implemented before the deadline for the project). This precipitated that the application be designed, implemented, and tested as fully as possible before it was deemed ready for its release. With a clear end goal for the project in mind, a suitable software development methodology for the development of its software product was selected, and a tentative project timetable (which was based on the chosen development methodology for the project) was constructed.

The project proposal suggested that the application be developed using a variation of the waterfall development model. The outcome of a project whose software product is developed using this development methodology largely depends on the quality and clarity of the produced requirements and design documents for it (because the aforementioned documents are usually rigidly followed during the implementation phase of such a project). For this reason, a significant period of time at the beginning of the project (three weeks, or one-quarter of the project's total duration) was allocated specifically to these two tasks (i.e., the production of requirements and design documents for the application).

In reality, while the actual development methodology that was used in developing the application generally followed the prescribed waterfall model software development process, it also borrowed certain agile development practices. For example, there was some client involvement in redefining the software requirements during the implementation phase of the project, and integration testing, debugging, and code refactoring sessions were intertwined with the process of implementing the features of the application. Also, partly because of the lately changed software requirements, there was a marked degree of flexibility in the way the application's features were implemented, and the final design of the produced application slightly deviated from the design that was initially planned for it.

5.2 Reading .xls Files and Database Development

The first steps in implementing the project's software product were to generate the application's database, to devise a method for importing datasets (which are in the same format as the Full Dataset) into the database, and to implement methods that provide interactive access to the database. These issues were addressed at the beginning of the implementation phase of the project because the produced application is heavily reliant on the presence of data. For example, the application's machine learning algorithms need data to operate on. Thus, it was felt that it would be beneficial to ensure the availability of data in the application (through the application's database) as early as possible.

Development of the DatasetDatabaseLoader Class

This was the first class of the application to be implemented. It is not reliant on any other modules of the application, and it can readily be reused in other applications. It provides solutions to two problems – it can generate the application's database (in case it does not exist) and it can insert new datasets in the database. The following methods of this class allow it to achieve its core functionality:

- the public method INSERTDATASETINTODATABASE
- the private method READDATAFROMEXCELFILE
- the private method INSERTDATAINTODATABASE

The INSERTDATASETINTODATABASE method accepts an .xls file as a parameter. It inserts the dataset contained within this file in the embedded database of the application. It accomplishes this by first calling the READDATAFROMEXCELFILE method, and subsequently calling the INSERTDATAINTODATABASE method. The dataset has to be on the first worksheet of the .xls file. The name of the dataset is inferred from the name of this worksheet, but it must not start with a number (because the name of a database table in Java DB cannot start with a number, and each dataset is stored in its own table that is named after itself in the application's database). The INSERTDATASETINTODATABASE method returns a boolean that signifies whether the dataset has been successfully inserted in the database of the application.

The READDATAFROMEXCELFILE method relies on the services provided by the Apache POI library to read the content of an Excel file (that it receives as a parameter) into primary memory. For the purposes of the application, the Excel file that this method receives as a parameter is the .xls dataset file that was originally passed to the INSERTDATASETINTODATABASE method. (This method's code was developed with extensive help from the following tutorial on the topic: [18].) The algorithm that this method uses to read the content of an Excel file into primary memory is described below (Algorithm 1).

Algorithm 1 Algorithm for reading the content of an Excel file into primary memory

```
1: Let $datasetReadSuccessfully$ be a boolean local variable that signifies whether the algorithm
   has been successful in reading the content of the Excel file into primary memory.
2: Let $dataTypes[\,]$ be a String array field that stores the data type of each column of the dataset.
3: Let $data[\,][\,]$ be a String matrix field that stores the value of each cell of the dataset.
4: Obtain an input stream to the Excel file.
5: Obtain the workbook of the Excel file.
6: Obtain the first worksheet of the Excel file.        ▷ %comment: It should contain the dataset.%
7: Obtain the number of rows and columns of the dataset.
8: for $i = 0$ to $numberOfRowsOfTheDataset$ do
9:     for $j = 0$ to $numberOfCellsInThisRow$ do
10:        Obtain the data type of cell $j$ on row $i$, and store it in the $dataTypes[\,]$ field.
11:        Obtain the value of cell $j$ on row $i$, and store it in the $data[\,][\,]$ field.
12:    end for
13: end for
14: return $datasetReadSuccessfully$
```

The INSERTDATAINTODATABASE method inserts the content of the .xls dataset (that was previously read into primary memory by the READDATAFROMEXCELFILE method) in the embedded database of the application. If the application's database does not exist, this method generates it. The application's database is stored in a sub-folder of the application's Java project. This method creates a new table in the application's database for each new dataset that needs to be inserted in the database of the application, and then inserts its schema and populates it with the data items of the corresponding dataset.

This method populates tables in the application's database manually, one item at a time. In practice, this process can take a significant amount of time if a very big dataset needs to be inserted (e.g., a 1000 x 1000 or bigger dataset). Taking the project's time frame into consideration, it was decided that the currently used database insertion technique was sufficient for the purposes of the application, although there are more efficient bulk import tools available.

There are two reasons why such tools are not used by the application. **The first reason is** that the Java DB bulk import tools cannot import data from .xls files (so the Full Dataset would have had to be converted into a different file format in order to be insertable into the application's database by the Java DB bulk import tools). **The second reason is** that the Java DB bulk import tools are only capable of inserting data into an existing database table, but they are not able to automatically create new tables and set their schemas.

Development of the DatabaseAccess Class

This class provides interactive access to the application's database. All SQL queries and updates to the database (within the application) are made through objects of its kind. In practice, this class was implemented during a refactoring session in the late stages of the implementation phase of the project. The reason its implementation is discussed here (and not in accordance with the chronological order of events of the implementation phase) is that its functionality is related to the topic discussed in this section.

This class was initially intended to contain methods for querying the application's database only. (For this reason, it was originally named DatabaseQuery.) However, its design changed after the feature that the application would retain datasets that have been inserted in its database between different runs of the application was added. The addition of this feature necessitated that users of the application were provided with the ability to delete dataset tables from the application's database. In order to increase the cohesion of the application's classes and to facilitate code reuse, it was decided to include the new code for issuing SQL updates to the application's database (such as "DELETE TABLE" SQL commands) in this class and to rename it to DatabaseAccess. This allows for all code that provides access to existing tables of the application's database to be contained within a single class.

5.3 Developing the Main View and Adding User Interaction

The next steps were to develop the application's GUI and to add user interaction. The events of this stage of the project took place in the following order:

1. The initial focus was on setting the layout of the application's main view.
2. The components that would be displayed on the main view at each application state were defined after that.
3. User interaction was added shortly thereafter.

Development of the MainView Class

During the design phase of the project, a set of GUI wireframes was produced – one for each state that the application could be in. The main goal of these wireframes was to convey what components would be displayed on the main view at each application state, as well as where on the application's main view each component would be displayed. The application is state-based, so the layout of its main view had to be conducive to the rapid changes in the application's state. The produced wireframes had not taken this into account, and attempting to implement them revealed the infeasibility of the GUI design suggested in them. Therefore, the application's main view was iteratively refined and improved instead.

The MainView class defines the main view of the application's GUI. The first step in the development of the main view was to split it into four parts. A menu was added to the top of the main view. A "Help" category was added to the menu, but no menu items were added at this stage. Three panel areas (i.e., the *topPanel*, *middlePanel*, and *bottomPanel*) were added underneath the menu in order to hold GUI components. Appropriate layout managers were then chosen for each of the three panels:

- GridBagLayout was chosen for the *topPanel*. (It was chosen because it allows the *infoLabel* to be centred both vertically and horizontally.)
- GridBagLayout was chosen for the *middlePanel*. (It was chosen because it offers significant flexibility in the arrangement and placement of GUI components.)

- BoxLayout was chosen for the *bottomPanel*. (It was chosen because it is perhaps the most lightweight layout manager to fully satisfy the needs of the *bottomPanel*.)

The next step was to implement methods that add components to each of the three parts of the main view (i.e. the LAYOUTTOP, LAYOUTMIDDLE, and LAYOUTBOTTOM methods) or update the existing ones if necessary. Each of these methods uses a switch statement in order to update the content of its corresponding panel in accordance with the state of the application. The aforementioned methods define what components are displayed in each area of the main view and in each state. For example, the following components were initially added to the application's start screen:

- The *infoLabel* (which is used to provide users of the application with information and instructions) was added to the *topPanel* (by the LAYOUTTOP method). A "Serif" logical font that is **bold** and *italic* and has a 48 size was selected for the text of the *infoLabel*.
- Radio buttons for choosing which dataset to analyse were added to the *middlePanel* (by the LAYOUTMIDDLE method).
- Navigation buttons were added to the *bottomPanel* (by the LAYOUTBOTTOM method). Initially the "Start Over" button was positioned on the left side of the *bottomPanel*, whereas the "Back" and "Next" buttons were positioned on the right side. (Please refer to Figure 5.1 for an early version of the application's start screen.)

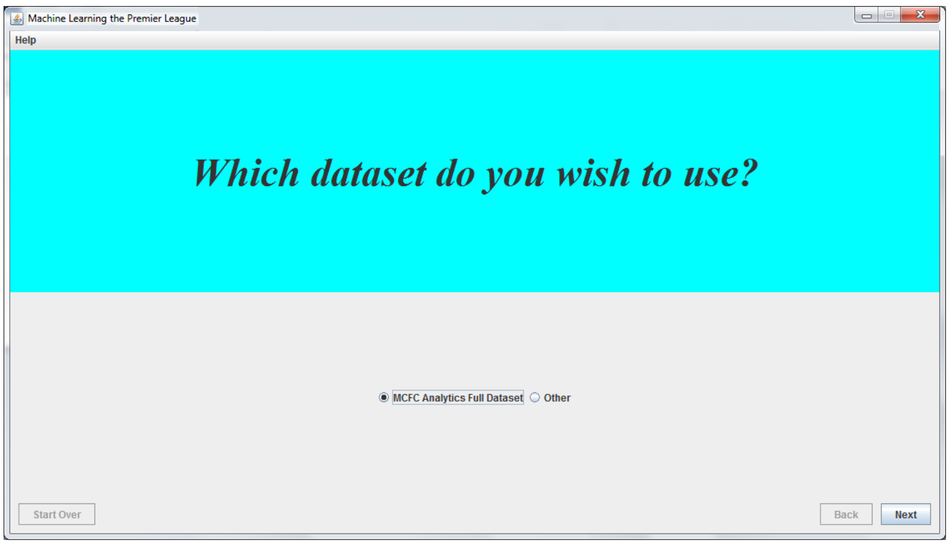

Machine Learning the Premier League
Help
Which dataset do you wish to use?
MCFC Analytics Full Dataset
Other
Start Over
Back
Next

Figure 5.1: Early version of the application's start screen

Specifying a logical font in Java implies that each operating system that the application is ran on would use one of its own fonts that satisfies the properties of the specified logical font for the text in question. This could result in slight inconsistencies in the fonts used from operating system to operating system. This leads to an uncertainty about what the text of the *infoLabel* would look like in operating systems other than Windows 7 (in which the application was developed).

