Combining Unsupervised and Supervised Machine Learning to Build User Models for Intelligent Learning Environments

by

Saleema Amin Amershi

B.Sc., The University of British Columbia, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

January 2007

Abstract

Traditional approaches to developing user models, especially for computer-based learning environments, are notoriously difficult and time-consuming because they rely heavily on expert-elicited knowledge about the target application and domain. Furthermore, because the expert-elicited knowledge used in the user model is application and domain specific, the entire model development process must be repeated for each new application.

In this thesis, we outline a data-based user modeling framework that uses both unsupervised and supervised machine learning in order to reduce the development costs of building user models, and facilitate transferability. We apply the framework to build user models of student interaction with two different learning environments (the CIspace Constraint Satisfaction Problem Applet for demonstrating an Artificial Intelligence algorithm, and the Adaptive Coach for Exploration for mathematical functions), and using two different data sources (logged interface and eye-tracking data). Although these two experiments are limited by the fact that we do not have large data sets, our results provide initial evidence that (*i*) the framework can automatically identify meaningful student interaction behaviors, and (*ii*) the user models built via the framework can recognize new student behaviors online. In addition, the similar results obtained from both of our experiments show framework transferability across applications and data types.

Table of Contents

List of Tables

List of Figures

Acknowledgements

Here, I would like to acknowledge those people whose help and support made this thesis possible. First, I would like to thank my supervisor, Cristina Conati, for taking me on as a Masters student. She encouraged and allowed me to experiment with my ideas early on in my course project, which, with her continuous help and guidance, eventually transformed into the work presented in thesis. Her influence on my academic career will last well beyond this degree. I would also like to thank NSERC and the Faculty of Graduate Studies for funding the research presented in this thesis.

I would like to thank my second reader, David Poole, for taking the time to read my thesis and for his helpful comments. I also want to thank Nando De Frietas for his teachings of machine learning which proved to be invaluable for my research.

I'm especially grateful to Heather, Kasia and Andrea for their advice, encouragement, and friendship through some of the hardest times during the writing of this thesis. I would also like to thank each of them for letting me bounce my ideas off of them on many occasions and for providing me with valuable feedback on my research. I also want to thank Heather in particular for everything I learned from her about user modeling, learning environments, conducting user studies and much more during our collaborations. And many thanks to Andrea and Kasia for making sure things were always under control, even during their break, and while I was halfway around the world.

I would also like to thank Alan Mackworth, David Poole, Guiseppe Carenini and the rest of the CIspace group for giving me my first opportunity to learn about the exciting world of research and academia.

Personally, I could not have come this far without the immeasurable support of my sister and best friend, Aliya, and my parents, Amin and Zaynul. My sister never failed to make me smile, even through my often irrational and ill-timed, personal and professional breakdowns. She truly is an amazing person and a wonderful friend. And my parent's heartfelt words and wisdom inspired me with the desire to learn at an early age, without which my academic pursuits would not be possible. I am forever grateful for their continual love, encouragement and support in every aspect of my life.

In addition, I want to thank my pseudo-family, Rekha, Sukhpreet, Dal, Nancy, Tasha, Sandy and "the girls" for giving me balance. I can always count on them to make me laugh, and be by my side when I cry. In particular, I want to thank Rekha for always for always understanding me and for her valuable advice, personally and professionally. I am also especially grateful to Dal for working day and night by my side during those last, and most intense, weeks of my thesis. She made that struggle a lot more bearable, and even sometimes fun. And finally, I want to thank all of the girls for always allowing me to rant about my frustrations, and for their consistent comfort and support.

And last, I have to thank Rav for encouraging (i.e., forcing) me to take those first AI courses which opened the doors to this amazing field and set me on the academic path that I'm on today.

SALEEMA AMERSHI

The University of British Columbia
January 2007

To my dad, mom, and sister.

Chapter 1

Introduction

1.1 Background and Motivation

In this thesis we propose a user modeling framework that uses both unsupervised and supervised machine learning to address two of the most cited difficulties of developing user models for computer-based learning environments (e.g., [13, 43, 73]): laborious effort required by application designers to construct models, and limited model transferability across applications.

The user model is a fundamental component of an *intelligent learning environment* (ILE), i.e., a computer-based system that can provide adaptive support for students much like how human educators can adapt their instructional styles and strategies so as to accommodate individual students and their changing needs. The user model guides the adaptation process by providing the ILE with an abstract representation of the learner in terms of relevant traits such as knowledge (e.g., [4, 16, 21, 54]), meta-cognitive ability (e.g., [18, 56]), learning behaviors (e.g., [9, 55]), learning style (e.g., [23]), and even affective state (e.g., [11, 35]). Some cognitive psychologists, e.g., [80], have conjectured that even human educators derive such models of students:

> "Two things are required for the teacher to [help and guide the student]: on the one hand, an adequate idea of where the student is and, on the other, an adequate idea of the destination. Neither is accessible to direct observation. What the student says and does can be interpreted in terms of a hypothetical model – and this is one area of educational research that every good teacher since Socrates has done intuitively."

Unfortunately, although the benefits of personalized computer-based instruction are well-recognized, so are the development costs, of which a considerable part is devoted to the user model [13, 58, 71]. This is especially true for *knowledge-based* user models, because they require eliciting the relevant domain and pedagogical knowledge from experts, a process that is often difficult and time consuming [36, 53]. Furthermore, pure knowledge-based approaches can typically recognize and interpret only expected student behaviors, and are unable to handle unanticipated ones. Thus, they tend to be suboptimal for novel applications for which real experts do not exist yet.

To circumvent the drawbacks of knowledge-based user models, some researchers have turned to the field of machine learning (e.g., [7, 11, 12]) to approximate functions that map observable student behaviors to different classes (e.g., the correctness of student answers). These functions can then predict the outcome of future student behaviors and inform adaptive facilities. However, this approach typically necessitates labeled data. When labels (e.g., student answers) are not readily available from the system, domain experts must resort back to manual labeling to supply them (e.g., reviewing recorded data and categorizing observed behaviors into relevant behavioral classes), which is again time-consuming and error prone.

1.2 Thesis Approach: A Statistical Pattern Recognition Approach to Building User Models for Intelligent Learning Environments

The user modeling framework we propose addresses the issue of cost-intensiveness by taking a statistical pattern recognition approach [48]. The general procedure for statistical pattern recognition is: data acquisition, processing, learning, and then testing [48]. Our framework defines this process specifically in the context of user modeling for ILEs. It uses unsupervised learning to automatically identify common learning behaviors and then applies supervised machine learning to these behaviors to train a classifier user model that can inform an adaptive ILE component. A key distinction between our modeling approach and knowledge-based or supervised approaches with hand-labeled data is that human intervention is delayed until after unsupervised machine learning automatically identifies behavioral patterns. That is, instead of having to observe individual student behaviors in search of meaningful patterns to model (e.g., student errors or misconceptions as in [20]) or to input to a supervised classifier (e.g., actions indicating motivational state as in [35] or instances of students misusing an existing ILE as in [7]) the developer is automatically presented with a picture of common behavioral patterns that can then be analyzed in terms of learning effects. Expert effort is potentially reduced further by using supervised learning to build the user model from the identified patterns.

In addition to reducing developer workload, our approach also facilitates transfer across different applications and data types. In this thesis, we demonstrate transferability by applying our user modeling framework to

(i) two different learning environments, the CIspace Constraint Satisfaction Problem (CSP) Applet for teaching AI algorithms [3] and the Adaptive Coach for Exploration (ACE) for teaching mathematical functions [19], and

(ii) two different data sets, one involving interface actions only and another involving both interface actions and eye-tracking data.

Both the CIspace CSP Applet and ACE are exploratory learning environments (ELEs) that are designed to support free, student-led exploration of a target domain with the premise that active discovery and construction of knowledge may promote deeper understandings than more controlled instruction [65] [15]. We chose ELEs as testbeds for our framework for two main reasons. First, previous research has shown the value of providing adaptive support for student exploration in these environments [72]. For example, while some research has shown that ELEs may enhance the learning experience for more adept or active students, passive students or students who find such unstructured environments difficult to navigate effectively may not learn well with them [72], and therefore may benefit from additional intelligent guidance on how to best use the ELE to learn.

The second reason is that traditional user modeling approaches are particularly difficult for ELEs. In [71], John Self highlighted the infeasibility of developing knowledge-intensive user models for complex or ill-structured domains, such as ELEs. He demonstrated that even for a relatively simple mathematical integration problem, building a comprehensive knowledge-based user model that could diagnose any student solution, would require analyzing an intractable number of possible solution paths. Some researchers have opted to design more restrictive learning environments that constrain student behavior and thereby minimize the number of elements to model (e.g., [4, 27]), but this type of restrictive learning conflicts with the very principle of open learning that ELEs are intended to support. In addition, because ELEs are a relatively novel learning paradigm, little practical knowledge exists about optimal learning strategies within these systems, increasing the difficulties of applying knowledge-based modeling approaches. Supervised machine learning approaches to user modeling are also unappealing for ELEs. ELEs provide no obvious output labels (e.g., in ELEs there is no notion of correctness and so clear labels such as correct/incorrect answers to questions are unavailable), and therefore output labels must be manually provided by application or domain experts. However, because the space of possible interaction behaviors within ELEs can be very large, manually observing distinct behaviors and interpreting them in terms of learning effects is especially difficult, even for experts.

1.3 Thesis Goals and Contributions

The overall objective of this research is to show the value of a statistical pattern recognition approach to user modeling in ILEs as an alternative to more knowledge-intensive approaches that tend to be complex or time-consuming, especially in highly unstructured exploratory environments. In order to meet this objective, we define the following goals of this thesis:

1. To outline a data-based user modeling framework that addresses the practical challenges faced in applying knowledge-intensive approaches to user modeling in ILEs, and particularly in ELEs.

2. To evaluate our proposed framework by answering the following questions:
    1. How well does the framework automatically identify meaningful learning behaviors?
    2. How well does a user model built via the framework perform at classifying new users online?

iii. How well does the framework transfer across different applications and data types?

The work in this thesis contributes to research in user modeling. Traditional approaches to user modeling, especially for ILEs, have been very knowledge-intensive. Many of these approaches have relied on the lengthy and laborious process of hand-constructing user models based on expert knowledge or intuition. Such approaches are application and domain specific and therefore do not facilitate transfer to new applications or domains. Other approaches to user modeling for intelligent learning environments have used machine learning algorithms to help reduce developer workload. However, unlike our approach that uses both unsupervised and supervised machine learning, the majority of these have only made use of supervised algorithms that require labeled data. In certain applications, such as the ELEs that we experiment with in this research, labels are not readily available and therefore must be supplied by hand. This is again time-consuming for model developers. Very little research has been reported on using unsupervised machine learning to help build user models for ILEs. The work presented in this thesis is a step towards this direction.

1.4 Outline

The rest of this thesis is organized as follows. In Chapter 2 we review previous research on knowledge and data-based approaches to user modeling. We also discuss some of the approaches that have been taken for user modeling in ELEs. Chapter 3 outlines our proposed user modeling framework and describes how we can evaluate user models built via our framework with limited available data. In Chapters 4 and 5 we apply and evaluate our modeling framework on two separate ELEs. In Chapter 4, we apply the framework to the CIspace Constraint Satisfaction Problem (CSP) Applet for demonstrating an Artificial Intelligence algorithm for solving CSPs [3]. In Chapter 5, we demonstrate the process on the Adaptive Coach for Exploration (ACE) for mathematical functions [19]. Then, in Chapter 6 we compare the results we obtained from our two experiments and discuss the limitations of our framework and this research. In Chapter 7, we discuss avenues for future work. And finally, in

Chapter 8 we conclude with a summary of the research and contributions of this thesis.

Chapter 2

Related Work

In this chapter, we review common approaches to user modeling, concentrating on the model development processes used in these approaches for comparison with our approach outlined in Chapter 3. These approaches range from knowledge-based approaches where models are hand constructed by experts, to supervised data-based approaches using machine learning along with system or expert provided labeled data, to unsupervised data-based approaches using machine learning with unlabeled data. Our approach falls towards the end of this spectrum as we use an unsupervised machine learning component to provide labels for supervised machine learning. This review focuses primarily on user modeling for learning environments (also called 'student modeling' in the literature) but also includes some relevant work on user modeling in other application areas. After reviewing previous approaches to user modeling in general, we then discuss relevant work on user modeling for exploratory learning environments (ELEs).

2.1 Knowledge-Based Approaches

By 'knowledge-based' user modeling we mean that the methods used to build an underlying representation of the user require explicit knowledge elicited from application and domain experts. In regards to intelligent learning environments (ILEs), estimates of the development time for a knowledge-based approach are in the range of 100 hours for just one hour of instruction [58], of which a considerable part is devoted to developing the user model. In this section we describe the development processes used in some of the most common knowledge-based approaches to user modeling in learning environments.

A common type of knowledge-based user model is the *overlay* model, which represents a student relative to a representation of ideal expert behavior or domain knowledge in the form of elements such as concepts, facts and rules (e.g., [4, 16, 21]). An overlay model can be used to expose missing student knowledge or deviations from ideal behavior by marking model elements as known, unknown or even partially known by the student based on observed interactions with a system. An adaptive system can then use this information to provide personalized support targeting the model elements in question.

Developing overlay models requires, first, the often difficult problem of recruiting expert participation [58]. Once experts are recruited, then begins the lengthy process of acquiring their knowledge in the form of large sets of model elements [77] and actually implementing the model. The knowledge acquisition and implementation process can be divided into three general stages [10]:

- *Knowledge Elicitation.* When the basic conceptual structure of the knowledge model is acquired from the expert (e.g., through systematic interviews) and implemented by the developer.
- *Knowledge Refinement*. When the initial model is debugged and extended by the expert (e.g., by testing the model and analyzing the results to identify errors).
- *Knowledge Reformulation*, when the model is adjusted to accomplish tasks (e.g., diagnose missing student knowledge) more efficiently.

GUIDON [16] is an example of an adaptive system that uses an overlay model to teach students how to diagnose infections diseases. The model consists of over 450 rules and facts about different diseases and their symptoms which took over four years of interactions with physicians to develop. Another example is the model used in Anderson et al.'s [4] LISP tutor, a learning environment for solving LISP programming problems, which consists of 500 if-then production rules detailing individual LISP programming steps. The authors of the LISP tutor stated that: "A major fraction of the tutor development has gone into developing [production] rules. It is a difficult task to define a set of rules that will solve a large class of problems." Furthermore, for even a simple problem the total number of solution paths that can be modeled can be intractable [71], and therefore there is no guarantee that even a large set of expert-defined model elements for a given domain will be complete.

Another knowledge-based approach to user modeling is to develop a *bug library* (e.g., [2, 20]). While the overlay model represents ideal behavior or correct expert domain knowledge, building a bug library requires experts to catalog as many erroneous user behaviors or misconceptions about a given domain (i.e., bugs) as possible. User behaviors can then be compared to the bug library in order to identify errors or misconceptions and make appropriate system adaptations.

As with the overlay approach, defining the bug library is time-consuming for experts because possible bugs must be defined by intuition (comparable to the process of defining model elements in overly models described above), or identified through manual analysis of numerous examples of user behaviors [77]. For instance, over 100 bugs were identified about arithmetic subtraction in Burton and Brown's DEBUGGY system [20] through manual analysis of thousands of student tests taken on subtraction [77].

Murray and Woolf [58] did a case study in developing an ILE for static forces in physics in order to analyze the development process and quantify the time required for each step in the process. They used a knowledge-based approach to developing a user model for the ILE that included both expert and buggy knowledge. They found that in developing the environment (which consisted of approximately six hours worth of instruction), the domain expert (a physics tutor) spent 277 hours performing tasks such as designing and debugging the user model, while two knowledge-base managers (responsible for implementing the user model) spent 240 hours performing tasks such as coding and debugging the model. In addition to being time-consuming and laborious, overlay and bug library approaches are particularly ill suited for ELEs where there is no notions of correctness or well-developed theories of exploratory learning to help guide experts in defining correct or faulty knowledge and behaviors.

Recently, Bayesian networks have also been employed for user modeling because of their capacity to manage uncertainty in the modeling task in a principled and mathematically sound way (e.g., [30, 57]). However, hand-coding a Bayesian network user model also requires expert elicited information and so this approach faces the same drawbacks as the overlay and bug library approaches described above. Here, experts must again define large sets of facts, rules and behaviors (represented as network nodes), as well as the relationships between these elements (represented as links between network nodes). In addition, the Bayesian network formalism requires encoding multi-valued prior and conditional probability tables for each network node. Appropriate values for these probability tables can be subjectively estimated by experts based on theoretical or prior knowledge about the domain or the user, but this can be difficult especially when dealing with internal and unobservable user states or complex relationships [55]. In Conati et al.'s ANDES system [28, 30] for solving physic problems for example, Bayesian networks were developed containing 200 interconnected nodes for simple physics problems and up to 1000 nodes for more complex problems. Bayesian networks may be even larger and more difficult to define for unstructured ELEs than for highly structured problem-solving applications such as ANDES.

2.2 Supervised Data-Based Approaches

To avoid what is sometimes called the 'knowledge bottleneck problem' [84] of knowledge-based approaches to user modeling (i.e., the laborious effort and extensive time required of experts to hand-construct user models), researchers have recently started investigating 'data-based' approaches for automatically learning user models from example user data. Most of these data-based approaches to user modeling have employed supervised machine learning techniques that require labeled example data [81]. From the labeled example data (i.e., the 'training data'), supervised machine learning techniques (e.g., linear regression and classification) learn mappings from input data (e.g., interface behaviors) to output labels (e.g., learning outcomes or user states). Based on the learned mappings, the models can then take newly observed data (i.e., the 'test data') and predict the output labels for that data. The predicted information can then be used to guide system adaptations. Here we describe some common approaches to building user models from data.

Many researchers taking data-based approaches to user modeling view the supervised machine learning algorithms as 'black boxes' [12] that take in labeled data and produce the user model. Therefore, the key steps in the model development process concern specifying the input data and output labels for the learning algorithm to use. Input data is typically behavioral (e.g., interface actions) and therefore often easily obtainable from data logs of application use [81, 84]. Once data is obtained, it must be processed so that only the relevant features of the data are used as input to the learning algorithm. These data collection and processing steps are equivalent to the initial steps in our proposed user modeling framework (see Chapter 3). However, our approach deviates from supervised data-based approaches in the need for labels for the input data.

There are two general methods typically used to obtain labels for input data. In the first method, data labels are obtained directly from the system. In the second, labels are manually supplied through expert observations and analyses. Because the later method requires expert interpretation of the data, it could be considered a knowledge-based approach to user modeling. However, this method is still data-intensive and relies on machine learning algorithms to build the models as opposed to hand constructing them. Therefore, we categorize this method as being a data-based approach.

The question-based, AnimalWatch ILE for teaching arithmetic [12] is an example of using a data-based approach with system-provided data labels to learn user models. Input data was obtained from data logs of actual students using the system to learn. The input data consisted of snap shots of the current state of the system, including the type and complexity of the current problem being solved. For one user model, the output label for each snap shot was the correctness of the student’s answer. For another user model, the output label was the time taken to solve the problem. It should be noted that answer correctness was assessed by a previously developed knowledge-based overlay user model [5], however, for the purposes of the research in [12], correctness (as well as time) was directly extracted from the log files and therefore this can be considered a system-provided data label. The input data and corresponding output labels were then used to train supervised linear regression user models that could predict either the correctness of a new student’s response or recognize when the student is having difficulty (i.e., if their predicted response time is over a predefined threshold indicating potential confusion).

