## **Server Load Prediction**

Suthee Chaidaroon (<u>unsuthee@stanford.edu</u>)
Joon Yeong Kim (<u>kim64@stanford.edu</u>)
Jonghan Seo (<u>jonghan@stanford.edu</u>)

### **Abstract**

Estimating server load average is one of the methods that can be used to reduce the cost of renting a cloud computing service. By making an intelligent guess of how busy a server would be in the near future, we can scale up or down computing requirements based on this information. In this experiment we have collected website access and server uptime logs from a web application company's website and extracted information from these data. We are able to build a prediction model of server load average by applying a linear regression, locally weight linear regression, multinomial logistic regression, and support vector regression. Our experiment shows a trade-off between computing efficiency and prediction accuracy among our selected learning algorithms. The locally weight linear regression has the most accurate prediction of server load average but takes a long time to converge. On the other hand, the support vector regression converges very quickly but performs poorly on sparse training samples. At the end of this paper, we will discuss on improvements we can do for future experiments.

#### Introduction

Our clients operate a website and rent the cloud servers of Amazon EC2. The EC2 allows them to scale the computing capacity up and down as they need and they only pay for the computing capacity that their website actually uses. This task is tedious and can be automated. We think that there should be a way to automatically this task without having clients to monitor their server activities all the time. Our solution is to estimate a server load average based on the website users' activities. We assume that the number of activities and transactions determines a server load average. We use this assumption to build a prediction model.

# Methodology

We collected website access logs and server uptime logs during the past few months and processed those two files by merging transactions and load average that occur in the same time interval. We then use this processed file as our training samples. Each line of a training sample file has a time information, number of unique IP addresses, the total size of contents, protocol, and the load average of the last 1, 5, and 15 minutes. Below are parts of access log and uptime files:

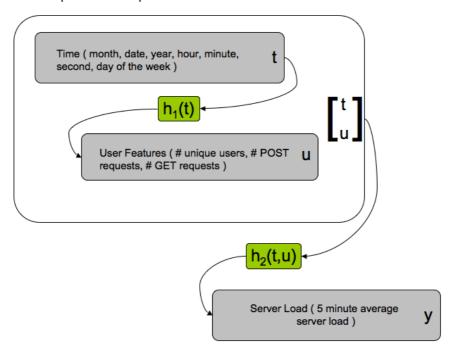
```
99.63.73.46 - [24/Oct/2009:00:00:05 +0000] "GET /cgi-bin/uptime.cgi HTTP/1.0" 200 71
99.63.73.46 - [24/Oct/2009:00:01:21 +0000] "GET /logs/error_log HTTP/1.1" 200 3259
99.63.73.46 - [24/Oct/2009:00:01:31 +0000] "GET /logs/error_log HTTP/1.1" 304 -
99.63.73.46 - [24/Oct/2009:00:05:04 +0000] "GET /cgi-bin/uptime.cgi HTTP/1.0" 200 71
99.63.73.46 - [24/Oct/2009:00:07:00 +0000] "GET /crossdomain.xml HTTP/1.1" 200 202
99.63.73.46 - [24/Oct/2009:00:07:00 +0000] "GET /m/s/248/f/1280/f248-1280-6.bpk HTTP/1.1" 200 680
99.63.73.46 - [24/Oct/2009:00:07:00 +0000] "GET /m/s/271/f/1301/f271-1301-6.bpk HTTP/1.1" 200 23000
```

Table 1. Part of access  $\log^1$ 

```
Mon Oct 12 15:50:02 PDT 2009:: 18:50:02 up 95 days, 27 min, 0 users, load average: 0.23, 0.06, 0.02
Mon Oct 12 15:55:01 PDT 2009:: 18:55:02 up 95 days, 32 min, 1 user, load average: 0.00, 0.02, 0.00
Mon Oct 12 16:00:02 PDT 2009:: 19:00:03 up 95 days, 37 min, 1 user, load average: 0.00, 0.00, 0.00
Mon Oct 12 16:05:06 PDT 2009:: 19:05:07 up 95 days, 42 min, 1 user, load average: 0.00, 0.00, 0.00
Mon Oct 12 16:10:01 PDT 2009:: 19:10:02 up 95 days, 47 min, 1 user, load average: 0.53, 0.41, 0.16
Mon Oct 12 16:15:01 PDT 2009:: 19:15:02 up 95 days, 52 min, 1 user, load average: 0.05, 0.20, 0.14
```

After parsing and processing the data, the features available for us to use are: day of the week (Monday, Tuesday, etc.), month, date, year, hour, minute, second, number of unique users, number of POST requests, number of GET requests, and content size of the interaction.

