A Machine Learning Teaching Aid
MSc Development Project

Daryl Weir
0508104

Supervisor: Dr Simon Rogers

A dissertation submitted in part fulfilment
of the requirement of the Degree of
Master of Science at The University of Glasgow

September 2010

Abstract

The aim of this project was to develop an application to assist in teaching some of the basic concepts of machine learning. This is a subject area at the intersection of computer science and applied statistics. It offers powerful techniques for inferring information about huge collections of data, and algorithms from the field are constantly being applied to new problems in a diverse range of subject areas.

Many of the algorithms involved are quite complicated from a mathematical standpoint, but lend themselves well to visualization. To this end, the application produced by the project implements simple two dimensional cases of a selection of these algorithms and allows users to explore them visually. In addition, it offers sophisticated features for generating data for use with these algorithms.

The user evaluation carried out at the conclusion of the project showed a favourable reaction to the teaching aid software. Users with machine learning experience indicated that it represents a powerful didactic tool. It is designed to be modular and easily extensible, so that further algorithms can be added in the future, further increasing the usefulness of the program to students of machine learning.

Acknowledgements

I'd like to take this opportunity to thank a number of people, without whom this project would not have been possible.
First and foremost, thanks to my project supervisor, Simon Rogers, for his invaluable assistance and insight throughout the past year.
Secondly, thanks to Clare and Amanda for keeping me sane over the course of the MSc IT. It's been quite a year.
Thanks also to everyone who helped test my application and provided valuable feedback.
Last but not least, thanks to my family for supporting me through my studies.

Daryl Weir
September 2010

Contents

Chapter 1

Introduction

The field of machine learning is a rapidly growing and increasingly important area of computing science. As sensing technologies have improved and the processing power of computers has continued to grow, a massive amount of information about a vast range of phenomena has been generated. Thanks to commensurate increases in storage capacity, retaining this data is no significant challenge. However, analysing such a large, unwieldy collection is extremely difficult and no traditional model exists for the task. The primary goal of machine learning is to develop algorithms which allow information to be inferred about the underlying behaviour of a system which has generated a set of data. As a result, the term *inference* is often used interchangably with machine learning within computing science.

Machine learning has its origins as a subfield of artificial intelligence (AI), where it arose to fulfil the need for systems which could build functional representations of phenomena based on observed data. Unlike pure AI research, machine learning methodologies are not concerned with understanding the nature of intelligence - their focus is on efficiently and accurately inferring relationships, and producing concrete results. For this reason, machine learning has grown significantly outwith the confines of artificial intelligence, and has been successfully used to model problems in such diverse fields as computer vision, information retrieval, biology and stock market analysis.

The subject is a fairly mathematical one, drawing many of its techniques from applied statistics. Hence, it can be an intimidating subject to learn for computer scientists who do not have a background in mathematics or statistics. Machine learning models are often expressed in terms of vectors and matrices representing the parameters, and it is not necessarily intuitive what changing these parameters will do to the solution. However, many machine learning concepts lend themselves very easily to visualization. For example, a common problem in the field is regression. This seeks to model

the relationship between a dependent variable and one or more independent variables. In the case where there is only one independent variable, the model is simply a planar curve. The effect of altering the parameters can readily be seen by observing how this curve changes.

The School of Computing Science at the University of Glasgow offers an elective in machine learning. In the current format of this course, lab sessions involve the use of MATLAB [16], a powerful numerical computing environment. While it is possible to implement a wide variety of machine learning algorithms using the MATLAB scripting language, it is often not easy or intuitive to do so. Additionally, most students taking the course have little to no experience of the software and so they must spend time learning the basics of the language and following step by step examples rather than focusing on improving their machine learning knowledge. This is not an ideal teaching situation, particularly given that lab slots are limited.

The aim of this project was to produce a teaching aid application which took advantage of the ease of visualization of a variety of machine learning algorithms to allow users to experiment directly with the effects of changing data and, where appropriate, parameters on the output of these algorithms. The software is intended to be used in lab sessions to reinforce students' understanding of the concepts they have already seen in lectures. The teaching aid allows the user to plot data in two dimensional space using a variety of techniques, and then train an algorithm on that data and observe the output.

As well as its usefulness in a lab setting, the teaching aid can also be used in lectures to interactively demonstrate new algorithms as they are encountered in the notes. This is more useful than a series of static slides for demonstrating purposes, and can still be done fairly quickly. It is currently time consuming and impractical to describe and run MATLAB scripts in the lectures, whereas using the teaching aid the lecturer can quickly and easily illustrate the effect of successive iterations or parameter changes in a variety of algorithms.

The remainder of this document will discuss the development of the teaching aid in detail.

Chapter 2

Problem Context

The teaching aid addresses the problem that there are no software products which offer a simple and intuitive introduction to the core concepts of machine learning. Existing inference software tends to focus on algorithm implementations which are robust enough to analyze complex real world datasets. This is by no means a bad thing, but such datasets can be difficult to visualize. For example, in classification over textual documents, the number of variables can range in the thousands. However, the same basic principles are at work when considering the algorithm applied to data in two variables, producing results which can readily be visualized. The teaching aid thus focuses on simple cases of the algorithms it implements, providing intuition about the mechanics of each algorithm. Students can take the concepts taught in this way and generalize them to more complex problems over data from real problems. The remainder of this chapter discusses the algorithms which are implemented in the application.

2.1 Implemented Algorithms

There are a huge number of machine learning algorithms in the literature, and more are being created all the time. It is impossible to demonstrate a comprehensive range of these in a teaching aid application, particularly given the timescale for the project. However, this enormous collection can readily be grouped into a small set of categories. The two most important categories are the algorithms for supervised learning, and those for unsupervised learning. For more information on these algorithms, see (for example) [22].

2.1.1 Supervised Learning

Supervised learning techniques are those which deduce a function given a set of example data. These examples, commonly called the training data, consist of input-output pairs. Given these, the supervised learner must find a function which can make sensible output predictions both on the training inputs, and more importantly on unseen test inputs. The supervised learning algorithms in the teaching aid are further subdivided into two categories — regression and classification.

Regression

Regression algorithms seek to fit a continuous function to the training data. Generally speaking, this function can be an output based on D independent variables, corresponding to a curve in $N+1$-dimensional space. The teaching aid considers the simplest case, $D = 1$. Given a set of (x, t) pairs, the goal of the regression algorithm is to model t as a function of x. The particular class of models implemented in the application is the family of linear models, where we seek a function of the form

$$t = \sum_{k=0}^{K} w_k h_k(x).$$

Here, the functions $h_k(x)$ are arbitrary functions of x and the w_k are constant parameters obtained by the algorithm. These models are linear in the sense of the parameters – in general the functions h can be nonlinear. In the teaching aid, the choices for the h include polynomial terms up to 8th order, $\sin x$, and e^x. Users can choose any combination of these terms, and the teaching aid computes the optimal parameters w_k based on the data provided, and draws the corresponding curve.

The teaching aid implements two regression algorithms. The most basic of these is the minimized loss algorithm. As its name suggests, this determines the optimal parameters by minimizing the loss — the average squared difference between the output function and the training points. This problem has an analytic solution. Given N points with x values $x_1, x_2, \ldots, x_N$ and t values $t_1, t_2, \ldots, t_N$, define the following matrix:

$$\mathbf{X} = \begin{bmatrix} h_0(x_1) & h_1(x_1) & \ldots & h_K(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ h_0(x_N) & h_1(x_N) & \ldots & h_K(x_N) \end{bmatrix}.$$

Let $\mathbf{t} = [t_1, t_2, \ldots, t_N]^{\mathrm{T}}$ and $\mathbf{w} = [w_0, w_1, \ldots, w_K]^{\mathrm{T}}$ be vectors of the training labels and parameters respectively. Then the parameters $\widehat{\mathbf{w}}$ which minimize

the loss are given by

$$\widehat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{t}.$$

The teaching aid includes a library for manipulating matrices, and so can compute this solution quickly then draw the corresponding curve, updating each time new data is added.

The second regression algorithm in the teaching aid is the maximum likelihood method. This works by defining a probability density over the output t. Evaluating this density at a particular training point t_N produces a value known as the likelihood of that point. The algorithm maximizes the joint likelihood of the data — that is, the product of the likelihoods of each of the training points. Compared to the minimized loss algorithm, this has the advantage of taking the noise in the data into account. This provides an estimate of the uncertainty both in the optimal parameters and in the predictions the model makes. The implementation in the teaching aid assumes Gaussian noise in the data, which leads to exactly the same expression for the optimal parameters as the minimized loss algorithm. However, the maximum likelihood method also produces a value for the variance, which can be used to draw errorbars showing the confidence of the algorithm in its predictions. Thus, while the two algorithms produce the same curve when trained on a given dataset, the latter provides a greater amount of useful information.

Classification

The other main supervised learning technique is classification. Rather than fitting a continuous function to some data, the goal here is to correctly label points as belonging to one of a number of classes. The training data consists of input objects with several attributes, and corresponding labels for those inputs. The classifier must find a rule for labelling new points based on their attributes. As a real world example, the attributes might be physiological measurements such as blood pressure and white blood cell count, and the labels reflect whether or not a patient has a given disease. After training on the data for a number of patients, the classifier would attempt to determine whether a new patient has the disease. In such applications, the number of attributes is likely to be high. For ease of visualisation, the application is restricted to only two attributes, represented as the x_1- and x_2-coordinates on the plot. The label of a point is indicated by the colour it is drawn in. The teaching aid shows the output of a classifier by colouring the plotting space. The plot is divided into a grid, and each square is coloured according to the output of the classifier on its center. This allows the boundaries between classes to be clearly seen.

Classification algorithms can be further subdivided into probabilistic and non-probabilistic techniques. The former give a measure of how sure the algorithm is about each classification, whereas the latter only makes discrete assignments of points to classes. The teaching aid implements two classifiers, both of which are non-probabilistic. There was insufficient development time to add a probabilistic algorithm. For each of the implemented classifiers, the application allows users to vary the associated parameters and observe the effect on the classification.

The simplest classification algorithm implemented is K Nearest Neighbours (KNN). Unlike most algorithms, this lacks a training phase to produce a classification rule — only a set of data and an integer K need be provided. To label a test point, the algorithm simply looks at the K nearest points to it in the training set, and assigns the test point the majority label from those K. In the case that there is no clear majority label among the K nearest neighbours, the algorithm breaks ties randomly.

The second classifier implemented in the application is the Support Vector Machine (SVM). This is a binary algorithm, which means it distinguishes between only two labels on the training data. It is substantially more complex than KNN, but in general is more powerful. The algorithm operates by finding the hyperplane which best separates the two classes. This hyperplane is 'best' in the sense that it maximizes the margin — the perpendicular distance between the hyperplane and the nearest points on each side. As usual, the teaching aid implements the simple two dimensional case, so that the separating hyperplane is a straight line through the data, with equation

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0.$$

If the two possible labels for the training data are $t_n = \pm 1$, it can be shown that the problem of maximizing the margin is equivalent to minimizing

$$\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w},$$

subject to constraints

$$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1.$$

By introducing non-negative Lagrange multipliers α_n for each training point, this expression can again be reformulated as a maximization over α of

$$\sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m,$$

subject to the constraints

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

and

$$0 \leq \alpha_n \leq C$$

where C is a parameter which controls the sensitivity of the SVM to outliers. This is an example of a standard class of optimization problem, the quadratic program. There are a variety of well-known algorithms to solve quadratic programming (QP) problems, and each problem has a single, global solution. This is useful because the optimal decision boundary is guaranteed to be found in all cases.

However, algorithms to solve QP problems are often inefficient and do not scale well to large datasets. Hence, in real world problems they are intractable for use in SVM problems. Platt [20] introduced a new algorithm for training SVMs, called Sequential Minimal Optimization (SMO). The memory requirement SMO is linear in the training set size, rather than cubic for a naive QP algorithm, and the calculations it performs have analytic solutions so that it is computationally faster than a QP problem. This makes it extremely desirable for any serious SVM implementation. The teaching aid uses an algorithm by Keerthi et al [15] which extends the SMO algorithm to further increase the speed of convergence. This allows the algorithm output to be recomputed and redrawn quite quickly as new data is added.

Kernel Methods

The SVM is an example of an algorithm which can be made much more powerful through the use of *kernel methods*. The algorithm, though fast, can only find linear decision boundaries, and many datasets are not linearly separable. However, in the optimization problem, the data only appear as inner products $\mathbf{x}^{\mathrm{T}}\mathbf{y}$. These inner products can be replaced by kernel functions $k(\mathbf{x}, \mathbf{y})$, which may be thought of as inner products in a higher dimensional space — that is, $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}^{\mathrm{T}})\phi(\mathbf{y})$ for some transformation ϕ. This allows the algorithm to operate in a projected space, where the data are linearly separable, but does not require the projections to be done explicitly. Indeed, the function ϕ does not even need to be known, only the kernel function k. The SVM is still only finding a linear boundary in the projected space, but this can correspond to a very complex decision boundary in the coordinates of the data. Thus, the same simple algorithm can classify complex data with a high degree of accuracy. This is a powerful advantage which costs little

extra from a computational standpoint, and has placed SVMs at the cutting edge of modern machine learning problems.

Many algorithms can be kernelised — any algorithm in which the data appear only as inner products. There are many well known kernel functions which can readily be used, allowing simple algorithms to solve complex problems. The drawback to using kernels is that they introduce additional parameters to tune, which can significantly increase the time to find a good model for the data. The teaching aid implements linear, Gaussian and polynomial kernel functions.

2.1.2 Unsupervised Learning

In supervised learning problems, the training data consists of both input attributes and some expected output. By contrast, training data for unsupervised learning contains only the attributes. Unsupervised learners are tasked with discovering how the data are organized, and identifying important patterns in the examples.

Clustering

The principal unsupervised learning technique of interest for the teaching aid is clustering. This involves grouping together data which are similar in some way. For example, retailers have increasingly large collections of information about which customers buy which products. Using clustering they can identify customers who have similar shopping habits, or groups of products often bought together. This can form the basis of a recommendation system. As with other examples which have been discussed, in real usage clustering deals with many attributes for each input object. The application is again restricted to two attributes for easy visualisation. The assigned cluster of a point is shown by colouring the point. Users are able to generate data, choose a number of clusters, and step through the iterations to see how the clusters evolve.

The teaching aid implements the K Means clustering algorithm, a simple but effective iterative scheme. A value of K is specified, and the algorithm assumes that there are K clusters, each represented by a mean point. These means are initialised randomly, and then all training points are assigned to their closest mean. The algorithm alternates between updating the means to the average of all points assigned to them, and reassigning points to the new means. This process continues until the assignments do not change — convergence is guaranteed, and is usually reached in very few steps. The

algorithm converges to a minimum of

$$\sum_n \sum_k z_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}(\mathbf{x}_n - \boldsymbol{\mu}_k),$$

where z_{nk} is 1 if point n is assigned to cluster k and 0 otherwise, and $\boldsymbol{\mu}_k$ is the k-th mean. However, this function has multiple local minima, and so convergence to the global minimum is not guaranteed. The algorithm is sensitive to the initial random placement of the means. Since the algorithm is fast, it is usually possible to overcome this problem by running it several times with different initializations and choosing the clustering with the smallest value of the minimized quantity.

The last algorithm implemented in the teaching aid is Kernel K Means, an extension to the standard K Means algorithm which replaces the Euclidean distance measure by a kernel function. This allows the algorithm to cluster datasets whose structure cannot be captured in Euclidean space. However, the introduction of kernels further increases the sensitivity of the algorithm to initial conditions. Thus, the algorithm often needs to be run many times to find the best clustering. Additionally, because the kernel algorithm is operating in a transformed space for which we do not know the transformation, the positions of the means are not known in the data's frame. As a result, while the standard K Means algorithm in the teaching aid is able to draw the means explicitly as large points, the kernel version can only colour the points to indicate cluster location.

2.1.3 Cross Validation

It is desirable to be able to compare different machine learning models against one another. With regression and classification models, the primary goal is to generalize the analysis of the training data to make accurate predictions on independent test data. It does not follow that a model which describes the training data closely will necessarily make good predictions on test points. For example, in minimized loss regression, increasing the model complexity by adding additional terms will always decrease the training loss, but the more complex model might well be overfitted to the data and make poor predictions on further data. To this end, a method for comparing the predictive performance of different models is needed — the teaching aid uses *10-fold cross validation*.

Cross validation is the practice of partitioning the training data into two sets, training the model on one and testing its performance on the other. This process is repeated several times with different partitions, and the results averaged to reduce the variability. In 10-fold cross validation, the data are

split into 10 folds of roughly equal size. The algorithm is trained on 9 of these folds, and the predictive performance on the held out fold is recorded. This is repeated 10 times, holding out a different fold each time, and the predictive performance is averaged across the 10 rounds. The performance measure used is algorithm dependent — the testing loss for the minimized loss method, test set likelihood for the maximum likelihood, and the number of misclassifications on the test set for the classifiers. The teaching aid carries this process out and displays the averaged performance when the appropriate button is pressed.