Similarly for the Reading Tutor [11], non-linear regression was used to model student engagement levels based on student response times in answering multiple choice questions about reading comprehension. That is, a non-linear regression function was trained with input-output sample data where the input data consisted of logged response times, question difficulties and student performance histories, and the output data labels were the correctness of student answers as determined by the system (since reading questions and their corresponding answers were generated automatically by the system by randomly removing words from sentences and then asking students to determine the removed word). The regression function could then predict how likely future student answers are at being correct and a manipulation of the function could be carried out to compute student engagement levels. And in the CAPIT system [55], the parameters of a Bayesian network user model were also learned from system-labeled data. In this case the input data was logged behavioral data and, again, the output labels were the correctness of student responses to canned punctuation and capitalization questions.

Several other examples of user modeling based on supervised data-based approaches with system-provided labels exist outside of educational applications. An example is the MailCat system [70] for automatically organizing user emails. MailCat uses email text as input and user-chosen folders as output labels to train a text classifier to predict the target folder of incoming emails. Here again, obtaining the output labels require no additional effort from experts or model developers. Some recommender systems also use supervised data-based approaches with system provided labels to model the user. For example, the Syskill and Webert recommender system [63] uses a naïve Bayesian classifier user model in order to make interesting web site recommendations to the user. The input data consists of visited web pages represented by the most informative words on the page. Each web page is explicitly labeled by the user in terms of its interestingness (by having the user rate the web page with a 'thumbs up' for interesting and 'thumbs down' for not interesting). Letizia [53] is another system for recommending interesting web sites to users. Here, the output labels are inferred by the system using heuristics (e.g., saving a web page or spending a lot of time reading a web page would indicate user interest) rather than being explicitly provided by the user.

Although supervised data-based approaches that make use of system-provided output labels can significantly reduce the time required of application and domain experts in building user models, when output labels are not readily available from the system (e.g., answers to questions or user ratings), such as in ELEs, then data labels must be provided by hand. The work of Baker, Corbett and Koedinger [7, 9] is one such example of using expert-labeled data in a supervised data-based approach to user modeling. In this research, the authors observed students using an ILE for solving scatterplot analysis problems watching for the occurrence of specific types of behaviors detrimental for learning that they named “gaming-the-system behaviors.” The process of observing students first required predefining which user behaviors signal attempts to game the system (e.g., systematic trial-and error). Defining these behaviors could be done because the learning environment supported a highly structured, well studied, problem-solving type of pedagogical interaction. This is harder for ELEs where effective and detrimental interaction behaviors are less obvious. Once gaming behaviors were defined, two researchers observed 70 students using the system. The researchers made their observations through their peripheral vision in order to reduce any behavioral changes that could result from students feeling watched. A total of 563 observations were made, taking 20 seconds per observation for a total of approximately three full hours worth of observations. The gaming observations corresponded to labels of the input data (logged interface events, such as user actions and latency between actions). The labeled data was then used to train a regression model that could predict instances of gaming with relatively high accuracy.

In the MOODS ILE for learning Japanese numbers [35], the authors asked experienced tutors/teachers to manually label recorded data (video and screen capture footage) in terms of pre-defined motivational variables (e.g., effort and satisfaction) in order to produce mappings from observable interface actions to the internal motivational states of students. These mappings could then be used to detect motivational states of new users based on their interaction behaviors. Baker et al. [8]

estimate that these kinds of human observations take between 1 and 10 minutes per classification. So this is again time-consuming and can be error prone.

2.3 Unsupervised Data-Based Approaches

The initial user modeling steps (data collection and preprocessing) in unsupervised data-based approaches are similar to that of supervised data-based approaches; however, because unsupervised machine learning techniques do not require output labels to learn models from data, these approaches avoid the high costs of obtaining labels, especially through manual observations.

Most research on unsupervised data-based approaches to user modeling has been for non-educational applications. For example, collaborative filtering (CF) systems employ unsupervised learning techniques to model user preferences and make item recommendations based on user similarities. The research in [62] is an example of a CF system that can recommend interesting websites or personalize search query results based on a user's navigation or query behavior in relation to the behavior of similar users. This is done by automatically grouping similar users according to their navigation or query behavior using unsupervised learning algorithms such as clustering, and then determining which group a new user is most similar to in order to make appropriate recommendations or adaptations. Similarly, other research has demonstrated the use of unsupervised learning on (*i*) words in a document to model and automatically manage emails based on a user's organization of previous emails [50]; (*ii*) frequencies of web pages access to automatically personalize and adapt web sites for new users based on similar users' preferences [64].

In contrast, research on using unsupervised machine learning for user modeling in educational systems remains rare [73]. A notable exception is MEDD [74] which uses unsupervised learning to discover novel classes of student errors to Prolog programming problems. Student programming solutions are first compared to a data base of correct solutions to determine discrepancies between the students' actual solutions and their intended solutions. Discrepancies are then automatically grouped based on their similarities using an unsupervised pattern recognition algorithm (clustering) in order to define common errors and build bug libraries. Like the expert-built bug libraries in knowledge-based approaches to user modeling (see Section 2.1), MEDD's bug libraries can be used to identify misconceptions in a new student's solution and then provide adaptive feedback targeting those misconceptions. Our approach to user modeling differs from this in that we are modeling student interaction behaviors in unstructured environments with no clear definition of correct behavior instead of static student solutions and errors.

The research in [46] and [76] are also related to ours because both works use statistical pattern recognition techniques, although they do not actually build user models. DIAGNOSER [46], like MEDD, uses unsupervised machine learning to discover errors in static student solutions to physics questions. More similar to what we do, [76] uses clustering on interface action frequencies, to detect behavioral patterns in an environment for collaborative learning. Our work differs in that we use higher dimensional data including action latency, measures of variance and gaze information. Furthermore, in both [46] and [76] the resulting patterns are given to instructors who can then use them to tailor instruction, whereas we take this process one step further to automatically build a user model. Our research is also broader because we show transfer of our user modeling approach across applications and data types.

2.4 User Modeling in Exploratory Learning Environments

In this section we review previous work in user modeling for exploratory learning environments (ELEs).

Ecolab [54] is an ELE for primary school students to explore the relationships between different organisms in a simulated ecosystem. The authors used an overlay knowledge-based approach (see Section 2.1) to build a Bayesian network user model that could inform the system's adaptations. Expert knowledge was extracted from several science textbooks on the subject [54] and manually encoded in the user model.

Bunt et al. [19] also used a knowledge-based approach and hand-constructed a complex Bayesian network to model effectiveness of student exploration in supporting the understanding of mathematical functions in an earlier version of one of the two ELEs that we use as a test-bed in our research (Chapter 5), the Adaptive Coach for Exploration (ACE). Model construction required enumerating all possible sets of parameters that students could explore for each type of mathematical function (called the possible exploration cases), specifying the relationships between the exploration cases and the mathematical concepts they aim to illustrate, and manually defining multi-valued probability tables for the Bayesian network nodes using prior knowledge or estimations from experts. Though the model can handle uncertainty in a principled way and was shown to be quite successful in providing students with personalized help in exploration [19], like all knowledge-based approaches to user modeling, this entire time-consuming and knowledge intensive process would have to be repeated for each new application.

In [56], the authors augmented the original user model for ACE [19] using a supervised data-based approach (see Section 2.3) in order to model the meta-cognitive skill of self-explanation (the process by which a student generates explanations to one-self to clarify instructional material [24]). The authors used time and eye-gaze information to represent student attention patterns during interaction ACE and showed that modeling these patterns improves assessment of student self-explanation. This in turn improves model assessment of the learning effectiveness of student exploration. The authors conducted user studies to collect interaction data from students using ACE that two experimenters subsequently hand-annotated for instances of self-explanation by analyzing audio-video footage in addition to the logged data. The conditional probability tables for the new self-explanation nodes added to the Bayesian network user model (i.e., discretized time and binary gaze-shift nodes) were set based on frequencies of the manually observed patterns in the interaction data. In addition to needing manually labeled input data, the authors were only able to use the portion of the original data that the two experimenters agreed upon to augment the model, while the rest was discarded due to lack of inter-coder reliability.

Gorniak and Poole [43] also use a data-based approach to automatically learn a user model for the other ELE that we experiment with in our research, the CIspace

Constraint Satisfaction Problem (CSP) applet (see Chapter 4). They used observable interface actions for both the input and output data used to train a stochastic state space user model that could be used to predict future user actions. This approach would be considered a supervised data-based one with system-provided output labels (i.e. the future interface actions). Our work with the CIspace CSP applet differs in that we use an unsupervised data-based approach to map observable input data into unobservable user states (i.e., learning). Furthermore, unlike our approach, this approach does not allow for the assessment of the quality or relevance of the predicted actions for the interaction goals.

Chapter 3

User Modeling Framework

Understanding and detecting common learning behaviors is important for providing students with adaptive support in exploratory learning environments (ELEs), especially for those students who tend not to learn well with unstructured and unguided systems. However, this is very difficult due to the open interaction that ELEs are designed to support, making it hard to foresee which of the many possible student interaction behaviors may be beneficial or detrimental to learning. Of the few existing approaches to this problem, most have been very knowledge intensive, relying on expert analysis of the target ELE, instructional domain and learning processes (e.g., [19, 54, 57]), or on extensive observations of students (e.g., [56]). Because these approaches are application specific, they are difficult to generalize to other systems and other domains.

In this chapter we describe our proposed data-based framework for developing user models for computer-based learning environments, and particularly for ELEs. This framework does not assume domain or application expertise, although some prior knowledge is necessary in order to perform certain activities. Therefore, usage of this framework should reduce the time and effort often required of domain, application and model experts to build user models for intelligent learning environments (ILEs). Furthermore, the data-based approach, making use of both unsupervised and supervised machine learning algorithms, facilitates transferability across applications, as long as the developer has access to data logs of student interactions with the learning environment of interest.

A version of this chapter has been accepted for publication:
Amershi, S. and Conati, C. (2007) Unsupervised and Supervised Machine Learning in User Modeling for Intelligent Learning Environments. To Appear in the Proceedings of the 10th International Conference on Intelligent User Interfaces.

Figure 3.1 shows the architecture of our proposed user modeling framework, which divides the modeling process into two major phases: offline identification (Section 3.1) and online recognition (Section 3.2). In the offline phase, raw, unlabelled data from student interaction with the target environment is first collected (Section 3.1.1) and then preprocessed (Section 3.1.2). The result of preprocessing is a set of feature vectors representing individual students in terms of their interaction behavior. These vectors are then used as input to an unsupervised machine learning technique, called 'clustering,' that groups them according to their similarity (Section 3.1.3). The resulting groups, or 'clusters', represent students who interact similarly with the environment. These clusters are then analyzed by the model developer in order to determine which interaction behaviors are effective or ineffective for learning (Section 3.1.4). In the online phase, the clusters identified in the offline phase are used directly in a classifier user model (Section 3.2.1). The user model's classifications and the learning behaviors identified by cluster analysis can then be used to inform an adaptive ILE component that can encourage effective learning behaviors and prevent detrimental ones.

In the next two sections (Sections 3.1 and 3.2), we detail the two phases supported by the framework including describing the algorithms we chose to complete these phases. Then in Section 3.3 we explain how we evaluate the user models that we developed for both of our experiments (see Chapters 4 and 5) with limited data.

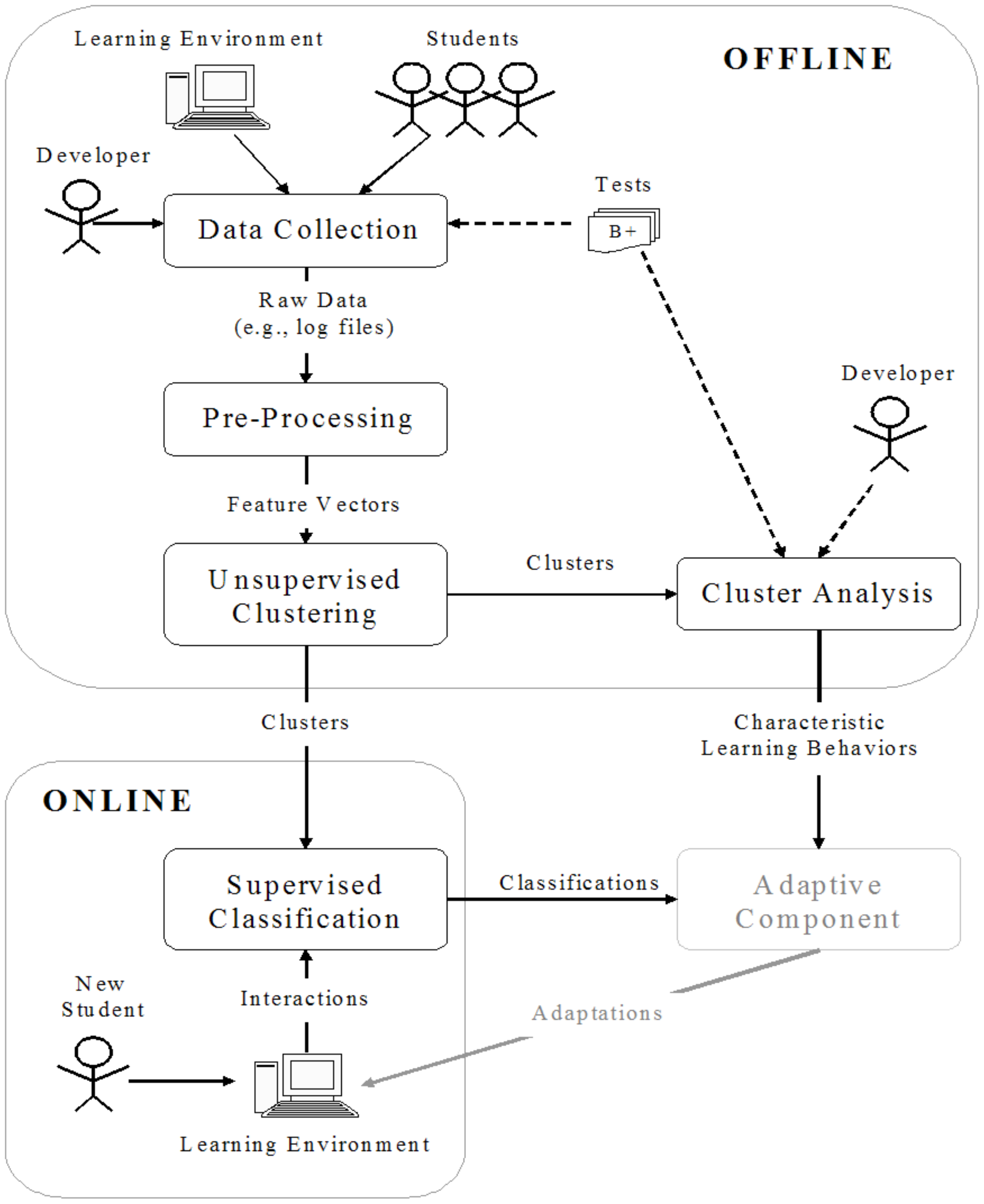


Figure 3.1. User modeling framework. Dotted lines represent optional input. Grayed out elements are outside of the framework.

3.1 Offline Identification

The first phase of our modeling framework takes a statistical pattern recognition approach to automatically identify distinct student interaction behaviors in unlabeled data. The rest of this section describes the different steps of the offline phase which is outlined at the top of Figure 3.1.

3.1.1 Data Collection

The first step in the offline phase is to log data from students interacting with the target learning environment. Here, the developer requires knowledge (or a catalog) of all possible primitive interaction events that can occur in the environment so that they can be logged (see in Figure 3.1 the solid arrow from 'Developer' to 'Data Collection'). In addition to interface actions, logged data can include events from any other data source that may help reveal meaningful behavioral patterns (e.g., an eye-tracker).

An optional, but highly desirable, additional form of data to collect is tests on student domain knowledge before and after using the learning environment (see the dotted arrow in Figure 3.1 from 'Tests' to 'Data Collection'). The purpose of these tests is to measure student learning with the system which can facilitate the cluster analysis step of our framework, as we will see below.

3.1.2 Preprocessing

Clustering operates on data points in a feature space, where features can be any measurable property of the data. Therefore, in order to find clusters of students who interact with a learning environment in similar ways, each student must be represented by a multidimensional data point or 'feature vector'. The second step in the offline phase is to generate these feature vectors by computing low level features from the data collected. We suggest features including (a) the frequency of each interface action, and (b) the mean and standard deviation of the latency between actions. The latency dimensions are intended to measure the average time a student spends reflecting on action results, as well as the general tendency for reflection (e.g., consistently rushing through actions vs. selectively attending to the results of actions). We compute these features from the data collected in both of our experiments (see Chapters 4 and 5). In our second experiment (Chapter 5) we also include features extracted from eye-tracking data (i.e., eye gaze movements) to demonstrate that a variety of input data can be used in our modeling framework.

In high-dimensional feature spaces, like the one in our second experiment, natural groupings of the data are often obscured by irrelevant features. Furthermore, as the number of dimensions increases, pattern recognition algorithms become increasingly more susceptible to the "curse of dimensionality" [14]. The "curse of dimensionality" refers to the need for exponentially larger data sets to compensate for the data sparsity that results when the number of dimensions, or the volume of space containing the data, increases. Data sparsity can make models learned by pattern detection algorithms prone to overfitting (i.e., low generalizability of the model to new and unseen data) [78]. Therefore, especially with limited data, determining the most salient features and removing noisy or irrelevant ones, called 'feature selection', can significantly improve results of the subsequent learning algorithm.

Dimensionality reduction can also reduce the computational costs of the learning task [47]. While in this research computational costs are of low impact because we are developing user models offline, if we wanted to incrementally update, or retrain, a model online as more data is accumulated, then reducing computational costs would be beneficial.

Prior domain or application knowledge can help guide manual feature selection, but estimates of feature utility are often unavailable or inaccurate leading to laborious trial-and-error evaluations of the features [47]. A substantial number of algorithms for automatic feature selection have been developed for supervised machine learning, but only recently have researchers started investigating principled ways of selecting features in an unsupervised setting (e.g., [22, 34, 41]). To avoid the effort and potential inaccuracies of manual feature selection, in our second experiment we employ an entropy-based unsupervised feature selection algorithm presented in [32]. For reference, we briefly outline this algorithm here.

First, we rank each of the D candidate features according to the entropy induced by the removal of that feature from the entire set of N available feature vectors. Entropy measures the amount of disorder in the data, and so the more entropy produced by the removal of a given feature, the more necessary it is to retain that feature for clustering. The entropy produced in the data by the removal of each individual feature, d, is computed in the following way:

$$E_d = -\sum_i \sum_j (S_{ij} \log S_{ij} + (1 - S_{ij}) \log (1 - S_{ij})) \qquad (3.1)$$

where $S_{ij} = S(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $i,j = 1 \ldots N$, is the similarity between multidimensional feature vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the space $\mathbb{R}^{D-1}$ (i.e., excluding dimension d). Similarity is given by:

$$S_{ij} = \exp(-\alpha \| \boldsymbol{x}_i - \boldsymbol{x}_j \|) \text{ and } \alpha = -N \ln 0.5 / \sum_i \sum_j \| \boldsymbol{x}_i - \boldsymbol{x}_j \| \qquad (3.2)$$

where $\|\cdot\|$ denotes the (normalized) L_2 norm or the Euclidean distance between feature vectors in the multidimensional feature space. This formulation essentially assigns low entropy to nearby feature vectors (such as those that should belong to the same cluster) or distant feature vectors (such as those belonging to well-separate clusters), and high entropy otherwise.

Next, we run forward selection on the ranked features. That is, incrementally larger subsets of features are evaluated in terms of their performance in clustering, and the subset that maximizes the quality of the clusters produced is selected as the final feature set. Cluster quality is defined as having maximum between-cluster variance and minimum within-cluster variance (see Equation 3.5 below).

