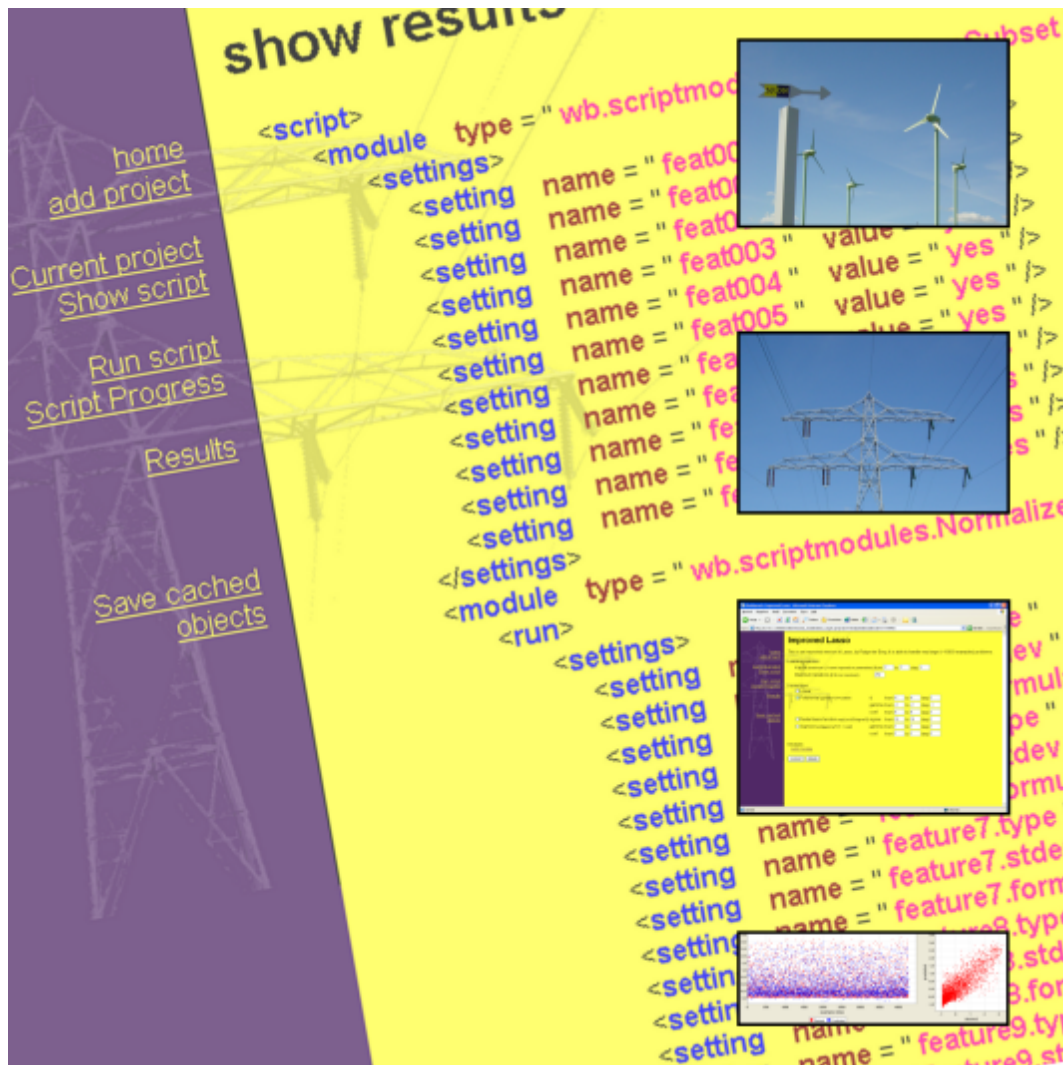


Workbench for Machine Learning Techniques



Master's thesis of Floris A. Ouwendijk

Delft University of Technology
Faculty of Information Technology and Systems
June 2003

 TU Delft

 NUON

Graduation committee

Prof. dr. H. Koppelaar
Prof. dr. ir. E.J.H. Kerckhoffs
Dr. K. van der Meer

Ir. R. ter Borg (NUON)
Ir. M.A. van den Berg (NUON)

Ouwendijk, Floris Anthonie (floris@ouwendijk.nl)

Master's thesis, June 2003
Workbench for Machine Learning Techniques

Delft University of Technology, The Netherlands
Faculty of Information Technology and Systems
Knowledge Based Systems group

Keywords: machine learning, workbench, distributed computing, modules, stack-able architecture

Acknowledgements

I would like the following people for their support and help during this project.

Henk Koppelaar, for accepting me as his thesis student and helping me with the choice of this project. The wise counsels and tips were useful during this project, and I am sure they will prove useful in the rest of my career.

Rutger ter Borg, for the direct guidance. The tip to use LaTeX (and some help during its use) has spared me from many text editor problems, and allowed me to focus on the text. Your critical comments on the text were helpfull in improving my style.

Thijs van den Berg, for allowing me to do this work at NUON. The nice words were a real stimulation.

My friends for the parties and good talks. I'll try to reduce the use of the sentences 'kan niet', 'geen zin' and 'geen tijd'.

Richard, Jeffrey en Willem for reading this thesis. Although not all the comments could be embedded, they sure did improve the quality of this work.

'Serious Sam', for the distraction in the final weeks before the completion of this work.

Mom and dad, for giving birth to this life, being there for me and your stimulation. I hope we will share many more beautiful moments together. I dedicate this work to the two of you.

Preface

This thesis represents the finalisation of five years study at the Delft University of Technology, faculty of Information Technology and Systems.

For this project I have created a tool that supports research into forecasting problems. This tool interfaces with a variety of machine learning tools that are available on the Internet. By supplying the user with extra modules and a stackable architecture, I hope to ease common problems that occur when one needs to create programs for prediction problems.

This project started the moment Henk Koppelaar suggested that I should go to NUON in Amsterdam to talk with two former students of him, one of whom was working on his PhD thesis. After a talk with these two enthusiastic persons, the decision was made quickly that I would support Rutger with his research by creating a workbench that would allow him to compare different machine learning techniques.

The first two months I have studied the theory of support vector machines, and made a start with the thesis. The months after that were a mix of design and implementation. The chosen technologies allowed the workbench to be built in an iterative process of implementing parts while designing the next parts. The base could be established quickly, and additions improved the base.

During the final months, the workbench was used more and more for real problems, instead of the toy examples that were used up to that point. Each time the workbench managed to attack a larger problem, an even larger problem was presented, which brought new bottlenecks to light. This process pressed the stability further and further.

I can look back at an intense but enjoyable period, working with bright people from whom I have learned a lot.

Abstract

Prediction of various values is a hot item for many companies. If values such as the volume of energy consumption, or the number of sales of a product can be predicted with good accuracy, then the efficiency of the company can be increased. Especially in this period of recession, efficiency can improve the chance of survival.

There is a huge variety of techniques to produce predictions. The optimal technique is ultimately determined by the problem at hand. To fine-tune the solution, a lot of smaller choices have to be made to find the best parameters.

The goal of this project is to offer researchers a computer program that can be used to optimize the techniques that create the predictions. This program should allow them to make changes in the parameters fast, create big tests, and visualise the results of these tests.

The program is implemented in Java as a web service. This way, multiple users can work with the system at the same time, using a web browser. Tests are distributed over multiple computers (by using JINI/JavaSpaces), to reduce the time one has to wait for the results.

The result is a program that can be used much like a web site. The user can insert the problem as a project and then create an outline for the tests. This outline is created by selecting building blocks that modify the data. These blocks can be stacked to create different results.

The results of different tests are gathered automatically, and the user can look at this data from different angles. Using these views of the results, the user can continue the search for a better prediction.

The use of this program offers multiple advantages. The first is that the creation of tests is easier. Instead of manually making changes or writing tools to do so, the user adds a module to the structure. This also maintains the complete picture of the test. The second advantage is that the system uses multiple computers to perform the time-consuming steps. This increases the number of tests that can be done in the same amount of time. The third advantage is that the program is extendible. New modifications or prediction techniques can be added fairly easily, and these new blocks can cooperate with all existing parts to create new predictions.

Contents

1	Introduction	1
1.1	Problem	2
1.2	Proposed solution	2
1.3	Related work	2
1.4	Document outline	3
2	Requirements	5
2.1	Flexibility	5
2.2	Insight	5
2.3	Multiple tests	6
2.4	Hypothesis	6
2.5	Different models	6
2.6	Integration	6
3	Machine learning techniques	7
3.1	k -Nearest neighbours	7
3.1.1	Theory	7
3.1.2	Improvements	10
3.1.3	Implications for the workbench	11
3.2	Neural networks	11
3.2.1	Theory	11
3.2.2	Improvements	13
3.2.3	Implications for the workbench	13
3.3	Support vector machines	14
3.3.1	Theory	14
3.3.2	Improvements	19
3.3.3	Implications for the workbench	19
3.4	Multiclass problems	20
3.4.1	Theory	20
3.4.2	Implications for the workbench	22
4	Design	23
4.1	Conceptual model	23
4.1.1	Example	25
4.2	Language and libraries	27
4.3	Structure	29
4.3.1	Datastore	30
4.3.2	Project	30
4.3.3	Dataset	31
4.3.4	Script	32
4.3.5	Results	32

4.3.6	Modules	34
4.3.7	Coupling	37
4.4	Implementation of requirements	38
5	Implementation	41
5.1	Environment	41
5.2	Framework	42
5.2.1	Datastore	42
5.2.2	Webformstore	45
5.2.3	Wrapper distribution	45
5.2.4	User-interface module enhancements	47
5.3	Modules	48
5.3.1	Basic module structure	49
5.3.2	Dataset modules	50
5.3.3	Script modules	54
5.3.4	Visualization modules	64
6	Evaluation	71
6.1	Cleveland heart disease	71
6.1.1	Dataset description	71
6.1.2	Previous results	72
6.1.3	Test	72
6.1.4	Conclusion	79
6.2	Iris	81
6.2.1	Dataset description	81
6.2.2	Previous results	81
6.2.3	Test	81
6.2.4	Conclusion	83
6.3	NUON wind prediction	84
6.3.1	Dataset description	84
6.3.2	Related work	84
6.3.3	Test	85
6.4	Conclusion	88
7	Conclusions	91
8	Future work	93
9	Glossary	95

The only thing that will redeem mankind is cooperation.

Bertrand Russell (1872 - 1970)

1

Introduction

For many companies, the ability to forecast events, demands and other strategic variables can give them a competitive edge. It is therefore no surprise that a lot of work is done on tasks such as building mathematical models, improving prediction algorithms, combining strategies and other related tasks.

Existing tools that are available can increase the depth of research as well as the speed of embedding of techniques. When research is performed it is good practice to compare ideas and techniques with multiple other techniques. If the tools for these other techniques are easy to obtain and use, the comparison becomes easier. The same goes for the embedding of a technique in a process.

The process of using tools for testing can be characterized as follows:

- Obtain the dataset. This step is usually only performed once for a certain problem.
- Enhance the dataset. The dataset might lack some data, or the data should be modified to increase the prediction ability.
- Transform the dataset into a format that the tool can use. The details for the dataset format of the tools differ. Some modifications are easy, some others require a restructuring of the data.
- Use the tool. This step can be repeated several times with different parameters to obtain the best results.
- Gather results. To use the results, they have to be extracted from the output of the tool.
- Use the results. This brings the results together, possible from different executions of the tool, or different tools. A statement can then be made about the results.

1.1 Problem

The implementation of the above process is often more complicated. The needed tool has to be obtained, and the process contains a lot of conversion steps.

The conversions of data occur multiple times. The original dataset has to be modified to allow enhancement. If tools are used for these enhancements these might also require conversions. The dataset needs to be converted to the format that the tool can use and the output of the tool has to be converted to a format so that it can be used.

Most of the time these conversions will be performed by hand, or by writing small tools to perform the specific conversions. A related problem occurs with the creation of derivatives of the dataset. It is common to create training- and test-sets for the whole test when comparing techniques or parameter-sets, because variations in these sets will lead to different results. These sets can be created using different tools each taking a step in the conversion process.

Conversions of the data format are needed between each of these steps to transform the output from one tool into the input for the next tool. After testing, the results are again in different formats, which need to be collected and converted. Therefore massive tests require lots of conversion steps and writing of additional code.

All these conversions and feeding output from one tool into another takes time. In addition, the fine-grained working process where each step is specified by the user, makes the overall process less clear.

1.2 Proposed solution

The essence of the problem is that many conversions and handwork are needed to perform tests. Part of this problem can be solved having a standard set of conversions to the required dataset, and a set of conversions that extract the data from the tool in a common format.

The use of a program that interconnects all the tools would facilitate the creation of tests even further. Conversion details can be shielded from the users, who can therefore concentrate on the creation of tests.

This project is an attempt to create a basis for researchers and students to ease the use of each others work, so that new models can easily be set off against other models, different models can be used on problems with ease, and the creation of new implementations is facilitated. This basis is given the form of a workbench.

A workbench is a program that can be used to explore, transform and process data in a variety of related ways. It acts as a container for data to which tools can be applied. The order of application of the tools determines the shape of the result.

The primary objectives of the workbench can be defined as follows:
The workbench should provide the user with a flexible tool that is able to give insight in the problem at hand, perform lots of tests in a short period of time to test hypotheses, and make it easier to do so with different machine learning models.

1.3 Related work

There is a wide variety of machine learning related programs available on the Internet. These programs can be divided in three categories, differing in their level of

interaction and ability to perform other tasks.

The first category is the set of basic tools. These tools come in a wide variety. Some of the tools are pre-compiled and runnable from the command-line. Their inputs are provided through files and command-line options. Other tools are provided as sources or libraries that are to be included in one's project. The common factor is that they are only aimed at training on a train set using a certain technique, and then testing the performance on another set.

The second category goes a step further. These tools provide a user-interface for setting up the options. Some also provide functions to modify the dataset. However, they are limited to a single machine learning technique.

The third category is that of programs that integrate different tools or machine learning techniques and augment that with other tools and techniques to make changes to the dataset and visualize the results. Programs in the third category come close to the objectives that were stated in the previous section. A distinction can be made within this category. There is a group of programs like Matlab and S-plus, where the user interacts by typing commands. These programs are more like interactive programming languages with a lot of functions that can be used for a variety of tasks. The other group provides the user with a graphical user-interface to perform operations visually.

Weka is the only tool that claims to be a machine learning workbench. The core of Weka is a Java-based library, filled with different machine-learning tools [Witten and Frank, 2000]. It also has capabilities to transform the dataset. In the latest versions a graphical user-interface is supplied, which facilitates the use of Weka. This interface allows the user to transform the dataset, run tests and visualize relations between input variables. However, the interface is still elementary, only one test can be run at a time, and there are no provisions to compare results of different tests.

1.4 Document outline

The objective that was stated for the workbench is extended in chapter 2, by investigating the requirements that the workbench should implement. Some practical requirements are added in chapter 3, by looking at some major machine learning techniques. A design that matches the requirements is created in chapter 4, followed by chapter 5 where the design is implemented, and the details and changes made are discussed. Chapter 6 evaluates the use of the workbench, by showing the implementation of some tests. The results of the implementation and tests are concluded in chapter 7, which reviews the end product. Chapter 8 gives suggestions for the road ahead. A glossary (chapter 9) has been included to clarify terms and indicate the definitions that are used in this thesis.

2

Requirements

The objectives for the workbench were given in the introduction. This chapter will explain and augment those objectives, to come to a list of requirements that are deemed necessary for the workbench to succeed.

2.1 Flexibility

The workbench should be flexible. Many of the existing tools are too rigid because they are aimed at a certain problem or domain. When is a tool flexible enough? The only true flexible tool is a programming language. But using a programming language to write a new tool for each job again and again is clearly not the right way either. A flexible tool is therefore a tool that is usable and extendible at multiple levels. When pieces exist they should be easy to re-use, and if they do not exist, it should be possible to build them from pieces at a lower level, or ultimately by programming a new piece. This should all be possible with a minimum of modifications to existing parts.

By combining pieces at lower levels, new tools at higher levels can be assembled quickly. This is reminiscent of programming languages. Functions or objects can be tied together in a new function or object, thus creating a more powerful one. The workbench has the advantage that parts of this assembling process can be done visually.

2.2 Insight

Insight is another term coined in the objective. Providing insight is a very broad subject. When one obtains a dataset, the basic properties of the features are usually known. In many cases the relations between features are not that clear at the start. To gain insight in these patterns, one should be able to plot sets of features, or be able to look at reports giving summaries of the data. This allows the user to get a feeling for the data, how it behaves, and what the peculiarities are. Insight is also needed to adjust parameters of learning techniques. By visualizing the

results of several tests, humans can easily extrapolate the results to come up with new guesses for improvements. Learning the strengths and weaknesses of different tools is hard to do with only raw numbers on their performance. Here visualization will play an important role too.

2.3 Multiple tests

The ability to perform multiple tests, and gather their results automatically, would be great improvement over other tools. Most tools only allow the user to start one test, and then wait for the results. The user then needs to gather the results and bring them together with the results of other tests. Only after all these steps, comparisons can be made. The workbench should allow users to select a large number of tests, and then execute all these tests.

Distributing the tests over multiple processors or hosts reduces the time that the user has to wait, so that the user can quickly review the results and schedule some additional tests.

2.4 Hypothesis

The use of machine learning techniques is usually not performed in a void. There are always expectations or questions that need to be tested. By providing the user with tools to use the techniques more easily, it becomes easier to test a hypothesis by checking if results confirm the hypothesis.

2.5 Different models

With today's abundance of tools, it is a huge task to test different machine learning techniques on a problem. This is complicated even more by their variety of modifications and parameters. It requires lots of dataset conversions and knowledge of how to use the tools. The workbench can facilitate this problem by supplying a common interface to these tools. The user should then be able to do tests with different models and use the best for the problem at hand.

2.6 Integration

The workbench should also enable integration of existing tools in a small amount of time. There is a wealth of good and proven tools in the market. Requesting all tools to be changed to work with the workbench would create a conflict from the point of view that re-implementing is a waste of time. It is also expected that many of the greatest additions to the workbench will come from tools that were created without the workbench in mind. The workbench should therefore be like transparent glue that streamlines the use of all those different tools, with more time for using them as the jackpot.

3

Machine learning techniques

The workbench should be developed with some machine learning techniques in mind as an addition to a more abstract idea of what a workbench should facilitate. The major reason behind the choice for multiple techniques in this chapter stems from the fact that every technique has some additional peculiarities. And even the standard issues can easily be overlooked. By studying the techniques in depth in advance, the abstract idea of the workbench can be adapted and enlivened with cases derived directly from the techniques.

The chosen techniques are k -nearest neighbours, neural networks and support vector machines. k -Nearest neighbours was chosen as a baseline. This very easy technique can still perform very well in some situations, and it would be nice to compare it to both neural networks as well as support vector machines. Neural networks are a natural choice as they are used for a plethora of fields, from recognition to prediction. Support vector machines is a new and promising technique. They allow for much faster, and in many cases also better, recognition and regression when compared to neural networks, and they are not hampered by some of the problems that neural networks have (like depending on the initial weights, and getting stuck in local minima). The following sections will delve deeper into the theory behind these techniques.

3.1 k -Nearest neighbours

The k -nearest neighbours (kNN) algorithm is the simplest one of these three techniques, and it is mainly used for classification [Cost and Salzberg, 1993]. It stems from the idea that points that 'look' approximately the same, will probably be classified the same. By incorporating not just the nearest neighbour, but the k nearest points, the algorithm gets more robust to errors in the examples.

3.1.1 Theory

Because 'looking the same' is not really objective, and visualizing data is hard to do in more than three or four dimensions, one uses a metric. A metric assigns a single

number to the relation between two points. Metrics are, by definition, subjected to several constraints,

$$d(x, y) \geq 0$$

$$d(x, y) = d(y, x)$$

$$d(x, z) \leq d(x, y) + d(y, z)$$

Most implementations of the k -nearest neighbours algorithm use the Euclidian distance function as their metric, where the differences in each dimension are squared and summed. This Euclidian distance function belongs to the class of functions known as vector p -norm, which is defined as

$$d_p(x, y) = \|x - y\|_p \quad \text{with } \|x\|_p = \left(\sum_{i=1}^N |x_i|^p \right)^{1/p}$$

These functions differ in their norm: Euclidian distance uses norm 2. Other examples are $p = 1$, which is known as the Manhattan distance, and $p = \infty$ which is the maximum absolute difference, also known as the Chebychev distance. The absolute values have to be used to satisfy the constraint that the resulting value has to be non-negative. The choice for the order depends on the problem at hand. For regression problems where one tries to minimize the difference between examples and target function, a common rule is that the higher the norm the more sensitive the function is to outliers. In the case of kNN, it follows that the higher the norm, the more sensitive the system becomes for differences in scale of features. Features that show the largest (absolute) differences will dominate the value of the metric. For the rest of this text the Euclidian distance will be used as it is the most common metric, and has some nice features: it is easy to calculate, it is easy to differentiate and it will return a value with the same unit as the original, which one can easily relate to: the length of a straight line from A to B.

For example, if one has a set of points x_{ik} where the subscripts i and k denote respectively the number of the example and the index in this vector, and $y_i \in \{-1, +1\}$ is the class that example i belongs to. The metric is defined as

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^N (x_{ik} - x_{jk})^2}$$

If one gets an unknown point x the distance to all examples x_i is calculated, and (if the j^{th} example has the smallest distance to the unknown point) the point is classified just like y_j .

There are some problems with this approach. If the dimensions of a point are different features with different units, the root of the distance has no meaning. But more important is that the scales can be different. If one dimension can get really large compared to other dimensions, the difference in this component will strongly influence the distance, and throw off the solution. As mentioned above, one could use a different metric (or just a different norm) to restrict the influence of certain dimensions. Another way to solve this problem is to scale the features, so the dimensions become comparable. It is even possible to use an arbitrary function for this scaling and use non-linear mappings to transform the problem into a problem that is easier to solve (because the mapped points might be linearly separable, or form distinct clusters). A frequently used option is to normalize the values to have

mean 0 and standard deviation 1. This brings the scales of the dimensions closer together, and removes much of the problems with scale differences. One is not restricted to mappings of one feature to a new one. It is even possible to map a set of features onto a new set. The section on support vector machines will go deeper into this.

The use of mappings to convert the original feature-space onto another, simpler feature-space is a good way to make many machine-learning techniques more usable. The technique need not be adapted to each specific problem, and the mapping can be used for several techniques. For the kNN this means that no special metrics have to be devised, but that one can use a specific mapping and then apply one of the common metrics. An added advantage is that mappings are more intuitive for most problems, or at least easier to visualize. A disadvantage is that optimisation becomes harder. Training and testing become essentially two-step processes. First one applies the mapping, and then one trains or tests the technique. Therefore optimisation of the parameters of the mapping involves constantly retraining the underlying technique.

Another problem is that stray, or misclassified points can have a great influence on the solution (figure 3.1). To solve this, one should not only look for the point with the minimum distance, but the k -points with minimum distance. If the points are from different classes, one can use a majority vote to determine the class of the unknown point, or weigh all classes of the points according to their metric, and take the average (and round the average to either -1 or $+1$). There are two special cases. If $k = 1$ then the nearest neighbour method already discussed above is obtained. The other special case occurs if k equals the number of points in the dataset. In that case there are two options. The first is to take a majority vote, and always get the same result: the class that occurred the most. The second is to weigh all points according to their distance. This results in very smooth boundaries between classes. Usually k is used to trade-off fault-tolerance and smoothing. To allow for one error, one should use $k = 3$. Small clusters (assuming that these small-clusters are not the result of random errors), can be removed if k is large, and a larger cluster of the opposite class is near.

Figure 3.1 shows the same distribution for different values for k . A random set of points was chosen and labelled (plotted as black crosses). The labelling was done according to the line $y = x^3$. Points above this line were labelled as class 'blue', points below this line were labelled as 'red'. After 'training' the kNN on this dataset, the resulting labels were predicted for a uniform subset of the feature space with $x \in [-1, 1], y \in [-1, 1]$. These points were plotted as red or blue squares. This results in two areas coloured blue and red, with a border that is near the original line that was used to label the initial set. The plot for $k = 1$ shows a highly ragged boundary. Absence of points across the line leads to sharp penetrations into the opposite class. As k is increased, the raggedness decreases. For $k = 3$ the border follows the line a bit better. However, if k is chosen much larger ($k = 30$) the border flattens too much. The reason to this is a lack of 'blue' points in the lower left and the same for 'red' points in the upper right corner.

Yet another problem is speed. The basic algorithm scales bad. For the classification of one point in a problem with n dimensions and m examples, using the Euclidian metric an amount of $m * n$ subtractions, $n * m$ multiplications, $n * (m - 1)$ additions, and $m - 1$ comparisons need to be calculated. This has led to a variety of solutions, like trees that narrow the search-space with every step down the tree,

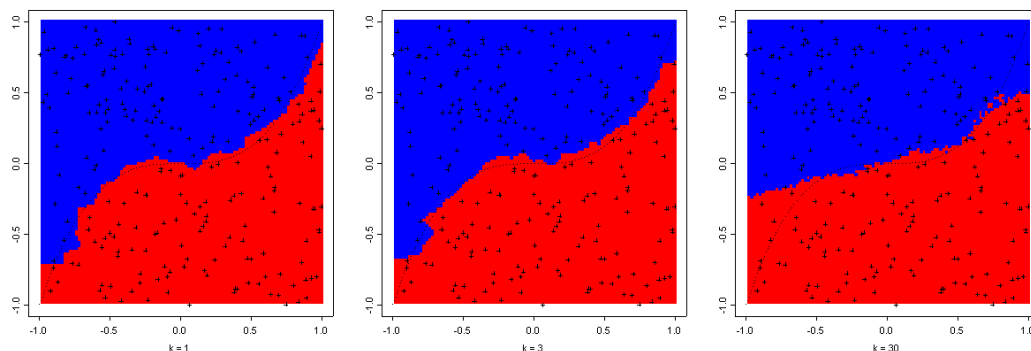


Figure 3.1: k -Nearest neighbour classifier, for different values of k

and bucketing, where one first checks the region near the unknown point, and only checks surrounding regions if no point (or not enough points) have been found.

If the restriction $y_i \in \{-1, +1\}$ is abandoned, regression-problems can also be solved using kNN. There are many ways to combine the nearest points for use in regression problems, each with their own (dis)advantages. The easiest way is by taking the average of the nearest points, but this easily results in large discontinuities in the output-space. A slightly different way is again to weigh the outputs according to the value of the metric. There are many ways to do so, mainly differing in how fast the influence diminishes with the distance. Again, the choice for a value for k determines the smoothness of the output-space.

So-called multiclass problems, where the output is a label to a category, can be solved too. There are restrictions on the chosen solutions. The case for $k = 1$ is easy: the resulting label is the label of the nearest point. When $k > 1$ there are again multiple solutions. The first solution is taking a vote. This is easy, but one needs to take precautions for the cases when ties occur. The other solution is to treat the labels as numbers. This is only allowed if the labels are numbers on an interval or ratio scale. See section 3.4 for more information and solutions to these problems.

3.1.2 Improvements

The basic kNN algorithm works well for smaller problems. When problems become large, the speed decreases as mentioned in the previous section. Different solutions have been proposed to improve the speed.

Berchtold et al. [1998] propose an algorithm to improve the speed for high dimensional spaces. Their solution is to create Voronoi cells for each of the points given during training. Approximations of the cells are stored in a datastructure that is more suited for high dimensional data (different types of trees).

Another example is the use of k -d-trees and cells by Palau and Snapp [1997]. In this paper it is shown that trees can be built to locate the right label more efficient. It is aimed at problems with lots of examples with a small Bayes risk (which means that the examples of different classes should not be mixed too much). The tree is built up by dividing the space in smaller portions until the Bayes risk inside a portion is sufficiently low. This reduces the number of inspected examples considerably.

Ritter et al. [1975] remove examples from the dataset that have been classified

correctly. This results in a less dense dataset consisting mainly of examples near borders between classes.

In bucketing [Welch, 1971] the space is divided in identical cells. If a new point is presented, all cells are ordered according to their distance to the new point. For each of the cells the distance to the points inside the cell are only computed if the distance to the cell is less than the smallest distance found until then.

3.1.3 Implications for the workbench

The basic version of the k -nearest neighbour algorithm has not too many variables that can be optimised, just k and the metric that is to be used. The need to apply mappings indicates that it might be helpful if the workbench provides tools to shape inputs into a different form. This could also prove to be useful in other machine-learning techniques. Being able to visualize the results of those changes is important to understand the underlying relations and patterns in the dataset.

Regression requires more flexibility, as distance functions and their parameters can have a major influence and should therefore be adjustable. Multiclass problems provide more challenges, as the available parameters differ depending on the type of the categories and the chosen solution.

3.2 Neural networks

Neural networks have been proposed as the solution to all machine-learning tasks, and they were assumed make the creation of human-like machines possible. This idea stems from the fact that the neural networks are modelled after ideas how the human-brain probably works. The first work on neural networks starts with McCulloch and Pitts [1943], when they wrote a paper on the theoretical working of neurons. This resulted in the creation of so-called 'perceptrons' in the 1950s. These perceptrons consisted of just two layers of neurons. In 1969, Minsky and Papert proved that these two layer structures are limited in their capabilities [Minsky and Papert, 1969]. This resulted in a diminished interest in neural networks. The interest rose again when Rumelhart proposed a new way to train networks with more than two layers in 1986 [Rumelhart and McClelland, 1986] (which was actually a re-discovery as it was proposed by Werbos in 1974). Because of this and the advent of more powerful computers, new possibilities arose and their use has become more widespread. Almost every major toolkit or workbench has options to train neural networks and test their performance.

3.2.1 Theory

Neural networks consist of multiple neurons, which sum weighted inputs, perform a function on that sum, and pass the results to other neurons. A standard picture of a schematic neuron is shown in figure 3.2. The standard configuration uses multiple layers of neurons that are fully connected, that is, every neuron has a connection to every neuron in the next layer. The function commonly used is a sigmoid function. Thus every neuron performs the following function:

$$f(x) = \frac{1}{1 + e^{-\alpha g(x)}} \quad \text{with} \quad g(x) = \sum_{i=1}^n w_i x_i + b$$

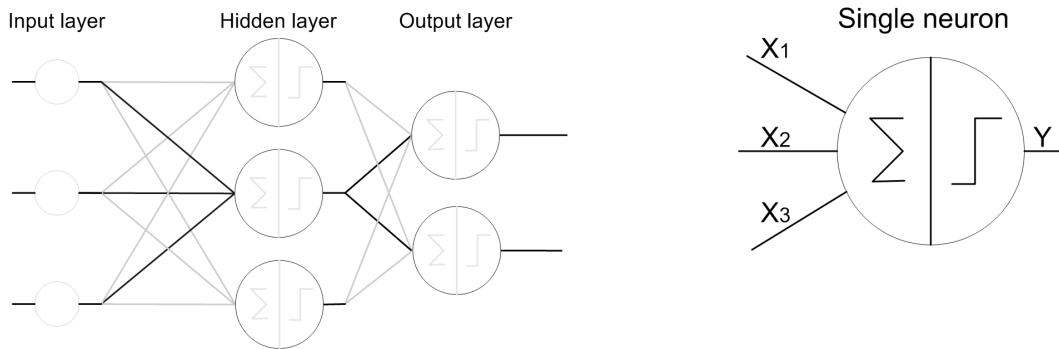


Figure 3.2: Schematic view of neural network and a single neuron

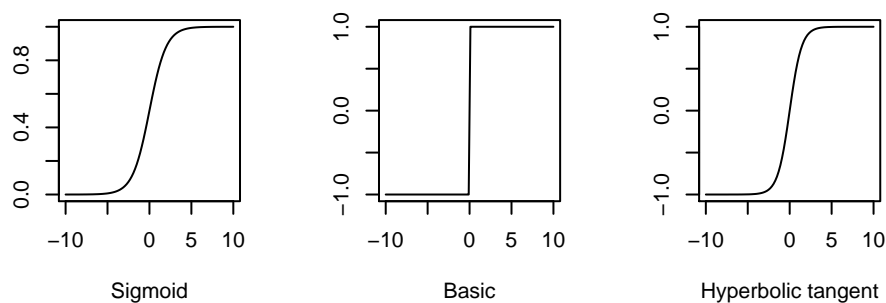


Figure 3.3: Different threshold functions

In this formula there are several variables: x_i is the i^{th} input of the neuron, w_i is the weight of the i^{th} input of the current neuron, and b is a bias, which can be treated as a weight connected to a neuron with constant output value 1. The variable α is used to control the steepness of the transition. The resulting value of this formula lies between 0 and 1. $f(x)$ is a type of threshold function. Examples of other threshold functions are the basic threshold function, which has only 2 levels, and the hyperbolic tangent where $f(x) = \tanh(\alpha g(x))$. Figure 3.3 shows plots of these threshold functions. There are also other possible functions, all of which have their own specific set of areas where they outperform other functions.