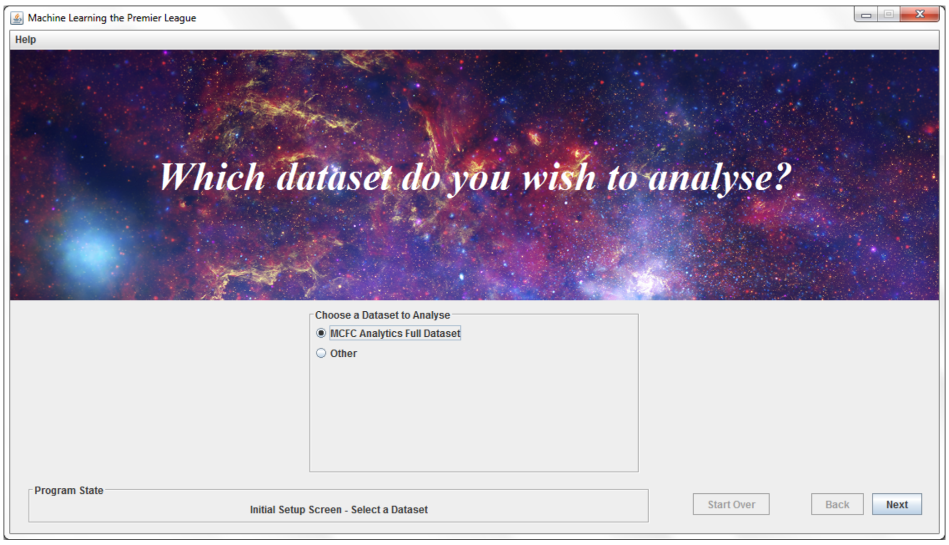
The components contained within the *topPanel* and the *bottomPanel* (i.e., the *infoLabel* and the navigation buttons) are permanent (i.e., the same components remain visible throughout all states of the application) – the LAYOUTTOP and LAYOUTBOTTOM methods just update their appearance (i.e., their text and whether they are enabled) for each application state. The components contained within the *middlePanel* change for each application state – the LAYOUTMIDDLE method replaces all old components contained within it, with new ones, on each change of state.

We will not go into details about all GUI components that could be displayed inside the *middlePanel* of the main view. The application can be in seventeen different states, and the *middlePanel* displays a different set of GUI components in each application state. (There are around fifty GUI components that could be visible inside the *middlePanel* at different times of a single run of the application.) In general, for each supported machine learning task (i.e., clustering, classification, and outlier detection), the *middlePanel* could display components for:

- selecting the features to be used by the respective algorithm. (Examples have been provided for users to choose from as well.)
- setting the parameters of the respective algorithm.
- viewing the results of the respective algorithm and enabling users of the application to take further action (e.g., users can request to "Visualise Results", "Save Results", etc.).

The following changes were then made to the layout of the main view and to the GUI components on the start screen:

- A background image was added to the *topPanel* and the colour of the *infoLabel* on top of it was changed to white.
- All navigation buttons were grouped together on the right side of the *bottomPanel*. (This allows users of the application to navigate the application with less effort.)
- A permanent *programStateLabel* (i.e., a label that notifies users of the current state of the application) was added on the left side of the *bottomPanel*. (Please refer to Figure 5.2 for the final version of the application's start screen.)


Machine Learning the Premier League
Help
Which dataset do you wish to analyse?
Start Over
Back
Next

Choose a Dataset to Analyse
(●) MCFC Analytics Full Dataset
( ) Other

Program State
Initial Setup Screen - Select a Dataset

Figure 5.2: Final version of the application's start screen

Given that the application is designed for non-experts in machine learning, it was important to make the application as easy to use as possible. The state-based design of the user interface of the application is based on the premise that users will continuously have to make choices while using the application. Explanatory tooltips were next added to most GUI components, in order to provide users with advice on how to make choices that would allow them to effectively navigate the application.

Finally, public methods that provide the application's controller (i.e., the Controller class) with control over the main view were added. These include:

- UPDATEVIEW (This method updates the main view by making successive calls to the LAYOUTTOP, LAYOUTMIDDLE, and LAYOUTBOTTOM methods.)
- setTextOfProgramStateLabel (This method enables the controller to change the text of the *programStateLabel*.)
- setTextOfAlgorithmOutputTextArea (This method enables the controller to set the text of the text area that is used to display information about the results of machine learning algorithms.)
- toggleNavigationButtons (This method allows the controller to enable or disable each navigation button.)
- toggleEndButtons (This method allows the controller to enable or disable each one of the "Visualise Results", "Plot ROC Curve", and "Save Results" buttons.)
- disableAllComponentsOfContainer (This method is usually used to disable all components of the *middlePanel*, so that users cannot interact with them when they should not.)

Development of the Controller Class

The next step was to add user interaction. The application manages user interaction in the following way:

1. Event listeners are attached to GUI components.
2. Events are triggered by users of the application through their interaction with the GUI.
3. Events are handled in the Controller class.

The Controller class is the only one of its kind within the application. It provides event handlers for events that are fired by components located in both the MainView and VisualisationView classes.

The *state* field (which is used to keep track of the state that the application is in) was the first class member to be added to the Controller class. The Controller class is required to provide an implementation of the ACTIONPERFORMED method because it implements the ActionListener interface. Hence, ACTIONPERFORMED was the first method to be implemented in the Controller class. It takes an ActionEvent object as a parameter. Within this context, this object represents an interaction event fired by one of the buttons on the application's GUI whenever it was pressed by a user of the application. The ACTIONPERFORMED method checks the source of the event (i.e., which button fired the event), and then calls the event's corresponding event handler. The following event handlers were initially included in the Controller class:

- PROCESSSTARTOVERBUTTONCLICK (This event handler processes clicks on the “Start Over” button.)
- PROCESSBACKBUTTONCLICK (This event handler processes clicks on the “Back” button.)
- PROCESSNEXTBUTTONCLICK (This event handler processes clicks on the “Next” button.)
- PROCESSABOUTMENUITEMCLICK (This event handler processes clicks on the “About” menu item.)
- PROCESSDELETEDATASETBUTTONCLICK (This event handler processes clicks on the “Delete Dataset from Database” button.)

The event handlers that correspond to mouse clicks on the “Start Over” and “Back” buttons change the application’s state, and order the main view to refresh itself with respect to the new state of the application. The event handler that corresponds to mouse clicks on the “Next” button can carry out computational tasks (e.g., it can create and interact with algorithm objects) in addition to these tasks. The PROCESSABOUTMENUITEMCLICK event handler creates an AboutFrame object (which is used to display information about the application). The PROCESSDELETEDATASETBUTTONCLICK event handler deletes a selected dataset from the application’s database.

Additional event handlers were added to the Controller class later on (because they required other functionality to be implemented first). These included:

- PROCESSVISUALISERESULTSBUTTONCLICK (This event handler processes clicks on the “Visualise Results” button.)
- PROCESSPLOTROCCURVEBUTTONCLICK (This event handler processes clicks on the “Plot ROC Curve” button.)
- PROCESSSAVERESULTSBUTTONCLICK (This event handler processes clicks on the “Save Results” button.)
- PROCESSUPDATEPLOTBUTTONCLICK (This event handler processes clicks on the “Update Plot” button.)

The PROCESSVISUALISERESULTSBUTTONCLICK event handler creates a visualisation view (with an interactive scatter plot embedded in it), in order to provide a visualisation of the results of a machine learning algorithm. The PROCESSPLOTROCCURVEBUTTONCLICK event handler orders the generation of an ROC curve (for evaluating the performance of a two-class classifier) in a new frame. The PROCESSSAVERESULTSBUTTONCLICK event handler saves the summary of the results of a machine learning algorithm to a .txt file specified by a user of the application. (Users can specify .txt files through a JFileChooser.) The PROCESSUPDATEPLOTBUTTONCLICK event handler updates the interactive scatter plot that is embedded in the visualisation view. (It is triggered whenever a user of the application requests different data attributes to be represented on the Cartesian axes of the interactive scatter plot.)

5.4 Implementing Algorithms

The next step was to add machine learning algorithms to the application. The abstract superclass Algorithm was the first class of the `algorithms` package to be developed. It was necessary to develop it first, because (by design) all other algorithm-related classes within the `algorithms` package inherit from it. The main challenge in developing it was to determine what class members to include in it. Given that its class members would be shared by all of its children classes, it was necessary to only include class members that would be required by all of its children classes.

The next decision that had to be made was which of its methods to provide implementations for and which ones to declare abstract. The Algorithm class provides abstract method definitions for the SETOPTIONS, TRAIN, and EVALUATE methods. The decision to declare the aforementioned methods as abstract is based on the expectation that all instantiable subclasses of Algorithm would need to contain these methods, but their actual implementations of these methods might have to differ from one another.

In addition, the Algorithm class also provides actual implementations for several methods. These include:

- SETINSTANCEQUERY (This method sets the SQL query statement that is used to request data instances from the application's database.)
- FETCHINSTANCES (This method executes the *instanceQuery* SQL query statement, and assigns the data instances that were returned by the application's database to the *trainingSet* variable.)
- SCALEANDMEANNORMALISEFEATURES (This method uses the "Normalize" Weka filter to bring all numeric features of the training set on the same scale, i.e., [0; 1]. The classes of the data instances of the training set are ignored in this process.)
- RENAMEATTRIBUTESOFINSTANCES (This method renames the attributes of an Instances object. This is necessary because originally the Weka Instances object that is used to hold the training set is not aware of the proper names of its attributes.)

The reason for selecting these particular methods to provide implementations for is that all of them are not only needed by all instantiable subclasses of Algorithm, but also a universal implementation of them would suit any instantiable subclass of Algorithm.

In terms of the fields of Algorithm, the most important ones are *trainingSet*, *originalTrainingSet*, and *featuresScaledAndMeanNormalised*. The last two in this list were not specified in the original design of the Algorithm class, but the need for them became apparent once the SCALEANDMEAN-NORMALISEFEATURES method was added. The reason for this is that while it might be beneficial for all features to be on the same scale (for the computations of the K-Means clustering algorithm and the SVM classification algorithm), it does not make much sense to visualise scaled features on a scatter plot. (For example, it would be confusing for users of the application to see that a player scored 0.81 goals.) Hence, it was decided to create the *originalTrainingSet* field that could store the original training set whose features have not been scaled and mean-normalised. This is useful for the purposes of visualising the results of machine learning algorithms.

5.4.1 K-Means Clustering

The first machine learning algorithm that was added to the application was the K-Means clustering algorithm. It was added first because it is arguably the most simple machine learning algorithm to add from the ones selected for inclusion in the application. The actual implementation of K-Means that the application relies on is called SimpleKMeans and is part of the Weka API. The classes of the application that are related to the K-Means clustering algorithm serve as wrappers for the aforementioned implementation of K-Means.

First, an abstract class ClusteringAlgorithm that extends Algorithm was developed. Its main purpose is to provide implementations for the TRAIN and EVALUATE methods (which were declared abstract in Algorithm) that would be common to all of its subclasses. This is possible because the application is designed to access all of its implementations of machine learning algorithms from (or through) the Weka API. It does not prevent implementations of clustering algorithms that have a different origin (than the Weka API) to be added to the application, but it facilitates the addition of clustering algorithms that are part of the Weka API. The ClusteringAlgorithm class also declares the fields *clusterer* (which is a reference to the default clusterer object) and *eval* (which is a reference to the evaluation object for *clusterer*).

