In the Proceedings of the Third International Conference on Development and Learning
(ICDL'04), La Jolla, California, October 20–22 2004.

MESO: Perceptual Memory to Support Online Learning in Adaptive Software*

E. P. Kasten and P. K. McKinley
Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{*kasten,mckinley*}*@cse.msu.edu*

Abstract

Adaptive and autonomic systems often must be able to detect and respond to errant behavior or changing conditions with little or no human intervention. Decision making is a critical issue in such systems, which must learn how and when to invoke corrective actions based on past experience. This paper describes the design, implementation and evaluation of a perceptual memory system, called MESO, that supports online decision-making in adaptive systems. MESO uses clustering and pattern classification methods while addressing the needs of online, incremental learning.

Keywords: adaptive software, decision making, imitative learning, machine learning, pattern classification, perceptual memory.

1 Introduction

Increasingly, software needs to adapt to dynamic external conditions involving hardware components, network connections, and changes in the surrounding physical environment [5, 10, 23]. For example, to meet the needs of mobile users, software integrated into hand-held, portable and wearable computing devices must balance several conflicting concerns, including quality-of-service, security, energy consumption, and user preferences. Moreover, the promise of autonomic computing systems [16], that enable software to dynamically self-heal and self-manage, appeals to system's administrators and users alike.

In adaptive applications, future decisions should benefit from past experience, helping to improve the fitness of the software with respect to its environment and/or function. However, learning from experience requires that a system remember appropriate responses to sensed environmental context. *Perceptual memory*, a type of long-term memory for remembering external stimulus patterns [7], plays an important role in supporting context-aware, adaptive software.

The main contribution of this paper is to present a perceptual memory system, called MESO[1], that applies pattern classification and clustering techniques [6] to online, dynamic learning systems. The benefits of MESO include: rapid incremental training, rapid reorganization of an existing classifier tree, high recall accuracy, lack of dependence on a priori knowledge of adaptive actions, and support for data compression. Each of these benefits is important to constructing a dynamic decision-maker. For instance, incremental training enables a system to learn over time, addressing changing user requirements or environments. Limiting the impact of a growing population of training patterns can be addressed using data compression, reducing the memory and processor requirements needed for managing large data sets. We show how MESO can be used to enable software decision-making by presenting preliminary results of experiments with an audio streaming application that can *imitatively learn* [1, 13] how to adapt to changing network conditions. Due to space limitations, many details of this study are omitted here, but can be found in the accompanying technical report [15].

2 Background and Related Work

An adaptive software system must include a decision-making component to realize adaptive behavior. Learning-based approaches [11, 22] show substantial promise for addressing the needs of decision makers. By observing its environment, an application can determine if it is operating

* This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants EIA-0000433, EIA-0130724, and ITR-0313142.

[1] The term MESO refers to the tree algorithm used by the system (Multi-Element Self-Organizing tree).

within acceptable limits. For instance, if a network application perceives a high packet loss rate, it might interpret this condition as detrimental to quality of service and decide to increase the level of error correction. Once invoked, this response is evaluated and if acceptable, assimilated into the decision maker's experience for possible future use in similar situations. *That is, adaptive systems need to remember and recall past experience.*

We explore pattern classification and clustering methods for associating adaptive responses with observed or sensed data pertinent to application quality-of-service. The embodiment of this method is a *classifier* [6] that uses supervised learning to produce a model of environmental stimuli. Comprising the operation of a classifier are two basic functions: *training* and *testing*. During training, patterns are added to the classifier, enabling the construction of an internal model of the training data. Each training pattern is an array of continuous, binary or nominal values labeled as belonging to a specific, real-world category. Once the classifier has been trained, the system can be queried using an unlabeled pattern. The classifier then tests this pattern and returns a label, indicating the category to which the tested pattern most likely belongs.

Clustering and pattern classification research is an active field of study. Recently, a number of projects have addressed clustering and classification of large data sets, a characteristic of decision making for autonomic software. Tantrum et al. [21] consider model-based refractionation for clustering large data sets. Yu et al. [24] use an hierarchical approach to clustering using support vector machines (SVMs). Kalton et al. [14] address the growing need for clustering by constructing a framework that supports many clustering algorithms. Methods for online clustering and classification have also been explored [4, 17]. Such online methods might be used as the basis for a perceptual memory system similar to MESO.

