INTELLIGENT WEB CACHING USING MACHINE LEARNING METHODS

Sarina Sulaiman[*], *Siti Mariyam Shamsuddin*[*], *Ajith Abraham*[†], *Shahida Sulaiman*[*]

Abstract: Web caching is a technology to improve network traffic on the Internet. It is a temporary storage of Web objects for later retrieval. Three significant advantages of Web caching include reduction in bandwidth consumption, server load, and latency. These advantages make the Web to be less expensive yet it provides better performance. This research aims to introduce an advanced machine learning method for a classification problem in Web caching that requires a decision to cache or not to cache Web objects in a proxy cache server. The challenges in this classification problem include the issues in identifying attributes ranking and improve the classification accuracy significantly. This research includes four methods that are Classification and Regression Trees (CART), Multivariate Adaptive Regression Splines (MARS), Random Forest (RF) and TreeNet (TN) for classification on Web caching. The experimental results reveal that CART performed extremely well in classifying Web objects from the existing log data with a size of Web objects as a significant attribute for Web cache performance enhancement.

Key words: *Classification of Web objects, Web caching, machine learning methods*

Received: February 2, 2010
Revised and accepted: September 22, 2011

1. Introduction

Caching operation can execute in a client application, and generally, it is embedded in most Web browsers [1, 2, 3, 4, 5, 6]. A number of products extend or replace the embedded caches with systems that contain larger storage, more features, or better performance. In most cases, these systems only cache net objects from many servers for a single user [7, 8].

Caching can also be operated between the client and the server as a part of proxy cache [9, 10, 11], which is often located close to network gateways to decrease the

[*]Sarina Sulaiman, Siti Mariyam Shamsuddin, Shahida Sulaiman
Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia, E-mail: `sarina@utm.my, mariyam@utm.my, shahida@utm.my`

[†]Ajith Abraham
Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, WA, USA; Faculty of Electrical Engineering and Computer Science, VSB – Technical University of Ostrava, Ostrava, Czech Republic, E-mail: `ajith.abraham@ieee.org`

bandwidth connections. These systems can serve many users or clients with cached objects from many servers. In fact, Davison [7] reported that the usefulness of Web caching is up to 80% for some installations based on Web cache objects requested by one client for later retrieval by another client. Even for better performance, many proxy caches are part of cache hierarchies [12]; a cache can appeal to neighboring caches for a requested document to lessen the need for direct fetching.

Furthermore, the location of caches can be directly in front of a particular server, to reduce the number of requests that the server must handle. Most proxy caches are available in this way with different names; reverse cache, inverse cache, or sometimes httpd accelerator, to replicate the fact that it caches objects for many clients but normally from one server [13, 14]. Many solutions are developed for caching either from proxy cache or on Web server using statistical or machine learning methods. In this study, we investigate the performance of Classification and Regression Trees (CART), Multivariate Adaptive Regression Splines (MARS), Random Forest (RF) and TreeNet (TN) as classifier of the Web cache objects in a proxy cache server.

The rest of the paper is organized as follows: Section 2 describes the related works, followed by the introduction on the machine learning methods in Section 3. Section 4 is on experimental setup and Section 5 illustrates the performance result of the proposed method. Section 6 discusses the result from the experiment and finally, Section 7 concludes the article.

2. Related Works

Many researchers have looked for ways to improve current caching techniques. Padmanabhan and Mogul [15] proposed a predictive model as a server hint. The proposed model is equipped with a server that is able to create Markov model by predicting the probability of object A tagging along with next n requests and object B (n is a parameter of the algorithm). The server will use the model to produce a forecast for subsequent references and proposed the forecast to the client. The client will use the forecast result to pre-fetch an object on the server only if that object is not in the client cache. The simulation shows that the proposed model is able to reduce the latency until 45%. However, their technique has a limitation as it also makes network traffic larger by two times than the pre-fetch without the proposed technique [15]. That is why many researchers try to acquire latency and network reduction at the same time.

In another work, Bestavros and Cunha [16] presented a model for the speculative dissemination of World Wide Web data. Their work illustrates that reference patterns from a Web server can be the main source of information for pre-fetching. They also discovered latency reduction up to 50%, besides the increment in the bandwidth utilization.

On the other hand, Pallis et al. [9] proposed a pre-fetching based on clustering method. Web pre-fetching is an attractive solution to reduce the network resources consumed by Web services as well as the access latencies perceived by Web users. Unlike Web caching, which exploits the temporal locality, Web pre-fetching utilizes the spatial locality of Web objects. Specifically, Web pre-fetching fetches objects that are likely accessible in the near future and stores them in advance. In this context, a sophisticated combination of these two techniques may cause significant improvements on the performance of the Web infrastructure.

Kroeger et al. [17] observed a local proxy caching which is able to decrease latency until 26%, while pre-fetching could decrease latency at 57%. The combination of both local proxy caching and pre-fetching contributes to better latency reduction until 60%. Furthermore, they also found that their proposed algorithm on pre-fetching has contribution to reducing latency. The work also shows that pre-fetching can provide double improvement on caching. However, it only decreases the latency [18], see Fig. 1.

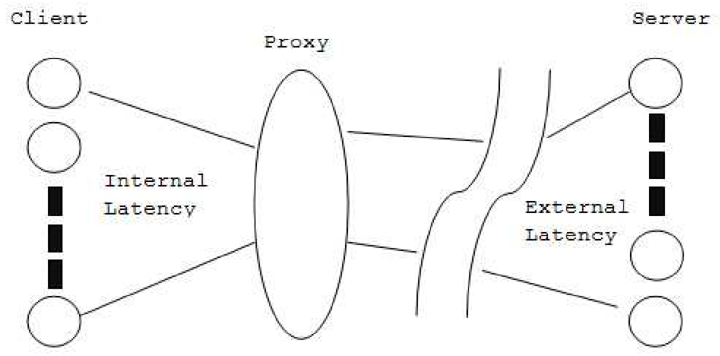


Fig. 1 *Proxy configuration [18].*

Xu et al. [19] proposed a solution by creating a proxy management. The caching dynamic content is obtained by generating data and personalise data that contributes up to 30-40% of the total traffic. These types of data are normally identified as "uncachable". To further improve Web performance, reverse caching can make dynamic content to be more cachable and manageable [9]. However, the challenge is still on how to maintain efficiently the consistency between the cached content and the data source that frequently changes. Another important issue is the analysis of query semantics to evaluate a complex query over the cached content.

The next issue is caching streaming objects: The prediction is that streaming media, such as music or video clips, symbolize a significant portion of the Web traffic over the Internet. Due to the distinct features of streaming objects like huge size, long duration, intensive use of bandwidth, and interactivity, conventional proxy caching techniques are unable to solve this problem. Hence, a number of studies have proposed partial caching algorithms in recent years [10, 11]. The proposed algorithms show that even if a small size of video is stored on the proxy, the consumption of network reduces significantly.

Xu et al. in [19] generally stated that the collaboration among proxies is based on the premise, and it would be faster and cheaper to fetch an object from another close proxy rather than the origin server. See Fig. 2 for the cooperative cache organization.

Teng et al. [20] proposed a combination between Web caching and Web pre-fetching. The combination is possible because the Web caching technique uses the

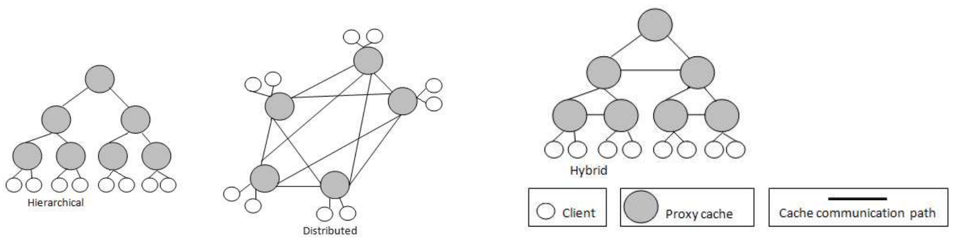


Fig. 2 *Examples of different cooperative cache organizations [19].*

temporal locality while the Web pre-fetching technique utilizes the spatial locality of Web objects. The proposed technique is obtained by evaluating the pre-fetching rules.