Note that because this feature selection algorithm must execute clustering on each candidate feature subset in order to assess cluster quality, it returns both a reduced feature set and the clusters resulting from clustering on that feature set. Therefore, clustering need not be performed again when using this automatic feature selection algorithm. When no feature selection is performed, or feature selection is done manually, then clustering must be carried out as a separate step to determine behavioral patterns, as described next.

3.1.3 Unsupervised Clustering

After forming feature vector representations of the data, the next step in the offline phase is to perform clustering on the feature vectors to automatically discover patterns, or clusters, of interaction behaviors in terms of these features. Clustering works by grouping feature vectors by their similarity, where here we define similarity to be the Euclidean distance between feature vectors in the normalized feature space. Normalization is necessary to eliminate feature bias due to scaling differences along the feature dimensions. We use the z-score transformation for data normalization which transforms the data so that each dimension has a mean of 0 and standard deviation of 1. This makes the different dimensions comparable because the feature values in each dimension are expressed by the magnitude of their deviations from the same mean.

We chose a partition-based algorithm called *k*-means [37] for clustering in both of our experiments. While there exists numerous clustering algorithms (see [47] for a survey) each with its own advantages/disadvantages, we chose *k*-means because it is popular in pattern recognition applications and intuitive to understand. Furthermore, the *k*-means algorithm scales up well if a large amount of data is available because the time complexity is linear in the number of feature vectors.

K-means takes as input feature vectors and a user-specified *k* value corresponding to the number of clusters that should be returned. Typically the *k* value is determined by intuition about the data or through cross-validation [37, 51]. Initially, *k* feature vectors are randomly selected to be the current cluster centroids. The remaining feature vectors are then assigned to the cluster whose current centroid minimizes the Euclidean point-to-centriod distance metric. After all feature vectors are assigned to a cluster, new cluster centroids are computed from these groupings. The process then repeats for a given number of iterations or until there are little or no changes in the clusters.

K-means converges to different local optima depending on the selection of the initial cluster centroids and so several trials are typically executed and the highest quality clusters are used. High quality clusters are commonly defined as having maximum between-cluster variance and minimum within-cluster variance. Variance is measured by the between and within-cluster scatter matrices respectively:

$$P_b = \sum_k (\boldsymbol{c}_k - \boldsymbol{c})\,(\boldsymbol{c}_k - \boldsymbol{c})^{\mathrm{T}} \tag{3.3}$$

$$P_w = \sum_k \sum_i (\boldsymbol{x}_{ik} - \boldsymbol{c}_k)\,(\boldsymbol{x}_{ik} - \boldsymbol{c}_k)^{\mathrm{T}} \tag{3.4}$$

where $k = 1\ldots K,$ with K being the number of clusters, $\boldsymbol{c}_k$ is the centroid of cluster k, and $\boldsymbol{c}$ is the centroid of all the data. We combine (3.3) and (3.4) based on Fisher's criterion [40] in discriminant analysis which reflects the ratio of between to within-cluster scatter. We compute the trace of the resulting matrix to form a single measure of quality as

$$\mathrm{trace}(P_w^{-1} P_b) \tag{3.5}$$

where the higher the value, the better the quality.

While *k*-means is intuitive to understand and efficient for large data sets (and therefore may be favorable for online educational technologies that have the potential to log large amounts of data), it does have limitations. First, *k*-means assumes feature dimensions are independent, whereas this may not always be the case in applications of our framework. For example, in application of our framework we suggest including the frequency of interface actions and the mean and standard deviation of the latency between actions as features (see Section 3.1.2), yet these features are not necessarily independent. However, violation of this independence assumption usually does not affect the quality of the resulting clusters [52]. Second, *k*-means assumes the clusters are elliptical, and would be unsuccessful at identifying more complex cluster shapes. For example, Figure 3.2 shows two concentric circles of data points. *K*-means with k set to 2 would likely return the sub-optimal clustering shown in Figure 3.3 instead of the clustering shown in Figure 3.4 which better represents the data. In this case, a more computationally expensive hierarchical algorithm [37] may be best. *K*-means also produces hard assignments of feature vectors to clusters, whereas it may be beneficial for an adaptive learning environment basing its tutorial decisions on cluster membership to know the uncertainty in the assignments. Here, a probabilistic version of *k*-means called Expectation Maximization [37] may be more appropriate. Thus, the choice of clustering algorithm should be informed by properties of the data or the type of application being studied. Although we use *k*-means as proof of concept throughout this research, we expect that other choices of clustering algorithms can be substituted for *k*-means in our proposed modeling framework.

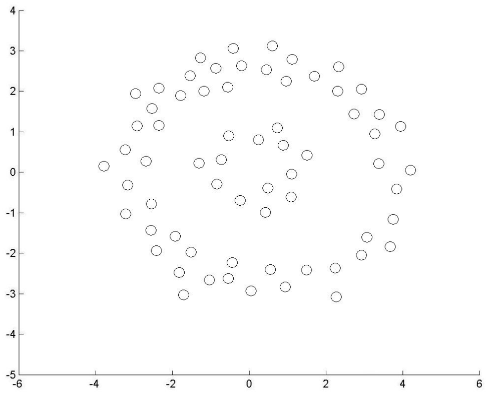


Figure 3.2. Original data points

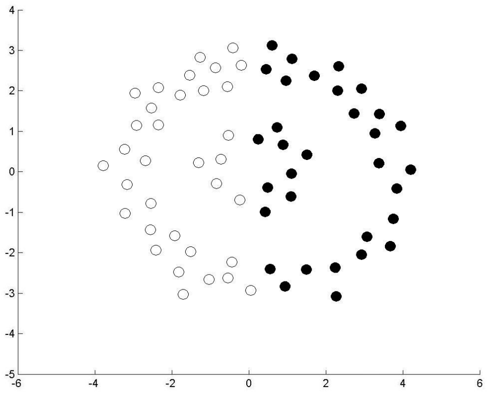


Figure 3.3. Sub-optimal clustering produced by *k*-means (*k*=2)

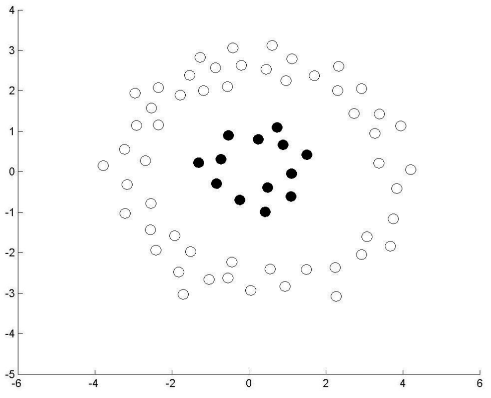


Figure 3.4. Optimal clustering (concentric circles)

3.1.4 Cluster Analysis

If the clusters detected by clustering are to be used in a user model to guide the adaptations of a learning environment (see Section 3.2), the clusters must be analyzed to determine which ones represent students showing effective vs. ineffective interaction behaviors so that appropriate adaptations can be made. This is best done by using objective information about learning gains from application use (e.g., improvements from pre to post-tests) to determine which clusters of students were successful learners and which were not (see dotted arrow marked 'Test Results' between 'Data Collection' and 'Cluster Analysis' in Figure 3.1). If learning gains are unknown, then intuition or expert evaluation is required to analyze and label the clusters in terms of learning outcomes (illustrated in Figure 3.1 by the dotted arrow from 'Developer' to 'Cluster Analysis'). In this case, developer or expert workload may still be reduced because they avoid the time-consuming process of having to observe individual student interactions and then look for meaningful patterns. Instead, they are automatically presented with a picture of common behavioral patterns (the clusters) from which they can make inferences about potential learning effects (by comparing cluster means for example). In this research, we take the first approach because we had pre and post-test results for both experiments. Furthermore, this approach allows us to validate whether or not clustering was in fact able to recognize meaningful interaction behaviors.

An additional step in cluster analysis is to explicitly characterize the student interaction behaviors represented by the different clusters by evaluating cluster similarities and dissimilarities along each of the feature dimensions. While this step is not strictly necessary for online recognition based on supervised learner classification (see Section 3.2), it is useful to help developers gain insights about the different learning behaviors and devise appropriate adaptive interventions targeting them.

In this research, we use formal tests to compare clusters in terms of learning and feature similarity. To compare two clusters (obtained when k is set to 2), we use Welch's t-test (Student's t-test corrected for unequal sample variances) which measures the difference between the means of two distributions (e.g., the distributions of learning gains represented by the two clusters). We consider a p value less than .05

to be a statistically significant difference, and a *p* value between .05 and .1 to be a marginally statistically significant difference. A *p* value below .05 means that there is a less then 5% possibility that the difference between the distributions was caused by chance, and a value between .05 and 1 means there is a 5-10% possibility. We also compute the practical significance of a difference [79] by measuring the effect size or the magnitude of the difference. Effect size is independent of sample size and therefore computing the practical significance of a difference is especially important with small samples, as is the case for both of our experiments, because with small samples the power to find a statistically significant difference, even when one exists, is low. For k=2 we use Cohen's *d* [25] to measure effect size. We consider a large effect (a *d* value greater than .8) to be practically significant, and a medium effect (a *d* value between .5 and .8) to be marginally significant as per Cohen's standard [25].

To compare three or more clusters (when k>2), we use one-way analysis of variances (ANOVAs) to measure the statistical difference between distributions. Again, we use .05 for statistical significance and .1 for marginal statistical significance. To determine practical significance when k>2, we use partial eta-squared (partial η^2) [25, 61] to measure effect size. We consider a partial η^2 value greater than .14 (a large effect) to be practically significant, and a value between .06 and .14 (a medium effect) to be marginally significant [25]. With ANOVAs, a significant value simply means a significant difference exists between some of the clusters; further pair-wise comparisons must be performed to determine where the significant differences lie. We use the Tukey HSD adjustments [38] for post-hoc pair-wise comparisons to find the significant differences in this case. Here again, we consider statistical significance to be a p_{HSD} value less than .05. We also compute Cohen's *d* during pair-wise comparisons.

3.2 Online Recognition

3.2.1 Supervised Classification

Understanding the effectiveness of a student's behavior for learning is mostly useful if a learning environment can adapt its interface or generate tailored interventions to improve this behavior *while* the student is interacting with the system. Thus, the second phase of our modeling framework (lower left of Figure 3.1) proposes training a supervised classifier user model with the distinct clusters identified in the offline phase to recognize, or classify, new learners as they interact with a learning environment. Classification information can be used to guide an adaptive component of the learning environment in providing online adaptations. Here we describe the online learner classification process.

The clusters of learners found by *k*-means in the offline phase can be used directly to train a classifier user model. That is, once *k*-means has found clusters of learners offline, an online *k*-means classifier user model can use those clusters to incrementally update the classification of a new student into one of the clusters as the student interacts with the learning environment. As an action occurs, the feature vector representing the student's behavior thus far is updated to reflect the new observation. For example, immediately after an action, the feature dimension representing the frequency of that action must be recomputed to take into account the current occurrence. In addition, any other feature dimensions affected by this action (e.g., the latency dimensions after this action) must also be recomputed. After the feature vector representing the student's behaviors is updated, the student's classification is computed by simply recalculating the distances between the updated vector and each cluster centroid and then assigning the feature vector to the cluster with the nearest centroid.

3.3 Model Evaluation

Ideally, any model of student learning should be evaluated by testing the model's ability to predict the learning outcomes for new students. However, time restrictions prevented us from running additional user studies to collect more data to test the classifier user models we developed in this research. Therefore, in both of our experiments (see Chapters 4 and 5) we performed leave-one-out cross validation (LOOCV) evaluations to make use of the available data and provide initial evidence of the predictive accuracies of the classifiers. Here we describe this evaluation strategy.

We performed an N fold LOOCV evaluation of the classifier user models generated in each experiment, where N is the total number of available data points (feature vectors). In each fold, one student's data was removed from the training set of feature vectors, and then the reduced set was re-clustered by *k*-means (Section 3.1.2). Next, the removed student's data (the test data) was fed into a classifier user model trained on the reduced feature vector set (Section 3.2), and online predictions were made for the incoming actions as described in Section 3.2.1. Model accuracy is evaluated as actions are observed, where accuracy is measured as the percentage of students correctly classified into the clusters to which they were assigned in the offline phase.

It should be noted, however, that by using a LOOCV strategy, we run the risk of altering the original clusters detected in the offline phase by using the entire feature vector set. Therefore, we should not expect to achieve 100% accuracy even after seeing all the actions because the user models are classifying incoming test data given the clusters found by LOOCV using the reduced set of feature vectors. In supervised machine learning, this issue is known as *hypothesis stability* [49]. In [51] the authors extend this notion to the unsupervised setting by defining a stability cost (SC), or expected empirical risk, which essentially quantifies the inconsistency between the original clusters and those produced by LOOCV. Thus, a low SC helps to ensure that the original clusters are relatively resistant to distortions caused by the removal of one feature vector. SC is computed by

$$SC = \mathbb{E}\,[\min_{\pi} (1/N)\, \Sigma_i\ I\{\ \pi(C_{X'}(x_i)) \neq C_X(x_i)\}] \tag{3.6}$$

where $\mathbb{E}\,[\cdot]$ is the expectation operator,

$I\{\cdot\}$ is the indicator function which is 1 when the bracketed expression is true and 0 otherwise,

$\pi\,(\cdot)$ denotes one of the $k!$ permutations of the cluster labels, where k is the number of clusters,

$i = 1\ldots N$, where N is the number of feature vectors,

$C_X(x_i)$ is the offline cluster label for feature vector x_i, and

$C_{X'}(x_i)$ is the LOOCV model prediction for feature vector x_i.

We can estimate SC by computing the quantity inside the square brackets in equation 3.6 for each fold of the cross validation and then taking the average. Perfect stability (SC=0) occurs when the training labels determined in the offline phase are unchanged by LOOCV and the label produced for the remaining test case is the same as the original offline label for that feature vector. Conversely, maximum instability (SC=1) occurs when none of the data labels are maintained by LOOCV. We compute the stability cost prior to assessing predictive accuracy to ensure that the models are essentially predicting what we would like it to predict, i.e., the membership of the removed student's behavioral patterns in one of the clusters defined in the offline phase.

A second shortcoming of our evaluation method is that we computed predictive accuracy by measuring the correspondence between the online classifications of a student (i.e., using only the actions observed up to a given point during that student's interaction) and the *final* classification of that student (as either an effective or ineffective learner) determined by offline clustering (i.e., using all of that student's observed actions). A more accurate evaluation would be to measure the correspondence between the predicted classification up to a given point and the student's actual classification up to that point. One way to do this would be to manually label every student action as effective or ineffective for learning in order to compare with the predicted classifications. However, such manual labeling is precisely what we are trying to avoid with our modeling framework. Nevertheless, while this evaluation is imperfect, it still provides a lower bound of our model's predictive power.

Chapter 4

Exploratory Learning Environment 1: CIspace CSP Applet

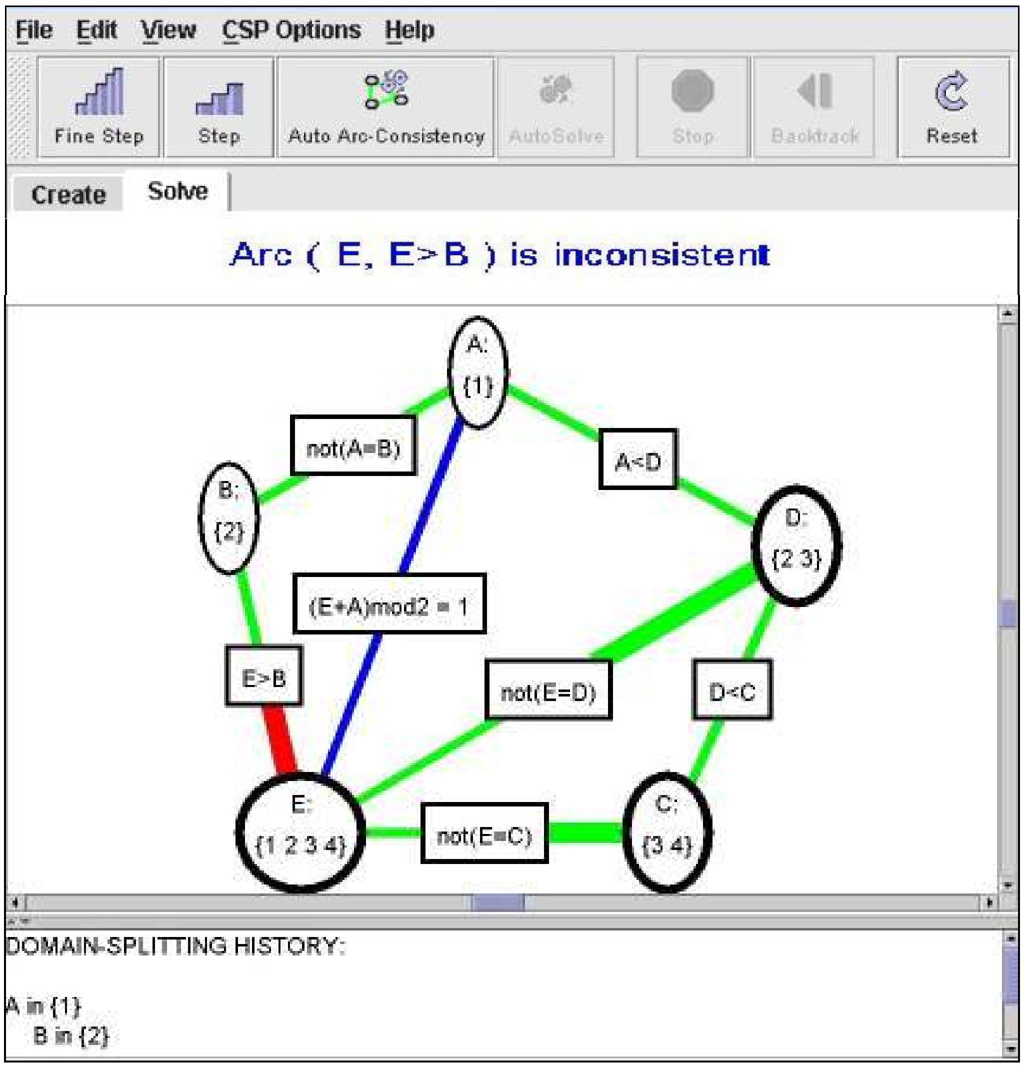
In our first attempt at applying our user modeling framework, we experiment on an exploratory learning environment (ELE) called the CIspace Constraint Satisfaction Problem (CSP) Applet. The CSP Applet is one of a collection of interactive algorithm visualization (AV) tools for learning common Artificial Intelligence (AI) algorithms (including algorithms for search, machine learning, and reasoning under uncertainty) called CIspace [3]. Through the use of graphs and animations, AVs aim to better demonstrate algorithm dynamics than traditionally static media alone. AV systems that fall under the category of ELEs also enable interactive exploratory learning and are motivated by similar theories about the benefits of active engagement as ELEs [60]. However, despite theories and intuitions behind AVs, reports on their pedagogical effectiveness have been mixed [45]. As for ELEs in general, there is evidence that the impact an AV will have on a user is dependent upon the way in which the tool is used, as well as distinguishing characteristics of the learner such as varying learning abilities and learning styles [45, 75]. Such reports emphasize the need for ELEs that use interactive AVs to provide adaptive support for individual students.

In this chapter, we demonstrate how we apply our framework (Chapter 3) to build a user model for the CIspace CSP Applet. We first describe the CSP Applet interface (Section 4.1), and then present and discuss the results of applying our framework to build a model for it (Section 4.2).

A version of this chapter has been published:
Amershi, S. and Conati, C. (2006) Automatic Recognition of Learner Groups in Exploratory Learning Environments. In the Proceedings of the 8th International Conference on Intelligent Tutoring Systems, pp. 463-472.