Next, an instantiable class KMeansClusterer that extends ClusteringAlgorithm was developed. It contains a constructor for instantiating its instance variables. In addition, it provides a second implementation of the TRAIN method. This implementation of the TRAIN method extends the default functionality provided by SimpleKMeans by adding support for automatic model selection through running the K-Means clustering algorithm multiple times. It has a different signature than the original TRAIN method (which was inherited from the ClusteringAlgorithm superclass) – the new implementation accepts an int argument *numberOfRuns*, which specifies the desired number of runs of K-Means. The algorithm used in this method (which extends the default functionality of SimpleKMeans, so that it supports automatic model selection through multiple runs of K-Means) is described below (Algorithm 2).

Algorithm 2 Algorithm for extending the default functionality of SimpleKMeans so that it supports automatic model selection through multiple runs of K-Means

```
1: Let $numberOfRuns$ be an int argument that represents the desired number of runs of K-Means.
2: Let $leastSquaredErrorSoFar$ be a double local variable that is initialised to positive infinity.
3: Let $clusterer$ be a global variable that references the default Clusterer object.
4: for $i = 0$ to $numberOfRuns$ do
5:     Let $testRunClusterer$ be a new SimpleKMeans clusterer.
6:     Set the options of $testRunClusterer$ as specified by user.
7:     Randomly initialise the cluster centroids of $testRunClusterer$.
8:     Train $testRunClusterer$.
9:     Let $squaredError$ be a double local variable that is set to the squared error of $testRunClusterer$.
10:    if $squaredError < leastSquaredErrosSoFar$ then
11:        $leastSquaredErrorSoFar = squaredError$.
12:        $clusterer = testRunClusterer$.
13:    end if
14: end for
```

5.4.2 SVM Classification

The second machine learning algorithm that was added to the application was the SVM classification algorithm. The actual implementation of the SVM classification algorithm that the application relies on is part of the LibSVM library. The classes of the application that are related to the SVM classification algorithm serve as wrappers for this implementation (although the application accesses LibSVM through Weka, and not directly).

As with the case of clustering, all class members that are common to all classification algorithms were placed inside a separate abstract subclass of the Algorithm superclass, the ClassificationAlgorithm class. Besides the methods whose implementation is required by the contract specified in the abstract Algorithm class, the following notable class methods were also implemented in the ClassificationAlgorithm class:

- NOMINALISEORDISCRETISEINSTANCES
- SETTARGETLABEL
- SPLITDATASET
- SETEVALUATIONOPTION

The NOMINALISEORDISCRETISEINSTANCES method uses data-preprocessing filters from the Weka API to nominalise or discretise data attributes. The NumericToNominal and Discretize filters from the Weka API are used for nominalisation and discretisation respectively. The first of these filters converts the format of specified data attributes from numeric (i.e., numbers) to nominal (i.e., text). The other filter reduces the number of classes (in classification problems). The use of the NOMINALISEORDISCRETISEINSTANCES method in the application is to ensure that the class attribute has nominal format and no more than five classes. The method's implementation enables it to use both of the aforementioned filters to achieve this objective.

The SETTARGETLABEL method is used for setting the class attribute (or target label) of the dataset (i.e., the data attribute that the classifier will try to predict). By coordinating with the NOMINALISEORDISCRETISEINSTANCES method, the SETTARGETLABEL method also ensures that the class attribute of the dataset is properly formatted. The SPLITDATASET method splits a dataset into two smaller datasets of specified proportions.

The SETEVALUATIONOPTION method enables users of the application to specify the method that should be used to evaluate a classifier's performance. Users are asked to choose among the following classifier evaluation methods:

- Test on training set
- Percentage split: 70% training / 30% test
- Cross-validation

These particular evaluation methods are supported by the application because each one of them is useful in different circumstances. Testing a classifier on the training set is useful for detecting when a classifier is overfitting the training data. Cross-validation is helpful in extensively testing a classifier's ability to make correct predictions, but it takes the most time (out of the three supported classifier evaluation methods) to complete. Evaluating a classifier by training it on 70% of the original dataset, and then testing it on the remaining 30% of the dataset is useful for analysing how well a classifier's hypothesis generalises to unseen examples, but this classifier evaluation method is generally not regarded as being as reliable as cross-validation in that regard.

Finally, an instantiable class SVMClassifier that extends the ClassificationAlgorithm class was developed. It is a relatively simple class – it only contains a constructor (which instantiates the instance variables of SVMClassifier objects) and a setter for the parameters of an SVM classifier. Objects of type SVMClassifier provide access to the full functionality of an SVM classifier.

5.4.3 Outlier Detection

Lastly, a filter for detecting outliers was added to the application. Two alternative implementations of different outlier detection algorithms were initially implemented. The least efficient one was later dropped from the application. The outlier detection algorithm that was implemented first is described below (Algorithm 3). (This algorithm was taken from [13].)

Algorithm 3 Gaussian probability density estimation outlier detection algorithm

$$p(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

$$Outlier\ if\ p(x) < \epsilon$$

$, where\ p(x) = probability\ of\ instance\ x,\ j = feature\ number,\ \mu = mean,\ \sigma = standard\ deviation,\ sigma^2 = variance,\ and\ epsilon = threshold\ for\ classifying\ outliers.$

Testing revealed two inherent limitations of this algorithm. **The first one is** that setting the epsilon parameter correctly requires a lot of fiddling. **The second one is** that the algorithm is not particularly suitable to discrete data (and many attributes of the Full Dataset take on discrete values), because discrete data does not usually follow a normal distribution. Due to these limitations, a decision was made not to include this algorithm in the final application. (It is still included in the `algorithms` package, in the AlternativeOutlierDetector class, in case other application developers want to make use of it.)

An Interquartile Range outlier detection filter (from the Weka API) was included in the application instead. This is not a machine learning algorithm per se (i.e., this algorithm does not learn from data), but it was included in the application regardless, because it is simpler to use (than the other implemented outlier detection algorithm). That is the case because making effective use of it does not require as much fiddling with its parameters as the alternative algorithm does. Despite not being a machine learning algorithm, its implementation (in the OutlierDetector class) was made to satisfy the (machine learning algorithm) contract set out by the abstract Algorithm superclass. Hence, the OutlierDetector class implements many of the same methods that machine learning algorithm classes within the application implement.

The included Interquartile Range outlier detection filter uses the algorithm described below to identify outliers (Algorithm 4). In essence, if the Outlier Factor (parameter of the algorithm) is set to 0, instances below the 25% quartile and above the 75% quartile will be classified as outliers. In contrast, if the Outlier Factor is set to be greater than 0, then fewer instances will be classified as outliers.

Algorithm 4 Interquartile Range outlier detection filter

$$\begin{aligned}&Outlier\ if:\\&x < Q1 - OF * IQR\\&or\\&Q3 + OF * IQR < x\end{aligned}$$

$, where\ x\ is\ an\ instance, OF = Outlier\ Factor, Q1 = 25\%\ quartile, Q3 = 75\%\ quartile,$ $and\ IQR = Interquartile\ Range, difference\ between\ Q1\ and\ Q3.$

5.5 Adding Interactive Scatter Plot Visualisations and Developing the Visualisation View

The application's design called for interactive 3-D scatter plotting (generated by the Jzy3d library) to be implemented in the application. The developer of the application had no prior experience with Jzy3d, and learning how to use it proved to be a daunting task. The available documentation for Jzy3d was scarce (to say the least). The official documentation for Jzy3d omitted to provide any descriptions for most of the methods of the library's classes. There is a paid developer's guide (which costs 49 euros) for using Jzy3d available, however it was decided not to invest in it. Thus, our only point of reference on how to use Jzy3d were the few freely available Jzy3d tutorials and demo code samples. In general, getting Jzy3d to do what was expected of it was a time-consuming process that involved a significant amount of experimentation.

The first part of this process consisted of adding Jzy3d and its dependencies to the application's Java project. The application uses the latest available version of Jzy3d (0.9.1). Its dependencies included Log4j, Gluegen, and JOGL. Jzy3d and all of its dependencies (all of these files were in jar format) were manually downloaded and added first to the application's Java project, and then to the project's build path. Some simple tests were then performed in order to ensure that Jzy3d was installed correctly.

Once the Java project's Jzy3d installation was verified to be working correctly, the next step was to delve deeper into exploring Jzy3d's capabilities. Four classes (PickablePointsScatter3D, PickableSpheresScatter3D, SelectablePointsScatter3D, and ScatterEmbeddableIntoSwing) were developed for experimental purposes, by adapting existing demo code samples. These classes were only developed as a learning tool on how Jzy3d works, and none of them were used in the final application. Experimenting with these classes led to a number of useful insights, such as:

- Although Jzy3d supports JOGL NEWT, Java AWT, and Java Swing chart formats, (in Jzy3d 0.9.1) only the Java AWT implementation was found to function well. The default chart format of Jzy3d is NEWT, but (in Jzy3d 0.9.1) it has apparent issues with mouse control. (Multiple mouse clicks on the chart cause it to freeze.) The Java Swing chart format is deprecated (in Jzy3d 0.9.1). Hence, the Java AWT chart format is the only viable option for use in the application.
- For the purposes of the application, pickable components (i.e., components that register mouse clicks on themselves) are preferable to use than selectable components (i.e., components that can be selected by drawing a rectangular region around them). The reason for this is that (having tested both options) pickable components seem to be easier to interact with than selectable components.
- Pickable points are better suited to the application than pickable spheres. The reason for this is that although both types of components register mouse clicks on themselves, only pickable points display the picking id of the data points that were clicked. It is possible to obtain this same information from pickable spheres, but it is significantly easier to accomplish this task with pickable points.

Development of the Interactive3dScatterPlot Class

The lessons learned from experimenting with the aforementioned experimental classes were applied in the development of the Interactive3dScatterPlot class. This class uses the Jzy3d library in order to provide the application's capability to generate interactive scatter plots. The first step in its developement was to implement Jzy3d's AbstractAnalysis interface. This interface gives access to Jzy3d's *chart* field, which serves as a reference to an interactive scatter plot object.

The code that is used to initiate interactive scatter plot objects was placed inside the INIT method, which is called automatically from the constructor of the Interactive3dScatterPlot class. The INIT method achieves its objective (to initiate an interactive scatter plot) in the following way:

1. The acquired *chart* field (from the AbstractAnalysis interface) is used to instantiate a scatter plot, which is in Java AWT format and has "Nicest" quality. ("Nicest" is the most resource-intensive quality setting for a Jzy3d chart, but it renders charts with the greatest possible level of detail.)
2. The background colour of the scatter plot is set to black, while the Cartesian axes of the scatter plot are set to be white.
3. The Cartesian axes of the scatter plot are named after the data attributes that they would represent.
4. Interactivity is added to the scatter plot:
    a) An AWTCameraMouseController is added to the *chart* object. (This makes the scatter plot responsive to mouse dragging.)

b) A CameraThreadController is added to the *chart* object and the AWTCameraMouseController. (This gives the scatter plot the ability to rotate automatically on user request.)

5. The "take a screenshot of plot" functionality is added. (This functionality was originally implemented through the AWTScreenshotKeyController class of Jzy3d, however a major bug was identified in this implementation. So, in the final version of the application, the "take a screenshot of plot" functionality was implemented from scratch instead.)