The diagram below represents our prediction model:



At the very bottom level, we predict the server load based on time. The problem of predicting the server load only based on time would be extremely difficult to solve. So, we decided to divide the problem into two parts, one to predict the server load, and another, to predict the user features based on time. For regressions and classifications tests in the following section, we will assume that the user features were correctly predicted.

The output value that we tested was chosen to be the 5-minute average server load because the uptime command was run every 5 minutes on the server.

## **Experimental Result**

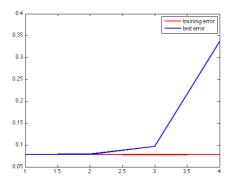
#### **Linear Regression**

We assume that the user features such as number of unique users and number of requests, etc. were correctly predicted by  $h_1(t)$ . This assumption had to be made because of amount of data which was not sufficient to predict the user features since the logs had been made only since this August. The features month and year were dropped from the feature lists to create a training matrix that is of full rank.

To create a larger set of features, we concatenated powers of the matrix X to the original matrix. So, our training matrix,

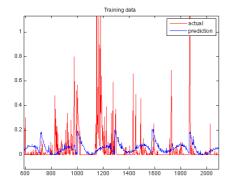
$$X = [X_{orig} X_{orig}^2 ... X_{orig}^n].$$

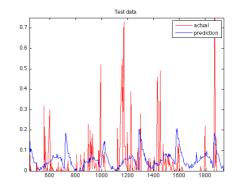
Since the full rank of matrix X could not be preserved from n = 5, we could only test the cases where n = 1,2,3,4. To come up with the optimal n, some tests were done to compute the average training and test errors. The following graph shows the errors.



As we can see from the graph above, the overall error actually got worse as we increased the number of features used. However, because we thought that n = 3 gave a prediction curve of shape closer to the actual data, we picked n = 3. Below are the plots of parts of the actual and predicted server load for training and test data. The corresponding errors with respect to n are listed below:

| n | average<br>training error | average test<br>error |  |
|---|---------------------------|-----------------------|--|
| 1 | 0.0784                    | 0.0783                |  |
| 2 | 0.0780                    | 0.0787                |  |
| 3 | 0.0787                    | 0.0922                |  |
| 4 | 0.0784                    | 0.3471                |  |





As the two graphs display, the predictions are not as accurate as it ought to be. However, the trends in the server load are quite nicely fit. For the above regression, the training error was 0.0652, and the test error was 0.0683. Since the shape of the curve is quite nicely fit, for our purpose of predicting the server load to increase or decrease the number of servers, we could amplify the prediction curve by a constant factor to reduce the latency experienced by the users, or do the opposite to reduce the number of operating servers.

### **Multinomial Logistic Regression**

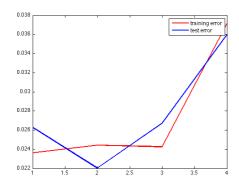
In order to transform our problem into a logistic regression problem, we classified the values of the server load (CPU usage) into ten values (0,1, ..., 9) as follows: if the CPU usage was less than or equal to 0.10, it would be 0; else if less than or equal to 0.20, it would be 1; ...; else if less than or

equal to 0.90, it would be 8; else it would be 9. We trained 10 separate hypotheses *hi* for each classification. For testing, the system would choose the classification that gives us the highest confidence level.