There is continuity in the research especially on clustering pre-fetching, caching on proxy level and designing a cache organization. The list of issues on Web caching and pre-fetching is available in Tab. I.

Considering that there have been several caching policies proposed in the past, the challenge is how to extend them by using data mining techniques. It presented a clustering-based pre-fetching scheme using a graph-based clustering algorithm that identifies clusters of “correlated” Web pages based on users’ access patterns and then it creates adaptive websites [21].

Nevertheless, the research in this paper proposes a scheme that integrates data mining techniques into a cache server for Web object classification, thus improving its performance. Through a simulation environment using the Salford Systems [22] and a real data set, the proposed scheme can be an effective way in improving the performance of the Web caching environment. The use of the real data set for classification of Web object data is based on two different Web log data from Boston University (BU) and E-learning@UTM (EL) websites.

3. Machine Learning Methods

3.1 Classification and regression trees

Classification and Regression Trees (CART) is a robust decision-tree tool for data mining, pre-processing and predictive modeling, suggested by Breiman et al. [23]. CART can analyze complex data for patterns and relationships and uncovering hidden structures [24]. Moreover, CART is a nonparametric technique that can select variables from a large data set and their interactions that are very important in determining the outcome variable to be analyzed.

Decision Tree (DT) induction is one of the classification algorithms in data mining. The classification algorithm is inductively learned to construct a model from the pre-classified data set. Inductive learning means making general assumptions from the specific examples in order to use those assumptions to classify unseen data. The inductively learned model of classification algorithm is known as classifier. Classifier may be viewed as a mapping from a set of attributes to a particular

Author	Proposed Solution	Weakness
Ali Ahmed and Shamsuddin (2011) [43]	– Proposed intelligent scheme based on neuro-fuzzy system by splitting cache to two caches, short-term cache and long-term cache, on a client computer for storing the ideal Web objects and removing the unwanted objects in the cache for more effective usage.	– Requires extra computational overhead for training process.
Albers (2010) [42]	– Reduced web caching with request reordering to a problem of computing batched service schedules using a deterministic algorithm.	– Do not know how to modify the online algorithm.
Hung et al. (2008) [11]	– Proposed two levels caching to improve the average of cache access time and an adaptive pre-fetch strategy.	– Still hard to adapt with different application.
Nair and Jayasudha (2007) [10]	– Proposed an intelligent agent that is able to monitor bandwidth usage and helps the prediction engine to decide the number of Web pages to be pre-fetched.	– Needs an improvement in artificial intelligence of the agent.
Jin et al. (2007) [41]	– Proposed an adaptive pre-fetching concept to be more intelligible in term of mobile context.	– Needs to be paired with Web caching technology in order to produce faster performance.
Pallis et al. (2007) [9]	– Proposed clustering method between object that needs to be fetched and user position.	– Needs more study on implementing another clustering algorithm.
Xu et al. (2006) [19]	– Proposed cooperative cache organizations between client and proxy cache.	– Needs more study on searching nearby proxy when some objects are requested.
Teng et al. (2005) [20]	– Proposed an algorithm that is called IWCP (Integration of Web Caching and Pre-fetching). – To create appropriate combination of pre-fetching and Web caching by utilizing innovative cache replacement.	– Still hard to be implemented in different platform, such as Web and mobile.

Tab. I *The comparison of previous work on pre-fetching and Web caching.*

class. Data items are defined by the values of their attributes and X is the vector of their values $\{x_1, x_2 \ \ldots . x_n\}$, where the value is either numeric or nominal. Attribute space is defined as a set containing all possible attribute vectors and is denoted by Z. Thus X is an element of Z ($X \in Z$). The set of all classes is denoted by $C = \{c_1, c_2, ..., c_n\}$. A classifier assigns a class $c \in C$ to every attribute of the vector $X \in Z$. The classifier can be considered as a mapping f, where f: $X\Diamond \quad C$. The use of this classifier is to classify the unseen data with a class label. A decision tree classifies the given data item using the values of its attributes. The decision tree is initially constructed from a set of pre-classified data. Each data item is defined by values of the attributes. The main issue is to select the attributes which best divides the data items into their classes. According to the values of these attributes, the data items are partitioned. This process is recursively applied to each partitioned subset of the data items. The process terminates when all the data items in the current subset belongs to the same class.

A decision tree consists of nodes, leaves and edges. A node of a decision tree specifies an attribute by which the data are to be partitioned. Each node has a number of edges, which are labeled according to a possible value of edges and a possible value of the attribute in the parent node. An edge connects either two nodes or a node and a leaf. Leaves are labeled with a decision value for categorization of the data. Induction of the decision tree uses the training data, which is described in terms of the attributes. The main problem here is on how to decide the attribute, which is the best partition of the data into various classes.

The ID3 algorithm uses the information theoretic method to solve this problem. Information theory uses the concept entropy, which measures the impurity of data items. Entropy specifies the number of bits required to encode the classification of a data item. The value of entropy is small when the class distribution is uneven, that is when all the data items belong to one class. The entropy value is higher when the class distribution is more even, that is when the data items have more classes. Information gain is a measure or the utility of each attribute in classifying the data items. It is measured using the entropy value. Information gain measures the decrease of the weighted average impurity (entropy) of the attributes compared with the impurity of the complete set of data items. Therefore, the attributes with the largest information gain are considered as the most useful attributes in classifying the data items.

To classify an unknown object, one starts at the root of the decision tree and follows the branch indicated by the outcome of each test until a leaf node is reached. The name of the class at the leaf node is the resulting classification. Several algorithms have implemented decision tree induction. This includes the basic ID3 that has the extension of C4.5 and C5.0. C4.5. C4.5 handles continuous attributes and is able to choose an appropriate attribute selection measure. It also deals with missing attribute values and improves computation efficiency. C4.5 builds the tree from a set of data items using the best attribute to test in order to divide the data item into subsets and then it uses the same procedure on each subset recursively. The selection of the best attribute to divide the subset at each stage uses the information gain of the attributes. For nominal valued attributes, a branch for each value of the attribute is formed, whereas for numeric valued attributes, a threshold is found, thus forming two branches.

3.1.1 CART methodology

The CART methodology [23] is recognized as a binary recursive partitioning. The process is binary due to splitting of parent nodes into exactly two child nodes. It is recursive because the process is repeatable by treating each child node as a parent. The key elements of a CART analysis include a set of rules for splitting each node in a tree; deciding when a tree is complete; and assigning each terminal node to a class outcome. CART always asks questions that have a 'yes' or 'no' answer to split a node into two child nodes; the *yes* answers to the left child node and the *no* answers to the right child node. CART's method is to look at all possible splits for all variables included in the analysis. Next, CART ranks the order of each splitting rule based on a quality-of-split criterion. One criterion commonly used is a measure of how well the splitting rule separates the classes contained in the parent node. Once it finds the best split, CART repeats the search process for each child node, continuously and recursively until further splitting is impossible or stopped. Splitting is impossible if only one case remains in a particular node or if all the cases in that node are exact copies of each other (on predictor variables). CART also allows splitting to stop for several other reasons, including that a node has too few cases. One simple criterion is the plurality rule: the group with the greatest representation determines the class assignment.

CART goes a step further: Because each node has the potential for being a terminal node, a class assignment is made for every node whether it is terminal or not. The rules of class assignment can be modified from simple plurality to account for the costs of making a mistake in classification and to adjust for over- or under-sampling from certain classes. A common technique among the first generation of tree classifiers is to continue splitting nodes (growing the tree) until some goodness-of-split criterion failed to be met. When the quality of a particular split fell below a certain threshold, the tree is not grown further along that branch. When all branches from the root reach terminal nodes, the tree is considered complete.

