Published in Spring-Verlag Lecture Notes in Artificial Intelligence
UM2003 User Modeling: Proceedings of the Ninth International Conference

Modeling Multitasking Users

Malcolm Slaney, Jayashree Subrahmonia, Paul Maglio

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120
malcolm@ieee.org, jays@us.ibm.com, pmaglio@almaden.ibm.com

Abstract. This paper describes an algorithm to cluster and segment sequences of low-level user actions into sequences of distinct high-level user tasks. The algorithm uses text contained in interface windows as evidence of the state of user–computer interaction. Window text is summarized using latent semantic indexing (LSI). Hierarchical models are built using expectation–maximization to represent users as macro models. User actions for each task are modeled with a micro model based on a Gaussian mixture model to represent the LSI space. The algorithm's performance is demonstrated in a test of web-browsing behavior, which also demonstrates the value of the temporal constraint provided by the macro model.

1 Problem Statement

To design interfaces that effectively support human-computer interaction, we must first understand the complex behavior computer users exhibit when carrying out their jobs. Apple Computer, for example, popularized an approach to interaction design based on observing users as they perform their work and then analyzing how users interact with specific software components [8]. This type of interaction design optimizes interfaces for individual users performing individual tasks. Our approach, by contrast, supposes individual users constantly shift among tasks, seamlessly interleaving low-level activities in the pursuit of high-level goals [3]. Here, we describe a method for modeling users engaged in a sequence of many different tasks.

Models of multitasking users can be used by adaptive or attentive user interfaces [2, 10, 11], which monitor user behavior in order to anticipate user needs. These sorts of systems aim to automatically provide users with additional information just when it would be most helpful. By relying on a user model that keeps very close tabs on users shifting tasks, such systems can potentially provide very precisely targeted information.

Our approach of modeling multitasking users has applications beyond the creation or adaptation of individual user interfaces. For instance, a corporation might want to understand the behavior of a computer system used by large number of employees, as they use a number of applications to perform their tasks, and optimize overall system cost. This sort of large-scale interface optimization requires understanding the behavior of a large number of users as they move from task to task during the day.

In this paper, we describe a method for discovering and building a hierarchical model of user activities from unlabeled data. Given a trace of user activities, we segment the user data and learn multiple micro models, each corresponding to a separate "task." Each task is defined by a set of actions that are represented as a single micro

model. A macro model controls switching between micro models of individual tasks. These models can take many forms, including discrete Markov chains and continuous hidden Markov models (HMM) [7]. We demonstrate the time-series clustering algorithm with a simple web-browsing example.

2 Related Work

Work in user modeling has focused mainly on building a single model of a user's activities. For instance, Davison and Hirsch [4] describe a system called IPAM that builds a table which predicts the next command given a list of past commands. At any point in time, they predict the most likely next command by indexing into a table with the last N commands (where N is between 0 and 5). The table is updated in real-time as the user enters new commands and the correct Unix command is predicted upwards of 70% of the time. In our approach, each micro model makes the same type of predictions as IPAM, but our macro model captures the relationship between a set of tasks with different probabilities.

Horvitz and his colleagues [6] describe a system that uses a Bayesian model to infer a software user's goals. They developed a language to link the user's and computer's actions to elemental features that can be used by the inference engine. In this work, we use the text showing in an active window to judge the state of the interactive system.

Westphal and Syeda-Mahmood [14] learn a model of user's behavioral state as they interact with a video browser. Given a sequence of low-level events such as "fast forward" or "play" the system learns user's states such as "aimless browse" or "found something interesting". Their approach is supervised; they train the system with a small set of labeled data. The user states their goal and the system then learns the corresponding pattern of low-level events. Our approach on the other hand is completely unsupervised. (Although in a real application somebody would probably look at the micro models and assign them names.)

Our work builds on the hidden Markov experts ideas proposed by Weigend [13]. His goal was to automatically cluster the time-series data and build models that predict different portions of the data with different experts. Our goal is to simply label the data and we extend Weigend's work by using a novel text feature to capture the user's state.