4.1 The CIspace CSP Applet Learning Environment

The CIspace CSP Applet (Figure 4.1) illustrates algorithm dynamics on graphs by the use of color and highlighting, and graphical state changes are reinforced through textual messages above and below the graph.



```
Arc ( E, E>B ) is inconsistent
```

```
DOMAIN-SPLITTING HISTORY:

A in {1}
   B in {2}
```

Figure 4.1. CIspace CSP Applet with example CSP

A CSP consists of a set of variables, variable domains and a set of constraints on legal variable-value assignments. The goal is to find an assignment that satisfies all constraints. The CSP Applet demonstrates the Arc Consistency 3 (AC-3) algorithm for solving CSPs [66]. AC-3 iteratively makes individual arcs consistent by removing variable domain values inconsistent with a given constraint until all arcs have been considered and the network is consistent. Then, if there remains a variable with more than one domain value, a procedure called domain splitting can be applied to that variable to split the CSP into disjoint cases so that AC-3 can recursively solve each case or sub-network.

A CSP is graphically represented in the CSP Applet as a network of variable nodes and constraint arcs (Figure 4.1 shows an example CSP in the Applet). The CSP Applet provides several mechanisms for the interactive execution of the AC-3 algorithm, accessible through the button toolbar shown at the top of Figure 4.1, or through direct manipulation of graph elements. Note that the applet also provides functionalities for creating CSP networks, but we limit our analysis to only those relevant to solving a predefined CSP as we believe these are most influential in learning and therefore these were the only mechanisms evaluated in the user study.

The *Fine Step* mechanism allows the student to manually advance through the AC-3 algorithm at a fine scale in order for the student to analyze detailed graphical state changes. *Fine Stepping* cycles through three stages carried by consecutive clicks of the *Fine Step* button. Initially, all the arcs in the network are colored blue indicating that they need to be tested for consistency. In the first stage, the Applet automatically selects a candidate blue arc, which then appears highlighted in the network. In the second stage, the Applet tests the arc for consistency. If it is found to be consistent, the arc's color will change to green and the *Fine Step* cycle terminates. If it is inconsistent, its color changes to red and a third *Fine Step* is needed. In this final stage, the Applet reduces the domain of the connected variable to remove the inconsistency and turns the arc green. Arcs that could have become inconsistent as a result of this domain reduction need to be retested and are again turned blue. The effect of each *Fine Step* is reinforced explicitly in text through a panel above the graph (see message above the CSP in Figure 4.1).

The *Step* mechanism executes the algorithm in coarser detail. One *Step* performs all three stages of *Fine Step* on an arc at once. The *Direct Arc Click* mechanism allows the student to perform a *Step* on a given arc by clicking directly on it. Therefore, this mechanism gives students more control over the algorithm than

Fine Stepping and *Stepping* by allowing them to decide which arc to make consistent rather than having the applet select arcs for them.

The *Domain Splitting* mechanism allows a student to divide the network into smaller sub-problems by splitting a variable's domain. This is done by clicking directly on a node in the network and then selecting values to keep in the dialog box that appears (Figure 4.2). The choice of variables to split on and values to keep affects the algorithm's efficiency in finding a solution.

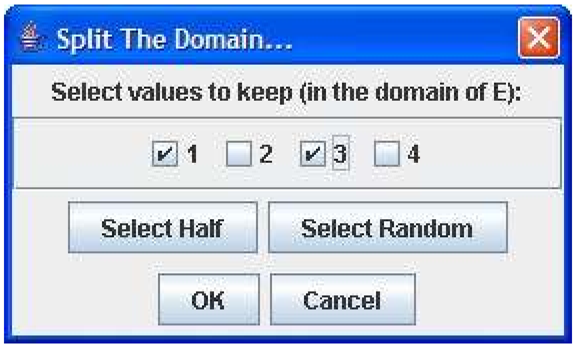


Figure 4.2. *Domain Splitting* dialog box

The *Backtrack* mechanism recovers the alternate sub-problem set aside by *Domain Splitting* allowing for recursive application of AC-3. After *Domain Splitting* or *Backtracking*, the network is updated to reflect the changes and a record of these actions will appear in the Domain-Splitting History panel at the bottom of the Applet window (see Figure 4.1).

The *Auto Arc Consistency (Auto AC)* mechanism automatically *Fine Steps* through the CSP network, at a user specified speed, until it is consistent. The *Auto Solve* mechanism iterates between making the CSP consistent (by *Fine Stepping*) and automatically splitting domains until a solution is found. If the CSP has more than one solution, then activating this mechanism again will first *Backtrack* to the sub-problem that was set aside during the last automatic *Domain Split,* and then iterate again between making the CSP consistent and domain splitting until another solution is found, and so on. The *Stop* mechanism lets the student stop execution of *Auto AC* or *Auto Solve* at any time.

The *Reset* mechanism restores the CSP to its initial state so that the student can re-examine the initial problem and restart the algorithm.

4.2 Applying Our User Modeling Framework to the CSP Applet

4.2.1 Data Collection for the CSP Applet

The data we use for this experiment was obtained from a previous user study investigating the effects of learning with the CSP Applet. Because one of the goals of the study was to gauge user preference for the CSP Applet compared to a traditional method of learning using paper-based sample problems (with static images and text), a within-subject design was used where students were exposed to both of these media during the learning portion of the user study. Therefore, we cannot attribute all of the learning gains to just the use of the CSP Applet, as would be ideal for the objectives of the current research (see Section 3.1.4). However, we still see some interesting and significant results from attempting our user modeling approach on this data, as will be discussed in the following sections.

It should also be noted that during our pilot tests, before the user study was conducted, we observed students misusing some of the CSP Applet features (the *Step* and *Auto Solve* mechanisms described in Section 4.1) and so these were subsequently removed from the study. It would have been useful to see if our proposed modeling framework was able recognize the misuse of these features as ineffective learning behaviors, yet, remarkably, it was still able to identify several other candidate behaviors that may lead to poor learning (see Section 4.2.4). The fact that by applying our framework we discovered other suboptimal learning behaviors that were not obvious to us when we observed students in the pilot tests, highlights how difficult it can be to recognize distinct learning behaviors in ELEs, even by application experts.

The user study typified a study scenario in which a student learns underlying concepts from text-based materials, studies relevant sample problems, and finally is tested for understanding of the instructional material. A total of 24 undergraduate computer science and engineering students participated in the user study. These students had sufficient background knowledge to learn about CSPs, but had no previous exposure to AI algorithms.

The study followed a within-subject design. First, all of the students were given one hour to read a textbook chapter on CSP problems [66]. Next, the students took a 20 minute pre-test on the material. The pre-test was marked out of 19 total marks. After the pre-test, each student studied two sample problems for 12 minutes using the CSP Applet for one problem and the paper-based medium for the other (presentation order was counterbalanced to avoid ordering effects). The student then had a choice to study with the CSP Applet or the paper-based medium for the final sample problem. After studying the sample problems, students were given another 20 minute, 19 marks post-test almost identical to the pre-test except for a few different domain values or arcs.

For the current experiment, we use the following data collected from the 24 students who participated in the user study: time-stamped logged data of user interactions with the CSP Applet, and results from the tests administered before and after the study session. From the logged data we obtained 1931 user actions over 205.3 minutes. These actions (detailed in Section 4.1) include:

- *Fine Step* – Executing detailed algorithm steps (selecting an arc, testing it for consistency, removing variable domain values).
- *Direct Arc Click* – Selecting an arc and making it consistent.
- *Auto Arc Consistency (Auto AC)* – Running the AC-3 automatically.
- *Stop* – Stopping *Auto AC*.
- *Reset* – Resetting the CSP to its initial state.
- *Domain Split* – Selecting a variable to split and specifying a sub-network for further application of AC-3.
- *Backtrack* – Recovering the alternate sub-network set aside by *DS*.

4.2.2 Preprocessing for the CSP Applet

From the logged user study data, we computed 24 feature vectors corresponding to the 24 study participants. The feature vectors had 21 dimensions, resulting from deriving three features for each of the seven actions described in the previous section:

(1) the average frequency of the action, (2) the average latency after the action, and (3) the standard deviation of the latency after the action. Recall from Chapter 3, that the second dimension is an indicator of student reflection, and the third dimension is an indicator of selectiveness since varied latency may indicate planned rather than impulsive or inattentive behavior and may not be as detrimental for learning.

Although, as a general rule of thumb, 5 to 10 times as many feature vectors as feature dimensions is recommended to prevent model overfitting [48], we decided not to perform the feature selection step (Section 3.1.2) for this experiment since the number of feature dimensions was still lower than the number of feature vectors. And, as will be discussed in the following sections, without the feature selection step, our framework was still able to find several meaningful behavioral patterns from the CSP Applet data.

4.2.3 Unsupervised Clustering for the CSP Applet

We applied *k*-means clustering to the study data with *k* set to 2, 3 and 4 because we only expected to find a few distinct clusters with our small sample size. For each trial, we executed *k*-means 20 times and used the clusters that produced the highest value for the discriminant criterion defined in equation 3.5 (see Section 3.1.3). The clusters found by *k* set to 4 were the same as those with *k* set to 3 with the exception of one data point forming a singleton cluster. This essentially corresponds to an outlier in the data, and so we report only the results for *k* set to 2 and *k* set to 3.

Figure 4.3 shows the two clusters found by *k*-means with *k* set to 2 (one cluster is marked with crosses while the other is marked with circles). For visualization purposes, Principal Components Analyses was done on the feature vectors to project the data from the 21D space to 2D. Therefore, low within-cluster variance and high between-cluster separation (as per the discriminant criterion) may not be entirely visible in the figure. One of the clusters found by *k*-means in this case consisted of four members, while the other had 20.

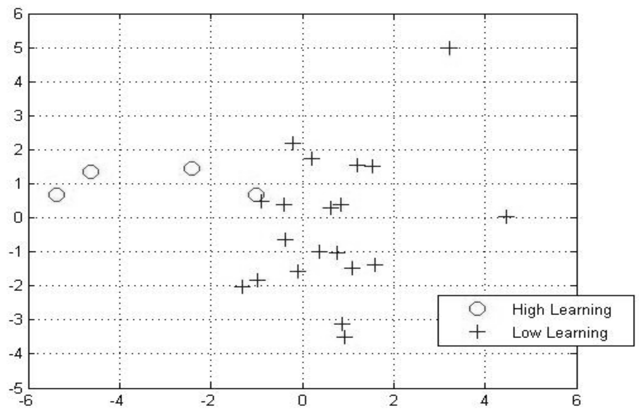


Figure 4.3. Clusters resulting from *k*-means clustering (*k*=2) on the CIspace CSP Applet data

The three clusters found by *k*-means with *k* set to 3 are also projected onto 2D, and shown in Figure 4.4. Again, one of the clusters (marked by circles) had four members. The second cluster (marked by crosses) had 12 members, and the third cluster (marked by asterisks) had eight members.

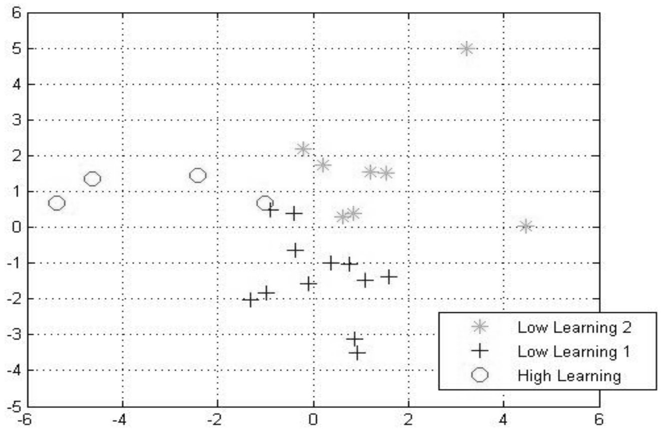


Figure 4.4. Clusters resulting from *k*-means clustering (*k*=3) on the CIspace CSP Applet data

4.2.4 Cluster Analysis for the CSP Applet

Cluster Analysis Results (*k*=2)

Recall from Chapter 3 (Section 3.1.4) that when *k*=2, we use Welch's t-test to measure statistical significance, and Cohen's *d* to measure practical significance. We determine a *p* value less than .05 to be statistically significant, a *p* value between .05 and .1 to be marginally statistically significant, and a *d* value greater than .8 (i.e. a large effect size) to be practically significant. A statistically and practically significant difference (*t*(4.23)=2.69, *p*=.026, *d*=1.21) was found in learning gains, from pre to post tests, between students in the two clusters found by *k*-means with *k* set to 2. The cluster with high average learning gains (HL from now on) consisted of four students (mean learning gain=7.0 marks, standard deviation=2.68 marks), and the cluster with low average learning gains (LL from now on), had 20 students (mean learning gain=3.08 marks, standard deviation=2.62 marks) (see Figure 4.3 in Section 4.2.3).

In order to characterize the HL and LL clusters in terms of student interaction behaviors, we computed the differences between the clusters along each of the 21 dimensions. Table 4.1 summarizes these differences. Feature dimensions where statistically or practically significant differences were found, and the corresponding significant values, are highlighted in bold.

Table 4.1. Pair-wise comparisons between HL and LL clusters along each of the 21 feature dimensions

Feature Description	**HL average**	**LL average**	**df**	**t**	***p***	***Cohen's d***
***Fine Step* frequency**	.025	.118	17.4	3.82	**6e-4***	**1.34***
***Fine Step* latency average**	10.2	3.08	3.25	3.91	**.013***	**1.90***
***Fine Step* latency SD**	12.2	4.06	4.01	4.48	**.005***	**2.04***
Direct Arc Click frequency	.050	.036	3.74	.057	.299	.267
Direct Arc Click latency average	4.18	5.71	5.47	.782	.233	.331
Direct Arc Click latency SD	3.63	5.74	11.14	.968	.177	.362
Auto AC frequency	.007	.003	3.30	1.45	.118	.700
Auto AC latency average	23.7	36.9	22.0	.955	.175	.316
Auto AC latency SD	20.4	12.8	4.07	.648	.276	.294
***Stop* frequency**	.003	7e-4	3.84	2.03	**.058**	**.935***
Stop latency average	1.75	1.60	16.4	.132	.448	.047
***Stop* latency SD**	1.06	0	3.0	2.33	**.051**	**1.16***
Reset frequency	.010	.008	3.89	.481	.329	.221
***Reset* latency average**	46.6	11.4	3.14	1.76	**.086**	**.866***
***Reset* latency SD**	24.4	9.56	7.75	3.81	**.003***	**1.51***
***Domain Split* frequency**	.003	.009	21.5	2.42	**.012***	**.783**
Domain Split latency average	6.75	4.61	5.64	1.11	.156	.465
Domain Split latency SD	1.37	3.04	14.92	1.66	.059	.596
Backtrack frequency	8e-4	.002	22.0	1.25	.113	.413
Backtrack latency average	1.75	3.68	19.8	.820	.211	.281
Backtrack latency SD	0	3.27	19.0	1.66	.057	.524

* Significant at $p<.05$ or $d>.8$ (feature description and values in bold)

Discussion of Cluster Analysis Results (k=2)

Here, we interpret the differences along the individual feature dimensions (listed in Table 4.1) where significant (statistically or practically), or marginally significant (statistically) differences were found, or where the results from a combination of dimensions were sensible.

The results on the use of the *Fine Step* feature are quite intuitive (see *Fine Step* entries in Table 4.1). From the first boxplot in Figure 4.5, we can see that the students in the LL cluster (LL students from now on) used this feature significantly more frequently than the HL students. In addition, both the latency averages and standard deviations after a *Fine Step* were significantly shorter for the LL cluster indicating that LL students *Fine Stepped* frequently and consistently too quickly (given by the combination of low *Fine Step* latency and standard deviation). This supports that LL students may be using this feature mechanically, without pausing long enough to consider the effects of each *Fine Step*, a behavior that may contribute to the low learning gains achieved by these students.

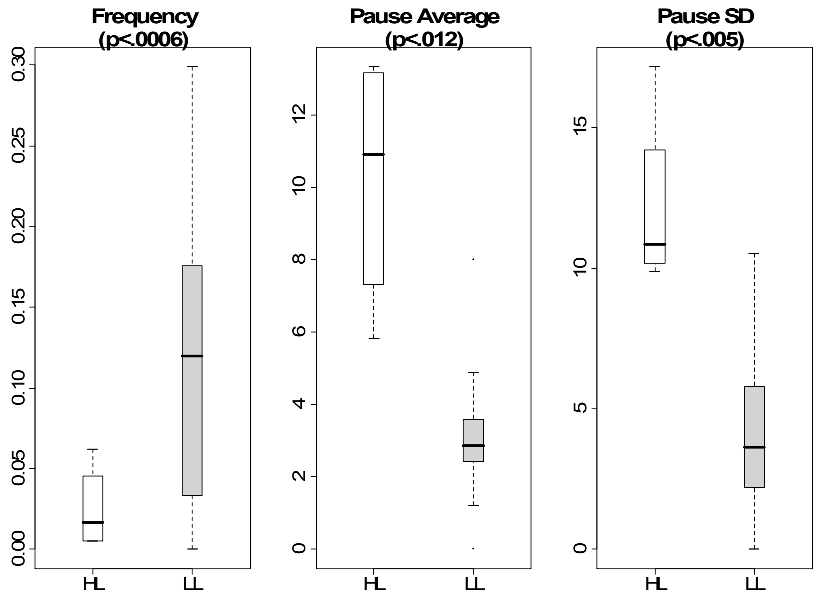


Figure 4.5. *Fine Step* boxplots between HL (white) and LL (gray) clusters. From left to right: frequency, latency average, and latency standard deviation

The HL cluster of students used the *Auto AC* feature more frequently than the LL cluster (see '*Auto AC* frequency' in Table 4.1), although the difference is not statistically significant. In isolation, this result appears unintuitive considering that simply watching the AC-3 algorithm in execution is an inactive form of learner engagement [60]. However, in combination with the significantly higher frequency of *Stopping* (see '*Stop* frequency' in Table 4.1), these behaviors suggests that the HL students could be using these features to forward through the AC-3 algorithm in larger steps to analyze it at a coarser scale rather than just passively watching the algorithm progress. Figure 4.6 shows boxplots of the *Auto AC* and *Stop* frequencies between the HL and LL clusters.

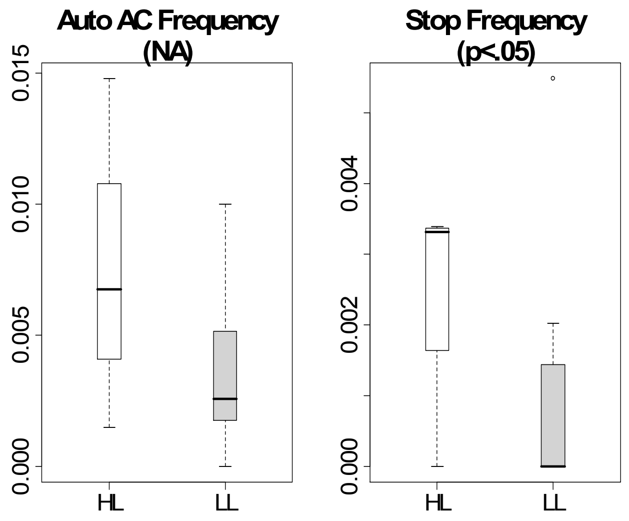


Figure 4.6. *Auto AC* frequency boxplot (left) and *Stop* frequency boxplot (right) between HL (white) and LL (gray) clusters

The HL students also paused longer and more selectively after *Resetting* than the LL students (see '*Reset* latency average' and '*Reset* latency SD' entries in Table 4.1). With the hindsight that these students were successful learners, we can interpret this behavior as an indication that they were reflecting on each problem more that the LL students. However, without the prescience of learning outcomes, it is likely that an application expert or educator observing the students would overlook this less obvious behavior.

