Analysing Machine Usage in Appleton Tower

Alexander Shearn

Fourth Year Project Report

Artificial Intelligence

School of Informatics

University of Edinburgh

2012

Abstract

The inspiration for this project came from personal experience of trying to schedule computer lab work only to find that on arrival at Appleton Tower, no machines were available. The objective of this project therefore was to build a web based application that allowed students to ascertain the availability of machines in Appleton Tower. A good one-line statement of the project would be:

> *"[to] Design and build a web interface to allow users to find and locate free machines in Appleton Tower, and to estimate when occupied machines may next be available based on historical data."*

Estimation relies on a range of different Artificial Intelligence methods, culminating in the use of Genetic Algorithms to train a weighted sum estimator. Use of Neural Networks is discussed and set aside due to architecture constraints and other system requirements. The performance of the different methods is analysed, showing that the weighted sum performs the best of the various methods implemented, meeting the accuracy requirements outlined in the project specification. Extensions to the project are also discussed, which include the collection of additional information about the machines and the possibility of packaging the software for deployment as a product to other networks. Further work is proposed that could make use of the system API to build a native smartphone application or improve the estimation.

The project is successfully in use and is 'live' at `http://project.shearn89.com`.

Acknowledgements

The project could not have been completed without the help and support of the following people: my supervisor Ewan Klein for keeping me on track throughout the year, and my flatmates for putting up with me talking about algorithms for months.

Thanks also go to Chris 'xoebus' Brown for building the report template.

The report style used is a modification of the `cs4rep` style used in the computer science department until 1998-9.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Alexander Shearn*)

Table of Contents

Chapter 1

Introduction

This project was to build a web-based application (or *'webapp'*) that could enable users to see at a glance how many available machines there were in the Appleton Tower computer laboratories, and to predict how many machines will be in use in the next 15 minutes. This is achieved by collecting data from the machines and using a range of machine learning methods and basic logic to estimate future usage. A sample of the landing page for the application may be seen in Figure 1.1.

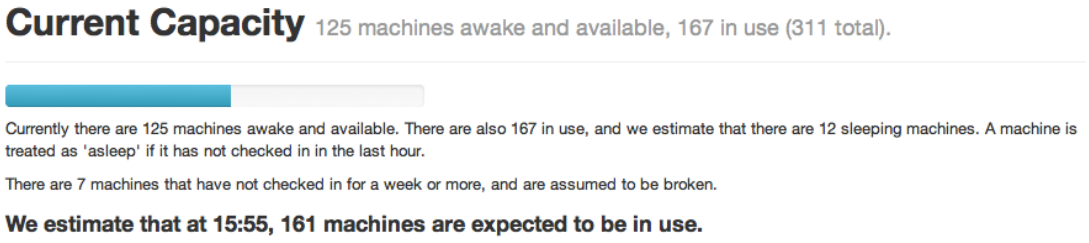


Figure 1.1: Part of the landing page for the site.

See Appendix A.2 for a full screenshot of the main page.

In the summer at the end of my second year at university, I attempted to create a simple client/server application that would allow friends to see who was working in the laboratories, and where they were.[1] This would have helped to foster a more social working environment, as groups of friends could meet in the same lab to discuss problems that they had encountered or ideas that they had. Unfortunately, this simple application relied on a server being 'always-on' for the clients to connect to, and finding suitable hosting that could provide the necessary open TCP socket was a problem that was never overcome.

[1] See `https://github.com/shearn89/inthelabs`. **Note:** currently in a very broken state.

The ability to be able to see which machines were in use at any one time would be something that would ease congestion during busy periods of the semester. Even the simple act of visualizing the machine usage over the last 24 hours would help users to find quieter times in which to work without struggling to locate a free machine. This project solves this problem, providing users with a simple and easy to use method to see how many machines are currently available, as well as where they are located. As a webapp it can be reached from mobile devices as well as desktop computers, making it easy to use whilst en route to the campus.

The project was inspired by two things: the simple application mentioned previously, and the work ethos of many students. Students are disinclined to speculatively visit the university campus, especially in inclement weather, only to discover that there are no free computers or that the only available machines do not meet their needs. This can be especially true during busy periods of the semester with multiple coursework deadlines, or in the days preceding group coursework deadlines such as for the System Design Project undertaken in third year. The ability to be able to check which machines were available before setting out would overcome this problem, as one would be able to assess the level of utilization of the laboratories and the incipient level of utilization in 15 minutes time. This would allow the user to be able to make a decision as to which floor to search for a free machine, or whether to reschedule the work.

Another use case could be for users already on campus. Instead of going to a particular floor in Appleton Tower, then searching room by room for a free computer, they could simply navigate to the website for this project and see that there are x machines available in lab *3.09*. The student could then head straight to the specific laboratory, without wasting time searching across all 3 floors to find a machine.

Finally, the system could be of benefit to resource managers such as the Computing Support department, allowing them to find and fix broken machines without waiting for them to be reported as such by students. It could also be used by them to analyze which machines are used most frequently and therefore which are more likely to break down.

1.1 Related Research

This project could be applied to similar situations in other areas, of which a good example would be power grid forecasting. Forecasting over short time frames (from single minutes to several dozen minutes) is referred to as very short-term load fore-

casting [Qingle and Min, 2010], which is essentially what this project hopes to undertake. As the aforementioned paper says, the methods used to perform short term load forecasting can be broadly split into three categories: parametric, non-parametric, and artificial intelligence methods. The parametric methods rely on formulating a relationship between the system load and the factors that affect the load. The non-parametric methods allow the future load to be calculated directly from the historical data. Artificial Intelligence methods would generally use a Neural Network combined with other estimation methods. This project focusses on the non-parametric and Artificial Intelligence methods, using simple non-parametric methods and then combining them into a weighted sum, trained with a genetic algorithm.[2]

However, when predicting over a very short window, the factors that might normally be used to make a prediction may not be as applicable, and so predictions must be made based on local patterns and observed motifs.[3] Recent research uses a neural network combined with rough set theory to create an accurate predictor [Qingle and Min, 2010]. This is similar other literature on the topic, such as in [Dai and Wang, 2007] and [Osman et al., 2009] which would use a neural network to predict usage; however other methods used include discretizing the data into ranges [Lin et al., 2007] and creating a state space model [Ma et al., 2011].

A neural network could be a good approach in this instance, however thought must be given as to how well it would fit the requirements of the project and what inputs would be used. This is discussed in section 2.1.3.

[2]For more information on genetic algorithms, see `http://en.wikipedia.org/wiki/Genetic_algorithm` and [Mitchell, 1998].

[3]See [Qingle and Min, 2010], Introduction, paragraph 4.

Chapter 2

Project Design

The requirements of the project placed some constraints on the architecture used. It was necessary to collect data from around 300 machines on an internal network, which then needed to be processed and analysed before being presented on a web server that could be accessed from anywhere. This structure limited the choice of hosting services available, which will be discussed in the next sections.

2.1 Requirements

The project had to perform three essential tasks which will be discussed in detail.

1. Be accessible from outside the School of Informatics network.
2. Show which machines were currently available.
3. Estimate (to a reasonable degree) the future availability.

2.1.1 External Availability

In order to be of practical use, the system must be available outside of the School of Informatics network, without use of remote login technologies such as VPN or SSH.[1] This would enable users to be able to check the usage of a particular machine or set of machines before travelling in to the university, and also make sure that if someone was using a mobile device they could still obtain the information with as little effort as possible.

[1] See `http://en.wikipedia.org/wiki/Virtual_private_network` and `http://en.wikipedia.org/wiki/Secure_Shell`.

This requirement placed some limits on the type of mechanisms that could be used to present the data. If the system was designed for a mobile platform as an OS-specific 'app' (e.g. for iOS or Android), then more issues would arise: the application would have to be listed on the relevant 'app store', and would have to conform to the standards and restrictions of the managing body behind that store. The project needed to be visible from outside the DICE network, but it need not be as openly advertised as it would be by placing it on an App Store. However, if the application was therefore not a mobile one but a desktop application, the availability would be greatly reduced as the user would require a laptop to be able to access it whilst on the go.

The solution therfore was to create a web-based application that presented the data in a format that was accessible to both desktop and mobile browsers, and worked even when the user was out and about. For this purpose, the data was required to be presented in such a way as to ensure that there are no privacy or security issues (such as displaying which users are logged on).

2.1.2 Available Machines

The system was required to show which machines in which laboratories are currently available. This meant that the data had to be collected by either passive or active means: that is, either a central location requested the data from the machines, or the machines updated some central data store. Another restriction was therefore placed upon the system: all the machines on the DICE network must be able to communicate with the central data store.

If either server (web server or data server) was outside of the DICE network then the other machine would require a secure method of communicating with it, therefore it would be simplest if both machines were inside the DICE network.

These requirements were further affected by the fact that data was already being collected by Computing Support. Each machine on the DICE network appended a single line to a text file every 15 minutes, and these text files were aggregated and stored in a central location once a week. A decision was made to try to take advantage of the existing infrastructure for data collection, which is discussed further in section 2.3.

2.1.3 Estimation of Availability

If the system is to be able to estimate the future usage, the algorithm for doing so must either be fast enough to run efficiently on a web server, or the machine doing the computation must be able to communicate with the web server. This is yet another restriction that must be adhered to. The time frame for estimation must also be considered: should the system try to predict the usage at specific times, or merely in the next X minutes?

The time frame in which usage is estimated also raises issues of scalability: depending on the prediction method, there would be both an upper limit on the number of machines used to estimate the usage level, and a lower limit on the time steps. If enough machines were added to the network, it would take more time to estimate the next value than it would to actually reach that value, and thus the system breaks down.

As mentioned in the Introduction, a neural network could be a good approach to solving this problem, as it is able to be implemented in such a way that the system becomes more accurate the longer it runs. The problem was going to be building the system in such a way that it remained fast and responsive.

2.2 Specification

Bringing together the requirements above, the following specification was decided upon.

- The web application should be available as simply as possible.
- If authentication is to be used, it should be easy to authenticate on a mobile device.
- If no authentication is to be used, then the data should respect privacy and security requirements and standards.
- A web framework such as Django, Ruby on Rails, Node.js, should be used for the web application, allowing greater functionality than a simple static website.
- The data should be stored in a central location in order to simplify maintenance and configuration.
- Communication to and from this central location should be secure, preventing alteration of the data by third parties.