Instead of attempting to decide whether a given node is terminal or not, CART proceeds by growing trees until it is not possible to grow any further. Once CART has generated a maximal tree, it examines smaller trees obtained by pruning away branches of the maximal tree. The important point is that CART trees are always grown larger than they need to be and then selectively pruned back. Unlike other methods, CART does not stop in the middle of the tree-growing process, because there might still be important information to be discovered by drilling down several more levels.

Once the maximal tree is grown and a set of sub-trees are derived from it, CART determines the best tree by testing for error rates or costs. With sufficient data, the simplest method is to divide the sample into learning and test sub-samples. The learning sample is used to grow an overly large tree. Then use the test sample to estimate the rate at which cases are misclassified (possibly adjusted by misclassification costs). The misclassification error rate is calculated for the largest tree and also for every sub-tree. The best sub-tree is the one with the lowest or near-lowest cost, which may be a relatively small tree.

The cross-validation method is used if the data is insufficient for a separate test sample. In such cases, CART grows a maximal tree on the *entire* learning sample. This is the tree that will be pruned back. This process is used to estimate the independent predictive accuracy of the tree. This means that we can know how well any tree will perform on completely fresh data even if we do not have an independent test sample. Because the conventional methods of assessing tree accuracy can be wildly optimistic, cross validation is the method CART normally uses to obtain objective measures for smaller data sets.

3.2 Multivariate adaptive regression splines

Splines can be an innovative mathematical process for complicated curve drawings and function approximation. Splines find ever-increasing application in the numerical methods, computer-aided design, and computer graphics areas. Mathematical formulas for circles, parabolas, or sine waves are easy to construct. In order to develop a formula to trace the shape of shared value fluctuations or any time series prediction is to break the complex shape into simpler pieces, and then use a stock formula for each piece. To develop a spline, the X-axis is broken into a convenient number of regions. The boundary between regions is known as a knot. With a sufficiently large number of knots virtually any shape can be well approximated. While it is easy to draw a spline in two dimensions by keying on knot locations (approximation using linear, quadratic or cubic polynomial regression etc.), manipulating the mathematics in higher dimensions is best accomplished using basis functions.

The Multivariate Adaptive Regression Splines (MARS) model is a spline regression model that uses a specific class of basis functions as predictors in place of the original data [25, 26]. The MARS basis function transform makes it possible to selectively blank out certain regions of a variable by making them zero, allowing MARS to focus on specific sub-regions of the data. MARS excels at finding optimal variable transformations and interactions, as well as the complex data structure that often hides in high-dimensional data.

Given the number of predictors in most data mining applications, it is infeasible to approximate a function $y = f(x)$ in a generalization of splines by summarizing y in each distinct region of x. Even if we could assume that each predictor x had only two distinct regions, a database with just 35 predictors contains 235 or more than 34 billion regions. This is known as the curse of dimensionality. For some variables, two regions may not be enough to track the specifics of the function. If the relationship of y to some xs is different in three or four regions, for example, with only 35 variables, the number of regions requiring examination can be even larger than 34 billion. Given that, neither the number of regions nor the knot locations can be specified as a priori and a procedure [26] that are needed to accomplish judicious selection of which regions to look at and their boundaries, and judicious determination of how many intervals are needed for each variable.

A successful method of region selection requires to be adaptive to the characteristics of the data. Such solution probably rejects quite a few variables besides the accomplishing of the selected variable. This solution takes into account only a few variables at a time, thus reducing the number of regions.

3.2.1 MARS smoothing, splines, knots selection and basis functions

A key concept underlying the spline is the knot, which marks the end of one region of data and the beginning of another. Thus, the knot is where the behavior of the function changes. Between knots, the model could be global (e.g., linear regression). In a classical spline, the knots are predetermined and evenly spaced, whereas in

MARS, the knots are determined by a search procedure. Only as many knots as needed are included in a MARS model. If a straight line is a good fit, there will be no interior knots. In MARS, however, there is always at least one "pseudo" knot that corresponds to the smallest observed value of the predictor.

Finding one best knot in a simple regression is a straightforward search problem: Simply examine a large number of potential knots and choose the one with the best R2. However, finding the best pair of knots requires far more computation, and finding the best set of knots when the actual number needed is unknown is an even more challenging task. MARS finds the location and number of needed knots in a forward/backward stepwise fashion. First, a model that is clearly overfit with too many knots is generated; then, those knots that contribute least to the overall fit are removed. Thus, the forward knot selection will include many incorrect knot locations, but these erroneous knots should eventually be deleted from the model in the backwards pruning step (although this is not guaranteed).

Knot selection works very well to illustrate splines in one dimension; however, this context is unwieldy for working with a large number of variables simultaneously. Both concise notation and easy to manipulate programming expressions are required. It is also not clear how to construct or represent interactions using knot locations. In MARS, Basis Functions (BFs) are the machinery used for generalizing the search for knots. BFs are a set of functions used to represent the information contained in one or more variables. Much like principal components, BFs essentially re-express the relationship of the predictor variables with the target variable. The hockey stick BF, which is the core building block of the MARS model, is often applied to a single variable multiple times. The hockey stick function maps variable X to new variable X^*:

max $(0, X - c)$, or

max $(0, c - X)$.

In the first form, X^* is set to 0 for all values of X up to some threshold value c and X^* is equal to X for all values of X greater than c. (Actually X^* is equal to the amount by which X exceeds threshold c.) The second form generates a mirror image of the first. It starts with a constant in the model and then begins the search for a variable-knot combination that improves the model the most (or, alternatively, worsens the model the least). The measurement of the improvement is part of the change in Mean Squared Error (MSE). Adding a basis function always reduces the MSE. MARS searches a pair of hockey stick basis functions, the primary and mirror image, even though only one might be linearly independent of the other functions. This search is then repeated, with MARS searching for the best variable to add given the basis functions that are already in the model. The brute search process theoretically continues until the addition of every possible basis function to the model.

In practice, the user specifies an upper limit to generate the number of knots in the forward stage. The limit should be large enough to ensure the capture of the true model. The rule of thumb in determining the minimum number is three to four times the number of basis functions in the optimal model. The setting of this limit can be trial and error.

3.3 Random Forest

Breiman [27] proposed the Random Forests (RF) algorithm in 1999 [27]. The algorithm is usable for both regression and classification, as well as for variable selection, interaction detection, clustering etc. This technology represents a substantial advance in data mining technology and it is based on novel ways of combining information from a number of decision trees [23, 27].

A Decision Tree Forest (DTF) is an ensemble (collection) of decision trees, in which the combination of predictions contributes to the overall prediction for the forest. A decision tree forest grows a number of independent trees in parallel, and those trees do not interact until after all of them have been built. Decision tree forest models often have a degree of accuracy that cannot be obtained using a large, single-tree model. An outline of the algorithm used to construct a decision tree forest consisting of N observations as below:

(1) Take a random sample of N observations from the data set with replacement. The selection of some observations is more than once, and there is no selection for others. On average, about 2/3 of the rows will be selected by the sampling. The remaining 1/3 of the rows are called the out of bag rows. A new random selection of rows is performed for each tree constructed.

(2) As the tree is built, allow only a subset of the total set of predictor variables to be considered as possible splitters for each node. Select the set of predictors to be considered as a random subset of the total set of available predictors. For example, if there are ten predictors, choose a random five as candidate splitters. Perform a new random selection for each split. Some predictors (possibly the best one) will not be considered for each split, but a predictor excluded from one split may be used for another split in the same tree.

(3) and (2) are repeated for a large number of times to construct a forest of trees.