Like our project, other works have explored the use of statistical methods and pattern classification and clustering techniques in learning systems. Some have used *developmental learning* algorithms that enable a system to learn online through interaction with the physical world. For example, Hwang and Weng [9] developed hierarchical discriminant regression (HDR) and applied it successfully as part of the developmental learning process in humanoid robots. In addition, Ivanov and Blumberg [11] developed the layered brain architecture [11], which was used for the construction of synthetic creatures, such as a "digital dog." That project used clustering and classification methods to construct perceptual models as part of the dog's developmental learning system. A notable aspect of that work is the use of compression schemes to limit the impact of large training sets on memory consumption and processing power requirements.

Imitative learning, where a learner acquires skills by observing and remembering the behavior of a teacher, has also been studied in the context of providing humanoid robots with motor skills. Amit and Matarić [1] used hidden Markov models (HMMs) to learn aerobic-style movements. The ability of the system to reconstruct motion sequences is encouraging, demonstrating the potential importance of imitative learning. Jebar and Pentland [13] conducted imitative learning experiments using a wearable computer system that included a camera and a microphone. A human subject was observed by the system during interactions with other people. The observed training data was used to train an HMM. Later the system was allowed to respond autonomously when presented with visual and audio stimuli, demonstrating a limited ability to reproduce correct responses. However, since learning by observing real human behavior is very complex, even limited recognizable response is significant and promising.

The key hypothesis of our project is that clustering and classification methods can be extended to support decision making in *adaptive software*. We focus in this paper on applications that operate over lossy wireless networks and thereby must accommodate periods of high packet loss. Since error correction or retransmission often consumes additional bandwidth and increases packet delay, applications must balance these concerns while correctly choosing a response for current conditions. By measuring the loss rate, bandwidth and other network and system behavior, a pattern can be constructed that enables a decision maker to "remember" an appropriate response.

3 MESO

As a first step in our study, we developed MESO, a perceptual memory system for adaptive software. At a basic level, MESO functions as a pattern classifier and can be used to incrementally classify environmental stimuli or other data while accomodating very large data sets. Indeed, in early experiments we used HDR [9] to classify environmental stimuli related to application quality of service. The insight gleaned from those experiments led to our design of MESO.

Basic Approach. As depicted in Figure 1, a unique feature of MESO's design is the use of small agglomerative clusters, called *sensitivity spheres*, that aggregate similar training patterns. In adaptive software, training patterns comprise observations related to quality of service or environmental context, such as network bandwidth or physical location. The quantity of training patterns collected while a system executes may be very large, requiring more memory and processing resources as new patterns are added to the classifier. However, many training patterns may be very similar, enabling their aggregation into a much smaller

number of sensitivity spheres while helping limit resource consumption.

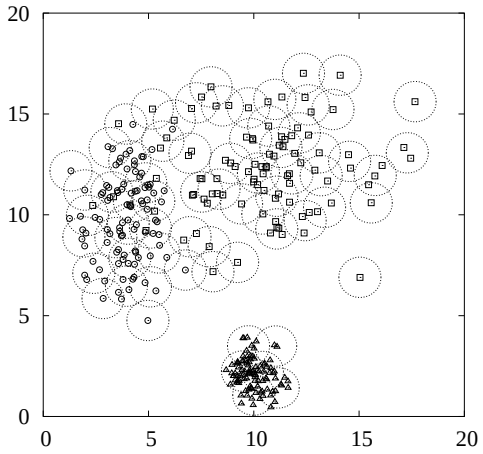

Figure 1. Sensitivity spheres for three 2D-Gaussian clusters. Circles represent the boundaries of the spheres as determined by the current δ. Each sphere contains one or more training patterns.

The size of the sensitivity spheres is determined by a δ value that specifies the sphere radius in terms of distance (e.g. Euclidean distance) from the sphere's center. A sphere's center is calculated as the mean of all patterns that have been added to that sphere. The δ is a ceiling value for determining if a training pattern should be added to a sphere, or if creation of a new sphere is required.