- The methods or algorithms used to estimate future usage should be quick enough that they do not adversely affect the performance of the website. Slow loading of the page because the server is trying to perform a huge calculation is not acceptable.

2.3 Architecture

Ruby on Rails was used for the web framework for a number of reasons: the author's personal interest in Ruby; the quality of support and documentation available; and the original client/server application referred to in the Introduction to this project being written in Ruby.

The web server would keep a local database with the most recent values, which would make sure that the website was responsive and conformed to the performance specifications decided upon.

At first the existing simple method of data collection was used with minimal alteration. This method relied on each machine running a `cron` task that appended a line to a text file, containing a time and a boolean value indicating availability. A shared folder on the AFS filesystem[2] was created to contain these files and instead of being copied to a central location once a week, they were simply created in the AFS share, allowing access from any machine that had been set up with AFS.

However, in order for all the machines to be able to write to this shared folder, it had to be globally writable. This raised security concerns and so another method of data collection was decided upon. The system was altered to use a PostgreSQL database hosted by the School of Informatics, and a script (Figure 2.1) was written to be run as a `cronjob` on each machine that inserted a new record in the database, using the following PostgreSQL schema:

```
1       host    | character varying(32)       |
2       usage   | integer                     |
3       logtime | timestamp without time zone | default now()
```

The web server runs a script every 15 minutes that updates its local database with the most recent values from the PostgreSQL database. When using the text file method for data storage, each file in the folder had to be opened, the last line parsed, and

[2]The AFS filesystem is used for storing of users files on the School of Informatics internal network. It caches files locally for speed. More information is available on Wikipedia: `http://en.wikipedia.org/wiki/Andrew_File_System`.

the database updated. This made the update method very slow, especially since the files were stored on a network filesystem (albeit one with good cache management). By using a relational database instead of a filesystem for data collection, the system gained a huge boost in speed, as well as addressing the security concerns with having a globally-writable folder. This script first gets the name of the machine on which it is

```
1 #!/bin/bash
2 name=`uname -n`
3 inuse=`who | grep -c ':0 '`
4 qry="INSERT INTO usage (host, usage) VALUES ('$name',$inuse);"
5 export PGPASSFILE='/public/homepages/s0700157/data/.pgpass'
6 /usr/bin/psql -h pgteach << eof
7 $qry
8 eof
```

Figure 2.1: The cron task run on each machine.

running (line two), then checks to see if a user is logged on by listing all users logged in, and seeing if any are logged in locally instead of remotely (line three). The script then inserts these values into a string, which is sent as a query to the remote PostgreSQL database. The sixth line specifies that the database is hosted on a dedicated server by the School of Informatics, which has the name `pgteach`.

A diagram of the system architecture can be seen in Figure 2.2. This shows the flow of information, from the machines themselves all the way to the client on a web browser. The diagram also shows the connection between the Rails system and the PostgreSQL database, which is queried when estimation occurs in order to retrieve data for the last 24 hours.

As well as the update script which runs every 15 minutes, the web server runs a cronjob every 8 hours that renews a Kerberos ticket, allowing it to stay in contact with the PostgreSQL database. When the data collection was first moved to the database this important step was overlooked, so although the data was being collected it was not being retrieved by the web server. By using Kerberos the data is protected from unauthorised access, and as both the web server and the database server are inside the DICE network they are protected by the institutional firewall.

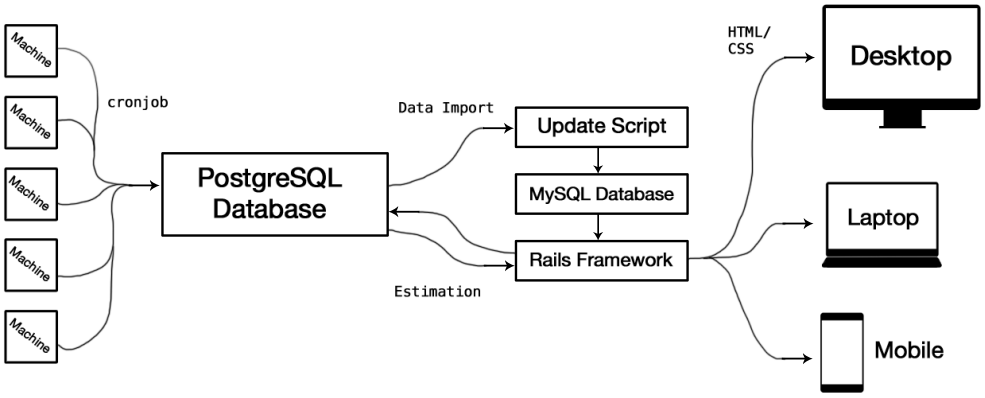


Figure 2.2: A diagram of how the various components fit together.

Chapter 3

Approach

The flow of data is as follows: every 15 minutes each machine creates a record in the PostgreSQL database on the School of Informatics' `pgteach` server. Shortly afterwards, the web server runs a script that gets the latest entries from this database and updates its local copy of the data, using simple logic to tell if a machine is available, offline, or stale/dead. This is then served to the webpages.

Caching is used for the controller methods that require querying of the PostgreSQL database, which are the creation of the sparkline graph visible on the web page (see Figure 3.1), and the estimation methods. This is to prevent Denial-of-Service attacks by 'spamming' requests for the graph or the estimation data. Each method is cached for 15 minutes as that is also the time between updates, and the script that updates the web server's local copy of the data clears the cache, ensuring that served data is not out-of-date.

This project combines a number of popular web development technologies. Version control is managed with `git`[1], the server itself is running `nginx`[2] and `mysql`[3] and the app is built using Ruby on Rails following the Model-View-Controller design pattern. The Capistrano gem[4] is used for deployment over ssh.

3.1 Estimation

In order to be able to give an indication of whether usage is rising or falling estimation of usage in the next 15 minutes must be accurate to a reasonable degree. If this

[1] `http://git-scm.com/`
[2] `http://wiki.nginx.org/Main`
[3] `http://www.mysql.com/`
[4] `https://github.com/capistrano/capistrano`

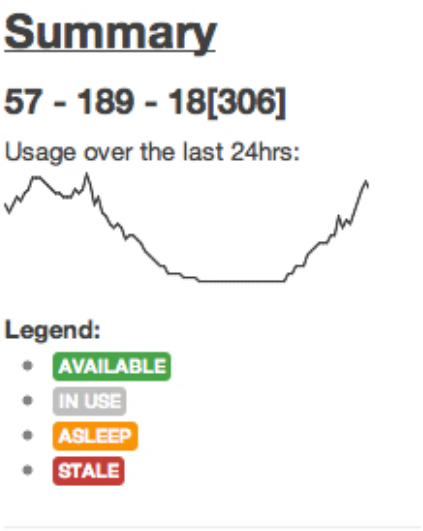


Figure 3.1: The sidebar, complete with sparkline image.

estimation is suitably accurate then users would know how quickly usage is changing. The estimation methods currently implemented are of three types and are detailed in the following sections. An evaluation of each method is presented in section 4.2.

3.1.1 Linear Interpolation

The first is the simple average of two points, as in Eq. (3.1): the most recent count of machines in use, and the point from 24 hours previously. Estimating the usage by this method relies on the pattern of use from day to day being very similar.

$$f(t) = \frac{(x_t - x_y)}{2} \tag{3.1}$$

Eq. (3.1) shows the calculation performed by this estimator. x_t is the current usage at time t, x_y is the usage from 23 hours 45 minutes previously. That is, the value from yesterday that corresponds to the incipient usage we are trying to predict.

As the first method implemented it seemed to be very simple mathematically, however the SQL required to get the necessary records from the database was surprisingly complicated. As requests to the database are the slowest part of the system it is desirable for the necessary values for the calculation to be obtained in a single SQL query.

The query uses aggregate functions to first find the minimum logtime for each host in a certain period, and then create a table of *"host, usage, logtime"*. Then another sub-table is created by finding the maximum logtime in the same way, and finally a natural join is performed on the two tables, giving a final result of: *"host, min_value,*

min_logtime, max_value, max_logtime". The SQL for the query is shown in Figure 3.2, using PostgreSQL's aggregate functions *min* and *max*. The data retrieved by this

```
SELECT *
FROM (SELECT u.host,u.usage AS min_val,r2.min
  FROM usage u
  JOIN (SELECT host,min(logtime) FROM usage GROUP BY host) r2
  ON u.host = r2.host AND u.logtime = r2.min) min_table
NATURAL JOIN (SELECT u.host,u.usage AS max_val,r1.max
  FROM usage u
  JOIN (SELECT host,max(logtime) FROM usage GROUP BY host) r1
  ON u.host = r1.host AND u.logtime = r1.max) max_table;
```

Figure 3.2: SQL query for 2-point linear interpolation.

query is required because it provides the means to calculate the two values needed in Eq. (3.1). Instead of retrieving data for the last 24 hours and then ignoring most of it, the query retrieves just the data for the current time and the time from yesterday. The entries are then aggregated based on time, providing the two numbers x_t and x_y needed by Eq. (3.1).

This uses a total of three JOINs which was overly complicated for what was imagined to be such a simple query, so after posting a question on the well-known StackOverflow help website [Brandstetter, 2011], this query was refactored using some PostgreSQL WINDOW functions into the query in Figure 3.3. **Note:** this version of the query also includes the WHERE clause that restricts logtimes to the last 24 hours.

3.1.2 Trigonometric Curve Fitting

The next method originally seemed more complicated as it required the estimation of various mathematical parameters, but turned out to be comparatively simple. By taking the last 24 hours and fitting a sine function to the curve, the next value is then estimated by extrapolating the function and calculating the value. This was partially inspired by [Eubank and Speckman, 1990], and by the shape of the historical data (Figure 3.4, repeated later as Figure 4.7).

The diagram in Figure 3.5 clarifies how the parameters for the sin function are found. First the mean value is found, which provides the displacement from the x-axis. The initial gradient and value is used to find the displacement from the y-axis. The amplitude is estimated as the difference between the average value and the minimum

```
SELECT DISTINCT
  u.host,
  first_value(usage) OVER w AS start_val,
  last_value(usage) OVER w AS end_val,
  first_value(logtime) OVER w AS start_time,
  last_value(logtime) OVER w AS end_time
FROM (
  SELECT * FROM usage WHERE
  logtime > (now() - INTERVAL '1 day')
  AND
  logtime < now()) u
WINDOW w AS (PARTITION BY host ORDER BY logtime, usage
  ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
ORDER BY host;
```

Figure 3.3: Final SQL query for 2-point linear interpolation.