6. A call is made to the UPDATEPOINTS method to lay out points (representing the data instances to be visualised) on the scatter plot.

The UPDATEPOINTS method lays out points on the scatter plot. It uses the algorithm described below (Algorithm 5) to accomplish its objective.

Algorithm 5 Algorithm for adding points to scatter plot

```
1: Let $coordinates[][]$ be a double matrix argument that contains the coordinates of all data instances to be visualised.
2: Let $numInstances$ be an int that represents the number of instances to be visualised.
3: Let $chart$ be a field that references the scatter plot object.
4: Let $points$ be a List field that stores references to all points on the scatter plot.
5: Let $numPoints$ be an int that represents the number of points in $points$ and on $chart$.
6: for $i = 0$ to $numPoints$ do
7:     Remove the $i^{th}$ point referenced in $points$ from $chart$.
8: end for
9: Let $points$ reference a new empty List.
10: for $i = 0$ to $numInstances$ do
11:     Let $point$ be a newly-created point.
12:     Set the position of $point$ on $chart$, as specified in the $i^{th}$ row of $coordinates$.
13:     Set the colour of $point$   ▷ %comment: according to the class assignment of the instance it represents%
14:     Set the width of $point$.   ▷ %comment: This varies only for points that represent instances that have been correctly classified by a classifier.%
15:     Add $point$ to $chart$.
16: end for
17: ENABLEPICKING($points$) ▷ %comment: Call the ENABLEPICKING method to make all points referenced in $points$ pickable.%
```

The ENABLEPICKING method makes each point on the scatter plot pickable (i.e., it enables each point on the scatter plot to recognise mouse clicks on itself). It stores the picking id of each point (i.e., the unique identifier of each point) in the *pickingIDs* int field, which is used to assist in identifying which data instance is represented by a point that has been picked. This method also instructs each pickable point to call the PROCESSPICKED event handler whenever a mouse click on itself has been registered.

The PROCESSPICKED method is an event handler for mouse clicks on the pickable points. It is automatically called every time at least one point (on the scatter plot) has been clicked on. With the help of the *pickingIDs* field, this method first identifies the data instances that are represented by the points that have been picked. It then builds a String object that contains textual information about these data instances, and displays this String object in an instanceInfoFrame (which extends JFrame) view. (Please refer to Figure 5.3 for a final version of an interactive scatter plot visualisation embedded in a visualisation view frame.)

Development of the VisualisationView Class

The VisualisationView class was the last class of the `gui` package (and one of the last classes in the application) to be implemented. Its main purpose is to enhance the presentability of the generated interactive scatter plot visualisations. Unlike the main view, the visualisation view is resizable. The reason for this is that the ability to increase the size of an interactive scatter plot can improve the visualisation experience.

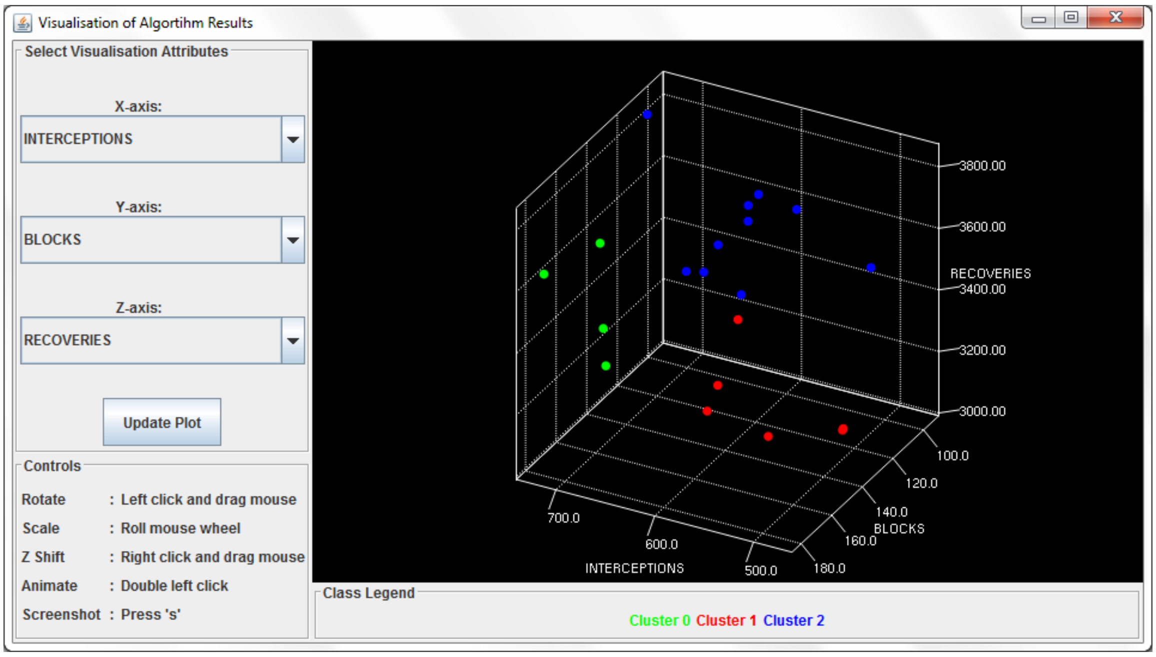
The VisualisationView class implements just two methods: LAYOUTLEFT and LAYOUTRIGHT. Both of these methods are responsible for adding GUI components to the visualisation view. The visualisation view is divided into two main areas – the *leftPanel* and the *rightPanel*. The LAYOUTLEFT method adds components to the *leftPanel*, wherease the LAYOUTRIGHT method adds components to the *rightPanel*.

The *rightPanel* is further subdivided into a *visualisationPanel* and a *legendPanel*. The VisualisationView class receives a reference to an Interactive3dScatterPlot object as a parameter to its constructor. The LAYOUTRIGHT method uses this reference to obtain the canvas of the actual interactive scatter plot visualisation, and then casts this canvas into a Component object. This enables the interactive scatter plot to be treated as any other GUI component, which permits its seamless integration in a JFrame view (such as the visualisation view). The referenced Interactive3dScatterPlot object was added to the *visualisationPanel* area of the visualisation view.

The VisualisationView class also receives a String array of class labels (which refer to the names of the possible classes of the data instances being visualised) as a parameter to its constructor. The LAYOUTRIGHT method uses these class labels to build a set of colour-coded JLabel objects, which together form a class legend. The aforementioned labels were added to the *legendPanel*. The class assignments of the data instances represented by points on the interactive scatter plot can now be inferred from the colour-coded class legend which is formed by these labels.

Similarly, the *leftPanel* is subdivided into an *axeSelectionPanel* area and a *controlsPanel* area. The LAYOUTLEFT method adds a label which contains instructions on how to effectively interact with the embedded interactive scatter plot visualisation to the *controlsPanel*. HTML code was used to set up the text of this label, so that the label could span multiple lines (i.e., rows).

Finally, the LAYOUTLEFT method adds components (to the *axeSelectionPanel*) which enable users of the application to update the data attributes that are represented on the Cartesian axes of the interactive scatter plot. These components include combo boxes for selecting attributes (to be represented on the Cartesian axes of the interactive scatter plot) and a button which users of the application can press to confirm their selection. The *axeSelectionPanel* was implemented using a vertical BoxLayout, which allows its components to space out nicely, even if the visualisation view is resized. All components that are part of the *axeSelectionPanel* are set to always maintain their original sizes (even if the size of the visualisation view is increased), for aesthetic reasons. (Please refer to Figure 5.3 for a final version of the application's visualisation view.)



Visualisation of Algortihm Results

Select Visualisation Attributes

X-axis: INTERCEPTIONS

Y-axis: BLOCKS

Z-axis: RECOVERIES

Update Plot

Controls

Rotate	: Left click and drag mouse
Scale	: Roll mouse wheel
Z Shift	: Right click and drag mouse
Animate	: Double left click
Screenshot	: Press 's'

Class Legend

Cluster 0 Cluster 1 Cluster 2

Figure 5.3: Final version of the application's visualisation view

5.6 Adding Multithreading

Once all of the application's core functionality had been added, the next step was to improve the safety and efficiency of the application's code. With these objectives in mind, the application's code was refactored so that it makes use of multiple threads. The application was made to be multithreaded due to the following reasons:

- So that its Java Swing GUI is only updated from the Event Dispatch Thread (EDT).
- So that its GUI remains responsive to user interaction at all times.
- So that it better utilises multi-core CPUs.

Java Swing is generally not thread-safe, i.e., updating most Java Swing components from any threads other than the EDT can result in unexpected errors. The produced application avoids this issue by ensuring that all updates to the GUI are performed on the EDT. The way it achieves this is by encapsulating all blocks of code that update the GUI inside anonymous inner classes that implement the Runnable interface, and then passing each such class as a parameter to either the INVOKELATER or INVOKEANDWAIT methods of the SwingUtilities class. The difference between these two methods is that the former is asynchronous, whereas the latter is synchronous.

Running computationally expensive tasks on the EDT can cause an application's GUI to freeze and become unresponsive until the EDT completes all of its scheduled tasks. The produced application has several operations that are expensive enough to potentially cause the application's GUI to become unresponsive for noticeable periods of time. These include all operations that involve access to the database, any algorithm operations that involve lots of data or many iterations (such as cross-validation), as well as the process of generating interactive 3-D plots. The produced application solves this issue by having the controller (where most resource-intensive operations are called from) execute all of its code on newly created threads.

Finally, most modern CPUs have multiple cores, which enables them to run multiple processes concurrently. An application that is multithreaded can better utilise all cores of such CPUs, because theoretically processes that belong to different threads can be executed concurrently by different CPU cores. This scenario would not be possible if the application only had one thread of execution.

Chapter 6

Testing and Evaluation

The produced application was evaluated by testing its usability with users. The functionality of the application was thoroughly tested and debugged before the application was tested with users.

6.1 System Testing

Having fully designed and implemented the project's application, the next step was to extensively test it and remove all identified software bugs within it. The purpose of this task was to ensure that the application was operating to acceptable standards before putting it in front of users.

6.1.1 Methodology

Two alternative system testing methodologies were considered before choosing which one of them to use. **The first option was** to test the application using the JUnit testing framework for Java. **The second option was** to test the application by performing a concert of black-box and white-box tests. Ultimately, the latter option was chosen. (The reasons for this choice are explained below.)