As with many other classifiers, MESO uses a hierarchical data structure, or tree, to organize training patterns for efficient retrieval. Figure 2 shows the organization of a MESO tree. The MESO tree is built starting with a root node that comprises the set of all sensitivity spheres. The root node is then split into subsets of similar spheres, producing child nodes. Each child node is further split into subsets until each child comprises only one sphere. Consolidating similar patterns into sensitivity spheres enables construction of a tree using only spheres rather than agglomerating individual patterns at tree nodes. Moreover, many clustering algorithms construct a tree by agglomerating individual patterns into large clusters near the root of the tree, and then split these clusters at greater tree depths. In such a structure, the tree must be reorganized using the training patterns directly. Conversely, MESO can be reorganized using only existing sensitivity spheres. The use of sensitivity spheres enables a MESO tree to be more rapidly reorganized than approaches that require direct consideration of training patterns.

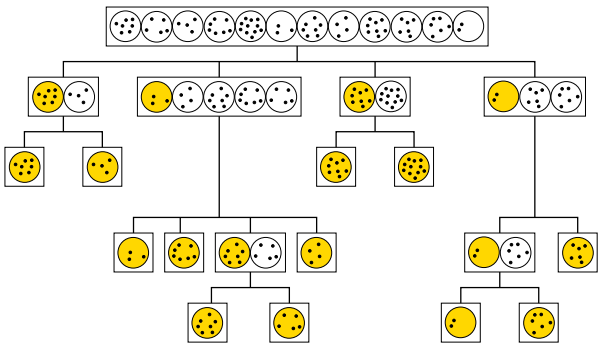

Figure 2. MESO tree organization. The rectangles are partitions and the shaded spheres are partition pivots. The root partition is split successively until a leaf is formed where a partition contains only one sphere.

As shown in Figure 2, sensitivity spheres are partitioned into sets during the construction of a tree. Each node of the tree comprises a collection of sensitivity spheres called a *partition*, defined as a subset of similar spheres. A partition may have one or more children, each comprising a subset of the parent's sensitivity spheres. A *pivot sphere* is selected for each partition and used to assign other spheres, nearest to the pivot, as members of the partition. Smaller partitions provide finer discrimination and better classification of test patterns. Moreover, the partitioning of sensitivity spheres produces a hierarchical model of the training data. That is, each partition is an internal representation of a subset of the training data that is produced by collecting those spheres that are most similar to a pivot sphere. At greater tree depths, parent partitions are split, producing smaller partitions of greater similarity. For each partition, classification proceeds by comparing a test pattern with each child's pivot and following the branch to the child containing the pivot that is most similar. In this way, the search continues at each tree depth. At a leaf node, a label is returned indicating the category to which the test pattern most likely belongs.

By leveraging this hierarchical structure, a relatively simple distance metric, even Euclidean distance, can be used to achieve high recall accuracy. As shown in Section 4, another advantage of this approach is that it requires a minimum amount of calculation, enabling rapid classifier training and testing. In addition, MESO supports incremental training, which enables construction of a MESO tree by adding one pattern at a time. As such, MESO can be trained and tested during concurrent interaction with users or other system components.

Sensitivity Sphere Size. An important consideration in building an efficient MESO tree is the appropriate value of

δ for constructing sensitivity spheres. For each data set, a δ value needs to be calculated. If δ is too small, training time increases dramatically. If too large, testing time becomes unacceptable. Moreover, data set compression requires a proper δ value to balance the tradeoff between compression rate and recall accuracy.

Since online learning is an incremental process, it is possible to adjust δ incrementally as MESO is trained. Key to incrementally calculating a good δ is determining a δ *growth function* that balances sphere creation with sphere growth. That is, rapid δ growth during early training produces few spheres, with very large δ's, that agglomerate many patterns. The result is a coarse-grained, inefficient tree where most testing time may be spent in a linear search of a single sphere. Conversely, overly slow δ growth produces a large number of very small spheres that contain only a few patterns. Having many small spheres is expensive, in terms of both space and time, to organize as a MESO tree.