3 Hierarchical Segmentation Algorithm

We cluster time-series data with a hierarchy of models. Figure 1 illustrates the basic model. In this work, a set of high-level models with three states (S_i) determines the high-level behavior of the signal. For instance, each macro state (S_i) might correspond to one speaker in a speaker-segmentation task, one user task in a user-interface interaction recognition task, or one type of multimedia content. We assume that the system can move from one macro state to another at any time, under control of transition probabilities that are assumed or learned from the data. Each macro state controls execution of its own micro model, which outputs feature vectors based on its own transition and output probabilities. The macro state of the system is hidden, except for the change in feature output probabilities captured by the different micro models. Each macro state controls a micro

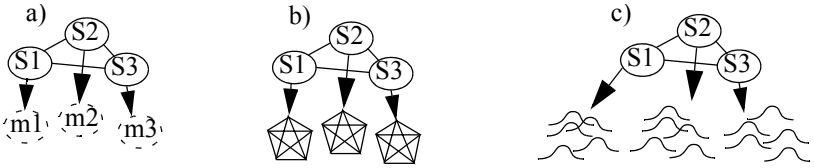

Fig. 1: A hierarchy of models. a) The general model is a macro model with states Si, each macro state has its own micro model (mi) of arbitrary complexity to generate output data. b) A specific form of hierarchical model with micro models implemented using five-state fully-connected Markov models. c) A continuos model where each micro model is implemented using Gaussian mixture models (GMM). This is the form described in this paper.

model that generates features we can observe. These micro models can take many different forms.

A discrete hierarchical model is shown in Figure 1b. In this case, we are interested, for example, in modeling a user's interaction with a computer interface using a discrete set of features. When a user is performing the "create a database" task, he or she will step through particular dialog boxes and tabbed windows. When in the "file open" dialog box, the probability that the user will go to the "name database" dialog box will depend on whether the "create a database" macro model is active. We model the system with a set of (hidden) macro states described by a Markov model. Each micro model is represented by its own Markov model, where the micro-state output is equal to the state label. The model in Figure 1b generates signals that switch between models. When described this way, the model is not a simple hidden Markov model.

Figure 1c shows the model structure described in this paper. In this case the macro states of the system are described with a Markov model, each macro state controlling a single micro model implemented with Gaussian mixture models (GMM). A GMM micro model corresponds to the output probabilities in a conventional HMM. This type of model is good for data with little structure and continuous features such as for speaker segmentation and for modeling and labeling multimedia data.

Figure 2 shows an overview of the hierarchical segmentation algorithm we use to cluster, segment and summarize the user's actions. We use the text on the user's screen at each point in time as input to this algorithm. Latent semantic indexing (LSI), described in Section 3.1, encodes the interface's text as a multidimensional feature vector as a function of time. The best segmentation of the user signal and the parameters of the micro model are estimated using the expectation–maximization (EM) algorithm as described in Section 3.2.

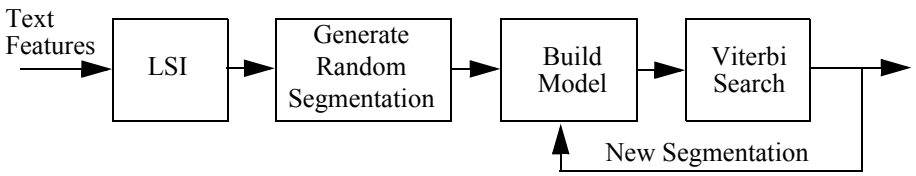

Fig. 2: Block diagram of the hierarchical clustering and segmentation algorithm.

3.1 Text Features via LSI

Actions need to be encoded in a manner that allows us to associate probabilities with the user's events. In this work we consider the text on the screen, due to the user's actions, as a good indication of what the user is trying to do. A "Print" dialog box contains words about printing, while a help page gives information about the options a user has. There are many ways to describe an action. One application might use the words "open document" while another might use "open file." Latent semantic indexing (LSI) gives us a feature set that spans these differences [5].

We use LSI to form a feature vector that summarizes the text contained in each window displayed to the user. LSI is often used in information retrieval to cluster documents and for determining the similarity between a document and a semantic query. LSI creates a bag-of-words model for each document. A term-frequency histogram is formed by counting how many times each word occurs in each document, regardless of what words precede or follow each word. This data is then used as input to a singular-value decomposition (SVD) that finds a low dimensional sub-space that approximates the original histogram space.