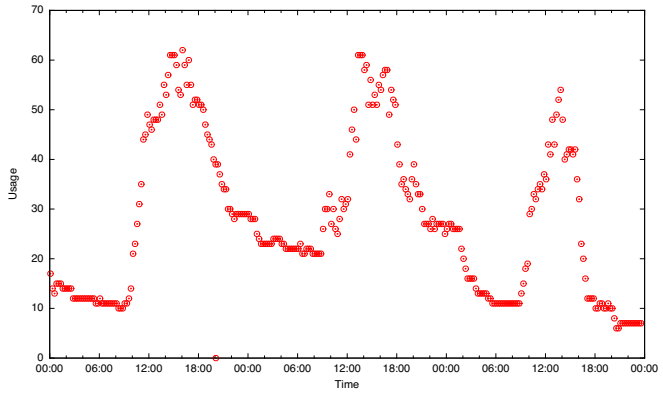


Figure 3.4: Plot of usage against time for a few days of historical data

or maximum value over the 24 hour period, depending on the shape of the curve. The period is more complicated: first all times at which the usage is the same as the initial value is found, which in the diagram is points **A**, **B** and **C**. The gradient is found at all these points to check in which direction the curve is travelling. The point where both the gradient and the value match our initial point is used to calculate the period (in this case, the period is from **A** to **C**).

Fitting a sine curve seemed at first to be a good idea, but was soon seen to be extremely variable in terms of accuracy. The method relied on the data staying very close to the sine function, and as soon as the usage changed dramatically or was slightly unusual, this method produced wildly inaccurate predictions. This is easily shown by

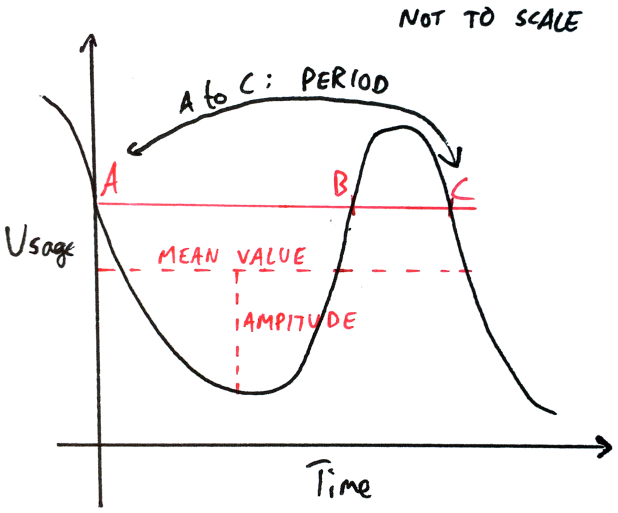


Figure 3.5: Diagram of the sin estimation.

looking at the histogram shown in Figure 3.6, which is discussed further in Section 4.2.2. This histogram is plotted to the same y-axis scale as the others, showing at a glance the difference in performance between the methods. The inaccuracy of the sine function is clearly visible by the lack of any peak at all, as well as the huge spread of the histogram.

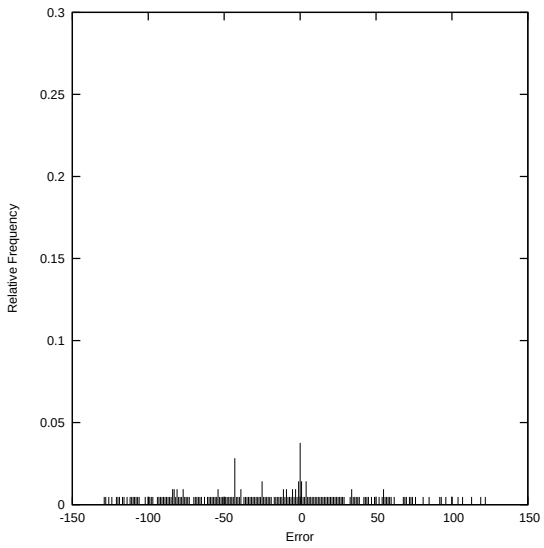


Figure 3.6: Relative frequency of the trigonometric prediction error.

3.1.3 Average Gradient

A simple but effective prediction method that was implemented was to find the average gradient over varying numbers of most recent values,[5] relying on local trends to produce good estimation. This method was basically an extension of the linear interpolation method, but relying on the local usage patterns rather than the daily usage patterns. Each time step is treated as one point on the x-axis, so a gap of 30 minutes would be two steps. The caclulation used to calculate this is shown in Eq. (3.2), and in a more concise form in Eq. (3.3).

$$f(t) = \frac{(x_t - x_{t-1}) + (x_{t-1} - x_{t-2})}{2} \tag{3.2}$$

$$f(t) = \frac{\sum_{i=0}^{n-1} (x_{t-i} - x_{t-(i+1)})}{n} \tag{3.3}$$

This method takes the **n** most recent values, calculates the difference between them, and averages it. This is then used to estimate the change between the current timestep and the value we are trying to predict.

The prediction became less accurate during periods of high usage, as the number of machines in use could increase or decrease dramatically in a very short space of time. The histogram for one of these predictors can be seen in Figure 3.7.

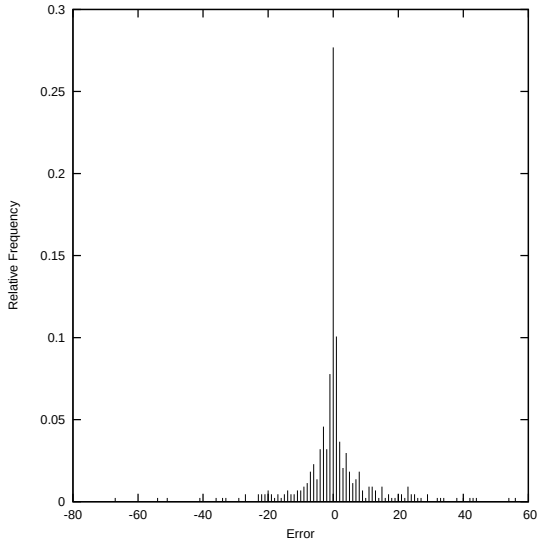


Figure 3.7: Relative frequency of the prediction error of the average gradient method, for the 2 most recent gradients.

[5] In this case: 2, 4, 8 and 16 most recent values were used.

3.1.4 Weighted Sum

Eventually the previous prediction methods used were analysed and improved upon, resulting in the final estimator which was a weighted sum of various values, based partly on methods mentioned in [Williams, 2010]. The weighted sum was a combination of two of the different gradient estimators for 2 steps and 4 steps, along with the value 24 hours previously, and the value from one week ago. By using this range of 'inputs' different levels of structure are taken advantage of: close local structure by using the average gradients, as well as daily and weekly structure by using historical values.

The weight vector was found by first generating random weightings and predicting values based on the historical data gathered over a period of three years. A random time in the available range was chosen and the next value was predicted and then compared to the actual value. Then, the average error was calculated and the vectors were ranked. The best performing vectors were chosen and used to predict a larger number of values, and the process was repeated. However, this process did not follow the formal structure of a genetic algorithm:[6] rather, the experiment was run a number of times and the vectors generated were evaluated by hand. The investment in time this required was considered worthwhile.

3.1.4.1 Genetic Algorithm Trained Weighted Sum

As much of the research in the area of load forecasting uses Neural Networks to predict values (See [Osman et al., 2009], [Dai and Wang, 2007], [Liao and Tsao, 2006], [Charytoniuk and Chen, 2000]), it was felt that some effort should be made to use a similar technique to improve prediction. A genetic algorithm was implemented in order to improve the weight vector in a more formal manner than that used previously. For this genetic algorithm, 100 random vectors of length 4 were initialised with floating point weights between 0 and 1. Then, 250 random time points were retrieved from the historical data training set. Each vector was used to predict the next value, and the fitness function was defined as 1 divided by the mean squared error over the whole set of 250 time points (Eq. (3.4), Eq. (3.5)). The fitness was then maximised

[6] This structure is described in detail in [Mitchell, 1998], and also on Wikipedia at `http://en.wikipedia.org/wiki/Genetic_algorithm`.

during the genetic algorithm.

$$mse = \sum_{t=0}^{n} \frac{(x_t - f(t))^2}{N} \tag{3.4}$$

$$fitness = \frac{1}{mse} \tag{3.5}$$

Then the usual genetic algorithm process of selection, reproduction and mutation was run on the population with the probabilities in Table 3.1. The selection percentage indicates the percentage of the population that is passed directly through to next generation, based on fitness. A total of 10,000 iterations was used to generate sufficiently 'fit' weight vectors. These 100 weight vectors were then further analysed by comparing the mean squared error over 25,000 historical time points, and the best performing vector was kept to compare to the vector that was then in use. The top five vectors from this process are shown in Table 3.2, ranked by cost, which was the mean squared error. This best performing vector was then compared against 10,000 time points from the historical training set. All the previous estimation methods were also tested against this data, resulting in the statistics shown in Table 3.3 (repeated in Section 4.2.6).

Probability of Mutation	Probability of Crossover	Selection Percentage
0.1	0.7	25%

Table 3.1

Cost	Vector
109.4719	[0.513, 0.475, 0.8216, 0.0615]
110.0131	[0.2368, 0.7696, 0.757, 0.0245]
110.1438	[0.1345, 0.8488, 0.7256, 0.1348]
112.1747	[0.649, 0.3645, 0.5924, 0.2672]
113.4133	[0.3828, 0.6159, 0.5381, 0.1907]

Table 3.2: The top 5 weight vectors from the Genetic Algorithm, ranked by cost function.

3.2 Data Collection and Migration

As mentioned in Section 2.1.2, originally the script that updated the application's database scraped a huge collection of text files that were stored on the AFS filesys-

Estimation Method	Mean Error	MSE	Std. Dev.
Hand-trained vector	3.277	171.978	12.698
Genetically trained vector	1.374	107.385	10.271
Linear interpolation	15.667	512.764	16.349
Average gradient (2 steps)	1.918	215.424	14.551
Average gradient (4 steps)	1.932	192.999	13.757
Sine curve estimation	-5.478	1076.503	32.35

Table 3.3: Comparison of estimation methods using 10000 training set instances.

tem used by the School of Informatics.[7] Each text file was made up of lines of the form "`2011-12-01H15:10 1`", which is a timestamp and a boolean value to tell if the machine was in use at that time. The use of text files may have been simple, but it had some major flaws. Firstly, the `afs` share that the files were stored in had to be world-writable in order for it to work for all the machines. Although acceptable for a short time period, it was undesirable to have such an open file space on the DICE network. The huge number of disk read/write operations required to scrape these files also made the system slow to update itself.