MESO's δ growth function is defined as:

$$grow_\delta = \frac{(d-\delta)\frac{\delta}{d}f}{1+\ln(d-\delta+1)^2},$$

where d is the distance between the new pattern and the nearest sensitivity sphere. The $\frac{\delta}{d}$ factor scales the result relative to the difference between the current δ and d. Intuitively, the denominator of $grow_\delta$ limits the growth rate based on how far the current δ is from d. That is, if d is close to δ then δ will grow to be nearly d. However, if d is much larger than δ, the increase will be only a small fraction of $d-\delta$.

The sigmoid-curve activation function, f, inhibits sensitivity sphere growth when the number of spheres is small compared to the number of patterns. This function is defined as:

$$f = \frac{1}{2} + \frac{\tanh(\frac{3r}{c}-3)}{2},$$

where $r = \frac{spheres}{patterns}$ and c is a configuration parameter in the range $[0, 1.0]$. Increasing c moves the center of the sigmoid curve to the right, suppressing sphere growth and encouraging the production of new spheres. The $\tanh()$ function is the hyperbolic tangent.

Our $grow_\delta$ function balances sphere production with sphere growth, producing good spheres for a wide range of values for c. Only for very large values of c is growth inhibited sufficiently to significantly impact training time. The $grow_\delta$ function promotes the production of trees that are comparable with good choices for fixed δ values.

Compression. Adaptive systems often must continue to function for long periods while addressing changing user preferences and the sensed environment. Such an enormous amount of data requires substantial processor and storage resources, potentially inhibiting timely response by decision makers or impacting application performance. Thus, perceptual memory systems may need to "forget" less informative training samples in favor of important or novel observations. Compression techniques eliminate training patterns while attempting to minimize information loss.

In MESO, compression takes place on a per sensitivity sphere basis. That is, rather than trying to compress the entire data set using a global criterion, the patterns in each sensitivity sphere are compressed separately. Moreover, compression is further limited so that all existing pattern labels are not eliminated from a sphere. We implemented three types of compression, the evaluation of which is discussed in Section 4.

Means compression reduces the set of patterns in each sensitivity sphere to the mean pattern vector for each label. This is the most aggressive and simple of the compression methods. Moreover, the computational requirements are quite low.

Spherical compression is a type of boundary compression [11] that treats patterns on the boundaries between sphere's as most important to the classification of test patterns. For each sphere, the feature values are converted to spherical coordinates. Along a given vector from the sphere center, only those patterns farthest from the sphere center are kept.

Orthogonal compression removes all the patterns that are not used for constructing an orthogonal representation of a sphere's patterns. The idea is to keep only those patterns that are most important as determined by their orthogonality. Samples that represent parallel vectors in m-dimensional space are removed.

Complexity. Table 1 shows the space and time complexities for training MESO and some well-known clustering algorithms [12]. In this table, n is the number of patterns, k is the number of clusters and l is the number of iterations to convergence. Without compression, MESO has a worst case space complexity of $O(n)$, comparable to the shortest spanning path algorithm. MESO's memory consumption can be significantly reduced with compression.

Intuitively, time complexity for training can be considered in terms of locating the sensitivity sphere nearest to a new pattern and adding the pattern to that sphere. If a sufficiently close sphere cannot be found, a new sphere is created. Locating the nearest sphere is an $O(\log_q k)$ operation. This search must be completed once for each of n patterns. Each pattern must also be added to a sensitivity sphere, and k sensitivity spheres must be created and added to the MESO tree. This process yields a complexity of $O(n \log_q k) + O(n) + O(k) + O(k \log_q k)$ which reduces to $O(n \log_q k)$. Assuming an appropriate δ selection and a data set of significant size, MESO has a time complexity better than the leader algorithm.

Table 1. Space and time complexities for several clustering algorithms [12].

Algorithm	Space	Time
MESO	$O(n)$	$O(n \log_q k)$
leader	$O(k)$	$O(kn)$
k-means	$O(k)$	$O(nkl)$
ISODATA	$O(k)$	$O(nkl)$
shortest spanning path	$O(n)$	$O(n^2)$
single-line	$O(n^2)$	$O(n^2 \log n)$
complete-line	$O(n^2)$	$O(n^2 \log n)$

The search complexity for classifying a test pattern using MESO is $O(\log_q k) + O(\bar{s})$ for a balanced tree, where q is the maximum number of children per node, $\bar{s}$ is the average number patterns agglomerated by a sensitivity sphere, and k represents the number of sensitivity spheres produced. The $\bar{s}$ component represents the number of operations required to assign a category label once the most similar sensitivity sphere has been located. Thus, the worst case search complexity occurs when only one cluster is formed and the search algorithm degenerates into a linear search of $O(n)$. Conversely, a best case search complexity of $O(\log_q k)$ occurs when one sensitivity sphere is formed for each training pattern.