A feature space based on LSI solves two difficult problems associated with semantic information retrieval: synonyms and polysemy. Often, two or more words have the same meaning—synonyms. For information retrieval, we want to use any synonym to retrieve the same information. Conversely, many words have multiple meanings—polysemy. For example, apple in a story about a grocery store is likely to have a different meaning from Apple in a story about a computer store.

Changes in semantic space are based on angles, rather than on distance. A simple "sentence" such as "Yes!" has the same semantic content as "Yes, yes!" Yet the second sentence contains twice as many words, and, in semantic space, it will have a vector magnitude that is twice as large. After computing the SVD and projecting the raw histogram data into a low-dimensional subspace, we normalize[1] each document vector so it has a vector length of 1. This normalized vector is a semantic feature that describes what we know about the user's actions at each point in time.

In this work, we use GMMs to form the micro models that describe each task. A GMM models the probability distribution of semantic events that form one task using a sum of N_m Gaussian "bumps" [7]. This is a simple model since any one topic (represented by one Gaussian) can follow any other topic (another Gaussian) with equal probability. Richer micro models are also possible.

3.2 Training

We use the EM algorithm to train a hierarchy of models. Given an initial set of models, we compute the best macro- and micro-level models. All the data corresponding to a macro state are used to train the micro model corresponding to the macro state. We re-

1. Note, normalizing all document vectors so they have unit length does not make the pattern discrimination easier—we lose information when this is done. But the resulting vectors all lie on the unit sphere and are a better match for the diagonal-covariance GMMs we use to model their distribution.

peat the procedure until we reach a stable solution. Training the micro model involves estimating the output probabilities of the GMM.

The segmentation training algorithm is a straightforward application of EM. We describe the approach by segmenting a discrete signal with N multidimensional points and finding N_c clusters with N_m states in each micro model. The topology is shown in Figure 1c. More details on this class of algorithms is in Weigend's paper [13].

Assumption: We assume that the macro model has N_c states, with a self-loop probability of $1-\varepsilon$ and a probability of a transition to any other state of $\varepsilon/(N_c-1)$. This simple model encourages temporal continuity on the macro-state sequence. It is equivalent to saying that on average a user spends $1/(1-\varepsilon)$ time steps performing each task, and any one task is equally likely to follow any other task. Richer models, as represented by different Markov macro models, do not change the algorithm shown below.

Initialization: Choose $N_c - 1$ points in the region $(1, N)$. These points define an initial segmentation. If any segment has too few points, choose a new segmentation. For each segment, build a micro model (m_i) which captures the transition probabilities. These are the initial m_i models. See Section 3.3 for more details.

E-step: Given the models, use the Viterbi algorithm [7] to find the path through the lattice that has the maximum likelihood. We can use this path to decode the signal and decide the macro and micro states that are most likely to generate each portion of the signal.

Termination test: Exit this loop when (a) the signal's temporal cluster assignments do not change or (b) after 10 iterations.

Degenerate check: Make sure that all models are used to cluster some portion of the signal. If there is no data assigned to one model (M_i) then find the cluster (M_j) with the largest temporal support. Concatenate all the segments assigned to M_j and split this signal at a random point. Model M_i is relearned from the portion of the signal before the split point; Model M_j is relearned from the portion of the signal after the split point. See Section 3.4

M-step: For each cluster (M_i) concatenate all the chosen portions of the signal. Build a new model (m_i) which captures the transition and output probabilities of the data at this state. Return to the E-step.

The micro model, m_i, used in the E-step and trained in the M-step can take many forms. It can be a time-varying Markov model or a simple GMM as we describe in the rest of this paper.

In practice, the performance of this algorithm depends on how the models are initialized and what happens when a model becomes degenerate because it no longer wins any of the time-series data in the E-step.

3.3 Initialization

Clusters with a k-means algorithm [7] are often initialized with a random data point from the data set. The temporal micro models are more complicated so we need to use more data. We had the best success segmenting the initial training data into N_c non-overlapping random-length segments and using each segment to train one of the N_c macro models that describe the data clusters. Initializing the models with random tran-

sition probabilities did not work since the space of random models is so large and often one model is much closer to the data than the rest and wins all the data points in the initial (E-step) segmentation.