There was also a significant difference in the frequency of *Domain Splitting* between the HL and LL clusters of students, with the LL cluster frequency being higher (see '*Domain Split* frequency' in Table 4.1). As it is, it is hard to find an intuitive explanation for this result in terms of its connection with HL versus LL student learning. However, analysis of the clusters in the following section shows finer distinctions along this dimension, as well as along the latency dimensions after a *Domain Split* action, between the three different clusters found by *k*-means with *k* set to 3. These latter findings are more revealing, indicating that there are likely more than two common learning patterns.

Cluster Analysis Results (*k*=3)

For comparing three clusters (see Section 3.1.4), we use one-way ANOVA to measure statistical significance (a *p* value less than .05), and partial η^2 to measure practical significance (a partial η^2 greater than .14, i.e., a large effect size). In addition, we use Tukey's HSD adjustments for post-hoc pair-wise comparisons (with p_{HSD} less than .05 being statistically significant), and Cohen's *d* for measuring pair-wise effect sizes (with a *d* value greater than .8 being practically significant).

Of the three clusters found by *k*-means with *k* set to 3 (see Figure 4.4 in Section 4.2.3), one cluster consisted of the same four students as was found by *k*-means with *k* set to 2 (mean learning gain=7.0 marks, standard deviation=2.68 marks), one consisted of eight students (mean learning gain=2.94 marks, standard deviation=2.56 marks), and one consisted of 12 students (mean learning gain=3.17 marks, standard deviation=2.77 marks). Here, we again found a statistically and practically significant difference (F(2,21)=3.58, *p*=.045, partial η^2=.254) in learning gains between the clusters. The cluster of four students showed significantly (statistically and practically) higher learning gains than both of the other two clusters (p_{HSD}=.029, *d*=1.27 for the cluster with eight students, and p_{HSD}=.029, *d*=1.16 for the cluster with 12 students). No significant differences in learning gains were found between the two clusters of students with lower learning gains, suggesting that students may use/misuse the CSP Applet ELE in a variety of distinctive ways. Hereafter, we will refer to the cluster with four students as 'HL' (high learners), the cluster with eight students as 'LL1' (low learners 1), and the cluster with 12 students as 'LL2' (low learners 2).

Table 4.2 summarizes the three-way comparisons amongst the three clusters along each of the 21 dimensions. Feature dimensions with statistically or practically significant differences, and the corresponding significant values, are highlighted in bold. Table 4.3 summarizes the post-hoc pair-wise comparisons between the clusters (i.e., HL compared to LL1, HL compared to LL2, and LL1 compared to LL2) along each of the dimensions. Here, feature dimensions along which any of the pair-wise comparisons showed significant differences is highlighted in bold. Again, the significant values are also highlighted in bold.

Table 4.2. Three-way comparisons between HL, LL1, and LL2 clusters along each of the 21 feature dimensions

Feature Description	**HL average**	**LL1 average**	**LL2 average**	**F**	***p***	***partial* η^2**
***Fine Step* frequency**	.025	.111	.122	1.98	.162	**.159***
***Fine Step* latency average**	10.2	3.07	3.08	20.4	**1e-5***	**.660***
***Fine Step* latency SD**	12.2	4.82	3.55	12.1	**3e-4***	**.536***
Direct Arc Click frequency	.050	.018	.046	1.55	.237	.128
Direct Arc Click latency average	4.18	6.66	5.07	.512	.606	.047
Direct Arc Click latency SD	3.63	5.06	6.18	.227	.799	.021
***Auto AC* frequency**	.007	.003	.004	2.66	.093	**.202***
Auto AC latency average	23.7	16.8	50.4	1.11	.347	.096
Auto AC latency SD	20.4	8.95	15.4	.481	.625	.044
***Stop* frequency**	.003	3e-4	9e-4	3.00	.071	**.222***
Stop latency average	1.75	.375	2.42	.676	.519	.060
***Stop* latency SD**	1.06	0	0	15.8	**6e-4***	**.600***
Reset frequency	.010	.008	.008	.160	.853	.015
***Reset* latency average**	46.6	18.7	6.52	6.94	**.005***	**.398***
***Reset* latency SD**	24.4	14.2	6.43	5.09	**.016***	**.327***
***Domain Split* frequency**	.003	.018	.003	12.0	**3e-4***	**.532***
***Domain Split* latency average**	6.75	8.68	1.89	12.0	**3e-4***	**.533***
***Domain Split* latency SD**	1.37	6.66	.622	27.7	**1e-6***	**.725***
Backtrack frequency	8e-4	.004	.002	.701	.508	.063
***Backtrack* latency average**	1.75	8.90	.202	3.21	.061	**.234***
***Backtrack* latency SD**	0	7.96	.138	2.92	.076	**.218***

* Significant at $p<.05$ or *partial* $\eta^2>.14$ (feature description and values in bold)

Table 4.3. Post-hoc pair-wise comparisons between HL, LL1, and LL2 clusters along each of the 21 feature dimensions

Feature Description	**HL vs. LL1**		**HL vs. LL2**		**LL1 vs. LL2**	
	p_{HSD}	d	p_{HSD}	d	p_{HSD}	D
***Fine Step* frequency**	.142	**1.10***	.078	**1.48***	.691	.106
***Fine Step* latency average**	**1e-5***	**1.98***	**1e-5***	**1.85***	.818	.007
***Fine Step* latency SD**	**.001***	**1.68***	**1e-4***	**2.33***	.395	.356
Direct Arc Click frequency	.216	.618	.751	.065	.147	.616
Direct Arc Click latency average	.389	.454	.667	.221	.449	.295
Direct Arc Click latency SD	.670	.254	.516	.420	.661	.126
***Auto AC* frequency**	**.046***	.745	.076	.666	.595	.228
Auto AC latency average	.72	.476	.403	.522	.198	.471
Auto AC latency SD	.386	.466	.631	.187	.491	.262
***Stop* frequency**	**.031***	**1.21***	.081	.783	.449	.296
***Stop* latency average**	.552	**.966***	.692	.169	.287	.387
***Stop* latency SD**	**5e-5***	**1.16***	**3e-5***	**1.16***	.823	0
Reset frequency	.562	.276	.620	.187	.744	.070
***Reset* latency average**	**.031***	.673	**.002***	**1.01***	.194	.867
***Reset* latency SD**	.136	**1.08***	**.007***	**1.84***	.125	.601
***Domain Split* frequency**	**.003***	**1.91***	.820	.011	**2e-4***	**1.69***
***Domain Split* latency average**	.350	.483	**.019***	**1.24***	**1e-4***	**1.79***
***Domain Split* latency SD**	**1e-4***	**2.83***	.488	.527	**0***	**2.73***
Backtrack frequency	.358	.611	.721	.220	.342	.365
***Backtrack* latency average**	.167	.745	.667	.648	**.028***	**.934***
***Backtrack* latency SD**	.118	**.867***	.811	.556	**.042***	**.851***

* Significant at p_{HSD}<.05 or d>.8 (feature description and values in bold)

Discussion of Cluster Analysis Results (k=3)

Here, we again interpret the results (see Tables 4.2 and 4.3) of individual or combinations of dimensions in order to characterize the interaction behaviors of students in each of the three clusters found by k-means with k set to 3.

Similar distinguishing *Fine Step* behaviors, as were identified by k-means with k set to 2, were also found between the HL cluster of students and both LL clusters. First, there was a trend of both the LL1 and LL2 clusters having a higher frequency of

Fine Stepping than the HL cluster (see HL, LL1 and LL2 averages of the '*Fine Step* frequency' dimension in Table 4.2). This, however, was only practically significant and not statistically significant (see '*Fine Step* frequency' in Tables 4.2 and 4.3). Second, the average latency after a *Fine Step* was significantly longer for the HL students than for both the LL1 and LL2 students (see *'Fine Step* latency average' entries in Table 4.2, and in Table 4.3 under 'HL vs. LL1' and 'HL vs. LL2'). And third, the standard deviation of the latency after a *Fine Step* was significantly higher for the HL students than for both the LL1 and LL2 students (see *'Fine Step* latency SD' entries in Table 4.2, and in Table 4.3 under 'HL vs. LL1' and 'HL vs. LL2'). As for the LL cluster found when *k* was set to 2, the latter two results show that LL students consistently tend to pause for a shorter duration after a *Fine Step* than HL students. Given that both the LL clusters showed low learning gains, these results would again suggest that LL students are typically less attentive to the results of a *Fine Step* action and this behavior may negatively affected their learning. Figure 4.7 illustrates the cluster differences along the *Fine Step* dimensions.

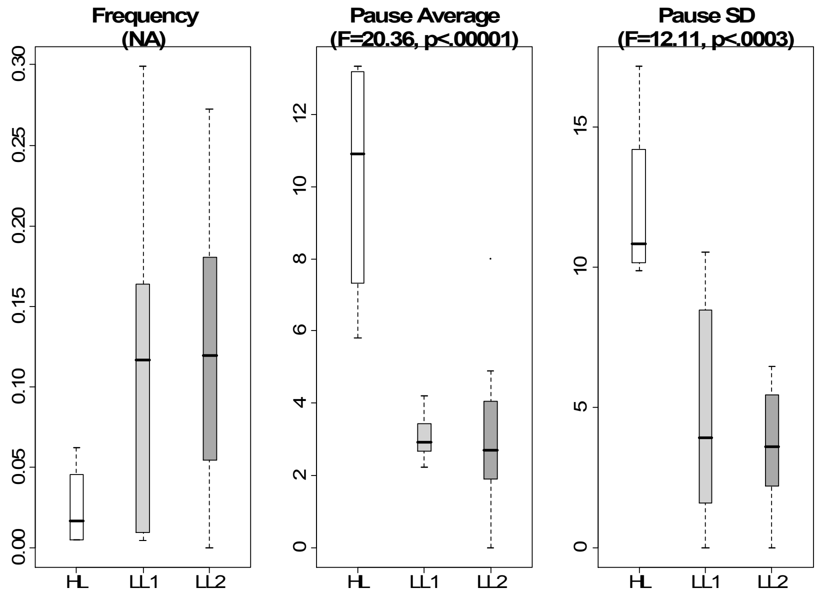


Figure 4.7. *Fine Step* boxplots between HL (white), LL1 (light gray), and LL2 (gray) clusters. From left to right: frequency, latency average, and latency standard deviation

The *Auto AC* frequency, *Stop* frequency, and *Reset* latency average and SD differences, between the HL and LL clusters found by *k*-means with *k* set to 2, were also replicated between the HL and LL clusters (both LL1 and LL2) with *k* set to 3. That is, the HL students used the *Auto AC* feature more frequently than the LL students. Unlike with the two clusters found with *k* set to 2 however, this difference was significant (see '*Auto AC* frequency' entry in Table 4.2), with the significant difference being between the HL and LL1 clusters (see '*Auto AC* frequency' entry in Table 4.3 under 'HL vs. LL1'). And again, the HL students *Stopped* the *Auto AC* feature more frequently than both the LL1 and LL2 students (see the '*Stop* frequency' averages under 'HL average', 'LL1 average' and 'LL2 average' in Table 4.2). This difference was significant (see '*Stop* frequency' entry in Table 4.2) with the significant difference again being between the HL and LL1 clusters (see '*Stop* frequency' entry in Table 4.3 under 'HL vs. LL1'). Like with the analysis of the clusters found with *k* set to 2, the combination of frequently starting and *Stopping* the *Auto AC* feature may indicate that the HL students were using these features to selectively forward through the AC-3 algorithm to learn. Finally, the HL students paused longer and more selectively after *Resetting* than both the LL1 and LL2 students (see '*Reset* latency average' and '*Reset* latency SD' entries in Tables 4.2 and 4.3), suggesting the HL students may be reflecting more on each problem.

This clustering also reveals several additional patterns, not only between the HL and LL clusters, but also between the two LL clusters, indicating that *k*=3 was better at discriminating between different learning patterns. For example, the clusters found with *k* set to 2 showed that the LL students used the *Domain Split* feature more frequently than the HL students, however, the clustering results from *k* set to 3 reveals a more complex pattern. This pattern is most easily visualized in the boxplots in Figure 4.8. An ANOVA showed that a significant difference did exist along the '*Domain Split* frequency' dimension (see corresponding entry in Table 4.2), but the pair-wise comparisons revealed that only the LL1 cluster made significantly more frequent use of the *Domain Split* feature (see '*Domain Split* frequency' entries in Table 4.3 under 'HL vs. LL1' and 'LL1 vs. LL2'). Furthermore, although the HL and LL2 clusters used the *Domain Split* feature comparably frequently, the HL students paused for significantly longer than the LL2 students after each *Domain Split* (see '*Domain Split* latency average' entries in Table 4.2, and in Table 4.3 under 'HL vs. LL2'). This feature is intended to require thought about efficiency in solving a CSP given different possible *Domain Splits*, and so longer pauses may be needed to thoroughly consider the choices. This would support the fact that the HL students showed higher learning gains than the LL2 students. However, it is interesting that the LL1 students paused for just as long after *Domain Splitting* as the HL students (i.e., significantly longer than the LL2 students as reported for the '*Domain Split* latency average' dimension in Table 4.3 under 'LL1 vs. LL2'), and more selectively than both the HL and LL2 students (see '*Domain Split* latency SD' entries in Table 4.2, and in Table 4.3 under 'HL vs. LL1' and 'LL1 vs. LL2'), yet still had low learning gains. The LL1 cluster is also characterized by longer pauses after *Backtracking* than both the HL and LL2 clusters (see HL, LL1 and LL2 averages for the '*Backtrack* latency average' dimension in Table 4.2). Long pauses after each of these actions (i.e., *Domain Splitting* and *Backtracking*) may indicate that the LL1 students were confused about these applet features or the concepts of domain splitting and backtracking. Once again, these complex behaviors may be difficult to identify through mere observation.

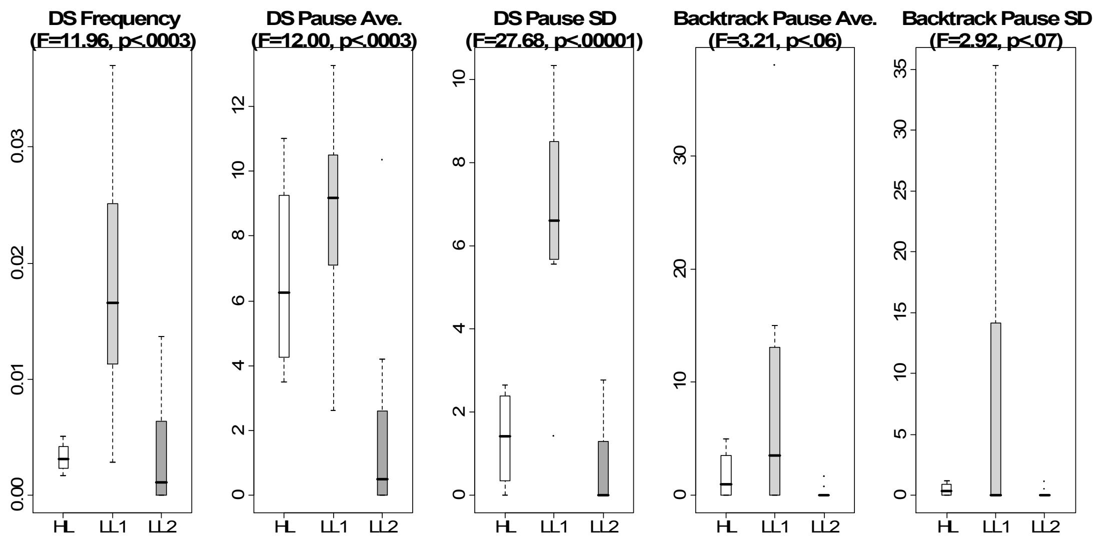


Figure 4.8. *Domain Split* and *Backtrack* boxplots between HL (white), LL1 (light gray), and LL2 (gray) clusters. From left to right: *Domain Split* frequency, *Domain Split* latency average, *Domain Split* latency standard deviation, *Backtrack* latency average, and *Backtrack* latency standard deviation

4.2.5 Supervised Classification for the CSP Applet

In this section we evaluate both a two and three-class *k*-means classifier user model (see Section 3.2.1) trained with clusters detected in the offline phase of our user modeling framework (described in the previous sections). These classifier user models can then take online data on a new student's interaction with the CSP Applet (represented as a sequence of feature vectors), and recognize or classify that student into one of the clusters detected in the offline phase. To evaluate both of the models, we followed the model evaluation process described in Section 3.3. That is, we performed a 24-fold leave one out cross validation (LOOCV) of the classifier models constructed for the CSP Applet via our user modeling framework. For both models, we first tested the stability of the original clusters (detected in the offline phase) against distortions caused by the removal of one data point using the LOOCV strategy. Recall that this is done by computing the stability cost (see Equation 3.6 in Section 3.3) which measures the inconsistency between the original clusters and those produced using the LOOCV strategy. We then evaluated the predictive accuracies of the models, or the percentage of students correctly classified into the clusters in which they were originally assigned to in the offline phase, in order to provide evidence of each model's ability to generalize to unseen data.

K-means Classifier Model Evaluation (*k*=2)

The estimated stability cost of using the LOOCV strategy to evaluate the classifier user model trained with the two clusters found with *k* set to 2 is 0.05 (averaging over the 24-folds). As discussed in Section 3.3, a low cost indicates that the two clusters (HL cluster=4 students, LL cluster=20 students) found by *k*-means clustering are relatively stable during the LOOCV evaluation. Therefore, a classification by the model means that the new student's learning behaviors are similar to those of either the HL or LL clusters identified in the offline phase and described in Section 4.2.4.

Figure 4.9 shows the performance of the two-class *k*-means classifier user model in predicting the correct classifications of new students (using the LOOCV strategy) as they interact with the CSP Applet. The percentage of correct classifications is shown as a function of the percentage of student actions the model has seen (solid line labeled 'Overall' in the figure's legend). The figure also shows the model's performance in classifying HL students into the HL cluster (dashed line) and LL students into the LL cluster (dotted line). For comparison purposes, the figure also shows the performance of a baseline model using a most-likely class classification method where new student actions are always classified into the most-likely, or largest, class. Therefore, this baseline model always classifies new student actions into the LL cluster (20 students). This is shown in Figure 4.9 by the dashed line straight across at the 83.3% (20 out of 24) classification accuracy level.