Decision tree forests have two stochastic elements: (1) the selection of data rows used as input for each tree, and (2) the set of predictor variables considered as candidates for each node split. For reasons that are not well understood, these randomizations along with combining the predictions from the trees significantly improve the overall predictive accuracy.

3.4 TreeNet

TreeNet (TN) is a robust multi-tree technology for data mining, predictive modeling and data processing. This technology is an exclusive implementation of Jerome Friedman's MART methodology [28]. It offers exceptional accuracy, blazing speed, and a high degree of fault tolerance for dirty and incomplete data. It can handle both classification and regression problems and has been proven to be remarkably effective in traditional numeric data mining and text mining [28].

TN is an enhancement of the CART model using stochastic gradient boosting [28]. Boosting means the endeavors to "boost" the accuracy of any given learning algorithm by fitting a series of models each having a low error rate, and then combining into an ensemble that may perform better [29, 30]. The key features of

TN models consist of [31]: automatic variable subset selection; ability to handle data without pre-processing; resistance to outliers; automatic handling of missing values; robustness to dirty and partially inaccurate data; high speed; and resistance to over-training. A TN model can be thought as a series expansion approximating the true functional relationship as shown in Equation 1 [29]:

$$F(X) = F_0 + \beta_1 T_1(X) + \beta_2 T_2(X) + \ldots + \beta_M T_M(X), \tag{1}$$

where T_i is a small tree. Each tree refines and improves its predecessors. TN models are typically composed of hundreds of small trees, each of which contributes a slight refinement to the overall model.

4. Experimental Setup

In this study, two different log records are utilized; browser logs data and server logs data. The first data from Boston University (BU) Web Trace (client-side or browser) [32] collected by Oceans Research Group at BU functions as the experiment data set. BU log data consists of 9,633 files, from 762 different users, and recording 109,759 requests for data transfer. The browser logs data (from November 1994 to May 1995) are obtained from Mosaic clients at BU (http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html) [33, 34, 35]. BU datasets consist of various activities among users, for example the access from different workstations that can cause multi Web log behavior. Hence, many uncertainties occur in this dataset, which can affect the findings of the study.

The second data are from the E-Learning@UTM (EL) Web server from Universiti Teknologi Malaysia (UTM). The server logs data that are obtained on 13 and 14 January 2008 with 65,015 record from one of EL Apache servers at Centre of Information and Communication Technology (CICT), UTM (http://elearning.utm.my/) [36]. Unlike BU datasets, EL datasets illustrate high similarity in Web log behavior and activities since these datasets are meant for E-Learning Website (E-Learning@UTM).

EL involved 17 faculties consisted of more than 2000 subjects who are undergraduates and postgraduates at UTM. Tab. II shows the number of hits for only an undergraduate subject that is Web Programming, in Semester 1, Session 2006/2007 (from July to November 2006). The data show the highest hit in the course module (31.20%), followed by forum (16.87%), resource (16.57%) and workshop (13.93%). Other modules are below 10% of the total hits. However, the log data for that session are not available, as the CICT administrator did not keep the previous log. Consequently, only the latest log data are available to be saved at that time.

4.1 Pre-processing and normalized data

Pre-processing is the key component to classify the object to cache. Fig. 3 shows the actual data prior to data pre-processing and Fig. 4 depicts the pre-process data. Each line of a condensed log in BU Web Trace corresponds to a single URL requested by the user; it contains the machine name, the time stamp when the

Module	**Hits**	**Percentage**
assignment	2,640	9.60
blog	37	0.13
choice	12	0.04
course	8,582	31.20
forum	4,641	16.87
label	17	0.06
quiz	1,343	4.88
resource	4,557	16.57
upload	813	2.96
user	1,032	3.75
workshop	3,830	13.93
Total	**27,504**	100

Tab. II *Number of hits and percentage for each module in Web programming subject.*

request is made (seconds and microseconds), the URL, the size of the document (in bytes), and the object retrieval time in seconds [32].

Meanwhile, each line in the EL file represents an incoming HTTP request, and Apache records information about it using a format known as the Common Log Format (CLF). Reading from left to right, this format contains the following information about the request; the source IP address, the client's identity, the remote user name (if using HTTP authentication), the date, time, and time zone of the request, the actual content of the request, the server's response code to the request, and the size of the data block returned to the client in bytes.

Three common variables or attributes have been identified in Web performance analysis [37, 38]. The attributes used in this study are:

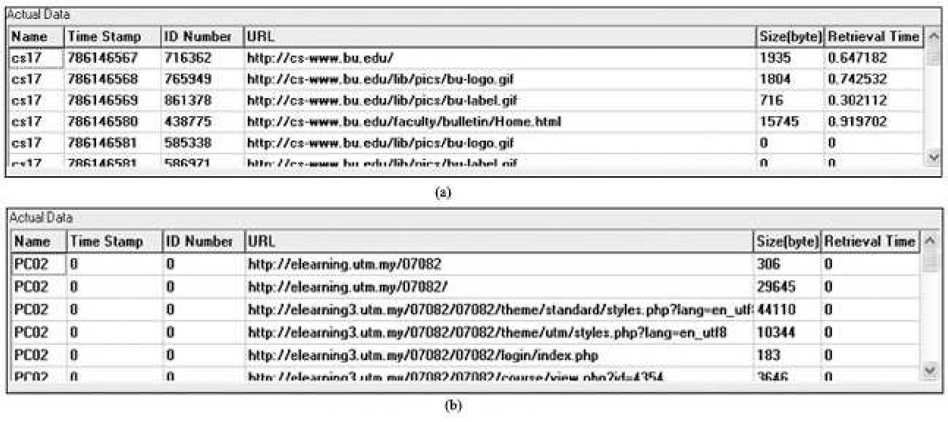
Time: *the counter that observes the time taken to receive data in seconds.*
Object Size: *the size is in bytes.*
Numbers of Hit: *the number of hits per data. Each completed request for a Web file increases the number of hit for the requested file.*

Each variable or attribute must be multiplied with defined Priority Value (PV) [39] to get the total of the attributes for target output generation of the network. An example is as follows:

$$\text{Expected target } = (\text{size} * 0.266667) + (\text{hit} * 0.200000) + (\text{retrieval time} * 0.066667) \qquad (2)$$

The total value determines the expected target for current data. It is compared to a threshold number, and these threshold values are dynamic. Koskela [40] proposed a threshold calculation based on the latency ratio on a request hit rate obtained for a cache with an infinite size.

The threshold is calculated and updated for every epoch of the training. If the *expected_target* is smaller than the threshold, then the expected target is 0, or else



Actual Data

Name	Time Stamp	ID Number	URL	Size(byte)	Retrieval Time
cs17	786146567	716362	http://cs-www.bu.edu/	1935	0.647182
cs17	786146568	765949	http://cs-www.bu.edu/lib/pics/bu-logo.gif	1804	0.742532
cs17	786146569	861378	http://cs-www.bu.edu/lib/pics/bu-label.gif	716	0.302112
cs17	786146580	438775	http://cs-www.bu.edu/faculty/bulletin/Home.html	15745	0.919702
cs17	786146581	585338	http://cs-www.bu.edu/lib/pics/bu-logo.gif	0	0
cs17	786146581	586971	http://cs-www.bu.edu/lib/pics/bu-label.gif	0	0

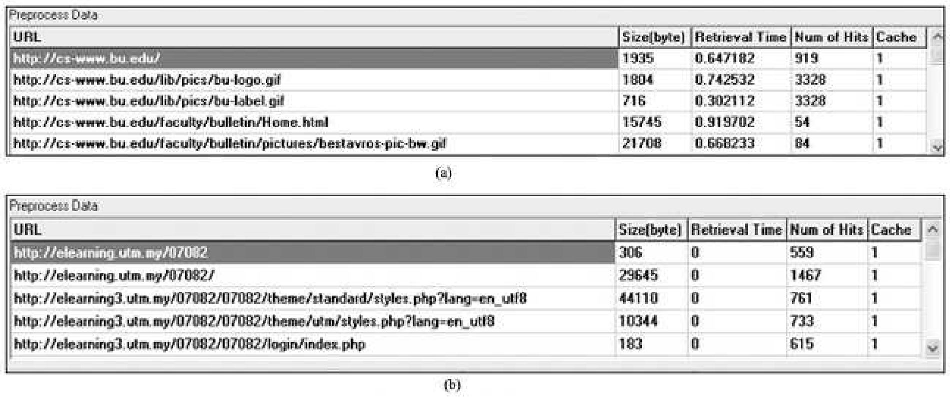
(a)