Midway through the project this system was scrapped and migrated to a PostgreSQL database for the data collection. This meant that the update script was much, much faster to run, and disk operations were limited to the retrieval of the records in the database on the `pgteach` server. A problem has been that occasionally there are too many machines accessing the database server at the same time, and so a very small amount of data is lost. This was almost completely overcome by adding a short random sleep delay to the script run by the cronjob, which ensures that there are fewer near simultaneous requests to the server.

There are still infrequent access problems, but these could be because the cronjob script is stored on a network filesystem, and all the tasks run at the same time, even if they send data to the server after various small delays. This could be fixed if the script was copied to every single machine, but then the file containing the database password information would also have to be copied, making it less secure. In general these faults do not occur every day, and never to more than 2 or 3 machines during that day, for 3 or 4 updates. Given that there are 86 updates by each of around 300 machines a day,

[7] See Footnote 2 on page 7 for more information on the AFS filesystem.

these are virtually insignificant errors.

3.3 Status Logic

The logic that determines when a machine is asleep or 'dead' is currently very simple and could use some improvement. If a machine has not 'checked in' for over an hour, it is assumed to be sleeping. If it has not checked in for over a week it is assumed there is a problem with it and it is marked as 'stale'. This method has the flaw that machines that are very infrequently used can occasionally be flagged as broken when they are not. Similarly if there are network problems and a machine has failed to check in for an hour, then it could be marked as asleep when it is actually awake.

The manner in which these statistics are reported has required considerable thought. Should machines that are asleep be considered to be available? Given that it would be rare for someone to let the machine sleep and yet still need it, it could be safe to report the number of 'available' machines to be the sum of those that are awake and marked available with the number of sleeping machines. However there is the case where a user may be working on a laptop or pen and paper whilst also using the machine for experiments or other tasks, and has allowed it to sleep while they continue work on something else. There is also the problem that if a machine has been marked as asleep for six days (one day short of being marked as 'dead'), it could well be broken and therefore not available for use even though it would still be flagged as asleep.

This problem of when to classify machines as being in a particular state is one that could be endlessly optimized, and which only received a small amount of attention during this project. Experiments could be done to attempt to find better thresholds, but they would depend greatly on the machine being analyzed. Some machines will always be used more than others: one solution to the problem could be to give each machine it's own threshold values, but this greatly increases the complexity of the system.

The real function of the system should also be kept in mind: if a user has already checked the site and discovered that there are 5 machines available in a particular lab, they would have no need to look at the asleep machines. The sleeping machines need only be considered if the user must work in a particular lab, or if the extreme case where all machines are in use.

Chapter 4

Evaluation

This chapter focusses on analysis and critique of the end-user interface through user feedback, and the performance analysis of the different estimation methods used.

The webapp has had Google Analytics running on it since it first went 'live'. Usage has steadily grown, with daily visits indicating a small userbase over recent weeks (see Figure 4.1.)

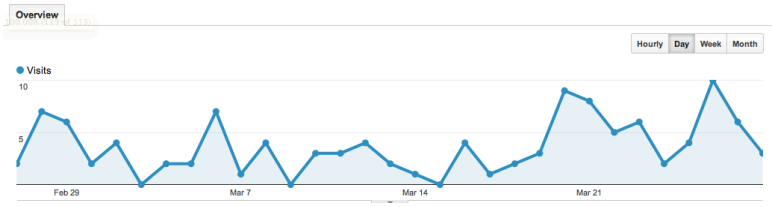


Figure 4.1: Google Analytics graph of daily site visits.

There was a very large spike in visits on the day when an email was sent to the Informatics Undergraduate student mailing list, indicating that around 200 people read the email and followed the link in order to see what the project was about. This is visible in Figure 4.2.

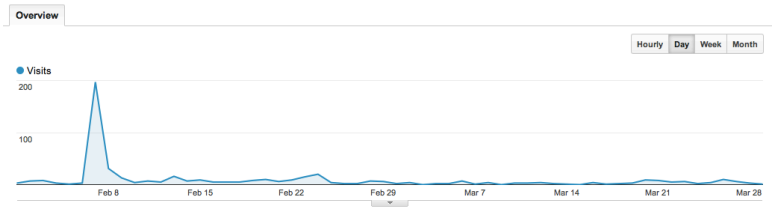


Figure 4.2: Google Analytics graph of the spike in site visits.

A flowchart of how visitors interact with the site can be seen in Appendix A.4, which shows that the majority of the visitors have looked at the main page, probably because that is where the summary information is available. If they wish to know how full the laboratories are in general, they would look at the landing page and then leave the site. If they wish to know information about specific machines or laboratories, they continue on to the 'available' page. The flowchart also shows that some people visit this page directly, indicating that they may have it bookmarked: they obviously wish to know the status of a particular machine they like to work on, or simply the first place to look for a free machine. This usage fits with the use cases outlined in chapter one.

4.1 User Evaluation

The application included a contact form in order to attempt to gather feedback from the users. The system was also advertised to students via email and through a number of presentations.

In general, the feedback was positive and included constructive criticism of the user interface, requests for additional functionality and similar comments. Some issues arose more than once. These included the following topics, discussed in the following subsections:

- Filtering machines and the 'Search' bar
- Visual Layout
- Definitions used and presentation of information
- Additional information

There are two examples of submitted feedback in Appendix A.5.

4.1.1 Filtering Machines

Users frequently commented on the placement and functionality of the search bar, even unknowingly. Originally it was placed at the top of the list of machines, under the informative text describing the website (see Appendix A.2). However this meant that the search bar was in a different position depending on which page was being viewed: if the user was on the main page the box was in one place; but on a page for a specific lab or machine then it was slightly different. In order to make sure that the search bar

was always in the same place it was moved to the toolbar along the top, placed in the upper right corner (Figure 4.3).

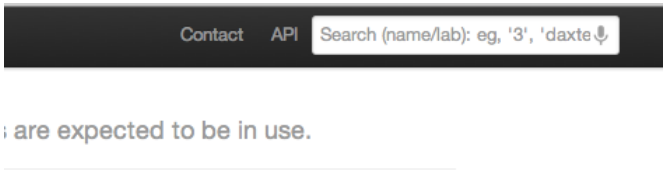


Figure 4.3: The final positioning of the search box.

However, this still presented problems: feedback was still provided that indicated that users were failing to find the search bar, or that they did not realise that it could be used to filter by laboratory. The toolbar 'sticks' to the top of the page no matter how much the user scrolls, so the problem was that users did not notice it, or did not realise the full functionality of the search bar. This could only be solved through user education: a help page could be added with informational text that describes in detail the functionality available throughout the site. Familiarity with the system would also help greatly, as users would learn where the bar was and how it behaved in relation to the current page.

4.1.2 Visual Layout

Multiple users requested some sort of visual aid as to where a specific machine was located. An example of this is that the West Lab on Level 5 of Appleton Tower has 51 machines in it. It is helpful to know that a particular machine is available in that lab, but finding it when you actually get there can be difficult. Figure 4.4 shows a photo of one half of the West Lab, demonstrating how difficult it may be to find a specific machine. This 'visual search' would also allow users to be able to find adjacent machines in order to facilitate social coding methods like pair programming. A simple solution to this problem would be to use Javascript to display an image as a popup, showing the layout of machines in the relevant laboratory. This could be extended to colour code specific machines based on availability. Time became a factor in deciding which work to pursue, and so this functionality was omitted.

Figure 4.4: Photograph of a part of the West Lab, Appleton Tower.

4.1.3 Definitions Used

Some feedback was provided that indicated users were uncertain what the definitions used on the site meant, or how particular information was presented. An example is the sidebar that is present on all pages, which shows summary information and a sparkline graph of the usage over the last 24 hours (shown in Figure 3.1, repeated here in Figure 4.5). Users reported that they did not know what the numbers meant, unless they were on the 'Information' page that the user lands on when first visiting the site. The sparkline also has no reference points on it, apart from the text that states that it is a plot of the last 24 hours: there is no indication of maximum or minimum values. The solutions to the problems with the sparkline are however restricted by the library used to generate the graph. In order for the graph to be compact, the library omits data such as axes and labels. This is part of the design of a 'sparkline' graph, however it does mean that the graph loses some relevance. Instead, the graph must be used simply as an indication of recent usage and as a means to make an intelligent guess with regards to whether the usage will increase or decrease in the near future. It should not be taken as a scientifically accurate data plot.

A user also reported that he was uncertain what the word 'stale' meant. As mentioned in Section 3.3, the word 'stale' is used to describe machines that have not updated the database for over a week. However this may not mean that they are broken, merely that they are in an infrequently visited lab, or even may have been unplugged

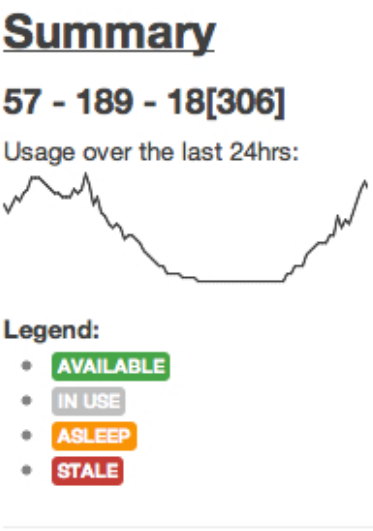


Figure 4.5: The sidebar, complete with sparkline image.

by someone wishing to charge a laptop. The word 'stale' could easily be replaced by some other definition, perhaps chosen by a user poll.

4.1.4 Additional Information

Much of the feedback from users focussed on additional features and information that could be presented, some of which raises other issues. The hardware-related information could be reported by adding columns to the database and altering the cron task script, and would be a simple matter to implement. Other information might involve considerably more work to add to the system.