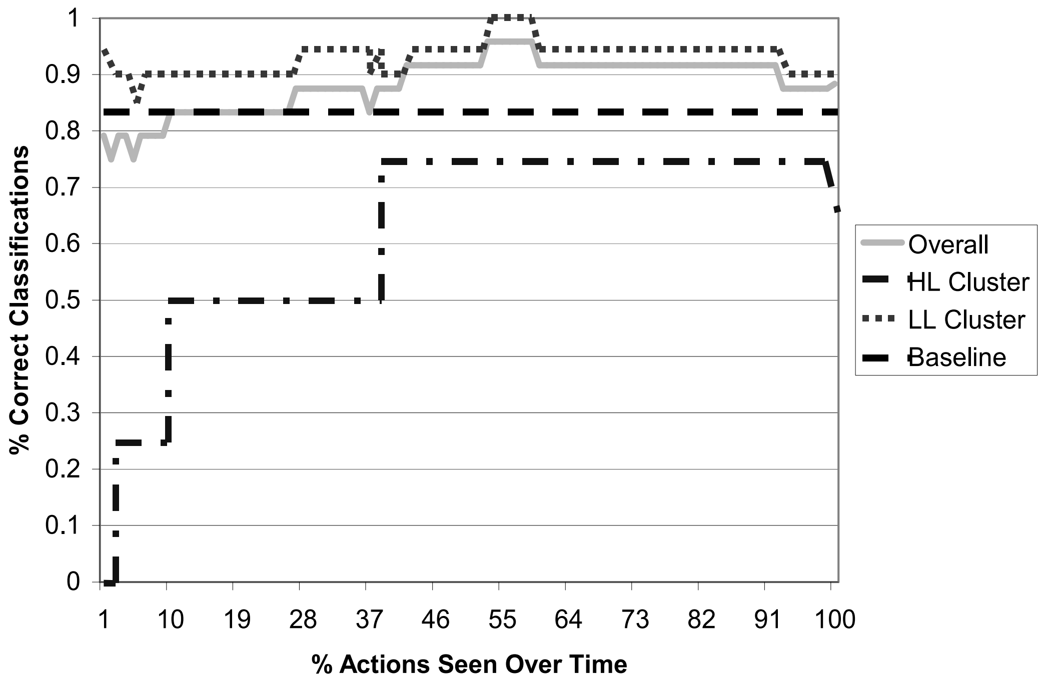


Figure 4.9. Performance of the CSP Applet user models (*k*=2) over time

These trends show that the overall accuracy of this classifier user model improves as more evidence is accumulated, converging to 87.5% after seeing all of the student's actions. Initially, the classifier user model performs slightly worse than the baseline model, but then outperforms the baseline model after seeing about 30% of the student's actions. The accuracy of the classifier model in recognizing LL students remains relatively consistent over time, converging to approximately 90%. In contrast, the accuracy of the model in recognizing HL students begins very low, reaches relatively acceptable performance after seeing approximately 40% of the student's actions, and eventually converges to approximately 75% after seeing all of the student's actions. It should be noted that the baseline approach would consistently misclassify HL students and thus interfere with the unconstrained nature of ELE interaction for these students.

The LL and HL cluster accuracies effectively measure the sensitivity and specificity of the classifier user model, respectively. That is, the LL cluster accuracy measures how well the model detects behaviors that may be suboptimal for learning when the learning outcomes for the student are poor, and the HL cluster accuracy measures the performance of the model at recognizing effective learning behaviors when the student's learning gains are indeed high. Table 4.4 shows the accuracy, sensitivity, and specificity of the classifier user model averaged over time, as well as the accuracy of the baseline model averaged over time.

Table 4.4. Classification accuracies of the CSP Applet user models (*k*=2) averaged over time

	K-means Classifier User Model	Baseline Model
Overall Accuracy	88.3%	83.3%
Sensitivity (True Positive Rate)	93.5%	100%
Specificity (True Negative Rate)	62.6%	0%

The accuracy results show that the classifier user model built via our modeling framework would outperform a baseline model that uses the most-likely class classification method (88.3% accuracy averaged over time for the classifier user model compared to 83.3% accuracy for the baseline model). Furthermore, the high sensitivity result of the classifier user model (93.5%) shows that this model would be almost as good as the baseline model for quickly recognizing when a student behaves in ways ineffective for learning, essential for providing adaptive support for students who do not learn well with a given learning environment. However, the relatively low specificity (62%) of the classifier model, although better than that of the baseline model, may result in the system interfering with an HL student's natural learning behavior, thus hindering student control, one of the key aspects of ELEs. The imbalance between sensitivity and specificity is likely due to the distribution of the sample data [82] as the HL cluster has fewer data points than the LL cluster (4 compared to 20). This is a common phenomenon observed in classifier learning. Collecting more training data to correct for this imbalance, even if the cluster sizes are representative of the natural population distributions, may help to increase the specificity rate of the classifier user model [82].

K-means Classifier Model Evaluation (*k*=3)

The stability cost for using the LOOCV strategy to evaluate the three-way classifier user model is estimated at 0.09. The stability cost for this classifier is slightly higher than for the two-class classifier, although it is still quite low. This is likely due to the decreased cluster sizes with the higher *k* value (HL cluster=4 students, LL1 cluster=8 students, LL2 cluster=12 students). With smaller clusters, removing a data point is more likely to produce different clusterings during LOOCV than with larger clusters, and therefore smaller clusters are less stable.

Figure 4.10 shows the overall prediction accuracy as a function of the number of observed student actions for this classifier user model (solid line). For comparison purposes, the figure also shows the performance of a most-likely class baseline user model (dashed line) which always classifies student actions into the largest class (LL2, with 12 students). Again, the classifier user model's accuracy improves with more observations, starting off at about 50% accuracy, but then reaching approximately 83.3% after seeing all of the actions. After seeing approximately 30% of the student actions, the classifier user model outperforms the baseline model which has a consistent, 50% (12 out of 24) accuracy rate.

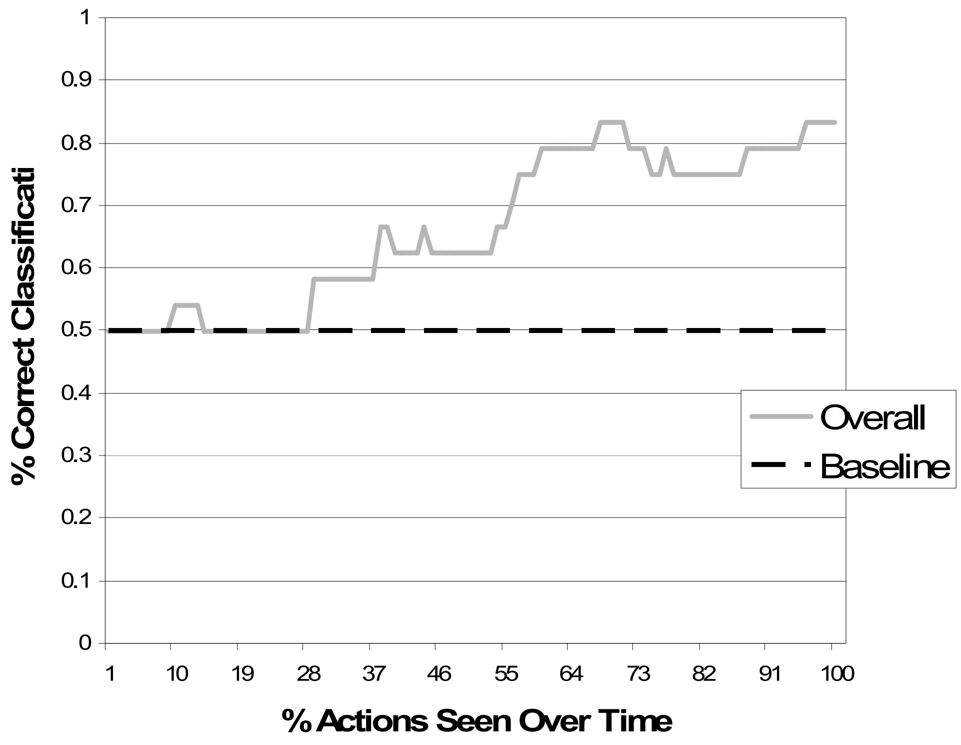


Figure 4.10. Performance of the CSP Applet user models (*k*=3) over time

Figure 4.11 shows the prediction accuracy trends for the individual clusters. For the HL cluster, the classification accuracy (dashed line) again begins very low, but reaches 75% after seeing about 40% of the actions, and then eventually reaches 100% after seeing all of the actions. The accuracy of the model at classifying LL1 students (dotted line), also begins low, but then reaches approximately 75% after seeing about 60% of the actions, and converges to approximately 85%. The accuracy for the LL2 students (solid line), remains relatively consistent as actions are observed, eventually reaching approximately 75%.

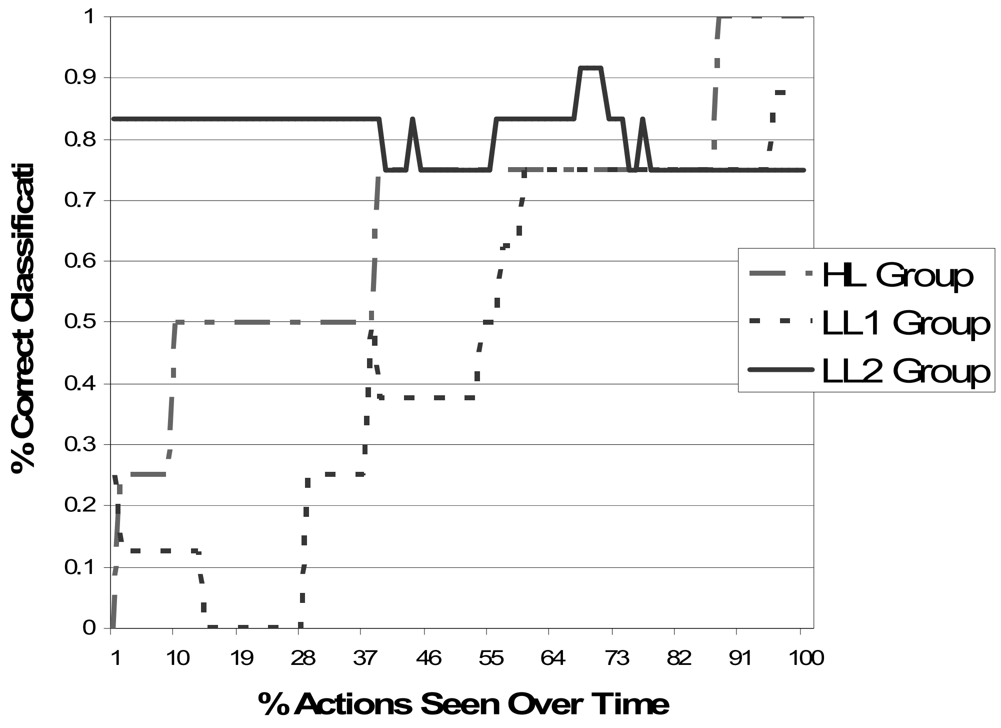


Figure 4.11. Performance of the CSP Applet user models (*k*=3) over time for the individual clusters

Table 4.5 reports the overall accuracy, sensitivity (averaged over both the LL clusters) and specificity of the three-way classifier user model averaged over time, as well as the overall accuracy of the baseline model. Again, the overall accuracy of the classifier user model is higher than the baseline model (66.2% compared to 50%). As with the increase in the stability cost, the lower accuracy, sensitivity and specificity of this classifier user model is likely an artifact of the fewer data points within each cluster. Further supporting this hypothesis is the fact that the LL2 cluster, which had 12 members, had the highest classification accuracy (80.3% averaged over time) (as seen in Figure 4.11), whereas the HL and LL1 clusters, which had only 4 and 8 members respectively, had visibly lower classification accuracies (66.3% and 44.9% averaged over time, respectively) (see Figure 4.11). Therefore, as the number of clusters increases, more training data should be collected and used when applying our user modeling framework.

Table 4.5. Classification accuracies of the CSP Applet user models (k=3) averaged over time

	K-means Classifier User Model	Baseline Model
Overall Accuracy	66.2%	50.0%
Sensitivity (True Positive Rate)	66.1%	100.0%
Specificity (True Negative Rate)	66.3%	0.0%

Chapter 5

Exploratory Learning Environment 2: ACE

The second application we use to test our proposed user modeling framework on is the Adaptive Coach for Exploration (ACE) [19], an intelligent ELE for the domain of mathematical functions. ACE's interface provides tools to support student-led exploration of the target domain while an adaptive Coach guided by a knowledge-based user model (see Section 5.1) provides tailored suggestions on how to improve exploration.

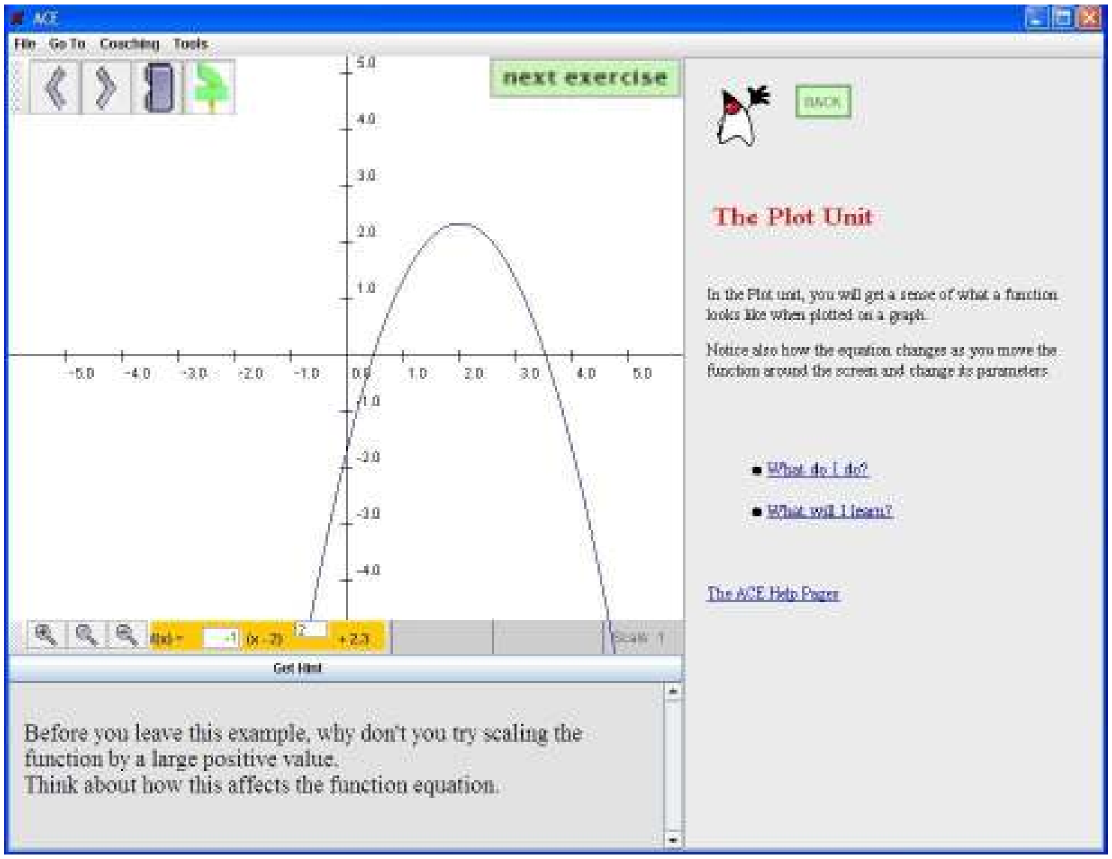
The ACE environment is comprised of three units, each designed to present concepts pertaining to mathematical functions in a distinct manner. The *Machine Unit* lets students feed various input values into a function and then observe the output values produced, with the aim of demonstrating the relationship between function inputs and outputs. Within the *Arrow Unit,* the student makes the connection between inputs and outputs herself by drawing lines between values in the given sets. ACE's *Plot Unit* offers the widest range of exploratory activities out of all the units by enabling students to experiment with textual as well as graphical function representations. This makes the Plot Unit an ideal candidate to evaluate our modeling framework and, therefore, we focus on the Plot Unit for the rest of this research.

In this chapter, we first describe ACE's Plot Unit and the interface actions that make up possible student interaction behaviors (Section 5.1). Then we present the results of applying our user modeling framework outlined in Chapter 3 to ACE (Section 5.2).

A version of this chapter has been accepted for publication:
Amershi, S. and Conati, C. (2007) Unsupervised and Supervised Machine Learning in User Modeling for Intelligent Learning Environments. To Appear in the Proceedings of the 10th International Conference on Intelligent User Interfaces.

5.1 The ACE Plot Unit

ACE’s Plot Unit interface (see Figure 5.1) is divided into three components. Function exploration occurs within the top-left panel of the interface. Directly below this is a panel through which ACE’s Coach displays tailored, on-demand hints to guide the student’s exploration. The component at the right contains hypertext *help* pages about ACE’s interface and the functions that can be explored.



f(x) = -1 (x - 2) 2 + 2.3

The Plot Unit

In the Plot unit, you will get a sense of what a function looks like when plotted on a graph.

Notice also how the equation changes as you move the function around the screen and change its parameters

- What do I do?
- What will I learn?

The ACE Help Pages

Before you leave this example, why don't you try scaling the function by a large positive value.
Think about how this affects the function equation.

Figure 5.1. ACE interface

The main exploration component of the Plot Unit (see Figure 5.2), visually demonstrates the relationship between mathematical equations and their corresponding plots in a Cartesian coordinate plane. Given a new function, students can experiment with different function parameters by editing the equation at the bottom of the panel (an *equation change* action) and analyzing the effects of the changes on the plot, or they can directly manipulate the plot (a *plot move* action) and examine how the equation parameters adjust to reflect the transformation. The middle magnifying glass button on the toolbar next to the function equation (see the magnifying glass icons at the lower left of Figure 5.2) lets the student *reset* the current function to its initial parameter values at any time.

Zooming capabilities are also accessible through the toolbar next to the function equation (see the zoom-in, '+', and zoom-out, '-', magnifying glass icons at the lower left of Figure 5.2). These allow students to inspect the plot at different scales.

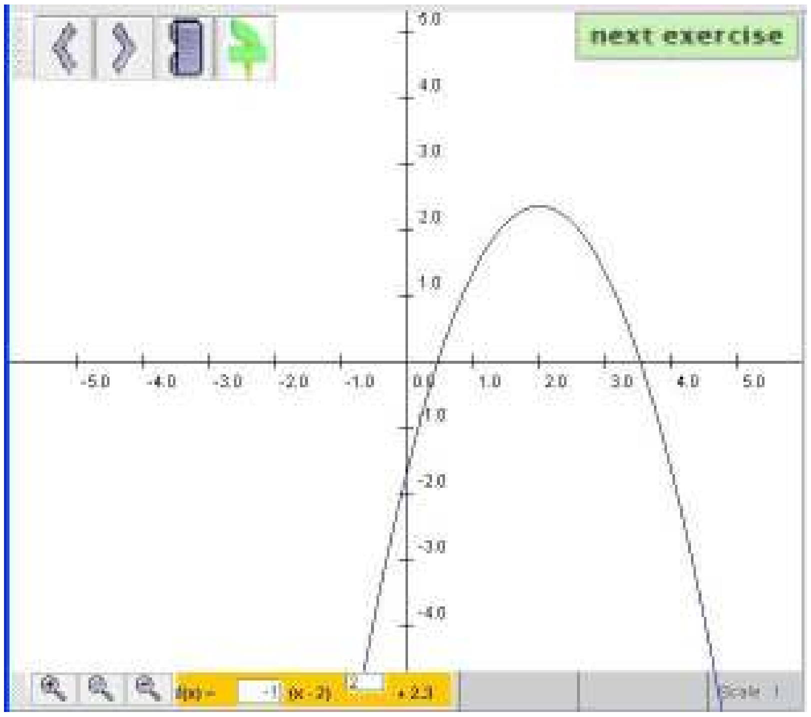


Figure 5.2. Main exploration component of ACE's Plot Unit

ACE's Plot Unit provides three types of functions, or exercises, for the student to explore: constant, linear and power functions. Each function type has an associated set of 'exploration cases' that together illustrate the full range of function attributes. For example, constant functions are defined by a single parameter that specifies the y-intercept attribute of the function. Therefore, in order to gain a broad understanding of constant functions, the student should study the two relevant exploration cases: positive intercepts and negative intercepts. In another example, linear functions are defined by two parameters specifying the function slope and the y-intercept. Here, the student should study the following relevant exploration cases: positive and negative intercepts, and positive, negative and zero slopes.

A standard curriculum is pre-defined for the Plot Unit and contains several exercises for students to explore. Students can advance sequentially through this curriculum by using the *next exercise* button at the top right of the coordinate plane (see Figure 5.2). The *forward arrow* on the button toolbar (top left of the coordinate plane in Figure 5.2) is equivalent to the *next exercise* button. Correspondingly, the *backward arrow* button steps backwards through the curriculum. Students are also free to explore the exercises in any order by using the *Lesson Browser* tool (accessible by clicking on the scroll icon on the toolbar). The Lesson Browser (see Figure 5.3) outlines the curriculum and lets the student decide which exercise to examine next.