3.4 Degenerate Models

Occasionally we saw cases were one (macro) model captures none of the data in the time series. There is no data to retrain the model so it needs to be reassigned. We tried several approaches to address this problem.

1) Look at each segment and its winning model. For each segment calculate the negative log-likelihood (NLL) of the data given its assigned model. We normalize each NLL by the number of data points so we can compare different sized segments. Choose the segment with the highest NLL (the worst fit to the data) and assign it to the missing cluster.

2) Split the model with the largest support by perturbing this model in two different directions. This is a common approach in data clustering, and is relatively easy to perform since one can do the perturbation along the major axis of the modeled data and effectively split the cluster into two along its major axis. We chose to perturb the transition matrix by a small random amount.

3) Find the cluster with the largest support (most data points) and split it at a random point in time. Train a new model for the original cluster with the first section of the data, and train a model for the missing cluster using the second half.

The last approach was the most successful, perhaps because the random segment sizes never came up with the same answer twice. Often the first attempt, in all the approaches above, fails and one model remains degenerate. The random segmentation approach often finds its way out of a locally degenerate situation.

3.5 Learning the Model Structure

There are two common means to build the necessary number of models and to decide how many models or clusters are necessary. Often in k-means a single model is learned from all the data. One model or cluster is added at each stage by splitting the largest cluster and retraining all models using the procedures described in Section 3.2. Alternatively, one can start with N random models and learn the right number of models all at once.

Meignier [11] suggests an approach where a single global model is split by removing a small portion of the data that best fits the model. This small portion is used to train a new model and the original model is trained on the remaining data. If the data is closely clustered then each iteration picks out the centroid of the data and builds the new model and leaves the remainder of the data to be modelled more poorly with the global model. This approach did not work well in our simulations compared to the random-segment approach.

3.6 Performance Evaluation

The approach described in this report is completely unsupervised; there is no reason that the (macro) model labels generated by our learned models should agree with the macro

labels used when generating the test data. We want to compare the structure of the learned model with the labeled data. There are information-theoretic approaches to finding the optimal mapping, but for small numbers of states we can use a brute-force approach to enumerate all possible label permutations and choose the one which gives the smallest decoding error.

4 Illustrative Example

In practice, we expect that the text content of all windows and dialog boxes displayed to the user will form the input signal for this work, but we have not instrumented our systems to capture this data yet. Instead, we illustrate the behavior of our algorithm using a multi-tasking web-browsing example. Using the log from a web proxy [9], we collected a sequence of uniform-resource locators (URLs) as a user looked for information on a series of three different topics: (1) PERL hash, (2) molecular biology hmm, (3) sudden oak disease. We used Google [1] to find appropriate pages on these topics. About 20 pages were selected from each topic in order, and then the three topics were revisited in order again.

For each URL that was logged, we used a text-only web browser to gather the contents of the page. We used simple heuristics based on the file suffix to remove uninteresting URLs (such as images and code) and then used simple heuristics based on the average distance between space characters to decide if the URL pointed to a web page containing text. The text on each web page is a single document for LSI analysis. In this experiment, the user visited a total of 155 web pages.

We used LSI to reduce the semantic feature vector from 8009 dimensions (the total number of distinct words, after removing stop words, in all documents) to a three-dimensional space. The resulting feature vectors are shown in Figure 3. Note the three clusters are distinguishable by drawing lines between the clusters, although there would be some errors with simple discriminators such as GMMs.

We modeled the data assuming a fully-connected three-state Markov model. We expected the model to use one state per semantic topic and properly segment the web-browsing data by topic. Each micro model was implemented using a two-component GMM. Each GMM estimates the probability that a (3D) point in semantic space is seen in this macro state.

Figure 4 shows the original and the reconstructed segmentation and macro-state sequences. The EM algorithm converged to the correct answer; each portion of the signal was assigned to the correct model although the labels are permuted. This reconstruction converged—reached a stable segmentation—after three iterations. Figure 5 summarizes the learned GMM micro models.

The macro model provides an important temporal constraint. In its simplest form, the self-loop probability suggests that a user stays in one task for a number of time steps before moving to a new task. Without this constraint, the model makes a locally optimal decision and is free to predict that at each time step the user jumps to a different task.