Cross validation for clustering algorithms is not included in the application, since performance measures for clustering are application- rather than algorithm-specific.

Chapter 3

Survey of Related Software

To the best of the author's knowledge, there is no existing software which acts as an effective machine learning teaching aid across a variety of concepts. However, as part of the preparation for the development of the teaching aid, a variety of software products which accomplish some similar functionality were studied. Additionally, several software libraries were considered for inclusion in the application in order to ease the implementation of certain features. An analysis of the strengths and weaknesses of these products follows, together with a description of any influence they had on the final teaching aid.

3.1 Machine Learning Applets

There are a number of interesting Java applets freely available online which illustrate machine learning concepts. [8] provides implementations of a variety of algorithms.

3.1.1 Regression

Firstly, the author provides a simple least squares applet for linear regression. This allows the user to plot points by clicking on the applet, and it draws the best fitting straight line. The line updates each time a new point is added. This ability to directly add new points is a highly useful one shared by the other applets in the series, and this feature was incorporated into the teaching aid.

A second applet extends the functionality of the first by additionally plotting polynomials up to fifth order for the data the user adds. This is implemented rather poorly, however. It plots all five polynomials simultaneously, and thus it is very difficult to distinguish between them. This is

worsened by the fact that all the curves are drawn in the same colour. The teaching aid improves on this in its regression algorithms — the user is able to add an arbitrary number of models, the terms for which are controlled independently. For each model added, the colour can be chosen from a selection of four. This allows more meaningful information to be extracted from the plot.

3.1.2 Classification

Also provided are a number of applets for classification, including one implementing the K-nearest neighbours algorithm. This allows the user to plot two types of point (crosses and circles), then shades the background to show the boundary between the two classes. The evolution of this decision boundary is intuitively clear as new points are added. However, the applet only implements the algorithm for the case $K = 1$, which is a significant drawback.

The shading feature as a means of communicating the output of the classifier is a very useful one, and was adopted as part of the teaching aid. However, the KNN implementation in the final application is significantly more robust than the one in this applet. It allows choice between five classes rather than two, and the value of K can be set to any natural number. This allows the benefits and drawbacks of the algorithm to be more fully explored.

3.1.3 Clustering

Lastly, the applet suite provides an implementation of the K-means clustering algorithm. Users are able to plot points, and the applet assigns them to the nearest cluster and colours the background to show the current shape of the clusters. It is a reasonably clear implementation, but suffers from the limitation that the number of clusters is fixed at four. Also, since the display is updated after each new point is added, the clustering is initially very unstable. This is somewhat visually confusing to an inexperienced user, and is not explained by the notes accompanying the applet.

Though a shading approach was found to be desirable for classification, for clustering it is less useful. In the teaching aid, the clusters are indicated by recolouring the points to the same colour as their means. Additionally, any number of clusters can be specified, not just four. This allows different clusterings to be compared against one another.

3.1.4 Discussion

A weakness of these applets in general is that there is no documentation beyond a brief set of instructions for each. They assume some knowledge of the algorithms they implement, which is a particular problem for the K-means applet. A novice to the subject would most likely be unable to take something meaningful from the symbols and colouring it uses. More documentation would make a significant improvement to its usefulness. This illustrated a need for a help system in the teaching aid, which is included in the final product. A collection of help files documenting each algorithm and feature of the application can be accessed at any time while using it.

The applets do have a number of positive features. First, the source code is freely available for inspection, and is fairly well documented and designed. This was instructive in the use of Java to implement machine learning algorithms. The code was studied during the proposal phase of the product, including the matrix implementation by the author. Though this was a useful introduction, ultimately the code for the teaching aid took few ideas from the applets. For one, the matrix implementation was not complete enough to easily perform all the required operations, such as Cholesky decompositions. Additionally, the drawbacks of the algorithm implementations were deemed too significant to attempt adapting the implementations for use in the final product.

The implementation as applets is also useful in that the software can be distributed online very easily, and no installation or configuration is required by users. The possibility of adapting the teaching aid into an applet was explored during the development, but was ultimately discarded due to time constraints. An adaptation of the software into an applet, or to run using Java's Web Start system, represents useful future work.

3.1.5 Gaussian Processes

Another machine learning applet is detailed in [9], which covers Gaussian processes. This is a more polished product than the other applets which have been discussed. It allows the user to plot some points, choose a type of covariance function, and choose the parameters of that function. The applet then draws the mean curve together with an indication of the variance. The software is supported by useful documentation and plentiful links to other information on the subject. However, the applet itself tends to be slow in responding to user input. Since the source is not available, the reason for this is not clear.

Gaussian Processes were not included in the teaching aid as there was

insufficient time to implement the algorithm. As quite a complex technique, it was deemed too risky to implement, particularly since the algorithm is not covered in the current machine learning course. Nonetheless, adding functionality similar to that found in this applet would be a useful addition in the future.

3.2 Weka and Java-ML

There are a number of Java libraries which implement machine learning algorithms. Two of the most prominent are Weka [12] and Java-ML [1]. Both are open source projects which provide powerful and robust implementations of a wide variety of algorithms. The code is well designed and extensively documented, and thus can provide insight into writing efficient machine learning programs. However, both are designed to be used on real world data sets and as such provide implementations which are more complicated than strictly necessary for the scope of this project. Their use was considered for assistance with the project implementation, but ultimately it was decided that the libraries would add significant 'bloat' to the application and the cost of integrating them with the desired functionality was too high.

3.3 MATLAB and Octave

3.3.1 MATLAB

MATLAB [16] is a powerful numerical computing application, and also the name of the scripting language used by that application. It is the current standard for lab use in the machine learning course, as well as enjoying widespead popularity in the academic and industrial communities. It provides sophisticated functionality for manipulating matrices and plotting data, which may be used to implement and visualize a wide variety of machine learning algorithms. However, despite this power, it is far from an ideal environment for the purpose of teaching inference ideas. While it is possible to generate data using MATLAB, it is far from intuitive for a novice user to do so, and similarly algorithm implementation can be tricky. There is a significant learning overhead associated with mastering the MATLAB scripting language which is often a hindrance in the course lab sessions. Indeed, in order to make progress on some of the more difficult algorithms, students are provided with implementations and left to observe the effect of altering the parameters. Even this is not simple – parameters must be changed in text files and the script must be re-executed after each change to see the effect.

This disconnect renders the system unintuitive for illustrating the algorithm concepts. The teaching aid application provides a continuous interaction style which is far more intuitive to use.

3.3.2 Octave

Another main drawback of MATLAB as a teaching tool is its cost. The software is proprietary, and individual licences for the software are very expensive, and as such it is unlikely a student will be able to afford one. Thus, students cannot install the software on personal machines and are limited to using MATLAB within the departmental labs. This is somewhat impractical, particularly during revision periods when students are not normally on campus. However, this problem is partially solved by the existence of GNU Octave [19], a free and open source numerical computing environment which is largely interoperable with MATLAB. Distributions of the software are available for Windows, Mac and most prominent Unix-like operating systems. Hence students can readily obtain a copy to experiment with at home. It shares many benefits with MATLAB, but also suffers from the same drawbacks for use as a teaching aid. Additionally, it uses a command line interface which is less intuitive than the MATLAB GUI. Octave's plotting capabilities are also slightly less powerful, utilising the open source gnuplot program.

3.3.3 JavaOctave

JavaOctave [13] is a software module which allows Octave calculations to be performed from inside a Java application. Its use as part of the teaching aid application was initially considered, since it would potentially simplify algorithm implementation by allowing reuse of scripts that were written as part of the machine learning course. However, this would restrict the application to use on systems where Octave was installed and it was decided that this was a poor design choice. JavaOctave was also considered for use as part of the development process, using calls to Octave to check the correctness of the Java algorithm implementations. However, some initial experiments with the software were not promising. Its syntax is quite unwieldy, and it proved difficult to perform even simple calculations without running into frequent error uninformative error messages. Ultimately it was decided that the cost of learning the syntax and integrating the testing with the teaching aid would be too high, and would produce only limited benefit. The use of the software was thus deemed unviable.

3.4 Mathematics and Statistics Libraries

Much of the code for the teaching aid application involves determination of parameters by performing operations on vectors and matrices, and the Java class libraries do not have support for these. In order to avoid writing code to perform basic matrix operations, which would take some time to do comprehensively, a number of software libraries implementing this functionality were considered.

3.4.1 Apache Commons Math

Commons Math [2] is a library produced by the Apache Software Foundation which provides a number of mathematical and statistical features not available in the Java platform. In particular, it provides linear algebra packages which enable matrix operations and the solution of systems of equations. The library also has features for random data generation in a variety of formats and for statistical functions including least squares regression. These features were certainly desirable, but the library also includes a large amount of functionality which is outwith the scope of the project. For example, it has classes for solving ordinary differential equations and implementing complex numbers. In total there are over 30 packages in the library, with the source code occupying around 6MB.

An additional point is that the code in the library is designed to be very general, and using it as part of the teaching aid would have required data to be stored in generic structures which are more complex than necessary for the purposes of the application. This would have added a further cost to integrating the library, detracting from useful development time. As with JavaML and Weka, the library was ultimately decided to be too bloated and complex for inclusion in the teaching aid code.

3.4.2 JAMA

JAMA [17] is a linear algebra package for Java, developed by the Mathworks and NIST. It provides operations for constructing and manipulating real matrices, together with functionality for solution of systems of equations and several common matrix decompositions. The implementation is compact, using only six classes in a single package, and the primary data structure is a single Matrix class. It provided sufficient operations to implement all of the algorithms in the teaching aid, and given its size was easy to integrate with the rest of the application. Further, the classes are well designed, readable and suitably commented, so that the implementation is easy to understand.

This made use of the library very simple. Hence, JAMA was used in the application to accomplish the required mathematical functionality.

3.5 Quadratic Programming

As discussed earlier in this document, one of the more complex algorithms in the teaching aid is the Support Vector Machine. The complexity arises over the solution of a quadratic programming (QP) problem as part of the algorithm. Ultimately, this problem was overcome by implementing Platt's SMO algorithm as an alternative to a direct QP solution. However, the author did not discover SMO until later on in the development process. Prior to this, a number of options were considered to act as a QP solver for the teaching aid.

3.5.1 LibSVM

LibSVM [4] is a software library for support vector classification in general. It provides implementations of the basic SVM classifier, as well as extensions for regression and distribution estimation. It also supports a multi-class extension of the SVM technique. Implementations are available in a variety of languages, including Java. However, there were a number of reasons to avoid using the library in the teaching aid. Firstly, the extensions it incorporates are well beyond the scope of the project. Secondly, the Java implementation provided is poorly laid out and difficult to read. Much of the code is organized in a single file which contains multiple inner classes and few comments. The algorithms used are not described well (if at all), and the time necessary to gain an understanding of the code and integrate it with the teaching aid would have been infeasibly long.

3.5.2 QuadProjJ

QuadProgJ [23] is a Java solver for strictly convex quadratic programming problems. It implements the dual active-set algorithm developed by Goldfarb and Idnani [11]. The code is clearly commented and easy to understand, and is contained in a single class. It was a potential candidate for use during the project, but was ultimately rejected for a number of reasons. For one, it uses a matrix implementation from Colt [3], a mathematics library developed at CERN which is similar in many ways to Commons Math and so was too complex for the needs of the application. If QuadProjJ were to be used, its implentation would have needed to be adapted to use JAMA

matrices. This would likely have been possible within the time allowed for the project. However, a second concern is that the code takes in the QP problem framed in a slightly different format to that required for the SVM, and so a certain amount of preprocessing would have been necessary within the teaching aid application. The combination of these factors led to the decision that integrating QuadProjJ with the teaching aid was too expensive in terms of development time.

3.5.3 New Implementation

The other option which saw serious consideration was to write an entirely new QP solver. The obvious advantage was that this provided freedom to choose the format to match the needs of the SVM algorithm. Chapter 12 of [14] provides an extensive guide to implementing a quadratic programming solver in Java, and would have been used as a basis for the code in the teaching aid. However, the SMO algorithm was encountered, and is considerably easier to implement than a QP solver, rendering this decision moot.

3.6 Plotting

Obviously, a significant part of the teaching aid application is its ability to plot data points and curves. A number of Java libraries were considered to assist this aspect of the functionality. These included JFreeChart [18] and the Scientific Graphics Toolkit (SGT) [7]. These, and other libraries like them, provide support for creating graphics in a variety of sophisticated ways. For example, many libraries allow time series plots, bar charts and histograms to be generated. Most of this functionality is redundant for the needs of the project – the teaching aid is primarily concerned with simple XY plots and with colouring the display based on a grid. The library which comes closest to matching these requirements is the SGT, but even that provides more than is needed.

It was therefore decided that the required plotting tools could most easily be implemented using the core classes provided in the Java Swing library. The plotting is all handled by a custom class which extends Java's JPanel.

Chapter 4

Requirements

4.1 Requirements Capture Process

From the outset, the project had a well defined aim: design and implement an application to teach some fundamental machine learning concepts. The earliest part of the development process was to refine this overarching aim into a basic set of requirements from which to build the application. The primary stakeholder in the project is Dr Simon Rogers, as both the project supervisor and a likely future user of the software. The largest part of the requirements gathering process took the form of a series of meetings with him, discussing the features that were desirable. This led to an initial list of requirements, prioritised according to the MoSCoW method. [5]

Additionally, the survey of related software highlighted a number of useful features which were not initially considered by the client. This was another source of refinements to the requirements document. In particular, the ability to add single points by clicking was not an initial requirement, but was added to the list after it was encountered in another piece of software.

The final part of the requirements gathering process was an email sent to the students undertaking the machine learning course in the 2009/2010 session. The students, as potential users of a teaching aid, were asked to identify features they would find useful in such an application. Only a small number of responses were obtained, which did not add any entirely new requirements. The value in these responses came in identifying which algorithms were most conceptually challenging, and hence should be prioritised in the implementation.

A brief overview of the requirements obtained now follows. The full list of requirements and a list of derived use cases are included in appendix A of this document. However, it should be noted that some of these are left

purposefully general. This decision was taken in order to allow flexibility of the precise details if any difficulties became apparent during the development process. Indeed, the formal requirements capture phase of the project was relatively short. This was in keeping with the chosen development model, an agile process which attempted to avoid excessive heavyweight design and focus instead on short, frequent iterations of design and implementation.

4.2 Requirements Overview

Starting from the basic aim of the project, two fundamental requirements were immediately obvious. First, it must be possible to generate data, and secondly it must be possible to run an algorithm on that data. These basic goals were then broken down into more detailed requirements. For example, the ability to add and remove single points directly, and to sample multiple points from a given distribution, were added as extensions of the data generation requirement.

A less critical, though still highly desirable, set of requirements arose around the storage of data. To enable interesting cases to be easily demonstrated, a requirement for the saving and loading of datasets was added. Additionally, the project supervisor requested that the facility to export images of the plot be added. This was added as "could have" under the MoSCoW system, as the teaching aid could easily be deemed functional without it.

A small number of non-functional requirements were also identified. Foremost among these was that the teaching aid should be developed in Java, to allow for easy deployment over a variety of platforms. Also, the GUI should be designed with users who have a basic grasp of machine learning terminology in mind. Nonetheless, it should be designed to be as usable as possible even for novice users. Finally, algorithm implementations should be made as efficient as possible in order to account for the possibility of large training sets.

Chapter 5

Analysis and Design

The next step in the development was to analyse the requirements and problem domain to identify a class structure for the teaching aid's core features. Individual algorithms were not designed in detail before the implementation began. Rather, a basic framework for the model was created, covering algorithms and data. Alongside this, a design for the key elements of the graphical user interface (GUI) was produced.

The overall design attempted to follow a Model View Controller architecture [21] — the Model is represented by the algorithm implementations, and the View by the plotting functionality present in the GUI. There are no pure Controller classes, however. Instead, each algorithm has an associated class which handles both the user interface aspects of setting options for the algorithm, and the logic of training the algorithm and addings its output to the display. Thus, these classes fall somewhere between the View and Controller roles.

A more detailed treatment of the design now follows.

5.1 The Domain Model

The domain model of the teaching aid is the part of the system which describes the machine learning entities and the relationships between them. From studying the requirements, it was immediately clear that two of the important objects to model were algorithms and data points. Each of these was chosen as a candidate class. Additionally, it was clear that a collection class for data points would be necessary — this became the Dataset class. These three classes formed the basis for the two core domain model packages, `algorithms` and `data`.

5.1.1 The `data` Package

Datapoints

The Datapoint class represents a single point. Each point is modelled by its location in two dimensional space together with a label. This is not entirely accurate to the underlying machine learning model for all algorithms. For example, in regression a point is defined by only its x coordinate and its label t. However, given that all plots produced by the teaching aid are in two dimensions it seemed a good idea to have the data kept in this standard structure across all algorithms. Though those two dimensions represent different quantities for different families of algorithm, this implementation decision is unseen from a user's perspective, and using a standard format makes algorithm implementation easier.