The ACE interface also offers an *Exploration Assistant* tool (accessible by clicking on the street-sign icon on the toolbar) that can help students monitor and strategically plan their exploration within each exercise. The tool displays the relevant exploration cases for the current function type, and marks the cases that the student has already explored. For example, Figure 5.4 shows that the student has explored the positive intercept and negative intercept cases of a linear function, but has not yet experimented with the zero, positive or negative slope cases. This kind of monitoring [42] is an important meta-cognitive activity believed to benefit learning by helping students be proactive in their exploration.

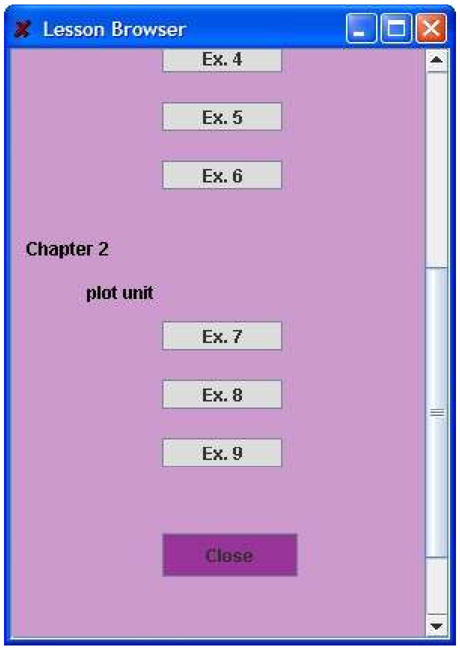
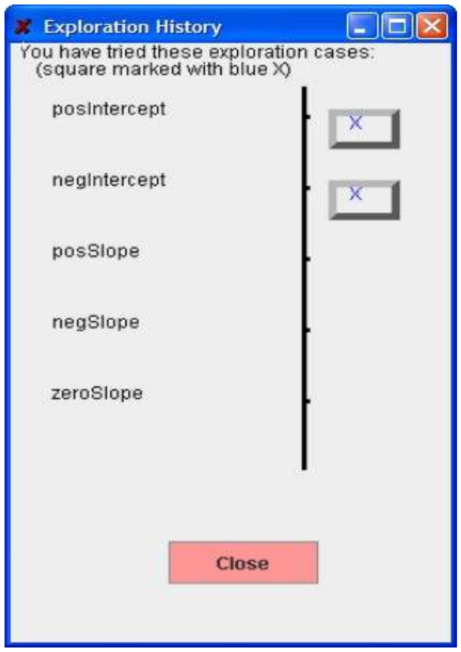


Figure 5.3. *Lesson Browser*

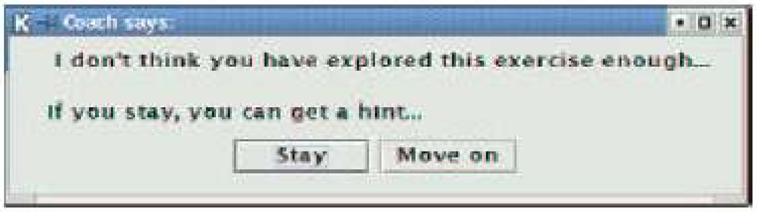


You have tried these exploration cases:
(square marked with blue X)

Figure 5.4. *Exploration Assistant*

ACE includes a knowledge-based user model of student exploration behavior (see Section 2.4) that guides the Coach's hints and interventions to improve those behaviors that are deemed to be suboptimal [17]. The model is a hand-constructed Dynamic Bayesian Network (DBN) that includes nodes to represent all possible exploration cases, nodes to represent student understanding of related mathematical concepts, and links representing how exploration of relevant cases relate to concept understanding. To assess whether a case has been explored effectively, the network includes information on student actions (only the *plot move* and *equation change* actions described above) and the latency between these actions. The latter is used as an estimate of a student's active reasoning on each exploration case. The network parameters (i.e., multi-valued prior and conditional probability tables for each node) were manually defined using prior knowledge or estimations.

On demand hints can be obtained from the Coach using the *Get Hint* button below the plot panel, which displays increasingly detailed hints on what cases to explore next based on the model's current assessment of the student's progress (see text at the bottom of Figure 5.1). In addition, the Coach intervenes through a dialog window (see Figure 5.5) if the student tries to move to a new exercise before sufficiently exploring the current one. In this situation the Coach tries to encourage the student to continue exploring the current exercise by offering them a hint if they stay. However, in keeping with the theme of student-controlled exploration, the student ultimately decides whether or not to move on by selecting either the *Stay* or *Move On* button presented in the dialog window.



I don't think you have explored this exercise enough...

If you stay, you can get a hint...

Figure 5.5. Example Coach intervention

5.2 Applying Our User Modeling Framework to ACE

5.2.1 Data Collection for ACE

The data we use for the current research was obtained from a previous user study investigating how to model meta-cognitive behaviors of students using the ACE Plot Unit [31, 56]. The particular meta-cognitive behavior studied is known as *self-explanation* [24]. Self-explanation is the domain-independent skill of self-generating interpretations and reasoning about instructional material. This behavior has been shown to improve learning [24, 26, 39] and so is modeled in ACE's DBN (discussed above in Section 5.1) as one of the factors that determine the effectiveness of a student's exploration.

The original ACE's user model used only the latency between two actions as implicit evidence of a self-explanation episode. However, latency alone is not a robust measure of self-explanation as it does not even assure that the student is attending to the material. For this reason, eye-tracking data was obtained during the user study to investigate whether the addition of specific eye-gaze patterns would better estimate self-explanation of individual exploration cases. The eye-gaze patterns explored were direct and indirect gaze shifts between the plot and equation areas, because, intuitively, shifting attention between the plot and equation areas is a good indication of self-explanation of the current exploration case. Figure 5.6 shows an example of a direct gaze shift pattern after an exploratory action in the Plot Unit. Here the student's gaze starts from the function equation and shifts directly to the plot region, suggesting that the student was trying to understand the connection between the equation and plot. An indirect gaze shift is defined as moving from one of these regions, to a non-salient region and then to the other salient region.

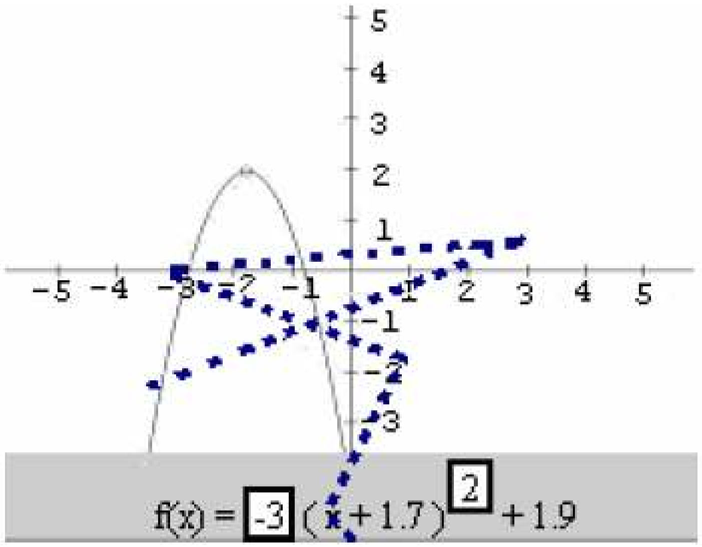


Figure 5.6. Example gaze shift

A total of 36 students participated in the user study. The participants were all university students that had not taken high school calculus or college level math. Before using ACE, each student took a 15 minute pre-test on mathematical functions. Then the student interacted with ACE for as much time as needed to experiment with all the units. Each unit was equipped with several exercises for the student to explore, with the Plot Unit providing three exercises corresponding to the three available types of functions (constant, linear, and power). While using ACE, students were asked to follow a think-aloud protocol in which they should try to verbalize all of their thoughts. The student's gaze was tracked by a head-mounted, Eyelink I eye-tracker developed by SR Research Ltd., Canada. In addition, all student interactions with ACE were time-stamped and logged, and were synchronized with the raw data from the eye tracker. Finally, the students took another 15 minute post-test that differed from the pre-test only on parameter values and question ordering.

For the research presented in this thesis we only used the data involving ACE's Plot Unit from the previous user study. The pre and post-tests used in the previous study were devised in such a way as to evaluate relevant knowledge gained from each ACE unit separately. This allowed us to extract the test results for only the Plot Unit. For the purposes of the current research, we use 3783 interface actions (recorded over 673.7 minutes) obtained from the previous study's log files, along with the accompanying gaze shift data computed from the eye-tracker. The 13 types of interface actions (detailed in the previous section) include:

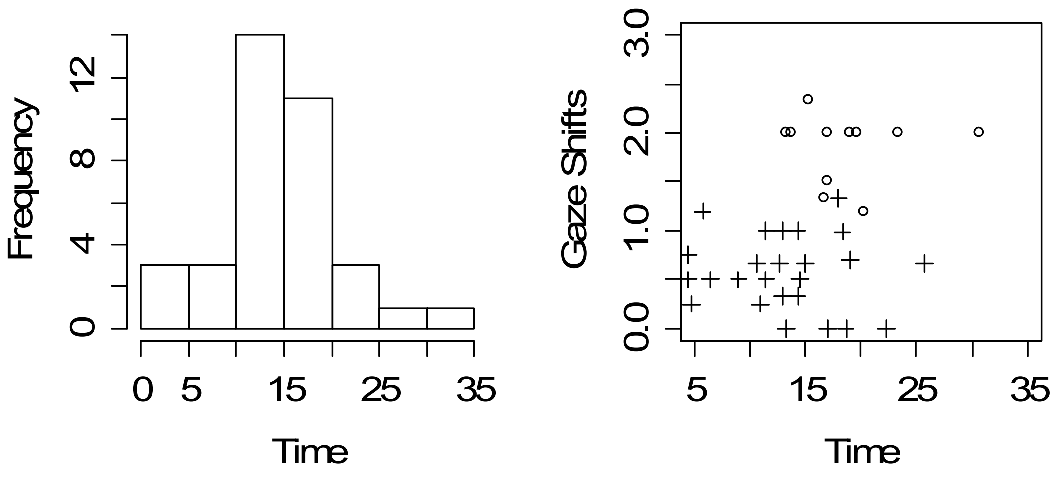
- *Plot Move* (*PM*) – Dragging the function plot around the screen. The parameters of the function's equation are automatically adjusted to reflect the transformation.
- *Equation Change* (*EC*) – Editing the function equation. ACE transforms the function plot accordingly.
- *Reset* – Resetting the function to its initial parameters.
- *Next Exercise* (*NE*) – Stepping sequentially forward to the next exercise in the pre-defined curriculum by clicking on the *NE* button.
- *Step Forward* (*SF*) – Same as *NE* action, but done by clicking on the forward arrow button.
- *Step Back* (*SB*) – Stepping backwards to the previous exercise by clicking on the backward arrow button.
- *Lesson Browser* (*LB*) – Opening the *LB* tool which outlines the curriculum and allows the student to jump to any exercise within the curriculum.
- *Exploration Assistant* (*EA*) – Opening the *EA* tool which displays the exploration cases already examined by the student and remaining to be examined.
- *Get Hint* (*GH*) – Requesting a hint from the Coach.
- *Stay* – Adhering to the Coach's advice to continue exploring the current exercise.
- *Move On* (*MO*) – Ignoring the Coach's advice to stay on the current exercise and moving on to another one.
- *Help* – Using the hypertext help pages.
- *Zoom* – Zooming into or out of the graph region.

The data from the previous user study was also used to build a new version of the ACE user model [29] using a supervised data-based approach with hand-labeled data (see Section 2.4). This new model used gaze information, in addition to latency between actions, to assess effectiveness of student exploration for learning. The use of the gaze information was limited to direct and indirect gaze shifts only after an *equation change* or *plot move* action. Although gaze shifts may also be relevant after other interface actions, this work was limited to *equation changes* and *plot moves* because of the effort required to generate the hand labeled data necessary to train the user model. Two researchers (to assure coding reliability) labeled each student's verbalizations after every *equation change* and *plot move* as an instance of reflection or speech not conducive to learning. Then, they mapped the labels onto presence/absence of gaze shifts and latency until the next action. This new model showed better performance in assessing effectiveness of student exploration than models using only action occurrences or action occurrences plus latency information, showing the value of eye-tracking data for this type of assessment. In the following sections, we compare the results of applying our framework to the data described above, with the results obtained by the supervised data-based approach in [29].

5.2.2 Preprocessing and Unsupervised Clustering for ACE

We extracted two different sets of features from the ACE study data. The first set (FeatureSet1) consisted only of interface features, i.e. frequencies of each of the 13 possible interface actions (see Section 5.2.1) and the mean and standard deviation of the latency between actions. Recall that the latency dimensions are intended to measure the average time a student spends reflecting on the results on an action, as well as the general tendency of the student to spend time reflecting on that action. This feature set is analogous to the one used in our first experiment (Chapter 4). We chose this feature set in order to evaluate how our modeling framework transfers across different applications using the same type of input data.

The second feature set (FeatureSet2) included features distilled from the eye-tracking data in addition to the above interface features. We chose this set for two reasons. First, we wanted to evaluate how our approach works on a range of different data sources. Second, we wanted to see if we could reproduce results in [29], showing that eye-tracking information improves assessment of the effectiveness of student exploration (see Section 5.2.1). In particular, we hypothesized that eye-tracking data would improve the performance of clustering in identifying groups of students with distinct learning proficiency. Figures 5.7a and 5.7b show an example of how the inclusion of eye-tracking data helps to reveal clusters of similar behavioral patterns. The histogram on the left shows the distribution of the time spent reflecting on the results of a particular action over all of the students. No clusters are visible within this roughly Gaussian distribution. However, two clusters are apparent within the plot of gaze shifts against time after the same action on the right in Figure 5.7b (one cluster is marked with crosses while the other is marked with circles). And in fact these two clusters are identified by clustering, using FeatureSet2, and are discussed in detail later in this section.



Figures 5.7a and 5.7b. Histogram of time spent after an action (left), plot of gaze shift versus time with two visible clusters (right)

Although in [29] the authors only considered gaze shifts after *plot move* and *equation* actions, they may be relevant after most ACE interface actions. For example, after a *next exercise* action, a new function appears on the screen requiring attention to both the plot and equation regions in order to understand the connection between the new function equation and its plot. In another example, when the Coach presents a hint after a *get hint* action (e.g., “Why don’t you try a large negative number as the input?”), the student may attend to the graph and equation regions while considering the Coach’s advice and deciding whether or not to follow the suggestion. Since, contrary to the supervised data-based approach in [29], considering more actions in our approach does not involve much extra work, we included gaze shift information for all of the 13 interface actions in FeatureSet2 by computing the mean and the standard deviation of the number of indirect and direct gaze shifts as additional features. It should be noted, however, that we do not see a meaningful connection between the *Lesson Browser* and the current function, and so we hypothesize that associations between gaze shift patterns and the use of this tool may be inappropriate. However, we retain gaze shift features for this tool in order to evaluate this conjecture with the results from feature selection.

FeatureSet1 and FeatureSet2 included 39 and 91 possibly influential features, respectively. With only 36 feature vectors corresponding to the 36 study participants, these high-dimensional feature spaces can result in data sparseness and may degrade the performance of clustering. Therefore, as outlined in our modeling framework (Chapter 3), we performed entropy-based feature selection (see Section 3.1.2) on each set in order to isolate the most discriminatory features. Recall that the feature selection method we use returns both a set of relevant features, as well as the resulting clusters using that feature set. It does this by, first, ranking each feature in a set according to the entropy (see Equation 3.1 in Section 3.1.2) induced by the removal of that feature from the data. Next, forward selection is run on the ranked features using the cluster quality criterion, defined as maximum between-cluster variance and minimum within-cluster variance (see Equation 3.5 in Section 3.1.2), to assess feature subset performance in clustering. As for our first experiment, we used k set to 2, 3 and 4 for the k-means clustering executed during forward selection. Again, we chose these values because our data set was relatively small and so we only expected to find a few clear groups with distinct learning outcomes.

Table 5.1 shows the number of features selected as relevant by feature selection on FeatureSet1 and the number of students assigned to each of the resulting clusters for each case (i.e., for k set to 2, 3 and 4). Figures 5.8 to 5.10 show the resulting clusters for each case projected from the original spaces (selected by feature selection on FeatureSet1) onto 2D for visualization purposes. As the table shows, most of the interface related features (between 34 and 37 features) were retained by the automatic feature selection method that we used (see the number of features selected in Table 5.1). The features that were removed were all related to the standard deviations of the latency after some of the interface actions. For example, when *k* was set to 2, the only four features found to be irrelevant were the standard deviations of the latency after *Lesson Browser*, *Exploration Assistant*, *Get Hint*, and *Stay* actions. Note that while the general rule of thumb would suggest using between 3 to 7 features (for 36 feature vectors) for model learning [48], because we are trying to reduce the time and effort required of application and domain experts with our user modeling framework, we decided against the time-consuming and potentially inaccurate manual analysis of the features that would have been required to reduce the dimensionality of the feature space further (see Section 3.1.2).

Table 5.1. Number of features selected and cluster sizes for FeatureSet1

k	**# selected / total features**	**Cluster 1 size**	**Cluster 2 size**	**Cluster 3 Size**	**Cluster 4 Size**
2	35/39	28	8		
3	34/39	7	13	16	
4*	37/39	5	9	2	18

* Two data points, forming singleton clusters, were removed.

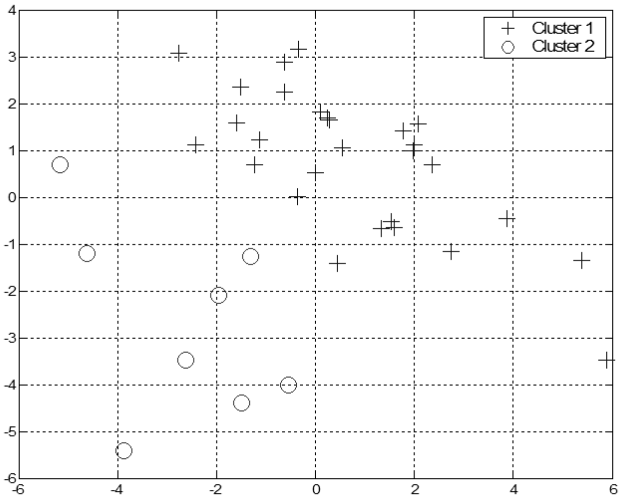


Figure 5.8. Clusters resulting from *k*-means clustering (*k*=2) on the features selected from FeatureSet1

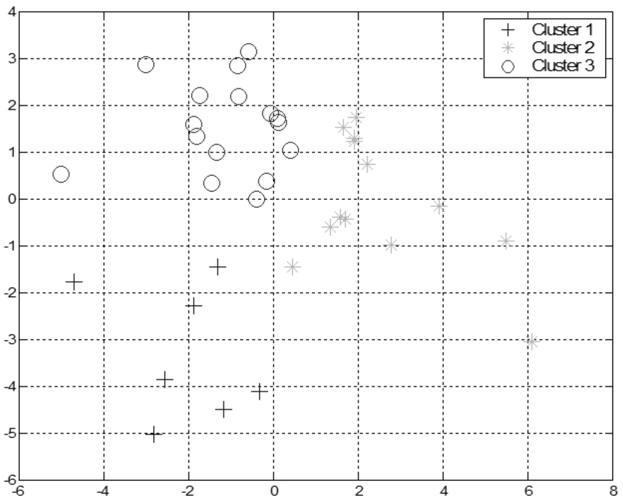


Figure 5.9. Clusters resulting from *k*-means clustering (*k*=3) on the features selected from FeatureSet1

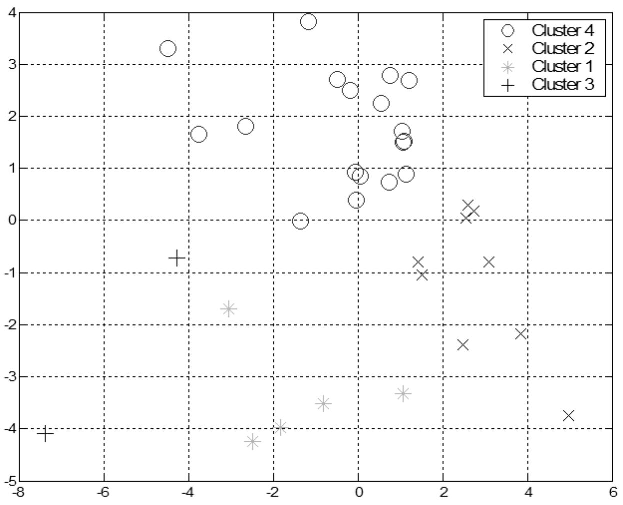


Figure 5.10. Clusters resulting from *k*-means clustering (*k*=4) on the features selected from FeatureSet1

Table 5.2 shows the number of features selected from FeatureSet2 and the number of students assigned to each of the resulting clusters for each case (i.e., for *k* set to 2, 3 and 4). Again, Figures 5.11 to 5.13 show the resulting clusters for each case projected onto 2D.