The time-series constraint is important because it smooths out noisy data. A user might visit a web page that is independent of task (i.e. the Google home page) or see a

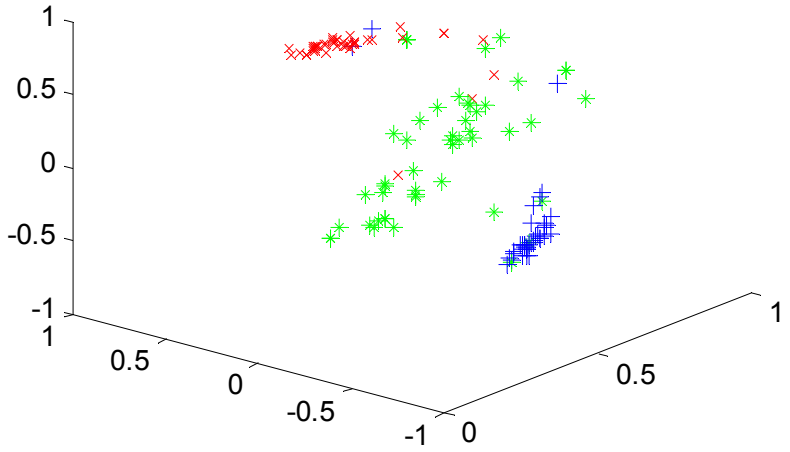

Fig. 3: Raw data in a 3D LSI feature space for the experiment described in Section 4. The data marked by 'x' represents PERL pages, '*' represents molecular biology, and "+' represents oak disease. Note the classes are not separable with simple decision surfaces.

dialog box that is common to all tasks (i.e. a print dialog). We want to ignore these common, information-free windows since they do not tell us anything about the task.

Figure 6 shows a typical segmentation learned by the hierarchical EM algorithm with no temporal constraint. The self-loop probability was set to 0.33 so that each state had a 33% probability of being used. In this example, the state labels match the original labels, but there are many single point errors. About 25% of these labels are incorrect when using the simple 2-component GMMs to represent each cluster probability. A more sophisticated model can make the proper distinctions and discriminate between these data. In this switching-task model of user behavior the macro model constrains the solution and allows a simple GMM to perform without errors (See Figure 4). The dif-

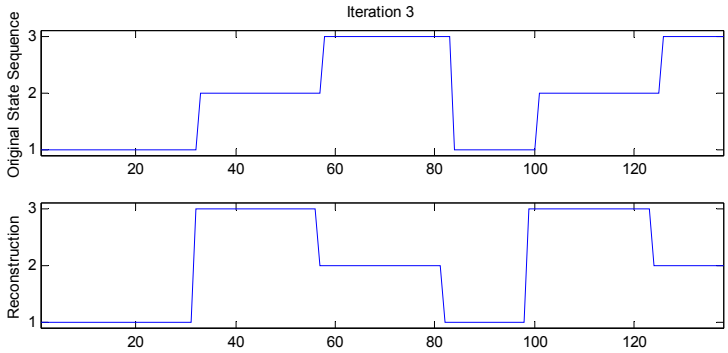

Fig. 4: The original and reconstructed sequence of user states. In the original data, the 3 different topics were encoded as: 1 is PERL, 2 is molecular biology, 3 is oak disease. The labels on the reconstructed state sequence are arbitrary, and can be permuted as they are in this example. With the ideal permutation, the reconstructed sequence matches the original exactly.

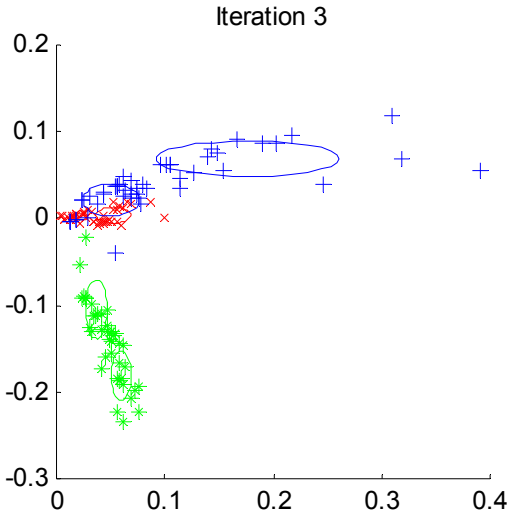

Fig. 5: Each of the three micro models are implemented with 2-component, 3-dimensional GMMs, which are summarized by the ellipses shown above. (Only the two of the three most important LSI directions are shown in this plot.)

ferences in the results shown in Figures 4 and 6 are due to the power of a global decision versus a local decision.