To perform useful functions, the variables of the neural net, w and b , should be chosen. For very simple objectives this can be done by hand, but if the problems grow larger, so does the amount of weights. To find the weights using only examples, a method called back-propagation was invented. Back-propagation is an iterative method that adjusts weights in search of a lower error. The error at an output-neuron is defined as the difference between the desired output, and the calculated output. This error is also called the injected error, as it is injected into the network. With this injected error, the local-errors for each neuron can be computed layer-by-layer. The algorithm starts at the output layer, where the local-error is defined as the injected-error at that neuron, times the gradient of the activation-function. For all other layers, the local-error is defined as the gradient of the activation-function times the sum of all weighted local-errors in the following layer. The final step is to update all weights, using the local errors. The formulas for the above procedure

for each neuron are:

$$\begin{aligned}
 e_i &= d_i - y_i && \text{for all neurons } i \text{ in the output-layer} \\
 \delta_i &= e_i f'(x) && \text{for all neurons } i \text{ in the output layer} \\
 \delta_i &= f'(x) \sum_{j=1}^n \delta_j w_{ij} && \text{for all neurons } i \text{ in the hidden layers} \\
 \Delta w_{ij} &= \eta \delta_j y_i && \text{for all neurons } i \text{ and } j \text{ that are connected} \\
 \Delta b_i &= \eta \delta_i && \text{for all neurons } i
 \end{aligned}$$

In these rules w_{ij} indicates the weight of the link from neuron i to neuron j . e_i is the injected-error, d_i is the desired output, and $y_i = f(x)$ is the output of neuron i . η is a factor that determines how much influence an error has and is therefore called the step-size. The local error δ_i is calculated for every neuron. The final step is to update the weights with an amount determined by Δw_{ij} . These formulas are derived for the case that the error function used is a L_2 norm error function (Euclidian distance). As stated earlier, the L_2 norm is commonly used. The advantage of using the L_2 norm for neural networks is that the back-propagation becomes very simple. To allow for other norms, one should apply a function to e_i . Note that in many implementations b is removed by treating it as just another weight, which is connected to an input with the value 1.

3.2.2 Improvements

A major problem with the back-propagation method is that it can get stuck at local minima. Near minima the gradient is flat or in the direction of the local minimum. Because the algorithm optimises the solution in the direction of the gradient, there is no way to escape the local minimum once the algorithm has encountered one. The global minimum, however, could be (far) lower than the encountered minimum. Several solutions to this problem have been invented. One can give the weight-update momentum. Which means that during every update of the weight a percentage of the previous update is added. This makes it possible to shoot through small minima (or the global minimum if it has a small area). Another way is to add random noise to the update. This amount of noise added could be reduced as more training rounds have been performed. This is called simulated annealing, as it resembles jumping molecules that become less-and-less jumpy as the temperature is lowered.

While neural networks are very powerful tools, they also take a huge amount of time to train. Numerous changes have been proposed to make training more efficient [Principe et al., 2000]. Some changes have to do with the initial values of the weights, others go even as far as introducing artificial-fatigue, where neurons get a penalty for firing too often.

3.2.3 Implications for the workbench

Neural networks come in a large variety, and with lots of parameters. This requires also a great flexibility of the workbench. Besides the flexibility training can take some time. If one wants to try a range of values for one or more parameters, the time taken by the training algorithm can be huge. Therefore fast implementations would be a good idea. Another improvement could be the distribution of training

jobs over multiple systems. By using more computers at the same time, the time to wait for all jobs to complete can be reduced drastically.

3.3 Support vector machines

Support vector machines (SVMs) are a relatively new concept in the machine learning area. One of the first publications on this subject is regarded to be by Vapnik et al. [1997], but only in the recent years many people have started to investigate the subject further (led by Vapnik with two books), and use SVMs for tasks that were originally the domain of neural networks (for instance image classification, handwriting recognition and other, mostly classification, tasks). As the field of support vector machines is growing fast, it is not possible to present a complete picture of all theory. Bennett and Campbell [2000] is a good introduction to the whole field of support vector machines, with introductions to extensions such as novelty detection and regression. Burges [1998] is a very thorough tutorial on the SVM theory, and looks at issues such as generalization versus accuracy. Smola and Schölkopf [1998] look into regression with the same thoroughness as Burges does with classification. It also looks into some implementation issues. Gunn [1998] gives a very nice readable introduction to both classification and regression. A very thorough reference is the book by Schölkopf and Smola [2002], which gives clear and detailed descriptions of the SVM algorithms, kernel methods and related techniques. It also provides information on implementation issues.

There are many examples of the use of SVMs. Some examples are: [Smeraldi et al., 1999], [Noone, 2000], [Salomon, 2001] and [Bahlmann et al., 2002]

3.3.1 Theory

The basis of support vector machines is finding a hyper plane that separates the examples, assumed that such a plane exists (that implies that the examples are separated in separate clusters). There can be many of those planes (see figure 3.4), so which one is best, the solid or the striped line? Support vector machines are based on the assumption that the separation between the two clusters is to be as wide as possible. The advantage of this choice is that points just outside the cluster have a higher probability to be correctly classified. If the widest margin between the two clusters has been found, the direction of the separating hyper plane has also been found, which would be parallel to the borders of the margin. The next assumption is that the 'best' line would lie in the middle of the margin, as both clusters get the same amount of slack. The following sections will look deeper into the theory, and start with the easiest case where the training samples are linear separable. From there the theory is extended to show how to solve the case where the samples are not separable, and non-linear. Finally is shown how one can use SVMs for regression.

Linear separable classification

For the classification, a training dataset is used containing points $\{x_i, y_i\}$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$, and $i \in \{1..l\}$. i is the index to the elements in the dataset, which has l elements. Because this is a separable dataset, the subset where

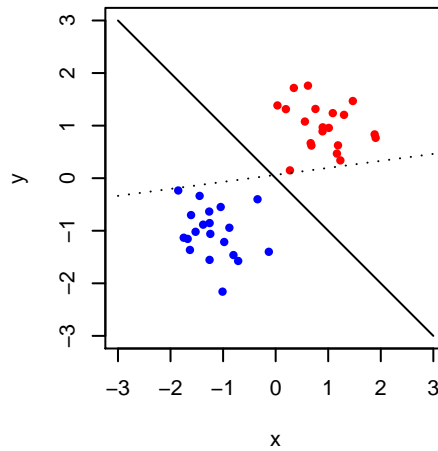


Figure 3.4: Two different separation lines

$y_i = -1$ and the subset where $y_i = +1$ are linearly separable (figure 3.4). The decision function is then defined as $f(x) = \text{sgn}((w \cdot x) + b)$

In this formula w is a vector normal to the separating plane, and b the offset of the line. The arbitrary choice of scale for w and b can be removed by constraining w and b to satisfy $|(w \cdot x) + b| = 1$ for the point nearest to the plane. By doing so, $1/\|w\|$ has become a measure of the margin. The maximum margin can now be found by minimizing w . This leads to the optimisation problem

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 \\ \text{st} \quad & y_i(w \cdot x_i + b) \geq 1 \quad i = 1 \dots \ell \end{aligned}$$

The first step to solve this constrained optimisation problem is to form the Lagrangian,

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{\ell} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{\ell} \alpha_i$$

$L(w, b, \alpha)$ should be minimized with respect to the (primal) variables w and b and maximized for the (dual) variables α_i . This means that a saddle point has to be found. This saddle point can be found by using the Wolfe dual [Fletcher, 1987] of the Lagrangian. This results in

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\ \text{st} \quad & \sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \quad i = 1 \dots \ell \end{aligned}$$

The solution contains one α for every pattern in the dataset. These variables

are only non-zero if the corresponding x_i is on the border of the margin. All other patterns in the dataset have $\alpha_i = 0$.

The original decision function can now be rewritten as

$$f(x) = \text{sgn}\left(\sum_{i=1}^{\ell} y_i \alpha_i x \cdot x_i + b\right)$$

The only part missing in this function is b . This variable can be computed solving $\alpha_i y_i (x_i \cdot w + b) - \alpha_i = 0$ with $i = 1 \dots \ell$

Linear nonseparable classification

In the case that the samples are nonseparable, it is impossible to fit a hyper plane between the sets, as there will always be one or more points separated from their true subset. To solve this problem, one relaxes the original constraints:

$$x_i \cdot w - b \geq +1 \quad \text{for } y_i = +1 \quad \rightarrow \quad x_i \cdot w - b \geq +1 - \xi_i \quad \text{for } y_i = +1 - \xi_i$$

$$x_i \cdot w - b \geq -1 \quad \text{for } y_i = -1 \quad \rightarrow \quad x_i \cdot w - b \geq -1 + \xi_i \quad \text{for } y_i = -1 + \xi_i$$

By propagating these changes throughout the original procedure, the following optimisation problem is obtained:

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\ \text{st} \quad & \sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \quad i = 1 \dots \ell \end{aligned}$$

The only change here is the introduction of C . This variable controls how much error is allowed: a larger C leads to less error (ultimately, for $C \rightarrow \infty$ the original equation is obtained, where no error was allowed). The introduction of C presents a new problem, which is finding the right value for C . There are no standard solutions to this problem. The best way is to try some values by brute-force and decide for what value of C the resulting SVM works best.

Kernels

In most real-world problems, there is no linear hyper plane that will neatly separate the points in the dataset. Instead of making a variant of the technique above, which could fit a non-linear function through dataset (which would complicate the solving process a lot!), one can also apply mappings to the inputs. An easy example is mapping points in 2D Cartesian space onto points in polar space (so every point is represented by its angle and distance from the origin). One can also translate the points into a (much) higher dimension, by making combinations: $[x, y] \rightarrow [x^2, \sqrt{2}xy, y^2]$ in which 2D points are transformed into 3D points (figure 3.5).

To incorporate the mapping in the Wolfe dual, $\phi(x_i)$ can be substituted for x_i . This is a function that performs the mapping from one space to another (with a

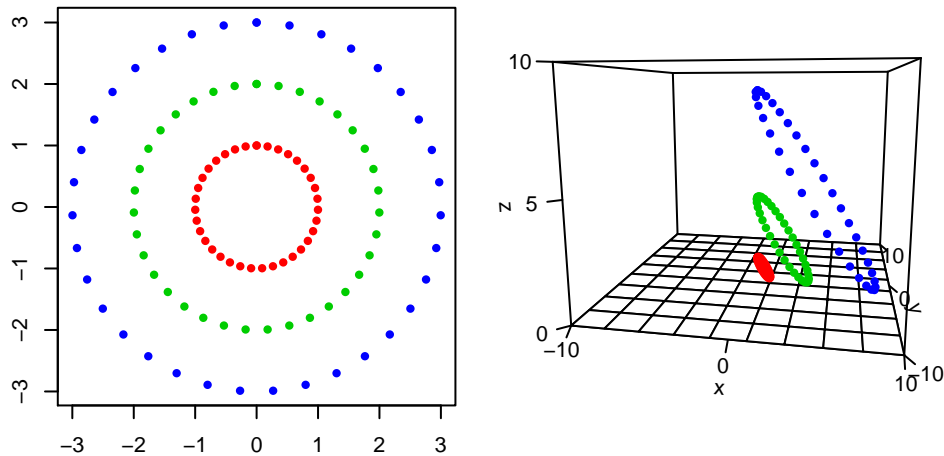


Figure 3.5: Mapping of points in 2D space to 3D space

possibly different dimension). This results in

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \phi(x_i) \cdot \phi(x_j) \\ \text{st} \quad & \sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \quad i = 1 \dots \ell \end{aligned}$$

In these new dimensions it can be much easier to fit a hyper plane. It is even possible to map a point onto an infinite-dimensional space, where all possible combinations of the original point are represented. There is only a 'minor' problem: mapping the input vectors would also take infinite time and memory. Fortunately, using kernels and slightly adapting the form of the optimisation problem can solve that problem too. Kernels define the inner product of two vectors in the mapped space:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

By substituting the kernel method for the inner product in the optimisation problem, the problem becomes:

$$\begin{aligned} \max \quad & W(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{st} \quad & \sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \quad i = 1 \dots \ell \end{aligned}$$

An example will clarify the power of using a kernel over the inner product of two mappings. This example is the above mapping from 2D to 3D (depicted in figure 3.5):

$$\phi(x) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]$$

The kernel $K(x, y) = (x \cdot y)^2$ will result in exactly the same values as $\phi(x) \cdot \phi(y)$. It should be noted though that this is not a 1-to-1 mapping: $\phi(x)$ could have equally well been

$$\phi(x) = \frac{1}{\sqrt{2}}[x_1^2 - x_2^2, 2x_1x_2, x_1^2 + x_2^2]$$

(or for that matter, a mapping to an even higher dimension). The potential power is the computational reduction. To compute the result of the kernel function requires $(n + 1)$ multiplications, and $(n - 1)$ additions, whereas the computation $\phi(x) \cdot \phi(y)$ takes $(6n + 4)$ multiplications and $2n$ additions. Results for other kernel functions and mappings may vary. Kernels allow us to transform data in a variety of ways. There are radial-basis-function kernels, exponential-radial-basis-function-kernels, even Fourier-kernels and multi-layer-perception-kernels with one hidden layer!

Novel kernels have been constructed for many research areas (among these kernels are the ones mentioned above). To construct new kernels, it is also possible to use linear combinations of kernels. Because (integral) transforms are linear, this enables us to (automatically) construct new kernels by using linear combinations of kernels.

Regression

The previous sections were all about classification problems. Support Vector Machines can also be used to solve regression problems by applying some modifications. To do so, w and b need to be found in the expression $f(x) = w \cdot x + b$

To find w , one constructs again a minimization problem. However, this time a loss function is used instead of the margin. The most common loss function is the ϵ -sensitive loss function. This is a function that neglects small errors. In this explanation this function will be used, but it should be remembered that there are other loss functions (such as quadratic-, Laplace- and Huber-loss functions) that all have their own range of problems on which they perform best. The strength of neglecting small errors is that one can influence the trade-off between curvature and the number of support-vectors. As the margin, defined by $(-\epsilon, \epsilon)$ is increased, more points will fit in the function. Using mappings with a higher-order curvature will therefore decrease the error only marginally. The user can therefore use the lower-order mapping, as lower-order mappings are generally considered to generalize better. The optimisation problem becomes:

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} (\xi_i + \xi_i^*) \\ \text{st} \quad & y_i - w_i \cdot x_i - b \leq \epsilon + \xi_i \\ & w_i \cdot x_i + b - y_i \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned}$$

The variable C is again a way to adjust the final result to deal with errors. By using the procedure that was used before for the classification SVM, creating the Wolfe dual of the Lagrangian, the problem can be transformed into a more man-

ageable form:

$$\begin{aligned} \max \quad & \sum_{i=1}^{\ell} y_i(\alpha_i - \alpha_i^*) - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) - \epsilon \sum_{i=1}^{\ell} (\alpha_i + \alpha_i^*) \\ \text{st} \quad & \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i, \alpha_i^* \leq C \end{aligned}$$

It follows that $f(x) = \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*)(x_i \cdot x) + b$. All variables are known, except b . To solve for b , one needs to use the Karush-Kuhn-Tucker conditions. These conditions mean that the product of dual variables and constraints is zero for the optimal solution. By solving b for these conditions, b can be computed as follows [Smola and Schölkopf, 1998]:

$$\begin{aligned} b &= y_i - (w \cdot x_i) - \epsilon \quad \text{for } \alpha_i \in (0, C) \\ b &= y_i - (w \cdot x_i) + \epsilon \quad \text{for } \alpha_i^* \in (0, C) \end{aligned}$$

3.3.2 Improvements

There have been many suggested improvements to the basic definitions above. Some focus on speed improvements, others on the reduction of the number of support vectors. The latter is important, because a model with fewer vectors with the same error is generally considered to be a better generalization. Downs et al. [2001] suggest a method where unnecessary support vectors are removed, based on the criterion if a support vector is linearly dependent on other support vectors. Gentile [2001] introduces a new algorithm to replace the core of the SVM for certain problems. It trades some accuracy for an increase in speed. Ben-Hur et al. [2001] present a clustering technique based on support vector machines. Mangasarian and Musicant [2000] and Musicant and Feinberg [2002] show that by reformulating the initial formulas, it is possible to create a faster type of support vector machine, which can even outperform the standard implementations.

3.3.3 Implications for the workbench

There are many (different) implementations of the support vector machines and related techniques. These implementations are available in different languages and binaries. To couple these and future versions, the benchmark system has to be flexible in supporting the different languages. Also the great variety in kernels must lead to easy option setting, so one can easily test different implementations with different kernels. Support vector machines do not support multiclass outputs. Different solutions have been proposed to solve this. Most of these solutions come down to training a multitude of SVMs, and combine their results. It would be a hassle to create a new program for every combination of SVM and multiclass solution. The workbench should therefore allow the user to choose both an SVM and a solution, and join them. As the variable C is important in producing a good SVM, the user should get tools to change C easily and visualize the influence of the choice.

3.4 Multiclass problems

In previous sections, it was mentioned that some techniques could not solve multiclass problems by default. To extend these techniques several methods are available. In this section some of these methods will be studied.

3.4.1 Theory

Multiclass problems are problems where the labels of an input or output feature fall in one of multiple (more than two) classes. If a technique is not able to handle these multiple classes, the classes need to be mapped onto another structure that can be handled and gives similar results.

This mapping is usually onto either a binary or a continuous value. The choice for either depends on the type of classes used. There are four scales on which the classes can be projected called nominal, ordinal, interval and ratio scales [Stevens, 1946]. The nominal scale only requires that classes have unique labels, which don't need to be numbers. The ordinal scale is a little more strict. On this scale the classes have to be numbers, and they need to have an order, so one can tell which one is greater than the other. The interval scale augments the previous scales, by requiring the subtraction of values results in valid values on this scale. That way, relative distances are preserved. The ratio scale requires that the use of the division operator maps onto valid values.

If the feature that needs to be converted is either a nominal or ordinal value, it should be mapped onto binary values. The reason is that if the classes were mapped onto numbers that are treated as if they were continuous values, wrong results arise. This is the result from the fact that numerical operations are not allowed on these numbers.

Interval and ratio scales can be mapped onto continuous values, although care should be taken when using operators more complex than the addition and subtraction operators on ordinal scales. Averaging should not lead to problems when the resulting values are mapped onto the nearest value on the scale, however the results after division are dependent on the used numbers.

The mapping of interval and ratio scales onto a continuous scale is straightforward. One can just treat the numbers as if they were continuous. For the mapping of nominal and ordinal scales onto binary values multiple solutions exist. The following parts will look into these mappings. The first will look at the solutions when the scale is an input feature, whereas the second will look at output features.

Multiclass input

There are many different ways to map the scales onto binary values for the input space. In this part some of these are discussed. Each mapping has its own strengths and weaknesses. Therefore, there is no best mapping in general, it depends on the problem one is dealing with.

The first mapping is the 1-of-n encoding. In this encoding each possible class is mapped onto a bitstring in which only the bit corresponding to the class is 'on'. This leads to a very sparse representation. The disadvantage of this mapping is that the dimensionality of the example is increased. Machine learning techniques that have difficulties dealing with 'the curse of dimensionality' can have troubles when this technique is applied [Bellman, 1961].

The second mapping is the 1-of-(n-1) encoding. This encoding is almost the same as the previous one, except that one class is not mapped onto its own bit, but defaults to the pattern where all bits are 'off'. It suffers from the same problems as the 1-of-n encoding, but it can sometimes avoid some problems in the techniques themselves.

A third mapping is specific for ordinal scales and is called the thermometer code [Masters, 1993]. For n classes it also uses n bits, but this time all bits after a certain class are 'on'. For example, in a six-class problem, the third class is encoded as 001111.

Other approaches are possible too, most of them with the aim to limit the number of bits used. This can vary from presenting the class as a binary number to using error-correcting codes (these will be discussed in the next part). These approaches are not guaranteed to give good results, as one imposes a structure onto the labels that is not necessarily present in reality.

Multiclass output

Multiclass problems are problems where the output falls in one of multiple, more than two, classes. If these classes cannot be fitted onto an interval or ratio-scale, they cannot be approximated by using a regression problem, as this would result in false results. The basis of solving multiclass problems is the reduction of the problem into multiple binary problems. The results of these problems can then be combined to answer the original multiclass problem.

The easiest solution is the one-against-one classifier. For every combination of two classes, a binary problem is created that votes for either class. For a ten-class problem this will result in $10 * (10 - 1) / 2 = 45$ smaller problems. The class that receives most of the votes is then elected as the final answer. For small numbers of classes this might be a viable solution, however the number of resulting binary problems grows fast as the number of classes is increased. As for each binary problem a complete classifier has to be trained, the required time and memory will soon become prohibitive.

A more practical method for large problems is the one-against-all approach. In this approach a binary problem is created for each of the classes. The binary classifiers must not be binary in the strict sense: some sort of confidence measure is needed. This could be the result of the method before rounding. Each binary problem is trained to classify the corresponding class as positive, and all other classes as negative. The resulting number of binary problems scales therefore linearly with the number of classes. The 'winning' class, is the one for which the corresponding binary problem returns the highest value. A possible problem arises if the values resulting from the binary classifiers lie close together. Small variations will then result in selecting different classes.

The ways to solve the multiclass problem treated above create either too much sub-problems, or are very sensitive to small errors in each of the sub-problems. The use of error-correcting codes allows the user to balance the amount of sub-problems created against robustness.

Error-correcting codes stem from the communication field. When (binary) data is sent over a line, noise can change bits. To detect these changes, one could use a parity bit, which can be compared to the parity of the received data. Although this can help in detecting errors, it cannot correct them. By sending multiple bits extra, and using a smart coding, it is possible to correct one or more errors. To do so, each

original code is mapped onto a longer codeword. Not all of these longer codewords are used, only the codewords that differ at least some predefined number of bits. When the longer codeword is changed during the transmission, it is possible to find the codeword that has the closest resemblance. This resemblance is based on the Hamming distance, which is defined as the number of non-matching bits. If one bit changes during transmission, the Hamming distance of the received code to the sent code is one. If one uses a code where the Hamming distance between each of the codewords is at least d , one can correct at most $(d - 1)/2$ errors.

To use error-correcting codes in multiclass problems, the classes are mapped onto the codewords. Then a separate classifier is trained for each of the bits in the codeword. To classify a new point, all of the classifiers are used, and the results are combined to form a codeword. This codeword is then compared to each of the codewords that were used for training. The codeword with the minimum distance is then used to classify the new point.

To obtain the codewords to use, one needs to choose a code. There are different classes of codes to use. One of the easiest codes is the simplex code. The first step in constructing a simplex code is the creation of a generator matrix. For example, a generator matrix of three rows:

1	0	0	1	1	0	1
0	1	0	1	0	1	1
0	0	1	0	1	1	1

This generator matrix is essentially a list of all non-zero bit strings with length 3, which are then placed in the columns of the generator matrix. The codewords of the simplex code are all linear combinations of the rows of the matrix. For generator matrices of k rows, this implies, that the code has a total of 2^k codewords (all linear combinations of the rows), each with a length of $2^k - 1$. The minimum distance between these codes is 2^{k-1} .

The generation of other codes is possible but often more difficult. Well-known examples are Reed-Solomon codes, BCH codes and other cyclic codes. It is possible to make modifications to tailor the code. Among these modifications are shortening, puncturing and concatenation.

Error-correcting output codes are shown to outperform other techniques on a variety of problems [Dietterich and Bakiri, 1991, 1995, Aha and Bankert, 1997]. Kong and Dietterich [1995] show that these codes are able to correct both bias and variance in multiclass problems.

3.4.2 Implications for the workbench

The workbench should allow the user to choose a suitable transformation for multi-class in- and outputs. This should be done independent from the machine learning techniques where possible.

4

Design

In this chapter the design of the workbench is discussed. The first section presents a conceptual model, which will introduce the terminology that will be used in this thesis, and to show how one might use the workbench. Thereafter, the choice of the programming language and libraries will be discussed. This base is used to look at the design of the structure of the workbench. The last section will review the requirements to check if they are met by this design.

4.1 Conceptual model

To gain an understanding of the functions that the workbench should perform or facilitate, a model description is given. This description will introduce the components that are to be included, and the terminology that will be used (refer to chapter 9 for definitions). This description is used as a use-case to identify different parts of the workbench.

First of all, there is the user-interface, which will provide forms with options to the user, and which will show the results of the tests. The parts that handle most of the interactions with the user, are the parts called modules. There is a module for each task, such as 'loading a dataset', 'adding a SVM to the script', 'plotting the results', etc. The modules communicate with each other via the workbench-framework. This framework provides data structures to store data that can be used by the modules, such as the dataset and the test-script. Because of the modular structure, additional functionality can be added at later times without the need to rewrite the workbench. It should even be possible for others to extend the workbench with relative ease.

A normal way to work with the workbench could be characterized as defining the dataset, setting up a test-script, run the script and view the results. Each of these steps will now be discussed.

The first step is to define a dataset. The dataset is a list of examples, consisting of the values of the features and the resulting values. The source of this data could be (but is not limited to) a regular file on the hard disk, a website that needs to be parsed, or a file on an FTP-server. To import the data into the workbench, different

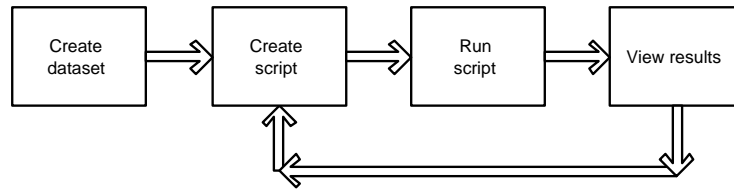


Figure 4.1: testing process flow

modules are provided, typically one module for each type of import. As this data can be in many formats, there will be some additional modules that can convert these data to an intermediate form. On this intermediate form, the user can specify operations, like scaling or normalizing values, and converting categorical items to other forms (for example multiple binary inputs, or a continuous value). These data can also be visualized and tabulated to provide the user with basic tools to gain insight in the structure of the data. You could think of plots that will offset some variables against each other, or reports that show the correlation of inputs.

The next step is to create a test-script (or script for short). Scripts consist of a tree-like structure of script-modules. The user-interface for these modules allows the user to make changes in the parameters that will be used during execution. When the user executes the script, the modules can make changes on the dataset, and pass the data to their children. Script-modules come in a wide variety. The leaves of the tree are modules that create the wrappers that will interface with the external tools. These modules need to be fed with transformed data. Therefore there are modules to make selections on the dataset (both for feature selection and example selection), modules that perform transformations on the dataset, and modules that split the dataset in parts for training and testing.

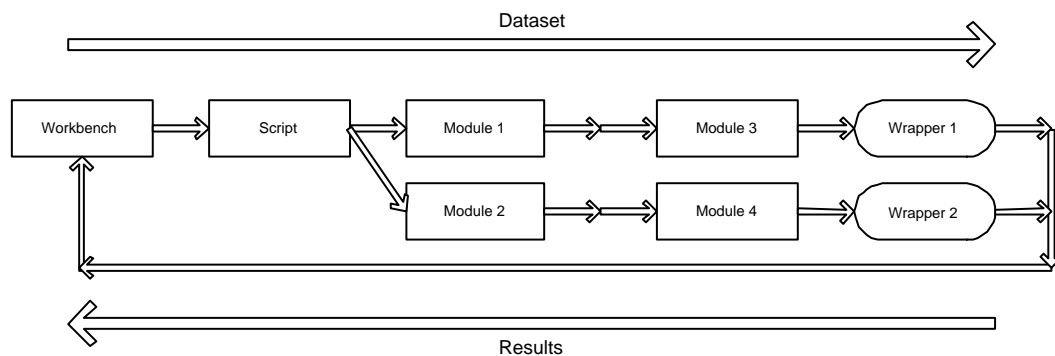


Figure 4.2: process flow for running script

The third step is to run the script when the user has completed it. The script-modules transform the dataset, and pass it on to their children. Ultimately (at the leaves of this tree-structure) this will result in the creation of wrappers that will interface with the real tools. The number of wrappers can be enormous, as a wrapper has to be created for every call to a tool with a different set of parameters. Because each module will call its children for every variation of the parameters (if desired), the total number of wrappers grows fast. Therefore these wrappers are distributed among a pool of clients that provide the necessary processing power. The wrap-

pers execute the tools, providing them with parameters and the dataset (possibly in several phases: separate phases for training and testing). Afterwards the results are converted and returned to the workbench.

Thus, all these modules make it possible to create a complete test-script that has multiple ways of splitting the dataset, selecting different sets of features, and train/test different machine learning tools. This makes it possible to compare the performance of different tools (for instance a SVM and a neural network with a certain topography) on exactly the same train- and test-set. Because of the recursive passing of the dataset it is even possible to do multiple tests with varying parameters by allowing the user to specify parameter intervals instead of single values. All these modules facilitate the creation of thorough comparison tests, and the selection of the best parameters for a certain problem.

The last step in a typical process is gathering and visualization of the results. Each wrapper returns some output, and all these pieces are combined to one big result-set. Modules aimed at visualization can read these results and present them in various forms to the user. These modules can be as specific or broad as required for the problem at hand. Examples are selecting some numbers to plot against some other measure, or modules that generate \LaTeX files that can be included in reports.

The keyword for this workbench is modules. All functionality that can be used for research is provided as a set of modules. This makes it possible to exchange modules with others, and to stack modules to create a much more powerful tool. The only requirement is that they share some common interfaces.

4.1.1 Example

The following is an example of how a user might use the workbench to test if either a neural network or a support vector machine suits his needs better. To perform this test, he will first need to import the data into the workbench. Therefore, he opens the window of the workbench, and makes a new project. Then he clicks on the name of the project to open it. He is now presented with a list of links to modules. As the data resides on the hard disk as a large file, he opens the 'upload raw data' module. Now he is shown a screen with some accompanying text and a field where he can enter the name of the file. By pressing the save button the workbench reads the data and stores it.

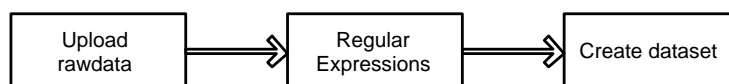


Figure 4.3: dataset modules

The raw data is not yet a dataset. As the raw data contained tab characters to separate values, and the 'raw data to dataset'-module requires comma-separated values, the user returns to the project screen and selects the 'regular-expressions'-module. Now the user is shown three fields, one for the text to look for, one for the text to replace found values with, and one with the raw data. He enters the command for tab in the first field, and a comma in the second. By clicking save the workbench translates all tabs to commas. The raw data is now ready to be converted, so the user clicks the module to convert comma-separated raw data to

the dataset. The workbench confirms the action, or shows where values are missing (if any).

As the dataset contains some values irrelevant to the test, the user first clicks (in the project-screen) on the feature-subset module. He selects the features that he would like to include in the test, and saves the data. To do a quick test, he adds a 'normal split' module by clicking the 'add module' link on the previous module. This brings up a screen where he can select the required module. After selecting the right module, he is shown the form with options for the module, and he selects a random split at 75%. After saving the form he can repeat the process to add a neural-network-module. In the form of this module he can select the number of layers and the number of neurons for each of the layers.

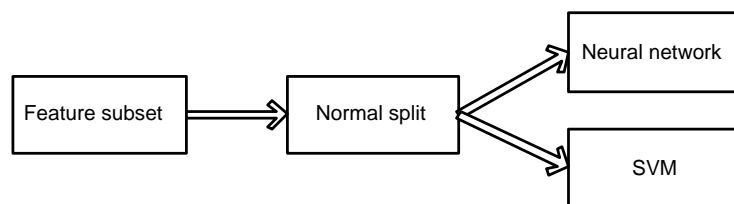


Figure 4.4: script modules

As the user wants to test the performance of a support vector machine as well, he goes to the script-screen. There he selects the 'normal split'-module that he has added earlier. Now he can select to add a module again, this time an SVM-module. The presented form allows him to make settings to influence the behaviour. He selects a simple variant and saves the form.

Now the script is ready to run. The user starts the execution, and the modules start to work on the dataset. This results in two wrappers, one for the neural network and one for the SVM. These wrappers are then executed. After a while the wrappers have completed their task and transferred the results to the workbench. As there are not many results, the user opens the XML generated for the test, and looks at the results.