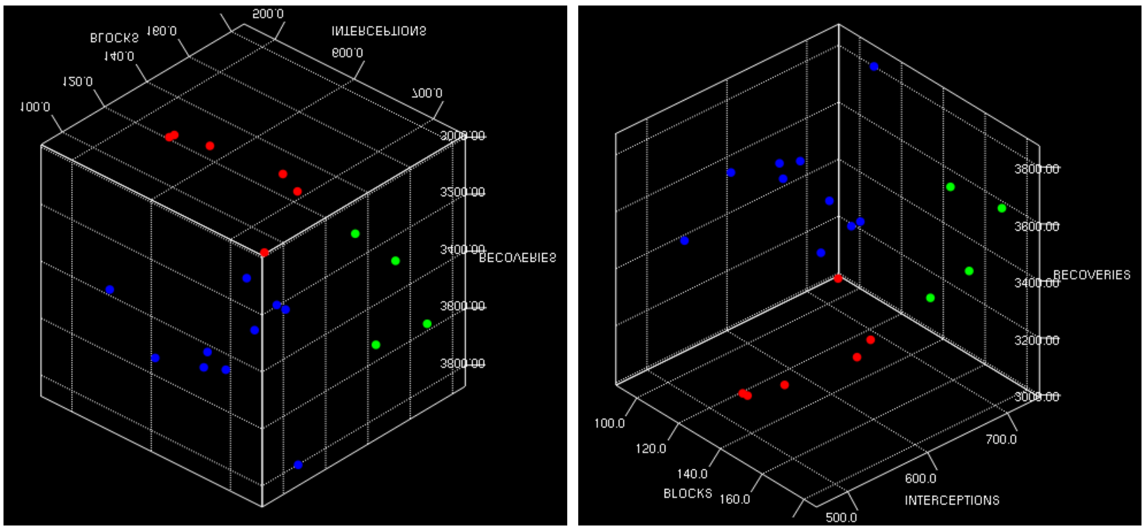
The initial plan was to test the main units of the application by using the JUnit testing framework, which is integrated in the Eclipse IDE (which was used to develop the application). It enables software testers to write unit test cases with which to evaluate whether methods (and the algorithms contained within these methods) have been implemented correctly and function as expected. Ultimately, it was decided that it was not feasible to use JUnit for testing the application. The main reason for this decision is that JUnit is not fully applicable to the software code of the application. The application relies on external libraries for its core components (machine learning and visualisation). These libraries are already well supported and tested (by teams of developers). Moreover, some machine learning algorithms that have been included in the produced application rely on random initialisations. For example, the results of the K-Means clustering algorithm are dependent on the random initialisations of its cluster centroids, which makes it impossible to logically anticipate the exact results of the algorithm (and this is an important aspect of how JUnit tests work). In hindsight, JUnit could have been used to test certain units of the application, but there were faster and simpler methods to accomplish this objective. (Some of these methods are explained below.)

In black-box testing, software testers do not have access to the code of the application they are testing. Instead, they focus on testing the application's functionality. This type of testing was determined to be much more likely to be effective in identifying bugs in the application (than JUnit testing). In contrast, in white-box testing, software testers are provided with access to the code of the application they are testing. The methodology that was used to test and debug the application was to conduct black-box tests in an attempt to identify bugs in the application, and then to conduct further white-box tests (i.e., a low-level analysis of the code) in order to identify what the causes of these bugs are and how the bugs can be eliminated.

6.1.2 Identified and Fixed Bugs

The system testing process led to the discovery and removal of several bugs in the software code of the application. Two such bugs affected the functionality of the application's interactive scatter plot visualisations. The first one of these bugs was that taking screenshots of the generated plot resulted in horizontal mirror images of what the plot actually looks like. (Please refer to Figure 6.1 for an illustration of this bug. Notice how the screenshot in Figure 6.1a, taken before the bug was fixed, is a horizontal mirror image of the screenshot in Figure 6.1b, which was taken after the bug was fixed. Both figures represent screenshots of the same scatter plot visualisation.)

Analysis of the issue revealed that this bug was caused by the *setMustFlipVertically* field of the screenshot image object holding an incorrect value. The program originally relied on the AWTScreenshotKeyController class (from the Weka API) to control the "take a screenshot" functionality. However, this class offered no access to the *setMustFlipVertically* field of its screenshot object, so there was no way for the developer of the application to fix the identified bug within the AWTScreenshotKeyController class. Eventually, this issue was resolved by implementing the desired functionality from scratch and ensuring that the *setMustFlipVertically* field of the screenshot object stored the correct setting (instead of relying on the AWTScreenshotKeyController class for this functionality).



(a) An Incorrectly Taken Screenshot (b) A Correctly Taken Screenshot

Figure 6.1: An illustration of the identified bug in the "take a screenshot of plot" functionality

The second identified bug in the functionality of the application's interactive scatter plot visualisations was that, after a few iterations of updating the data points on the plot, the application was losing its ability to correctly identify data instances that have been picked on the plot. It turned out that this issue was caused by the fact that every new pickable point on the plot was getting an incremented picking ID, even though the points with lesser picking IDs may have been disposed of already. For example, a user of the application may request the plot visualisation of 20 data instances. The points representing these instances on the scatter plot would get picking IDs from 1 to 20. Then, if the user requests a new plot visualisation of 15 data instances, the points representing them would get picking IDs of 21 to 35, even though the first 20 points have now been deleted (when the plot was updated) and no longer exist. This made it impossible for the application to keep track of which data instances were picked using remainder logic. The solution to this issue was to introduce a companion array, which stores the picking IDs of the current points on the scatter plot. This way, the application can identify which data instance has been picked, by getting the array index of the picking ID of the point that represents it.

Another significant bug was identified in the application's machine learning functionality. The bug was that when a classifier was supposed to be trained on 70% of the original dataset and tested on the remaining 30% of the dataset (as requested by a user of the application), in reality, the classifier was actually being trained on 100% of the original dataset and tested on the remaining 30% of the dataset. This bug was identified by observing that in such cases (i.e., when users of the application requested the aforementioned evaluation method) the resulting evaluation of the classifier (on the test set) was achieving an unusually high rate of successful predictions. This bug has now been removed.

6.2 Usability Testing

Once the functionality of the application was thoroughly tested and all identified bugs in the software had been removed, a decision was made to proceed with testing the application with users. The motivation for testing the application with users was to evaluate how usable and useful the application was in reality.

6.2.1 Process

The application was tested with users in a three-step process:

1. Test users were provided with some background information about the project and the application.
2. The users were then asked to attempt to perform a list of tasks using the application.
3. Finally, the users were asked to provide some feedback about their experience with using the application.

(Please refer to Appendix C of this document for the evaluation documents that were used in each step of the usability testing process.)

The background information about the project and the application that test users were provided with was meant to just get users acquainted with the project and the application's goal. It deliberately avoided supplying too much information on machine learning or on how to use the application. The reason for this is that if test users knew too much about these things in advance, then they would no longer be the type of users the application was originally intended for (which would have invalidated the usability testing results).

The devised list of tasks for users to perform encompasses all three of the general machine learning tasks that the application offers (i.e., clustering, classification, and outlier detection) and it touches on all functionality of the application. Completing all tasks on the list takes fifteen minutes on average. In addition to being asked to complete the list of tasks, test users were also given the opportunity to wilfully play and experiment with the application.

Finally, after users have had the opportunity to complete all tasks on the list, they were asked to answer a seven question questionnaire about their experience with using the application. The first six questions of the questionnaire are closed-ended (i.e., a list of possible answers to choose from is supplied with these questions), whereas the last question of the questionnaire is open-ended (i.e., it gives respondents the opportunity to explain their answers). Four of the closed-ended questions use answer options on a Likert scale (i.e., they prompt users to rate or quantify the extent of a certain construct). In addition to the questionnaire, some informal discussions about the application were carried out with users who helped test the application.

6.2.2 Results

The application was tested with six users in total. The test users were of diverse demographics, encompassing different nationalities and genders. None of them reported having any knowledge of or experience with machine learning. They expressed different levels of interest in football, but all of them were familiar with the game and the Premier League.

Questionnaire Feedback

The design of the application's user interface was rated as "very good" by four users, and as "good" by the remaining two users. All users answered that they think the provided algorithms are useful for analysing the MCFC Analytics Full Dataset (with five test users specifying "very useful"), although comments were made by some users that they were not able to judge this due to not being experts in this area. The quality of the visualisations provided by the application was rated as "very good" by five users, and as "good" by the one remaining user. Four users reported having completely learned what the provided machine learning algorithms do, while the remaining two users reported that they have only somewhat learned what the algorithms do.

Overall, users rated the application positively. Also, the application seemed to serve its intended purpose well. Observing users interact with the application, all of them were able to complete the list of tasks to perform fairly quickly and without much trouble. Also, it was encouraging to see that some users were able to grasp how to do certain tasks they were not even asked to do almost instantaneously. This could only be a positive sign of the usability of the application's user interface.

Comments and Suggestions

One user commented that the provided scatter plots look cluttered at times. While this is true, this is something normal in cases where a large numbers of data instances need to be visualised. Moreover, the visualisation view (and the plot visualisation contained within it) can be maximised if needed (which makes the plot look a little less cluttered). Users also gave a few interesting suggestions about potential improvements that could be made to the application. These include:

- Making it easier to add features [to feature vectors]. (A few test users indicated that adding features one by one and having to scroll is hard.)
- Adding functionality to easily identify players from a given team on a scatter plot visualisation. (This is only applicable at the player level of analysis of the Full Dataset; One test user was a staunch Liverpool F.C. supporter. He really wanted to identify players from this team only on a scatter plot visualisation, and he was not interested at all in seeing instance information about any players from other teams. Presently, the only way to identify players from a given team on the scatter plot is through trial and error, which could be a frustrating and time-consuming process.)

Identified and Fixed Bugs

One significant bug was identified in the application's software code during the usability testing process. The bug was that every time a dataset other than the Full Dataset was analysed after the Full Dataset had been analysed (in a single application run), the application was unable to retrieve data instances from its embedded database. Analysis of this issue revealed that the bug was caused by the *levelOfAnalysis* field of the Controller class being initialised by the Full Dataset (when the Full Dataset was first analysed), and then retaining its value when other datasets were later analysed.

The application treats the Full Dataset and other datasets slightly differently. For example, the *levelOfAnalysis* field is only applicable to the Full Dataset – it is used to indicate whether users want to analyse individual player performances, player performances over the season, or team performances over the season (based on data from the Full Dataset). The retained value of the *levelOfAnalysis* field (from the previous analysis of the Full Dataset) confused the application that any dataset that was analysed after the Full Dataset was in fact the Full Dataset. So, in essence all datasets (that were analysed after the Full Dataset) were being treated by the application as if they were the Full Dataset (which resulted in SQL errors). This bug has now been fixed.

Chapter 7

Conclusion

7.1 Summary of the Project

The project's goal was the production of an application that would enable non-experts in statistical analysis to apply machine learning algorithms for the analysis of the Full Dataset (although the produced application is a general tool that works on other datasets as well). To that end, the produced application successfully achieved what it set out to achieve. The produced application fully satisfies its (pre-set) requirements, and offers at least one extra feature on top of its requirements (i.e., ROC curve plotting). The scatter plot visualisation capabilities of the produced application are more advanced and sophisticated than the default ones of similar applications (such as Weka), and have been received well by test users of the application. Another distinguishing feature of the produced application is that it allows new datasets to be added to (and deleted from) its embedded database, which allows their efficient filtering using SQL.

Even users who know nothing about machine learning are generally able to make good use of the produced application (as indicated by the usability testing of the application that has been carried out). The produced application is somewhat of an exception in that regard, given that most existing applications for machine learning analysis either require some previous machine learning knowledge (e.g., Weka) or are not designed to operate on large datasets such as the Full Dataset (e.g., Teaching Aid). Perhaps Microsoft Azure Machine Learning comes closest to the produced application in that regard, but still, its exclusive focus on predictive analytics makes it quite different to the produced application.