Requests included information on:

CPU speed - One user used the project to run coursework experiments for a course on Compiler Optimisation [Pitidis, 2012]. By screen-scraping the website, he could find free machines, and run the experiment over ssh. He mentions:

> *"For example, some machines are i3 Quad Cores at 3.0GHz, while others are rated Core 2 Duo at 1.8GHz. It should be relatively straightforward to retrieve the specs of each machine and assign benchmarks to machines with adequate performance and no load – ideally such information should be provided in the original listing though."*

Kernel version - Similar to the last point about CPU speed, this may be useful to students who need to compare experiments based on linux kernel version, although it is unlikely that this varies very much on a managed network such as DICE.

Available RAM - This could be useful to students who are running experiments that require large quantities of memory, or a specific amount of memory.

Logged in User(s) - The ability to see which user was logged in would be hugely beneficial. It could be used by students to see if they have friends in a particular lab, or to see if there are members from a particular year group in a certain lab. This information would need to be secured though, as having usernames (or even real names) displayed on the internet for anyone to see would raise issues of privacy. The site could show one set of information to unauthenticated users and a different set to users authenticated via the DICE weblogin form.

Booking information - If the system were connected to the Informatics Room Booking System, machines could be flagged if the room they are in has been booked for specific events such as tutorials or scheduled lab sessions. This would also improve the estimation, as the number of 'available' machines could be adjusted to show machines available to everyone. As an example the West Lab is meant to be available only to first year students during the afternoons on weekdays: if a non-first year student is in there and a first year needs the machine, they are obliged to move. It would be helpful for users of this prediction system to be able to see this information, in order to avoid having to move labs during a work session.

4.2 System/Data Evaluation

Experiments were conducted on two different sets of data: a training set of historical data collected by Computing Support over a period of around 3 years, and a test set of 'live' data gathered while the system was operational.

4.2.1 Linear Interpolation

This method was the simplest implemented and used a very straight-forward equation to calculate the next value: the average of the most recent value and the value from 24 hours previously, as shown in Eq. (4.1). By using the value from the previous day, we allow for some exhibiting of a daily pattern in usage, instead of using an extrapolation method which would focus solely on the local pattern.

$$f(t) = \frac{(x_t - x_y)}{2} \tag{4.1}$$

This method was generally unreliable, and the estimation was consistently low when compared to the actual value. A normalized histogram of the error between the estimated value and the actual value can be seen in Figure 4.6.

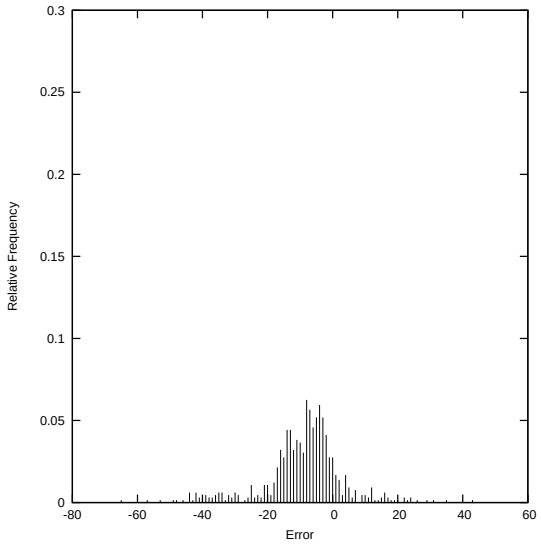


Figure 4.6: Relative frequency of the linear interpolation method error.

As shown in this histogram, the performance was not good. There is a small cluster of points centered slightly below 0: this emphasizes the point made about the estimation being consistently low. Ideally, we want an estimator that has a distribution centered on 0 with a low standard deviation.

4.2.2 Trigonometric Curve Fitting

The trigonometric method seemed to be a good fit based on historical data. In general, the usage would follow a clear trigonometric curve over a daily or weekly period (Figure 4.7).

In practice although the data generally did follow the shape of a sine curve, the local variation was too great to make the method effective. This is easily demonstrated by the shape of the normalized histogram shown in Figure 4.8: extremely wide, with almost no peak at the mean.

4.2.3 Average Gradient

The average gradient method proved a good estimator. The mean was close to 0 in training (Table 4.1 on page 28), although this did not remain true when using the test

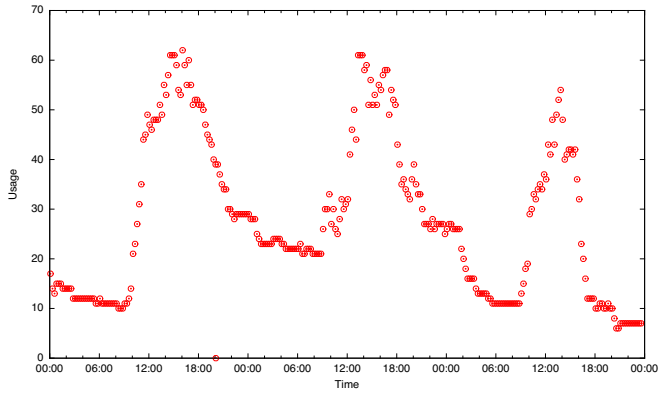


Figure 4.7: Plot of usage against time for a few days of historical data

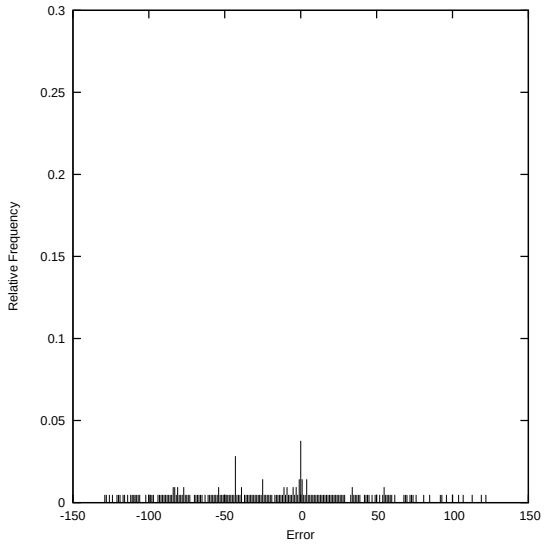


Figure 4.8: Relative frequency of the trigonometric prediction error.

set. It also showed good resilience to variation, responding well to fluctuations in the usage that the other methods did not necessarily handle as well. As shown by the histogram in Figure 4.9, there is a very high peak at the mean, close to 0, and a much narrower spread than the previous two methods.

4.2.4 Weighted Sum

As mentioned mentioned in Section 3.1.4, a weight vector that combined some of the previous methods was trained by hand. Random vectors were generated and compared on a subset of the training data and the best performing ones were then tested on a larger set of the training data. The best performing vector was the vector [0.65, 0.3, 0, 0.05],

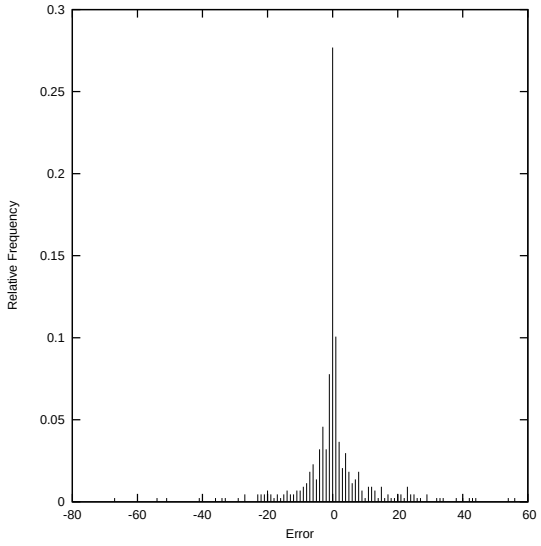


Figure 4.9: Relative frequency of the prediction error of the average gradient method, for the 2 most recent gradients.

the performance of which can be seen in Figure 4.10 and 4.11. As seen in Figure 4.10, the predictor is still prone to higher levels of inaccuracy during busy times of day (early afternoon in particular), but in general performs well: the distribution (Figure 4.11) is narrower overall, with a wider clustering around 0. This method works so well because it factors in parts of the other methods. The vector applies weights to estimations based on the current usage, the recent gradient of the curve, the usage from yesterday and the usage from a week ago. By combining all these different factors, resilience to local fluctuations is added to the predictor and it is ensured that the estimation does not rely too heavily on one particular feature of the recent usage. These plots show both the clustering of the errors around zero, indicating that the predictor performs much better than other methods. Plots of time against error and relative histograms of all methods are shown in Appendices A.6 to A.9.

4.2.5 Genetic Algorithm

After having trained the weight vector using a genetic algorithm, performance was expected to improve. On the training data this was the case, showing improvements over previous methods in all statistics measured (Table 4.1), most notably almost halving the mean from 3.277 down to 1.374, and a large drop in the mean squared error: 171.978 down to 107.385. On the test set (the data for which is shown in Table 4.2)

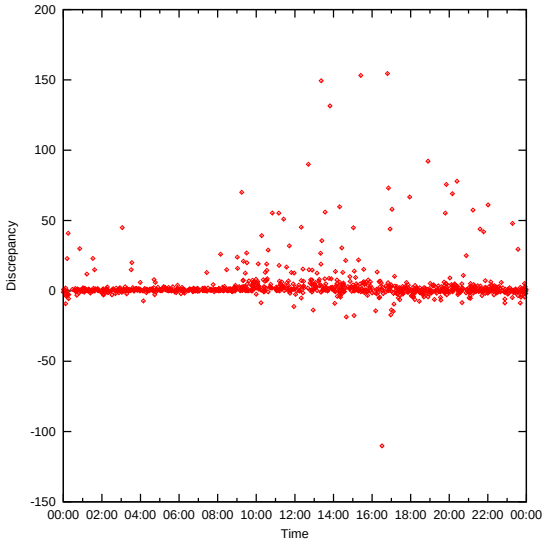


Figure 4.10: Plot of time of day against error for the weighted sum predictor.