As the normal splitting is not really representative, the user removes the current script and rebuilds it. Instead of using the normal-split module, he now uses the cross-validation module, with certain parameters. To this module, all previous modules can be added in the same way. The user now also selects to vary the number of neurons in a certain range, and gives ranges of values for the parameters of the SVM.

When the user chooses to run the script, lots of wrappers are created. A new wrapper is created and executed for every subset of data, indicated by the cross validation module, and every combination of parameters. This time, the amount of resulting data is too big to be comprehensible at a glance. The user therefore selects a visualization module, which allows him to display subsets of information. He could for example plot the relation between the number of neurons and the resulting error.

By using the results of the tests, he can now alter the parameters in more specific directions, or decide that for the task either the neural network or the support vector machine is the better choice.

4.2 Language and libraries

The basis of the design starts with determining the 'right' languages and libraries to use. This choice determines how easy or how hard certain steps in the implementation will be. Important factors for this project are the flexibility of the language, ease of use, and the availability of, and knowledge about, the language at places where the workbench is to be used. Speed is an important factor too, but only for some components (mainly those components that will be used for training and testing). As most machine learning tools are built in many different languages, interfacing should be a key element in the language. This will ease the integration of existing tools with the workbench.

Java was chosen as the main language for the workbench. Java is a relative new language, which has lifted on the success of the Internet. A lot of universities teach Java as the primary programming language. Among these are the three technical universities in the Netherlands, as well as MIT (USA), Cambridge (UK) and Berkeley (USA). Reason to do so is that it is an easy, yet powerful and expressive language. It takes some chores like memory-management away from the programmer, which decreases the time needed for development.

The libraries provided in the standard Java distribution provide the programmer with a variety of interfacing options. Functions are built-in to interface to programs over the command-line, but it is also possible to create connections by using remote method invocation (RMI) when necessary, for which libraries are included, just as there are libraries for connecting to services via sockets and even complete HTTP libraries.

Because Java is taught at many universities, and is easy to learn, it has acquired a huge community. There are lots of libraries available over the Internet, often provided with the source code. Lots of machine learning algorithms are freely available in Java.

Java is available on most platforms and does not require recompilation when porting to a different platform. This also creates one of the biggest drawbacks for Java, which is regarded to be performance. Because Java uses interpreted bytecode to create the independence of the computing platform, it is not a race monster. However, much research is being done on improving the performance, and in some specific cases the performance of C++ has been matched. As Java has great interfacing options, speed will not be too much of a problem: pre-compiled tools can be used for the computations (although this will make the portability somewhat harder, as one needs to find tools that are available for all intended target platforms).

However, most languages (and that includes Java at certain points) have the problem that building user-interfaces is time-consuming (especially if no GUI-builder is available, or if it has insufficient functionality). In the light of the ease of building websites, it is not strange that more and more applications are web based or have a web based front-end. Web based applications have the advantage that a description of the user-interface can be built in HTML (using either a text editor or another tool). Web browsers will transform these descriptions to graphical screens. The browser also performs the task of transferring the user-input to the application in a standard format. The drawback of this approach is the lack of state: each page should contain enough information to inform the workbench where the user was in the process. Fortunately this is not an issue for the workbench. The amount

of data needed by the workbench to do so is small. Moreover, submitting state-information, and not relying on a previous state can reduce the number of bugs. There is less chance of interference between variables as there are no shared values between steps, other than the data submitted by the user and the data in the workbench.

Java ServerPages (JSP)/Java Servlets were chosen to give the workbench a web based front-end. Java Servlets are Java classes that send their output over HTTP. Most of these Servlets return HTML, but it is also possible to return images or other types of information. Servlets must be compiled by the user and placed on a web server. This web server will transform the HTTP requests into function calls on the Servlet. The resulting output is sent to the user. Java ServerPages is the technique to integrate Java code in HTML pages. These pages are compiled at the moment that they are requested by the user. This has the advantage that the webserver automates the compilation process and automatically reloads itself when needed. When changes are made to Servlets, the user has to manually compile the Servlets (or use a tool to do so), and restart the web server.

JSP and Java Servlets are defined in a specification by Sun Microsystems, which also provides standard implementations. These standards provide a nice basis for the workbench by removing many chores from the user. The disadvantage is that this requires the installation of a web server with support for Servlets (but this gets easier as more web servers are released that are easier to install). This also has the added advantage that it will be possible to work with the workbench without having it installed on your own system. This opens new possibilities, like collaboration of multiple people on a project, or checking the progress at home!

To facilitate the distribution of jobs over multiple hosts, a technology called JavaSpaces was chosen. JavaSpaces is based on David Gelernter's TupleSpaces. The core idea in this technology is that there is a shared memory where processes can store data and even new processes. All stored items have additional information that can be used to find them. JavaSpaces enhances the TupleSpaces idea with Java concepts. The stored items are all objects. To store an object a process can simply put the object into the space. A process looking for a certain item can then give criteria to match, such as the class of the object, or certain values that the variables should have. If an object matching the criteria is found it is returned to the process. JavaSpaces can be extended with transactions, which provides stability to the system in the light of possible crashes of processes.

The JavaSpaces technology is part of the Jini network technology architecture, initially created by Sun Microsystems (under leadership of Bill Joy and Jim Waldo) which gave it to the Jini community, continues the development. Jini provides the basic functionalities to the system, such as locating a Java-Space, leasing mechanisms and code mobility. Leasing is a technique to cope with network errors and failing processes. Processes need to acquire leases on services provided by Jini. These leases need to be renewed regularly. If a lease is not renewed then it expires, and processes can no longer make use of the service. This might occur if the network fails or a computer crashes. To keep the system from being filled with unusable services, all other services and processes can remove their references that point to services of which the lease has not been renewed. Code mobility is another part that the workbench will use. To retrieve an object from the JavaSpace, the original classes are needed. Jini provides methods to transfer classes from one machine to another, where the objects can then be reconstructed. It also allows clients to

execute new programs that have not explicitly been installed on the client.

4.3 Structure

The structure of the elements that make up the workbench is described in this section. This description will follow the structure of the objects as almost everything in the workbench involves the data contained in these objects.

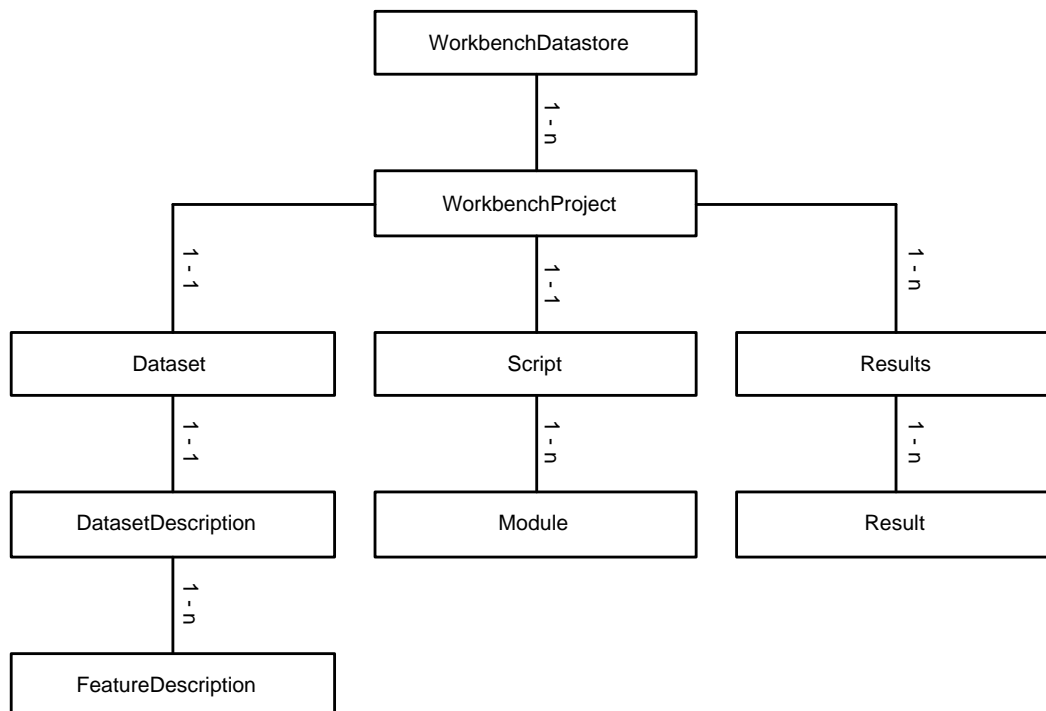


Figure 4.5: class diagram

The base of the structure is a tree. The following sections will describe this tree, following a top-down approach, starting with the root, and working towards the leaves. The design process of this structure was bottom-up, as this design followed from the conceptual elements and their relations. The elements that were identified in the conceptual model were the dataset, the script, modules and the results. The dataset, script and results share a common relation: they are all parts of a project. Therefore a new element can be placed that connects with the three elements. An element called the datastore was placed above the projects to allow multiple projects to coexist and provide the programmer with a single interface to access these projects. The script can be divided in modules. Each module can have several modules as children. The results consist of multiple result parts, each containing the data for a run of the script.

As this tree structure follows from the conceptual model, it is easy to follow for new users. The tree structure also allows parts of the tree to be exchanged for enhanced parts, without compromising the existing structure.

4.3.1 Datastore

The datastore can be viewed as the top of the data structure. It serves as an access point to the rest of the data. The datastore should be the only object to keep references to projects. By doing so, the method of storage can later be decided upon.

To allow users to have a unique access point to the data, the datastore-class adheres to the singleton design pattern. This pattern shields the creation of instances of this class from all other objects. The only way to obtain an instance is to ask the datastore for an instance via a static method. This instance is created during the loading of the class. This way there is always one object that contains references to all data that might be needed.

The datastore contains several functions to list all projects and access individual projects. The identification of projects is done by giving each of the projects a unique name (specified by the user). This name is used with functions that are provided by the datastore to add, locate and remove projects, as well as listing all available projects.

As the datastore is the top of the datastructure, and is always created when an object refers to it (because of the singleton pattern), it is also the place where future enhancements can be implemented. It could be used to start threads for monitoring the JavaSpace or save data at regular intervals.

WorkbenchDatastore
projects : Hashtable
projectPath : String
<u>wds : WorkbenchDatastore</u>
<<create>> WorkbenchDatastore()
<u>getInstance() : WorkbenchDatastore</u>
addProject(p: String) : void
hasProject(p: String) : boolean
getProject(p: String) : WorkbenchProject
getProjects() : String[]
setProjectPath(path: String) : void
getProjectPath() : String
saveProject(p: String) : void
saveProject(p: WorkbenchProject) : void
removeProject(p: String) : void

4.3.2 Project

The project class is the portal to all data for each project. This keeps data of different projects neatly separated.

Each project consists of a dataset, a script and all results, as well as some supporting data. The project object gives programmers access to these parts. The choice was made to have only one dataset and script in each project, but to keep the results of all previous executions of dataset and script setups. The reason to do so results from the fact that tests are evolving over time and earlier versions will not be used anymore. A dataset is usually created once for each project. The script performs most of the changes on the dataset right before the dataset is passed to the wrappers. After a script is executed there are usually some modifications that are to be made to the script. After several rounds of modifications the script might have evolved into a script that has little resemblance of the old script. Because the results are still available there is no need to go back to an earlier version of the script. The only reason to go back is to use it as a shortcut for creating a different script. The ease of use of the modules should minimize the impact of not having this ability. The advantage for the user is that the situation is kept simple and clear. The alternative would have been to show the user a group of select boxes each time he wants to perform a test. These select boxes would allow him to choose the dataset and script to use. As

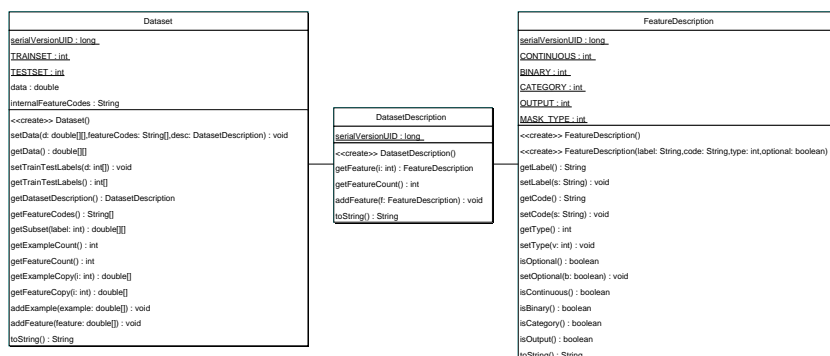
WorkbenchProject
name : String
vars : Hashtable
dataset : Dataset
results : Results
<<create>> WorkbenchProject()
<<create>> WorkbenchProject(name: String)
setVar(id: String, o: Object) : void
getVar(id: String) : Object
getVars() : Enumeration
removeVar(id: String) : void
setDataset(ds: Dataset) : void
getDataset() : Dataset
getScript() : Script
setScript(s: Script) : void
getName() : String
setName(n: String) : void
getResults() : Results
setResults(n: Results) : void

some of these combinations cannot work together (mismatches in the given and expected number or type of features for example) there would have to be provisions to warn the user etcetera.

The dataset, script and results are described in the following sections. The supporting data that was mentioned before can be used as glue for different modules. This storage is facilitated by providing three functions. The first is to store an object under an identification code provided by the programmer. The second one is to list all identification codes of stored objects. The third is to retrieve an object based on the identification. Different types of objects can be stored easily as all objects inherit from a class called 'object'. The programmer has to take care that the object returned by the project is converted to the right type. The workbench will take care of the persistence of these objects by saving them. The advantage of simple object-storage is that users can provide their own data-objects if needed, and store those in the project. All modules that know the identification-code can share this data. This allows users to create sub steps in the process. For example, one module could process some data and store it in the project as an object. A second module can then retrieve the data, process it and store the values in the dataset. New types of modules can easily be integrated within the workbench if they follow the same conventions.

4.3.3 Dataset

The dataset contains the data that is used to execute the tests. It consists of all data values, a description object and functions to access those. The values will be stored as doubles to provide enough precision. Values of binary and multi-class features should be properly encoded before they are stored in the workbench.



The description object is used to present the user with the right information on features. For each feature some information is kept, such as a name that the user can use, and the type such as continuous or binary, as well as a flag that indicates that the feature is to be used as an input or output. It also contains a label that the modules can use internally to refer to features.

The description object should not allow programmers to retrieve information on features by an index number, only by the internal label. The dataset should therefore store a mapping of the column number and the label of the corresponding feature. The reason to do so is that the order and the amount of features in the

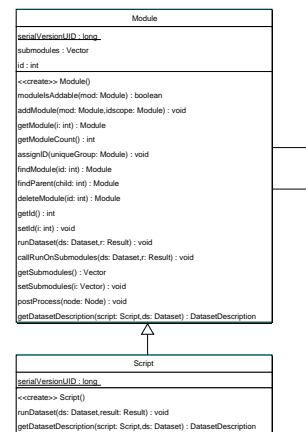
description object need not to correspond to the order or number of columns. The only requirement is that each column in the dataset has a label, and that a feature description is available for each of these labels. This prevents modules from storing parameters for features based on an index that might change if the dataset is changed.

The default implementation of the dataset will make use of a two-dimensional array for the storage of the values. By only allowing these values to be accessed via get and set functions new dataset objects can be introduced. This would allow the workbench to use sparse dataset formats or dataset that are generated from functions. As other parts of the workbench only communicate via functions defined in the public interface of the dataset class, and not directly with the internal objects, these new datasets are easier to integrate.

4.3.4 Script

The meaning of script is two-fold. It means both the complete script that is used to execute tests, as well as an object that acts as the starting point of the complete test-script. Here we will look at the object. The script object is an extension of the module class (see section 4.3.6). This means that modules can be added to this script. When a project is created, a script object is added automatically. The script may then be accessed via the project object.

The script object in the project is also used as a starting point to locate other modules or start the execution of the script. For this execution, the script is given the responsibility to initialise a new result object, and pass this result object as well as the dataset to the modules in the script.



4.3.5 Results

The results object contains all result objects that were produced by executing tests in the project. It functions as an access point to the individual results. To do so it contains similar functions as the datastore object for adding a result, requesting a result, requesting the identification keys and removing a result. Every execution leads to a new result object being added to the list of results. The modules that make up the script then fill this result object. After the execution the results object is used to get access to the older result objects so that they can be reviewed using visualization modules.

Result

This is the object that is saved into the results object. It stores information related to the execution of a script and provides functions to store and access that information.

The primary source of information is a string containing XML. Several functions are provided to fill this XML string with data. First of all, each module that is executed should register itself with the result object. This results in some house-keeping tasks such as incrementing the number of modules, as well as appending

a tag with identification to the XML. A function that embeds XML inside the tag belonging to the module is provided to allow modules to insert information in the XML that might be useful for the user. Finally, when the modules called by the module have been finished, the module should call a function on the result object to close the tags.

Modules that produce wrappers register their wrappers with the result object. This again allows the result object to perform housekeeping jobs. The result object also sends the wrapper to the JavaSpace.

To allow results from wrappers to be integrated afterwards, each module that produces wrappers should include a tag with the name 'results' in the XML. This allows the function 'integrateResults' to insert the resulting XML at the right place in the XML, by using an XSLT transformation.

The 'integrateResults' function also records the 'WrapperResult' object that was passed by the wrapper. In this object additional information is kept, such as the raw output generated by the tool, and exceptions that were thrown. This data can be used to detect problems in the tools that might otherwise go unnoticed.

If all results have been integrated (according to the count of the number of wrappers accepted and the number of results that were integrated) the function 'postProcess' is called. The 'postProcess' method will iterate depth-first over all registered modules. For each module an instantiation of the object is created. An object containing XML for the object and all sub objects is passed to this instance. It then has a chance to make changes on these data. An example of this is adding new XML that contains summaries of the data produced by sub modules.

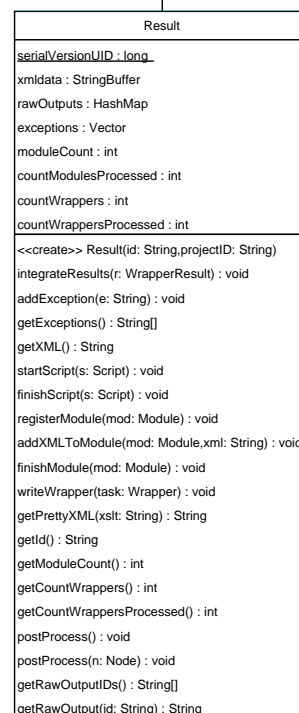
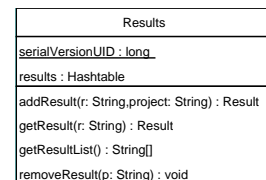
To retrieve the XML in a different format, a utility function is provided to apply an XSLT transform (passed by the calling function) on the XML and return the results of this transformation.

XML format

The structure of the XML that modules can use is very flexible. There are only some small restrictions. Modules can divert from these guidelines if there are good reasons to do so.

The XML should start with a 'script' tag. This acts as a container for the complete XML. Within this script several 'module' tags can be placed, as well as future tags. Modules can place their settings within the tag 'settings', which is to be placed within the module block. Within this settings block, tags with the name 'setting' should be placed. The attributes 'name' and 'value' are used to show the right information. This structure allows modules to extract the settings in a uniform way.

Module tags can be placed within the tag of another module. These tags can be placed right within each other, or wrapped inside tags that are appropriate for



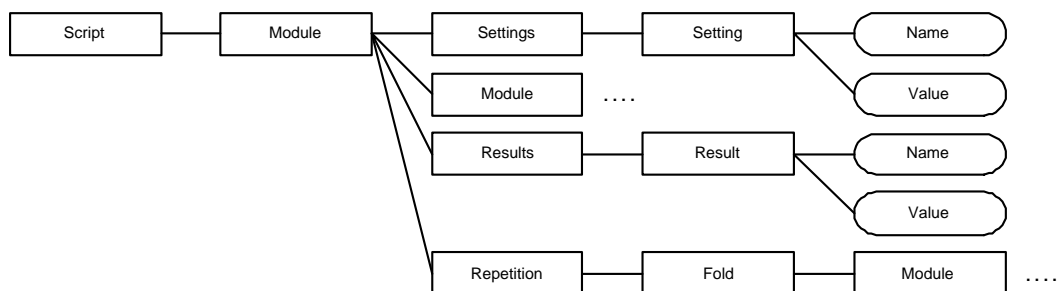


Figure 4.6: XML structure

the embedding module. A cross-validation module can wrap sub modules within repetitions and folds. The choice of tag names and attributes is free to the module designer. The block with settings can also be placed within these inner tags, next to the tags of the sub modules. This indicates to the user and visualization modules that the settings are only valid for this block, not for other blocks.

Another important block is that of the results. Results are kept within a 'results' tag. Within this block each result is indicated with a 'result' tag, containing two attributes, called 'name' and 'value'. By structuring data this way it also improves the uniformity of the XML, thereby easing the automatic retrieval.

4.3.6 Modules

The whole process of creating and performing tests consists of a series of steps. Parts of the process are interchangeable for other parts to create different results. A natural way to match this structure is to use a modular structure. The workbench therefore has modules for all common tasks.

The different modules will be discussed as three different categories: dataset, script and visualization modules. This choice results from the conceptual and functional division of these modules. The functionality and peculiarities of the types are different, and will be discussed in their respective sections.

Modules also differ from the objects that make up the framework of the workbench, as these modules are usually JSPs. The script modules have an object counterpart, but the other modules are mainly JSPs that act on the data stored in the project.

Dataset modules

Dataset modules are the modules that work on the dataset, or provide functions that facilitate other modules to make changes on the dataset.

The general structure of these JSPs is to show a form with boxes that the user can fill with data. When the user presses a button the data is sent to the same JSP. This will notice that data was submitted, and take appropriate actions. It should then redirect the user to another page, show a message, or allow the user to make further changes. The bundling of form and processing has multiple advantages. The first advantage is that modifications of the form or processing are easy to make. The programmer can load the source code of the JSP in an editor, make changes, and the web server will recompile the JSP. The second advantage is that all information on the modifications is kept in one place. The form keeps some information as well

as the routines to perform the actual modifications. If these were separated the editing would become more complex as multiple files need to be edited.

Modules in this category can be divided in modules that read data from the user, modules that make modifications to this data, and modules that convert the modified data to the dataset. To allow modules to access the data from the user, a variable is stored in the supporting data section of the project. This variable is stored under the key 'rawdata'.

Modules that read data from the user should store the data they read in this 'rawData' variable as a text object. This allows multiple ways of adding data by the user, and hides the way the data was obtained from other modules.

The modules that make modifications on the rawdata, read the data from the project. The user can then specify the type of modifications, such as replacing substrings by other strings. The modified data overwrites the data that was stored under the key 'rawdata'.

The conversion modules convert this raw data to a dataset that can be used by the rest of the workbench. They do this by reading the data in the rawdata variable and convert it into a dataset. The user can specify the right module for the task. If the raw data contains comma-separated values, the user should use the module that can convert this format into the dataset. These modules should also provide a simple description object for the data.

Script modules

Script modules consist of two or three parts, which are the JSP, a class inheriting from the module class, and an optional wrapper-class. This makes script modules different from the dataset and visualization modules, which only need one JSP to function. The JSP side of the module acts as a front-end to the objects of the corresponding module class. It shows all settings to the user and transfers changes back to the object. One JSP can therefore serve multiple objects from the same class. Modules that instantiate wrappers should also provide the classes needed for the wrapper.

The JSP will provide the user with a web form containing settings that can be made for the corresponding module object. It should also allow the user to couple the module to the script or to another module. When the user requests the JSP, the page should locate the requested module in the script. The settings are then read from the module, and placed in a web form. When the user makes changes and sends the data to the workbench, it will locate the module again and place the new data in it.

The classes inherit from the module class. The module class provides utility functions to create scripts. This makes it possible to connect modules to a module in a tree-like fashion. The utility functions also allow other classes to traverse the script. Searching for an object with a specific identification code, results in a call on the script object (which extends module). This will in turn query each of its children, and so on until either all modules have been queried or the module with the given code is found. For the task of locating the parent of a module in the tree a similar routine is available, in which each module checks if the given code belongs to one of its children. If so, it returns itself. The module class also specifies functions that modules should override, such as 'runDataset' and 'postProcess'. These functions are called during or after the execution of the script to perform the required operations.

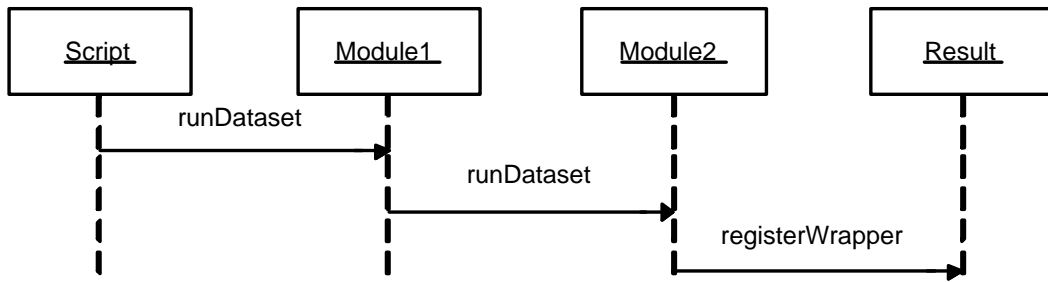


Figure 4.7: sequence diagram for running the script

The extending classes implement the specific steps that are needed when executing the script. This means that they should override the 'runDataset', and, if needed, the 'postProcess' and 'moduleIsAddable' functions. The 'runDataset' method receives the dataset that was produced by its parent module (which could be the script or another module). The dataset can then be processed according to the settings that were made by the user via the JSP. The new dataset can then be sent to child-modules (by calling the 'callRunOnSubmodules' function which is part of the module class). If the module produces wrappers, the wrappers are created filled with parameters and the dataset, and registered with the result object, which takes care of sending the wrapper to the JavaSpace.

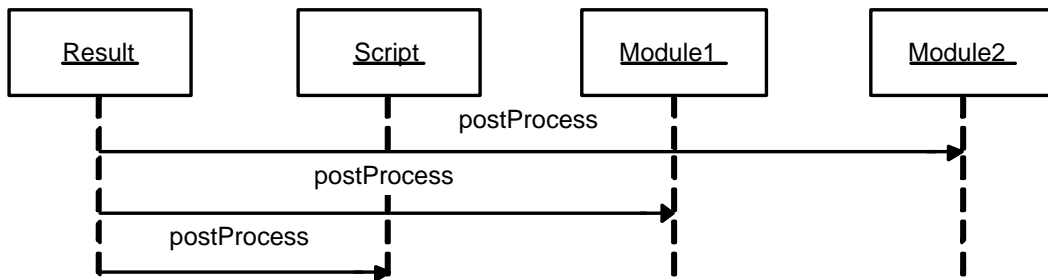


Figure 4.8: sequence diagram for postprocessing

The 'postProcess' function receives the XML and can make modifications to it if necessary. This allows modules to add data such as summaries. The 'moduleIsAddable' method is used by the workbench to determine if certain modules can be added to a module. Wrapper producing modules are endpoints for the dataset, and should therefore return 'false'. If certain combinations are not allowed, programmers can override the method and check for these specifics.

Wrappers are tightly coupled with the modules too. Wrappers should inherit from the 'Wrapper' class. This provides utility functions that wrappers might need, such as the execution of programs, as well as functions that the wrappers should override. The most important function is the 'execute' function. Each wrapper should override this function with the code needed to perform its function. When the distributed systems receive a new wrapper, they call the 'execute' function of the wrapper. The function should return a 'WrapperResult' object, which is filled with the resulting XML, identification codes, as well as additional output, exceptions and predictions. The 'WrapperResult' object is placed in the JavaSpace and

retrieved by the workbench.

JSPs that allow the user to make settings for individual features of the dataset, should request a dataset description from their parent module, and not work with the dataset description of the dataset. During execution the modules are allowed to make changes to the dataset that was passed. These changes can be anything including adding or removing examples and features. The changed dataset is passed on to child modules. The dataset that these child modules receive might therefore have changed radically from the original dataset. The use of the base dataset in all modules to set parameters is therefore not a good idea. A possibility to solve this is to request the dataset from parent modules, all the way up to the script-object, which would return the dataset that is stored in the project. If each module changes the dataset, the requesting module will receive a complete picture. Unfortunately, this approach would fail in many cases. For larger datasets, the amount of data to pump through the system could cripple performance. Additionally, the picture of the dataset that the requesting module receives is not in all cases correct. Because of the ability of modules to make more changes, propagating them to child modules multiple times with different sets of features would make the picture incomplete.

To solve these problems, not the complete dataset is requested, but only the description object. In addition, the description of features is augmented with a flag to indicate that the feature is optional or always in the dataset at that point. Modules should save their settings using the labels in this description. When the script is executed, the module can inspect the dataset to see which features are being passed. It can then use the settings to make changes. The fact that features have no numerical code helps in this respect, as it forces programmers to use the labels when referring to features.

Visualization modules

These modules provide the user with tools to visualize the dataset and the results in different ways. These modules will consist mainly of JSPs that perform some action for the user, such as showing the results or a derivative thereof. As images are very important to provide the user with good insight into the results, these JSPs will be supported by image generating Servlets.

There is not one structure that fits all modules, because of the variety in visualization modules. The overall structure will be that the user is presented with settings to make, which are submitted to the workbench. The workbench will then apply the changes, and show the results, along with the settings. These settings can then be changed to obtain different results.

4.3.7 Coupling

All the above parts need to be bound together to form a complete tool that can be used. This coupling is provided by some additional JSPs. The first is the main page, which provides an overview of all projects. When clicking a project, the name of the project is sent along with the name of the project index page. This way the page can identify the right project. All subsequent pages will need to pass the name of the project to each other. That way all pages can locate the right project in the datastore. Additional JSPs are needed to allow users to manage the projects. This means that there are pages to add and remove projects.

The project index page gives links to all modules that can be used. This way the modules are bound to the project. Clicking a link sends the user to the corresponding module. To access modules that have already been added to the script, a page showing the structure of the script can be used. The links on this page will supply the JSPs with the identification code of the module to retrieve from the script, which can then be used to fill the forms.

A special page is provided to start the script. This results in the creation of a new result and supplying the result and the dataset to the script object. To check the progress of the wrappers that are to be executed, a page has to be made to show for each of the results the amount of wrappers that were created and the amount that have been executed. These statistics are available from the result objects.

Additional pages are required to work on the results. Large tests can take a lot of space. The user should therefore be able to remove unused results. To monitor the execution, pages are provided to look at the resulting XML, the output of the wrappers, and possible exceptions.

4.4 Implementation of requirements

The requirements that the workbench should meet were described in chapter 2. In this section each of the requirements is reviewed to see if they are met by the design.

The requirement of flexibility is met easily by this design. The workbench consists of several layers at which it can be extended. Users can rewrite parts of the data structure to meet their needs better. By adding libraries of functions other programmers can create new modules. The coupling of modules allows users to create new scripts for each task, without custom programming. The addition of new modules can potentially extend the possibilities of multiple existing modules.

The amount of insight that the workbench is able to provide depends on the visualization modules. The flexibility allows new modules to be integrated easily. Basic functionality can be obtained by modules that display relations between parameters and their effects on error measures. Advanced visualization modules for specific tasks can be added later.

This design facilitates the execution of massive tests. The first reason for this is that the modules can call their sub modules for different combinations of parameters. This allows users to create tests over multiple parameter ranges with ease. The use of JavaSpaces will distribute the most time consuming parts over multiple processors, thereby decreasing the time one has to wait for all results to be available.

The combination of visualization tools and parameter ranges or choices in modules allow the user to quickly set off different hypotheses against each other. This design has no explicit facilities to test the results of the hypotheses. This means that the user is not forced to work in a certain way, but also that the workbench offers no help in the form of telling that the hypothesis should or should not be rejected. The main problem in this discussion is that the definition of hypotheses and their testing methods can vary. Users with specific definitions should find it not hard to implement modules to test hypotheses according to their definition.

The use of modules and wrappers gives users a common (user-) interface to different machine learning models. Their use is reduced to giving some parameters

and reviewing the results.

Integration of existing tools should be a breeze by using wrappers. Most tools follow one of the more standard patterns that can quickly be used in wrappers. Wrappers can easily share functions to transform the dataset to the specific format of the underlying tools.