4 MESO Assessment

We evaluated MESO as a pattern classifier using several standard data sets in cross-validation experiments. The results illustrate the recall accuracy of MESO. We also compare MESO with the IND classifier [3] to provide a benchmark for MESO performance.

The Data Sets. Table 2 lists the data sets used to assess MESO. The number of patterns and features per pattern are shown for each data set. All but one of the data sets were retrieved from the UCI [2] and KDD [8] machine learning repositories. The ATT faces [20] data set was acquired from AT&T Laboratories Cambridge. Most of these data sets comprise continuous, integer or binary feature measurements. For instance, the Cover Type data set contains continuous features, measuring features such as elevation or slope, and binary values indicating whether a pattern is a particular soil type. However, the Mushroom data set consists entirely of nominal values encoded as alpha characters converted to their ASCII equivalent for processing by MESO. In contrast, the ATT Faces data set comprises image pixel values of human faces.

The Japanese Vowel data set requires further description. This data set comprises 640 time series blocks where each block consists of a set of records. Each record comprises 12 continuous measurements of utterances from nine male speakers. The 9,859 patterns are produced by treating each record as an independent pattern and randomizing the data set. As such, no understanding of utterance order is retained. Thus, the classification task is to identify the speaker of each utterance independent of its position in a time series.

Table 2. Data set sizes and feature counts.

Data Set	Size	Features	Classes
Iris	150	4	3
ATT Faces	360	10,304	40
Mult. Feature	2,000	649	10
Mushroom	8,124	22	2
Japanese Vowel	9,859	12	9
Letter	20,000	16	26
Cover Type	581,012	54	7

Experimental Method. We tested MESO using cross-validation experiments as described by Murthy et al. [19]. The experiment proceeds as follows:

1. Randomly divide the training data into p equal-sized partitions.
2. For each partition, train the classifier using all the data outside of the selected partition. Test the classifier using the data in the selected partition.
3. Calculate the classification accuracy by dividing the sum of all correct classifications by the total number of patterns tested.
4. Repeat the preceding steps t times, and calculate the mean and standard deviation for the t iterations.

In our tests we divide each data set into 10 equal-sized partitions and repeat the test 10 times. Thus, MESO is trained and tested 100 times for each mean and standard deviation calculated.

Experimental Results. Table 3 shows MESO's cross-validation accuracy and standard deviations for each of the data sets. For comparison, accuracy using the IND classifier is also presented. IND can be used to build a classifier in several modes. Here we include results using CART, ID3 and Bayesian IND modes. As shown, MESO is competitive with IND, exhibiting better accuracy for most data sets. Moreover, MESO has good accuracy for data sets of different size and pattern feature count. The NC designation indicates that IND could not complete a particular test. In the case of ATT Faces, lack of memory prevented IND from completing a data set encoding process, which must be completed before IND can be trained.

It is noteworthy that MESO shows high accuracy for the Mushroom data set, since this data set consists entirely of nominal values. Such pattern values have no comparative numeric value but rather indicate characteristics, such as cap shape, by name. IND addresses such nominal values explicitly by designation of some features as nominal. MESO does not explicitly address nominal values, but still

Table 3. MESO accuracy compared to IND.

Data set	MESO	IND		
		CART	ID3	Bayesian
Iris	95.1%±0.0%	92.8%±0.3%	93.5%±0.7%	94.2%±1.1%
ATT Faces	94.1%±1.0%	NC	NC	NC
Mult. Feature	94.0%±0.6%	93.1%±0.6%	94.2%±0.2%	94.4%±1.1%
Mushroom	100.0%±0.0%	99.9%±0.0%	100.0%±0.0%	100.0%±0.0%
Japanese Vowel	91.9%±0.2%	82.3%±0.3%	84.2%±0.3%	84.7%±0.3%
Letter	90.1%±0.2%	84.4%±0.3%	87.9%±0.1%	88.6%±0.2%
Cover Type	96.0%±0.0%	93.9%±0.9%	95.2%±0.2%	94.4%±0.3%

Table 4. Results when using compression.