The initial Datapoint design included a number of other features. First, each point can have an associated error bar, represented by a boolean variable indicating whether or not to draw the bar and a double representing the size of the error. This was added to allow regression algorithms with a measure of predictive uncertainty to show this uncertainty when the output is drawn. Secondly, each point also holds a boolean indicating whether the point is to be highlighted when drawn. This was added with the intention of visually showing which points are support vectors for the SVM algorithm, but algorithms added in the future could also conceivably highlight points, hence why the feature was added to the Datapoint class rather than to the SVM algorithm itself.

Useful methods in the class include one to determine whether a point 'contains' an (x, y) pair or not. This allows the teaching aid to determine when a point has been clicked on. Also, a method was added to determine the Euclidean distance of the point from a second point passed as a parameter. This is useful for a variety of other features in the teaching aid, such as finding the neighbours in the KNN algorithm.

Datasets

As well as operations on single points, it is important for the teaching aid to be able to operate on collections of points. The Dataset class controls this functionality. It maintains a collection of Datapoint objects, implemented as a list since the order of points can be important — for example, when interpolating between points to draw a line. An integer field called `style` is also maintained, which describes how the dataset should be drawn. Options include discrete points, interpolation in order between the points, and shading. This field is accessed by the GUI when a dataset is plotted.

The majority of methods in the class are for adding, removing and accessing points. Additionally, there are two noteworthy methods which should be mentioned. The first is `kNearest`, which takes a Datapoint and an integer k as parameters and returns a Dataset object containing the k nearest points in the set. This is clearly useful for implementing the KNN algorithm, and keeps the internal representation of a Dataset hidden from client code. The second important method is `fold`. This takes an integer n as a parameter and returns an array of n Dataset objects, each containing approximately an n-th of the points in the dataset. The partitioning of points between folds is random. This method is intended to simplify the implementation of cross validation.

Sampling

One of the requirements is the ability to sample data from a given distribution. At this stage of the design, methods for doing this were not studied in great detail. However, in order to facilitate this at a later stage, an abstract class Sampler was added to the `data` package. This specifies an abstract method, `nextPoint`, which returns a Datapoint. Individual samplers can supply an implementation of this method, which can the be called by the GUI as many times as necessary to generate the required data.

5.1.2 The `algorithms` Package

The `Algorithm` Class

The primary class in this package is Algorithm. This is an abstract class which defines the core methods any algorithm must implement. It also defines two fields which all algorithms require — one is the training data, a Dataset object, and the other is a boolean variable indicating whether the output of the algorithm should be drawn or not.

The two most important methods in Algorithm are `train` and `predict`. The former takes in a Dataset object and trains the algorithm on the points in the set. The latter takes a Datapoint and returns the numerical output of the algorithm on that point. These methods represent the two fundamental machine learning operations that the teaching aid performs.

The Algorithm class also specifies an abstract method `getOutput` which is used to obtain an algorithm's output for plotting. This method takes a Dataset as a parameter, and returns another Dataset with the appropriate style and values for plotting. The type of Dataset passed in and out depends on the particular type of algorithm. For example, a regression algorithm is

only concerned about the x values of the points in the input set, and returns a set of (x, y) pairs which can be interpolated to draw a curve. A classifier, on the other hand, takes a grid of points as an input and returns the same grid, but with the points coloured according to the output of the classifier. This distinction led to the introduction of a further set of abstract classes into the design.

Abstract Subclasses

The logic of plotting an algorithm's output varies between the different families of algorithm — classifiers, clusterers and regression techniques — but does not vary inside those families. For example, KNN and SVM both produce a coloured grid as their output. Thus, a second level of abstraction was added to the design. Rather than have individual algorithms extend the Algorithm class directly, three intermediary classes were identified. These are the RegressionAlgorithm, Clusterer and Classifier classes. Each of these extends Algorithm with some new features specific to their family of methods, but all three classes are still abstract. The actual algorithms in the teaching aid all extend on of these classes.

There are a number of advantages to this design. First, it enables the use of the heterogeneous list pattern. When training an algorithm, the teaching aid need not know the specific technique it is training, and can instead call the `train` method from Algorithm. On the other hand, the additional layer of abstract classes allows enough flexibility to plot the different types of algorithm, but is not so vague that each algorithm requires its own plotting code. This simplifies the GUI implementation.

The RegressionAlgorithm class adds a number of features to the basic Algorithm class. It defines eleven integer constants representing the functions $h(x)$ which the teaching aid recognises as valid terms in a regression problem. The integers 0–8 represent polynomial terms from x^0 up to x^8, 9 represents $\sin x$ and 10 is e^x. Whether or not a given term is used by the algorithm is governed by a boolean array `terms` of length eleven. If the i-th entry of this array is true, then the function corresponding to the class constant i is included in the regression. For example, if `terms[0]`, `terms[3]` and `terms[9]` are true and the remaining entries are false, the algorithm will fit a model of the form $w_0 + w_1 x^3 + w_2 \sin x$. This design has the advantage that it is easily extensible through the addition of new class constants for new functions and an increase in the length of the array.

This class also contains a variant of the `getOutput` method which takes an array of doubles representing the x coordinates rather than a Dataset as its input. While this variant is not strictly necessary it is a useful convenience

method — computing the output values is faster since the input does not need to be wrapped in a Dataset.

Other fields introduced in this class include three Matrix objects named `X`, `t` and `parameters`. These correspond to the matrix $\mathbf{X}$ and the vectors $\mathbf{t}$ and $\mathbf{w}$ as defined in section 2.1.1. The notation for these quantities is standard across a wide variety of linear regression techniques, so it seems obvious to include them as instance variables.

The class Classifier adds no new fields to the basic Algorithm structure. It does, however, add two new methods. One is `classificationErrors`, which takes a Dataset and returns the number of misclassifications the algorithm makes on the data. A misclassification occurs when the algorithm's prediction for the label on an (x_1, x_2) pair does not match the actual label on the point. To make this simpler at the implementation stage, a second new method, `predictLabel`, was added. This abstract method forces a classifier to assign a label to a Datapoint. For some algorithms, such as KNN, this will return the same result as the basic `predict` method, but for others the two methods will return different numbers. For example, `predict` for an SVM will return some real value on a given point, and `predictLabel` will assign a label to the point based on the sign of that prediction. The decision to implement `predictLabel` as a separate method was taken to reduce repetition in the design. The abstraction of the label assignment means the misclassification count can be implemented at the Classifier level, rather than having separate methods in each class which extends Classifier.

The Clusterer class adds no new instance variables or methods. Its only purpose is to allow algorithms to be referenced as clusterers by the GUI, without needing to know the specific algorithm.

Specific Algorithms

Individual algorithms were not designed in detail at this phase of the development. It was decided that the framework described above was a sufficient basis from which to start the development process. Many algorithms can be implemented fully using only the abstract methods defined in the framework. Some, particularly the clusterers, were later found to require additional methods specific to each algorithm. However, the risk posed by designing and implementing those methods as they were encountered in the development process was deemed very low.

A final detail added to the design of the domain model was to add three subpackages to the `algorithms` package, one for each of the main algorithm families. This makes the structure of the domain clearer, and groups logically related classes together. A class diagram for the domain model as of the

completion of the project is shown in figure 5.1.

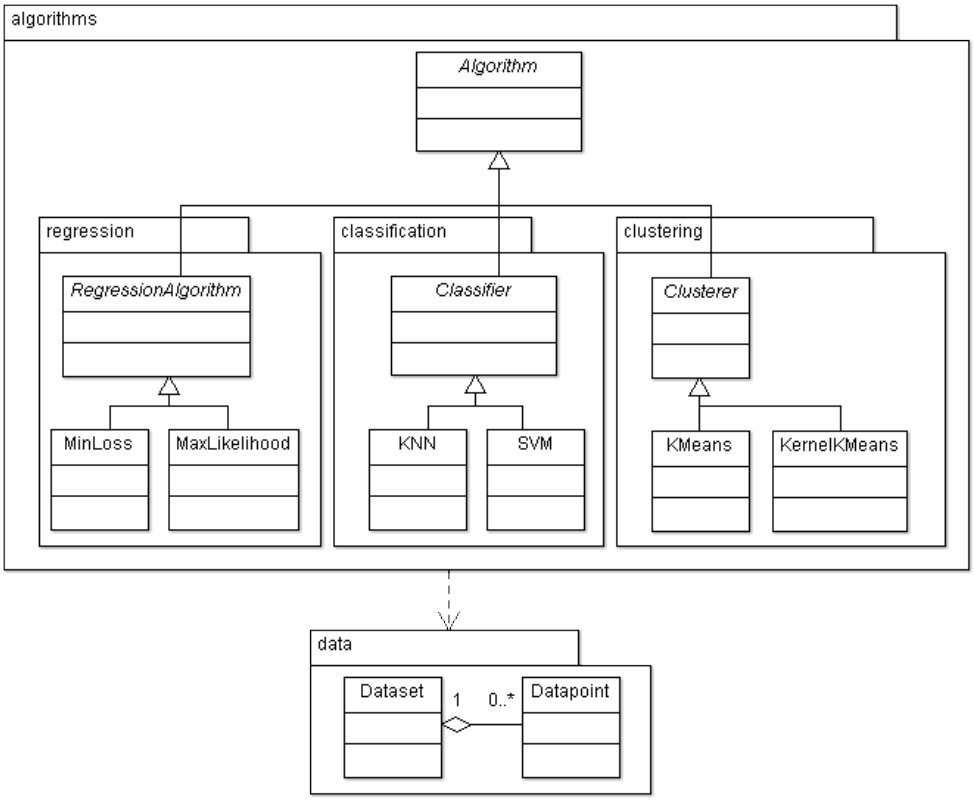


Figure 5.1: Domain Model Class Diagram

5.2 Graphical User Interface Design

As discussed in section 3.6, the proposal phase of the project identified that the optimal method to implement the teaching aid's GUI was to make use of Java's Swing framework. This toolkit, a core Java component contained in the `javax.swing` package, provides lightweight implementations of a variety of GUI components such as buttons, panels, tables and trees. It provides a number of flexible layout managers for controlling the appearance of an application, and combined with the `java.awt.event` package offers support for firing and handling a variety of events. It is also simple to implement a custom component by extending a Swing class and overriding its `paintComponent` method.

5.2.1 Basic Layout

The first step in designing the GUI was to identify the key information the interface had to communicate, and to determine a possible layout for this information. The important information identified was:

- The available algorithms, and the one currently selected
- Any options for configuring the selected algorithm
- Options for generating data
- Most importantly, the plot of the data and the algorithm output

The next step was to draw up a mock interface showing a possible layout for these four items. A simple drawing of the proposed layout is shown in figure 5.2. This is very close to the final layout of the program. A slightly earlier version of the design had data generation as an additional option on the list of algorithms, but on further reflection this was a poor idea. For one thing, the number of classes allowed varies by algorithm, so a single independent data generation feature would work poorly. Additionally, it is useful to be able to add new data after an initial model has been trained. Thus, the second version of the design includes an explicit area for data generation options.

Having identified this layout, a more detailed GUI design was produced, specifying a class structure and identifying which Swing components would be used to implement particular sections. The classes were not designed in precise detail at this stage — the buttons, drop down menus and so on for specific options were left to be added at implementation time. It was more important to capture the main interface classes and the relationships between them.

To prevent any one class from becoming too complex, the various parts of the functionality listed above were given their own class. The first of these is AlgorithmPanel, a class which handles algorithm selection. Swing provides a number of widgets which implement selection from a set of options. The two most appropriate components are JList, which simply displays a list of elements and allows mouse selection of the entries, and JTree, which displays a hierarchically ordered list. The latter was chosen because the tree allows the models to be clearly grouped together by type. The class AlgorithmPanel extends Swing's JPanel, a generic container class, and has a JTree field to show the algorithms.

Data generation is handled by the class DataPanel, which also extends JPanel. This class displays its options in two subpanels. The first controls

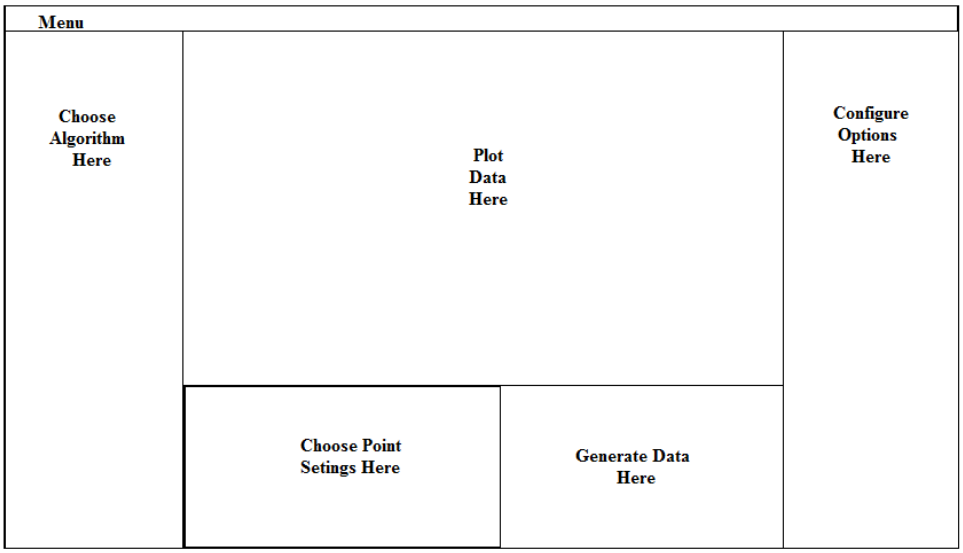


Figure 5.2: A basic interface mockup, showing the core features

the settings for the points to be added — a set of radio buttons for choosing between the training and test data sets, a button to remove all points from the display, and a drop down menu to choose the class. The last of these only appears for classifiers, and the number of classes available varies as appropriate for the selected algorithm.

The other subpanel in Datapanel provides the options for data generation. At the design stage, the full range of generation techniques was not well defined. The initial suggestion from the project supervisor was to add polynomial sampling. Thus, the instance variables added at this stage were two drop down menus (JComboBox) to select the polynomial order and a nominal value for the noise. Additionally, a text field to specify the number of points and a button to generate the data were added to the class design. Other types of data generation and their associated widgets were added to the class as neessary during the implementation phase.

The interface as a whole is controlled by the class MainGUI. This extends Swing's JFrame, a heavyweight class which implements a window to contain all the other GUI elements. The main responsibility of this class is to create and layout the other panels. Additionally, it has a JMenuBar to control the application settings which are not specific to any one algorithm. Another important responsibility of the class is to provide the logic for loading an algorithm. When the user selects a new algorithm from the tree, the con-

figuration panel for that algorithm has to be added to the display, and the appropriate data generation options enabled. Having AlgorithmPanel handles these tasks would introduce a high degree of interaction coupling into its design and reduce the overall cohesion of the class. Thus, MainGUI has a `loadAlgorithm` method which accomplishes this setup process. This is a cleaner design, reducing the degree to which the GUI's component classes need reference one another.

5.2.2 Plotting

The central goal of the teaching aid is the visualization of machine learning algorithms, and so the plotting functionality is the most important area of the application. The design delegates all plotting tasks to the class Plotter, another JPanel extension. Having the semantics of plotting separate from the algorithm implementation increases the cohesion of the design, in that an algorithm is only concerned with machine learning problems and need not be concerned with how its output is drawn. This offers a further advantage in that the plotting mechanism can be altered in the future without requiring changes to the underlying algorithms.

The important fields in Plotter include a Set of Algorithm objects, a HashMap relating double values to colours, and two Datasets — one for the training data, and one for the test data. Maintaining a collection of Algorithms allows multiple models to be drawn at the same time, a useful feature for regression methods. The mapping of doubles to colours allows the label on a Datapoint to be used to determine its colour when it is drawn. The presence of a separate test Dataset allows the performance of a model to be evaluated independently of its training set, an important feature since good test predictions are the goal of the majority of machine learning techniques.

Plotter is the largest class in the teaching aid, and this was the case from the design stage onwards. The central method is `plot`, which takes a Dataset and draws its output on the screen. The same method is used for plotting the training and test sets, and for plotting the set returned by each algorithm's `getOutput` method. Different types of dataset — points, curves and shaded grids — are distinguished by their `style` field. A supplementary method, `plotErrorBar` is included to draw the error bars for any points which are flagged to show them. There is also a method to draw the axes on the plot. These three methods were designed to be called from within Plotter's `paintComponent` method, which guarantees that all changes to the display are made in a thread safe manner.

Plotter also has methods for converting back and forth between the coordinate system used by the physical panel and the data's frame of reference.

Swing components have an internal coordinate system which places the origin at the top left corner and marks coordinates in integer steps, with the x coordinate increasing to the right and the y coordinate increasing downwards. This results in many points having large values, which can make matrix inversion more computationally intensive, a potential problem for regression algorithms in particular. As a result, the coordinates of the data are transformed into the range $[-10, 10] \times [-10, 10]$ for storage. The integer coordinates are recovered through an inverse transformation when points need to be drawn. This design has the additional advantage that resizing the display is very simple, since the height and width of the panel can be used dynamically in the transformation.