This review shows that all requirements are met, or at least not hampered by the design. The workbench can therefore be implemented according to this design. The next chapter will look at the implementation of the workbench.

5

Implementation

This chapter will review the implementation of the workbench. The first section will briefly look at the libraries that will be used for the workbench. The section following that will describe the implementation of the framework and choices that were made. The last section will look into all modules that were built for the workbench.

5.1 Environment

The workbench is based on several platforms and makes use of some predefined libraries. The core is based on the Java 2 Platform, Standard Edition (J2SE). The version chosen is JDK 1.4.1. This is a stable version, available for all major platforms. The API for this version contains many useful packages, which would otherwise have to be downloaded separately. Examples of these packages are the 'RegEx' package, which provides classes to apply regular expressions to data, and the XML package, which provides access to DOM and SAX parsers. These packages have been introduced in version 1.4.0 of the JDK.

Apache Tomcat 4.1 was chosen for the web server. Tomcat 4.1 is the latest web server from Apache software foundation. It is open-source and freely available, and makes therefore a good choice to build the workbench on. The release implements the Servlet 2.3 and JSP 1.2 specifications. Tomcat comes with additional libraries too, implemented by developers of the Apache project. Among these libraries are Xerces, an XML parser, and Xalan, an XSLT Processor, which are both considered to be among the best implementations available.

To support JavaSpaces, the Jini Technology Starter Kit v1.2.1 was chosen, which is the reference implementation of this technology, and contains everything to get the JavaSpace and all supporting services running.

5.2 Framework

This section will look at the parts of the framework. The framework of the workbench consists of all parts that allow the modules to interact. It consists of the data structures as well as some JSPs that are needed to tie modules together.

The following sections will not describe the code in detail. Only parts that were implemented different from what the design described, or implementation issues will be discussed. Most of the parts of the data structure were implemented as the design required. Only some parts that were open for multiple ways of implementation or fixes of the design that resulted from actually working with the workbench were implemented differently and those choices are described here.

The first subsection looks at the datastore, and describes the storage method that was chosen and the enhancements. The second subsection looks at the webformstore, which is a utility to ease the creation of JSPs and modules. The distribution of wrappers over multiple clients is discussed in the third subsection. The fourth subsection looks at smaller changes in the design.

5.2.1 Datastore

As mentioned in the design, the datastore is the centre of the workbench. It stores all data needed to run tests and provide the user with results.

The following sections look at choices and changes that were made to enhance the initial design of the datastore. The first section will describe the choice for serialization over a database management system. The second section discusses some enhancements to the serialization, to allow large tests to be performed. The third section reviews some small additions.

Serialization vs. Database Management System

The design described the data in the workbench as a tree-like structure of objects. To allow the data in this structure to persist, it needs to be kept on disk. The first way to do so is serialization. Serialization is a way to transform data in objects in a structure into a flat binary file that can be used to recreate the structure and fill it with data again. The considered alternative was the use of a Database Management System (DBMS). A DBMS is the complete set of database and the functions to access and manage the data in the database. Access to data performed by asking the DBMS for pieces of the structure.

Both methods have their own advantages and disadvantages. The storage and retrieval of data using serialization is based on loading the complete object structure into memory, making modifications and serializing the objects to disk. A DBMS allows the system to operate on a much finer scale. Every piece of information is accessible by a separate query on the database. This makes the storage and retrieval more efficient. An added advantage is that a DBMS can allow filtering, thus only returning a subset of the data, thereby saving the programmer from constructing code to filter results.

An advantage of serialization over the DBMS is that the programmer does not need to create separate queries on the database, and that all information on the data in the object is only specified in the object. The Java specification defines how to serialize objects, so that the only thing users have to do is to set a flag that the object is allowed to be saved. All primitive data types in the object are automatically

saved, and objects that are referenced from the object are automatically serialized if they were not already. When using a DBMS, the user would need to create database structures for every module. This results in a proliferation of the information on the module.

When transferring modules from one user to another, the database structures need to be transferred too. This would result in more work for the user, especially when he just wants to try a new module. A possible solution would be to incorporate the specification of the database structure in the modules. The workbench could then retrieve the structure and recreate it if needed. This solution would also alleviate the proliferation of definitions throughout the code.

There are three major grounds for choosing serialization over the DBMS, all based on ease of use. The first is that by using serialization, the module programmer does not need to be concerned with the physical details of the storage and retrieval. The workbench can save the objects at regular intervals, and retrieve them when data is needed. The second reason is that the use of a DBMS would require all module programmers to know the DBMS language. For most databases this would be SQL. Although SQL is a standard language, not all programmers have sufficient knowledge about it, and this would create a barrier to the use of the workbench. The last reason is that to use a DBMS, yet another program needs to be installed and configured. This would heighten the barrier to start working with the workbench.

Storage and retrieval

Some modifications were made to improve the initial design where the complete datastore is loaded and saved each time a page had been opened. The first modification was to save each project separately in its own file. The potential amount of data that would need to be saved each time was reduced drastically. Only active projects are retrieved when needed, while the inactive projects are kept on disk. The second modification was to keep a cache of loaded projects. If a project is already in cache it does not need to be loaded again, and it can be used immediately.

The main problem when loading and saving projects for multiple users (or one user who is working with multiple web pages at once) is that the files can become inconsistent. This happens if each process loads the file at the beginning, and writes it back again when it is done. If one (slow) process takes a long time, and another (fast) process modifies the data in between, the modifications of the fast process are lost. It would be better if both processes were working on the same object structure. A cache can help in this respect: projects are first located in the cache, and if it is not in the cache, it is loaded from disk and then placed in the cache. Processes ask the cache for the project, and work on the same instance. To reclaim the memory taken by the project, it must be removed from the cache, or else the garbage collector, which is part of Java, cannot reclaim the memory the project takes. The problem is that the removal from the cache does not mean that there are no processes still processing the project. The web server has multiple threads processing queries from users. Each of the processes might still have a reference to the project. Keeping counters on the number of threads that are working on the project might fail: threads can terminate prematurely, or programmers might forget to let the code update the counters.

Two features of Java were used to implement a more elegant algorithm. The first is the use of the 'SoftReference' class. This is a class that holds a reference

to an object. However the object may be garbage collected (which is not the case with normal references). By placing these `SoftReferences` in the cache instead of the references to the projects, the projects can be removed from memory when no processes are using it anymore. The other feature was used to make sure that projects are saved before they are removed from memory. This is implemented by overriding the finalization function. When a project is to be garbage collected, the garbage collector will call the finalization function. This gives the project a chance to request the datastore to save the project. After the storage, it is removed from memory. The only weakness in this process is that it is not mandatory to call the finalization function when the virtual machine is terminated. To prevent a possible loss of data, all projects in the cache are saved at a regular interval.

Very few changes to the modules were necessary to incorporate the above enhancements because of the structure of the workbench where the data in each project is accessed through the datastore.

During tests with the workbench the amount of large result objects became a problem as most results were not used but were constantly being loaded from and saved to disk because they are part of the project object. As some of the results had data in the order of tens of megabytes this quickly became a problem. The same routine was employed to allow results to be stored separately from the rest of the data. To do so, the datastore was enhanced with some functions. The first function locates objects in the cache, or if it is not available there the object is loaded from disk. A second function saves the object to disk. A third function searches the disk for objects of a certain type within a certain project. These modifications are generalized to allow all types of objects that are related to projects to be saved independently.

The results class was modified slightly to incorporate the loading and saving. Instead of keeping references to the results, all calls for those results were redirected to the datastore. The result class was also changed to incorporate a finalization function to automatically save the object when the garbage collector tries to reclaim the memory.

To further improve the stability, backups are kept of the last save. This is done in a three-step process. First the old backup is removed. Then the last save is then renamed to create the new backup. Only then a new file is created with the latest data. This prevents failures if the server is terminated at the moment that a file is written to disk. If the file being written is read back, the process will throw an exception because the file is incomplete and the objects cannot be reconstructed. If this happens, the datastore tries to read the backup. Only if this fails too, the datastore will throw an error to inform the user that something is wrong with the reading process.

Additions

Because the datastore is always available once the user has started the workbench, this is the place to start some service threads. The workbench has two of these. The first of these threads is to save projects and their files at regular intervals. To do so, the thread iterates over the cache. Each entry that has been removed from memory (which is indicated by the `SoftReference` returning a null-object) is also removed from the cache. Otherwise the functions to save the objects are called.

The second thread is the integrator thread. It requests all finished wrapper results. Each of these results is sent to the corresponding result object to be integrated

with the result object.

5.2.2 Webformstore

The webform store is a small part of the system designed to facilitate the creation of the web forms for script-modules. Because of the tight coupling of a front-end, the object containing the module and possibly a wrapper, descriptions of the needed data are distributed over all these parts and need to be kept synchronized. The webform store is an attempt to ease this.

In the situation without the webform store, there are many places where information on the used variables is stored. The first is the place where variables are defined in the object. Then there might also be several get- and set- methods in the object to access each of the variables. The JSPs follow the structure that one shows the form-elements such as textboxes and drop down lists, validates the data, and finally stores the data if it is valid. Each of these parts of this structure contains some information on the variables. A minor change would require the programmer to check all those places to see if modifications are needed at that place.

The webform store solves this by concentrating all information on the encapsulated data in one place: the webform. All forms consist of a collection of known elements such as input boxes for numbers, check boxes and drop down lists. The programmer can now use the webform store to output HTML code for those elements, and supply the webform store with additional information on data checks at the same time. This results in a situation where there is no need for explicit data checks or storing of data in the object.

All variables registered with the webform store are automatically updated when the form is submitted. The webform store also stores the additional information provided in the form. This additional information can later be used for data consistency checks. Some of these checks are inherent to the element used. A numerical value, for example, will be tested to see if it is really a number. But the programmer can also supply the webform store with boundaries for the value, or directions that it should only apply tests to the value if a certain checkbox is checked.

When all variables pass the tests they can be stored in the module, so that it always has a valid set of variables. To save only the variables, and not all additional information that the webform store needs, the variables are stored in a `WebformVariables`-object, which provides extra services to the module, such as transforming the saved data to suitable primitive data types, such as doubles for numbers and Booleans for checkboxes.

The number of places where information on the variables resides is reduced to the form and the usage of the variables by the module. This should make maintenance of the modules easier.

5.2.3 Wrapper distribution

The wrappers created by the modules are distributed over multiple clients. The first choice to do so was by using JavaSpaces. After the implementation of this part it became clear that installing Jini services and starting a JavaSpace can be a problem, especially if one would just like to test the workbench to see if it suits the needs.

Therefore an abstraction was introduced, called the wrapper schedulers. Each wrapper scheduler represents a certain way of scheduling. A basic scheduler keeps

a queue of the wrappers that are to be executed and executes them one by one. The Jini scheduler is more sophisticated and distributes the wrappers over multiple clients. A wrapper scheduler factory is used to hide the creation of these schedulers from the user.

Wrapper scheduler factory

The wrapper scheduler factory is an object adhering to the factory design pattern [Gamma et al., 1995]. This pattern has been made to ease the switching of objects. In the case of the workbench and the wrappers this means that the factory has the ability to create a wrapper scheduler that suits the needs of the user.

The factory asks the class loader (which is part of the Java Virtual Machine) for some classes specific to Jini. If successful, it tries to initiate the Jini scheduler. If this succeeds too, the Jini scheduler is selected as the scheduler to use, and every call for a scheduler will result in this scheduler being returned. If either the class lookup or the initialisation of the Jini scheduler fails, the factory will create a local execution scheduler.

The pattern used can also be extended for other schedulers. If a new technique would arrive that is able to run the wrappers, but is based on different architectures, the new scheduler would be easy to integrate. The workbench could switch to the new scheduler by making some additional checks. This switch would be transparent from the modules, which only communicate to the schedulers via an interface provided by the factory.

Jini scheduler

The Jini scheduler adheres to the 'WrapperScheduler' interface, and therefore implements several functions: adding wrappers, retrieving results, and terminating one or more wrappers. It also sets up a connection with the JavaSpace where the wrappers will be stored.

To use the JavaSpace scheduler, all services needed must be running. The first service is a small web server aimed at the distribution of JAR files (which are files containing class files). These classes are needed to create the proxies to the services. The next service is the RMI registry, which takes care of the registration of services. The third service is the service locator, which allows processes to find services. The next is the transaction manager, and the fifth is the JavaSpace service.

The addition of wrappers involves only slightly more than posting the wrapper to the JavaSpace (which only involves calling a service function on the proxy of the JavaSpace). The only additional step is to encapsulate the wrapper in an entry, which provides information on the wrapper to the clients. The encapsulation is also needed to shield wrappers from some issues related to entries (which need to have all attributes set as public, because else the field would not be transferred).

Retrieving results consists of some simple steps. The first is the creation of a template to match. This means that an object of the encapsulating class has to be instantiated, and filled with the values that must be matched. The second step is to query the JavaSpace by sending it the template. The JavaSpace will then try all objects in the space to see if they match the class of the template, and have all not-null variables equal. The first matching result is returned to the scheduler and removed from the JavaSpace. The scheduler returns the object to the calling function.

Termination is needed to stop tests that take too long to finish. To do so, all objects in the JavaSpace that match the criteria are removed. Then a 'TerminateRequest' object is placed in the JavaSpace to notify clients to terminate any running jobs that match the criteria. The leasing mechanism is used to ensure that the 'TerminateRequest' object expires after thirty seconds, which should be enough to notify all clients.

Execution of wrappers is performed by the clients, which start an instance of the object 'WrapperWorker'. Instances of this class connect with the JavaSpace and look for entries of the wrapper type. These are taken from the space and started. Results returned by the wrapper are wrapped in another entry object and placed in the JavaSpace for the workbench to retrieve. The WrapperWorker also starts another thread which monitors the JavaSpace for terminate requests. If such a request is received, and it matches the current wrapper that is being executed, a request to terminate is sent to the wrapper. The wrapper is then to take appropriate actions, such as killing the command-line tool.

Local execution scheduler

The local execution scheduler was created to simplify the installation of the workbench. It is always possible to use this scheduler as it is based on only the most basic classes in the Java architecture.

The process of adding wrappers, retrieving results and terminating jobs is essentially the same as with the Jini version. When placing wrappers in the queue, the scheduler makes a deep copy of the wrapper and places the copy on the queue. This is essentially the same as the Jini scheduler that transfers the values to the JavaSpace. The copying is needed to make sure that modules cannot make changes to the wrapper that is placed on the queue anymore.

On initialisation of the scheduler, it starts itself as a thread, waiting for wrappers to be entered in the queue. If wrappers are available, it will execute them asynchronous from the workbench, one wrapper at a time.

Termination of wrappers is performed by first removing all wrappers that are in the queue and that match the criteria. Then the wrapper that is currently being executed is sent a request to terminate.

5.2.4 User-interface module enhancements

The creation of the user-interface to modules requires some code. Some of this code is common to all modules. These parts are factored out and combined in the 'UIModule' class. To facilitate the addition and removal of modules to the workbench, without modifying existing parts, the 'UIModules' class was created. Both classes are described below.

UIModule

The 'UIModule' class contains methods that facilitate common tasks in JSPs, with regard to the coupling of JSP and corresponding object. The user can obtain an instance of the UIModule class by requesting one from the 'UIModules' object, which is discussed in the following section.

The obtained instance should then be initialized. This initialization checks if the right module code is given to the JSP and this code is used to obtain the corresponding 'Module' object. It also initializes a new 'WebformStore' object.

In the JSP the user can call methods of the initialized 'UIModule' to check the submitted variables, and to save the variables to the underlying 'Module' object. The 'UIModule' also supplies the user with a function to obtain a good dataset description object.

UIModules

When a new module is added to the workbench, it is cumbersome to modify all the pages where references to modules are shown. To ease the addition of modules, the 'UIModules' class was created, which works in close cooperation with 'UIModule' objects.

This object is created as a singleton. There is only one instance, and that can be obtained by calling a static function on this class. All calls on the methods of this instance result in a call on a method that locates all modules.

All modules should be placed in one of three directories. There is a directory for each of the types: dataset modules, script modules and visualization modules. The 'UIModules' object traverses these directories to see if there are any new modules, based on the time-stamp of their JSP-file. When a new module is found the file is read and a header is located. This header is implemented within Java comment-tags. From this header instructions are retrieved and stored in a new 'UIModule' object. These instructions can contain the title of the module, a description, or necessary classes, such as the name of the corresponding 'Module' class.

The pages that make up the structure of the workbench, can ask the 'UIModules' instance for a list of all modules in a directory. A list with 'UIModule' objects is then returned. Because these objects contain the descriptions, the JSPs can show the user a list with modules.

This object also allows conversions from the name of a module or a 'Module' object to a 'UIModule' object.

5.3 Modules

The modules fill the framework with functionality to create and perform tests, and to process the resulting data. This is the part of the workbench where users can add parts to extend the functionality.

Although all modules share common characteristics, there are some distinctions between them to divide them into three groups. This is closely related to the complete structure of the workbench, and the steps the user takes during normal use. The dataset modules work primarily on the dataset or variables that contain the future dataset. The script modules consist of multiple files, one for the user interface and one for the workbench to use internally. There can also be an additional wrapper with these modules. The visualization modules vary in their complexity, but make no changes to the existing information, only derivatives.

The following sections will follow the distinction. The first section describes the way most modules work. The sections thereafter will discuss the dataset, script and visualization modules that are incorporated in the workbench.

5.3.1 Basic module structure

Because JSPs are web pages, they follow a standard pattern. This is basically a non-interactive structure. The JSP receives input data in the form of a list with (submitted) variables. The JSP will take appropriate actions based on the values of these variables. The results are sent back as an HTML page to the user.

Interactions with pages are created by first showing the user some information and options that can be changed. When the user submits the changes to the JSP (by clicking a button), the JSP can apply the new values that were submitted and provide the user with the results and a new page with options. This repeating process creates an interactive flow of information between the user and the workbench.

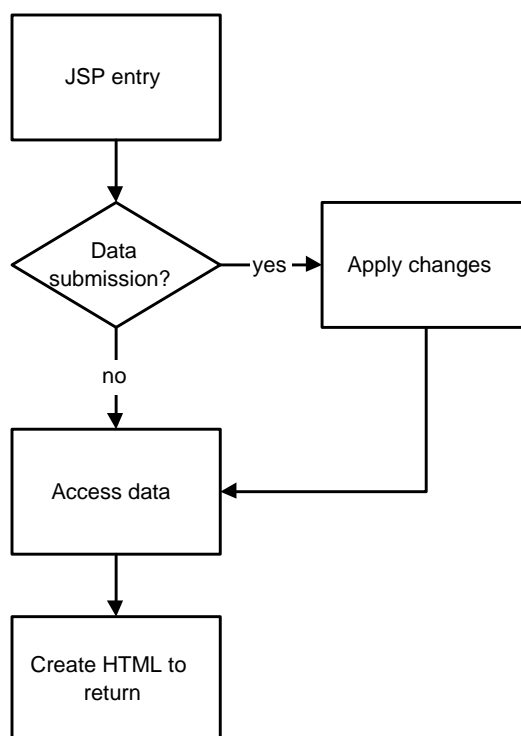


Figure 5.1: basic module process

The basic structure in these pages is most commonly a series of code blocks. The first block checks the submitted values. Actions are performed in this block on basis of these values. The next block retrieves the data that is needed to return a new page to the user. The third block transforms this data into a HTML page, possibly with links or forms that can be used to provide the JSP with new information.

The dataset and visualization modules will adhere loosely to this structure. The modules can have many series of blocks, for different steps in a process. An example is a JSP that first shows a form with initial settings. After submission of the settings, another block is used to show more advanced options that were dependent on the first settings. The script modules adhere very strong to the above structure, which is further discussed in section 5.3.3.

Figure 5.2 shows the basic layout of a module (in this case the 'LibSVM' script-module). This figure shows the web browser containing a module. The purple column is the menu bar. All common tasks are reachable from this menu. The

yellow part is different for each page and module. The common layout for these pages is to show the title of the page or module the top. Below the title a short description is given, along with the form with elements.

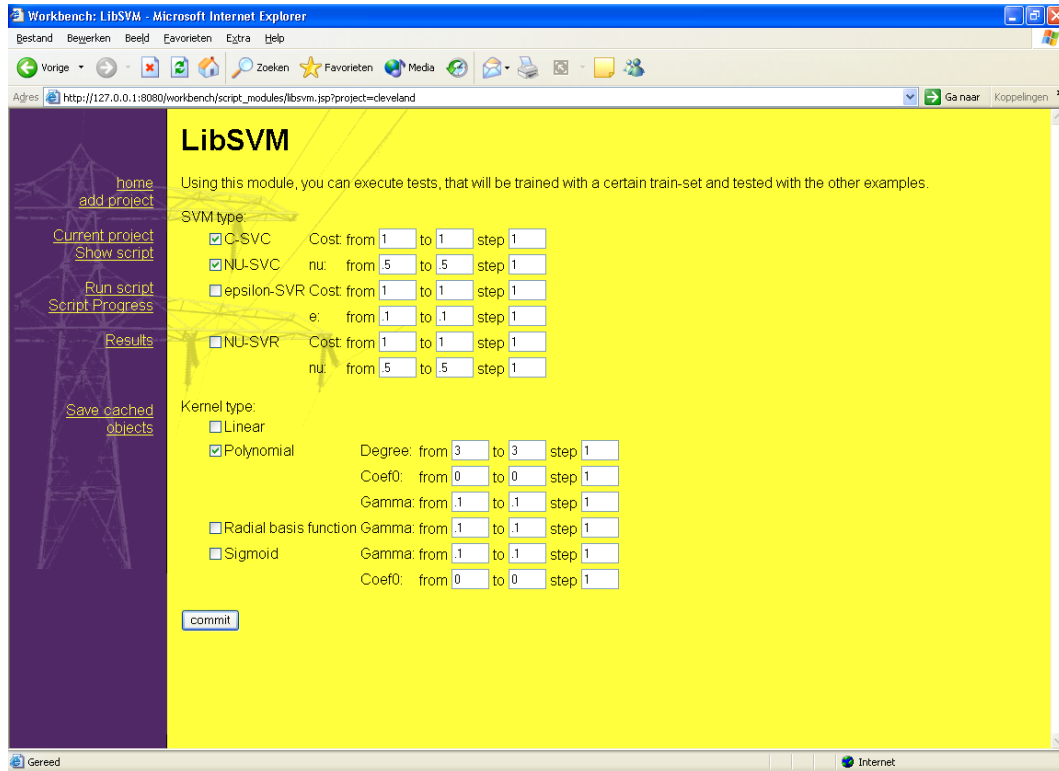
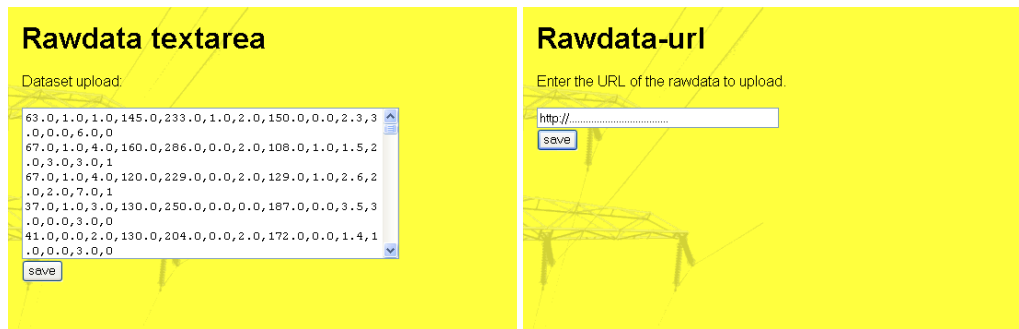


Figure 5.2: basic module interface

5.3.2 Dataset modules

The dataset modules transform the dataset into a format that the other modules can use. The design of the dataset modules was implemented without significant changes. Some modules insert data into the rawdata variable. Another series of modules transform this data. The data can be converted to a dataset by yet another module. To make further changes to the completed dataset another set of modules is provided.

The basic process in these modules is the same as sketched before. The module first shows a form with some options, or textboxes. When the user submits the form, the module processes the new data. The module can then send the user to an intermediary page, which displays a message that the action was performed before returning the user to the list with options. Alternatively, the user can be shown the form again, which might be useful in some situations. The choice for these two alternatives depends on the programmer's idea of the interaction pattern of the user with the module.

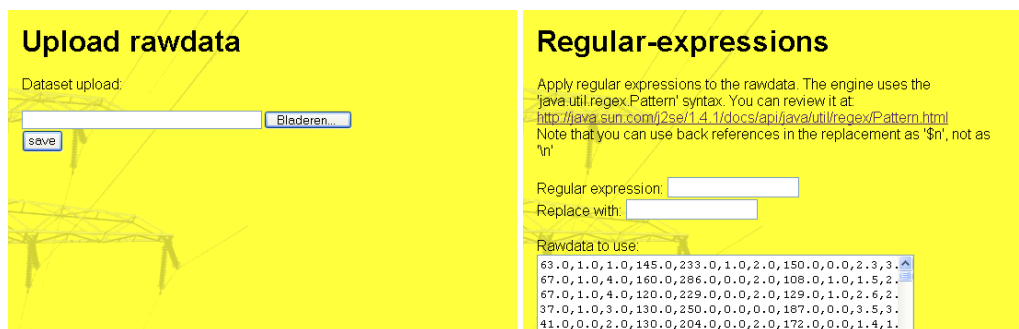


Rawdata textarea

This is usually the first module a user would use. The module presents the user with a textarea, where the user can put his data. The data is put into the datastore in the list with supporting data, under the name 'rawdata', when the form is submitted. This data can then be used for further processing.

Rawdata URL

When datasets are larger, or available over Internet, the 'rawdata URL' module can be used to insert the data into the workbench. The user enters an URL, which can point to a file on an FTP-server or HTTP-server, and submits it to the workbench. The workbench will then attempt to read the data, and insert it into the datastore (under the name 'rawdata').



Upload rawdata

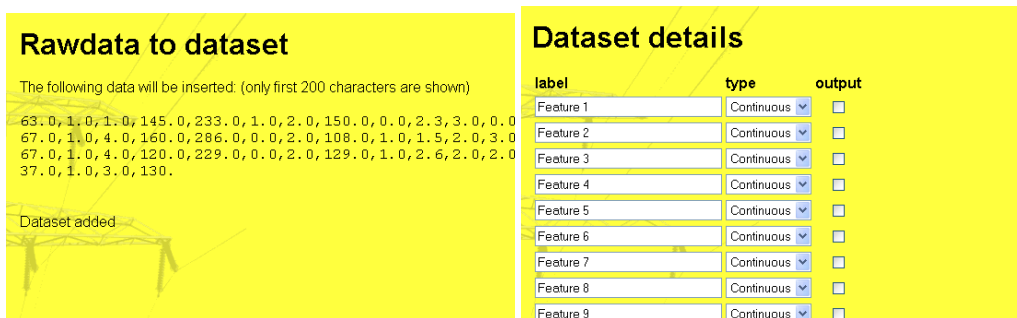
When the data is available on a disk, it is also possible to upload the data. The user is presented a screen where he can enter the location of the file, or use a dialog to choose the file that needs to be uploaded. This file is then transferred to the workbench and stored in the datastore.

To receive the file, a filter needed to be installed into Tomcat. To allow web browsers to send files, the encoding of submitted data has to be changed. Tomcat cannot handle this encoding by default. A free filter was obtained which translates the data into a format that can be used by the Tomcat and the workbench.

Regular expressions

Data inserted in the rawdata variable need not be ready to be converted into a dataset. There might be additional data that needs to be erased or the separation character is not right. The regular expression module is a powerful tool for the user to make these changes. It shows the user two textareas. The first is the rawdata that is used. The user is allowed to make changes directly on this data (without going through the 'Rawdata textarea' module). The second textarea shows the results of the regular expression, without storing it directly in the rawdata variable. This allows the user to test the expressions and review the results before committing them.

The regular expressions are performed by the Java package bundled with JDK 1.4.1. This provides the workbench with Perl-like regular expressions [Friedl, 2002].



Rawdata to dataset

The following data will be inserted: (only first 200 characters are shown)

```
63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0
67.0,1.0,4.0,160.0,286.0,0.0,2.0,108.0,1.0,1.5,2.0,3.0
67.0,1.0,4.0,120.0,229.0,0.0,2.0,129.0,1.0,2.6,2.0,2.0
37.0,1.0,3.0,130.
```

Dataset added

Dataset details

label	type	output
Feature 1	Continuous	<input type="checkbox"/>
Feature 2	Continuous	<input type="checkbox"/>
Feature 3	Continuous	<input type="checkbox"/>
Feature 4	Continuous	<input type="checkbox"/>
Feature 5	Continuous	<input type="checkbox"/>
Feature 6	Continuous	<input type="checkbox"/>
Feature 7	Continuous	<input type="checkbox"/>
Feature 8	Continuous	<input type="checkbox"/>
Feature 9	Continuous	<input type="checkbox"/>

Rawdata to dataset

This module converts the data in the rawdata variable to a dataset that can be used by the workbench. The values in the data need to be separated by commas, and one example per line. The data is parsed into a two dimensional array of values, which is inserted into the dataset object. This data is augmented with a dataset description object. The default values in this object give the features a sequential number, and label all features as being continuous, with the last value being the output.

Dataset details

This module is used to change the details of the dataset. It shows all features available in the dataset, and allows the user to change the name, type and indicate if it is an input or output. The type is a label that indicated that the feature is either continuous, multiclass or binary. The data that the user enters is stored in the dataset description object in the dataset. This data can be used by other modules to take appropriate actions.

View dataset

The data that is stored in the dataset might be different from the data in the rawdata. This module can be used to quickly see the data in the dataset, and if the

View dataset		Normalize a feature	
63.0	1.0	1.0	145.0
67.0	1.0	4.0	160.0
67.0	1.0	4.0	120.0
37.0	1.0	3.0	130.0
41.0	0.0	2.0	130.0
56.0	1.0	2.0	120.0
62.0	0.0	4.0	140.0
57.0	0.0	4.0	120.0
63.0	1.0	4.0	130.0
53.0	1.0	4.0	140.0
57.0	1.0	4.0	140.0
56.0	0.0	2.0	140.0
56.0	1.0	3.0	130.0
150.0	0.0	0.0	2.3
108.0	1.0	1.5	2.0
129.0	1.0	2.6	2.0
187.0	0.0	3.5	3.0
172.0	0.0	1.4	1.0
178.0	0.0	0.8	1.0
160.0	0.0	3.6	3.0
163.0	1.0	0.6	1.0
147.0	0.0	1.4	2.0
155.0	1.0	3.1	3.0
148.0	0.0	0.4	2.0
153.0	0.0	1.3	2.0
142.0	1.0	0.6	2.0
2.3	3.0	0.0	6.0
1.5	2.0	3.0	3.0
2.6	2.0	2.0	7.0
3.5	3.0	0.0	3.0
1.4	1.0	0.0	3.0
0.8	1.0	0.0	3.0
3.6	3.0	2.0	3.0
1.0	0.6	1.0	0.0
1.4	2.0	1.0	7.0
3.1	3.0	0.0	7.0
0.4	2.0	0.0	6.0
1.3	2.0	0.0	3.0
0.6	2.0	1.0	6.0

	Average	St.dev
Normalize 'Feature 1'	54.54208754208754	9.049735681096765
Normalize 'Feature 2'	0.6767676767676768	0.46849996744100164
Normalize 'Feature 3'	3.1582491582491583	0.9648594099420302
Normalize 'Feature 4'	131.69360269360268	17.762806366598998
Normalize 'Feature 5'	247.35016835016836	51.99758253513897
Normalize 'Feature 6'	0.1447811447811448	0.35247393412545347
Normalize 'Feature 7'	0.9966329966329966	0.9949138102637306
Normalize 'Feature 8'	149.5993265993266	22.941562061360813
Normalize 'Feature 9'	0.3265993265993266	0.4697608121961864
Normalize 'Feature 10'	1.0555555555555558	1.1661227818468953
Normalize 'Feature 11'	1.6026936026936027	0.6181867696375288
Normalize 'Feature 12'	0.6767676767676768	0.93896452630245

transformation by the 'Rawdata to dataset' module was performed right. It shows a matrix with all values in the dataset.

Normalize feature

Most machine learning algorithms work better with features that have their values near zero, possibly normalized. This module allows the user to choose a feature and normalize it (with mean zero, and standard deviation one).