Actual Data

Name	Time Stamp	ID Number	URL	Size(byte)	Retrieval Time
PC02	0	0	http://elearning.utm.my/07082	306	0
PC02	0	0	http://elearning.utm.my/07082/	29645	0
PC02	0	0	http://elearning3.utm.my/07082/07082/theme/standard/styles.php?lang=en_utf	44110	0
PC02	0	0	http://elearning3.utm.my/07082/07082/theme/utm/styles.php?lang=en_utf8	10344	0
PC02	0	0	http://elearning3.utm.my/07082/07082/login/index.php	183	0
PC02	0	0	http://elearning3.utm.my/07082/07082/course/view.php?id=4354	3646	0

(b)

Fig. 3 *Examples of data from BU and EL log data: (a) BU (b) EL.*



Preprocess Data

URL	Size(byte)	Retrieval Time	Num of Hits	Cache
http://cs-www.bu.edu/	1935	0.647182	919	1
http://cs-www.bu.edu/lib/pics/bu-logo.gif	1804	0.742532	3328	1
http://cs-www.bu.edu/lib/pics/bu-label.gif	716	0.302112	3328	1
http://cs-www.bu.edu/faculty/bulletin/Home.html	15745	0.919702	54	1
http://cs-www.bu.edu/faculty/bulletin/pictures/bestavros-pic-bw.gif	21708	0.668233	84	1

(a)

Preprocess Data

URL	Size(byte)	Retrieval Time	Num of Hits	Cache
http://elearning.utm.my/07082	306	0	559	1
http://elearning.utm.my/07082/	29645	0	1467	1
http://elearning3.utm.my/07082/07082/theme/standard/styles.php?lang=en_utf8	44110	0	761	1
http://elearning3.utm.my/07082/07082/theme/utm/styles.php?lang=en_utf8	10344	0	733	1
http://elearning3.utm.my/07082/07082/login/index.php	183	0	615	1

(b)

Fig. 4 *Pre-process BU and EL log data: (a) BU (b) EL.*

it becomes 1 if the *expected_target* is equal or greater than that of the threshold [33, 36] shown below:

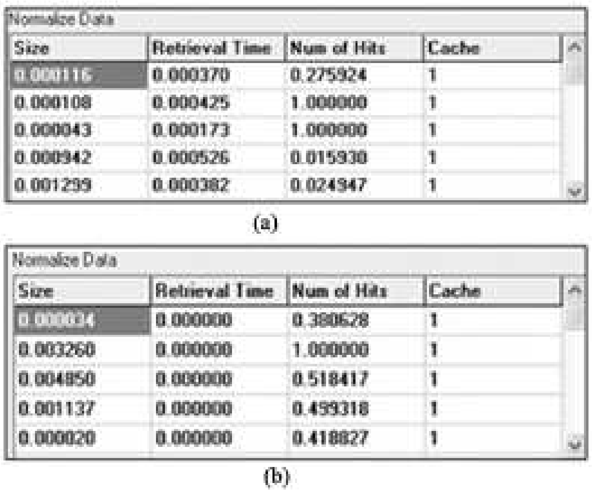
$$Expected\ Network\ Output\ = \begin{cases} 0 \text{ if expected\_target} < threshold, \\ 1 \text{ if expected\_target} \geq threshold. \end{cases} \quad (3)$$

The network incorporates simplicity in generating the output for the Web caching, to cache or not to cache. For each output generated from the non-training mode, the outputs can be illustrated by employing sigmoid function that is bounded between 0 and 1. For each output value that represents the values between the interval of 0.5 and 1, the data are cached in the caching storage, and for each output that represents values less than 0.5 the data are fetched directly from the originating database resource, in case the data are not found in the cache storage [33, 36, 40].

Normalization process (see Fig. 5) is done by determining the maximum and minimum value for each attribute. The end values are between 0 and 1 to improve training characteristics. Min-max normalization is given as in formula 4:

$$X* = \frac{X\text{- }\min(X)}{\max(X)\text{ - }\min(X)}. \tag{4}$$

Let X refer to an original attribute value and X^* refer to the normalized attribute value. From this formula, the summary of the results is as shown in Tab. III.



Normalize Data

Size	Retrieval Time	Num of Hits	Cache
0.000116	0.000370	0.275924	1
0.000108	0.000425	1.000000	1
0.000043	0.000173	1.000000	1
0.000942	0.000526	0.015930	1
0.001299	0.000382	0.024947	1

(a)

Normalize Data

Size	Retrieval Time	Num of Hits	Cache
0.000024	0.000000	0.380628	1
0.003260	0.000000	1.000000	1
0.004850	0.000000	0.518417	1
0.001137	0.000000	0.459318	1
0.000020	0.000000	0.418827	1

(b)

Fig. 5 *Normalize BU and EL log data (a) BU (b) EL.*

Summary	**BU**	**EL**
Number of actual data	109 759	54 605
Number of pre-process data	17224	23 105
Maximum size (byte)	16 384 015	16 384 015
Longest retrieval time (seconds)	1 749	0
Highest hits	3 328	1 141

Tab. III *Summarization of BU and EL log data after the normalization.*

4.2 Training and testing

The actual BU log data consist of 109,759 records and EL log data involve 65,015 records. 90% of it is used for training and the remaining is for testing purposes. K-folds cross validation as a statistical technique is used to validate the performance of the machine learning methods for both Web logs dataset. These experiments are carried out on a Core Duo CPU, 2GHz Machine and the codes are executed using Salford System tools. The details setting of CART, MARS, RF and TN using the BU and EL datasets are shown in Tab. IV.

Parameter Setting	**BU**	**EL**
CART		
Number of predictors	3	2
Important	3	2
Nodes	31	5
Min node cases	1	328-374
MARS		
Number of predictors	3	2
Max basis functions	12	8
Number of effective parameters	1	13-15
Min observation between knots	100	100
RF		
Number of predictors	3	2
Max terminal nodes	7752	10398
Trees grown	500	500
Parent node min cases	1	2
TN		
Number of predictors	3	2
Tree size	6	6
Tree grown	200	200
Last tree data fraction	0	0

Tab. IV *Parameter setting for CART, MARS, RF and TN for BU and EL log data.*

5. Results and Analysis

Tabs. V and VI summarize the comparison of performances for CART, MARS, RF and TN in terms of training and test error rate including the accuracy of classification for BU and EL log data. Mean and standard deviation are used as a statistical validation to verify the performance of CART, MARS, RF, and TN for both datasets.

The results have revealed that CART has the lowest mean error rate for BU training (0.00191), test set (0.00721) and 0 for both EL training and test set. The second lowest in error rate for BU training set is 0.00264 for TN. Nonetheless, the highest error rate is 0.05896 for training set and 0.05932 for test set by using MARS for BU dataset. As can be seen, Tab. VI reports TN for EL training and test set as the highest error rate (0.45). Tab. V also explains that a standard deviation of CART for both EL training and test set has the lowest value, 0 for both EL training and test set compared with other statistical models.