The final important feature of Plotter is that it implements the MouseListener interface specified by `java.awt.event`, allowing the panel to listen for mouse clicks on itself. The class has a number of methods which can be called to add and remove Datapoints when the user clicks on the plot.

5.2.3 The OptionPanel Class

Each algorithm in the teaching aid has its own particular set of configuration options, so an all purpose class to manage the interface for these would be a poor idea. At best such a class would be visually crowded and confusing, and in all likelihood would be unusable. From a design and implementation standpoint, a generic options class would have poor cohesion and would be difficult to maintain and extend. Clearly, a stronger design is necessary.

To this end, the teaching aid has the OptionPanel class. This is a very simple JPanel extension which acts as a placeholder for the right hand part of the display when no algorithm is loaded. It specifies a single instance variable, an Algorithm. The options for each individual algorithm are controlled by a class extending OptionPanel. The base class specifies a protected method `layoutPanel` which adds the introductory text to the panel, but subclasses can override this method to layout the GUI elements they require. This design allows as much customization as needed, but does not require the rest of the GUI to know about each of the different subclasses. In particular, MainGUI maintains an instance variable of type OptionPanel as part of the layout, but this can be set to any of the specific subclasses by calling the appropriate constructor in MainGUI's `loadAlgorithm`. This is a simple design, but is still flexible and easily extendable. The class OptionPanel and its subclasses make up the `gui.data` subpackage.

5.3 Modularity

From the outset of the project, one of the design features which was deemed most important was modularity. This refers to the ability of the teaching aid to be easily extended to incorporate new algorithms. The motivation for this was twofold. First, designing to ensure modularity encourages focused, cohesive classes and a clear separation between the interface and the underlying models. Secondly, if another developer wishes to extend the teaching aid in the future by adding new algorithms, a modular design clearly makes their task simpler.

The decision to keep plotting and algorithms separate means that developers can add new algorithms without altering the existing Plotter class. They need only implement their algorithm along with an OptionPanel subclass to control its settings. The steps necessary to add a new algorithm aid are as follows:

1. Write a class to implement the algorithm, extending either Classifier, RegressionAlgorithm or Clusterer
2. Write a control class for the algorithm, extending OptionPanel
3. Add the name of the algorithm to the JTree in the AlgorithmPanel class (1 line of code)
4. Add code to `loadAlgorithm` in MainGUI to handle the specifics of loading your algorithm ($\sim$ 4 lines of code)

Clearly, the process is a simple one. Excluding any difficulties inherent in implementing the algorithm, very little effort is required to integrate the new algorithm with the teaching aid.

5.4 Final Design

During the implementation process, a number of other packages and classes were added to the teaching aid over and above those which were designed initially. The final package diagram, along with class diagrams for each package, are included in appendix B of this document. The interesting features of some of the additional classes are discussed at greater length in the next chapter. For the classes discussed in this chapter, the final designs reflect more methods and instance variables than were mentioned in the discussion. These represent routine implementation details which were not interesting enough to discuss in detail here, and added nothing novel to the design. A

description of the purpose of each of the variables and methods in the teaching aid is available in the automatically generated Javadoc, which may be found on the accompanying CD.

Chapter 6

Implementation

With a strong design in place, the next phase of the project was to implement the teaching aid. This chapter gives an overview of the implementation process, and describes in detail some of the interesting difficulties which arose during the development, and the corresponding solutions.

The application was written in Java, using the Eclipse IDE. It was built and tested primarily using version 1.6.0 of Java, running on Microsoft Windows Vista. Some additional testing was carried out in Ubuntu Linux, and the weekly demonstrations of the teaching aid to the project supervisor used Apple's OSX. Thus the application saw use on a variety of platforms and the core functionality worked as expected on all of them.

6.1 Development Method

The initial project proposal called for the development to be carried out according to the Feature Driven Development (FDD) [6] model, which involves successive iterations of design, implementation and testing to produce small modules of code called features. This was chosen because it seemed to fit well with the desired modular nature of the project — each algorithm is essentially a feature.

However, due to the time constraints involved in the project, it was decided that the FDD model was too granular in its definition of development phases. It places emphasis on planning and designing on a per feature basis, then refining the overall design model before actually writing any code for a given feature. This seemed somewhat redundant in the case of the teaching aid, because care was taken in the initial design to ensure good use of inheritance structures so that any one algorithm implementation should require very little additional design. Therefore, the development model chosen

amounted to an agile development spin on the FDD model.

Each new algorithm was briefly analysed to identify the additional methods, if any, which would be required to implement the algorithm. The algorithm was then immediately implemented, followed by its option controller class. The controller classes were designed with quick sketches identifying the buttons and other GUI elements which were necessary to control the corresponding algorithm. Next, the controller was integrated with the GUI — by design, this was a very quick and easy process. Finally, the new algorithm was tested to ensure its output was correct and there were no plotting errors. Once it was clear that the algorithm was working as intended, development moved on to the next feature.

6.2 First Prototype

Before implementing any algorithms, the first step of the implementation process was to build the basic framework for the GUI. No interaction was added at this stage. Rather, the primary concern was to get the layout correct and add the core GUI elements. This initial prototype corresponded to a basic implementation of the MainGUI, AlgorithmPanel, DataPanel and OptionPanel classes. Most of this process was straightforward and did not deviate from the design. A screenshot of the application at the end of this phase in the development is shown in figure 6.1.

In AlgorithmPanel, the tree showing the basic algorithm types is included, but at this stage the only explicit algorithm displayed is minimized loss. This is because the list of algorithms had not been finalized at this point in the development. Minimized loss was chosen as the first because it is a very simple algorithm and would be easy to implement. The plan for adding more algorithms was to implement one of each family as quickly as possible, then to add as many further algorithms as the development time allowed.

The OptionPanel class contains only a label inviting the user to choose an algorithm. Since no algorithms were coded, no child classes of OptionPanel were added either. However, one useful feature that was added to the design at this part of the process was placing the options in a tab pane, represented as a JTabbedPane in the MainGUI class. This can be seen in the screenshot as the tab with "Welcome" written on it. This was added to allow for the possibility of multiple simultaneous models for some algorithms. Each model can have its option panel in an individual tab. The other option considered for multiple models was to open successive option panels in small separate frames. This was rejected because it would add too much clutter to the screen and would make it too easy to lose a panel behind other windows. The tab

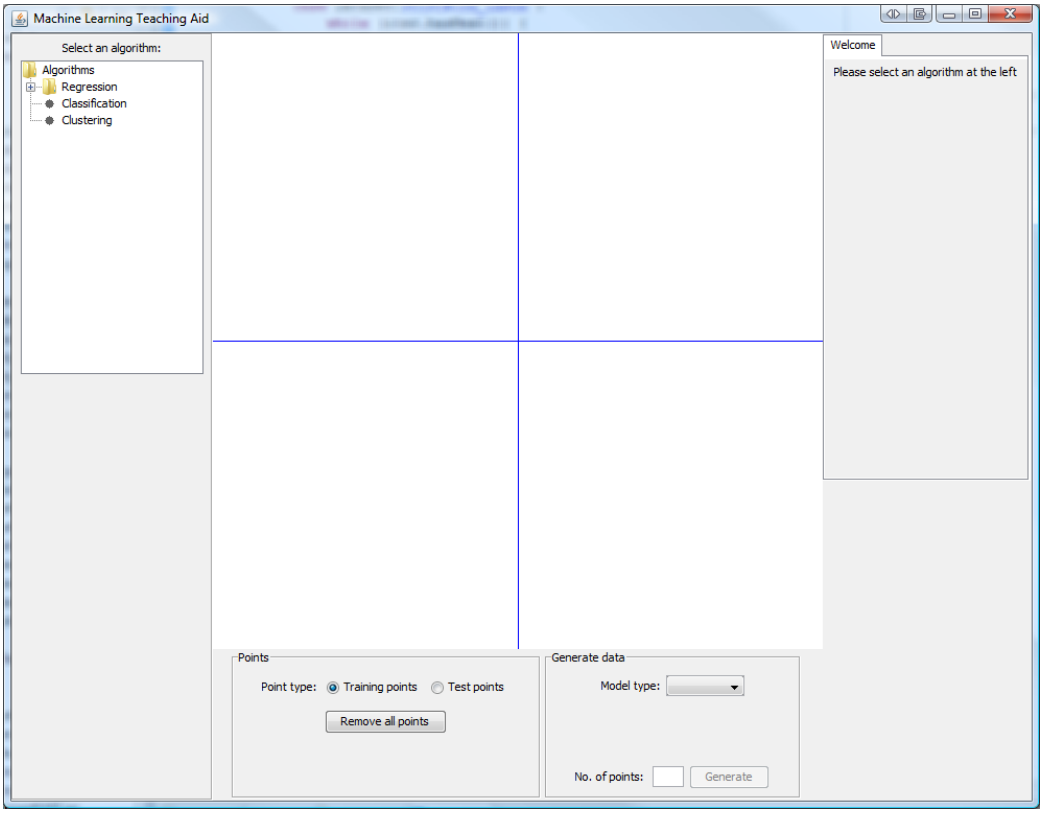


Figure 6.1: Early GUI Prototype

pane approach allows many models to be controlled using a small area of screen space, and makes switching between models clear and intuitive.

The version of Plotter implemented in the prototype was very basic. It implemented the MouseListener interface but did nothing in response to any MouseEvents. The only things drawn by the `paintComponent` method were the axes.

The DataPanel class has a number of its key features in place in the prototype. The two subpanels are clearly identified and labelled. The 'Points' subpanel has two JRadioButtons to choose between training and testing points, and a JButton to clear all points. The 'Generate Data' subpanel has a JComboBox to select the type of model to sample from, although no options were added to it until later. Finally, a JTextField to enter the number of points to sample and a JButton to carry out the sampling are included.

Note that no menu bar was added to the GUI for this prototype because the functions for the menu had not been clearly defined.

While most of the implementation up to this point was simple, a small

number of problems did occur. First, it was discovered that adding a JTree object directly to the AlgorithmPanel caused strange visual effects. The white background of the panel shrunk to match the exact size of the tree, leaving large gaps of grey space which were visually unappealing. This problem was solved by adding the JTree to a JScrollPane, and adding that pane to the panel as a whole. This kept the background at the full size of the panel, as intended. This adds an additional benefit in that it allows scroll bars to be added in the event that the tree becomes too wide to be displayed in the allocated width.

A second problem was that laying out the component panels of the GUI was not as simple as anticipated. Adding the component panels directly to MainGUI was tried using a variety of Java's available layout managers, but it was impossible to line up the components as desired and have them maintain their relative positions as the frame was resized. An initial solution considered was to fix the size of the window, but this is a suboptimal solution as users with large screens should be able to increase the size of the teaching aid and gain a clearer view of the plot. The problem was eventually solved through the introduction of a number of additional JPanels in MainGUI, which were used to contain the key GUI components and allow more precise control over the layout. This makes the layout code fairly involved and adds a variety of instance variables which have little to do with the overall goals of the teaching aid, but these side effects were unavoidable.

6.3 Adding Interaction

6.3.1 Datapoints and Datasets

Once the basic interface framework was ready, the next task was to add core functionality by implementing the use cases. For this to happen, the Datapoint and Dataset classes had to be implemented. Datapoint is a very simple class, and none of the methods caused any implementation difficulties. The same is largely true of Dataset, with the exception of the `kNearest` and `fold` methods. These required additional planning to implement successfully.

The algorithm created to split a dataset into folds is shown in 6.3.1. Essentially, the algorithm distributes the points in the dataset as evenly as possible across the folds, then assigns any that are left over to random folds. The order in which points are chosen to be assigned to a fold is random. All random numbers required for the algorithm are generated by the Random class, a part of the core `java.util` package. An unexpected benefit of the decision to use a List to represent the contents of a dataset was

that implementing this algorithm in Java became very easy. Once the index of the next point to be added to a fold was identified, the steps of removing the index from the List `indices` and getting the corresponding Datapoint could be carried out in a single line of code.

An initial mistake made when implementing the algorithm was to pass the points from the main dataset to the folds by reference. Thus the same point was referenced by both Datasets, and the original could have the state of its points changed when this was not intended. This could occur during cross validation, for example. To solve this problem, the Datapoint class was altered to include a method `clone` which returns a new Datapoint object with the same position and label. This is called when a point is to be assigned to a fold.

The code for `kNearest` was easier to write than that for `fold`, but still required some thought. The problem is that the points in the set must be sorted in order of their distance from a test point. When sorting arrays, Java's default `Arrays.sort` method expects the objects in the array to implement the Comparable interface, which specifies a global ordering of the members of a class. A global ordering was no good in this instance, since the ordering is dependent on the location of the test point, which is not known in advance. A possible solution was to implement a sorting algorithm manually, but this seemed wasteful. The Java API provided a better solution in the form of a version of `Arrays.sort` which takes a Comparator object as a second parameter. Comparator⟨T⟩ is a generic interface which specifies a single method, the purpose of which is to compare two objects of type T based on a given ordering. By defining a Comparator inside the body of the `kNearest` method, two Datapoints from the set can be compared so that one is 'less' than the other if it is closer to the test point. The array of data is then sorted using this metric, and a Dataset consisting of the first K points is returned.

6.3.2 Adding Points

The next function added was the ability to plot points directly by clicking on the plot. This was done by overriding the `mouseClicked` method in the Plotter class. This method is specified in the MouseListener interface, and is called automatically by Java in response to the user clicking on the component. The initial implementation simply got the location of the mouse click, and added a Datapoint at the corresponding location in the transformed data coordinates to the training data. To transform from the panel coordinates, which are integers with the origin in the upper left corner, to the data coordinates, a standard Cartesian system with the origin in the center of the

Algorithm 1 Algorithm for folding a dataset

To partition the Dataset `data` into an array of *numFolds* smaller Datasets:
Let `folds` be a Dataset array of length *numFolds*
Let n be the size of `data`
Let *foldSize* be the integer part of n divided by *numFolds*
if *foldSize* $= 0$ **then**
  foldSize $\leftarrow 1$
end if
Let `indices` be a list of the integers from 0 up to $n - 1$
Let `assignments` be a 2D array of size *numFolds*×*foldSize*
for $i = 0$ **to** *numFolds* $- 1$ **do**
  for $j = 0$ **to** *foldSize* $- 1$ **do**
    if `indices` is non-empty **then**
      Remove a random element of `indices` into `assignments`$[i][j]$
    else
      `assignments`$[i][j] \leftarrow -1$
    end if
  end for
end for
for $i = 0$ **to** *numFolds* $- 1$ **do**
  Let `folds`$[i]$ be an empty Dataset.
  for $j = 0$ **to** *foldSize* $- 1$ **do**
    Let k be the value of `assignments`$[i][j]$
    if $k \neq -1$ **then**
      Add the k-th point of `data` to `folds`$[i]$
    end if
  end for
end for
while `indices` is non-empty **do**
  Let i be a random integer between 0 and *numFolds*
  Let j be a random entry of `indices`
  Add the j-th entry of `data` to `folds`$[i]$
end while
return `folds`

panel, the following transformations were used:

$$x_{data} = \frac{2 \times \texttt{XRANGE} \times x_{panel}}{w} - \texttt{XRANGE},$$

$$y_{data} = \texttt{YRANGE} - \frac{2 \times \texttt{YRANGE} \times y_{panel}}{h}.$$

`XRANGE` and `YRANGE` are class constants for plotter which define the size of the intervals to transform the data to. Both were initially set to 1, although later in the development the values were changed to 10. w and h are respectively the current width and height of the Plotter, and these values are obtained dynamically each time the transformation method is called. Thus, when applying the inverse transformation, the points keep the same data coordinates but their plotted locations are scaled to match the size of the plot. This prevents any problems when resizing the window.

Actually drawing the points required the implementation of the `plot` method. This is a general method for plotting any type of dataset. It determines how to plot a given Dataset by switching on the `style` variable. The training data were to be drawn as discrete points, so the code for plotting datasets whose style was `POINTWISE` was added. This is accomplished by iterating over the dataset and drawing a circle centered on each point. Painting in Java is done using an object of the Graphics class, which has methods for drawing assorted 2D shapes including ovals, so plotting points was simple. The colour of a point is determined by looking up the label of the point in the Plotter's map of doubles to colours. The circle is filled in this colour, and the outline is drawn in black.

It is important to be able to choose whether new points are added to the test or training set. This is controlled by the DataPanel class, which has two radio buttons for set choice. These radio buttons change the value of a field called `mode` between the two class constants `TRAIN` and `TEST`. However, in the initial design Plotter had no access to this information. This oversight was corrected by adding a reference to the GUI's DataPanel object in the Plotter class. The Plotter can then query it to obtain the current mode, and add new points to the correct Dataset.

Since the first algorithm to be implemented was a regression method, the options for specifying the class of new points were not added to the GUI at this point.