We use the same approach as linear regression case to find the maximum power of the matrix we want to use when coming up with the matrix X, where

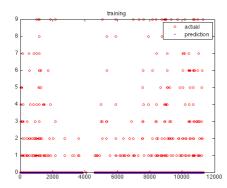
$$X = [Xorig Xorig^2 ... Xorig^n].$$

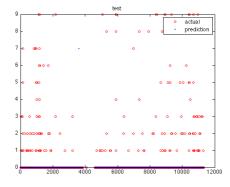
The following plots for training and test errors were acquired:



| n | average<br>training error | average test<br>error |
|---|---------------------------|-----------------------|
| 1 | 0.0236                    | 0.0263                |
| 2 | 0.0244                    | 0.0220                |
| 3 | 0.0243                    | 0.0267                |
| 4 | 0.0371                    | 0.0360                |

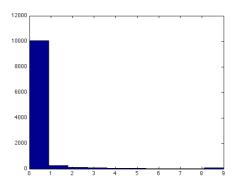
As in the linear regression case, we chose n = 3 for logistic regression. The following is the resulting plot of our prediction compared to the actual data.





As we can see, the multinomial logistic regression always outputs 0 as the classification no matter what the features are. This is in part because too many of our data points has Y-values of 0 as the below histogram shows. Because the company we got the data from has just launched, the server load is usually very low. As the website becomes more popular, we would be able to obtain a better

result since the Y-values will be more evenly distributed.

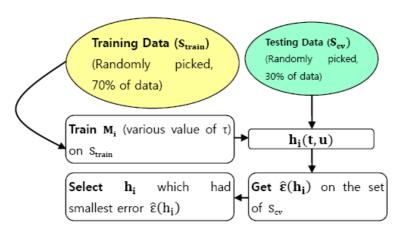


### Locally weight linear regression

Like linear regression, we first assumed all features used as a input, such as number of users, total number of requests and so on, were correctly predicted by  $h_1(t)$  for the same reason. For building up a hypothesis, we had to determine what weight function we would use. In common sense, it is reasonable that server loads after five or ten minutes would be dependent on previous several hours of behavior so we decided to use

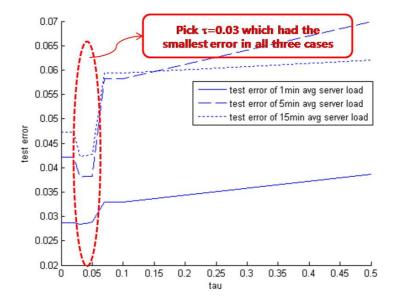
$$w^{(i)} = exp\left(-\frac{(x^{(i)}-x)^2}{2\tau^2}\right)$$

as a weight function in our hypothesis. However, for using the weight function we were needed to find out the optimal value of  $\tau$  which is the bandwidth of the bell-shaped weight function. So we used Hold-out Cross Validation for picking up the optimal  $\tau$  and the way and results are as follows.

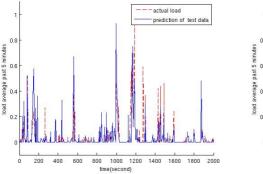


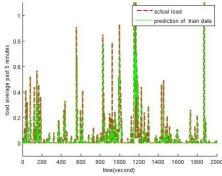
| au     | $\widehat{\varepsilon}_{1min}$ | $\widehat{arepsilon}_{5min}$ | $\widehat{\varepsilon}_{15min}$ |
|--------|--------------------------------|------------------------------|---------------------------------|
| 0.0001 | 0.0287                         | 0.0422                       | 0.0472                          |
| 0.005  | 0.0287                         | 0.0422                       | 0.0427                          |
| 0.01   | 0.0287                         | 0.0422                       | 0.0472                          |
| 0.02   | 0.0287                         | 0.0422                       | 0.0472                          |
| 0.03   | 0.0284                         | 0.0380                       | 0.0423                          |
| 0.04   | 0.0286                         | 0.0382                       | 0.0424                          |
| 0.05   | 0.0288                         | 0.0382                       | 0.0427                          |
| 0.07   | 0.0329                         | 0.0582                       | 0.0594                          |
| 0.1    | 0.0330                         | 0.0584                       | 0.0594                          |
| 0.5    | 0.0387                         | 0.0700                       | 0.0620                          |

| 1  | 0.0441 | 0.0636 | 0.0572 |
|----|--------|--------|--------|
| 5  | 0.0609 | 0.0666 | 0.0600 |
| 10 | 0.0650 | 0.0621 | 0.0563 |