At the same time, the testing process is done to determine the accuracy of the output generated by all machine learning classifiers. The accuracy is executed, based on the difference in results between the actual value and the generated value by the CART, MARS, RF, and TN. In this research, the accuracy is measured as

Fold	Error Rate of Training and Test Set for BU							
	Training Set				*Test Set*			
	CART	***MARS***	***RF***	***TN***	***CART***	***MARS***	***RF***	***TN***
1	0.00100	0.05941	0.00880	0.00460	0.00030	0.04678	0.00060	0.00030
2	0.00395	0.05960	0.00870	0.00110	0.01315	0.04594	0.01820	0.01395
3	0.00080	0.05911	0.00805	0.00075	0.01295	0.06286	0.01525	0.01230
4	0.00080	0.05839	0.00380	0.00035	0.01265	0.06786	0.01625	0.01500
5	0.00210	0.05940	0.00855	0.01110	0.00220	0.05285	0.00065	0.00095
6	0.00105	0.05880	0.00830	0.00305	0.00160	0.05619	0.00345	0.00345
7	0.00115	0.05804	0.00920	0.00075	0.01650	0.06931	0.05865	0.03640
8	0.00280	0.05934	0.00725	0.00080	0.00385	0.06087	0.00710	0.00095
9	0.00420	0.05990	0.00940	0.00235	0.00660	0.05580	0.00345	0.00690
10	0.00120	0.05765	0.00870	0.00155	0.00230	0.07477	0.00065	0.00345
Mean	**0.00191**	0.05896	0.00808	0.00264	**0.00721**	0.05932	0.01243	0.00937
Standard Deviation	0.00131	0.00073	0.00162	0.00325	0.00600	0.00958	0.01764	0.01102

Tab. V *Mean and standard deviation for training and test set for BU dataset.*

Fold	Error Rate of Training and Test Set for EL							
	Training Set				*Test Set*			
	CART	***MARS***	***RF***	***TN***	***CART***	***MARS***	***RF***	***TN***
1	0.00000	0.07615	0.00025	0.50000	0.00000	0.08060	0.00035	0.50000
2	0.00000	0.08241	0.00000	0.50000	0.00000	0.09516	0.00000	0.50000
3	0.00000	0.08383	0.00010	0.00000	0.00000	0.08049	0.00000	0.00000
4	0.00000	0.07534	0.00035	0.50000	0.00000	0.06890	0.00000	0.50000
5	0.00000	0.08265	0.00005	0.50000	0.00000	0.08708	0.00000	0.50000
6	0.00000	0.08394	0.00015	0.50000	0.00000	0.08140	0.00045	0.50000
7	0.00000	0.08445	0.00000	0.50000	0.00000	0.07008	0.00000	0.50000
8	0.00000	0.08582	0.00010	0.50000	0.00000	0.07048	0.00000	0.50000
9	0.00000	0.08111	0.00030	0.50000	0.00000	0.10824	0.00000	0.50000
10	0.00000	0.08300	0.00620	0.50000	0.00000	0.08390	0.00440	0.50000
Mean	**0.00000**	0.08187	0.00075	0.45000	**0.00000**	0.08263	0.00052	0.45000
Standard Deviation	0.00000	0.00347	0.00192	0.15811	0.00000	0.01219	0.00137	0.15811

Tab. VI *Mean and standard deviation for training and test set for EL dataset.*

shown in (5):

$$Accuracy = \frac{\text{Number of correct data}}{\text{Total data}} \times 100\%. \tag{5}$$

Based on Equation 5, the CART accuracy is 99.81% and 100% for BU and EL training dataset, respectively, which has the best accuracy compared to other classifiers. The second highest is TN, 99.74% for BU training dataset and RF, 99.93% for EL training dataset. This depicts that the CART model can classify better (refer to Tabs. VII and VIII) with less nodes than others except MARS (see Tab. IX). This condition occurred and might affect the time to generate the final

Fold	**Accuracy of Training and Test Set for BU**							
	Training Set				***Test Set***			
	CART	***MARS***	***RF***	***TN***	***CART***	***MARS***	***RF***	***TN***
1	99.90	66.93	99.12	99.54	99.97	76.00	99.94	99.97
2	99.61	71.43	99.13	99.89	98.69	72.98	98.18	98.61
3	99.92	68.85	99.19	99.93	98.71	65.07	98.47	98.77
4	99.92	70.03	99.62	99.96	98.74	71.00	98.37	98.50
5	99.79	71.07	99.14	98.89	99.78	68.48	99.93	99.91
6	99.89	71.53	99.17	99.69	99.84	70.31	99.65	99.66
7	99.88	69.24	99.08	99.93	98.35	69.23	94.13	96.36
8	99.72	69.73	99.27	99.92	99.62	70.12	99.29	99.90
9	99.58	69.10	99.06	99.76	99.34	74.25	99.65	99.31
10	99.88	71.81	99.13	99.84	99.77	71.26	99.93	99.66
Mean	**99.81**	69.97	99.19	99.74	**99.28**	70.87	98.75	99.07
Standard Deviation	0.12905	1.52781	0.16169	0.32500	0.59893	3.07824	1.76397	1.10195

Tab. VII *Mean and standard deviation for accuracy of training and test set for fold 1 to 10 BU dataset.*

Fold	**Accuracy of Training and Test Set for EL**							
	Training Set				***Test Set***			
	CART	***MARS***	***RF***	***TN***	***CART***	***MARS***	***RF***	***TN***
1	100.00	96.05	99.98	50.00	100.00	98.20	99.97	50.00
2	100.00	63.83	100.00	50.00	100.00	66.92	100.00	50.00
3	100.00	63.96	99.99	100.00	100.00	66.30	100.00	100.00
4	100.00	63.31	99.97	50.00	100.00	63.49	100.00	50.00
5	100.00	64.24	100.00	50.00	100.00	62.67	100.00	50.00
6	100.00	64.21	99.99	50.00	100.00	64.13	99.96	50.00
7	100.00	63.92	100.00	50.00	100.00	64.70	100.00	50.00
8	100.00	65.03	99.99	50.00	100.00	82.13	100.00	50.00
9	100.00	64.69	99.97	50.00	100.00	61.80	100.00	50.00
10	100.00	64.24	99.38	50.00	100.00	64.00	99.56	50.00
Mean	**100.00**	67.35	99.93	55.00	**100.00**	69.43	99.95	55.00
Standard Deviation	0.00000	10.09577	0.19253	15.81139	0.00000	11.66307	0.13747	15.81139

Tab. VIII *Mean and standard deviation for accuracy of training and test set for fold 1 to 10 EL dataset.*

Classifier	Nodes/Basis Functions/Trees Grown	
	BU	***EL***
CART	30-37	5
MARS	9	4-6
RF	500	500
TN	200	200

Tab. IX *Number of nodes, basis functions, and tree grown for both the training and test set for fold 1 to 10 BU and EL log dataset.*

classification, to cache or not to cache the Web objects, which is faster by using CART as compared to other classifiers.

Moreover, Tab. X reports the receiver operating characteristic (ROC) as a graphical plot of the confusion matrix for a two-class classifier, either to cache-1 or not to cache-0. ROC analysis has recently been introduced in various fields like medicine, radiology and others. Conversely, it has been introduced recently in other areas, for instance data mining and machine learning. ROC approved that CART and TN for EL dataset is the best classifier compared to other machine learning classifiers, and RF is the best classifier for BU dataset.

In addition, Tab. XI shows the important level of three variables based on the means of each variable for a fold 1 to 10 of BU log data. Nonetheless, the EL log data only provided the size and the number of hits variable. The most significant variable is the size, followed by the number of hits and retrieval time for each BU and EL log data. It has been proven that size constructs an effective use of space for Web caching in the cache server.

Previous results in Tabs. VII and VIII show that CART classifier is one of the best classifiers of Web cache objects especially for the EL dataset. Due to the earlier analysis by using paired-samples t-test, the significant analysis for CART classifier is evaluated. The results of evaluation are available in Tabs. XII and XIII. Positive means revealed that CART is significantly better than all other classifiers for BU and EL datasets. As evident in the same table, the accuracy criterion for CART is significantly better than all other classifiers with a p-value < 0.05 excluding TN for both BU training and test datasets and RF for BU and EL test datasets.