6.3.3 Removing Points

The next obvious function to add was the ability to remove points. The data generation panel already had a button to remove all points from the display.

In order for this to work, the DataPanel had to know about the Plotter in order to call its method for clearing the points. However, the Plotter already holds a reference to the DataPanel, so this creates a cycle of references when attempting to create the two panels as the application starts. To solve this, the reference in Plotter was replaced with an instance variable `parent` of type MainGUI. When the Plotter requires information about the DataPanel, it can call `getDataPanel` from the parent object. Since such a call is only made after all the GUI elements are set up, there is no problem with null references.

In addition to removing all points, the requirements call for the ability to remove individual points to investigate the effect of small changes to the training data. The obvious method for doing this is to click on an existing point to remove it. To accomplish this, a method was added to the Dataset class indicating which, if any, of the points in the set 'contain' a mouse click, in the sense that the click is inside the circle representing that point. This is a slight overlap between the theoretical nature of the Dataset class and the requirements of drawing the data, but the decision keeps the code to remove an individual point simple.

6.4 Algorithm Implementation

With the basic GUI and the data framework in place, the next step was to implement algorithms and their associated GUI controls, beginning with the minimized loss method. This section contains a detailed description of the challenges raised by this process. The algorithms are described in the order they were implemented.

The original project proposal expected that the plotting aspects of the GUI would be implemented separately from the algorithms. However, once development had begun it quickly became apparent that this was an infeasible suggestion. Creating the plotting functionality in isolation would require the manual generation of test Datasets for drawing. This seemed an inprudent use of the development time, so it was decided that the code for plotting a particular type of Dataset would be implemented alongside the first algorithm which required that plotting mode.

6.4.1 Minimized Loss

This algorithm trains by building up the matrix $\mathbf{X}$ and vector $\mathbf{t}$ as defined in 2.1.1, then computing the optimum parameters $\widehat{\mathbf{w}}$ from

$$\widehat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{t},$$

and the average training loss from

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\widehat{\mathbf{w}})^{\mathrm{T}}(\mathbf{t} - \mathbf{X}\widehat{\mathbf{w}}).$$

The implementing class, MinLoss, maintains a variable holding the current value of the training loss, as well as a method to compute the loss of the trained model on any Dataset. All of the operations the algorithm has to perform involve matrix calculations, and thanks to the Jama library, these are very easy to implement. The most complex part of the code is that which builds $\mathbf{X}$ according to the terms which have been enabled, and that is only slightly involved rather than actively difficult. Once that is done, the optimal parameters can be computed in only a few lines of code.

To plot the output of a regression method, the `plot` method in Plotter was extended to include support for Dataset objects with the style `INTERPOLATED`. The interpolation used is very simple, in that the code just iterates through the Dataset and draws straight lines between successive pairs of points. When the method is called, the teaching aid creates the array of x values to be evaluated dynamically based on the size of the Plotter. By passing in x values spaced two pixels apart, the resulting Datapoints are close enough that the curve drawn appears smooth. A screenshot of the application operating in this fashion is shown in figure 6.2

Figure 6.2 also shows the option panel for the minimized loss algorithm, an object of the class MinLossPanel. The selection of regression terms is controlled by a number of check boxes, which fits well with the use of booleans to indicate the terms which are enabled. There are also buttons to train the model and run cross validation.

The option panel has two labels which display the current training and testing loss. Implementing these initially seemed a simple process, but a problem emerged when testing. Each time a new training point is added to the display, the algorithm is retrained on the modified dataset, which changes the training loss. However, the panel is unaware of this change and its labels are not updated. To attempt to solve this, code was added to MinLossPanel to check the training and testing losses and set the label text each time the panel was repainted. This was initially unsuccessful, because only the Plotter was being marked for repainting and the labels still did not update. The solution to this was to add a method to MainGUI which forces all the option panels in the tab pane to repaint. Since Plotter already has a reference to its parent MainGUI, it can call this method every time the display is redrawn and correctly update the labels. All other algorithms which show some performance measure on their option panel now implement similar code to set their label text correctly before repainting.

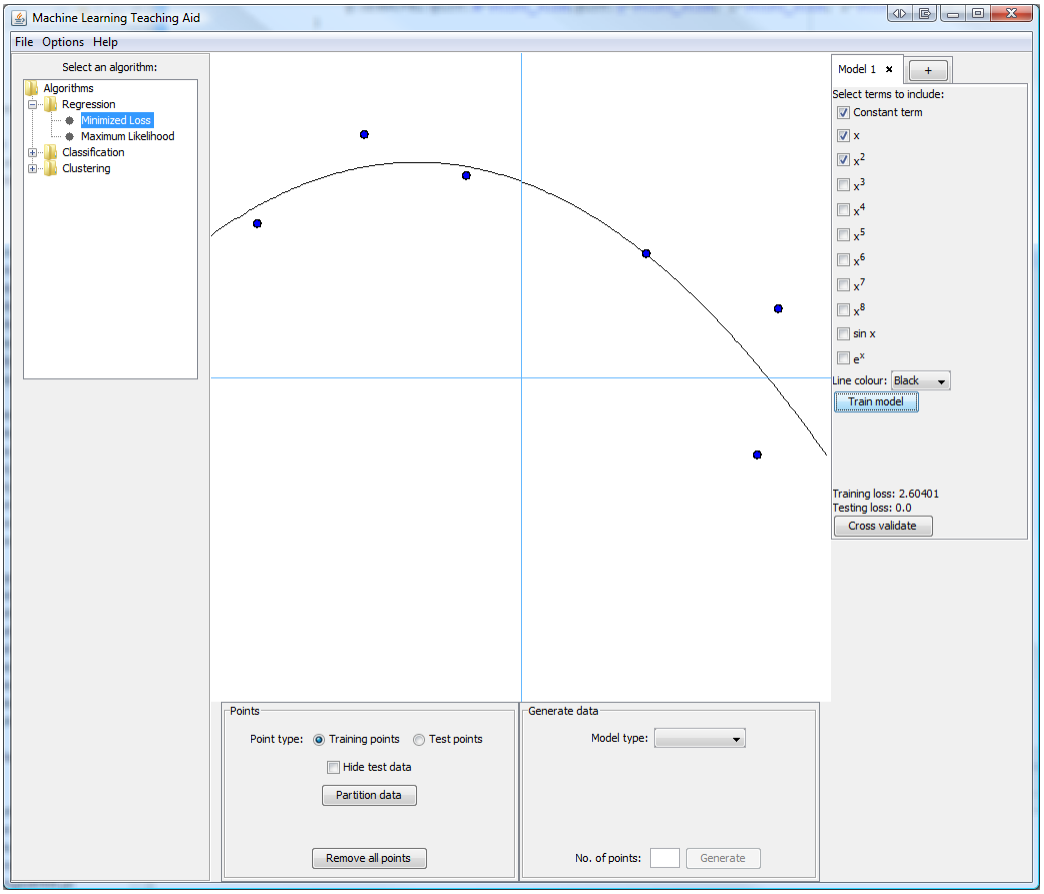


Figure 6.2: The minimized loss algorithm — the output is the curve, based on the blue points

The tab interface for adding multiple models was also implemented alongside this algorithm, and can be seen in the figure. New models are added by clicking the button in the rightmost tab, causing MainGUI to query the current OptionPanel to find its constructor method, and add a new tab with a fresh instance of the class. Removing models was a slightly more difficult prospect. This was accomplished by adding a class TabCloser, which is a small panel containing the name of a particular model and a button to close it. A TabCloser is used as the component shown in the tab selector for each model.

In the initial implementation, all models were drawn with black lines, but this was unclear when many models were present. One solution explored was assigning random colours to new models, but this was abandoned because the colours generated were often unpleasant. Additionally, the random colours

were sometimes too similar to one another to be readily distinguished. Instead, a drop down menu was added to the option panel allowing the line colour for each model to be chosen from a set selection.

6.4.2 KNN

The next algorithm implemented was K Nearest Neighbours. This posed no significant development problems, since the `kNearest` method in the Dataset class rendered the implementation almost trivial. The only setting to configure on the option panel is the value of K, which is controlled with a text field. Two labels on the panel show the misclassifications the algorithm makes on the training and test sets.

Most of the development time for this algorithm was spent adding the framework functions to allow classification output to be shown — namely, a menu to choose the class of added points and the ability to plot datasets with the `SHADED` style. Both were relatively simple tasks. To plot such a dataset, the `plot` method in Plotter iterates over the dataset, and for each point it gets the appropriate colour from the map and draws a square with the upper left corner at the point in question. The size of the square is controlled by a class constant. Increases the size improves the performance of the teaching aid when repainting, but makes the boundary between classes less smooth. It was found through trial and error that a good trade off between fidelity of image and speed was achieved with a grid size of five pixels.

When the `plot` method is called for a Classifier, the teaching aid passes a grid of points based on the current size of the Plotter. This grid is spaced slightly closer together than the grid which is drawn. If this step was not taken, the output was found to contain patterns of lines which were not coloured correctly.

Figure 6.3 shows a screenshot of the KNN algorithm in operation.

6.4.3 K Means

The K Means algorithm was next to be implemented. As an iterative clusterer, the output of this method evolves over time. It is important for a teaching tool to reflect this, allowing the user to step through the iterations. Thus, the standard `train` method was not particularly useful, since the way it is called by the GUI means it cannot easily reflect the changes over successive iterations. Instead, two methods were added — one to assign all the points to their closest mean, and one to update the means. The former of these methods returns a boolean indicating whether any of the assignments

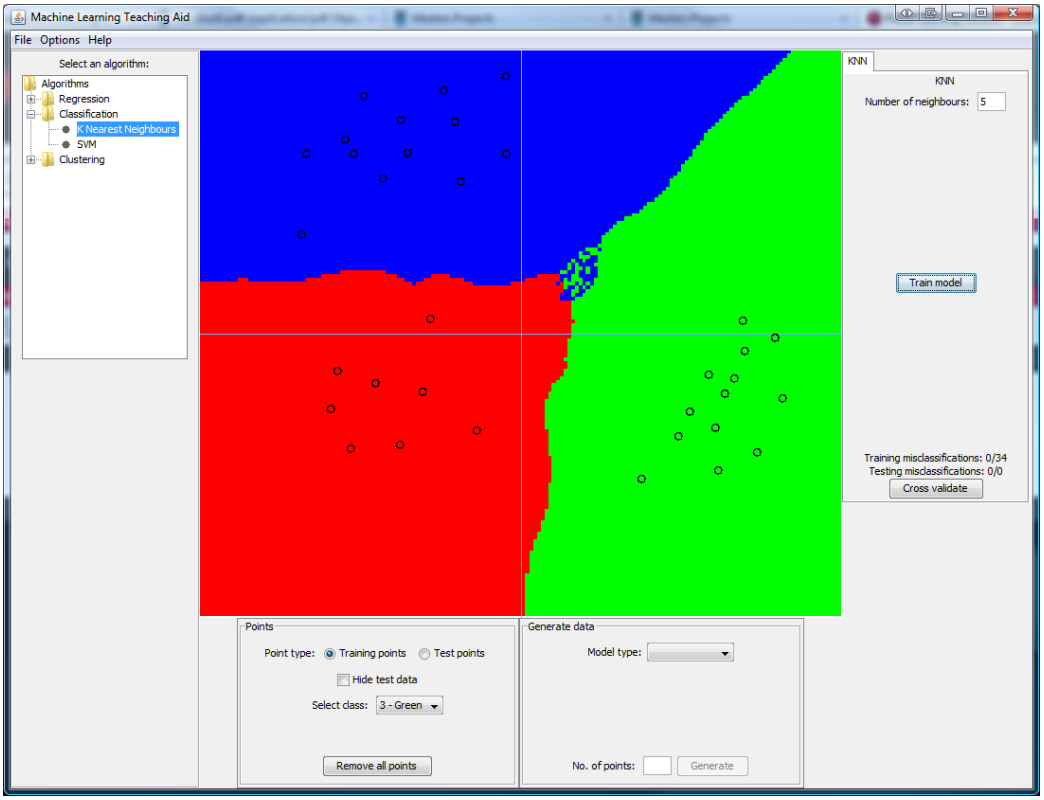


Figure 6.3: The KNN algorithm — the output is the shading showing the points classified as red, blue or green

changed, which is used to check for convergence of the algorithm. The controller class, KMeansPanel, has a 'Next Iteration' button that calls each of these methods once when it is clicked. After some initial testing, the project supervisor suggested the addition of a second button to jump to the solution at convergence. This allows users to quickly test a given value of K.

The colours for the clusters were initially assigned randomly each time the algorithm started clustering. This worked reasonably well in some cases, but in others the colours were too similar and it was extremely difficult to distinguish between clusters. As a compromise measure, the first five clusters were assigned fixed colours so they could always be easily distinguished. Any subsequent clusters are coloured randomly.

As well as training differently from other algorithms, it was found that KNN requires a different type of output. The algorithm communicates its output primarily by colouring the training data. The `getOutput` method returns a dataset showing the means. This dataset has the style `POINTWISE_LARGE`, so the means are drawn as circles three times larger than standard points.

This enables them to be clearly identified as separate from the data.

The semantics of testing are not well defined for clustering in the teaching aid. No obvious performance measure exists to evaluate on the test set, so currently nothing is done. This is an area for future improvement to the software.

Figure 6.4 shows the K Means algorithm in operation.

Figure 6.4: The K Means algorithm — the algorithm has found four clusters in the data, with the means represented as large points

6.4.4 SVM

With the three basic types of algorithm represented in the teaching aid, the development was free to choose any type of algorithm to add. The requirements gathering exercise suggested that students on the machine learning course found SVM to be one of the most conceptually challenging algorithms on the syllabus, so it was a high priority for addition. If a viable quadratic

programming solver had been available, it would have been fairly straightforward to implement an SVM, but without one the proposition was more challenging. Thankfully, the project supervisor proposed the SMO algorithm as a strong alternative. Implementing this algorithm required a fair amount of work, but still less than an entirely new QP solver.

The full details of the SMO algorithm are beyond the scope of this report, but the basic concept is that pairs of Lagrange multipliers are optimized analytically, as opposed to the QP approach which solves numerically for all the multipliers at once. The SMO approach thus scales much better with the size of the training data, and in some cases has been shown to be several orders of magnitude faster than competing algorithms. The modification proposed by Keerthi et al [15] is what the class SVM actually implements — this improves on several small deficiencies of the SMO technique, further increasing the speed of convergence.

Keerthi et al suggest that the implementation should maintain five sets, I0 to I4, which represent subsets of the α_n that meet certain conditions. However, writing the code to add and remove values from these sets seems overly complex, particularly given that the conditions for membership of any the sets amount to two equality tests on primitive types. Thus, the teaching aid implementation replaces these five sets with methods which test a given α to see if it meets the conditions for the corresponding set. This was much simpler than implementing the sets explicitly, and would only be significantly slower for huge datasets — much larger than the typical sets used in the teaching aid.

SVM was the first kernel algorithm implemented, so a framework for kernel functions was implemented alongside it. This takes the form of an interface Kernel which specifies a method `evaluate`. This method takes two points $\mathbf{x}$ and $\mathbf{y}$ and returns the kernel function $k(\mathbf{x}, \mathbf{y})$. The teaching aid currently specifies three kernels, each represented by a class implementing the Kernel interface. These are LinearKernel, with

$$k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^{\mathrm{T}}\mathbf{y},$$

GaussianKernel, with

$$k(\mathbf{x}, \mathbf{y}) = \exp\left[-\beta(\mathbf{x} - \mathbf{y})^{\mathrm{T}}(\mathbf{x} - \mathbf{y})\right],$$

and PolynomialKernel, with

$$k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^{\mathrm{T}}\mathbf{y})^{\beta}.$$

In the Gaussian and polynomial kernels, the additional parameter β is controlled by a text field in the option panel.

The initial implementation was producing an output successfully, but behaved incorrectly, producing too many support vectors. This was eventually found to be a result of the `predict` method returning a hard assignment of -1 or 1, when in fact it needed to return the raw output of the SVM. It was this problem which led to the introduction of the `predictLabel` method into the Classifier class — the need for this distinction had not been identified during the design phase.

The project supervisor suggested that in addition to drawing the output showing the hard assignment of each point to one of the two classes, it would be nice to show a shading of the display based on the raw value of the SVM at each point. In effect, this would display a kind of confidence measure for the algorithm across the plotting space. To accomplish this, an additional style, `COMPLEX_SHADED`, was added to the Dataset class. This is currently defined only for binary classifiers. Plotting is done by assigning pure blue to the point with the minimum value of SVM output across the whole space, and pure red to the point with the maximum value. For other points, the Plotter computes the appropriate proportion of red and blue based on the position of the SVM output relative to the maximum and minimum. In addition to being a useful visualization, the effect is also quite pleasing visually. A screenshot showing this mode of operation can be seen in figure 6.5.