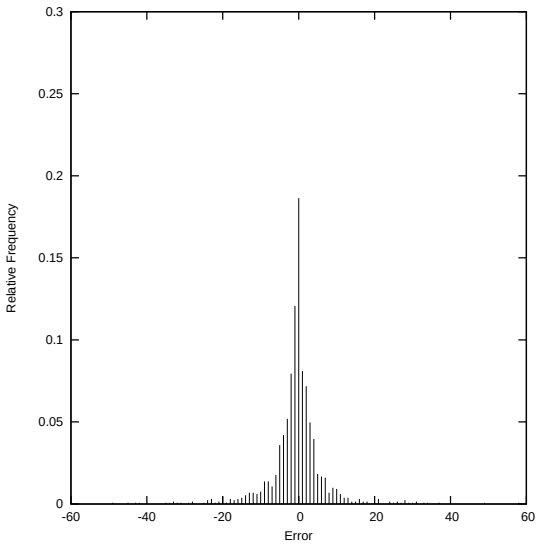


Figure 4.11: Relative frequency of the prediction error of the weighted sum predictor.

this difference in performance is even more pronounced, the only differnece being that the average gradient methods perform very similarly to the genetically trained vector. This is discussed further in section 4.2.6.

Finally, the new weight vector was used to plot a graph of time of day against the errors in prediction, similar to Figure 4.10. This is shown in Figure 4.12.

Estimation Method	Mean Error	MSE	Std. Dev.
Hand-trained vector	3.277	171.978	12.698
Genetically trained vector	1.374	107.385	10.271
Linear interpolation	15.667	512.764	16.349
Average gradient (2 steps)	1.918	215.424	14.551
Average gradient (4 steps)	1.932	192.999	13.757
Sine curve estimation	-5.478	1076.503	32.35

Table 4.1: Comparison of estimation methods using 10000 training set instances.

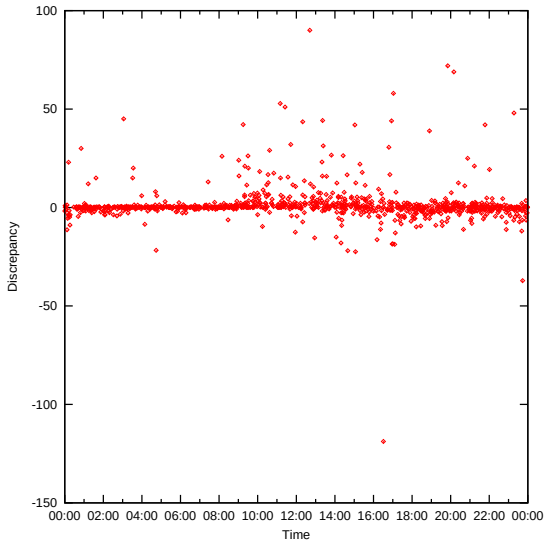


Figure 4.12: Plot of time of day against error for the genetic algorithm trained weight vector.

4.2.6 Overall Comparison

The results of comparing all methods on 10,000 samples from the training set are shown in Table 4.1, and the results of testing on a week of test data from the production database are shown in Table 4.2. This simply highlights the points made in the previous subsections. Figure 4.13 shows a side-by-side comparison of the performance of all methods: this graph plots the cumulative absolute error over 1,000 training data samples on the x-axis, and the cumulative frequency as a percentage on the y-axis. The shape of the curves shows that for a small sample all methods may be similar, but that the sine curve method and the linear interpolation method rapidly lose accuracy.

The other three methods are close in terms of performance, although as shown in

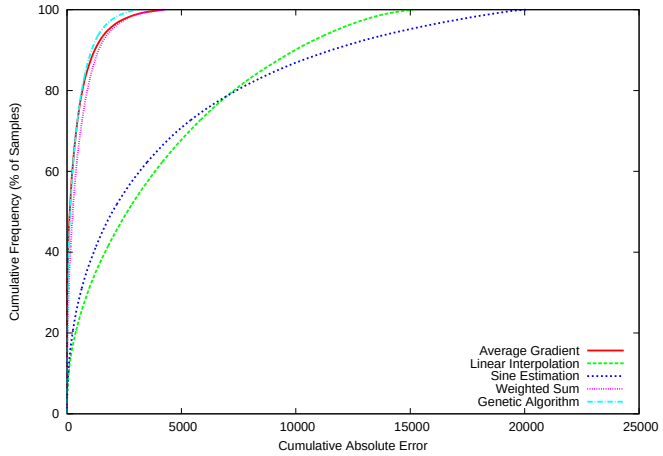


Figure 4.13: Graph of cumulative error for all methods.

Figure 4.14, the weighted vector trained by a genetic algorithm still performs the best on the training data.

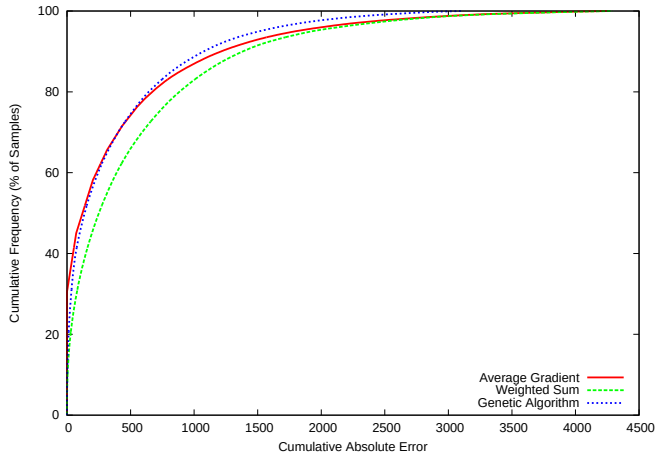


Figure 4.14: Graph of cumulative error for the three best methods.

In table 4.2, the difference in performance shown on the training data is even more pronounced. The genetically-trained vector shows a large improvement on the hand-trained vector: the mean error drops from 2.045 to 0.473. The only slight anomaly is that the average gradient methods perform much better on the test data, with a lower mean error than the more complex weighted vector methods. However both average gradient methods show both a higher MSE and a larger standard deviation than the genetically trained vector. This could be due to overfitting by the genetic algorithm used to find the weights for the vector.

Estimation Method	Mean Error	MSE	Std. Dev.
Hand-trained vector	2.045	65.359	7.821
Genetically trained vector	0.473	61.363	7.819
Linear interpolation	20.669	708.656	16.777
Average gradient (2 steps)	-0.01	66.175	8.135
Average gradient (4 steps)	-0.007	64.0	8.0
Sine curve estimation	-23.152	2985.289	49.49

Table 4.2: Comparison of estimation methods using one week of test set instances.

The improvement shown by the two weighted vector methods (especially the genetic algorithm trained vector) is significant on both the training and the test sets. This improvement is certainly due to the nature of the method: by taking the initial basic methods and combining them, the advantages of the methods are extracted and the flaws in each method are partially overcome. This could be further improved by using two different vectors depending on the time of day: one for night time where there is little change in usage, and one for the high variation daytime period.

Having mentioned use of neural networks in chapter one, explanation should be given as to why one has not been implemented in this case. A neural network may have shown even more improvement over the weighted vector method, but would have been much harder to implement within the structure of a Rails webapp. Due to the hidden values of the middle layers and the difficulty of storing such values, it would have been difficult to build a neural network in such a way that it performs well for single predictions, as each prediction only needs to happen a maximum of once in a 15 minute period thanks to the caching of the result. The hidden values of the middle layers would have had to be stored to allow the predictor to continue again in 15 minutes, or the code would have had to 'sleep' for 15 minutes. The second method would still have required some way to store the hidden values of the nodes, as otherwise this data would be lost when the system restarted or crashed. Trying to construct the network whilst maintaining a reasonable standard of code readability and class separation would also have been very difficult, if the Model-View-Controller structure of the Rails system was to be adhered to.

The focus during this project was to build a system that gave a reasonable indication of whether the usage was increasing or decreasing. If the system was to be packaged as a saleable product, then this would lose focus in favour of more accurate forecasting.

In that case, the system could be redesigned with a different structure and the neural network could perhaps be implemented successfully.

Chapter 5

Conclusion

In this project, a working system has been built that can estimate incipient usage of machines in Appleton Tower. The specification outlined in Section 2.2 (listed again below) has been met, and the project delivered successfully. Various methods of predicting future usage have been implemented and evaluated, and a reasonable standard of prediction reached. There remains further work that could be done to improve the prediction yet more.

The original specification was:

Simple availability - This has been achieved through use of current web technologies such as CSS3, allowing the system to be used from a mobile device as well as from a laptop or desktop.

Unobtrusive authentication - this was not required, as no sensitive data has been collected or displayed.

Adherence to data protection standards - since no sensitive data has been collected, this was not an issue.

Use of a modern web framework - the Ruby on Rails framework has been used, as well as technologies such as Javascript to allow for 'smart' behaviour such as searching as you type.

Central data storage - initially stored in a network shared folder, the data is now stored on a central PostgreSQL database.

Secure communication to and from this data store - the PostgreSQL server is protected by the Kerberos[1] authentication system used by the School of Informatics.

[1] See `http://web.mit.edu/kerberos/#what_is`.

Fast performance of estimation algorithms - the method used to estimate the future usage runs in a matter of seconds. To increase performance further and add security the result is cached for 15 minutes, meaning that the method runs at most once in each 15 minute window.

Problems have been encountered and overcome, and the project has adapted to account for changes as they have occurred. The data collection changed from a filesystem to a database due to security issues, which took a small amount of time but was a simple problem. Some bugs in the CSS code still remain, but these do not affect the functionality of the project and are merely cosmetic. During the project the code has been altered and structured continuously in order to maintain performance standards: database queries have been changed in order to reduce the number of requests that the web server needs to make, and code on the web server has been profiled before being integrated to the live web system. This process has helped to find 'bottlenecks' in the code and to overcome them. Other problems that have been considered and (in part) overcome include those such as the 'threshold' for the status of machines ('asleep' or 'stale'), and issues raised by user feedback. The interface underwent a major overhaul: a landing page was added with additional information, and the manner in which data is presented has changed from the initial version of the system. This can be seen in Appendices A.1, A.2 and A.3.

There is much scope for extensions to this project. The system could be integrated into the Computing Support form, in order to provide additional information with help requests. Additional information could be collected from the machines, allowing users to use the system for more than simply finding free computers. Authentication could be added to the system to allow more sensitive information (such as who is logged in) to be available. By using the API, native mobile phone applications could be created to provide a better interface on Android or iOS devices.