6. Discussions

Various methodology and approaches to manage proxy cache have been proposed [15-20]. This particular study applied different computational models to decide and classify the objects on Web documents and to either cache or not to cache the objects.

The results imply that CART and TN have a distinct advantage over MARS and RF in classifying the cache objects. This scenario occurred due to the strengths and weaknesses of the models themselves (refer to Tab. XIV).

Classifier	Mean of ROC for Fold 1 to 10							
	Training Set				***Test Set***			
	BU		***EL***		***BU***		***EL***	
	Class1(0)	*Class2(1)*	*Class1(0)*	*Class2(1)*	*Class1(0)*	*Class2(1)*	*Class1(0)*	*Class2(1)*
CART	**0.99941**	**0.99941**	**1.00000**	**1.00000**	0.99421	0.99421	**1.00000**	**1.00000**
MARS	0.98917	0.98917	0.99992	0.99994	0.98900	0.98900	0.99992	0.99994
RF	0.99929	0.99958	0.99284	0.99998	**0.99986**	**0.99984**	0.99999	**1.00000**
TN	0.99509	0.97901	**1.00000**	**1.00000**	0.99344	0.96467	**1.00000**	**1.00000**

Tab. X *Mean of ROC comparison for fold 1 to 10 BU and EL log data.*

Variable	Mean of Important Variable for Fold 1 to 10							
	BU				***EL***			
	CART	***MARS***	***RF***	***TN***	***CART***	***MARS***	***RF***	***TN***
SIZE	**100.00**	93.66	**100.00**	80.66	**100.00**	**100.00**	**100.00**	**100.00**
NUM_OF_HITS	86.32	99.20	83.07	86.34	26.67	51.37	28.01	86.34
RETRIEVAL_ TIME	41.27	49.11	35.64	70.85	-	-	-	-

Tab. XI *Mean of important variables for BU and EL log data.*

Paired Samples	Paired-samples T-test Result for BU									
	Training Set					*Test Set*				
	Mean	*SD*	*t*	*df*	*p-value*	*Mean*	*SD*	*T*	*df*	*p-value*
CART - MARS	**29.83700**	1.55341	60.739	9	0.000	**28.41100**	2.92758	30.689	9	0.000
CART - RF	0.61800	0.16956	11.526	9	0.000	0.52700	1.32396	1.259	9	0.240
CART - TN	0.07400	0.34664	0.675	9	0.517	0.21600	0.64134	1.065	9	0.315

Tab. XII *Paired-samples t-test for accuracy of BU training and test set.*

Paired	Paired-samples T-test Result for EL									
	Training Set					*Test Set*				
	Mean	*SD*	*t*	*df*	*p-value*	*Mean*	*SD*	*t*	*df*	*p-value*
CART - MARS	32.65200	10.09577	10.228	9	0.000	30.56600	11.66307	8.288	9	0.000
CART - RF	0.07300	0.19253	1.199	9	0.261	0.05100	0.13747	1.173	9	0.271
CART-TN	**45.00000**	15.81139	9.000	9	0.000	**45.00000**	15.81139	9.000	9	0.000

Tab. XIII *Paired-samples t-test for accuracy of EL training and test set.*

Classifier	Method	Strengths	Weaknesses
CART	Uses a decision tree to display how data may be classified or predicted. Automatically searches for important relationships and uncovers hidden structure even in highly complex data.	Selection of predictors, scalability, excellent ability to deal with interactions, non-linearity and local basis functions. Fast, recursive binary splitting, backward pruning algorithm. Handles missing values easily via surrogates. Training scales up well and testing is very fast. Always produces model summary. Invariant to monotonic predictor variable transformations that give the same tree model. High degree of resistance to outliers and irrelevant predictors. Provided relative variable importance and ease of use with a few tuneable parameters.	Lack of smoothness, difficulty in capturing linear and additive structure, unstable and low interpretability in this context. Tree models are regionally constant within a given region. Leads to potential instability where small changes in predictors lead to dramatic changes in predictions and instability likely worse for larger as well. Small datasets at real disadvantage.
TN	Depends on linear combination of many small trees, each typically containing about six terminal nodes. Each tree is devoted to contributing a small portion of the overall model and the final model for prediction is constructed by adding up all of the individual tree contributions.	Maintains all of the strengths of single CART tree except interpretability, scales well to large problems, often dramatic increase in accuracy, much more stable by averaging large number of trees and degree of data fragmentation reduced by using small trees.	Poor for outlier cases and does not have a significant tendency towards either underestimation or overestimation.
RF	Creates multiple boot-strapped regression trees without pruning and averages the outputs. Each tree is grown with a randomized subset of predictors. Typically, 500 to 2,000 trees are grown and the results aggregated by averaging.	Growing large numbers of trees does not overfit the data, and random predictor selection keeps bias low. Provides better classifier for prediction.	Even more of a “black box” model. Can be very demanding in terms of time and computer resources.
MARS	Builds localized regression models by fitting separate splines using basis functions to distinct intervals of predictor variables.	Because splitting rules are replaced by continuous smooth functions, MARS is better at detecting global and linear data structure. Moreover, the output is smoother and less coarse-grained.	Tends to be excessively guided by the local nature of the data, making prediction with new data very unstable. Selecting values for input. Parameters can be cumbersome.

Tab. XIV *Comparison of the four classifiers.*

Subsequently, the modeling of the data set can be a suitability factor of the models. Based on the experimental results in this study, the summary is as below:

- Intelligent Web caching is able to store ideal objects and remove unwanted objects, which may cause insufficient cache. Thus, the caching insufficiency can be improved.
- Both CART and TN achieve correct classification accuracy in the range of 99.8% to 100% for testing data of BU and EL log data, and in the range of 0 to 0.002 for training error rate data for both data compared to MARS and RF, respectively.
- ROC for CART has the highest sensitivity and specificity for testing. Consequently, CART is identified as the best classifier that is closest to the convex hull.
- In all conditions, MARS is the worst model to be applied in classifying all log data because MARS is highly sensitive to extrapolation caused by the local nature of the basis functions. A change in the predictor value towards the end of its range can cause the prediction to go largely off scale.
- Size of Web objects is recognized as the most important variable in affecting the performance of the proxy cache server.

7. Conclusions and Future Work

In this research, an accomplishment of different machine learning methods for Web caching technology promises alleviation of congestion in the Internet access mainly for BU and EL. Therefore, this study proves that the classification of Web objects through log mining using CART, MARS, RF and TN models can be applied in cache server. Hence, this situation can affect the size of data in the cache server and the time to retrieve the data from the cache server. Future work includes the evaluation and comparison of performance analysis for machine learning methods with other hybrid soft computing methods for Web caching technology using BU and EL datasets.

Acknowledgment

This work is supported by MOSTI and RMC, Universiti Teknologi Malaysia (UTM), MALAYSIA. The authors would like to express their deepest gratitude to the Boston University, Center of Information & Communication Technology and Center of Teaching & Learning, UTM especially Mr. Azmi Kamis, Assoc. Prof. Dr. Jamalludin Harun and Jo Ann Kelly (Salford Systems) for their collaboration in making this research a success.

References

[1] Ali W., Shamsuddin S. M.: Intelligent Client-side Web Caching Scheme Based on Least recently Used Algorithm and Neuro-Fuzzy System. The Sixth International Symposium on

Neural Networks (ISNN 2009), Lecture Notes in Computer Science (LNCS), Springer-Verlag Berlin Heidelberg, 5552, 2009, pp. 70–79.

[2] Tan Y., Ji Y., Mookerjee V. S.: Analyzing Document-Duplication Effects on Policies for Browser and Proxy Caching. INFORMS Journal on Computing, **18**, 4, 2006, pp. 506–522.