6.4.5 Kernel K Means

The kernel version of K Means was implemented next. Given that the algorithm operates very similarly to standard K Means, and the kernel framework was already in place from SVM, coding this algorithm was fairly straightforward. The first attempt at implementing the algorithm failed, since the assignments of the points were updated as the algorithm went, and so the assignments of the earlier points in the dataset were used to update those of the later points. This was pointed out quickly by the project supervisor, and the problem was corrected by maintaining a separate set of new assignments, and copying this to the training data only after all the new assignments were made.

This algorithm was found in testing to be very sensitive to initial conditions. The first implementation randomly assigned the starting cluster of each point, which sometimes led to one class dominating the others and a suboptimal clustering being produced. The problem was partially solved by using a different system of initial assignments. The first $K - 1$ clusters are each assigned a single random point, and the K-th cluster is assigned the remainder of the data. This improved the number of cases where all clusters were represented in the solution at convergence, but the algorithm remains

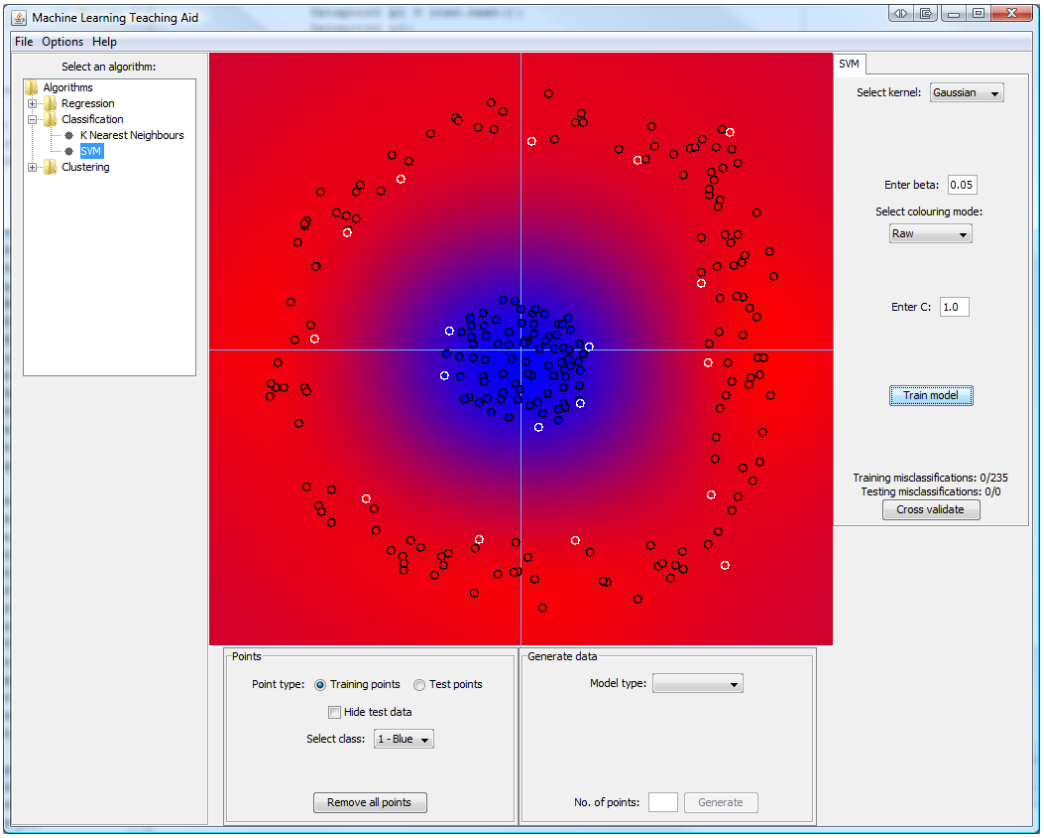


Figure 6.5: The SVM algorithm — a Gaussian kernel has been used. Raw colouring mode is shown, and the support vectors are highlighted in white.

sensitive to the random starting positions of the single point clusters. This seems to be a deficiency of the algorithm itself, rather than the implementation.

6.4.6 Maximum Likelihood

The final algorithm which was implemented was the maximum likelihood regression method. This was fairly simple, since much of the code is identical to that for minimized loss. The optimal parameters are the same, and the optimal variance is computed in the same way as the average training loss for minimized loss. Thus a lot of the code was reused from the earlier algorithm. However, there were two main differences.

First, the code for adding error bars. For points which errors had been enabled, the MaxLikelihood version of `getOutput` sets the size of the error

equal to the predictive variance,

$$v_{new} = \mathbf{x}_{new}^{\mathrm{T}}(\operatorname{cov} \widehat{\mathbf{w}})\mathbf{x}_{new}.$$

Error bars are drawn as vertical bars extending a distance v_{new} up and down from the point, along with two horizontal lines to mark the ends of the error bar.

Secondly, to compute the likelihood of test sets, it is necessary to evaluate Gaussians with given means and covariance. To facilitate this, a class Gaussian was created that represents a one dimensional Gaussian distribution. The class has methods to set the mean and variance, and to evaluate the Gaussian for a given x. When computing the likelihood of a Dataset, one of these objects is created for each point, and the values returned by the `evaluate` method at each point are multiplier together to produce the likelihood.

6.5 Sampling Data

Alongside the algorithm implementations, a variety of techniques for generating data were added to the teaching aid. These were added at various points during the development, but the discussion on them is collected here for clarity. For each technique, a class extending the Sampler class was created, specifying an implementation of the `nextPoint` method.

All of these data generation techniques are controlled via the DataPanel class. When an option is selected from the drop down menu, the panel displays the appropriate configuration options. This is accomplished using a CardLayout, which is similar in function to a tab pane, except that different views are selected by a mechanism other than tabs.

6.5.1 Polynomial Sampling

The first data generation technique was sampling from a polynomial. This is done by the PolySampler class, and is fairly straightforward. The user specifies a polynomial order and a value for the noise. The coefficients of a polynomial of the appropriate order are randomly generated. A random x value in the range $[-10, 10]$ is chosen, and the value of the polynomial at that x is computed. The effect of noise is simulated by adding or substracting a random value, multiplied by the noise value which was previously set. If the resulting t value has $|t| > 10$, the point is rejected and a new x value is tried to ensure that the sampled data is visible on the plot.

The difficulty in implementing this technique arises in scaling the coefficients appropriately so that the polynomial which is sampled from is 'interesting.' In particular, it should have clear turning points in the visible data range, and should not be too steep. A heuristic that scales the coefficients so that higher order terms have lower coefficients on average has been implemented, which results in interesting polynomials a fair proportion of the time. There is still room for improvement, however, as some of the polynomials generated are uninteresting in the plotted space.

Once the desired number of points have been sampled, the coefficients of the polynomial are randomized again. Previously, a separate button was included to randomize the coefficients, but user feedback suggested that most users would only want to sample from a given polynomial once anyway.

6.5.2 Gaussian Sampling

The class GaussianSampler implements sampling from a two dimensional Gaussian distribution. The mean vector and covariance matrix of this Gaussian are set by clicking and dragging on the display to draw an ellipse. For simplicity, only ellipses with axes orthogonal to the plot's axes can be drawn. These correspond to Gaussians with diagonal covariance matrices. Once the user finishes dragging, the mean is set to the center of the ellipse and the diagonal components of the covariance to the sizes of the semimajor and semiminor axes.

The particular sampling algorithm implemented is taken from [10]. If $\mathbf{z}$ is a vector of independent normally distributed values, $\boldsymbol{\mu}$ is the mean of the Gaussian, and $\boldsymbol{\Sigma}$ is the covariance, then a vector $\mathbf{d}$ which is a sample from the Gaussian can be produced by

$$\mathbf{d} = \boldsymbol{\mu} + \mathbf{A}\mathbf{z}.$$

Here, $\mathbf{A}$ is the lower triangular matrix produced by the Cholesky decomposition of $\boldsymbol{\Sigma}$. Since Java's Random class provides a method `nextGaussian`, it is trivial to create $\mathbf{z}$. The Jama matrix library is able to do a Cholesky decomposition, and so this sampling algorithm was easy to implement.

6.5.3 Polygon Sampling

The project supervisor expressed an interest in being able to generate data from inside a user specified polygon. It was unclear how feasible this was, but on inspection of Java's graphics API, the class Polygon was found. This represents a set of points joined in order to create a polygon. It includes

a method to test if a given point is inside the polygon or not, which is the difficult part of sampling from a polygon. Additionally, the Graphics class has a method to draw a polygon, so no additional code is needed for that.

The teaching aid implements this function by allowing users to click on the display to define the vertices of the polygon. When the user clicks the button to generate data, the application simply samples random points in the plotting space, keeping those which fall inside the polygon and rejecting others, until the requisite number of points is generated. This rejection sampling technique is fairly inefficient, but given the speed of modern computers, the process still takes a very short time. This technique is a very powerful one, allowing data to be produced from very complex shapes with minimal effort.

One main difficulty arose in implementing polygon data generation. This was related to the fact that the Polygon class only accepts points with integer coordinates. The first implementation simply kept a direct reference to the polygon, and so when the display was resized, the polygon did not resize with it. To overcome this problem, the PolygonSampler class was altered to act as a 'wrapper' for the Polygon object. A list of points in the data space was maintained, and the integer vertices of the polygon were recomputed each time the Plotter was resized. An unavoidable side effect of this was that PolygonSampler had to maintain a reference to the Plotter, so that data could be transformed back and forward between integer and data coordinates. This introduced an undesirable coupling into the design, but was the only apparent way to add the functionality correctly.

6.6 Data storage

The final use cases to be implemented were those concerning the storage of data. The ability to save and reload datasets was added, along with the ability to export an image of the plot. All of these functions are accessed in the application through the File menu.

6.6.1 Saving/Loading Data

The ability to save and load data is a useful one. It allows users to keep a record of interesting datasets they have found, and share them with other users if they wish. Datasets are saved to text files using the custom *.dset extension. A specification of this format is given in appendix D.

Files are loaded and saved by the FileManager class in the `io` package. This class consists of several static methods — one for writing a dataset

to file, one for reading a dataset from file, and one for reading from an InputStream object. The last of these is used to ensure that it is possible to load files located inside a JAR file. The implementation details are relatively uninteresting — saving involves iterating over the Dataset and writing the details of each point to the file, and loading parses the file and extracts the relevant details, creating a Datapoint for each line of the file.

6.6.2 Example Datasets

As well as allowing the user to save and load their own files, this functionality is used to package a number of interesting datasets directly into the teaching aid. These are accessed through the data generation panel, and include various cases where the algorithms 'break'. This allows useful examples to be a core part of the application rather than requiring users to obtain the example files themselves. The example datasets are simply .dset files located inside the JAR files for the program. These are accessed using the FileManager class, just as for standard files.

6.6.3 Exporting Images

Finally, a facility for exporting images of the current state of the Plotter was added. This was added as a "would like to have" requirement, as it was unclear how easy or difficult the task would be. Fortunately, it proved very easy as Java as a number of built in features supporting the creation of images. The method is as follows: create an object of the BufferedImage class, get its Graphics object, call Plotter's `paintComponent` method on that Graphics object. Finally, using the static `ImageIO.write` method, the BufferedImage can be written to one of a number of file formats. This process is also carried out by a static method in the FileManager class. Currently, the teaching aid only supports saving the display in the PNG format. This decision was taken because PNG is a lossless format, and produces a good representation of the plot. Other formats could easily be added in the future, however.

Chapter 7

Testing and Evaluation

An important part of any software development exercise is testing. For this project, the testing took two main forms. First, continuous system testing throughout the development. This was used to ensure that individual code entities were functioning as intended. Secondly, at the end of the implementation phase a user evaluation took place. The purpose of this was twofold — to assess the performance of the teaching aid against the original requirements, and to allow a representative sample of end users to try the final program and obtain their feedback. A discussion of both types of testing is presented below.

7.1 System Testing

One of the main goals of testing is to expose and remove as many defects as possible from the program. The value of testing is particularly emphasised in agile software development methods. Since the project was developed using an agile, incremental model, regular integration and testing was done throughout the implementation process. After each new piece of functionality was written and added to the GUI, it was tested as fully as possible before moving on to the next feature.

For the machine learning algorithms, the project proposal called for unit tests to be written to check that each algorithm produced the correct output. To do this, a test set must be produced and the algorithm's on the set computed externally — either by hand or using another software implementation which is known to be correct. A Java framework, JUnit, exists for writing unit tests and making assertions about the state of a program after a specific series of method calls. The initial test plan involved writing a suite of JUnit tests for each algorithm, and ensuring that it passed all its tests

before moving to a new phase of the development.

The problem with this is that creating good test sets and evaluating the output externally to the teaching aid is very time consuming. For regression models, this was manageable. It is reasonably simple to generate data from a known polynomial, then train a regression algorithm on those data and check that the parameters match those of the original polynomial. Indeed, this was done for the minimized loss algorithm, and the implementation proved to be correct.

However, with other algorithms the process of creating validation sets is much more difficult. Hand executing an SVM, for example, is a daunting prospect. Even using a program such as MATLAB to generate the data takes time, since processing must be done to get the data into a format the teaching aid will recognise. Given more development time, this would be possible and algorithm correctness could be checked formally, but for the purposes of the project it was decided unit testing was infeasible.

Instead, system testing took the form of regular user tests by the author. After each change, the program was run and time was spent trying out the new feature. Additionally, the program was demonstated to the project supervisor on a weekly basis, and he also tested it. In this way, the various configuration options for each algorithm were tested using a number of different settings. This relied on the expert knowledge of the testers to assess whether an algorithm was working or not. This is one of the biggest potential weaknesses of the project as a whole — there is no definitive evidence that any one implementation is correct. However, given that the algorithms all appear visually to be performing as they should, and respond to parameter changes in the expected fashion, the risk posed by this less formal testing procedure appears to have been avoided.

In addition to testing the logic of individual features, testing was carried out to ensure the program is robust against malformed input — in each instance where the user is asked to provide input though a text field, the value supplied is checked to ensure that errors are identified and communicated to the user in a useful fashion. Each text field has been tested with a variety of inputs, both correct and incorrect, and the program has passed all of these tests.

The system testing as a whole was successful, in that no major bugs or faulty logic is exhibited in the final software deliverable. All of the original use cases can be performed without incident in the teaching aid, so the requirements are validated.

7.2 User Evaluation

The development of the teaching aid concluded with a user evaluation. The primary aims of this were:

- To investigate how easily users were able to perform a selection of tasks in the application
- To identify any bugs which had been missed during the implementation
- To gain general feedback about any aspects of the program which were difficult to understand.

A total of eight users took part in the evaluation. Of these, three were former Machine Learning (M) students, three indicated other previous machine learning experience, and two had no machine learning experience at all. The last group were included in the evaluation to provide an indication of the system's usability from the perspective of a novice. Additionally, having no prior experience of the concepts of the subject, they were able to provide unbiased commentary on the quality of the help files built into the application.

7.2.1 Methodology

The evaluation gathered information in a number of ways, each of which is discussed below.

Task List

Participants were first given a document introducing the layout of the teaching aid and explaining the basic steps to run an algorithm. With this information, evaluators are then presented with a task list to work through. The tasks include generating data using each of the available methods, and trying each of the implemented algorithms using a variety of settings. Users are also asked to read some of the help files, and to use the file handling capabilities of the teaching aid to ensure they work.

The tasks on the list are kept fairly general in their wording — this was to assess how obvious it was to each user which control related to each part of the functionality.

Direct Observation

In conjunction with the task list, the participants were observed as they used the application and notes were taken on any occasions where they were

unsure how to proceed, and on any bugs they encountered. If they were unsure, participants were encouraged to "think aloud" about the process of navigating over the GUI to find the correct option. This provided insight into a number of usability issues. If they were unable to find the correct way to proceed, participants were directed to the help files.

Questionnaire

After working through the task list, participants were asked to fill out a short questionnaire. The questions were not designed to be analysed in great statistical detail. Rather, their purpose was to gain a sense of the participant's overall impressions of the teaching aid and to prompt further discussion. The questions posed and the responses collected are discussed in the next section. The task and an unfilled questionnaire can be seen in appendix C.

Interview

The evaluation ended with an informal interview. No set questions were asked. Instead, participants were invited to discuss their questionnaire responses to gain a more detailed understanding of any issues they faced. Further, during the interviews a number of comments were raised which did not fit into any other part of the evaluation. Brief notes were taken summarising the salient points of these interviews.

7.2.2 Results

Questionnaire Responses

The questionnaire contained two Likert style questions, asking evaluators to rate the overall design somewhere between excellent and very poor, and to rate the usefulness of the help files between useless and very useful. All respondents rated the design either very good or excellent, indicating that the majority of tasks were simple and intuitive to perform but that some required additional help. The help files were primarily deemed useful, with a few participants opting for very useful. The general consensus is that the help files provide a good accompaniment to the main application. One evaluator explicitly noted in the questionnaire that although they did not understand the mathematical content of the machine learning material, the help files still provided them with a better understanding of what the algorithms were doing.