Shift outputs	Sum rows
<p>This module is used to shift the outputs one place back. This allows one to use a dataset that contains events to be transformed into a set that can be used for prediction.</p> <p>shift</p>	<p>Amount of rows to sum:</p> <p>2</p> <p>sum</p>

Shift outputs

Many datasets are not prepared for prediction purposes. These sets are tables of all data for a certain moment. For example: the amount of light, the temperature and the use of energy. To transform such a dataset into a dataset that can be used for prediction, the feature to predict should be shifted one or more places. The shift outputs module shifts all output features one place upwards, and truncates the complete dataset by one row. The truncation is needed, as the last example will miss output values. The output values of the first example are lost (as these cannot be combined with input features).

Sum rows

Sometimes the data contained in the data-set has a wrong time sampling. For example, if data were available for 15--minute intervals, but one would like data for hourly intervals. This module adds rows of the dataset together to create a new dataset. The user can specify the number of rows to sum.



```

Datastore info
vars
rawdata 63 0,1 0,1 0,145 0,233 0,1 0,2 0,150 0,0 0,2 3,3 0,0 0,6 0,0
67 0,1 0,4 0,160 0,286 0,0 0,2 0,108 0,1 0,1 5,2 0,3 0,3 0,1
67 0,1 0,4 0,120 0,229 0,0 0,2 0,129 0,1 0,2 6,2 0,2 0,7 0,1
37 0,1 0,3 0,130 0,250 0,0 0,0 0,187 0,0 0,3 5,3 0,0 0,3 0,0
41 0,0 0,2 0,130 0,204 0,0 0,2 0,172 0,0 0,1 4,1 0,0 0,3 0,0
56 0,1 0,2 0,120 0,236 0,0 0,0 0,178 0,0 0,0 8,1 0,0 0,3 0,0
62 0,0 0,4 0,140 0,268 0,0 0,2 0,160 0,0 0,3 6,3 0,2 0,3 0,1
57 0,0 0,4 0,120 0,354 0,0 0,0 0,163 0,1 0,0 6,1 0,0 0,3 0,0
63 0,1 0,4 0,...TRUNCATED, total length=18108 characters.
visual_xslt {2 - svmt - sigmoid={additionalpaths=[[Ljava.lang.String.@f2c96c
[[Ljava.lang.String.@15780d9, [[Ljava.lang.String.@14b2f1a,
[[Ljava.lang.String.@2a5ab9], mainpath edit=[[Ljava.lang.String.@

```

Datastore info

This module allows the user to inspect the supporting data in the datastore. It shows all variables that are currently set, and their values. Objects of the string type that are too big (for example a big dataset) are truncated, so that only the first part (to give an indication of the contents) and the total length of the data is shown.

5.3.3 Script modules

As noted before, the script modules adhere strong to the basic module structure. Each JSP has a connection to the corresponding object that is relevant to the module. If the request is to create a new module to add to the script, a new object is created. The module to which the new module is tied is indicated by a parameter that is passed by a link that the user clicked.

The first block in the JSP is to see if the user submitted the information. If this is not the case the block is skipped. The next block fills the webform store with the data from the module object that is requested. Again the JSP 'knows' which module to use by looking at a parameter with as value the identification code of the object. The last block displays the form. In this form all values that were retrieved from the object are displayed. These data can be changed by the user.

When the user submits the form, the first block is activated. This block first looks at the submitted information to see if the data are valid. Most of these checks are performed by the webform store, but addition consistency checks can be made. If the data are valid, the JSP locates the corresponding object, or creates a new one if it does not yet exist. The object is filled with the new information. The datastore takes care of the storage of the data on a persistent medium.

The use of the 'UIModule' and 'webformstore' classes reduced the blocks to a small number of lines, by factoring all common code out of the modules, and into these classes.

The process of running scripts and plugins was implemented according to the design.

Add constant

This module allows the user to add a constant to features. The constant to add to each of the features can be specified as an interval. This means that the user specifies the start, end and step size. When the script is executed, each value in the interval is tried. If the user specifies a start value of 10, end value of 20 and step size 5, this means that the sub modules are called 3 times, one time with 10 being

Add constant

Choose intervals of constants to add to the features. All combinations are applied.

age	constant from	0	to	0	step	1
sex	constant from	0	to	0	step	1
cp	constant from	0	to	0	step	1
trestbps	constant from	0	to	0	step	1
chol	constant from	0	to	0	step	1
fbs	constant from	0	to	0	step	1
restecg	constant from	0	to	0	step	1
thalach	constant from	0	to	0	step	1

Noise adder

Using this module, you can add noise to each of the features. You can add different types of noise: uniform and gaussian. For the uniform noise, noise in the range [-amount, amount] will be added to the selected feature. Gaussian noise is added to features with the specified mean and standard deviation (stdev).

age	<input type="checkbox"/> uniform	Amount from		to		step	1
	<input type="checkbox"/> gaussian	Mean from		to		stdev from	
							step 1
sex	<input type="checkbox"/> uniform	Amount from		to		step	1
	<input type="checkbox"/> gaussian	Mean from		to		stdev from	
							step 1
cp	<input type="checkbox"/> uniform	Amount from		to		step	1
	<input type="checkbox"/> gaussian	Mean from		to		stdev from	
							step 1
trestbps	<input type="checkbox"/> uniform	Amount from		to		step	1
	<input type="checkbox"/> gaussian	Mean from		to		stdev from	
							step 1
chol	<input type="checkbox"/> uniform	Amount from		to		step	1
	<input type="checkbox"/> gaussian	Mean from		to		stdev from	
							step 1

added, one time with 15 and another time with 20. If the user specifies multiple intervals (one for each feature) all combinations will be tried.

Noise adder

This module is used to add noise to features. The user is presented with fields for each of the features. These fields let the user choose which type of noise to add (Gaussian or uniform), and the corresponding parameters. Each of the parameters is an interval where the user can specify the starting and end points as well as the step size. This means that sub modules are called for each combination of parameters. This should be used with great care, as it is easy to start a huge number of tests by just changing some numbers.

This module acts according to the normal interaction pattern of JSP and object. Each time it is invoked it locates the right 'UIModule' object. Then a description of the features in this point in the script is requested. For each of the features some lines are presented to the user: one for each noise type. The values of the form elements that are returned to the user, result from the 'WebformStore' object that was created by the UIModule. When the user saves the form, the UIModule is located again and the WebformStore is requested to check the values. If these are all right, the UIModule is asked by the JSP to save the values in the object.

When the script is executed the 'runDataset' method of the module is called with a dataset and result object. To make sure that modifications on the dataset will not interfere with preceding modules, a copy is made that will be passed to sub modules. For each feature in the dataset is checked if the user made settings for that feature. If so, the new dataset is modified by applying calculations to the original data and storing the results in the new dataset.

The checking of settings and calling of sub modules is structured in such a way that all combinations of settings are used. A recursive function runs over all features one by one. If settings indicate that a feature needs Gaussian noise, a loop over the interval is performed. For these parameters the function is called again, with the request to process the following feature, until there are no features left. Then all sub modules are called, and control is returned to the last function, which can then try another combination (or if the combinations are exhausted, it returns control to its caller, and so on).

Normalize/scale

The aim of this module is to allow users to specify scaling or normalization intervals. The module works the same way as the noise adding module, only the func-

Normalize/scale

Using this module, you can normalize/scale each of the features.
When normalizing, the standard deviation to use can be specified.
For scaling a minimum and size can be specified. All values will lie between the minimum and minimum+size.

age normalize stdev. from 1 to 1 step 1
 scale min from 0 to 0 step 1 size from 1 to 1 step 1
 subtract, then divide subtract 0 divide 1

sex normalize stdev. from 1 to 1 step 1
 scale min from 0 to 0 step 1 size from 1 to 1 step 1
 subtract, then divide subtract 0 divide 1

cp normalize stdev. from 1 to 1 step 1
 scale min from 0 to 0 step 1 size from 1 to 1 step 1
 subtract, then divide subtract 0 divide 1

trestbps normalize stdev. from 1 to 1 step 1
 scale min from 0 to 0 step 1 size from 1 to 1 step 1
 subtract, then divide subtract 0 divide 1

chol normalize stdev. from 1 to 1 step 1

example subset

Using this module, you can select a subset of the available examples.

Rules are executed in the order they appear. Examples that are 'allowed' by some to the next module, examples that are 'denied' are not tested further either, and removed.

Rule 1 allow age < 50 remove
 Rule 2 allow trestbps != 1 remove
 Rule 3 allow oldpeak = 1 remove

Default allow
[add new rule](#)

tion performed on the features is different and it uses different variables. For each feature the user can specify that the features must be scaled and/or normalized. For scaling, the user can specify the boundaries (defined by the minimum value and the size of the interval). For normalization the user can specify the standard deviation. The user can specify intervals for all values. The module will create all specified combinations and pass those to sub modules for further processing.

The XML contains the settings that were used, such as the method and the parameters. In addition, the formula that is used to transform each of the values for a feature is shown. This formula can be used to transform data when the machine learning algorithm is used outside the workbench in a real-world situation.

Example subset

Not all examples are always wanted in the dataset. Sometimes the prediction rates can be increased enormously by dividing the examples according to some criteria, and then predicting only that particular subset. For example, creating separate prediction routines for several age groups might improve prediction ratios.

To make the subsets, the module allows the user specify rules. Each rule consists of the specification of the feature, a value and an operator to apply to the feature and the value (equal, greater-than etc). If the rule matches, it will perform the action that the user specifies: accept the example or deny it from the dataset. If the rule does not match, the next rule is applied to the example. If none of the rules match, the default action specifies to accept or deny the example. After applying the rules to all examples, the dataset is passed to the child modules.

feature subset

Using this module, you can select a subset of the available features.

age
 sex
 cp
 trestbps
 chol
 fbs
 restecg
 thalach
 exang
 oldpeak
 slope
 ca
 thal
 disease

time series embedding

This module allows you to repeat values from older examples.
Because the first examples lack historical data, they will be removed from the dataset.

Direction NL from 0 to 0 step 1
 Direction DeKooy from 0 to 0 step 1
 Direction Hoogeveen from 0 to 0 step 1
 Direction Leeuwarden from 0 to 0 step 1
 Direction Schiphol from 0 to 0 step 1
 Speed NL from 0 to 0 step 1
 Speed DeKooy from 0 to 0 step 1
 Speed Hoogeveen from 0 to 0 step 1
 Speed Leeuwarden from 0 to 0 step 1
 Speed Schiphol from 0 to 0 step 1
 Production from 0 to 0 step 1

Feature subset

The dataset might contain more features than needed at a certain moment. This module is used to remove the features from the dataset, without modifying the original dataset. The user is shown all features (including ones that are not in the original dataset, but were created by parent modules), and allows the user to check the ones that he would like to use.

When the script is executed, the module will look at the dataset, which is given by its parent module, and look at each feature to see if it should pass. A new dataset is then created with only the wanted features, and that dataset is passed to the child modules.

Time series embedding

Incorporation of previous values of a feature can improve the results. This module shows the user all features, and allows the user to select the amount of lag to apply to each of the features. The number of time steps can be specified in intervals for each feature individually.

The module overrides the 'getDatasetDescription' method. This allows the module to modify the dataset description object that child modules receive. The lagged features are added to this description. Features that are not always available because of the selected interval are labelled as being optional.

time series embedding 2

This module allows you to repeat values from older examples. Make sure that the dataset is sorted on the right! Because the first examples lack historical data, they will be removed from the dataset.

Max number of examples to reach back: 23

Check all boxes of features to reach back to.

	t-1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Direction NL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Direction DeKooy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Direction Hoogeveen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Direction Leeuwarden	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Direction Schiphol	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Speed NL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Speed DeKooy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Speed Hoogeveen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Speed Leeuwarden	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Speed Schiphol	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Production	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

time series embedding 3

Using this module, you can select a subset of the available examples.

This module is used for sparse time series embedding.

Embedding 1	Direction NL	1	<input type="button" value="remove"/>
Embedding 2	Direction NL	2	<input type="button" value="remove"/>
Embedding 3	Direction NL	3	<input type="button" value="remove"/>
Embedding 4	Speed NL	1	<input type="button" value="remove"/>
Embedding 5	Speed NL	2	<input type="button" value="remove"/>
Embedding 6	Speed NL	3	<input type="button" value="remove"/>

[add new embedding](#)

Time series embedding 2

When lags are needed without all sub steps this embedding module can be used. An example of such a situation is when a user would like to embed the feature at one, two, and twenty-four time steps back. The previous module will embed all previous times, which results in 24 features being added.

This module asks the user for the maximum amount of lag to use. A grid with checkboxes is then presented to the user. The grid has the dimension of the maximum amount of lag, times the number of features. Each checkbox represents a possible lag. The boxes that are checked are lagged and fed to the next module.

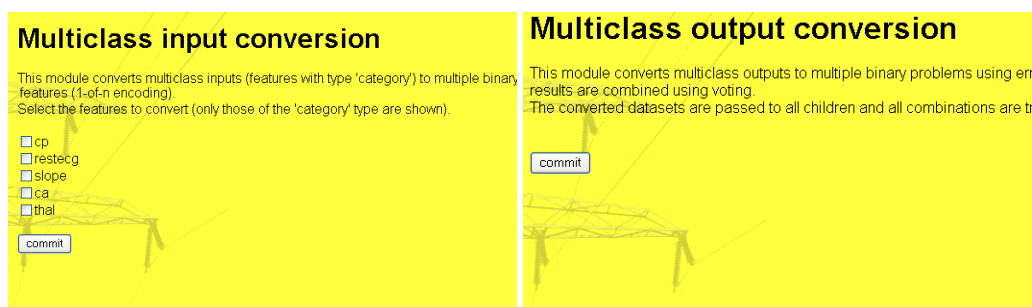
This module has no options to try different combinations during one test using the same module. The module overrides the 'getDatasetDescription' method too.

Time series embedding 3

Sometimes the maximum lag is larger than the maximum amount that can be specified in the 'Time series embedding 2' module (which is set to a maximum of 99 time steps). This situation occurs when one is using data that is sampled at very small time steps, and one would like to use values that are several days older.

Instead of increasing the possible amount of lag in the previous module, this module uses rules, reminiscent of the example subset module. The user can add rules, and each rule specifies the feature and the amount of lag.

Apart from the interface, and a small part that parses the rules, the process in the module is the same as in the previous module.



Multiclass input

The multiclass input module converts features that are labelled as 'category' to multiple binary features. When the user adds the module to the script, the user is shown a list with all category features. A checkbox indicated if the feature is to be converted.

When the script is run, all features are scanned to see if a checkbox was checked for that feature. If so, the feature is converted to a 1-of-N encoding (see section 3.4.1). The new dataset is passed to the child modules for further processing.

Multiclass output ECC

The previous module converts input features. Output features should not only be converted, but post-processed too. This module assumes that there is only one output in the dataset, or else it takes the first output of the 'category' type.

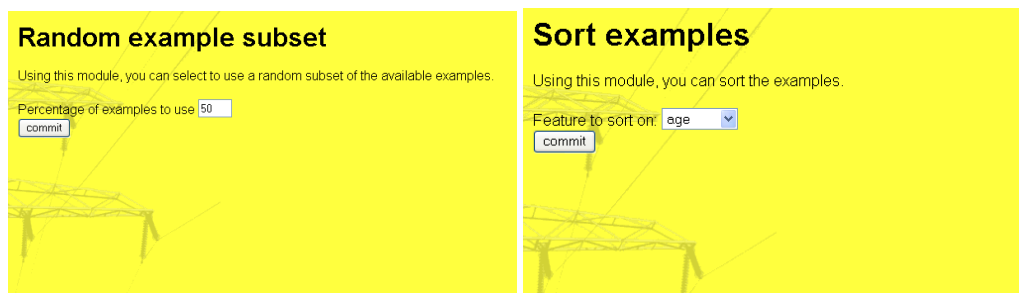
When the module receives the dataset, it determines the number of categories. From this number, a table with conversion codes is calculated (see section 3.4.1). Assume that the length of each codeword is N . The child modules are then called N times. Each time the output has been transformed to contain one of the N bits of the codewords.

When all wrappers have been executed, this module gets a chance to post-process. During post-processing the conversion codes and mappings are first retrieved from the 'Result' object. Then lists are created to pair wrappers. For executions that do not result in errors, the sets for each of the N bits contain wrappers with exactly the same parameters. Each of the wrappers in the first set, is then combined with wrappers at the same position in the other sets.

Each of these wrappers contains the desired and predicted values. These values are used to reconstruct the desired and predicted codewords. An error measure is

calculated for each of the sets available in the wrappers. This error measure is stored in the XML.

The choice was made to combine only wrappers in the same positions. The reason for this choice is that creating all possible combinations, and calculating error measures for those combination, could easily result in a combinatorial explosion. If for each of the N bits 50 combinations were created, this would result in a total of N^{50} combinations that need to be calculated and stored in the XML. It is clear that this is not feasible.



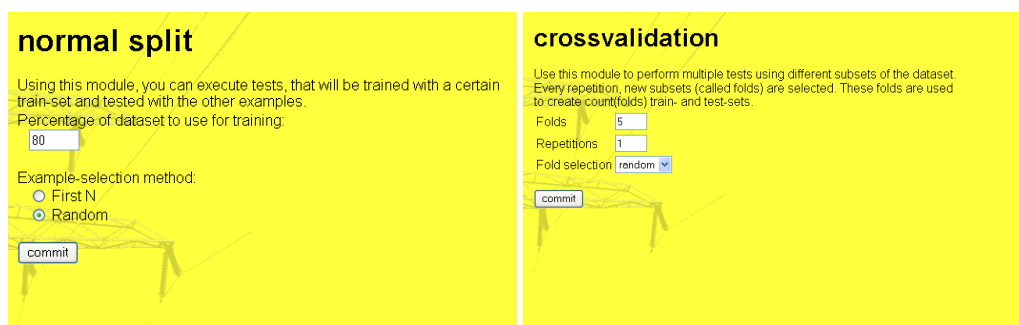
Random example subset

Huge sets of examples can take a while to train. When searching for parameters this can pose a problem. It is often possible to find good parameters with only a part of the examples. For a definitive error the complete dataset can be used afterwards.

This dataset creates such a subset. The user can specify the amount of examples to use as a percentage. When the script is executed a number of examples is selected at random from the dataset.

Sort examples

When a certain order is required in the dataset, this module can be used to sort the data. The user can specify the feature to use to sort the examples.



Normal split

All wrappers assume they receive both a train and a test set. A fast method to divide the dataset in a train and test set is the use of the normal split module. The user can specify the percentage of examples to use for training, and the selection

method. The selection method can be 'first N', which marks the first examples as examples to use for training and the rest for testing, or 'random', where the train and test examples are chosen at random.

When the script is executed, the module will provide the dataset with a one-dimensional array containing a label for each example, indicating that the example is to be used for training or for testing. Wrappers will request a particular subset from the dataset containing either training examples or testing examples.

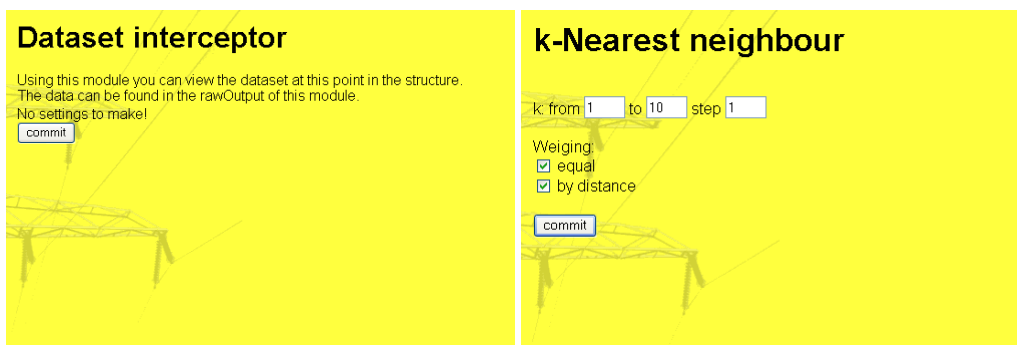
Cross validation

To get a good idea of the future performance of a predictor, it is not enough to test on one separate test set. A better idea of the performance is obtained by training and testing multiple times on different datasets.

The cross validation module splits the total set in multiple subsets called folds. Each fold contains approximately the same number of examples. The folds are then used to create multiple tests. In each test all folds are used in a different combination. Each time four folds are used for training while the remaining fold is used for testing. Using five folds results in five train and test cycles, each time with another set of test examples.

The module gives the user the opportunity to perform the creation of folds multiple times, and thus repeat the complete process multiple times, thereby giving an even better look at the distribution of errors as a result of using different train and test sets.

When predicting time series, some machine learning algorithms or parameter choices or features may lead to the algorithm not learning to generalize but only to return memorized outputs. This occurs especially if some of the inputs have a strong correlation with the time of the series. This leads to good results on the train and test sets, but not for future predictions. To prevent this situation the user can specify that the folds are to be selected at random or straight. The straight selection creates folds that contain consecutive examples. This way, consecutive parts of the dataset are left out when training and used for testing. This means that the algorithm is not able to memorize these examples (or examples that are close), so that the results for predicting the test set will be bad.



Dataset interceptor

The dataset might have been changed considerably after all processing steps of the previous modules. This module is used for debugging purposes or to check if the

steps are performed the way the user thinks they are performed. The module can be added as a child of any other module. Other modules can be placed as children, so that the flow of data can be monitored.

When executed, it records the dataset, including the description of the features as rawoutput in the WrapperResult. That way the dataset can be viewed like the output of any other wrapper.

k-Nearest neighbours

This module is an implementation of the k-nearest neighbour technique. The JSP provides the user with an interval to choose for k , and a choice to average the values of the k nearest neighbours or to weigh them according to their distance. When the user selects both types, the workbench will create wrappers for each of the options.

When the module is executed it iterates over the value of k and the chosen weighing types. For each combination a module is registered with the result object. A block of settings is added to the XML in the result object for each of the registered combinations. The corresponding wrapper is filled with the dataset and settings and handed to the result object.

The wrapper initialises the data by requesting all training examples from the dataset. These training examples are then used to test the performance. For $k = 1$ and no identical inputs, the error is expected to be 0.0, larger values of k will return a larger error. After the test on the train set, the test set is requested. This set is also used.

The resulting values and the values that were specified in the dataset are used to calculate some error measures. The output type determines the type of error measures to use. Measures for a continuous values output include the mean squared error and the mean absolute error. For binary values, measures such as the amount of correct classifications and the ratio of errors on the total number of examples are used. The wrapper object is used to calculate most of these measures. Some additional measures are also included in the XML, such as the running time.

Neural Network with Joone

This module trains and tests neural networks by using the Joone toolkit. The Joone toolkit is a pure Java neural network implementation, capable of using multiple layers and multiple threshold functions.

The JSP provides a form with options to the user. The user can specify the number of layers and the number and type of neurons for each of these layers.

The number of layers is fixed for each script execution. All combinations of the given number of neurons are tried during execution. Each of the combinations is registered with the result object, and a wrapper is provided.

To give the user more insight in the generalization over the number of epochs, without rerunning the training multiple times, the wrapper returns multiple sets of results at once. The network is trained for the first number of epochs. At that point a tests are performed for the train and test set. The results of these tests are stored, and training is resumed. After the step size of the epochs, the tests are performed again, and the results are stored again. This is repeated until the maximum number of epochs that was specified is reached. All results are bundled in a 'WrapperResult' object, ready to be retrieved by visualization modules.

LibSVM

LibSVM is a well-known tool used to train and test support vector machines [Chang and Lin, 2001]. Multiple variations of SVMs can be used (two for classification and two for regression), as well as four kernels and their parameters.

The JSP provides intervals for all parameters, as well as choices for the kernel and SVM types to use and the corresponding parameters. The module follows the standard procedure of filling the wrapper, writing data to the result and registering the wrapper with the result object.

The wrapper writes the dataset to file in a format that LibSVM can read. It then looks at the settings that are provided by the executing client for the path of LibSVM. This is combined with the name of the executable file and the command line options. This is executed by the native system by calling the 'execute' method of the 'Runtime' object, which is part of the standard Java distribution. The output is read until the process is finished or a terminate request is received.

After the execution, two new executions are performed to test the performance on the train and test sets. These results are again summarized and returned along with the desired and predicted values, all combined in a 'WrapperResult' object. The number of support vectors is obtained by applying a regular expression to the output of the training. This is converted to a 'setting' tag in the resulting XML.

SVMTorch

Using this module, you can execute tests, that will be trained with a certain train-set

Learning options:

C (trade-off between training error and the margin) from 100 to 100 step 1

eps (the width of the error pipe in regression mode) from 5 to 5 step 1

Kernel type:

Linear

Polynomial ((s*a^b + r)^d)

d: from 2 to 2 step 1

s: from 1 to 1 step 1

r: from 1 to 1 step 1

Radial basis function (exp(-|a-b|² / (std*std)))

std: from 10 to 10 step 1

Sigmoid (tanh(s*a*b + r))

s: from 1 to 1 step 1

r: from 1 to 1 step 1

Jeffreys

Using this module, you can execute tests, that will be trained with a certain train-set and tested with the other examples.

No settings to make!

SVMTorch

SVM Torch is another well-known tool [Collobert et al., 2002]. No executable was available for Microsoft Windows so that one had to be compiled from the sources

that were offered. The obtained executable allows the user to set some parameters and choose a kernel (along with another set of parameters).

The module follows the same line as the LibSVM module. All steps are the same, only the details such as parameter names and format of the dataset vary. To return the number of iterations that was performed, the output is scanned for all occurrences of the number of iterations by using a regular expression. The last match is used as the amount of iterations, which is inserted into the XML.

Jeffrey's prior

This module makes use of an algorithm much like support vector machines, called 'Adaptive Sparseness using Jeffrey's Prior'. The implementation obtained was available for R, which is an open-source statistical package compatible with the S language and environment, which was developed at Bell Laboratories. There are no settings that can be made for this tool.

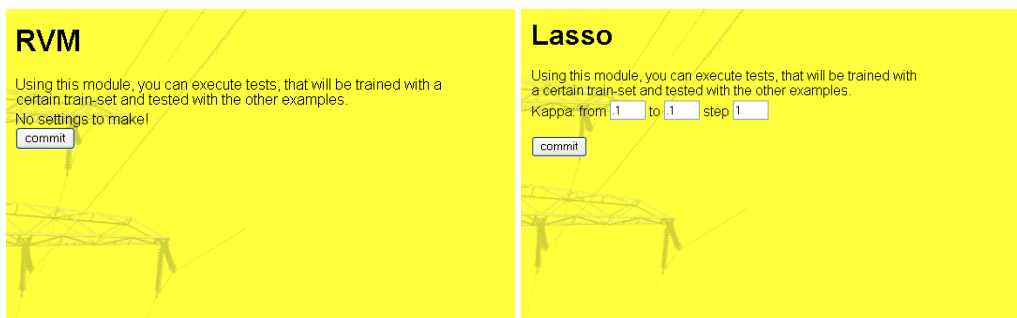
As more tools were integrated that interface with R, a separate class was made, WrapperR, which creates the interface with R. It defines some abstract functions that subclasses, such as WrapperJeffreysPrior, must implement.

WrapperR starts by creating train and test sets, which are converted to R code. It then calls the appropriate function of the subclass to insert its R code. The code is written to file and the R command line executable is started. The location of this executable is found by looking at the settings for the client machine.

The output of the R executable is gathered by the function in WrapperR. The subclass is given a chance to parse the data and return XML with results. The results are bound together in one XML string and sent back as a 'WrapperResult' object.

The 'WrapperJeffreysPrior' object implements the 'generateRCode' method and the 'parseResults' method. The 'generateRCode' method is called by WrapperR to insert programming code that is specific to Jeffrey's prior into the code that is to be executed. This module inserts code that constructs a model using the train set. The model is used to predict the values of both the train and test data. These predictions and the model data are printed so they can be used later.

The 'parseResults' method receives the data that was retrieved from R. In this case it will contain the model description and the predictions. These data are parsed using regular expressions to obtain the number of support vectors and the predictions. The predictions are passed to the Wrapper class to be summarized. The results are returned to WrapperR.



RVM

The RVM is another technique that is similar to support vector machines [Tipping, 1999]. It was implemented, just like Jeffrey's Prior, in R. The procedure is exactly the same as for Jeffrey's Prior with only some name changes and the call to another function in R.

Lasso

Lasso is based on work by Roth [2001]. It was implemented in R, so the same procedure could be used as for Jeffrey's prior and the RVM. The JSP was augmented with a parameter that is passed by the module to the wrapper, which includes it in the script that is executed by R.

Improved Lasso

This is an improved version of Lasso, by Rutger ter Borg. It is able to handle very large (>10000 examples) problems.

Learning options:

Kappa (maximum L1-norm imposed on parameters) from 2 to 2 step 1

Maximum iterations (0 for no maximum) 0

Kernel type:

Linear

Polynomial $(\gamma \cdot u^v + \text{coef})^d$

d: from 2 to 2 step 1

gamma: from 1 to 1 step 1

coef: from 1 to 1 step 1

Radial basis function $\exp(-(u-v)^2/\sigma^2)$

sigma: from 10 to 10 step 1

Sigmoid $\tanh(\gamma \cdot u^v + \text{coef})$

gamma: from 1 to 1 step 1

coef: from 1 to 1 step 1

commit

Improved Lasso

This module interfaces with the improved version of Lasso, which was created by Rutger ter Borg. The original version cannot cope with large datasets. The memory and time needed to solve the optimisation problem becomes very large soon. The improvements allow the Lasso algorithm to use much larger datasets by using a smart technique to multiply the internal kernel matrix.

The tool was implemented with a command line interface. The options are similar to that of SVM Torch. The kernel can be selected as well as a diversity of parameters. The JSP looks similar to that of SVM torch. The module creates wrappers for all parameter combinations.

The wrapper is similar to the other wrappers that interface with the command line. A command is prepared with the required options, and options where to read and write data and model files. After the execution of the training, two test runs are executed for the train set and the test set. The results are read from a file and parsed. The results are then summarized and transformed into XML. This is sent back to the workbench as a 'WrapperResult' object.

5.3.4 Visualization modules

Visualization modules provide the user with data on the dataset and the results. This data can be presented in various forms, such as tables and plots. The modules are therefore mainly JSP based, which makes them easy to modify, and supported by Servlets that create plots.

Dataset plots

The dataset plots module is essentially an extension of the correlations module. It displays scatter plots of all pairs of features. In addition, one column and one row are added with the index number of the example. This shows the values in their order of appearance in the dataset. For large numbers of features the number of plots can take a while to process.

Manual XSLT

This module allows the user to extract data from the resulting XML. The user can select the results to use and enter a valid XSL transformation in a text area and press the commit button. The XSL is then applied to all selected results. The results are shown to the user.

A list of plugins is shown below the results. The names of the plugins are retrieved from the plugins-directory. When the user clicks the link to one of the plugins, the results of the XSLT are sent to the plugin.

Visual XSLT

The creation of XSL transformations is not easy, and even standard transformations can take some time to get right if done by hand. This module was made to ease the creation of simple transformations that generate tables with related values. The idea is that the user selects the values in the XML that he wants to use. The module locates the relations between the values and shows the results to the user. The user can refine the results by selecting some constraints.

The XSL transformation that will be created consists of one template that should match an element, and a series of statements that select values. The latter statements all start their match from the location of the match of the template. The element that is used for the template to match is called the main attribute, while the others are extra attributes.

If a user wants to display the errors and a certain parameter that was varied, he should select the value of the error for one of the results. As extra attribute he selects the parameter. Care should be taken to click on the parameters that are in the same block as the selected error. Otherwise the following steps will read the wrong values.

To allow the user to select such tags and elements, an XSLT is applied to the result, which transforms all tag names and attributes in clickable links. Each of the links is given a parameter that contains the complete path to the tag or attribute.

The module extracts all elements from the main attribute that the user selected. These elements are shown along with checkboxes that can be used to limit results to be for a certain module. The elements of extra attributes are extracted too. All consecutive elements of these attributes that match the elements of the main attribute are then removed. The result is that the module has one long path to an attribute, and several shorter ones.

The XSLT is created based on these paths. The main attribute and the limiting elements are used to select a certain element from the XML. Each of the extra attributes is located by going up a certain number of levels (to get to the point where the paths failed to match). From that point the path is used to locate the attribute and insert into the resulting output.

The XML is printed on screen along with the resulting output. These can be used to change the settings. The XSLT can also be used in combination with the manual XSLT module. The output can also be sent to the plugins, which are listed the same way as in the manual XSLT module. This module also sends the names of the selected attributes to the plugins. This allows plugins to use sensible names for the columns.