5 Future Work

In this paper, we reported simulation results for a single type of hierarchical model. Much more remains to be done. A thorough test and evaluation requires collecting more data of web browsing and ordinary computer use to try to characterize realistic user tasks. Attentive user interfaces (e.g., [10]) can be built around this sort of complex mul-

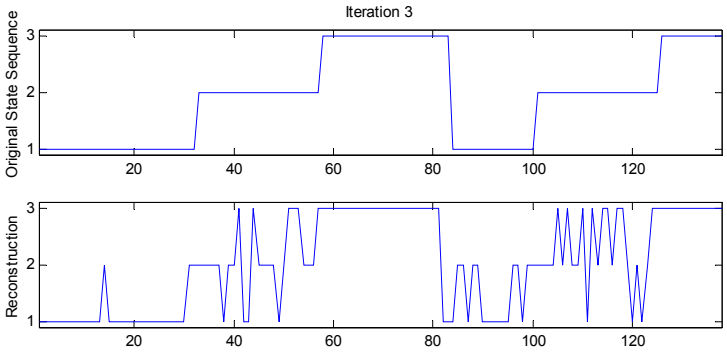

Fig. 6: Reconstrctured state sequence without the temporal constraint. This is equivalent to clustering the raw data into three clusters and thus the actions at many points in time are misclassified.

titasking user model, and can be evaluated in comparison to simpler text-based user models. Data from large numbers of computer users in large corporations can be aggregated and analyzed to begin to understand how people really spend their time, possibly informing the design of corporate applications and systems. By capturing user behavior in models that take account of multitasking, we can finally develop tools that support how people naturally work.

6 Acknowledgments

We appreciate the support we received from Myron Flickner, Daniel Russell and the anonymous reviewers.

References

1. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh World-Wide Web Conference,* 1998.
2. J. Budzik and K. J. Hammond. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 5th international conference on Intelligent user interfaces,* New Orleans, pp. 44–51, 2000.
3. A. Cypher. The Structure of Users' Activities. In Norman D. and Draper S. eds., *User Centered System Design,* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986, pp. 243–263.
4. B. D. Davison and H. Hirsh. Predicting Sequences of User Actions. In *Proceedings of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis 1,* AAAI Press, pp. 5–12, 1998.
5. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6), pp. 391–407, 1990.
6. E. Horvitz, et al. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of the 14th Conf. on Uncertainty in AI,* Madison, WI, pp. 256–265, 1998.
7. F. Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MA, 1998.
8. B. Laurel, editor. *The Art of Human Computer Interface Design.* Addison-Wesley, 1990.
9. P. P. Maglio and R. Barrett. Intermediaries personalize information streams. *Communications of the ACM,* 43(8), pp.96–101, 2000.
10. P. P. Maglio, C. S. Campbell, R. Barrett, T. Selker. An architecture for developing attentive information systems. *Knowledge-Based Systems* 14, pp. 103–110, 2000.
11. S. Meignier, Jean-Francois Bonastre and S. Igounet. E-HMM approach for learning and adapting sound models for speaker indexing. *2001: A Speaker Odyssey,* Crete, Greece, June 2001.
12. B. J. Rhodes. Margin notes: building a contextually aware associative memory. In *Proceedings of the 5th international conference on Intelligent user interfaces,* New Orleans, pp. 219–224, 2000.
13. S. Shi and A. S. Weigend. Markov gated experts for time series analysis: Beyond regression. In *Proceedings of IEEE International Conference on Neural Networks,* Houston, TX pp. 2039–2044.
14. B. Westphal, T. Syeda-Mahmood. On learning video browsing behavior from user interactions. *Proceedings of the Eleventh International World Wide Web Conference,* Honolulu, Hawaii, USA, 2002.