Another question asked if users were able to complete all tasks on the list — encouragingly, all evaluators indicated that they were. Additionally,

all users who had previous machine learning experience indicated that they thought the application was a good introduction to the basic concepts it covers. All the MLM students who took part in the evaluation indicated that the teaching aid would have been beneficial to them during the course.

However, not all users completed the task list without encountering at least one unexpected error. A description of some of the issues encountered follows.

Bugs

A number of bugs were encountered by the evaluators. These included:

- If a regression model was trained, and the points cleared, it became impossible to plot the vertices of a polygon. Erroneous error messages were displayed after each mouse click.
- When dragging to define a Gaussian, dragging up and to the right or down and to the left did not work correctly. A straight line was produced instead of an ellipse.
- If the radio button for choosing the active dataset is set to 'Test points' when a new algorithm is set, the setting is not changed back to 'Training points' correctly. This leads to algorithms appearing to break.
- If the user attempts to sample from a polygon without first defining the shape, the program crashes.
- When changing to a new data generation technique, if there is already an ellipse or polygon drawn on the screen, it does not clear until the display is clicked.
- When adding a new regression model, the new tab is not automatically focused.
- The labels showing the misclassification data for KNN and SVM did not update from 0/0.

The majority of these bugs were due to small errors in the logic of the program. These have all been fixed in the delivered version of the teaching aid. The only exception is the last item, regarding the misclassification labels. This error seems to be specific to the machines in the MSc lab where most of the evaluations were carried out. It was not possible to reproduce the error on either of the machines primarily used for development, and the error did

not occurs in evaluations conducted in locations other than the lab. The exact cause of the problem remains unclear, as insufficient time was available in the lab to debug the issue.

Comments and Observations

In addition to highlighting defects in the program, the evaluations raised a number of interesting comments about the design and suggestions for improvement. These are summarised below.

The single most common design issue raised was that users expected the plotted polygon or ellipse to clear after data was generated. The initial design left the objects in place, in case users decided they wanted more points from the previous distribution. In practice, none of the evaluators used the functionality in this way, and many encountered confusion when they attempted to draw a new polygon only to have the existing one extended instead. The feedback with regard to this was so overwhelming that the implementation was changed to match the user suggestion. One evaluator additionally suggested that on a related point, there was no need for the polynomial data generator to have a separate button to randomize the coefficients. Users would only sample from a given polynomial once, so the coefficients should randomize automatically after generation. This change was also implemented.

It was suggested that the help files should be rewritten to show a clearer delineation between the theoretical aspects of an algorithm and the details of its implementation in the teaching aid. This is a valid concern. The help files were written in HTML, since Java offers the ability to parse and display HTML files natively. However, this made it difficult to include mathematical notation, since the default implementation is unable to parse MathML. Thus, the files were kept to plain text as far as possible, which led to some intermingling of the two details. No changes were made as a result of this comment, but an improvement of the help files would be a useful future extension of the project.

On a related note, some users felt that having the help files displayed in a separate window was not optimal, and they would have preferred to have the help shown in one of the free parts of the layout. This is another possible candidate for future work.

A general observation made by a number of evaluators was that the data generation aspects of the GUI were harder to use than the algorithmic parts. In particular, several were unclear about the exact purpose of the train/test dataset selector, and others did not realise immediately how to change the class of new points. The common expectation was that all options related

to data would be in one panel, and so users were unclear as to the reason for the 'Points' subpanel. Enough people raised these concerns to warrant a possible redesign of the interface in the future, perhaps merging the two subpanels into one larger structure.

Several users highlighted the known issue that polynomial data are sometimes sampled from 'uninteresting' functions. This reinforces the need to improve this feature in the future.

One evaluator expressed an interest in an option to turn off the automatic retraining after new points are added, and instead have retraining done only when the appropriate button is pushed. This is one of the few points raised by the evaluation that I actively disagree with. One of the primary ideas behind the teaching aid is that it allows users to explore changes to models in real time. If retraining could be turned off, the state of the model would become out of alignment with what the user might expect given the data. This feature would likely be visually confusing, and this should be avoided.

It was suggested that functionality be added for saving the settings of an algorithm alongside a dataset so that the two could be loaded together as a complete worked example. This is an excellent suggestion, which had not previously been considered. There was insufficient time to add such a feature before the deadline, but adding this would be an excellent future addition to the teaching aid.

The final suggestion for improvement that came from the evaluations was a request for a session history function. This would allow users to move back through the history of algorithms and datasets they have used during the current session, in case they wish to recover an interesting case but forgot to save the data or cannot remember what settings were used for the algorithm. This would take a significant investment of time to implement, but could be useful.

Not all the comments received were criticisms or suggestions for improvement. Encouragingly, a number of users praised the visual design of the teaching aid and expressed a strong belief in its usefulness. In particular, one person commented that the use of shading to show classification output was very instructive, and was much clearer than other diagrams they had seen previously. This is a good indicator that this particular design choice was the right one.

Chapter 8

Conclusion

8.1 Project Status

From the outset, this project had a clear aim: to implement a program to teach basic machine learning concepts. This goal has been achieved. The final version of the teaching aid is a robust, well implemented piece of software, and it meets all of its original requirements. User evaluation has indicated that the program is simple to use, and has substantial merit as a didactic tool.

The project proposal called for a modular, extensible design, which has been achieved through the use of inheritance structures to define the required functionality for algorithms. The logic of implementing an algorithm is separate from the user interface, which means the GUI can easily be upgraded without extensive changes to the domain model. The interface classes exhibit a degree of coupling, but this is very hard to avoid in a graphical user interface. Aside from this, the design makes good use of object orientation to define focused, cohesive classes which are organized in a clear, logical package structure.

While a number of challenges arose during the development, none of these proved insurmountable, and the final product is free from 'deep' errors. There is certainly room for the teaching aid to expand and improve in the future, but as it stands the software offers a solid introduction to a variety of machine learning techniques, and the project can be deemed a success.

8.2 Outstanding Issues

There is one major outstanding issue in the teaching aid as it stands — the look and feel. It was decided that the program would be set to use the

system look and feel, which is different across different operating systems, rather than the consistent but unappealing Java look and feel. The evaluation has highlighted that while the program runs fine on all platforms it has been tested against, there are a number of visual deficiencies on OSX. The Apple implementation of Java uses larger fonts by default than other platforms, so additional scroll bars appear in undesirable places and some labels are cut off because Swing classes do not wrap text automatically. Since a Mac was not available for regular testing, it was impossible to identify and correct all of these issues.

To allow for this, two versions of the executable JAR file for the teaching aid have been submitted. One uses the system look and feel, which works fine on Windows and Linux. The other uses the Java look and feel. While this is not particularly attractive, it at least renders in the same way across all platforms and allows all the information to be seen. Given more development time and access to a Mac, it would be good to correct these GUI errors.

8.3 Areas for Further Work

There are a number of potential areas for further work on the teaching aid. Some of these have already been discussed, but the suggestions are collected here for easy reference.

First, and most obviously, more algorithms could be added. Indeed, one of the reasons care was taken to produce a modular design was so that someone else could extend the project in the future to increase the range of algorithms it covers. There are a huge number of available machine learning techniques, many of which can benefit from visualization, so there is no shortage of potential work in this area.

As well as new algorithms, the data generation capabilities of the teaching aid could also be extended. While the polygon sampling is very flexible and can be used to approximate other sampling techniques, there are potentially other distributions which could be implemented. Also, there is scope for the improvement of the polynomial generator. A better method of scaling the coefficients so that interesting data are produced could be found, for example. Additionally, it could be possible to add nonlinear terms in addition to the polynomial itself. Since the teaching aid can fit sine curves and exponentials, it seems reasonable that it would also be able to generate data according to such a model. This would require some redesign of the interface for polynomial sampling.

The fact that the display is constrained to show a particular interval is something of a limiting factor. A useful extension would be to implement

zooming on the plotting space, so that data can be plotted over a wider range of values. It would also allow more of a given regression model to be seen, so that users can gain an intuition of how the predictive performance extends outside the range of the data.

The help subsystem is still fairly basic. The quality of the files could be improved by adding a MathML or LaTeXparser, to allow easy typesetting of mathematics in the files. Also, a redesign of the interface placing the help for an algorithm on screen whenever the algorithm is loaded could be useful. It is currently slightly unwieldy to switch back and forth between the teaching aid and the help window when additional information is required. Indeed, an overhaul of the interface in general could be beneficial. The current structure of DataPanel, in particular, caused problems for some users during evaluations, so a review of that is in order.

Finally, some evaluators had novel ideas for improving the application in the future. A history feature, though it would probably take a lot of work, would be an excellent enhancement and offer a lot of useful functionality for end users. Similarly, the ability to save algorithm settings together with datasets and load them as worked examples would be a good improvement.

Appendix A

Statement of Requirements

A.1 Functional Requirements

Data generation:

- It must be possible to plot a single data point.
- It must be possible to generate a number of points from a given polynomial or distribution.
- It must be possible to remove individual points.
- It must be possible to assign the label on a point, for algorithms which require labels.
- It must be possible to distinguish between training and testing points when plotting.

Algorithms

- It must be possible to select an algorithm to run on the plotted data.
- It must be possible to configure any options for the selected algorithm.
- It must be possible to run an algorithm and plot the resulting model.
- It should be possible to view multiple models over the same data set simultaneously for regression algorithms.

Cross Validation

- It must be possible to compute the performance of an algorithm using cross validation over the training set (where appropriate performance measures exist).

- It could be possible to compare the performance of different models using cross validation.

Data storage

- It should be possible to save the training and test sets so they can be recovered later.
- It would be useful to be able to export an image of the current plot.

A.2 Non-functional Requirements

- The system should be developed in Java.
- The interface should be designed assuming a basic knowledge of machine learning
- The interface should be designed to be intuitive and easily usable.
- Algorithm implementations should be as efficient as possible — no algorithm should take longer than a few seconds to train on a set of less than 1000 points.

A.3 Use Cases

From the requirements above, a number of candidate use cases were extracted. A diagram showing these is included below.

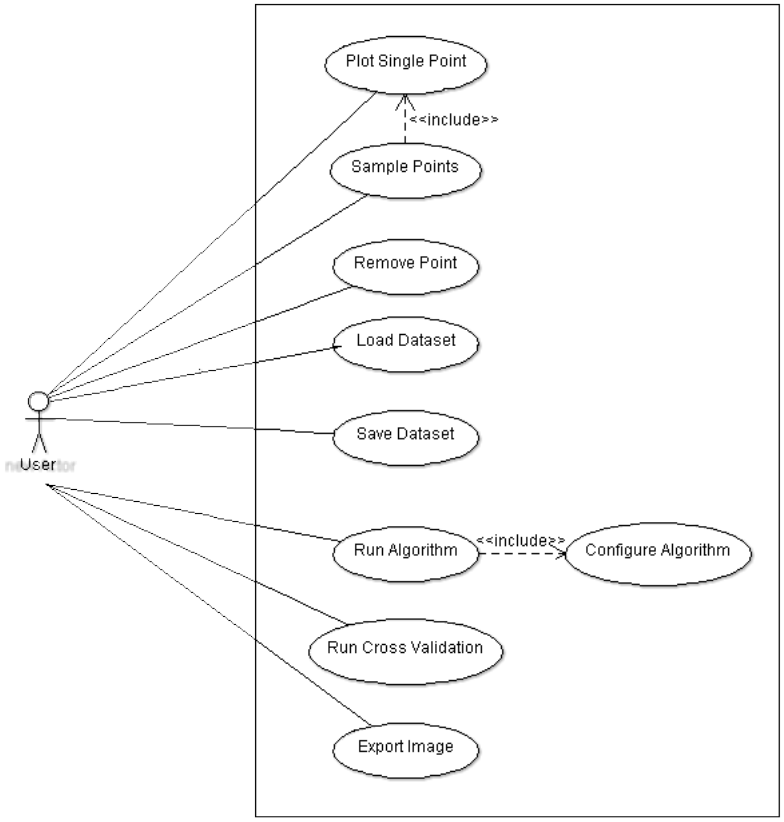


Figure A.1: Use Case Diagram

Appendix B

Design Documents

This section contains a package diagram showing the overall structure of the teaching aid, followed by a collection of class diagrams showing the individual contents of the packages. For clarity, instance variables and methods are not shown on these diagrams. To obtain this information, see the Javadoc on the accompanying CD.