Because of the complexity of the module the process is divided in several steps. If information is still missing to perform the next step, the next step is not shown. This way the user is only shown information that is relevant at that moment.

Wrapper output

Although statistics such as error measures over the desired and predicted outputs give an idea of the performance, they are unable to give the user a quick idea what is going wrong. The wrapper output module shows the desired and predicted values for the wrappers. In regression problems this can give the user a quick overview of the performance of the wrapper.

To gather the data, all wrappers should store the desired and predicted values in the 'WrapperResult' object, which is sent to the workbench and saved for later use. This is done by calling the 'addOutputdata' method on the 'WrapperResult' object. This method creates a 'WrapperOutputdata' object and stores it in the 'WrapperResult'. The workbench can access them later via a call to the 'getOutputdata' method.

The values are stored along with a label. That way a wrapper can store multiple sets, for example for training and testing. Other wrappers might need to store even more values if they produce multiple predictions (as the improved lasso does for kappa, or the neural network module for different epochs).

The JSP shows the user with a list of results, and uses the first one by default. For the selected result the wrappers are located. Their identification code, along with the direct parameters of the wrapper, are also displayed to be selected (and the first one is selected by default). The parameters are extracted by using an XSL transformation, which extracts all name and value pairs from the settings block belonging to the wrapper.

All settings that are related to the module are displayed (also extracted by using an XSL transformation) below these select boxes.

The actual plots contained in the wrapperresult for the selected wrapper are displayed then. The label that the wrapper stored precedes each plot. For each WrapperOutputdata object two plots are made. One plot shows the desired and predicted values in different colours according to their index. The other plot sets

the desired and predicted values off against each other. If this plot is a diagonal line this means that all predicted values are (almost) equal to the desired values, and that the predictions were correct. Deviations from the diagonal indicate problems.

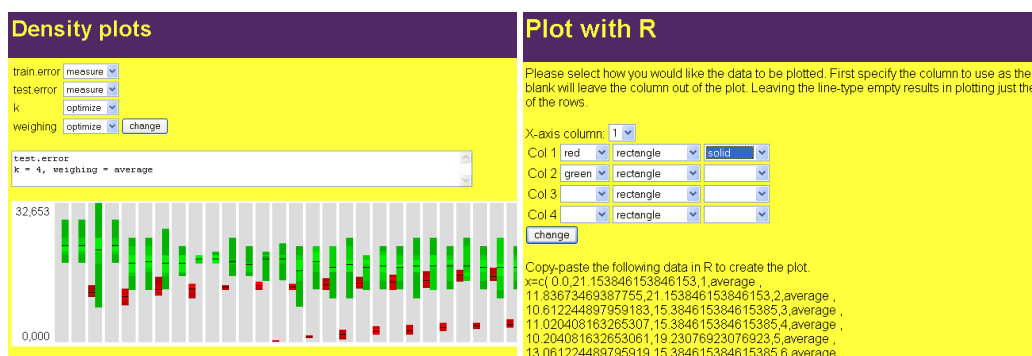
Html table				Matrix for R
train_error	test_error	k	weighing	x=c(0, 0, 21, 153846153846153, 1, average ,
0,0	21, 153846153846153	1	average	11, 83673469387755, 21, 153846153846153, 2, average ,
0,0	21, 153846153846153	1	distance	10, 612244897959183, 15, 384615384615385, 3, average ,
0,0	21, 153846153846153	2	distance	11, 020408163265307, 15, 384615384615385, 4, average ,
0,0	17, 307692307692307	3	distance	10, 204081632653061, 19, 23076923076923, 5, average ,
0,0	15, 384615384615385	4	distance	13, 061224489795919, 15, 384615384615385, 6, average ,
0,0	21, 153846153846153	5	distance	12, 853061224489797, 21, 153846153846153, 7, average ,
0,0	15, 384615384615385	6	distance	13, 46938775510204, 19, 23076923076923, 8, average ,
0,0	23, 076923076923077	7	distance	14, 285714285714286, 17, 307692307692307, 9, average ,
0,0	20, 408163265306122	1	average	13, 877551020408163, 19, 23076923076923, 10, average ,
0,0	20, 408163265306122	1	distance	15, 10204081632653, 17, 307692307692307, 11, average ,
0,0	20, 408163265306122	2	distance	14, 693877551020408, 17, 307692307692307, 12, average ,
0,0	20, 408163265306122	2	distance	15, 510204081632653, 15, 384615384615385, 13, average ,
0,0	20, 408163265306122	3	distance	15, 10204081632653, 15, 384615384615385, 14, average ,
0,0	20, 408163265306122	4	distance	15, 918367346938776, 17, 307692307692307, 15, average ,
0,0	18, 367346938775512	5	distance	15, 510204081632653, 17, 307692307692307, 16, average ,
0,0	18, 367346938775512	6	distance	16, 3265306122449, 19, 23076923076923, 17, average ,
0,0	22, 448979591836736	1	average	15, 918367346938776, 13, 461538461538462, 18, average ,
0,0	22, 448979591836736	1	distance	14, 693877551020408, 15, 384615384615385, 19, average ,
0,0	22, 448979591836736	2	distance	14, 693877551020408, 13, 461538461538462, 20, average ,
0,0	22, 448979591836736	2	distance	0, 0, 21, 153846153846153, 1, distance ,

HTML table plugin

This plugin receives a comma-separated list of values, which might have been created by a visual XSLT. The list of values is transformed into a HTML table, which makes the values easier to read. Above each column in the table are the column labels that were passed by the calling module. Clicking the label sorts the table on the values of the corresponding column.

R matrix plugin

This plugin outputs the values as code that can be used in R. This makes it possible to post process the results by using R. The code can be copy-pasted into R.



R plot plugin

R can export plots to a variety of formats. This module was created to automate the creation of plots in R. The result of this module is again a piece of code that can be copy-pasted into R. The plugin displays a list with select boxes. The first select box is to select the column of the data that is to be used as value on the x-axis. Below this box there is a line for each of the columns. Each line has multiple boxes to allow the appearance of the plot to be changed. The user can select the colour of

the points (and line) for the values of this column as well as the type of the points and the line.

Density plot plugin

When a large number of tests are performed, the exact results of parameter changes get lost in the numbers. The density plot plugin allows the user to visualize the results of these changes.

The top of the page shows the selected columns and a select box with the action to take. The available options are 'optimise', 'measure' and 'ignore'. Optimise is used for the parameters that the user wants to optimise. The values of the corresponding column are plotted over the x-axis. Measure is to measure the effects of the columns that are to be optimised. Ignore is to indicate that the column is to be left out.

The resulting plot shows a number of coloured bars. The order of the bars is determined by the order of the columns. The bars start with the lowest value for the first column to optimise and end with the highest value (if the value of the column is not a number the values are ordered lexicographic). Within this pattern the bars are sorted on the next column to optimise until there are no more columns to optimise. Then a bar is drawn for each of the columns to measure. These bars are coloured differently for each measure (if there are more than three columns to be measured the colours are repeated).

The bars are plotted by taking all values for a certain column to be measured that have the same values in the columns to optimise. For these values the 100%, 80%, 50% and 20% value intervals are taken, along with the average. These values are plotted in different shades of the colour of the bar.

6

Evaluation

To test the workbench and show its capabilities, three datasets were selected. These datasets are the Cleveland heart disease dataset, the iris dataset and a dataset by NUON for the prediction of the total output of several Dutch wind mill parks. The first two were selected because of their widespread use in the research community. The last dataset was selected to apply the workbench to a new, real world problem.

The datasets have different types of values to predict. The Cleveland dataset is a binary classification problem, and the iris dataset is a multiclass classification problem. The problem for NUON is a regression problem. This variety helps to show both the general and specific issues concerning the workbench.

The goal of the Cleveland heart disease and iris datasets is to provide a use-case in which the workbench is used. The steps of using the workbench are presented in depth, especially for the Cleveland test. The aim is to obtain results comparable to those found in literature.

The goal of the third dataset is to obtain a model that can be used by NUON to improve their predictions of the power output. The global process of using the workbench is shown, and augmented with details for those places where the process differs from the previous tests.

6.1 Cleveland heart disease

This dataset is from a set of four heart disease datasets, of which the Cleveland dataset is the largest. The dataset was obtained from the UCI Machine Learning Repository [Blake and Merz, 1998]. A slightly modified version is used where the three different forms of disease are reduced to one class.

6.1.1 Dataset description

The dataset consists of 303 examples, of which seven are not usable due to missing values. Each example is based on fourteen features. The first features are the age of the patient, and the sex. Then there is a multiclass feature indicating the type of chest pain (typical angina, atypical angina, non-anginal pain and asymptomatic).

The fourth feature is *trestbps*, which is the resting blood pressure in mm Hg on admission to the hospital. Then there is the serum cholesterol in mg/dl. After this comes a binary feature called *fbs*, which indicates if the fasting blood sugar is larger than 120 mg/dl. The seventh feature is the resting electrocardiographic results and has three values: normal, having ST-T wave abnormality or showing probable or definite left ventricular hypertrophy by Estes' criteria. The next feature is the maximum heart rate achieved. After this comes another binary feature, which indicates if exercise induced angina. The tenth feature is *oldpeak*, which is ST depression induced by exercise relative to rest. This feature is followed by the slope of the peak exercise ST segment (three values, upsloping, flat or downsloping). The next feature is the number of major vessels (0-3) coloured by fluoroscopy. The thirteenth feature is *thal*, which can be normal, fixed defect or reversible defect. The last feature is the class to predict: healthy or sick.

The following diagram (which was created by the workbench) summarizes the features.

Name	Type	Output?	min	max	avg	st.dev
age	continuous	no	29,00	77,00	54,54	9,03
sex	binary	no	0,00	1,00	0,68	0,47
cp	category	no	1,00	4,00	3,16	0,96
trestbps	continuous	no	94,00	200,00	131,69	17,73
chol	continuous	no	126,00	564,00	247,35	51,91
fbs	binary	no	0,00	1,00	0,14	0,35
restecg	category	no	0,00	2,00	1,00	0,99
thalach	continuous	no	71,00	202,00	149,60	22,90
exang	binary	no	0,00	1,00	0,33	0,47
oldpeak	continuous	no	0,00	6,20	1,06	1,16
slope	category	no	1,00	3,00	1,60	0,62
ca	category	no	0,00	3,00	0,68	0,94
thal	category	no	3,00	7,00	4,73	1,94
disease	binary	yes	0,00	1,00	0,46	0,50

6.1.2 Previous results

Skalak [1994] applied different techniques to some datasets, among which is the Cleveland heart disease dataset. The tests were performed using 5-fold cross-validation. For k -nearest neighbours, with $k = 1$ he reports a score of 74.3% examples classified correctly. Monte Carlo MC1 results in 80.7% correct classifications. Two variants of random mutation hill climbing (RMHC) result in 82.3% and 80.7% rates. Different combining strategies and techniques were used by Kuncheva [2002]. Among the used techniques were majority vote, naive Bayes, kNN and neural networks. The used dataset was also reduced to a binary classification problem. The results varied around 82% correct classification rate. Support vector machines have also been applied to this problem. Fung and Mangasarian [2000] reports a classification rate of 85.8%.

6.1.3 Test

Before the workbench can be used, a new project has to be added to the workbench. After the addition and selecting the project, the index page of the project is

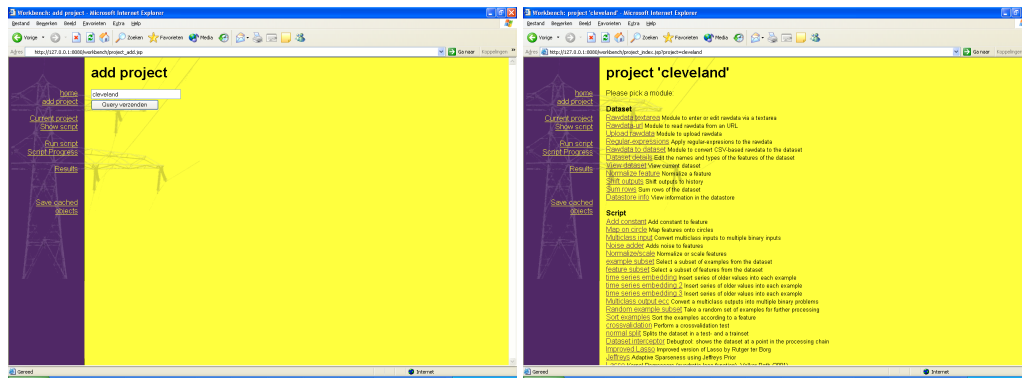


Figure 6.1: 'Add project' module, 'Project index page'

shown. This shows all modules in the order dataset modules, script modules and visualization modules.

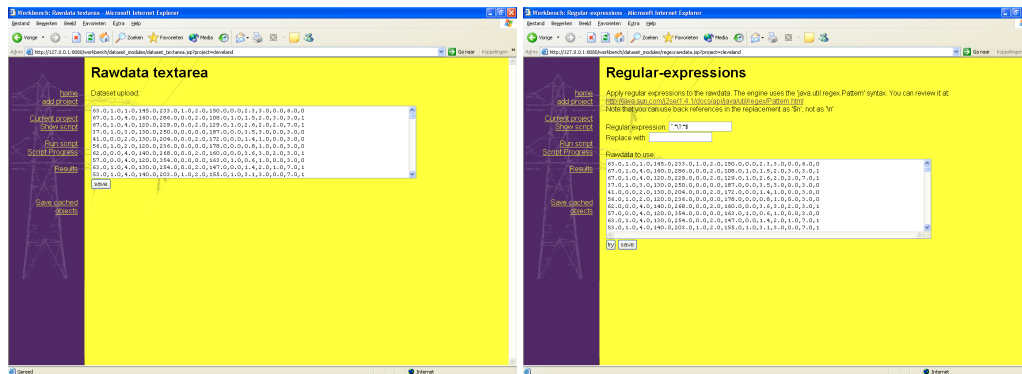


Figure 6.2: 'Rawdata textarea' module, 'Regular expressions'

The first step in this test is creating the dataset. To insert the data, the 'Rawdata textarea' module is selected from the project index screen. The data is copied from the original file, and pasted into the textarea of the 'Rawdata textarea' module. This results in the data being saved in the variable called 'rawdata', and the project index being displayed again. As the data contained some missing values, the regular expressions module was used to filter these examples out. The pattern to match was set to `".*?.*$"` and this was to be replaced with empty string. The expression tells the system to match all lines that contain a question mark, and replace those by an empty line.

To create the dataset, the 'rawdata to dataset' module was selected from the project index page. This converts the data in the rawdata variable to a complete dataset object. This module shows the first characters of the dataset and the results, but does not return to the index page. To go to the index page, the 'Current project' link in the menu is selected. To provide the workbench with additional data on the dataset the 'dataset details' module was opened. The names and types of the various features were changed so that summaries show the right names (instead of 'feature1' etc), and the dataset plots can be made in different colours.

The first inspection of the dataset was performed by looking at the correlations

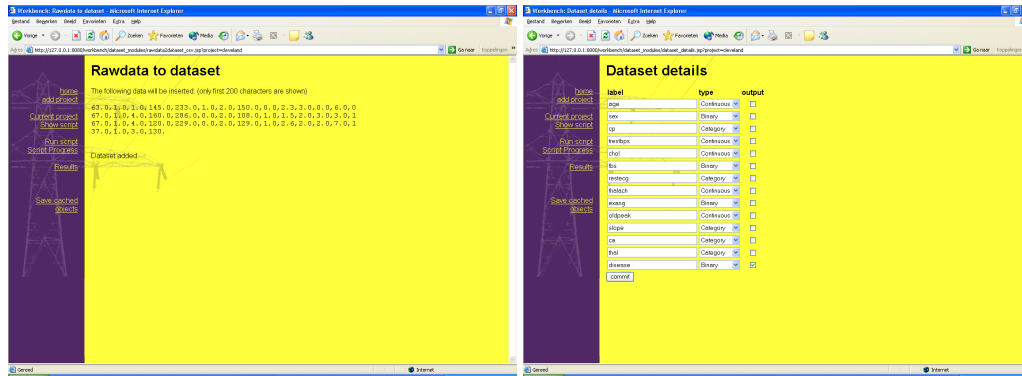


Figure 6.3: 'Rawdata to dataset' module, 'Dataset details' module

and the dataset plots. These modules are visualization modules and can be accessed under that header at the index page. The correlations module reveals no strong correlations between input and output features. The same conclusion can be drawn from the plots. Most plots show highly overlapping areas of patients with and without disease.

Three techniques are used to create a predictor for the Cleveland set. The first is the k -nearest neighbours, the second the neural network and the third SVM Torch. The first run of tests is performed using the same script. The first module is the cross-validation module. It is added to the script from the project index page. It uses random selection of 6 folds. The next is a normalization module, which normalises all inputs to mean zero and standard deviation one. This is done to obtain comparable dimensions for all variables, as well as help the algorithms as most algorithms work better with the input values around zero. This module and the subsequent ones, are added by clicking the link titled 'add module' from either the script page or the module. This results in a pop-up being shown. By selecting a module, it is automatically added to the script.

The k -nearest neighbour module was added first. It was set to vary k from one to ten, and try both averaging of outputs as well as weighing the contribution of examples by their distance.

The script was started by clicking the link 'run script' from the menu. After clicking 'start' the script is started. The progress can be monitored from the progress page. After a short while the 120 tests were completed.

The resulting XML showed varying performance, but the amount of numbers was too large to make a conclusion immediately. Therefore the visual XSLT module was opened. A new XSL transformation was created, given a name and the newly created result was selected. As main feature the value of the train error of the first kNN module was selected. To this, the test error, the value of k , and the weighing method were added. The paths were set to match the names of these attributes.

The resulting XSLT is shown below. It selects all results with the name attribute set to 'train.error'. It then locates the other values that are to be included in the table: a result with name='test.error', a setting with name='k' and a setting with the name 'weighing'. Of all these tags, the value is selected, and displayed.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>
  <xsl:template match="script/module/repetition/fold/module/run/module/module/
```

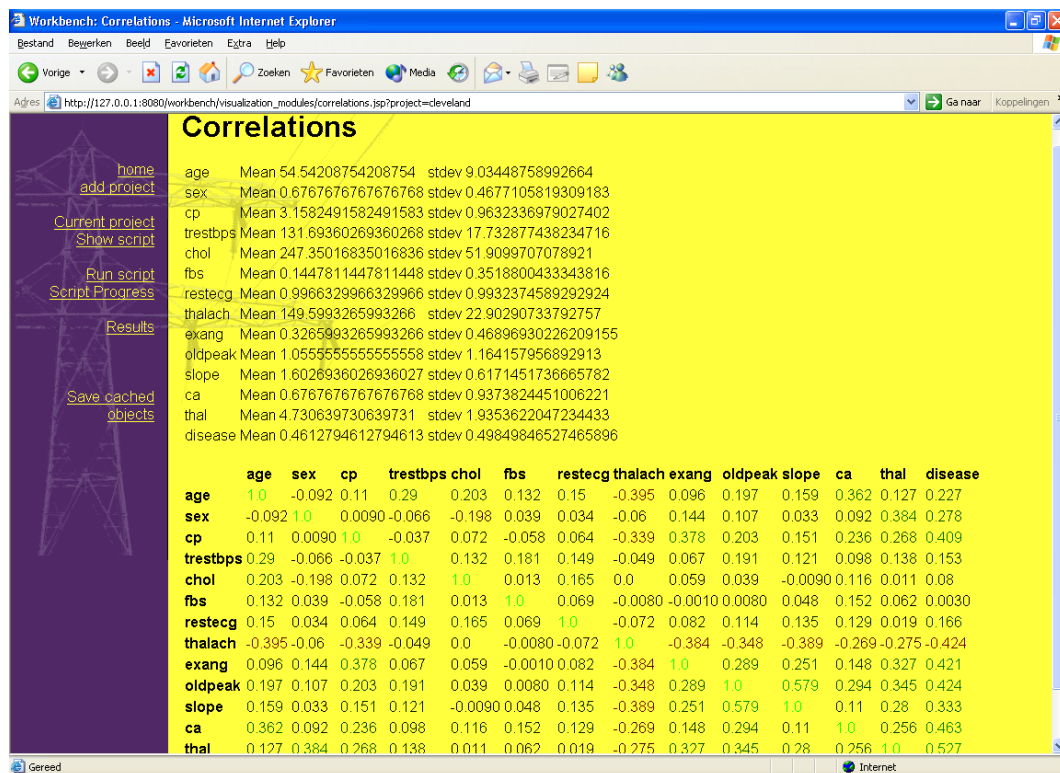


Figure 6.4: 'Correlations' module