Data set		Uncompressed	Means	Spherical	Orthogonal
Iris	Accuracy%	95.1%±0.0%	95.7%±1.0%	96.0%±1.3%	96.2%±2.27%
	Compression%	0.0%	0.02%±0.0%	0.0%±0.0%	1.9%±0.07%
ATT Faces	Accuracy%	94.1%±1.0%	93.2%±1.1%	93.7%±1.5%	93.9%±1.7%
	Compression%	0.0%	0.0%±0.0%	0.0%±0.0%	0.0%±0.0%
Mult. Feature	Accuracy%	94.0%±0.6%	94.2%±0.6%	94.3%±0.5%	94.2%±0.5%
	Compression%	0.0%	0.3%±0.0%	0.3%±0.0%	0.3%±0.0%
Mushroom	Accuracy%	100.0%±0.0%	99.9%±0.0%	99.7%±0.0%	99.9%±0.0%
	Compression%	0.0%	90.3%±0.0%	71.9%±0.5%	90.1%±0.0%
Japanese Vowel	Accuracy%	91.9%±0.2%	81.0%±0.4%	90.1%±0.5%	80.6%±0.7%
	Compression%	0.0%	93.9%±0.1%	26.5%±1.3%	93.9%±0.0%
Letter	Accuracy%	90.1%±0.2%	87.8%±0.3%	90.2%±0.4%	87.5%±0.4%
	Compression%	0.0%	88.9%±0.2%	22.0%±1.0%	89.0%±0.2%
Cover Type	Accuracy%	96.0%±0.0%	82.5%±0.7%	95.1%±0.0%	82.1%±0.3%
	Compression%	0.0%	98.2%±0.2%	48.0%±0.9%	98.3%±0.1%

accurately classifies these nominal patterns. The high recall accuracy may be due to MESO's ability to capture m-dimensional structure. Determining how MESO addresses nominal and mixtures of nominal and continous values warrants further exploration.

Compression. Table 4 shows MESO results using the three data compression methods described earlier. All results were generated using cross-validation. Means compression provides high compression and good accuracy. This result can be attributed to applying compression on a per sensitivty sphere basis. As such, the ability of MESO to capture the m-dimensional structure of the training data can be leveraged to limit information loss during compression. For all compression methods, small data sets are compressed very little. Limited compression is expected since δ growth is inhibited during early training, and spheres contain only a few samples. However, since processor and storage usage is low for small data sets, compression is of limited importance. Larger data sets are compressed significantly while recall accuracy remains high.

5 The Network Application

We explored the use of MESO to support learning in adaptive software by applying it to the implementation of an audio streaming network application, called XNetApp, that can adapt to changes in packet loss rate in a wireless network. A stationary workstation transmits an audio data stream to a mobile receiver over the wireless network. Our goal is to enable the mobile device to adapt to the wireless packet loss encountered as a user roams about a wireless cell. One way to address the high loss rates of wireless channels is to insert redundant information into the data stream, enabling a receiver to correct some losses without contacting the sender for retransmission. This study focuses on *erasures* of packets. An (n, k) *block erasure code* [18] converts k source packets into n encoded packets, such that any k of the n encoded packets can be used to reconstruct the k source packets.

The XNetApp's decision maker uses MESO for "remembering" user preferences for balancing packet loss with bandwidth consumption. By autonomously modifying (n,k) settings and packet size, the decision maker can adapt the XNetApp based on current environmental conditions. In our experiments, the decision maker learns about a user's preferences through imitative learning. Thus, a user shows the XNetApp how to adapt to a rising loss rate by selecting an (n,k) setting with greater redundancy. If the new setting reduces the perceived loss rate to an acceptable level, the user reinforces the new configuration (e.g., by pressing a particular key), and the XNetApp "remembers" the sensed environment and current configuration by storing it using MESO. When operating autonomously, the decision maker senses current environmental conditions and calculates time-sampled and Fourier features, constructing a pattern. Using this pattern, the XNetApp queries MESO for a system configuration that most likely addresses current

conditions. Then, the decision maker emulates the user's actions and adapts the XNetApp, changing the configuration to match that returned from MESO.