7.2 Suggestions for Further Work

Albeit the project accomplished much, the produced application is far from perfect. There are several improvements that could be made to the application, and there are quite a few new features that could be added to it. Perhaps the most obvious improvement that can be made to the application is to add support for more machine learning algorithms. Presently, the produced application only supports three machine learning tasks (clustering, classification, and outlier detection), and it only includes one algorithm for each of these tasks, so there is plenty room for improvement in this domain. For example, more classification algorithms (such as Logistic Regression, Neural Networks, etc.), more clustering algorithms (such as Expectation-maximisation, Farthest First, etc.), and regression algorithms (such as Linear Regression) could be added to the produced application.

Adding support for new machine learning algorithms to the application is a two-step process. **The first step is** to add an implementation of a new machine learning algorithm to the `algorithms` package. **The second step is** to integrate the newly-added machine learning algorithm with the GUI of the produced application.

The `algorithms` package of the application is designed to be conducive to the easy addition of new machine learning algorithms to it. Given that this package was designed to be compatible with the Weka API, it would be most straightforward to add implementations of machine learning algorithms from the Weka API. However, adapting machine learning algorithm implementations from elsewhere (either from a different Java machine learning library such as Java-ML, or custom-developed implementations of machine learning algorithms) so that they could be added to the `algorithms` package should not pose a significant challenge either.

Integrating newly-added machine learning algorithms with the application's GUI could pose a slightly bigger challenge. The produced application is state-based, so definitions describing how the main view should look at each step of execution of the newly-added algorithm would need to be added to the application's MainView class. In addition, new handlers for events related to components associated with the newly-added algorithm would need to be added to the application's Controller class. This choice of design for the application's GUI makes the application slightly more difficult to extend (than a single-state GUI design would have been), but at the same time it is more usable (than the alternative), which is arguably more important for the GUI design of the application.

Test users of the application requested that two new features be added to the application. **The first one is** an improved method or a tool that would facilitate the addition of features to feature vectors for machine learning algorithms (e.g., some kind of a search box). **The second one is** the functionality to quickly identify all players from a given team on a scatter plot visualisation (when the results of machine learning algorithms that have analysed the Full Dataset at a player level are being visualised). The first requested feature would be useful and relatively straightforward to implement. The second requested feature would require the implementation of some search algorithm. From a user perspective, it could work in the following way: all points on the scatter plot that represent players of a given team could be highlighted (i.e., change their colour) on user request.

At the present time, the produced application is slightly biased toward the Full Dataset in terms of SQL filtering. The application does not currently provide users with the opportunity to build any advanced SQL queries when analysing datasets other than the Full Dataset. For example, the application sometimes applies the SUM and GROUP BY functions when building SQL queries to fetch data instances from the Full Dataset, but no similar functionality is provided in cases where the dataset that is being analysed is not the Full Dataset. Perhaps a more advanced (and universal) facility for building SQL queries could be devised and implemented in a future version of the application.

The application was developed and debugged in Windows 7 (because of which it would likely work best in that operating system). Late system tests of the application on Linux 16 Petra revealed that the application does not function as well (as it does in Windows 7) in this operating system.

The generated interactive scatter plot visualisations are hardware supported, which means that they require that the device on which the application runs has a video card of a certain standard and an appropriate video card driver installed. For example, the application's interactive scatter plot visualisations do not function satisfactory in some default installations of Linux distributions. It is suspected that this is due to the fact that most Linux installations do not use proprietary video card drivers. Also, different operating systems use different fonts which could make the application's GUI look off on some operating systems. The application has not been tested on Mac OS yet, so it is not clear how well the application would work on it. It would be useful to carry out future work aimed at addressing these issues.

Appendix A

Requirements Documents

A.1 Functional Requirements

Datasets:

- The Full Dataset must be embedded in the application and be readily available for analysis.
- It must be possible to load other datasets.
- It must be possible to select which dataset to analyse.
- It should be possible to filter a selected dataset in order to obtain a set of more specific data instances (which to pass to a machine learning algorithm).
- It should be possible to split a dataset into smaller datasets (e.g., into training, validation, and test sets).

Machine Learning Algorithms:

- The application must enable its users to run algorithms for classification, clustering, and outlier detection tasks.
- It must be possible to select which algorithm (or type of task) to run (or perform).
- It must be possible to set an algorithm's options.
- It could be possible (where appropriate) to automatically pick the optimal algorithm options (i.e., to perform automatic model selection).
- It must be possible to evaluate an algorithm's performance.
- It should be possible to view a text summary of the results of an algorithm.
- It would be advantageous if generated text summaries of algorithm results could be saved.

Visualisation of Algorithm Results:

- It must be possible to visualise the results of an algorithm (where appropriate, and preferably in as many dimensions as possible).
- It would be advantageous if generated visualisations of algorithm results could be saved.

A.2 Non-functional Requirements

- The application should not assume any prior machine learning knowledge from its users.
- The application should be easy and intuitive to use.
- The application's GUI should never freeze. (It should always remain responsive to user interaction.)
- Loading new datasets should be as fast as possible (it shouldn't take more than 10 seconds to load a new dataset containing 1000 rows and 100 columns of data).

A.3 Use Cases

Use Case:	Perform Clustering
Description:	This use case represents the functionality of running the K-Means clustering algorithm.
Actors:	All users of the system
Rationale:	The system must allow users to run a clustering algorithm.
Priority:	Must have
Trigger:	When prompted what task the system should perform, the user selects a “Clustering” radio button and then clicks on a “Next” button.
Pre-conditions:	A dataset is loaded onto the system and is selected. (If the Full Dataset is selected, then a desired level of analysis must be indicated as well).
Post-conditions:	The user is provided with the following options: • Visualise the algorithm’s results in an interactive 3-D scatter plot. • Save the summary of the algorithm’s results to a text file. • Go back to the start screen and start over.

Steps:

1. The user selects what features to cluster by, and specifies whether the features should be scaled and mean-normalised.
2. The user specifies parameters for the K-Means clustering algorithm (i.e., the desired number of clusters and the desired number of runs of K-Means).
3. The user is provided with a summary of the results of the K-Means clustering algorithm.

Figure A.1: **Perform Clustering** use case

Use Case:	Perform Classification
Description:	This use case represents the functionality of running the SVM classification algorithm.
Actors:	All users of the system
Rationale:	The system must allow users to run a classification algorithm.
Priority:	Must have
Trigger:	When prompted what task the system should perform, the user selects a “Classification” radio button and then clicks on a “Next” button.
Pre-conditions:	A dataset is loaded onto the system and is selected. (If the Full Dataset is selected, then a desired level of analysis must be indicated as well).
Post-conditions:	The user is provided with the following options: • Visualise the algorithm’s results in an interactive 3-D scatter plot. • Plot an ROC curve of the classifier’s performance. (This option is only available if the number of classes is 2.) • Save the summary of the algorithm’s results to a text file. • Go back to the start screen and start over.

Steps:

1. The user selects what features to make classification predictions based on, and specifies whether the features should be scaled and mean-normalised.
2. The user selects a target label.
3. The user specifies parameters for the SVM classification algorithm (i.e., the desired kernel type, the desired regularization parameter C, and in case a Gaussian kernel is selected - the parameter of the Gaussian kernel gamma).
4. The user selects the desired method for evaluating the performance of the SVM classification algorithm.
5. The user is provided with a summary of the results of the SVM classification algorithm.

Figure A.2: **Perform Classification** use case

Use Case:	Perform Outlier Detection
Description:	This use case represents the functionality of running the Interquartile Range outlier detection filter.
Actors:	All users of the system
Rationale:	The system must allow users to run an Outlier Detection algorithm.
Priority:	Must have
Trigger:	When prompted what task the system should perform, the user selects an “Outlier Detection” radio button and then clicks on a “Next” button.
Pre-conditions:	A dataset is loaded onto the system and is selected. (If the Full Dataset is selected, then a desired level of analysis must be indicated as well).
Post-conditions:	The user is provided with the following options: • Visualise the algorithm’s results in an interactive 3-D scatter plot. • Save the summary of the algorithm’s results to a text file. • Go back to the start screen and start over.

Steps:

1. The user selects what features to identify outliers based on.
2. The user specifies parameters for the Interquartile Range outlier detection filter (i.e., the outlier factor, which is used to determine the threshold for classifying outliers).
3. The user is provided with a summary of the results of the Interquartile Range outlier detection filter.

Figure A.3: **Perform Outlier Detection** use case

Appendix B

Design Documents

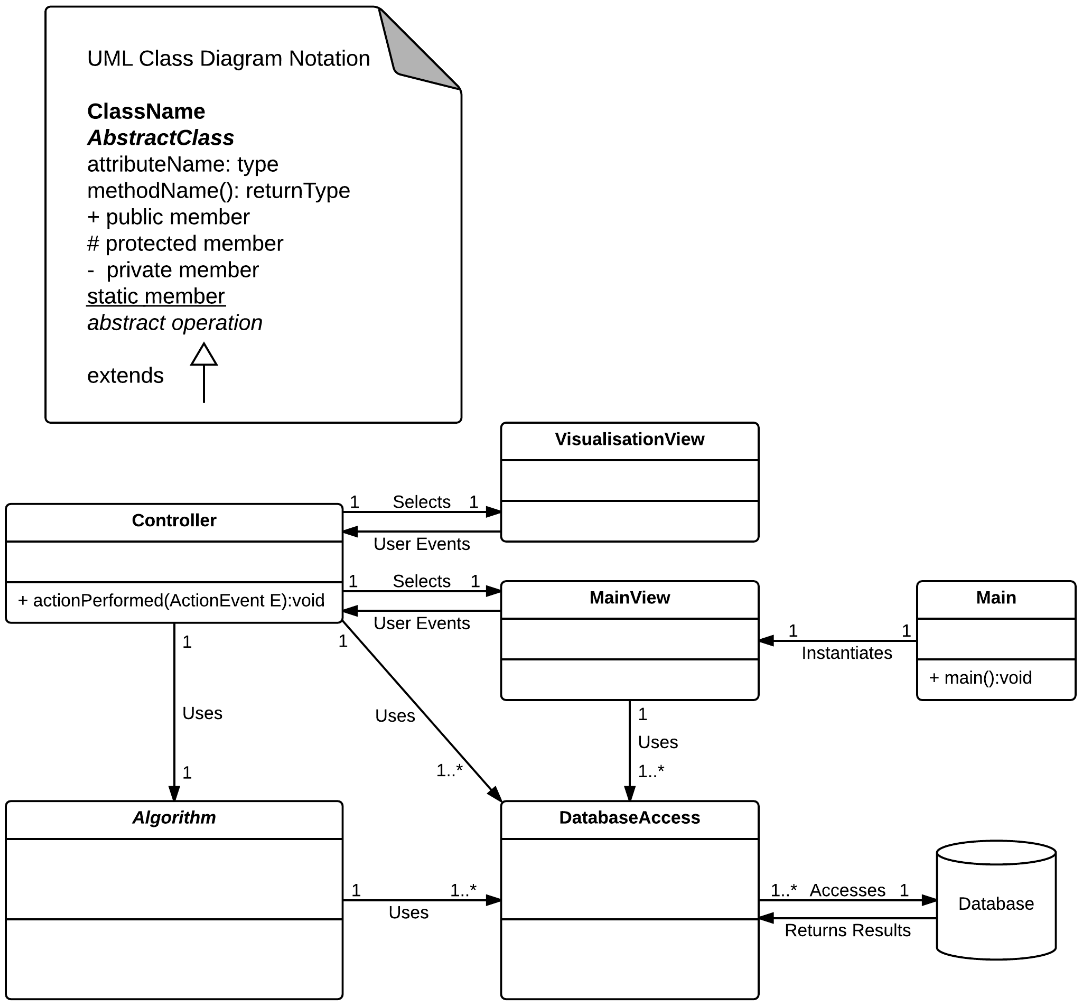


Figure B.1: UML class diagram of the overall software architecture of the application