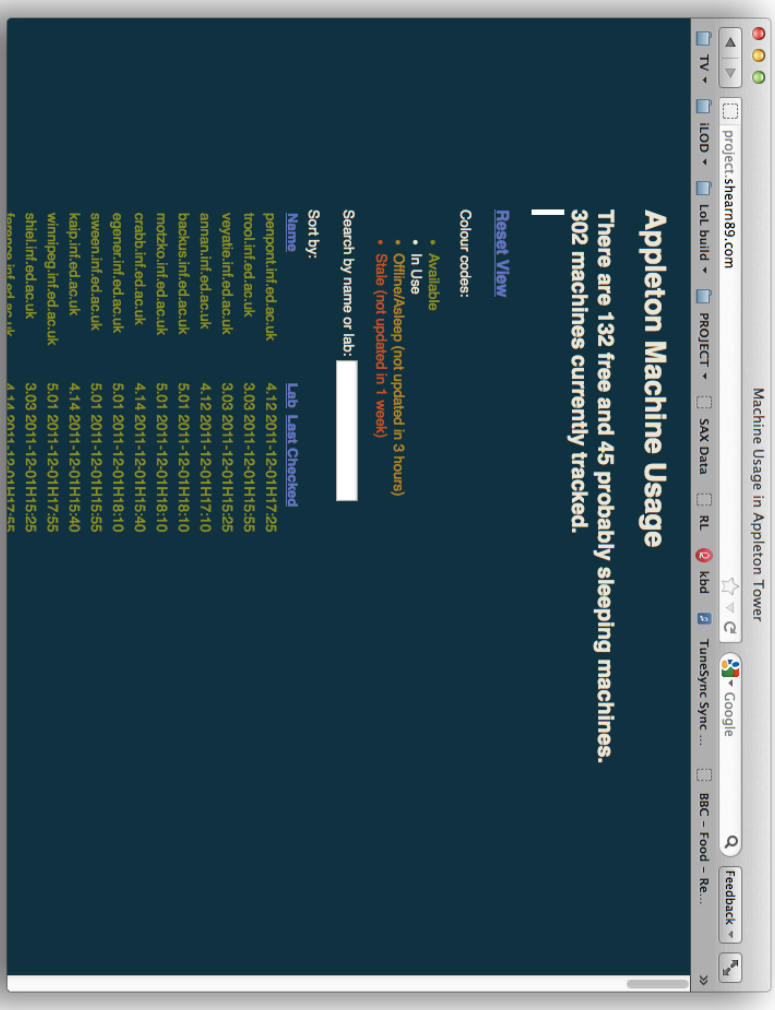
As mentioned at the end of the previous chapter, the system could also be packaged for deployment to other networks for schools, universities or commercial companies. This would not be impossible: the cron script could be re-written as a compiled executable to be installed on the client machines, with a configuration file that directed it to the central database. The server code could be easily packaged for redistribution as source files, with only small background knowledge of web servers required to set up the system. Maintenance would be limited to checking that client machines have a network connection to reach the database, and (currently) adding some information by hand: a process that could easily be automated.

It is hoped that this project will stay in use next year, and perhaps be taken on and extended as part of either another 4th Year Project, or simply as a self-interest project. There has also been some discussion with Computing Support about them taking on maintenance of the project.

Appendix A

Appendix

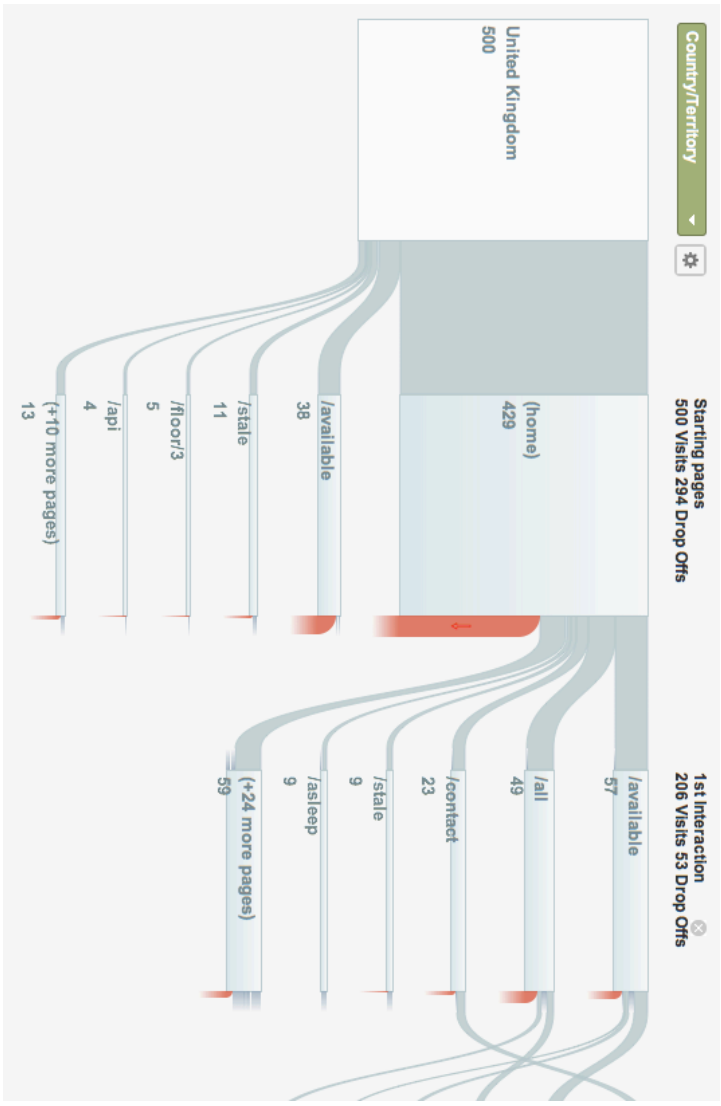
A.1 Initial Main Page View



A.2 Initial Main Page View

A.3 Final Main Page View

A.4 Visitor Flow



A.5 Example Feedback Submissions

A.5.1 Example 1

Dear Alex,

I'd like to tell you about a <feedback> with your project.

Message: Looking really nice. One thing I noticed is that the what the top bar options do isn't immediately obvious. Maybe 'Main Page' could be 'All' to suggest that they are all essentially filters?

From: Callum S.

A.5.2 Example 2

Dear Alex,

I'd like to tell you about a <feedback> with your project.

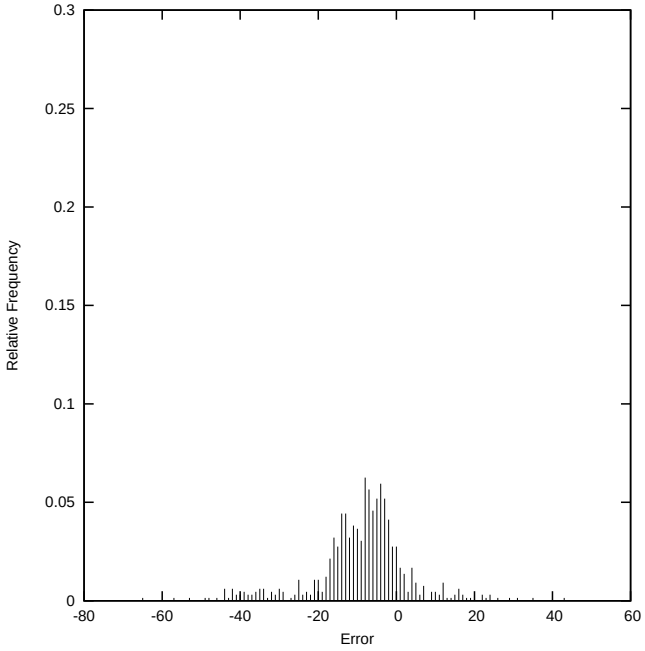
Message: Very nice, useful for checking if a room is probably free (i.e. if there are no machines taken its probably free).

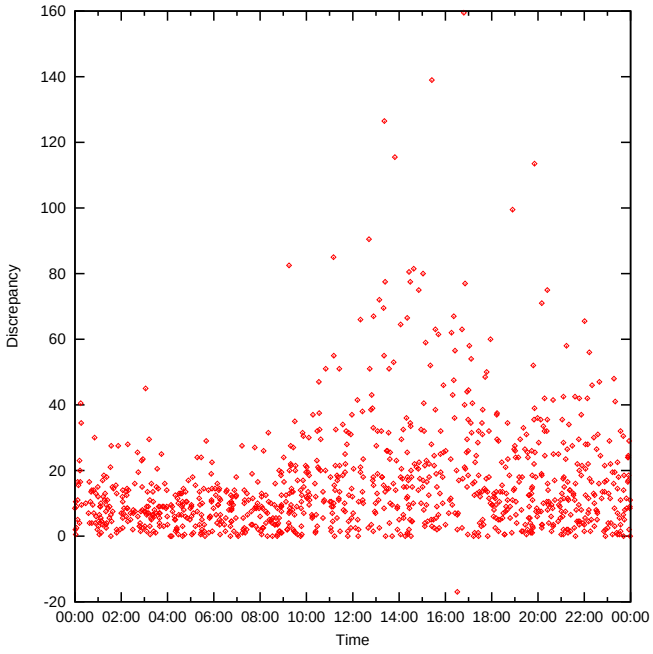
Some thoughts:

- Usernames and the ability to search by them would be useful (especially to find out where **I** am logged in), but may cause ethical issues.
- The numbers below the summary on the left make no sense without the context of the 'info'; page. Even now I can't remember what they mean.
- It would be useful to be able to filter by lab on the 'All Machines', 'Available', etc pages, instead of having to go to 'Floors'; or 'Labs'.
- 'Stale' isn't that descriptive to me as a random user. Is that just broken?
- Some machines don't have lab names - are they useful if this is the case?
- Yay api!

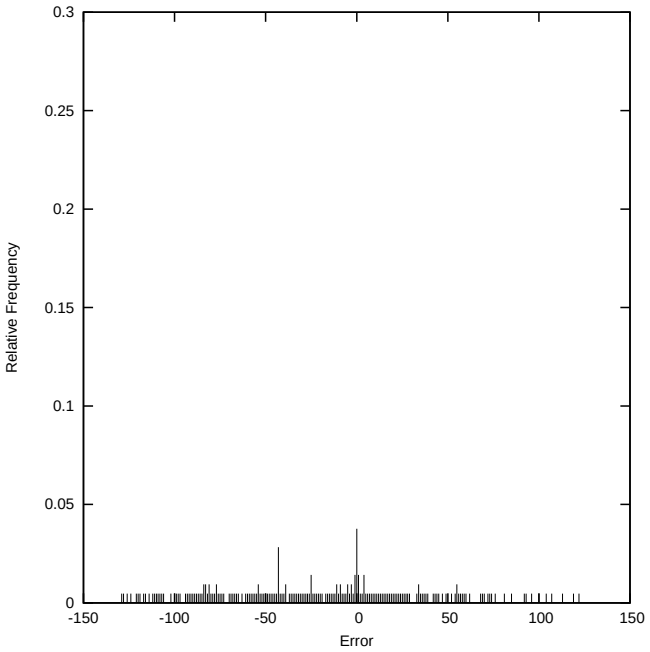
From: Stephen M.