As shown in Table 5, 56 environmental features are sensed or calculated and used as input to the decision making process. The first 4 features are instantaneous measurements. Perceived features are observed from the application's viewpoint. That is, perceived packet loss represents the packet loss as witnessed by the application following error correction, while real packet loss is the number of packets actually dropped prior to error correction. For each of the first four features, 7 time-sampled measurements and 6 Fourier spectrums are calculated.

Table 5. Features used for training and testing the XNetApp.

#	Feature Description
1–4	Instantaneous measurements: bandwidth, perceived packet delay, perceived loss and real loss.
5–32	Time-sampled measurements: median, average, average deviation, standard deviation, skewness, kurtosis and derivative.
33-56	Fourier spectrum of the time-sampled measurements: median, average, average deviation, standard deviation, skewness and kurtosis.

6 Results

For testing, we studied the ability of the XNetApp to autonomously balance real packet loss and bandwidth consumption. The IP multicast protocol was used for transmission of the data stream. Data was collected as a user roamed about a wireless cell carrying a notebook computer running an XNetApp receiver.

We experimented with the XNetApp using a 1.5GHz AMD Athlon workstation for data transmission. A 500MHz X20 IBM Thinkpad notebook was used as a mobile receiver. All systems run the Linux operating system. We tested atop an 11Mb 802.11b wireless network as a user roamed about a wireless cell. The XNetApp was trained using an emulated loss rate in the range $[0, 0.6]$.

Figure 3 shows a comparison of real versus perceived loss observed as the user roamed about the cell. The XNetApp was able to adapt to changing real loss rates, and minimize application loss. The center plot depicts the redundancy ratio, defined as $\frac{(n-k)}{n}$, showing the responsiveness of the XNetApp on a real wireless network. Greater redundancy is required during periods of high loss. However, this redundancy consumes greater network bandwidth. The XNetApp did not simply choose a high (n, k) ratio, but changed parameters to correspond with the changing real loss rate.

Table 6 shows results from running cross-validation tests using the data acquired during XNetApp training. This

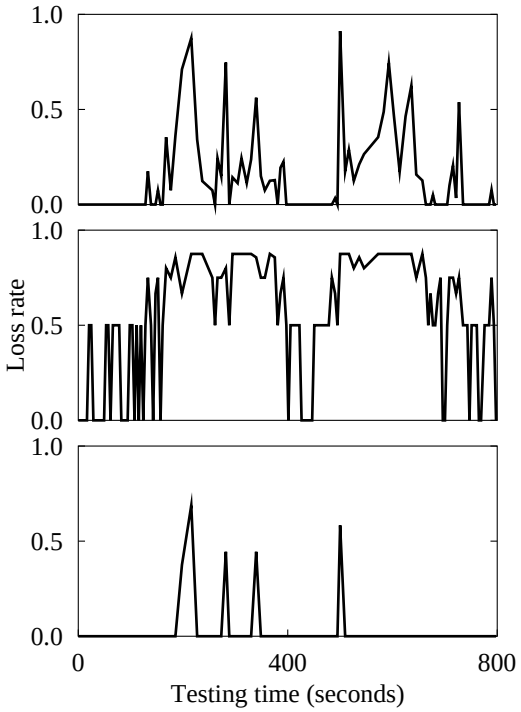

Figure 3. Comparison of real loss (top), redundancy ratio (center) and perceived loss (bottom).

data was produced during training for autonomous XNetApp operation on a real wireless network. The final training set contained 32,709 patterns in 10 classes. This table shows recall accuracy, with and without compression, helping quantify how well the XNetApp can be expected to imitate a user. In all cases, the XNetApp was highly accurate at "remembering" a user's preferences for balancing loss rate with bandwidth consumption.

Table 6. XNetApp results with and without compression.