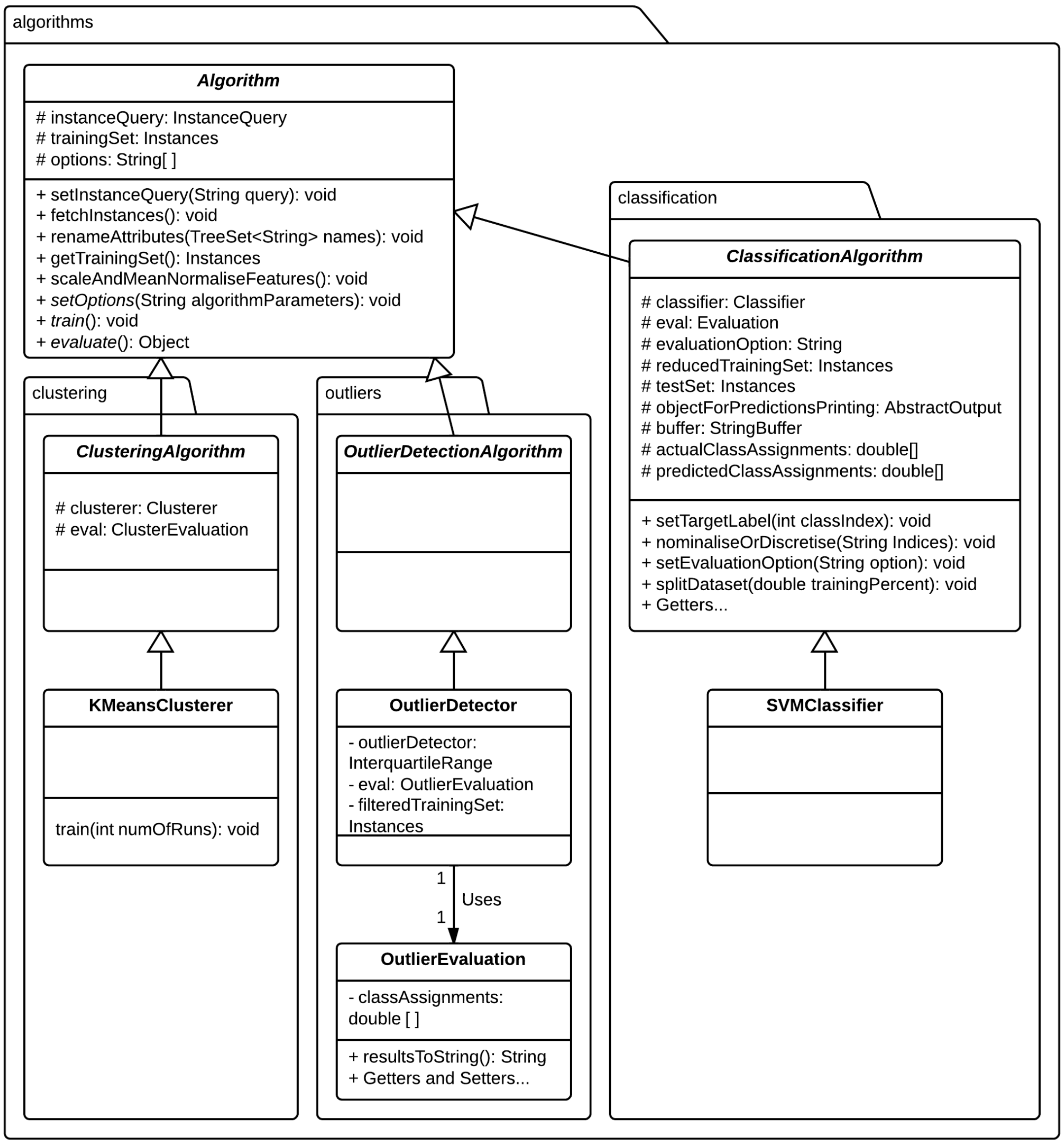

algorithms
Algorithm
# instanceQuery: InstanceQuery
# trainingSet: Instances
# options: String[ ]
+ setInstanceQuery(String query): void
+ fetchInstances(): void
+ renameAttributes(TreeSet<String> names): void
+ getTrainingSet(): Instances
+ scaleAndMeanNormaliseFeatures(): void
+ setOptions(String algorithmParameters): void
+ train(): void
+ evaluate(): Object
classification
ClassificationAlgorithm
# classifier: Classifier
# eval: Evaluation
# evaluationOption: String
# reducedTrainingSet: Instances
# testSet: Instances
# objectForPredictionsPrinting: AbstractOutput
# buffer: StringBuffer
# actualClassAssignments: double[]
# predictedClassAssignments: double[]
+ setTargetLabel(int classIndex): void
+ nominaliseOrDiscretise(String Indices): void
+ setEvaluationOption(String option): void
+ splitDataset(double trainingPercent): void
+ Getters...
SVMClassifier
clustering
ClusteringAlgorithm
# clusterer: Clusterer
# eval: ClusterEvaluation
KMeansClusterer
train(int numOfRuns): void
outliers
OutlierDetectionAlgorithm
OutlierDetector
- outlierDetector: InterquartileRange
- eval: OutlierEvaluation
- filteredTrainingSet: Instances
1
Uses
1
OutlierEvaluation
- classAssignments: double [ ]
+ resultsToString(): String
+ Getters and Setters...

Figure B.2: UML class diagram of the `algorithms` package

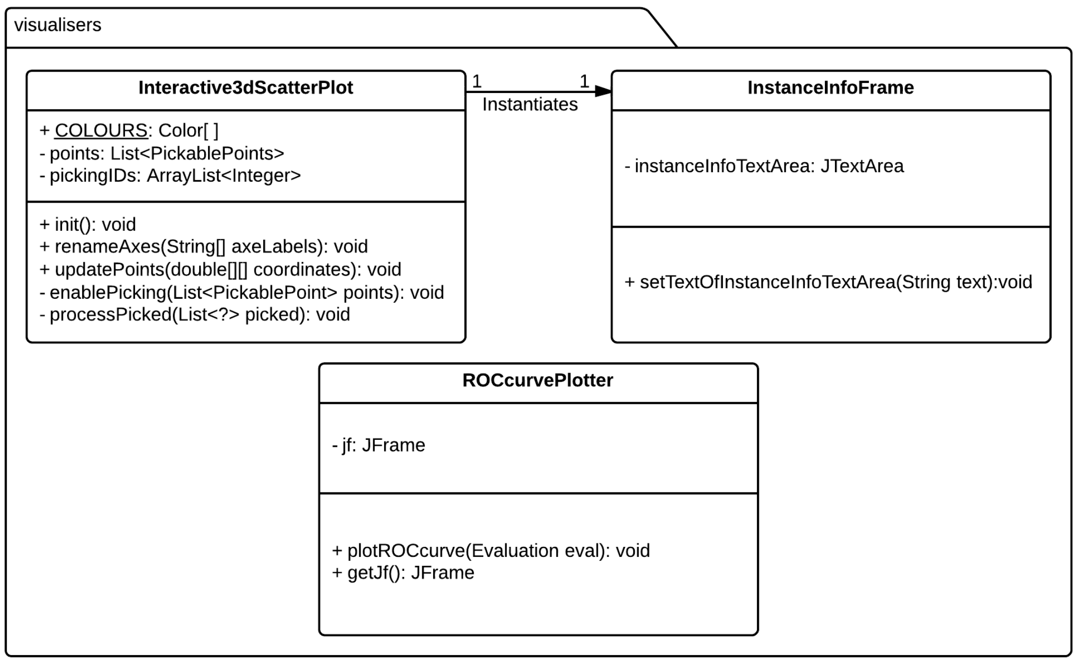

visualisers
Interactive3dScatterPlot
+ COLOURS: Color[ ]
- points: List<PickablePoints>
- pickingIDs: ArrayList<Integer>
+ init(): void
+ renameAxes(String[] axeLabels): void
+ updatePoints(double[][] coordinates): void
- enablePicking(List<PickablePoint> points): void
- processPicked(List<?> picked): void
1
1
Instantiates
InstanceInfoFrame
- instanceInfoTextArea: JTextArea
+ setTextOfInstanceInfoTextArea(String text):void
ROCcurvePlotter
- jf: JFrame
+ plotROCcurve(Evaluation eval): void
+ getJf(): JFrame