[3] Sieminski A.: Changeability of Web objects – browser perspective. Proceeding of the 5th International Conference on Intelligent Systems Design and Applications, 2005, pp. 476–48.

[4] Mookerjee V. S., Tan Y.: Analysis of a least recently used cache management policy for Web browsers. Operations Research, Linthicum, **50**, 2, 2002, pp. 345–357.

[5] Lu Y., Zhan B.: A Study for IE Browser Cache through Browser Tracing Tool, 2000. Retrieved December 15, 2007, from http://pages.cs.wisc.edu/∼remzi/Classes/736/Spring2000/Project-Writeups/YongBo.ps.

[6] Reddy M., Fletcher G. P.: An adaptive mechanism for Web browser cache management. IEEE Internet Computing. IEEE Educational Activities Department Piscataway, NJ, USA, **2**, 1, 1998, pp. 78–81.

[7] Davison B. D.: Web Caching Overview. 2008. Available from http://www.Web caching.com/welcome.html.

[8] Davison B. D.: A Survey of Proxy Cache Evaluation Techniques. In: Proceedings of the Fourth International WWW Caching Workshop (WCW99), San Diego, CA, March 1999.

[9] Pallis G., Vakali A., Pokorny J.: A clustering-based prefetching scheme on a Web cache environment, Comput. Electr. Eng., **34**, 4, July 2008, pp. 309–323.

[10] Nair A. S., Jayasudha J. S.: Dynamic Web pre-fetching technique for latency reduction, In: Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007) – Volume 04 (ICCIMA '07), IEEE Computer Society, Washington, DC, USA, **4**, 2007, pp. 202–206.

[11] Hung S.-H., Wu C.-C., Tu C.-H.: Optimizing the embedded caching and prefetching software on a network-attached storage system. In: Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing – Volume 01 (EUC '08), IEEE Computer Society, Washington, DC, USA, **1**, 2008, pp. 152–161.

[12] Mahanti A., Williamson C., Eager D.: Traffic analysis of a Web proxy caching hierarchy. IEEE Network, **14**, 3, May/June 2000, pp. 16–23.

[13] Web Caching: Caching Tutorial for Web Authors. 2008. Available from http://www.web-caching.com/mnot_tutorial/intro.html.

[14] Kennedy J., Eberhart R. C.: Particle Swarm Optimization, Proc. IEEE Int'l Conf. on Neural Networks IV, Piscataway, 1995, pp. 1942–1948.

[15] Padmanabhan V. N., Mogul J. C.: Using Predictive Prefetching to Improve World-Wide Web Latency, ACM, SIGCOMM'96, 1996.

[16] Bestavros A., Cunha C.: A Prefetching Protocol Using Client Speculation for The WWW, Tech. Rep. TR-95-011, Boston University, Department of Computer Science, Boston, MA 02215, Apr. 1995.

[17] Kroeger T. M., Long D. E., Mogul J. C.: Exploring the bounds of web latency reduction from caching and prefetching. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems (USITS'97). USENIX Association, Berkeley, CA, USA, 1997, 2-2.

[18] Ye F., Li Q., Chen E.: Adaptive Caching with Heterogeneous Devices in Mobile Peer to Peer Network, SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil, ACM.

[19] Xu J., Liu J., Li B., Jia X.: Caching and Prefetching for Web Content Distribution, IEEE Computing in Science and Engineering Magazine: Special Issue on Web Engineering, 2006.

[20] Teng W. G., Chang C. Y., Chen M. S.: Integrating Web Caching and Web Prefetching in Client-Side Proxies, IEEE Transactions on Parallel and Distributed Systems, **16**, 5, May 2005.

[21] Lee J. H., Shiu W.-K.: An adaptive website system to improve efficiency with web mining techniques, Advanced Engineering Informatics, **18**, 2004, pp. 129–142.

[22] Salford System. Salford Systems – Data Mining Solutions. 2011. Available from http://www.salford-systems.com/

[23] Breiman L., Friedman J., Olshen R., Stone C.: Classification and Regression Trees, Chapman & Hall/CRC Press, Boca Raton, FL, 1984.

[24] Gey S., Nédélec E.: Model selection for CART regression trees. IEEE Trans. Inf. Theory **51**, 2005, pp. 658–670.

[25] Friedman J. H.: Multivariate adaptive regression splines (with discussion). Annals of Statistics, **19**, 1991, pp. 1–141.

[26] Abraham A., Steinberg D.: MARS: Still an Alien Planet in Soft Computing? In: International Conference on Computational Science (Proceedings), Springer, San Francisco, California, USA, Part II, **2**, 2001, pp. 235–244.

[27] Breiman L.: Random Forests, Kluwer Academic Publishers, Machine Learning, **45**, 2001, pp. 5–32.

[28] Friedman J. H.: Stochastic gradient boosting, Computational Statistics & Data Analysis, **38**, 2002, pp. 367–378.

[29] Elish M. O., Elish K. O.: Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study, Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference, 2009, pp. 69–78.

[30] Schapire R.: A Brief Introduction to Boosting, Proc. 16th International Joint Conference on Artificial Intelligence, 1999, pp. 1401–1405.

[31] Friedman J., Meulman J.: Multiple additive regression trees with application in epidemiology, Statistics in Medicine, **22**, 2003, pp. 1365–1381.

[32] Cunha C., Bestavros A., Crovella M.: Characteristics of WWW Client-Based Traces. Technical Report. UMI Order Number: 1995-010, Boston University, 1995.

[33] Sulaiman S., Shamsuddin S. M., Forkan F., Abraham A.: Intelligent Web Caching Using Neurocomputing and Particle Swarm Optimization Algorithm, Second Asia International Conference on Modeling and Simulation, AMS 2008, IEEE Computer Society Press, USA, 2008, pp. 642–647.

[34] Sulaiman S., Shamsuddin S. M., Abraham A.: Rough Web Caching, Rough Set Theory: A True Landmark in Data Analysis, Studies in Computational Intelligence, Springer Verlag, Germany, 2009, pp. 187–211.

[35] Sulaiman S., Shamsuddin S. M., Abraham A.: An Implementation of Rough Set in Optimizing Mobile Web Caching Performance, Tenth International Conference on Computer Modeling and Simulation, UKSiM/EUROSiM 2008, Cambridge, UK, IEEE Computer Society Press, USA, 2008, pp. 655–660.

[36] Sulaiman S., Shamsuddin S. M., Forkan F., Abraham A.: Intelligent Web Caching for E-Learning Log Data, Third Asia International Conference on Modelling and Simulation, AMS 2009, IEEE Computer Society Press, USA, 2009, pp. 136–141.

[37] Rousskov A., Soloviev V.: On Performance of Caching Proxies, Short version appears as poster paper in ACM SIGMETRIC'98 Conference, 1998.

[38] Liu M., Wang F. Y., Zeng D., Yang L.: An Overview of World Wide Web Caching, IEEE International Conference on Systems, Man, and Cybernetics, **5**, 2001, pp. 3045–3050.

[39] Mohamed F.: Intelligent Web Caching Architecture, Master thesis, Faculty of Computer Science and Information System, Universiti Teknologi Malaysia, 2007.

[40] Koskela T.: Neural Network Method in Analysing and Modelling Time Varying Processes, PhD dissertation, Helsinki University of Technology, 2004.

[41] Jin B., Tian S., Lin S., Ren X., Huang Y.: An Integrated Prefetching and Caching Scheme for Mobile Web Caching System, The 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, IEEE, 2007.

[42] Albers S.: New Results on Web Caching with Request Reordering, ALGORITHMICA (0178-4617), 2010, **58**, 2, pp. 461–477.

[43] Ali Ahmed W., Shamsuddin S. M.: Neuro-fuzzy system in partitioned client-side Web cache, Expert Systems with Applications 38, 2011, pp. 14715–14725.