Uncompressed	Accuracy%	94.1%±0.2%
	Compression%	0.0%
Means	Accuracy%	87.7%±0.2%
	Compression%	91.8%±0.1%
Spherical	Accuracy%	92.4%±0.7%
	Compression%	5.8%±0.2%
Orthogonal	Accuracy%	87.3%±0.4%
	Compression%	91.8%±0.1%

7 Conclusions

We have presented a perceptual memory approach, called MESO, that uses pattern classification and clustering techniques while addressing issues important to sup-

port online developmental learning. We showed that, when used as a pattern classifier, MESO can accurately classify patterns from standard data sets. We also implemented an application that imitatively learns how to make decisions through interaction with a user. Preliminary results show that the imitative learning approach, used by the XNetApp, has promise. We postulate that such software, which can be trained to make good decisions, may simplify the integration of software into new or pervasive computing environments.

Further information. This work is part of the RAPIDware project, which addresses the design of high-assurance adaptive software. Related papers and software packages can be found at `http://www.cse.msu.edu/rapidware`.

Acknowledgements. The authors would like to thank Professor Juyang Weng, Xiao Huang and Dave Knoester at Michigan State University for their contributions to this work.

References

[1] R. Amit and M. Matarić. Learning movement sequences from demonstration. In *Proceedings of the 2nd International Conference on Development and Learning*, pages 165–171, Boston, MA, June 2002.

[2] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[3] W. Buntine. Tree classification software. In *Third National Technology Transfer Conference and Exposition*, Baltimore, MD, December 1992.

[4] K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, Cambridge, MA, 2003.

[5] Proceedings of the distributed auto-adaptive and reconfigurable systems (DARES'04), held in conjunction with the 24nd international conference on distributed computing systems ICDCS'04, March 2004.

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification, Second Edition*. John Wiley and Sons, Incorporated, New York, NY, 2001.

[7] J. M. Fuster. *Memory in the Cerebral Cortex: An Empirical Approach to Neural Networks in the Human and Nonhuman Primate*. The MIT Press, Cambridge, MA, 1995.

[8] S. Hettich and S. D. Bay. UCI KDD archive. http://kdd.ics.uci.edu, 1999.

[9] W.-S. Hwang and J. Weng. Hierarchical discriminant regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), November 2000.

[10] Proceedings of the international conference on autonomic computing (ICAC'04), May 2004.

[11] Y. A. Ivanov and B. M. Blumberg. Developmental learning of memory-based perceptual models. In *Proceedings of the 2nd International Conference on Development and Learning*, pages 165–171, Boston, MA, June 2002.

[12] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computer Surveys*, 31(3):264–323, September 1999.

[13] T. Jebara and A. Pentland. Statistical imitative learning from perceptual data. In *Proceedings of the 2nd International Conference on Development and Learning*, pages 191–196, Boston, MA, June 2002.

[14] A. Kalton, P. Langley, K. Wagstaff, and J. Yoo. Generalized clustering, supervised learning, and data assignment. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 299–304, San Francisco, CA, August 2001.

[15] E. P. Kasten and P. K. McKinley. MESO: Perceptual memory to support online learning. Technical Report MSU-CSE-04-15, Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, April 2004.

[16] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, pages 41–50, January 2003.

[17] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, Cambridge, MA, 2002.

[18] A. J. McAuley. Reliable broadband communications using burst erasure correcting code. In *Proceedings of ACM SIGCOMM*, pages 287–306, September 1990.

[19] S. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research (JAIR)*, 2:1–32, 1994.

[20] F. Samaria and A. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 2nd IEEE Workshop on Applications of Computer Vision*, Sarasota, FL, December 1994.

[21] J. Tantrum, A. Murua, and W. Stuetzle. Hierarchical model-based clustering of large datasets through fractionation and refractionation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 183–190, Edmonton, Alberta, CA, July 2002.

[22] J. Weng and W.-S. Hwang. An incremental learning algorithm with automatically derived discriminating features. In *Proceedings Asian Conference on Computer Vision*, pages 426–431, Taipei, Taiwan, January 2000.

[23] Proceedings of the ACM workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), held in conjunction with the 16th annual ACM international conference on supercomputing, June 2002.

[24] H. Yu, J. Yang, and J. Han. Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 306–315, Washington, D.C., August 2003.