```

results/result[@name='train.error']">
  <xsl:value-of select="@value"/>,
  <xsl:value-of select=" ../result[@name='test.error']/@value"/>,
  <xsl:value-of select=" ../settings/setting[@name='k']/@value"/>,
  <xsl:value-of select=" ../settings/setting[@name='weighing']/@value"/>
</xsl:template>
<xsl:template match='@*|node() '><xsl:apply-templates/></xsl:template>
</xsl:stylesheet>

```

The results of this XSL transformation are 120 lines with results. The best way to select the optimum is to use the 'density plot plug-in'. The parameters of this plug-in were set to optimize k and weighing, and to measure both error measures. The resulting plot shows the error measures (red for train error and green for test error) for k varying from left to right (left side $k = 1$, right side $k = 10$) and weighing varying at each two sets of bars (first the average then the distance based weighing).

The plot shows that the results for $k \geq 3$ do not vary much. As optimum was chosen to use $k = 4$ and weigh by taking the average. Averaging was chosen over distance weighing because it brings the train and test error closer, which means a more consistent performance. The results are summarised here:

Error measure	Value
Percentage errors trainset	11, 62
Percentage errors testset	18, 86

The next test was performed by replacing the kNN module with the 'neural network with Joone' module. The test was again performed with 6 folds, all inputs normalized. The number of hidden layers was set to one with 3 to 30 neurons in

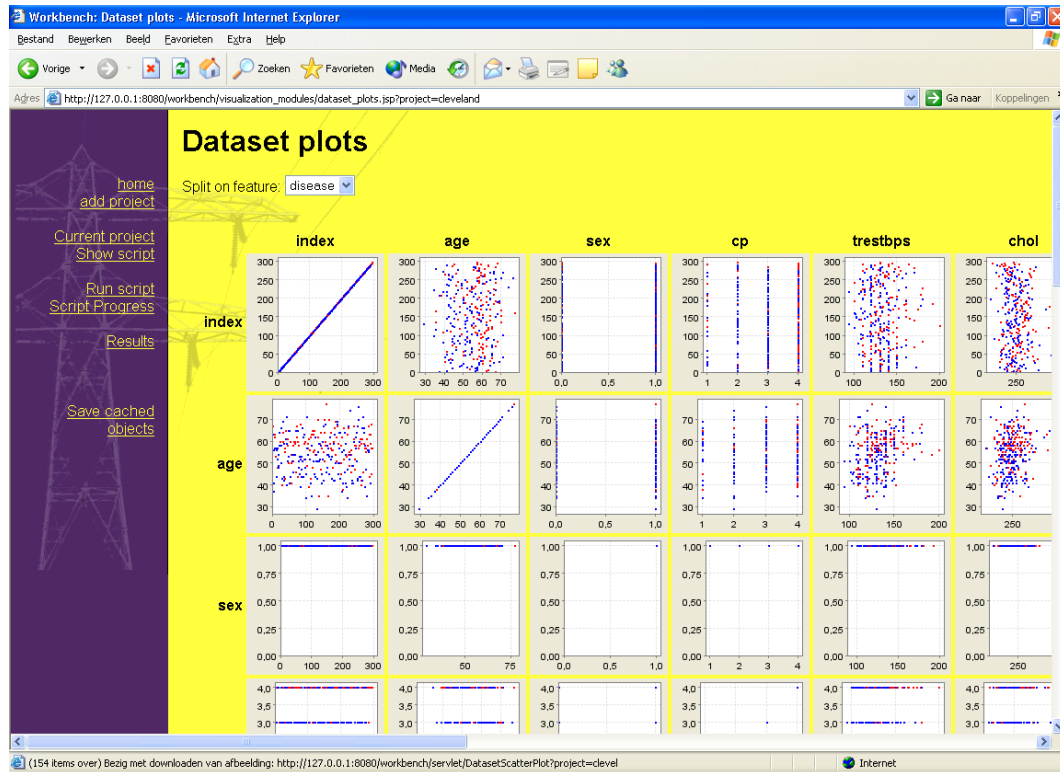


Figure 6.5: 'Dataset plots' module

steps of 4. The number of epochs was set to an interval so that the performance at different training times could be measured. The script was started and when all tests had been performed the visual XSLT module was used again. After selecting the right results the necessary measures were selected from the XML. These measures were the train and test error, as well as the number of neurons and epochs. The density plot revealed that good results are obtained after 400 epochs with 11 neurons. This corresponds to the following average error measures:

Error measure	Value
Percentage errors trainset	2,36
Percentage errors testset	18,53

SVMtorch was used to test the performance on an SVM. The same basis of the script was used, only the neural network module was replaced with the SVMTorch module. SVMTorch supplies a large number of parameters. Therefore four tests were performed, each with a different kernel. The first kernel is the linear kernel. Varying the parameters showed no significant changes.

Error measure	Value
Percentage errors trainset	14,50
Percentage errors testset	17,97

To determine the optimum for SVMTorch with a polynomial kernel, many SVM-Torch executions were performed: 1968. Different combinations were tried. The

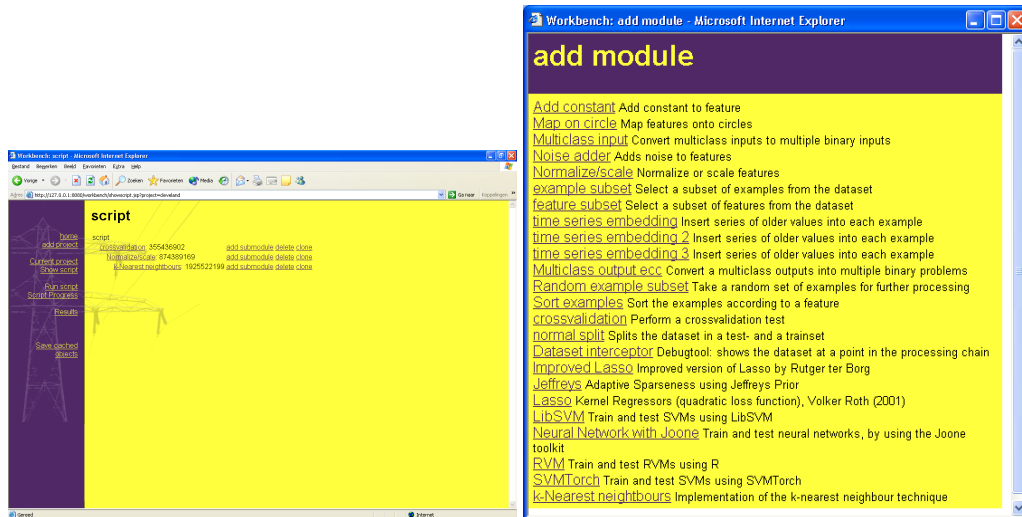


Figure 6.6: Script page, 'add module' pop-up

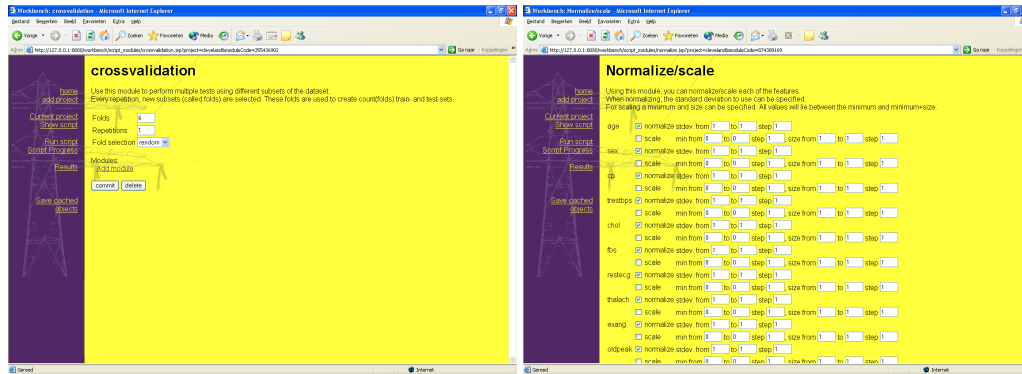


Figure 6.7: 'Crossvalidation' module, 'Normalize/scale' module

optimum was determined to be for: learning.c = 0.5, d = 4.0, s = 2.0, r = 4.0.

Error measure	Value
Percentage errors trainset	0
Percentage errors testset	23, 27

The next test of SVMTorch was performed with an RBF kernel, with varying parameters. Three runs were needed (with a total of 600 executions of SVMTorch) to determine that the best results are obtained at learning.c = 0.3, std = 3.0.

Error measure	Value
Percentage errors trainset	10, 91
Percentage errors testset	17, 51

The last test on SVMTorch was using the Sigmoid kernel. A good prediction was obtained at learning.c = 0.01, kernel = sigmoid, s = 1.0, r = 5.0.

Error measure	Value
Percentage errors trainset	16, 50
Percentage errors testset	15, 89

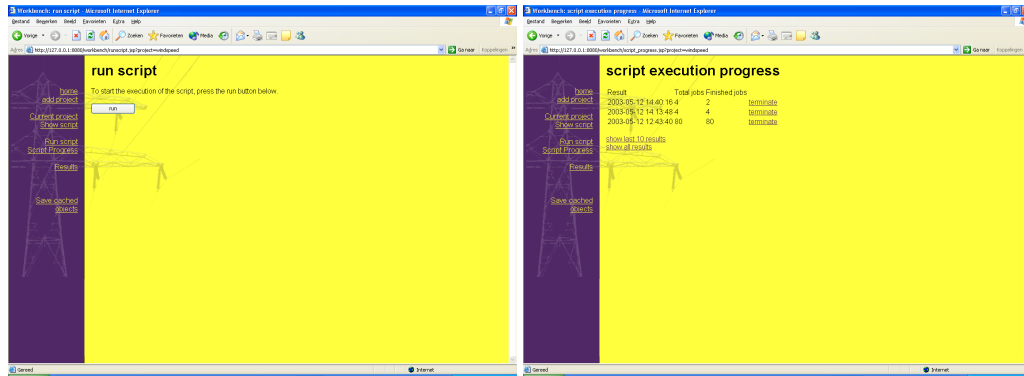


Figure 6.8: Run page, progress page

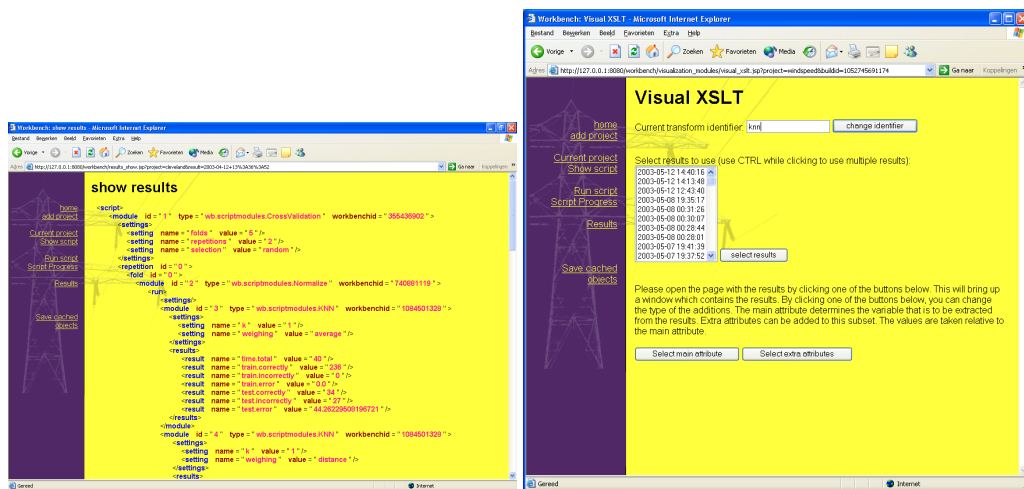


Figure 6.9: XML shown in the result page, first step in 'visual XSLT' module

The previous results were obtained using only a slight modification of the input features. Some of these features are features with nominal and ordinal values, which should not be treated as continuous values. Therefore the last module was replaced by the 'multiclass input' module. This module was set to transform all multiclass inputs to the one-of-n encoding. To this module the SVM Torch module was added with the optimal parameters of the last test. The results did not improve, but the parameters were not selected for this new configuration either.

Error measure	Value
Percentage errors trainset	21, 21
Percentage errors testset	24, 93

By varying the parameters and using the density plot to select the best results the following parameters were selected. $learning.c = 0.01$, $s = 0.5$, $r = 3.0$. This led to the following errors.

Error measure	Value
Percentage errors trainset	17, 71
Percentage errors testset	17, 55

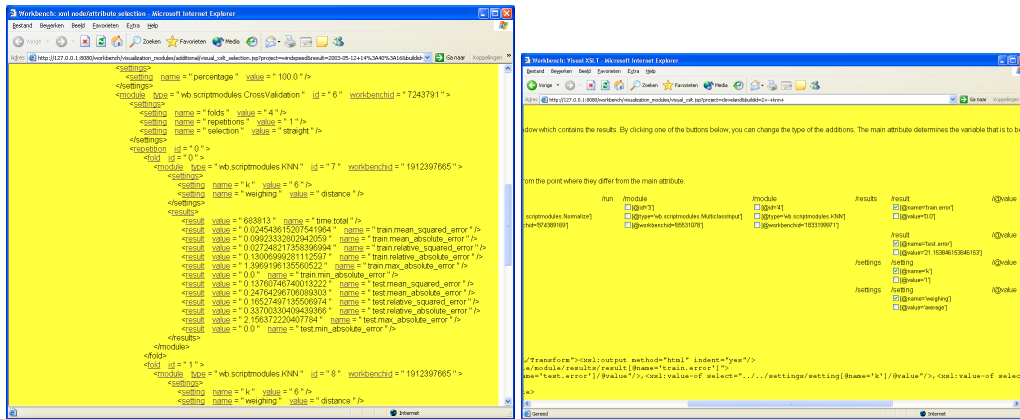


Figure 6.10: selection of tags to include in XSLT, selected features

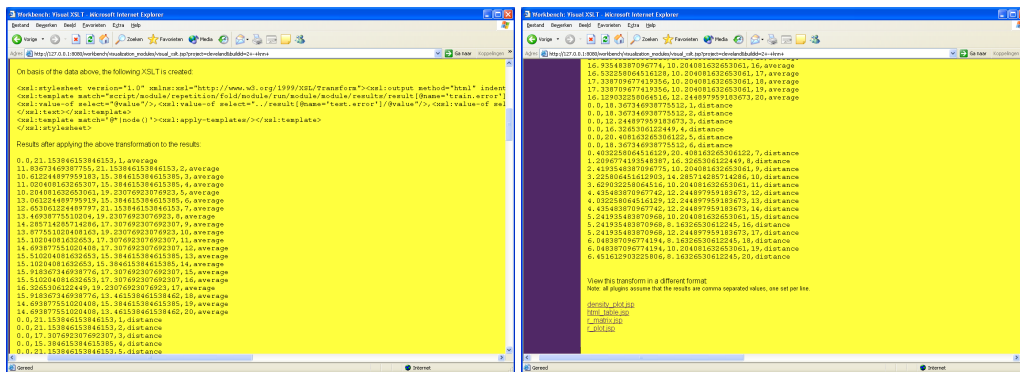


Figure 6.11: part of the XSLT and results, plugin selection

Repeating the process for the kNN module led to $k = 8$, weighing = average, with results:

Error measure	Value
Percentage errors trainset	13,00
Percentage errors testset	16,81

6.1.4 Conclusion

This section has show that the workbench is easy to use. The creation of the script was accomplished by clicking and selecting values to try. The extraction of the results needs some steps that are not totally clear at the start. However, once one is accustomed to this way of working, it becomes a much more powerful way to combine data.

The results obtained during the tests showed that results comparable to those in literature were easily reproduced.

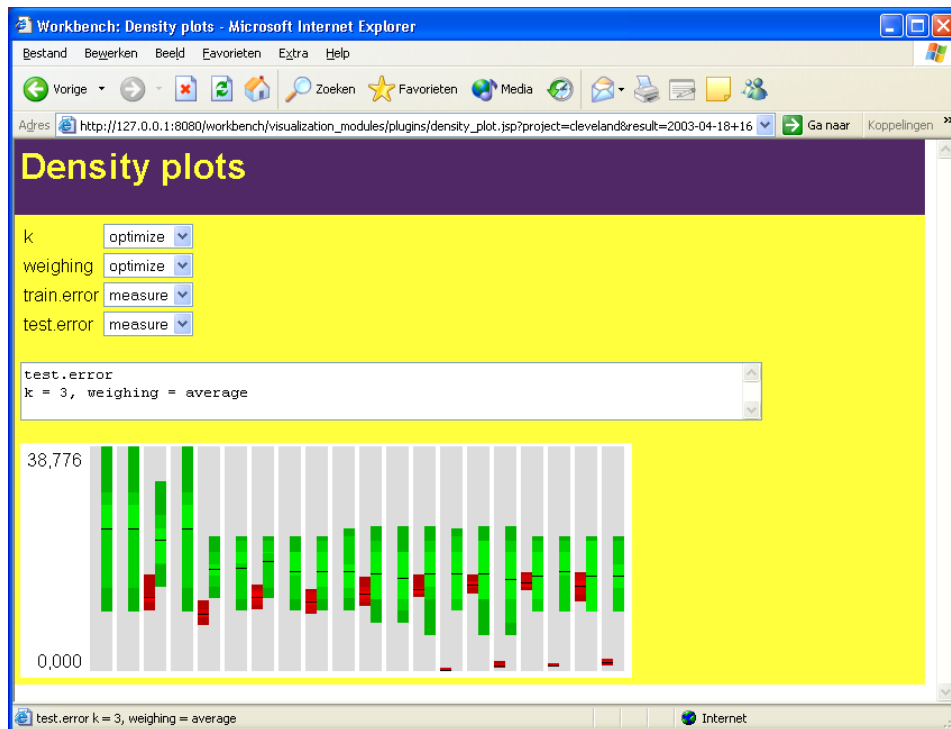


Figure 6.12: results for kNN module on the cleveland dataset shown in the 'density plot' plugin

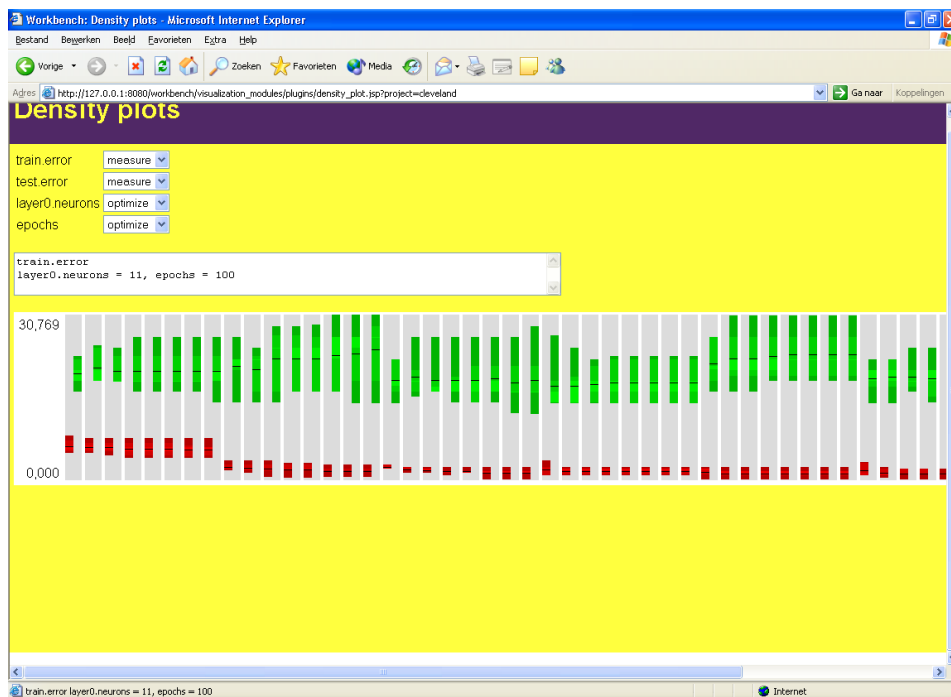


Figure 6.13: results for neural network module on the cleveland dataset shown in the 'density plot' plugin

6.2 Iris

The iris classification problem is a classic problem where the type of iris is to be determined. There are three iris plant types in the dataset to be classified.

6.2.1 Dataset description

The dataset consists of 150 examples. There are 50 examples for each of the three types. There are four input features. The first is the sepal (which is the green leaf) length, the second the sepal width. The third is the petal (coloured leaf) length, and the fourth is the petal width. The output feature is the type of iris.

One of the classes is linearly separable from the others. The others are not linearly separable. The dataset has been used as a test for many machine learning algorithms. Today it is not considered to be a hard problem, but it can be used to show features of the workbench related to multiclass problems.

Name	Type	Output?	min	max	avg	st.dev
sepal length	continuous	no	4,30	7,90	5,84	0,83
sepal width	continuous	no	2,00	4,40	3,05	0,43
petal length	continuous	no	1,00	6,90	3,76	1,76
petal width	continuous	no	0,10	2,50	1,20	0,76
type	category	yes	1,00	3,00	2,00	0,82

6.2.2 Previous results

Fisher [1936] used the complete dataset to get an lower bound for the classification error. He used the complete dataset to come with a theoretical bound of 2.5%. Skalak [1994] also reported performances on the iris dataset. For k -nearest neighbours, with $k = 1$ this results in a score of 93.3%. MC1 and the RMHC techniques vary from 93.3% to 94.7%. Setiono and Liu [1996] reports an accuracy of 94.55% on test data by using neural networks.

6.2.3 Test

The dataset was inserted using the rawdata textarea module. The regex module was then used to convert the names of the iris types to numbers. This was converted to the dataset using the 'rawdata to dataset' module.

All features have high correlations with the type, especially the petal length and width (0.949 and 0.956 respectively). The plots show that some of the features are very usable to separate the different types.

A script was then constructed. The first module normalizes all inputs. The next is the feature subset module. Currently it allows all features, but it was added for future use. To this a crossvalidation module was added with 5 random folds. These modules create the transformed dataset. To this the prediction techniques can be added. Because the iris problem is a multiclass problem, the machine learning techniques can not be applied directly. The output feature has to be converted to multiple binary problems. To do so, the multiclass ecc module was added. This module splits the problem into multiple binary problems, and converts the results to the original predictions. To the multiclass module, the kNN module was added with k varying from 1 to 10.

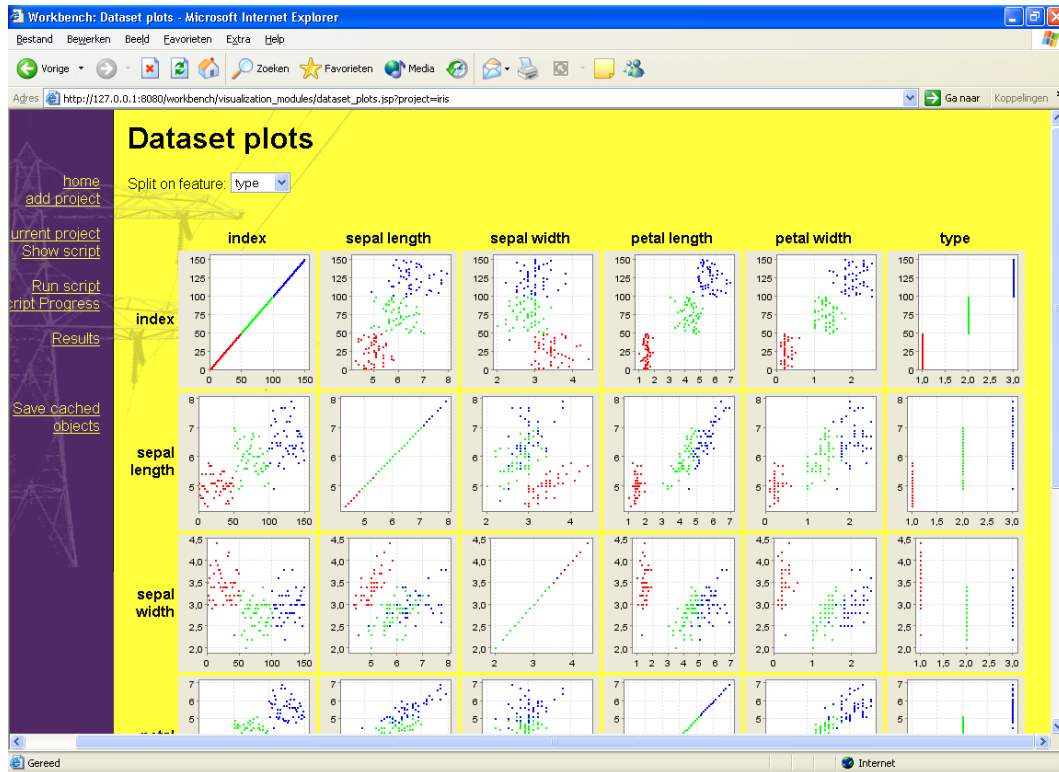


Figure 6.14: plots of different features of the iris dataset

The test was started, and when the results were obtained, the multiclass module automatically converted the results from the kNN modules to the three types of iris flowers.

To measure the performance, the visual XSLT module was used. The XSLT consisted of the 'id'-attribute of the 'set'-tag of the multiclass module. Extra parameters were the error and the type (train and test set). The resulting data was visualized using the density plot plugin. After the best results were chosen, the corresponding settings of the tools were located in the XML. The best results were obtained for $k=7$, weighing by averaging.

Error measure	Value
Percentage errors trainset	3,83
Percentage errors testset	3,33

The application of neural networks to the problem followed the same structure. Instead of the kNN module, the neural network module was added to the script, right after the multiclass module. The neural network module was set to vary the number of neurons. The script was executed, and a XSLT was created visually. The results were plotted using the density plot module, and the best settings were obtained for three neurons in the hidden layer.

Error measure	Value
Percentage errors trainset	2,33
Percentage errors testset	4,00

SVMTorch was used to obtain the results for support vector machines. Different kernels were used. Of the RBF kernel with learning.c = 10, and std=5, the measures were

Error measure	Value
Percentage errors trainset	2,00
Percentage errors testset	2,67

The sigmoid kernel was no able to give good results. The polynomial kernel with learning.c=0.5, d=3, s=0.4 and r=0.7 resulted in

Error measure	Value
Percentage errors trainset	2,33
Percentage errors testset	2,00

The results obtained using SVMTorch varied widely for different runs. Apparently, in this small dataset, the selected examples have lot of influence on the solution.

The above parameters were fixed, to measure the results when features were removed, as research indicated that some of the features add more noise then they improve the dataset. The results are summarized in the table below.

Removed feature	Train set error	Test set error
Petal width	4,37	6,53
petal length	3,17	6,00
Sepal width	2,73	4,67
Sepal length	3,60	5,73

This shows that sepal width is a good candidate for removal.

6.2.4 Conclusion

This test shows, that a more sophisticated problem that contains a multiclass output, can also be solved with the workbench. Problems such as the encoding and decoding, the creation of multiple variants and the recombination of results are all hidden from the user.

6.3 NUON wind prediction

NUON is the biggest wind energy producer in The Netherlands. Approximately 50% of the 600MW installed wind energy in The Netherlands is managed by NUON. The produced energy is sold 24 hours in advance on the energy market. It is therefore important to know the amount of energy that will probably be produced, for differences between predicted and delivered amounts either cost money or result in a price that is sub optimal. The cost of power imbalance is around 6 Euro/MWh. This means that the energy produced by the wind parks is worth 6 Euro less than that of perfectly predictable energy sources.

For a part of the total wind park, hourly data is available for the last two years. Predictions are based on predicted wind speeds and wind directions for different locations in the Netherlands. These weather predictions are also available 24 hours in advance. Because of the inherent error in these data, it will be impossible to create a perfect forecast.

This test presents an attempt to improve the current predictions of the energy amount produced by the wind parks.

6.3.1 Dataset description

The input features are the forecasted wind speed and wind direction of four locations in the Netherlands (DeKooy, Hoogeveen, Leeuwarden and Schiphol), as well as an average of the forecasted wind speed and wind direction at fifteen sites. These values are available 24 hours in advance from the national weather agency. The dataset consists of 18935 examples of data. This is the data for the period February 2001 to March 2003. Each example represents one hour of data in that year. Each example consists of ten input features and one output. The wind speed is measured in meters per second, discretized to 1 m/s intervals. The wind direction is measured in degrees. The locations of the forecasts and the locations of the wind parks are not the same. The output feature is the total energy output of a set of wind parks in the given hour.

Name	Type	Output?	min	max	avg	st.dev
Direction NL	continuous	no	0.00	360.00	188.58	91.83
Direction DeKooy	continuous	no	0.00	360.00	186.81	95.41
Direction Hoogeveen	continuous	no	0.00	360.00	184.82	92.07
Direction Leeuwarden	continuous	no	0.00	360.00	185.79	94.49
Direction Schiphol	continuous	no	0.00	360.00	184.86	94.60
Speed NL	continuous	no	0.00	13.00	4.38	1.91
Speed DeKooy	continuous	no	0.00	15.00	5.73	2.42
Speed Hoogeveen	continuous	no	0.00	23.00	4.25	1.88
Speed Leeuwarden	continuous	no	0.00	21.00	4.63	2.01
Speed Schiphol	continuous	no	0.00	14.00	5.19	2.24
Production	continuous	yes	-72.00	84235.00	18366.91	19524.47

6.3.2 Related work

There are not many articles concerning the application of machine learning techniques to the prediction of complete wind farms.

Li et al. [2001] use neural networks for the prediction of the energy output for one turbine. Their results match the actual performance.

Denison et al. [2001] predict the wind energy production on the longer term, based on historical data, by using a Bayesian multivariate adaptive regression spline model.

Nielsen and Nielsen [1999] use statistical models to create production forecasts using meteorological data for wind farms. This led to the creation of WPPT.

Beyer et al. [1999] use the numerical weather prediction model HIRLAM for wind parks in Germany. RMS error rates of 12 to 20% are reported.

6.3.3 Test

After all data was gathered, a new project was created in the workbench, and the data was inserted using the 'Upload rawdata' module.

Correlations of the features show that the wind directions as well as speeds are highly correlated, which was expected, as the Netherlands is not a big country. The output is highly correlated with the wind speeds, and slightly with the wind directions.

Plots projecting each of the features against the another, show a seasonal pattern where the wind speed or energy drops during July and August (figure 6.15). The wind speed is highest when the direction is around 230 degrees (figure 6.16 left). The plots for 'wind speed vs output' show that the wind speed is only an upper bound for the production (figure 6.16 right). For average wind speeds the production varies between zero and the upper bound. At high wind speeds the turbines have to be shutdown to prevent them from being damaged.

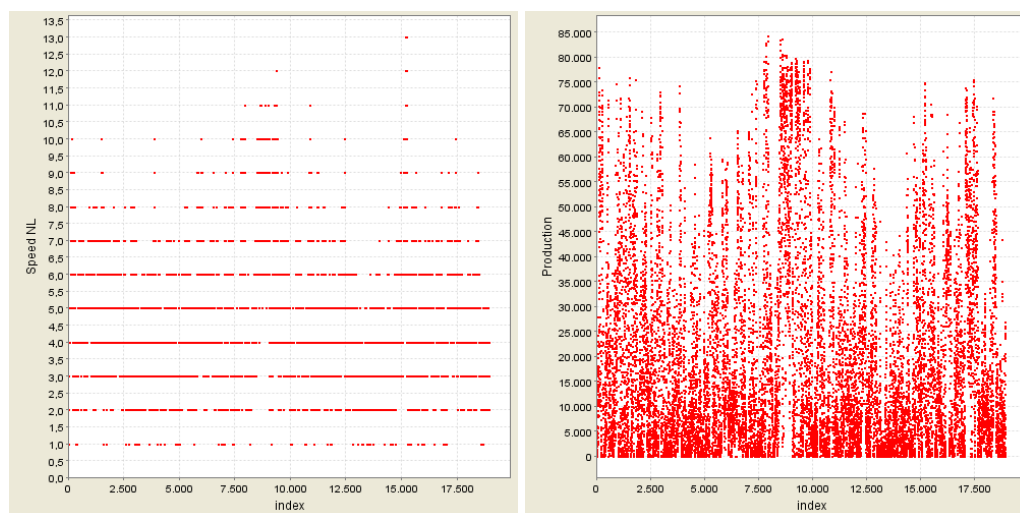


Figure 6.15: Distribution of wind speed and production over 2 years.

The first test was to apply the k-nearest neighbour module to the data. To find suitable parameters only a subset of the data was taken, as the complete dataset would take a long time to train. The script that was constructed started with the normalize/scale module, which normalizes all inputs and the output to mean zero

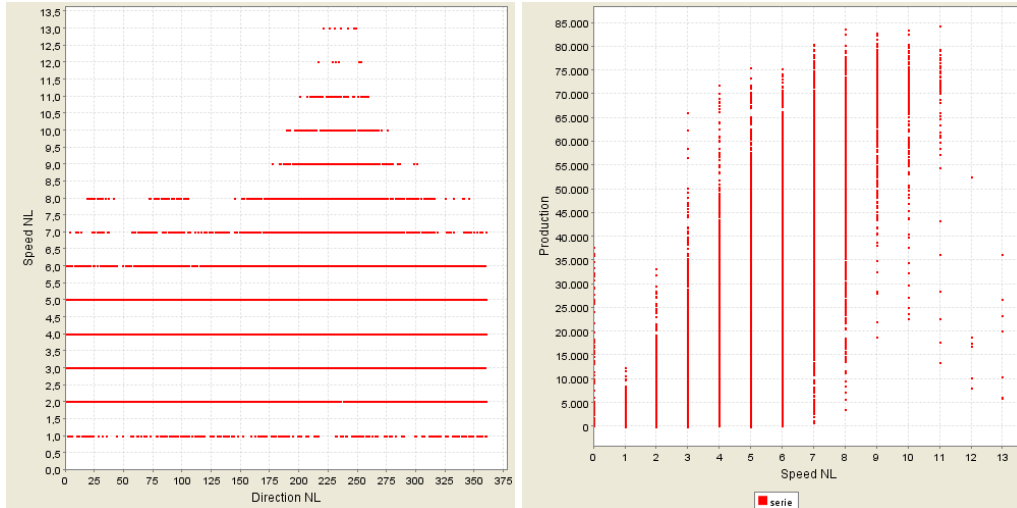


Figure 6.16: Left: wind direction vs wind speed, right: speed vs production

and standard deviation one. The output is normalized too, because several techniques work better that way, and all following results become comparable. Then a random subset of 30% is taken from the data. This is passed to a cross validation module, which is set to create four straight folds. This dataset is passed to the kNN module which varies k over one to ten, and weighs both by distance and average.

The density plot module shows that good results are obtained at $k = 6$ and weighing by distance. The plot shows the train set error (red) and test set error (green), while going over k and the weighing method. When $k = 6$ and weighing by distance, the results have a mean squared error on the train set of 0,04 and on the test set 0,29. The 'random subset' module was then set to use all data, and the test is repeated with k and the weighing method fixed. The following results are obtained.

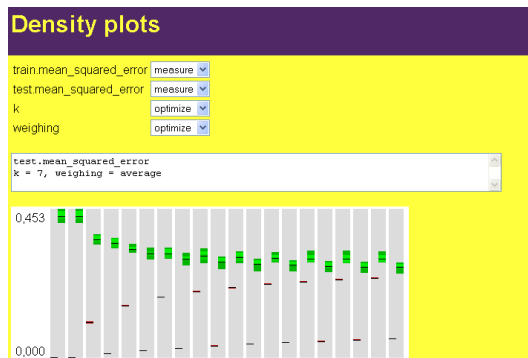


Figure 6.17: 'Density plot kNN'

Error measure	Average	Min	Max
MSE train set	0.03	0.03	0.03
MSE test set	0.23	0.22	0.24
MAE train set	0.11	0.11	0.11
MAE test set	0.32	0.31	0.32

The next tests were performed using the Lasso module. They are based on the same script in the workbench, with the k -nearest neighbour module being replaced by the Lasso module. Two kernels were tried: the RBF and the polynomial kernel.

The RBF kernel was tried for different values of kappa and sigma. The density plot module showed that for different values for kappa, there were always some values of sigma that performed best. The differences between these pairs were negligible. For both train and test set, the MSE varied around 0.31.

The polynomial kernel was tried for different degrees, and different values for gamma and the coefficient, in addition to kappa. The polynomial of the fourth degree was the most effective. For parameters kappa=0.7, gamma=0.3 and coefficient=3.0 the best results were obtained:

Error measure	Average	Min	Max
MSE train set	0.29	0.29	0.29
MSE test set	0.30	0.29	0.31
MAE train set	0.39	0.38	0.39
MAE test set	0.39	0.39	0.40

When the amount of examples that is allowed in the dataset is increased, these values do not change.

Modified wind directions

The wind directions varied over 0 to 360 (before scaling them around zero). The jump from 360 to 0 is quite significant, whereas in reality there is no distinction. To clear this problem, the directions were all mapped onto a circle. That way, the values become better comparable. The original wind directions are then removed from the dataset, and sine and cosine values of the mappings are added.

To incorporate this mapping, the 'map on circle' module was added to the script, right after the normalizing module. The settings in the scaling module were modified, so that it does not scale the wind directions. These are converted by the next module to the sine and cosine. The rest of the script (random subset, cross validation and Lasso) was rebuilt on top of the circle module.

To measure the results, the polynomial kernel was used, again with degree 4. The initial results (with all settings kept the same) improved over the previous test.

Error measure	Average	Min	Max
MSE train set	0.26	0.25	0.28
MSE test set	0.27	0.27	0.27
MAE train set	0.37	0.36	0.38
MAE test set	0.37	0.37	0.38

When the number of examples that was used was increased to 70% the numbers did not change. By varying kappa, gamma and coefficient, it was found that the parameters were still optimal.

Adding lags

Because of the discretized wind speed values, we assume that better predictions are obtained by incorporating lags of previous wind speeds. The script was modified so that after the mapping of directions onto circles all input features were lagged for periods 1, 2 and 5 time steps. The incorporation of these lags was done by the 'timeseries embedding module'. The script was ran three times, each time with a different amount of lag added to each of the inputs. For lag=5, this resulted in a dataset consisting of 90 input features (for each of the five locations, the wind speed and the sine and cosine of the direction are available, and these values are repeated for t-1, t-2, t-3, t-4 and t-5).

The tests were performed with the same settings as for the previous test. The results are summarized below.

Error measure	Average	Min	Max
MSE train set	0.25	0.24	0.26
MSE test set	0.29	0.26	0.31
MAE train set	0.36	0.36	0.37
MAE test set	0.39	0.37	0.39

Table 6.1: Error measures for Lasso with polynomial kernel on dataset with modified wind directions and a lag of one time step

Error measure	Average	Min	Max
MSE train set	0.26	0.26	0.27
MSE test set	0.31	0.29	0.32
MAE train set	0.37	0.36	0.37
MAE test set	0.39	0.38	0.40

Table 6.2: Error measures for Lasso with polynomial kernel on dataset with modified wind directions and a lag of two time steps

Error measure	Average	Min	Max
MSE train set	0.25	0.24	0.26
MSE test set	0.30	0.28	0.32
MAE train set	0.36	0.36	0.37
MAE test set	0.39	0.38	0.40

Table 6.3: Error measures for Lasso with polynomial kernel on dataset with modified wind directions and a lag of five time steps

Because of the good initial results with k -nearest neighbours and the varying results with Lasso, an additional test was performed with the kNN module, with lag=5 on 30% of the examples. This resulted in the following measures:

Error measure	Average	Min	Max
MSE train set	0.05	0.04	0.05
MSE test set	0.24	0.23	0.24
MAE train set	0.14	0.14	0.14
MAE test set	0.34	0.33	0.34

As results for kNN improved as more examples were used for training, the test is repeated with all examples, and a lag of one time step. These results are summarized:

Error measure	Average	Min	Max
MSE train set	0.03	0.03	0.04
MSE test set	0.20	0.19	0.20
MAE train set	0.12	0.12	0.12
MAE test set	0.29	0.29	0.30

6.4 Conclusion

The application of the workbench to this real world dataset shows that the workbench can be used for all sorts of datasets. The use of different modules made it

easier to try different combinations.

Two techniques were applied to the NUON wind speed dataset. These techniques were k -nearest neighbours and Lasso. Successive improvements were made to the dataset to obtain better results.

The k -nearest neighbour technique was able to obtain very good results. This is probably due to the huge amount of examples. Good examples are always near and the pattern is obviously stable.

Lasso is a sophisticated technique. From the comparison with the kNN technique, it seems that modelling the results of the predictions on the energy output is hard. The advantage is that the performance is more stable than that of the kNN technique.

Conversion of the wind directions to a more natural format improved the predictions. Results for adding lags vary. Apparently the addition adds less information to the data than the increase in complexity is able to justify. Adding a small amount of lag improves the results (minimum average MSE and MAE drop slightly), but the average errors on the test set increase.

The use of Lasso with a fourth degree polynomial kernel results in average errors of 9-10%. By using k -nearest neighbours with distance based weighing of the 6 nearest neighbours the average error is expected to be in the range 3-8%.

The direction in which education starts a man will determine his future life.

Plato (427 BC - 347 BC)

7

Conclusions

The previous chapters have described the construction, as well as the use of the workbench. The design chapter described a use-case. After the choice of the language and libraries, the use-case was extended to obtain a data structure and instructions for the implementation. The requirements from chapter 2 were checked to see that the design matches the requirements.

The implementation chapter described only the noteworthy points of the implementation, as most of the implementation was implemented according to the design. Especially the extensions of the design were described, such as the persistence of the data, and enhancements that will ease the extension of the workbench. The last part of the chapter described all modules that were created.

The evaluation chapter showed the workbench in practice. Three datasets were used to test the workbench. In all three cases results comparable to existing results were found, or the existing results were improved. This showed that this workbench is a tool that is easy to use, and which allows tests to be performed fast and with different techniques.

The requirements chapter introduced several requirements. Each of those is now checked to see if the workbench implements it. The first was flexibility. The workbench is clearly flexible. Different modules can be coupled to create different tests. The forward propagation of the dataset allows the dataset to be shaped in different ways. The back propagation of results gives each of the modules a change to apply further calculations or provide summaries relevant to that module. The choice for Java and dynamic module integration enables a large number of users to add their own custom modules.

Insight in the problem and results is provided by several modules. The dataset plot module gives an idea of the dataset. The visual XSLT module, in combination with the plugins, allows users to retrieve the relevant results and plot those to obtain better parameter indications. The wrapper output module shows the actual results and gives an idea what is going wrong.

Multiple tests are easily started and executed. Selection of intervals for parameters and running tests for all combinations requires little work for the user. The distribution of wrappers over multiple clients reduces the time one has to wait

for all results to be available. These clients can be of different types. Clock speed and memory do not need to be equal, so that slower clients do not hamper fast clients. The JavaSpace and workbench synchronise the integration of results. Even the operating systems need not be equal, as long as the tools are compiled for and installed on the target platforms.

Testing of hypothesis is possible by the ease with which users can start different tests and compare the results.

The use of different machine learning techniques on a problem is made easy by wrapping the tools inside modules and wrappers. To the user, a tool is no more than a module with some settings. The module and wrapper perform all translations to and from the data format that the tools use. The workbench with the current list of modules supports already eight different tools.

The integration of these tools is fairly easy, and need not be done by the programmer of the original tool. All that is required is that someone writes a user-interface JSP for the tool, a class that inherits from 'Module', which creates the wrapper and fills it with data, and create a class that inherits from 'Wrapper' and calls the tool. The current modules already contain a variety of techniques so that the new classes can re-use code and ideas from these modules.

In the introduction, the following objective was stated:

The workbench should provide the user with a flexible tool that is able to give insight in the problem at hand, perform lots of tests in a short period of time to test hypotheses, and make it easier to do so with different machine learning models.

This objective has clearly been met. The workbench is flexible and able to handle substantial tests. It supplies the user with a good platform to start process of solving the problem, from classification to prediction problems.

8

Future work

Although the workbench is already a good tool that can be employed in various situations, there is place for some improvements. Some of these improvements only require the creation of new modules, some others stem from working with the workbench and require some changes in the structure.

The workbench is currently an open platform. Anybody who knows the name of the server can use the workbench. Some sort of user management would be a good addition. Basic checks are to allow users access. Further management could include managing the claimed resources.

Scheduling of jobs is currently performed by the JavaSpace. The JavaSpaces technology makes no guaranties about the order in which objects are retrieved from the space, but in practice this seems to be first-in, first-out. Although this might be fair, this results in huge scripts claiming all the systems, and leaving small scripts waiting. An improvement would be to allow all projects or users an equal chance to access the systems.

Other improvements are to provide the user with a better description of what systems and the workbench are doing. During large tests it may seem like nothing happens. Some information on the status of progress and running threads would show the user that work is being performed.

The amount of wrappers can be too large if several combinations are tried at once. The current version gives no indication of the number of wrappers that will be started. By providing an indication of this number just before starting the script some accidents can be prevented. In addition, a limit on the number of wrappers would also help.

The workbench contains a number of modules, but the functionality is extended with every module that is added. More specialized script modules for applying calculations to features, such as mathematical functions, would allow even more complex transformations of the dataset.

More visualization modules would help too. Three-dimensional plots can convey an even better idea of the dataset and results. Showing relations between results and dataset might also improve the understanding of why tools perform the way they do. Offering the user more analysis tools, such as auto-correlation plots,

would also be a good enhancement.

Other enhancements can improve the usability further. A clipboard for each result, where the user can log information such as the reason for the test, expectations and results would give results a longer long levity. The layout and structure of the modules could be improved too. These are all minor enhancements that are easy to implement and do not influence the existing functions.

The range of enhancements shows that the workbench is truly a platform that can be extended to suit different needs, and is ready to grow beyond the original needs and expectations.

9

Glossary

Cache

A cache is a temporary storage to access data with less overhead. To reduce disk accesses, a memory-based cache can be used. To reduce network traffic, the cache can be implemented in either memory or disk.

Class

Definition of an object. Specifies what data is included in the object, and which methods are available for interacting with this data.

Client

A computer that accepts tasks from the main computer. It provides processing power to the rest of the system.

Data structure

A structure of objects containing data that might be needed by the workbench.

Database management system

The complete system around the database. The management system provides functions to tune the performance of the database, to store data and to request data that has certain properties from the database.

Dataset

The dataset is a collection of examples. Each example consists of values for the input and output features. In the workbench, this collection is extended with the types of the features, and an indication that tells the system to use the example for training or for testing.

Deep copy