Figure B.3: UML class diagram of the `visualisers` package

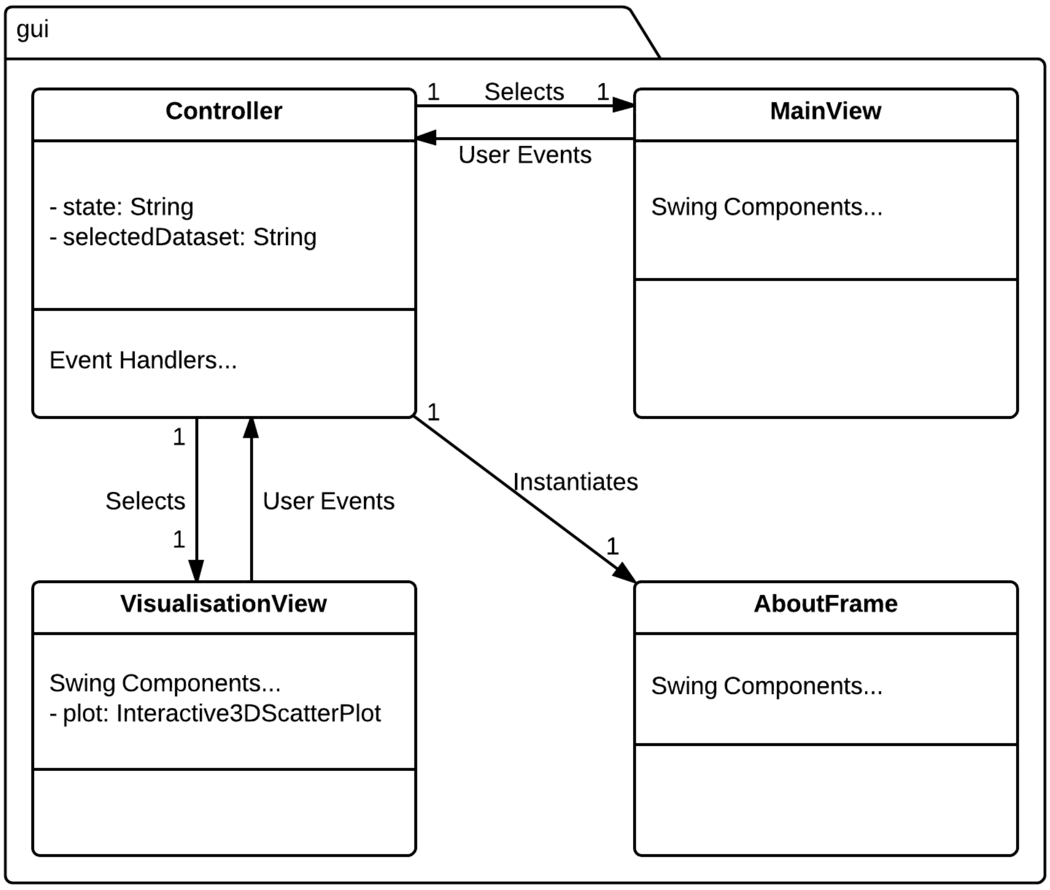


Figure B.4: UML class diagram of the `gui` package

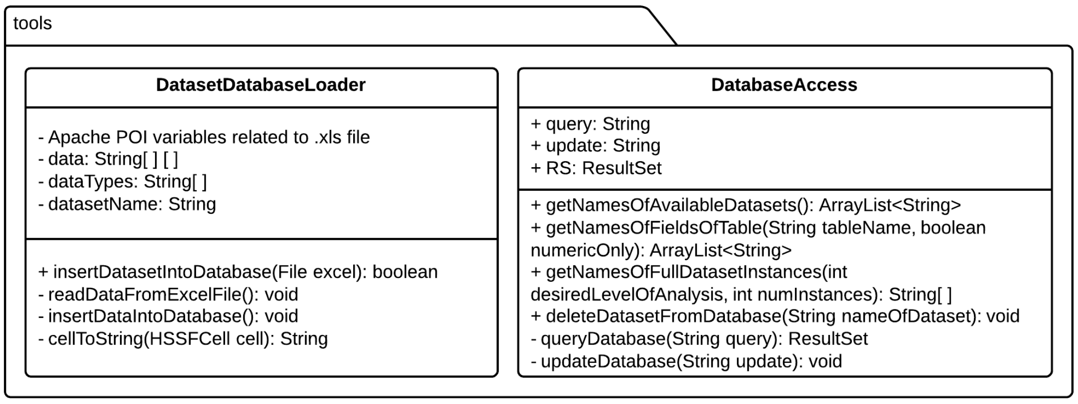

tools
DatasetDatabaseLoader
- Apache POI variables related to .xls file
- data: String[ ] [ ]
- dataTypes: String[ ]
- datasetName: String
+ insertDatasetIntoDatabase(File excel): boolean
- readDataFromExcelFile(): void
- insertDataIntoDatabase(): void
- cellToString(HSSFCell cell): String
DatabaseAccess
+ query: String
+ update: String
+ RS: ResultSet
+ getNamesOfAvailableDatasets(): ArrayList<String>
+ getNamesOfFieldsOfTable(String tableName, boolean numericOnly): ArrayList<String>
+ getNamesOfFullDatasetInstances(int desiredLevelOfAnalysis, int numInstances): String[ ]
+ deleteDatasetFromDatabase(String nameOfDataset): void
- queryDatabase(String query): ResultSet
- updateDatabase(String update): void

Figure B.5: UML class diagram of the `dbtools` package

Appendix C

Evaluation Documents

C.1 Background Information

The application you are asked to test gives you the ability to run powerful machine learning algorithms in order to analyse the MCFC Analytics Full Dataset (as well as other datasets). The MCFC Analytics Full Dataset contains information about all match events that have occurred during the 2011-2012 season of the English Premier League. (The MCFC Analytics Full Dataset is included in application's directory – it might help if you familiarise yourself with it before using the application.) Please pay close attention to the tooltips that are displayed on mouse-hover over most components at each step. This should allow you to better understand what the algorithms provided by the application are doing.

C.2 Tasks to Perform

Please attempt to perform the following tasks:

Task 1 – Run a Clustering Algorithm

1. Analyse the MCFC Analytics Full Dataset.
2. Analyse team performances summed up over the season.
3. Choose to run a clustering algorithm. (Remember to pay attention to the tooltips that are displayed on mouse-hover over the algorithm selection components.)
4. Identify groups of similar items within the data based on the following features:
   - Goals
   - Key Passes
   - Passes Forward
5. Identify 4 groups of similar items within the data.

6. Analyse the results of the clustering algorithm. (Pay attention to how many data items (i.e., teams) have been allocated to each group of similar items within the data.)
7. Visualise the results of the clustering algorithm.
8. Play around with the provided interactive visualisation plot. (You can learn how to better interact with it by reading the instructions that are displayed in the lower left of the visualisation frame).
9. Click on some data points on the interactive visualisation plot, in order to learn more about the instances from the MCFC Analytics Full Dataset that they represent.
10. Take a screenshot of the interactive visualisation plot, and check if it saved correctly.

Task 2 – Run a Classification Algorithm

1. Without quitting the application, go back to the start screen. (Start over with a new data analysis.)
2. Again, choose to analyse the MCFC Analytics Full Dataset.
3. Analyse player performances summed up over the season.
4. Choose to run a classification algorithm.
5. Make predictions based on the following features:
   - Big Chances
   - Successful Passes Final Third
   - Key Passes

   (Pay attention to the provided examples on the left.)
6. Predict the number of goals scored.
7. Use a Gaussian Kernel. (Use the tooltips that are displayed on mouse-hover over the parameter components, in order to learn what the effects of changing the parameters would be.)
8. Have the algorithm learn from 70 % of the data and make predictions on the remaining 30 % of the data.
9. Analyse the results of the classification algorithm. (How many data items have been predicted correctly? How many data items have been predicted incorrectly/misclassified?)
10. Visualise the results of the classification algorithm.
11. Play around with the provided interactive visualisation plot.
12. Pick both correctly and incorrectly classified data instances (by clicking on data points on the interactive visualisation plot), in order to learn more about them.

Task 3 – Run an Outlier Detection Filter

1. Without quitting the application, go back to the start screen. (Start over with a new data analysis.)
2. Choose to analyse the alternative dataset that has been provided in the application's directory. (This dataset contains the first 100 items of the original MCFC Analytics Full Dataset.)
3. Choose to run an outlier detection filter.
4. Identify outliers based on the following features:
   - Duels Won
   - Passes Forward
   - Time Played
   - Touches
5. Use the default Outlier Factor parameter. (You can use the tooltip that is displayed on mouse-hover over the Outlier Factor component to learn how changing this parameters would affect the results of the outlier detection filter.)
6. Analyse the results of the outlier detection filter.
7. Save the results of the outlier detection filter to a text file on your computer, and verify if the file saved correctly.
8. Visualise the results of the outlier detection filter.
9. Change the attribute that is currently displayed on the Cartesian Z-axis to an attribute that is not currently measured on any of the Cartesian axes.
10. Go back to the start screen and delete the alternative dataset from the database.

C.3 Questionnaire

1. Do you have any knowledge of (or experience in) Machine Learning?
   - Yes
   - No
2. How interested are you in football?
   - Very interested
   - Somewhat interested
   - Not very interested
   - Not at all interested
3. How good is the design of the user interface of the application?
   - Very good

- Good
- Fair
- Poor
- Very poor

4. How useful do you think the provided algorithms are for analysing the MCFC Analytics Full Dataset?
    - Very useful
    - Somewhat useful
    - Not very useful
    - Not at all useful
5. How is the quality of the provided visualisations?
    - Very good
    - Good
    - Fair
    - Poor
    - Very poor
6. Did you learn what the provided algorithms do?
    - Yes
    - Somewhat
    - No
7. What other feedback do you wish to provide?

Bibliography

[1] T. Abeel, Y. Van de Peer, and Y. Saeys. Java-ml: A machine learning library. *The Journal of Machine Learning Research*, 10:931–934, 2009. Software available at `http://java-ml.sourceforge.net/`.

[2] F. Brooks Jr. No silver bullet – essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] J. W. Eaton. GNU Octave. `https://www.gnu.org/software/octave/` (accessed 20 July 2014).

[5] Edgewall Software. Comparison of Open-Source Java plotting libraries. `http://trac.erichseifert.de/gral/wiki/Comparison` (accessed 29 July 2014).

[6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. Software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009. Software available at `http://www.cs.waikato.ac.nz/ml/weka/`.

[8] jXLS Team. jxls. `http://jxls.sourceforge.net/` (accessed 26 July 2014).

[9] A. Khan. Java Excel API - A Java API to read, write, and modify Excel spreadsheets. `http://jexcelapi.sourceforge.net/` (accessed 26 July 2014).

[10] MathWorks United Kingdom. MATLAB – The Language of Technical Computing. `http://www.mathworks.co.uk/products/matlab/` (accessed 20 July 2014).

[11] MCFCAnalytics. (mcfcanalytics@mcfc.co.uk). MCFC Analytics - Full Data Set, Operational Definitions, Terms & Conditions and Formation/Position Key. Email to: Undisclosed recipients. 21 Sep 2012.

[12] Microsoft. Microsoft Azure Machine Learning. `https://azure.microsoft.com/en-us/services/machine-learning/` (accessed 22 July 2014).

[13] A. Ng. Anomaly Detection Algorithm. `https://d396qusza40orc.cloudfront.net/ml/docs/slides/Lecture15.pdf`, 2013. Demo Code (accessed 5 September 2014).

[14] Object Refinery Limited. Orson Charts. `http://www.object-refinery.com/orsoncharts/` (accessed 29 July 2014).

[15] Oracle Corporation. Java DB. `http://www.oracle.com/technetwork/java/javadb/overview/index.html` (accessed 23 July 2014).

[16] Oracle Corporation. Oracle Database Software. `http://www.oracle.com/us/products/database/overview/index.html` (accessed 23 July 2014).

[17] Oracle Corporation. Why MySQL? `http://www.mysql.com/why-mysql/` (accessed 23 July 2014).

[18] Oresoft LWC. How To Read Excel File Using Apache POI. `https://www.youtube.com/watch?v=GYZzkid7nno`, 2012. Tutorial Video (accessed 16 August 2014).

[19] M. Pernollet. Jzy3d - Scientific 3d plotting. `http://jzy3d.org/` (accessed 29 July 2014).

[20] J. A. L. Snyder. What Actually Wins Soccer Matches: Prediction of the 2011-2012 Premier League for Fun and Profit. A.B. Thesis, Princeton University, 2013. `http://homes.cs.washington.edu/~jasnyder/papers/thesis.pdf` (accessed 20 July 2014).

[21] M. Taylor. How Fouls Turn Into Cards. `http://thepowerofgoals.blogspot.co.uk/2012/08/how-fouls-turn-into-cards.html?m=1`, 2012. Blog (accessed 20 July 2014).

[22] M. Taylor. How Teams Win from the MCFC Data. `http://thepowerofgoals.blogspot.co.uk/2012/09/how-teams-win-from-mcfc-data.html?m=1`, 2012. Blog (accessed 20 July 2014).

[23] TeamDev. JExcel – Use Microsoft Excel in your Java app. `http://www.teamdev.com/jexcel` (accessed 26 July 2014).

[24] The Apache Software Foundation. Apache POI - the Java API for Microsoft Documents. `http://poi.apache.org/` (accessed 26 July 2014).

[25] D. Weir. A Machine Learning Teaching Aid. Msc dissertation, The University of Glasgow, 2010.

[26] Weka. Generating ROC curve. `http://weka.wikispaces.com/Generating+ROC+curve`, 2014. Demo Code (accessed 23 August 2014).