First of all, we had picked 70% of the original data randomly and named it as training data and did again 30% and named testing data. Then we trained the hypothesis with various values of  $\tau$  on the training data and applied testing data on the hypotheses and got test errors in each value of  $\tau$ . Above chart shows test errors of sever loads tested on hypothesis h(t,u) with various  $\tau$ . Above plot displays the pattern of the test errors with respect to  $\tau$ . As we can see in the chart and graph, at  $\tau$ =0.03, the test error is the smallest value on all of three cases so we picked up this one for our weight function.



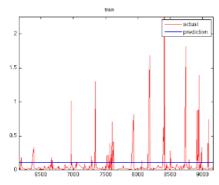


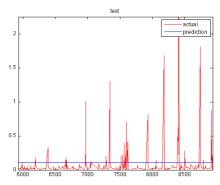
Above plots show the actual server load and prediction of test and train data, respectively. As making a prediction on every unit of query point x is prohibitively time consuming process, the prediction is made on each query point in train and test data. Though it seems that it followed the pattern of actual load, sometimes it could not follow spikes and it needed very long time to predict even on query points of test so it might not be suitable to be used in a real time system which would be needed in a general sever load prediction system. Thus, it is desirable that locally weighted regression will be trained just on previous several hours not on whole previous history if sufficient training data were given.

#### **Support Vector Regression**

We use SVM Light<sup>3</sup> to perform a regression. We remove all zero features (which indicating that a server is idle) and select training samples randomly. The trade-off between training error and margin is 0.1 and epsilon width of the tube is 1.0. We use a linear kernel and found that there are 148 support

vectors and the norm of the weight vectors are 0.00083. The training error is **0.479** and the testing error is **0.319**.





Above plots are parts of the results from the Support vector regression. The red curve is an actual data and the blue curve is the prediction. Although the support vector regression is quickly converged for 6,000 training data set, its prediction has higher training and testing errors than the rest of learning algorithms. We tried to adjust an epsilon width and found that the higher epsilon width will only offset an estimation by a fixed constant.

#### Conclusion

The prediction from the locally weight linear regression yields the smallest testing error but it takes an hour to process 6,000 training samples. The linear regression also yields an acceptable prediction when we concatenated the same feature vectors up to a degree of 3. Both Multinomial logistic regression and support vector regression do not perform well with sparse training data. Support vector regression, however, takes only a few minutes to converge, but yields the largest training error.

#### **Issues**

#### Bias with sparse data

Our learning algorithms cannot estimate an uptime reliably because there are almost 90% of the training samples have an uptime value between 0.0 and 0.1. We believe that the data we have collected are not diverse due to the low number of activities of our testing web server.

#### Not enough features

In our experiment, a dimension of the feature vectors are 10. 7 of them are part of a timestamp. We think that we do not have enough data to capture the current physical state of the cloud servers such as the memory usage, bandwidth, and network latency.

#### weak prediction model

We think that our decision to use the number of transactions and unique users as part of the feature vectors does not accurately model the computing requirements of the cloud servers. An access log file and uptime file are generated separately by different servers. Our assumption that a user activity is the only main cause of high server load average may not be accurate. It is possible that a single transaction could cause a high demand for a computing requirement.

#### **Future work**

We have found a research paper that describes the use of website access logs to predict a user's activities by clustering users based on their recent web activities, we like this idea and think that it is possible to categorize website users based on their computing demands. When these type of users are using the website, we think there is a high chance of increasing the computing requirements.

#### References

- [1] Access Log specification. <a href="http://httpd.apache.org/docs/2.0/logs.html">http://httpd.apache.org/docs/2.0/logs.html</a>
  [2] Uptime specification. <a href="http://en.wikipedia.org/wiki/Uptime">http://en.wikipedia.org/wiki/Uptime</a>
  [3] <a href="https://en.wikipedia.org/wiki/Uptime">SVMLight</a>. Vers. 6.02. 382K. Aug 14. 2008. Thorsten Joachims. Nov 15. 2009.