When assigning an object to a variable, only the reference to the object is copied. The disadvantage is that modifications of the data lead to modifications of the original object, which might lead to problems. The solution is to make a deep copy.

That way, all objects that are referenced from the object are copied as well so that all modifications are made only in this structure.

Epoch

A complete training round in neural networks. All examples are propagated once forward and backward.

Example

A set of values that indicate the values of the outputs that correspond to certain input values.

Feature

Set of corresponding values. Each example contains values for each of the features.

Form

Part of a web page where the user can enter values.

Framework

The workbench without the modules. The framework performs the functions to interconnect modules and allow the user to access the modules.

Garbage collector

Part of the Java architecture. The garbage collector locates all objects that take memory, but are not usable anymore (because there are no more references to access them). This frees the programmer from explicitly deallocating memory.

GUI-builder

Program that assists the programmer during the creation of visual screens for other programs.

HTML

HyperText Markup Language. Language to add layout instructions and hyperlinks to text. The language is mainly used on the Internet.

JSP

Java Server Page. Script page in a web server that combines HTML and Java code to provide interactive web pages to the user.

LaTeX

Language that adds layout instructions to text. Mainly used for scientific publications.

Link

A small piece of text in an HTML page. By clicking the link, the user is shown another page.

Machine learning

This field attempts to allow computers to acquire knowledge from examples and apply this knowledge to give good predictions for examples that have not been

shown before.

Module

Part of the workbench. Modules are custom parts that increase the flexibility of the workbench. Each module should provide a user-interface, and a class that makes the necessary changes to the dataset. Modules for machine learning techniques require another class of the wrapper type. This wrapper interfaces with the underlying tools.

Object

Instance of a class. An object can hold run-time data that the program can read and modify.

Persistence

Persistence is the storage of data in less-volatile memory, so that consecutive runs of a program (the workbench) can use the data from a previous run.

Plugin

A JSP or Servlet that can be used by different modules. The visualization modules in the workbench use plugins to perform common tasks, such as making plots or converting data to another format.

Proxy

A proxy in relation to JINI is an object that clients can use to access a service. When a client wishes to use a service, it asks the registry for a proxy. This proxy 'knows' how to communicate with the service. Calls from the client to the service to perform a certain task, are routed via the proxy.

Reference

The Java equivalent of a pointer. A reference points to an object.

Serialization

Serialization is a process in which an object, along with all the objects and data it is dependent on, is transformed into a stream of data. This means that the data contained in the objects is stored in a format from which the original object, and all related objects, can be reconstructed. This is used for persistence and the transfer of objects to the JavaSpace.

Service

A program, connected via the JINI registry, that provides services to other processes.

SQL

SQL is a language to modify data in databases, and to request data from them.

Submit

The process of sending the data from form in the browser to the web server.

Test

Either the complete execution of a test script, or the execution of a tool. The last means that a tool is trained on a train set, and the performance is measured on a test set.

Test script

A script that describes all the steps that are to be performed during the test. This includes modifications of the dataset, as well as the execution of tools using certain parameters.

Textarea

HTML user-interface element. This element is used to allow the user to enter multiple lines of text and send it to the workbench.

Thread

Threads are separate lines of execution in a single process. Threads act as if they are running at the same time (which is possible if the system uses multiple processors). The advantage of using threads is that slow processes, or processes that are waiting for data, do not stall the system. Each thread is given some time to execute, and then another thread can run for some time.

Tool

Computer program that is supplied data for training on a problem and testing the performance on another dataset.

User-interface

The user-interface is the part of the program that allows the user to interact with the program. This can be anything from command-line options to graphical screens with textareas, although it is most commonly used for the graphical screens (also known as graphical user-interface or GUI).

Web-based application

Computer application that has a user interface based on HTML pages (or related standards).

Web server

A program that serves data, mostly based on HTML. The workbench makes use of a Java web server with support for Servlets and JSPs, which means that the web server translates requests for pages, and forwards the request to a Servlet or JSP. The results are sent back to the requesting program (usually a web browser)

Workbench

A workbench is a program that can be used to explore, transform and process data in a variety of related ways. It acts as a container for data to which tools can be applied. The order of application of the tools determines the shape of the result.

Wrapper

An object containing all the data and parameters that are required to start a tool. The wrapper can be transferred to a client that converts the data into a format that the tool can use, executes the tool, and extracts the results.

Bibliography

- D. Aha and R. Bankert. Cloud classification using error-correcting output codes, 1997. URL <http://citeseer.nj.nec.com/aha96cloud.html>.
- C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines - a kernel approach. In *Proc. of the 8th IWFHR*, pages 49–54, 2002. URL <http://citeseer.nj.nec.com/bahlmann02line.html>.
- R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
- A. Ben-Hur, D. Horn, H.T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001. URL <http://www.ai.mit.edu/projects/jmlr/papers/volume2/horn01a/rev1/horn01a1r.pdf>.
- K. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2(2):1–14, 2000. URL <http://www.acm.org/sigs/sigkdd/explorations/issue2-2/bennett.ps>.
- S. Berchtold, B. Ertl, D.A. Keim, H. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional spaces. In *Proc. 14th IEEE Conf. Data Engineering, ICDE*. IEEE Computer Society, 23–27 1998. ISBN 0-8186-8289-2. URL <http://citeseer.nj.nec.com/berchtold98fast.html>.
- H.G. Beyer, D. Heinemann, H. Mellinghoff, K. Monnich, and H. Waldl. Forecast of regional power output of wind turbines, 1999. URL http://ehf.uni-oldenburg.de/wind/download/ewec99/prediction_ewec99_un%i-ol.pdf.
- C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. URL <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. ISSN 1384-5810. URL <http://citeseer.nj.nec.com/burges98tutorial.html>.
- C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. URL <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- R. Collobert, S. Bengio, and J. Marithoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002. URL <http://www.idiap.ch/~bengio/publications/pdf/rr02-46.pdf>.
- S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993. URL <http://citeseer.nj.nec.com/cost93weighted.html>.
- D.G.T. Denison, P. Dellaportas, and B.K. Mallick. Wind speed prediction in a complex terrain, 2001. URL <http://citeseer.nj.nec.com/401179.html>.
- T. G. Dietterich and G. Bakiri. Error-correcting output codes: a general method for improving multiclass inductive learning programs. In T. L. Dean and K. McKeown, editors, *Proceedings of the Ninth AAAI National Conference on Artificial Intelligence*, pages

- 572–577, Menlo Park, CA, 1991. AAAI Press. URL <http://citeseer.nj.nec.com/dietterich91errorcorrecting.html>.
- T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995. URL <http://citeseer.nj.nec.com/dietterich95solving.html>.
- T. Downs, K.E. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, 2001. URL <http://www.ai.mit.edu/projects/jmlr/papers/volume2/downs01a/downs01a.pdf>.
- R.A. Fisher. The use of multiple measurements in taxonomic problems, 1936.
- R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, Chichester, 2nd edition, 1987. ISBN 0-471-49463-1.
- J.E.F. Friedl. *Mastering Regular Expressions, 2nd Edition*. O'Reilly, 2002. ISBN 0-596-00289-0. URL <http://www.oreilly.com/catalog/regex2/>.
- G. Fung and O. L. Mangasarian. Data selection for support vector machine classifiers. In *Knowledge Discovery and Data Mining*, pages 64–70, 2000. URL <http://citeseer.nj.nec.com/fung00data.html>.
- E. Gamma, R. Helm, and R. Johnson. *Design patterns; elements of reusable object-oriented software*. Addison-Wesley, 1995. ISBN 0-201-63361-2.
- C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001. URL <http://www.ai.mit.edu/projects/jmlr/papers/volume2/gentile01a/gentile01a.pdf>.
- S.R. Gunn. Support vector machines for classification and regression. Technical report, University of Southampton, 1998. URL <http://www.ecs.soton.ac.uk/~srg/publications/pdf/SVM.pdf>.
- E.B. Kong and T.G. Dietterich. Error-correcting output coding corrects bias and variance. In *International Conference on Machine Learning*, pages 313–321, 1995. URL <http://citeseer.nj.nec.com/kong95errorcorrecting.html>.
- L. Kuncheva. Switching between selection and fusion in combining classifiers: An experiment, 2002. URL <http://citeseer.nj.nec.com/kuncheva02switching.html>.
- S. Li, D.C. Wunsch, E.A. O'Hair, and M.G. Giesselmann. Using neural networks to estimate wind turbine power generation energy conversion. *IEEE Transaction on energy conversion*, 16(3):276–282, September 2001. URL http://www.ece.umn.edu/acil/Publications/JOURNAL/USING_NEURAL_NETWORK%S.pdf.
- O. Mangasarian and D. Musicant. Active support vector machine classification. Technical Report 00-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, April 2000. URL <ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-04.ps>.
- T. Masters. *Practical Neural Network Recipes in C++*. Academic Press, Boston, 1993. ISBN 0-12-479040-2.
- W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

- D. Musicant and A. Feinberg. Active set support vector regression. Technical Report 01-02, Department of Mathematics and Computer Science, Carleton College, Northfield, Minnesota, July 2002. URL <http://www.mathcs.carleton.edu/faculty/dmusicant/tr0102.ps>.
- H. A. Nielsen and T. S. Nielsen. Using meteorological forecasts in on-line predictions of wind power. chapter 5: Estimation methods, 1999. ISBN: 87-90707-18-4.
- G.P. Noone. Radar pulse classification using support vector machines, 2000. URL <http://www.ips.gov.au/IPSHosted/NCRS/wars2000/commc/noone.pdf>.
- A. M. Palau and R. R. Snapp. The labeled cell classifier: A fast approximation to k nearest neighbors, 1997. URL <http://citeseer.nj.nec.com/palau97labeled.html>.
- J.C. Principe, N.R. Euliano, and W.C.Lefebvre. *Neural and adaptive systems : fundamentals through simulations*. Wiley, New York, 2000. ISBN 0-471-35167-9.
- G.L. Ritter, H.B. Woodruff, S.R. Lowry, and T.L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 1975. URL <http://citeseer.nj.nec.com/context/24540/0>.
- V. Roth. Sparse kernel regressors. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Proceedings of the International Conference Vienna (ICANN'01)*, volume 2130 of *Lecture Notes in Computer Science*, pages 339–346. Springer-Verlag, 2001. ISBN 3-540-42486-5. URL http://www.informatik.uni-bonn.de/~roth/icann_pdf.pdf.
- D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing; explorations in the microstructure of cognition. Vol. 1. Foundations*. Computational models of cognition and perception. MIT Press, 1986. ISBN 0-262-18120-7.
- J. Salomon. Support vector machines for phoneme classification, 2001. URL <http://citeseer.nj.nec.com/salomon01support.html>.
- B. Schölkopf and A. Smola. *Learning with Kernels*. Adaptive Computation and Machine Learning. MIT Press, 2002. ISBN 0-262-19475-9. URL <http://www.learning-with-kernels.org>.
- R. Setiono and H. Liu. Symbolic representation of neural networks. *IEEE Computer*, 29(3):71–77, 1996. URL <http://citeseer.nj.nec.com/article/setiono96symbolic.html>.
- D.B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *International Conference on Machine Learning*, pages 293–301, 1994. URL <http://citeseer.nj.nec.com/skalak94prototype.html>.
- F. Smeraldi, N. Capdevielle, and J. Bigun. Face authentication by retinotopic sampling of the gabor decomposition and support vector machines. In *Audio and Video based Person Authentication - AVBPA99*, pages 125–129, 1999. URL <http://citeseer.nj.nec.com/smeraldi99face.html>.
- A. Smola and B. Schölkopf. A tutorial on support vector regression. NeuroCOLT Technical Report NC2-TR-1998-030, University of London, 1998. URL <http://www.kernel-machines.org/papers/tr-30-1998.ps.gz>.
- S.S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.
- M. Tipping. The relevance vector machine. In Sara Solla, Todd Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems (NIPS'99)*, volume 12, pages 652–658, Cambridge, Massachusetts, USA, 1999. The MIT Press. ISBN 0-262-19450-3. URL ftp://ftp.research.microsoft.com/users/tipping/rvm_nips.ps.gz.

- V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In Michael Mozer, Michael Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems (NIPS'97)*, volume 9, pages 281–287, Cambridge, Massachusetts, USA, 1997. The MIT Press. ISBN 0-262-10065-7. URL <http://www.kernel-machines.org/papers/vapgolsmo96.ps.gz>.
- T. Welch. Bounds on the information retrieval efficiency of static file structures. Technical Report 88, MIT, 1971. URL <http://citeseer.nj.nec.com/context/448058/0>.
- I.H. Witten and E. Frank. *Data mining : practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 2000. ISBN 1-55860-552-5.

Wind energy production forecasting

Floris Ouwendijk ^a Henk Koppelaar ^a Rutger ter Borg ^b
Thijs van den Berg ^b

^a Delft University of Technology, PO box 5031, 2600 GA Delft, the Netherlands

^b NUON Energy Trade and Wholesale, Spaklerweg 20, Amsterdam, the Netherlands

Abstract

This paper presents a study to forecast the total production of some wind parks in The Netherlands. The predictions are based on several forecasts of the wind speed and wind direction. Two techniques are used: k -nearest neighbours and Lasso. These techniques are applied to three versions of the problem. The initial version scales the features. The second version modifies the wind directions to a more natural data format. The third version incorporates lags in the dataset. Each version improves the results somewhat, with the k -nearest neighbour technique having the lowest errors, and Lasso the most stable performance.

1 Introduction to the problem

NUON is the biggest wind energy producer in The Netherlands. Approximately 50% of the 600MW installed wind energy in The Netherlands is managed by NUON. The produced energy is sold 24 hours in advance on the energy market. It is therefore important to know the amount of energy that will probably be produced, for differences between predicted and delivered amounts either cost money or result in a price that is sub optimal. The cost of power imbalance is around 6 Euro/MWh. This means that the energy produced by the wind parks is worth 6 Euro less than that of perfectly predictable energy sources.

For a part of the total wind park, hourly data is available for the last two years. Predictions are based on predicted wind speeds and wind directions for different locations in the Netherlands. These weather predictions are also available 24 hours in advance. Because of the inherent error in these data, it will be impossible to create a perfect forecast.

This paper presents an attempt to improve the current predictions of the energy amount produced by the wind parks.

2 Previous work

There are not many articles concerning the application of machine learning techniques to the prediction of complete wind farms.

Li et al. [3] use neural networks for the prediction of the energy output for one turbine. Their results match the actual performance.

Denison et al. [2] predict the wind energy production on the longer term, based on historical data, by using a Bayesian multivariate adaptive regression spline model. Nielsen and Nielsen [4] use statistical models to create production forecasts using meteorological data for wind farms. This led to the creation of WPPT.

Beyer et al. [1] use the numerical weather prediction model HIRLAM for wind parks in Germany. Error rates of 12 to 20% are reported.

Unfortunately, because of the lack of articles regarding the same type of problem, it is hard to compare the results. We have been able to measure the error on the same dataset using the current model in use at NUON. These results are described in the section of the current model.

3 Dataset description

The input features are the forecasted wind speed and wind direction of four locations in the Netherlands (DeKooy, Hoogeveen, Leeuwarden and Schiphol), as well as an average of the forecasted wind speed and wind direction at fifteen sites. These values are available 24 hours in advance from the national weather agency. The dataset consists of 18935 examples of data. This is the data for the period February 2001 to March 2003. Each example represents one hour of data in that year. Each example consists of ten input features and one output. The wind speed is measured in meters per second, discretized to 1 m/s intervals. The wind direction is measured in degrees. The locations of the forecasts and the locations of the wind parks are not the same. The output feature is the total energy output of a set of wind parks in the given hour.

Name	Output?	min	max	avg	st.dev
Direction NL	no	0.00	360.00	188.58	91.83
Direction DeKooy	no	0.00	360.00	186.81	95.41
Direction Hoogeveen	no	0.00	360.00	184.82	92.07
Direction Leeuwarden	no	0.00	360.00	185.79	94.49
Direction Schiphol	no	0.00	360.00	184.86	94.60
Speed NL	no	0.00	13.00	4.38	1.91
Speed DeKooy	no	0.00	15.00	5.73	2.42
Speed Hoogeveen	no	0.00	23.00	4.25	1.88
Speed Leeuwarden	no	0.00	21.00	4.63	2.01
Speed Schiphol	no	0.00	14.00	5.19	2.24
Production	yes	-72.00	84235.00	18366.91	19524.47

Correlations of the features show that the wind directions as well as speeds are highly correlated, which was expected, as the Netherlands is not a big country. The output is highly correlated with the wind speeds, and slightly with the wind directions.

Plots projecting each of the features against the another, show a seasonal pattern where the wind speed or energy drops during July and August (figure 1).

The wind speed is highest when the direction is around 230 degrees (figure 2 left). The plots for 'wind speed vs output' show that the wind speed is only an upper bound for the production (figure 2 right). For average wind speeds the production varies between zero and the upper bound. At high wind speeds the turbines have to be shut-down to prevent them from being damaged.

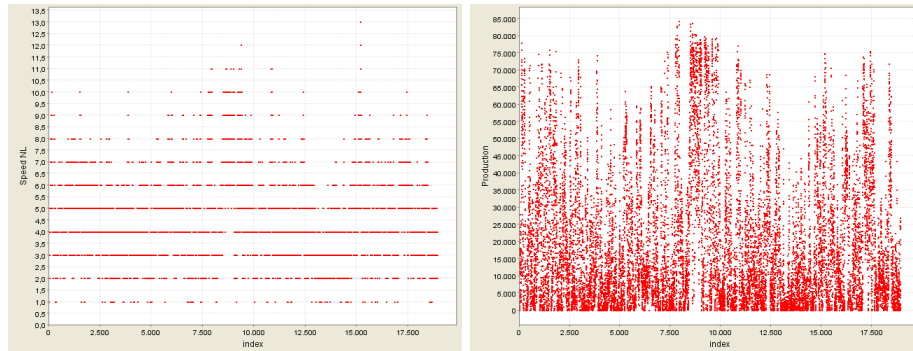


Figure 1: Distribution of wind speed and production over 2 years.

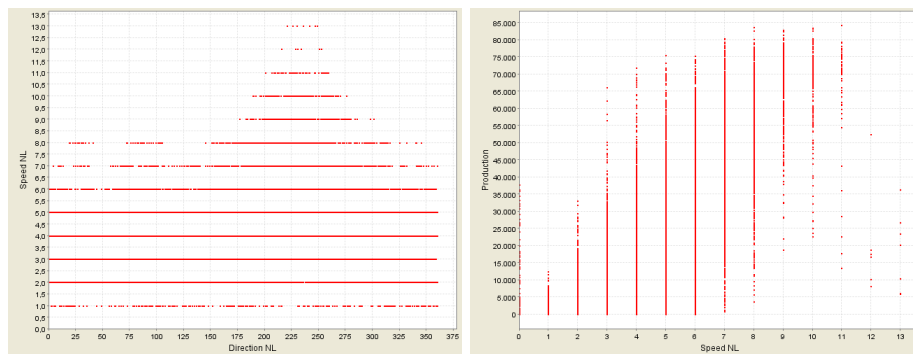


Figure 2: Left: wind direction vs wind speed, right: speed vs production

4 Our approach

This research was performed using the 'Workbench for Machine Learning Techniques' [5]. This is a tool that allows the user to easily create test scripts for testing machine learning techniques and applying these techniques to problems such as this wind energy prediction problem. These scripts consist of modules placed in a stackable architecture.

Each of the modules in the script modifies the dataset. The modified dataset is passed on to subsequent modules, or used to apply a machine learning algorithm to

a train and test set. The user-interface allows the parameters to the modules and learning techniques to be varied. The execution of the techniques is distributed over multiple computer systems. After the execution, different error measures are available. These measures can be set-off against the parameters using visualization modules. This allows us to quickly find optimal parameters.

Two machine learning techniques are used. The first is the k -nearest neighbour algorithm for regression problems. The algorithm is initialised with a set of examples x_i and their corresponding output y_i . The distance to all examples is calculated for each new vector:

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^N (x_{im} - x_{jm})^2}$$

The examples are ordered according to this distance, and the k examples with the least distance are selected. The predicted value for the new vector is then either the average of these points, or a weighed sum where examples with a smaller distance contribute more to the predicted output.

The second technique is Lasso [6]. Lasso is based on relevance vector machines, which produce sparse models using kernel functions. It has the advantage that the error distributions for train set and test set are comparable. Predictions are calculated by

$$f(x) = \alpha_0 + \sum_{i=1}^n \alpha_i K(x_i, x)$$

in which α is a vector weights for each of the examples, and K is the kernel function. To obtain α , the problem

$$\begin{aligned} &\text{minimise} && \|y - 1\alpha_0 - \mathbf{K}\alpha\|_2^2 \\ &\text{subject to} && \|\alpha\|_1 \leq \text{kappa} \end{aligned}$$

is solved. Kappa controls the complexity of the solution. Two kernel functions were used; the radial basis function (RBF) kernel and the polynomial kernel.

$$\begin{aligned} \text{Polynomial} && K(u, v) &= (\text{gamma } u \cdot v + \text{coef})^d \\ \text{RBF} && K(u, v) &= e^{-(u-v)^2/\sigma^2} \end{aligned}$$

Three approaches are taken to come to improve the predictions that are currently obtained (section 4.1). The first approach was to use scaled data. The second modifies the wind directions in the dataset to a more natural system. The third approach extends the dataset even further by adding lags of the input features to the dataset.

The script used in the workbench consists of several modules. The first module in the script is the 'normalize' module. This module scales all inputs and the output to mean 0 and standard deviation 1. The reason to do so is that most techniques, including k -nearest neighbours and Lasso, work better if these values are close to zero. After this scaling all values lie between -1 and 3 .

The second module is a 'random subset' module. This module picks a random set of examples from the dataset. The size of this set was set to 30%. The reason to do so is that the dataset contains a large number of examples, of which many are similar. This slows down the training while the results are presumed to be similar when compared to training on the complete dataset. During the tests, this presumption will be tested.

The third module is a cross-validation module that creates four folds. This allows us to see if there are large variations in the results.

The fourth and final module is the machine learning technique. For the first attempt, this is initially the k -nearest neighbour module, where k can be varied, and has two weighing modes. Thereafter the Lasso module is used, which allows kappa to be varied, as well as choosing a kernel function and its parameters.

4.1 Current model

The current model in use at NUON is a second order polynomial. This polynomial is fit on the complete dataset to obtain the least mean squared error solution.

Using this model, predictions for all examples in the dataset were obtained. This results in a mean absolute error of 0.44 and a mean squared error of 0.36 between the actual and predicted values.

4.2 Basic approach

The first attempt to improve the current model is by using k -nearest neighbours. The workbench was given a script that modified the dataset (normalize all features and output), and to search for the best settings. These were determined to be at $k = 6$ with the nearest neighbours being weighted by their distance to the given example and then summed. The mean squared error for these settings was 0.04 and on the test set 0.29. With these settings fixed, the test was repeated with the complete dataset. The error dropped to the score given in table 1.

Error measure	Average	Min	Max	Error measure	Average	Min	Max
MSE train set	0.03	0.03	0.03	MSE train set	0.29	0.29	0.29
MSE test set	0.23	0.22	0.24	MSE test set	0.30	0.29	0.31
MAE train set	0.11	0.11	0.11	MAE train set	0.39	0.38	0.39
MAE test set	0.32	0.31	0.32	MAE test set	0.39	0.39	0.40

Table 1: Error measures for k -nearest neighbours on scaled dataset

Table 2: Error measures for Lasso with polynomial kernel on scaled dataset

The next tests are performed using Lasso. They are based on the same script in the workbench, with the k -nearest neighbour module being replaced by the Lasso module. Two kernels were tried: the RBF and the polynomial kernel.

The RBF kernel was tried for different values of kappa and sigma. In the Lasso algorithm, kappa is the upper bound for the L1 norm of the active set. For

different values for kappa, there were always some values of sigma that performed best. The differences between these pairs however are negligible. For both train and test set, the MSE varied around 0.31.

The polynomial kernel was tried for different degrees, and different values for gamma and the coefficient, in addition to kappa. The polynomial of the fourth degree was the most effective. For parameters kappa=0.7, gamma=0.3 and coefficient=3.0 the best results were obtained (table 2).

When the amount of examples that is allowed in the dataset is increased, these values do not change.

4.3 Modified wind directions

The wind directions varied over 0 to 360 (before scaling them around zero). The jump from 360 to 0 is quite significant, whereas in reality there is no distinction. To clear this problem, the directions were all mapped onto a circle. That way, the values become better comparable. The original wind directions are then removed from the dataset, and sine and cosine values of the mappings are added.

To measure the results, the polynomial kernel was used, again with degree 4. The initial results (with all settings kept the same) improved over the previous test.

Error measure	Average	Min	Max	Error measure	Average	Min	Max
MSE train set	0.26	0.25	0.28	MSE train set	0.25	0.24	0.26
MSE test set	0.27	0.27	0.27	MSE test set	0.29	0.26	0.31
MAE train set	0.37	0.36	0.38	MAE train set	0.36	0.36	0.37
MAE test set	0.37	0.37	0.38	MAE test set	0.39	0.37	0.39

Table 3: Error measures for Lasso with polynomial kernel on dataset with modified wind directions

Table 4: Error measures for Lasso with polynomial kernel on dataset with modified wind directions and a lag of one time step

When the number of examples that was used was increased to 70% the numbers did not change. By varying kappa, gamma and coefficient, it was found that the parameters were still optimal.

4.4 Adding lags

Because of the discretized wind speed values, we assume that better predictions are obtained by incorporating lags of previous wind speeds. The script was modified so that after the mapping of directions onto circles all input features were lagged for periods 1, 2 and 5 time steps. For lag=5, this resulted in a dataset consisting of 90 input features (for each of the five locations, the wind speed and the sine and cosine of the direction are available, and these values are repeated for t-1, t-2, t-3, t-4 and t-5).

The tests were performed with the same settings as for the previous test. The results are summarized in tables 4, 5 and 6.

Error measure	Average	Min	Max	Error measure	Average	Min	Max
MSE train set	0.26	0.26	0.27	MSE train set	0.25	0.24	0.26
MSE test set	0.31	0.29	0.32	MSE test set	0.30	0.28	0.32
MAE train set	0.37	0.36	0.37	MAE train set	0.36	0.36	0.37
MAE test set	0.39	0.38	0.40	MAE test set	0.39	0.38	0.40

Table 5: Error measures for Lasso with polynomial kernel on dataset with modified wind directions and a lag of two time steps

Table 6: Error measures for Lasso with polynomial kernel on dataset with modified wind directions and a lag of five time steps

Because of the good initial results with k -nearest neighbours and the varying results with Lasso, an additional test was performed with the kNN module, with lag=5 on 30% of the examples. This resulted in the measures in table 7.

Error measure	Average	Min	Max	Error measure	Average	Min	Max
MSE train set	0.05	0.04	0.05	MSE train set	0.03	0.03	0.04
MSE test set	0.24	0.23	0.24	MSE test set	0.20	0.19	0.20
MAE train set	0.14	0.14	0.14	MAE train set	0.12	0.12	0.12
MAE test set	0.34	0.33	0.34	MAE test set	0.29	0.29	0.30

Table 7: Error measures for kNN on dataset with modified wind directions and a lag of five time steps

Table 8: Error measures for kNN on dataset with modified wind directions and a lag of one time step on the complete dataset

As results for kNN improved as more examples were used for training, the test is repeated with all examples, and a lag of one time step. These results are summarized in table 8.

5 Conclusion

Two techniques were applied to the NUON wind speed dataset. These techniques were k -nearest neighbours and Lasso. Successive improvements were made to the dataset to obtain better results.

The k -nearest neighbour technique was able to obtain very good results. This is probably due to the huge amount of examples. Good examples are always near and the pattern is obviously stable.

Lasso is a sophisticated technique. From the comparison with the kNN technique, it seems that modelling the results of the predictions on the energy output

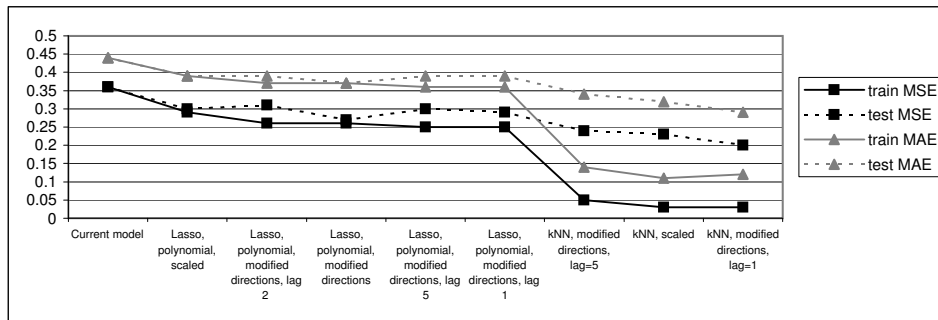


Figure 3: Results of the tests

is hard. The advantage is that the performance is more stable than that of the kNN technique.

Conversion of the wind directions to a more natural format improved the predictions. Results for adding lags vary. Apparently the addition adds less information to the data than the increase in complexity is able to justify. Adding a small amount of lag improves the results (minimum average MSE and MAE drop slightly), but the average errors on the test set increase.

The use of Lasso with a fourth degree polynomial kernel results in average errors of 9-10%. By using k -nearest neighbours with distance based weighing of the 6 nearest neighbours the average error is expected to be in the range 3-8%.

References

- [1] H.G. Beyer, D. Heinemann, H. Mellinghoff, K. Monnich, and H. Waldl. Forecast of regional power output of wind turbines, 1999.
- [2] D.G.T. Denison, P. Dellaportas, and B.K. Mallick. Wind speed prediction in a complex terrain, 2001.
- [3] S. Li, D.C. Wunsch, E.A. O’Hair, and M.G. Giesselmann. Using neural networks to estimate wind turbine power generation energy conversion. *IEEE Transaction on energy conversion*, 16(3):276–282, September 2001.
- [4] H. A. Nielsen and T. S. Nielsen. Using meteorological forecasts in on-line predictions of wind power. chapter 5: Estimation methods, 1999. ISBN: 87-90707-18-4.
- [5] F.A. Ouwendijk. Workbench for machine learning techniques. Master’s thesis, Delft University of Technology, June 2003.
- [6] V. Roth. Sparse kernel regressors. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Proceedings of the International Conference Vienna (ICANN’01)*, volume 2130 of *Lecture Notes in Computer Science*, pages 339–346. Springer-Verlag, 2001.