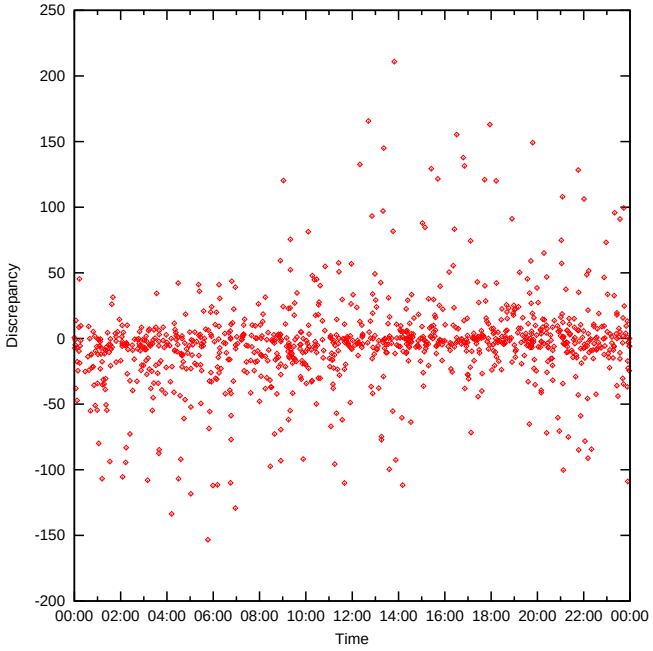
A.6 Histogram and Time plot for the linear interpolation method



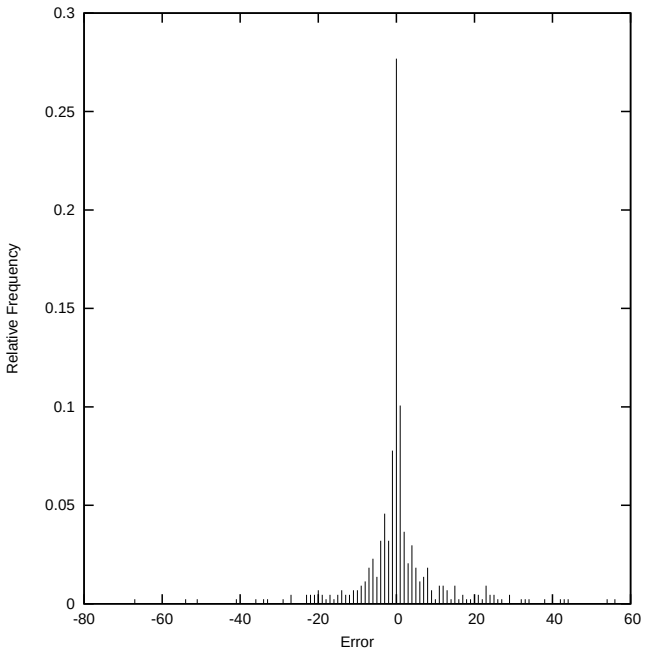


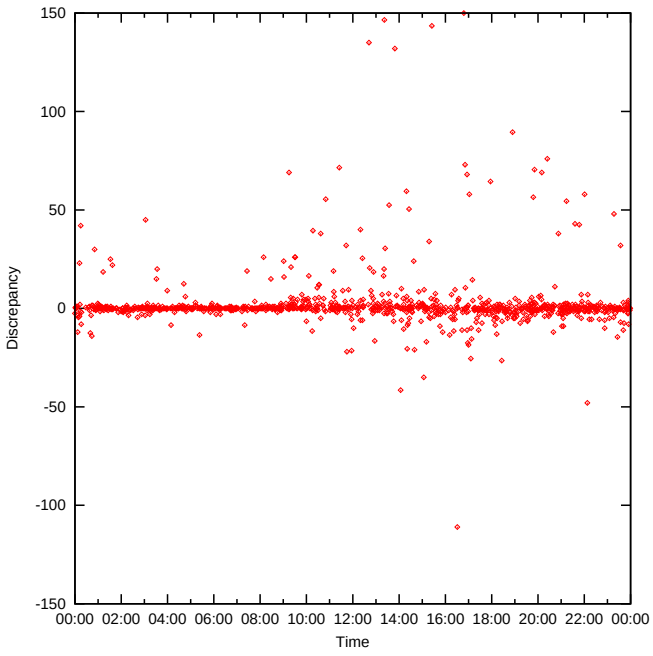
A.7 Histogram and Time plot for the sin estimation method



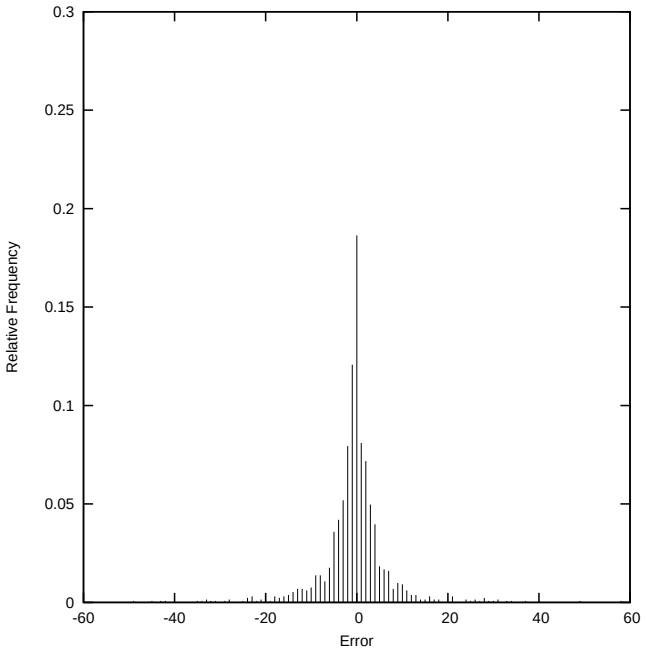


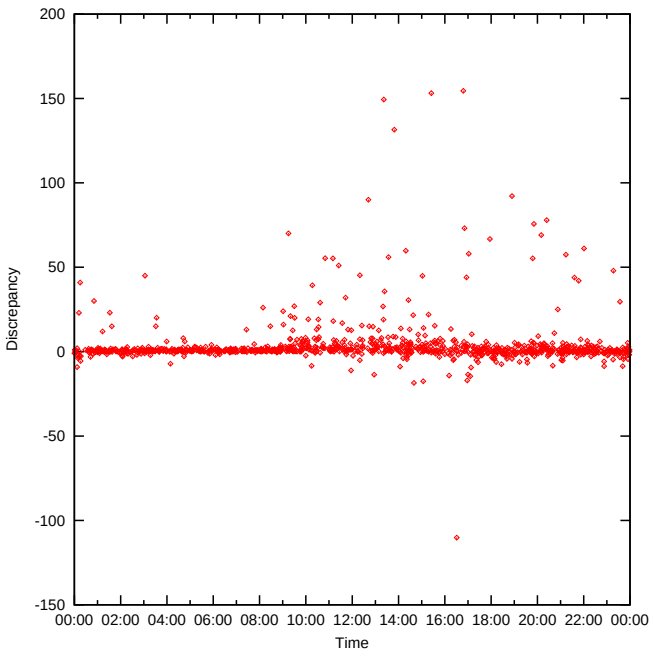
A.8 Histogram and Time plot for the average gradient method





A.9 Histogram and Time plot for the weighted sum method





Bibliography

[Brandstetter, 2011] Brandstetter, E. (2011). Answer to question: "SQL - JOIN on two result tables, ideas to refactor?". http://stackoverflow.com/questions/8689061/sql-join-on-two-result-tables-ideas-to-refactor.

[Charytoniuk and Chen, 2000] Charytoniuk, W. and Chen, M.-S. (2000). Very short-term load forecasting using artificial neural networks. *Power Systems, IEEE Transactions on*, 15(1):263 –268.

[Dai and Wang, 2007] Dai, W. and Wang, P. (2007). Application of pattern recognition and artificial neural network to load forecasting in electric power system. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 1, pages 381 –385.

[Eubank and Speckman, 1990] Eubank, R. L. and Speckman, P. (1990). Curve fitting by polynomial-trigonometric regression. *Biometrika*, 77(1):1 – 9.

[Liao and Tsao, 2006] Liao, G.-C. and Tsao, T.-P. (2006). Application of a fuzzy neural network combined with a chaos genetic algorithm and simulated annealing to short-term load forecasting. *Evolutionary Computation, IEEE Transactions on*, 10(3):330 – 340.

[Lin et al., 2007] Lin, J., Keogh, E., Wei, L., and Lonardi, S. (2007). Experiencing sax: a novel symbolic representation of time series. *DATA MINING AND KNOWLEDGE DISCOVERY*, 15(2):107 – 144.

[Ma et al., 2011] Ma, X., Li, H., and Djouadi, S. (2011). Stochastic modeling of short-term power consumption for smart grid: A state space approach and real measurement demonstration. In *Information Sciences and Systems (CISS), 2011 45th Annual Conference on*, pages 1 –5.

[Mitchell, 1998] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA.

[Osman et al., 2009] Osman, Z., Awad, M., and Mahmoud, T. (2009). Neural network based approach for short-term load forecasting. In *Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES*, pages 1 –8.

[Pitidis, 2012] Pitidis, M. (2012). Benchmark load balancing. `http://mpitid.github.com/posts/2012-02-08-benchmark-load-balancing.html`.

[Qingle and Min, 2010] Qingle, P. and Min, Z. (2010). Very short-term load forecasting based on neural network and rough set. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, volume 3, pages 1132 –1135.

[Williams, 2010] Williams, C. (2010). "time series modelling: AR, MA, ARMA and All That". Handout from the University of Edinburgh's Probabilistic Modelling and Reasoning course.