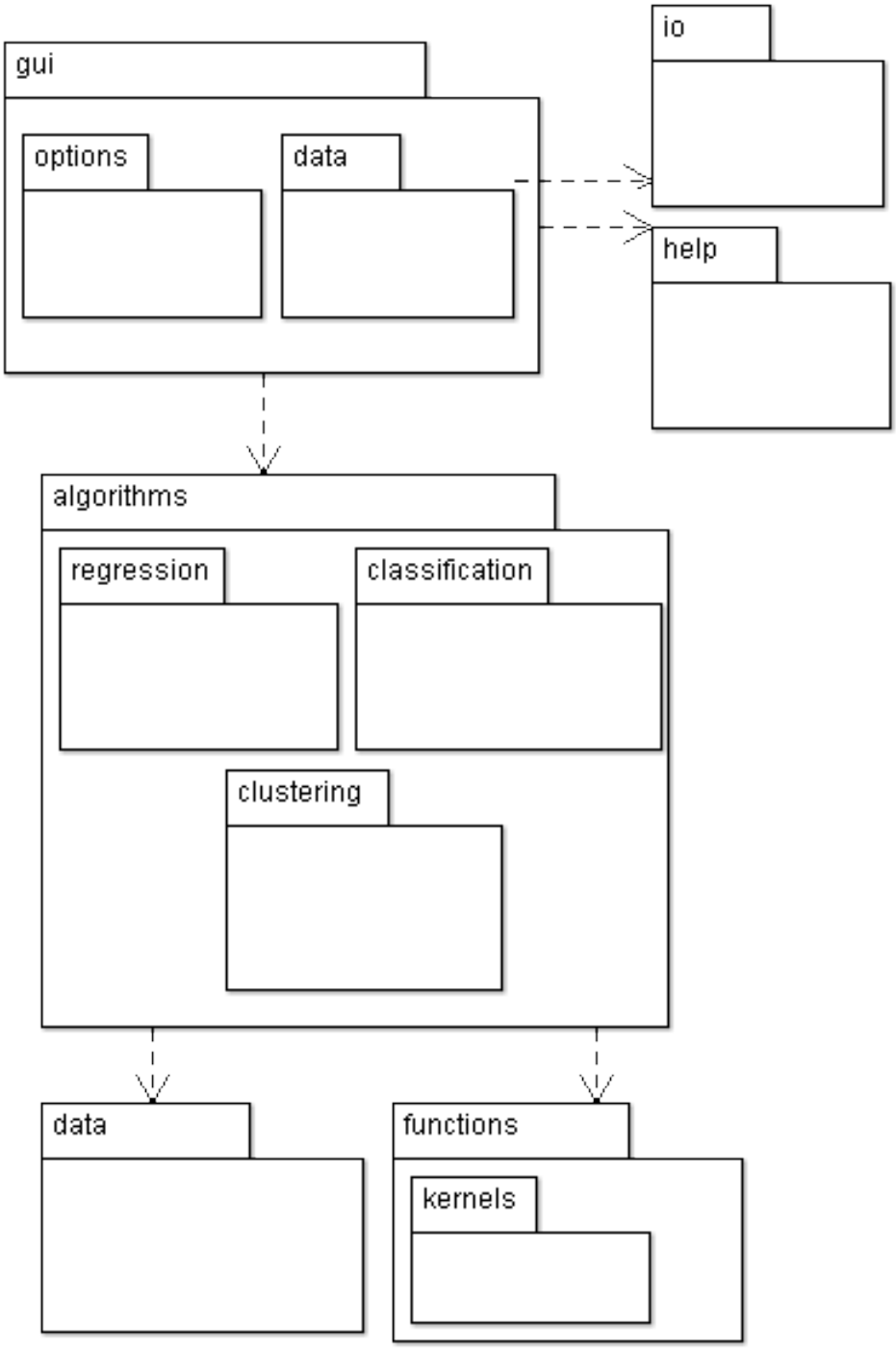


Figure B.1: The overall package structure of the teaching aid

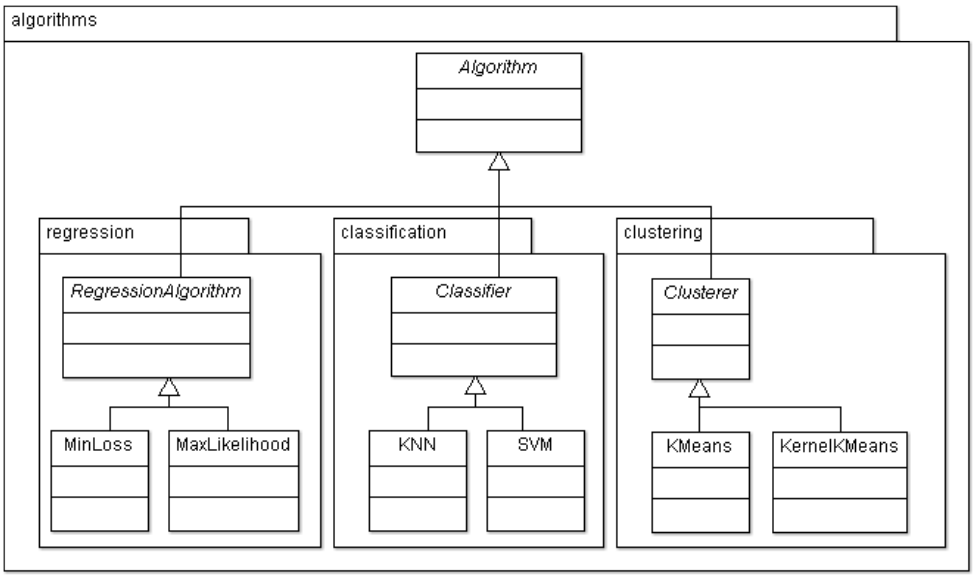


Figure B.2: The `algorithms` package

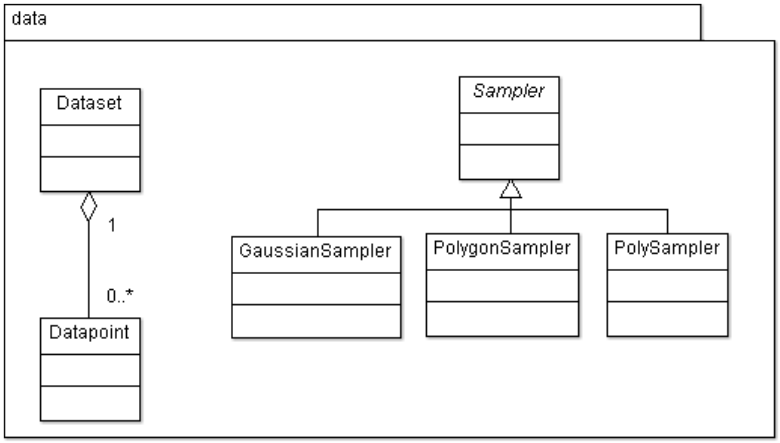


Figure B.3: The `data` package

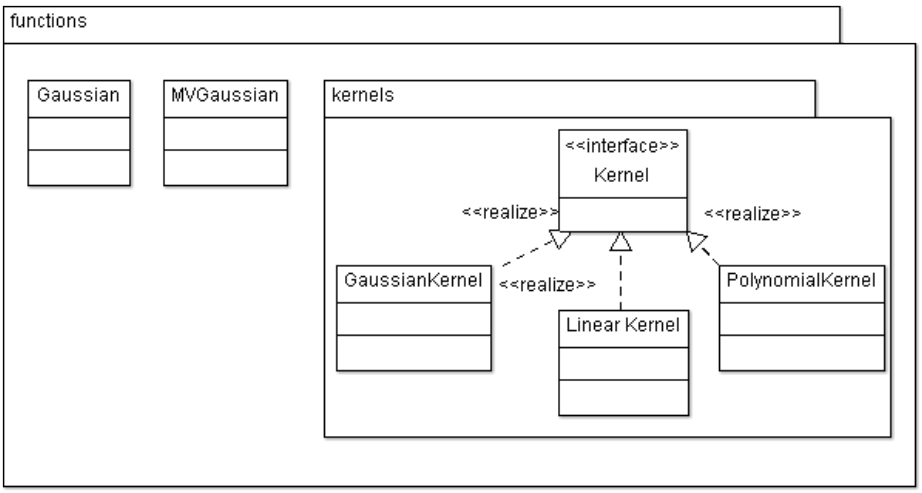


Figure B.4: The `functions` package

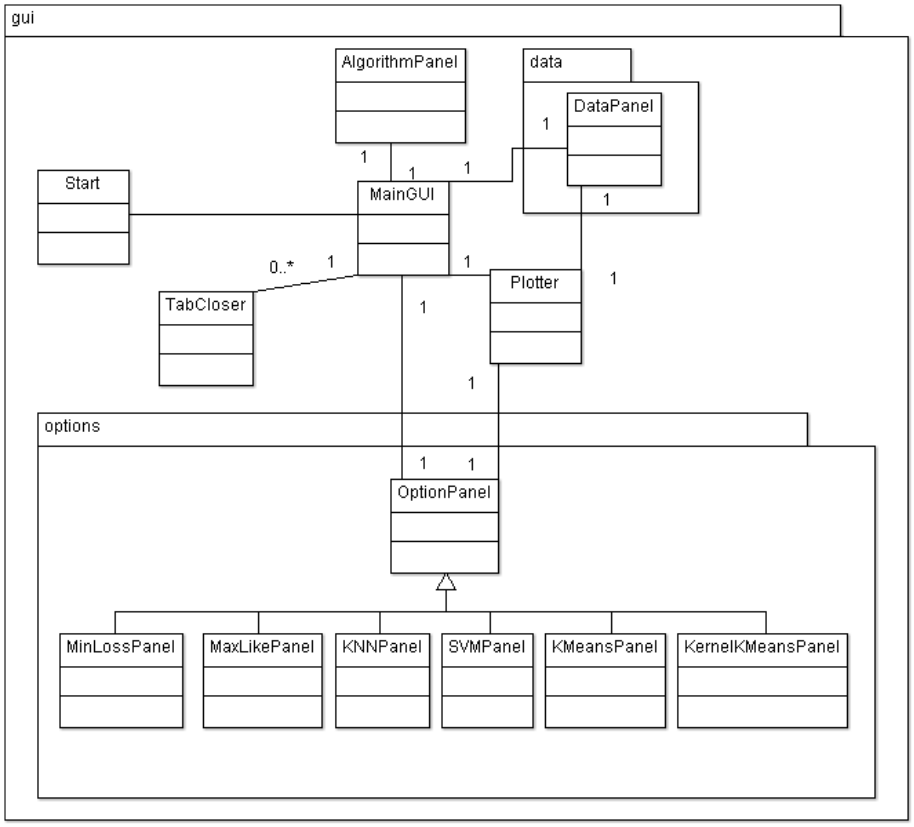


Figure B.5: The `gui` package

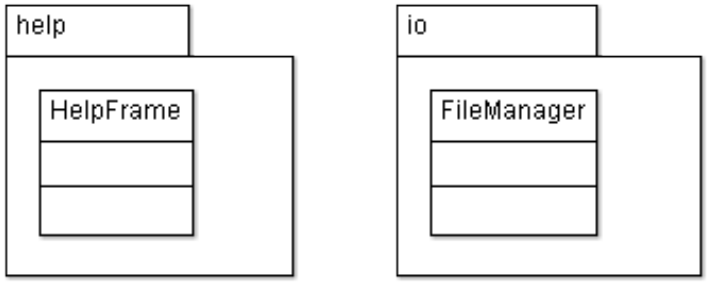


Figure B.6: The `help` and `io` packages

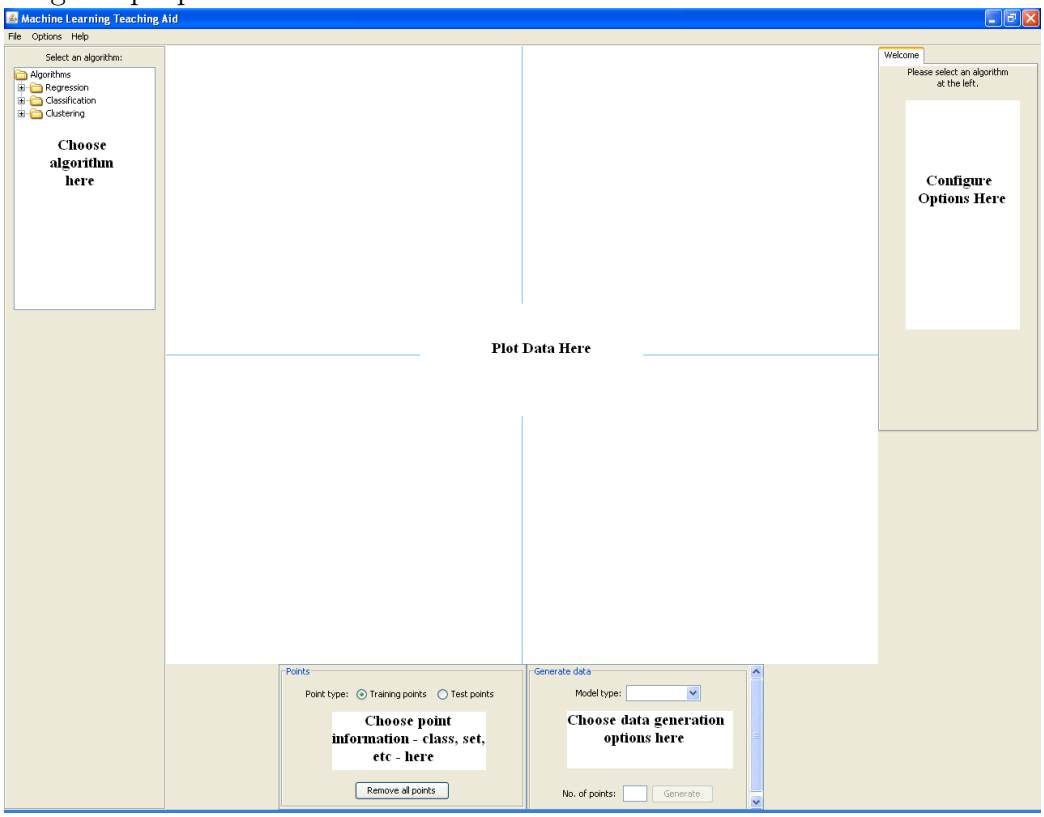
Appendix C

Evaluation Documents

This appendix contains the documents which were presented to participants of the user evaluation for the teaching aid. These include a basic information document detailing the operation of the teaching aid, a task list, and a questionnaire. The original documents were created using Microsoft Word, but for this document have been rewritten in LaTeX.

C.1 Basic Information Document

The basic layout of the teaching aid application is shown below, with notes showing the purpose of each section of the interface.



The basic steps to use the program are as follows:

1. Choose an algorithm in the left hand panel
2. Generate some data — either click on the plot or use the options in the bottom panel
3. Configure the options for your chosen algorithm in the right hand panel, then click 'Train Model'
4. Individual points can be clicked on to remove them. Add or remove points to see how the output of an algorithm changes

Use these basic instructions to work through the task list on the following pages. If you need more information about an algorithm or feature, try the help files. These are available through the menu by choosing Help -> Show Help.

C.2 Task List

Minimized Loss

- Plot some data by clicking on the display
- Choose some terms at the right hand side and train a model
- Run cross validation on your model

Maximum Likelihood

- Choose the maximum likelihood algorithm, then generate some data from a polynomial
- Choose some terms and train a model
- Add a second model, and change its line colour
- Click the Partition Data button a few times and observe the effect

KNN

- Generate data from a few Gaussians use several different classes for the points.
- Train a model with K (Number of neighbours) = 1
- Try changing the value of K to see the effect

SVM

- Use the polygon option to generate some data make sure to add both red and blue points.
- Train the model using a linear kernel
- Change to Raw colouring mode
- Try adjusting the value of C
- Change to a different kernel
- Change the value of the kernel parameter
- Run cross validation

K Means

- Plot some data using any method you want
- Click the Start Clustering / Next Iteration button until the algorithm converges
- Reset the algorithm, then change the value of K and start again.

Kernel K Means

- Load the Concentric Rings example dataset
- Use a linear kernel to cluster the data note that it fails to capture the structure
- Use a different kernel to cluster the data change the parameters until you are able to capture the structure

Other Features

- If you have not used the help system up to this point, please read a few of the entries now.
- After training a model, try exporting an image of the plot.
- Try saving and reloading a dataset.

C.3 Questionnaire

1. How clear was the design of the teaching aid?
   - Excellent — all tasks were easy and intuitive to perform
   - Very good — most tasks were easy, but some required additionaly help
   - Okay — some tasks were easy, some were difficult
   - Poor — many tasks were hard to perform
   - Very poor — the application was extremely difficult to use

   Specific comments on the design, if any?
2. Were you able to complete all the tasks?

- Yes
- No

(a) If no, which tasks were you unable to complete?

3. How useful were the help files?

   - Very useful — the help text aided my understanding of one or more features
   - Useful
   - Not useful
   - Useless — the help text failed to clarify one or more problems

   Specific comments on the help files, if any?

4. Did you encounter any unexpected errors while using the program?

   - Yes
   - No

   (a) If yes, please describe these errors.

5. Did you take the Machine Learning (M) course as part of your studies, or did you have experience of machine learning before undertaking this evaluation?

   - Yes — MLM student
   - Yes — other previous experience
   - No

   (a) If yes, do you think the teaching aid offers a useful introduction to the concepts it covers?

      - Yes
      - No

   (b) If you took the course, would the teaching aid have been useful to you in lectures and lab sessions?

      - Yes
      - No

6. Suggestions for improvement / Any other comments

Appendix D

Dataset file format

Datasets are saved to file with the .dset extension. The file format is plain text, with a very basic format.
The first line of the file contains '#Dataset' and nothing else.
The second line of the file contains '#size:' followed by the number of points in the dataset.
The third line of the file contains '#Num classes:' followed by the number of distinct labels on the points of the dataset.
Each subsequent line of the file represents 1 datapoint. Each line has three double values, separated by spaces. In order, these are the x coordinate, the y coordinate, and the label of the point.

Appendix E

Contents of Accompanying CD

The CD accompanying this dissertation contains a number of folders. A summary of the contents of these is presented below.

Reports:
: This folder holds a PDF copy of this dissertation, along with a copy of the original project proposal.

Documentation:
: This folder contains the Javadoc output generated from the teaching aid code.

Source Code:
: This folder contains the Java source code for the teaching aid application.

Executables:
: This folder contains two JAR files. `teachingaidfinal.jar` is an executable for the teaching aid, using the system look and feel.
`teachingaidfinalJ.jar` is the same executable, but using the Java look and feel.

Diagrams:
: This folder contains full size versions of all the diagrams included in this report.

Bibliography

[1] T. Abeel, Y.V. de Peer and Y. Saeys, *Java-ML: A Machine Learning Library*, Journal of Machine Learning Research, 2009, 10, 931-934

[2] Apache Software Foundation, *Apache Commons Math*, Available at: `http://commons.apache.org/math/`

[3] CERN, *Colt Project*, Available at: `http://acs.lbl.gov/~hoschek/colt/`

[4] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines*, 2001, Available at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[5] Dai Clegg and Richard Barker, *Case Method Fast-Track: A RAD Approach*, Addision-Wesley, 1994

[6] Jeff De Luca, *Feature Driven Development*, Available at: `http://www.nebulon.com/fdd/index.html`

[7] Donald W. Denbo, *Scientific Graphics Toolkit*, NOAA/PMEL/EPIC Group, Available at: `http://www.epic.noaa.gov/java/sgt/`

[8] End of the World Productions, *Machine Learning Applets*, Available at: `www.theparticle.com/applets/ml/index.html`

[9] Andreas Geiger, *Gaussian Processes for Machine Learning Java Applet*, Available at: `www.rainsoft.de/projects/gausspro.html`

[10] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin, *Bayesian Data Analysis*, CRC Press, 2003

[11] Donald Goldfarb and Ashok Udhawdas Idnani, *A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs*, Mathematical Programming, 1983, Vol. 27, pp. 1-33

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten, *The WEKA Data Mining Software: An Update*, SIGKDD Explorations, 2009, Volume 11, Issue 1

[13] Kim Hansen, *JavaOctave*, Available at: `http://kenai.com/projects/javaoctave/pages/Home`

[14] Hang Tong Lau, *A Java Library of Graph Algorithms and Optimization*, CRC Press, 2007

[15] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, *Improvements to Platts SMO algorithm for SVM classifier design*, Technical Report CD-99-14, National University of Singapore, Available at: `http://guppy.mpe.nus.edu.sg/~mpessk`

[16] The Mathworks Inc., *MATLAB*, Natick, Massachusetts, 1984-2010

[17] The Mathworks Inc. and the National Institute of Standards and Technology, *JAMA*, Available at: `http://math.nist.gov/javanumerics/jama/`

[18] Object Refinery Limited, *JFreeChart*, Available at: `http://www.jfree.org/jfreechart/`

[19] Octave Development Community, *GNU Octave*, Available at: `www.octave.org`

[20] John C. Platt, *Fast training of support vector machines using sequential minimal optimization*, Advances in Kernel Methods: Support Vector Machines, MIT Press, December 1998

[21] Trygve Reenskaug, *Thing-Model-View Editor: An Example From a Planning System*, Xerox PARC Technical Note, May 1979

[22] Simon Rogers and Mark Girolami, *A First Course in Machine Learning*, CRC Press, 2010

[23] Junjie Sun and Leigh Tesfatsion, *QuadProgJ*, 2006, Available at: `http://econ2.econ.iastate.edu/tesfatsi/DCOPFJHome.htm`