Table 5.2. Number of features selected and cluster sizes for FeatureSet2

k	**# selected / total features**	**Cluster 1 size**	**Cluster 2 size**	**Cluster 3 size**	**Cluster 4 Size**
2	36/91	11	25		
3	33/91	10	10	16	
4	37/91	8	8	4	16

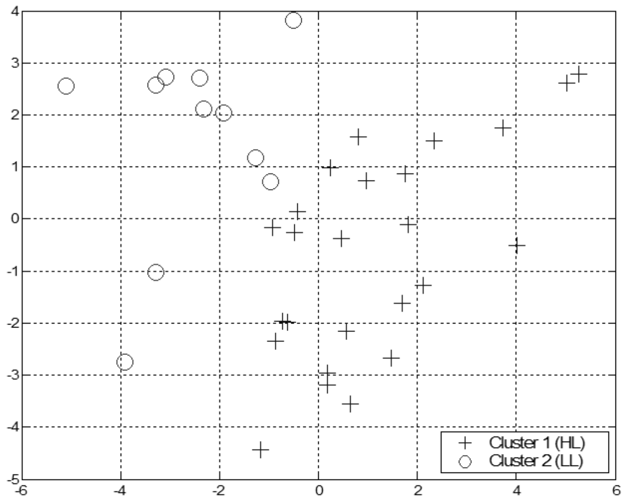


Figure 5.11. Clusters resulting from *k*-means clustering (*k*=2) on the features selected from FeatureSet2

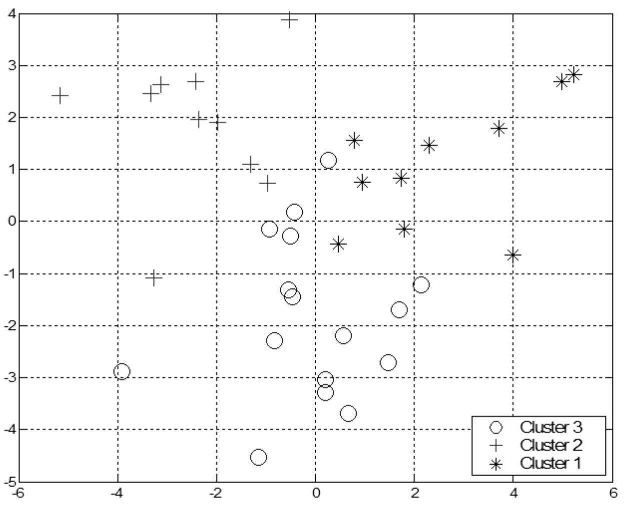


Figure 5.12. Clusters resulting from *k*-means clustering (*k*=3) on the features selected from FeatureSet2

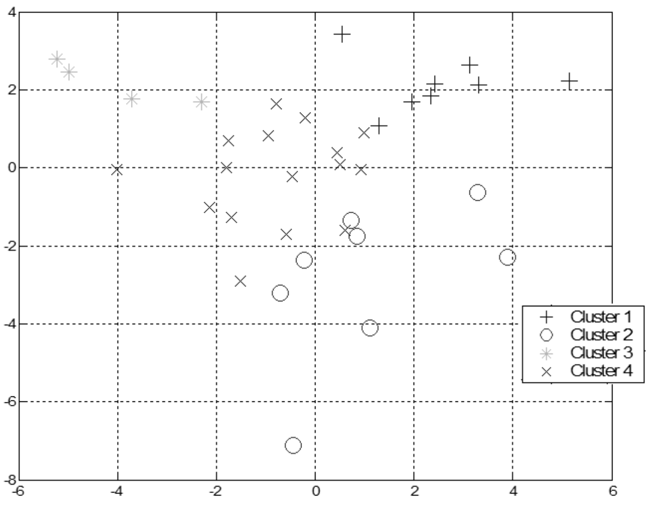


Figure 5.13. Clusters resulting from *k*-means clustering (*k*=4) on the features selected from FeatureSet2.

Approximately one third of the same features from FeatureSet2 were found to be relevant by feature selection for each of the three cases, i.e., *k* set to 2, 3 and 4 (see the number of features selected in Table 5.2). Again, we decided against performing manual feature selection to reduce the number of feature dimensions further. As we shall see in the following *Cluster Analysis* section, we only found significant differences in student learning gains between the two clusters obtained by feature selection (and consequent *k*-means clustering) on FeatureSet2 with *k* set to 2 (see Figure 5.11). Therefore, for the rest of this section we focus on the feature selection results from this case. Table 5.3 lists the top 36 features selected by feature selection from FeatureSet2 (*k*=2) and their rank. Table 5.4 lists the 55 (ranked) features discarded by feature selection for this case. It is important to note that the rankings obtained do not necessarily relate directly to the relevance of each individual feature. For example, some features may only be important in combination with another top ranked feature or subset of features and may be inconsequential independently. In light of this, we do not take rank into consideration when discussing the selected features in this section.

Table 5.3 Features selected from FeatureSet2 (*k*=2)

Relevant Features	
Rank	**Feature Description**
14	*PM* frequency
31	*EC* frequency
9	*EC* latency average
30	*EC* latency SD
21	*EC* indirect average
19	*EC* indirect SD
8	*Reset* frequency
36	*Reset* latency SD
20	*NE* frequency
5	*NE* latency average
18	*NE* latency SD
6	*NE* indirect average
7	*NE* indirect SD
10	*NE* direct average
22	*NE* direct SD
27	*SF* frequency
2	*SF* latency average
1	*SF* latency SD
28	*SB* frequency
23	*SB* latency average
33	*SB* indirect average
32	*LB* frequency
11	*EA* frequency
26	*EA* latency average
13	*GH* frequency

Table 5.4 Features discarded from FeatureSet2 (*k*=2)

Irrelevant Features	
Rank	**Feature Description**
40	*PM* latency average
67	*PM* latency SD
45	*PM* indirect average
50	*PM* indirect SD
66	*PM* direct average
72	*PM* direct SD
81	*EC* direct average
84	*EC* direct SD
51	*Reset* latency average
58	*Reset* indirect average
42	*Reset* indirect SD
53	*Reset* direct average
38	*Reset* direct SD
46	*SF* indirect average
49	*SF* indirect SD
47	*SF* direct average
52	*SF* direct SD
61	*SB* latency SD
62	*SB* indirect SD
63	*SB* direct average
64	*SB* direct SD
70	*LB* latency average
76	*LB* latency SD
54	*LB* indirect average
37	*LB* indirect SD
55	*LB* direct average

Rank	Feature Description
3	*Stay* latency average
25	*Stay* indirect average
4	*MO* frequency
34	*MO* latency average
35	*MO* latency SD
12	*Help* frequency
24	*Help* latency average
17	*Zoom* frequency
29	*Zoom* latency average
15	*Zoom* latency SD
16	*Zoom* direct SD

Rank	Feature Description
65	*LB* direct SD
77	*EA* latency SD
48	*EA* indirect average
73	*EA* indirect SD
44	*EA* direct average
39	*EA* direct SD
75	*GH* latency average
74	*GH* latency SD
71	*GH* indirect average
68	*GH* indirect SD
79	*GH* direct average
78	*GH* direct SD
59	*Stay* frequency
82	*Stay* latency SD
91	*Stay* indirect SD
69	*Stay* direct average
86	*Stay* direct SD
80	*MO* indirect average
85	*MO* indirect SD
89	*MO* direct average
90	*MO* direct SD
47	*Help* latency SD
60	*Help* indirect average
83	*Help* indirect SD
88	*Help* direct average
87	*Help* direct SD
57	*Zoom* indirect average
43	*Zoom* indirect SD
56	*Zoom* direct average

Feature selection reveals that all action frequencies are important in cluster formation, except in the case of a *stay* action (see '*Stay* frequency' in Table 5.4). Gaze shift dimensions are only identified as important in the presence of the corresponding latency dimensions after an action. For example, indirect gaze shifts were found to be relevant in addition to the latency dimensions after an *equation change* action (see *EC* in Table 5.3), and similarly for *next exercise*, *step back*, *stay*, and *zoom* actions (see *NE*, *SB*, *Stay*, and *Zoom* in Table 5.3). Conversely, latency was found to be relevant independently of gaze shift features after some actions (namely after *reset*, *step forward*, *Exploration Assistant*, *move on*, and *help* actions, see corresponding entries in Table 5.3). This agrees with the findings in [56] that gaze shifts may be important mostly in discriminating between time spent self-explaining the results of an action and idle time. Also, gaze shifts were not found to be relevant when using the *Lesson Browser* tool as was anticipated (see *LB* in Table 5.4).

Interestingly, neither latency nor gaze shifts were found to be relevant after a function *plot move* (see *PM* entries in Table 5.4). Given that both *plot moves* and *equation changes* are exploratory actions requiring equivalent self-explanations, this result is unintuitive especially considering that latency and gaze shifts were found to be important after *equation changes.* This could be an artifact of the forward selection strategy used. Forward selection can efficiently search through the exponential space of possible feature subsets, but it may prematurely rule out certain features that may only be important in combination with features not yet included in the subset (i.e. lower ranked features). To test our hypothesis that the *plot move* latency and gaze shift features were indeed important, we attempted clustering on the same feature set with the addition of these features. No significant differences in learning gains were found between any of the clusters (see Table 5.5). This does not rule out the possibility that these *plot move* features are important. It is possible that the *plot move* latency and gaze shift features may be important in combination with some of the other features removed by feature selection. Alternatively, only some subset of the *plot move* latency and gaze shift features may be important. For example, since the *plot move* latency average and *plot move* latency standard deviation features were retained by feature selection in FeatureSet1, adding only these, and not the gaze features, to FeatureSet2 and re-attempting clustering could reveal significant differences in learning gains. Attempting our user modeling framework on alternative feature sets, such as this, will be discussed further in the Future Work section (see Chapter 7). Nevertheless, as will be discussed in the following section, clustering was in fact able to find distinct learner groups using only the features returned by feature selection. Therefore, these findings could challenge out prior beliefs about the utility of *plot move* exploratory actions for learning.

Table 5.5. Summary of clustering results for FeatureSet2 (after feature selection and with *PM* latency and gaze shift features added)

k	**Cluster 1 size/ave. learning**	**Cluster 2 size/ave. learning**	**Cluster 3 size/ave. learning**	**Cluster 4 size/ave. learning**	**df(s)**	***T* or F**	***p***	***d* or *partial* η^2**
2	21/1.33	15/2.27			31.1	.922	.182	.310
3*	6/2.17	19/.684	10/3.20		2,32	2.48	.101	.138
4	7/3.86	20/.900	3/1.00	5/2.00	3,31	1.84	.160	.131

* Two data points, forming singleton clusters, were removed.

5.2.3 Cluster Analysis for ACE

As dictated by our framework, in this phase we first compare the clusters returned by feature selection on FeatureSet1 and FeatureSet2 in terms of student learning gains (derived from the pre and post-test scores available from the user study described earlier). When significantly (or marginally significantly) different learning gains were found, we then compared the clusters in terms of differences in behavioral patterns. Recall from Section 3.1.4 that we use Welch's t-test and Cohen's *d* to measure statistical and practical significance when comparing two clusters (i.e., when k=2) in terms of learning gains or behaviors, and we use ANOVAs and partial η^2 when comparing greater than two clusters (i.e., when k>2). We also use Tukey's HSD adjustments and Cohen's *d* for post-hoc pair-wise comparisons when a significant difference is detected by an ANOVA or by partial η^2.

Cluster Analysis Results

Tables 5.6 and 5.7 show the results from comparing average learning gains between the clusters found using FeatureSet1 and FeatureSet2, respectively. The tables report the average learning gains for each cluster, along with the results from the statistical and practical significance tests.

Table 5.6. Summary of learning gains comparisons between the clusters found using FeatureSet1

k	**Cluster 1 ave. learning**	**Cluster 2 ave. learning**	**Cluster 3 ave. learning**	**Cluster 4 ave. learning**	**df(s)**	***t* or F**	***P***	***d* or *partial* η^2**
2	1.39	2.88			10.0	1.12	.144	.471
3	1.88	.692	2.50		2,33	1.33	.280	.074
4	1.80	.556	2.50	3.00	3,30	.627	.603	.059

Table 5.7. Summary of learning gains comparisons between the clusters found using FeatureSet2

k	**Cluster 1 ave. learning**	**Cluster 2 ave. learning**	**Cluster 3 ave. learning**	**Cluster 4 ave. learning**	**df(s)**	***t* or F**	***P***	***d* or *partial* η^2**
2	2.91	1.20			17.9	1.55	**.068***	**.571***
3	.300	2.20	2.31		2,33	1.60	.217	.089
4	2.50	2.89	2.75	.500	3,32	1.70	.187	.137

* Significant at p<.05, d>.8, or *partial* η^2>.14 (feature description and values in bold)

Table 5.6 shows that for all values of *k*, we found no significant differences in learning gains amongst the clusters found using FeatureSet1. Therefore we cannot use these clusters as the basis for the online modeling phase. Interestingly, in our first experiment (see Chapter 4), we were able to find distinct clusters of learners using only interface actions on a data set comparable in size to the data set we are using here. We hypothesize that this discrepancy is due to the differences in the nature of the domains and the interfaces of the two learning environments. This is discussed further in our comparison of our two experiments in Chapter 6.

Table 5.7 shows that we did find a marginally significant difference in learning gains between the two clusters returned by *k*-means (with *k* set to 2) on FeatureSet2. Therefore, in the following *Discussion of Cluster Analysis Results* section, we proceed to characterize these two clusters in terms of the interaction behaviors they represent. Hereafter, we refer to the cluster with high and low average learning gains (Cluster 1 and Cluster 2, in Table 5.7) as the ‘HL’ and ‘LL’ clusters respectively. Table 5.8 presents the results of the pair-wise analysis we did on each of the 36 feature dimensions. Significant values, and the corresponding feature dimensions are highlighted in bold.

Table 5.8. Pair-wise comparisons between HL and LL clusters along each of the 36 feature dimensions

Feature Description	**HL average**	**LL average**	**Df**	**t**	***p***	***Cohen's d***
PM frequency	.024	.034	25.6	1.23	.116	.418
EC frequency	.015	.019	19.8	.850	.203	.305
***EC* latency average**	21.8	16.1	15.3	1.78	**.047***	.677
EC latency SD	10.4	6.08	11.5	1.56	.073	.636
***EC* indirect average**	1.21	.440	12.6	2.57	**.012***	**1.02***
***EC* indirect SD**	1.12	.556	13.0	2.24	**.022***	**.886***
***Reset* frequency**	0	.001	24.0	2.60	**.008***	.735
Reset latency SD	0	.051	24.0	1.44	.082	.406
***NE* frequency**	.005	.009	33.2	2.73	**.005***	**.827***
***NE* latency average**	18.7	13.2	21.3	3.01	**.003***	**1.07***
NE latency SD	10.3	7.44	18.8	1.64	.059	.594
***NE* indirect average**	1.74	.625	16.0	5.79	**1e-5***	**2.18***
***NE* indirect SD**	2.09	.715	13.5	5.03	**1e-4***	**1.97***
***NE* direct average**	1.30	.201	10.6	3.36	**.003***	**1.41***
***NE* direct SD**	1.79	.362	10.8	3.01	**.006***	**1.25***
***SF* frequency**	.008	.011	30.3	1.90	**.034***	.621
***SF* latency average**	7.65	4.65	15.4	2.71	**.008***	**1.03***
***SF* latency SD**	9.96	5.83	19.3	2.37	**.014***	**.858***
SB frequency	0	2e-4	24.0	1.20	.122	.338
SB latency average	0	.400	24.0	1.68	.053	.475
SB indirect average	0	.040	24.0	1.00	.164	.283
***LB* frequency**	0	6e-4	24.0	1.87	**.037***	.528
***EA* frequency**	1e-4	.001	27.6	2.22	**.018***	.640
EA latency average	5.27	5.65	16.7	.072	.472	.027
GH frequency	3e-4	4e-4	34.0	.311	.312	.155
***Stay* latency average**	6.55	2.54	14.5	2.01	**.032***	.775
Stay indirect average	.152	.100	22.5	.389	.350	.136

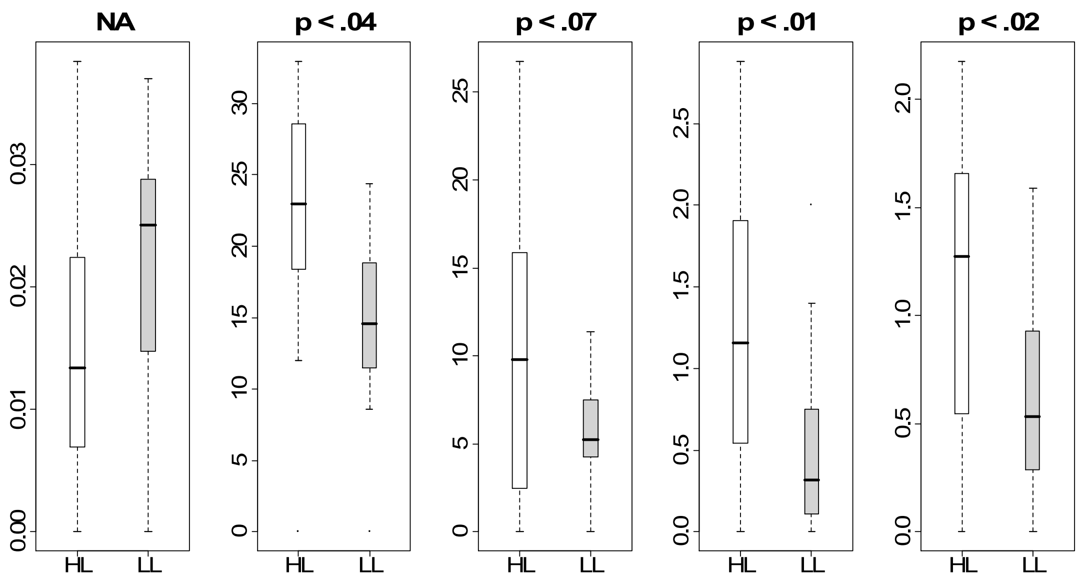
Feature Description	**HL average**	**LL average**	**Df**	**t**	***p***	***Cohen's d***
***MO* frequency**	.003	.005	32.1	2.31	**.014***	.744
MO latency average	2.23	232	24.0	1.00	.163	.283
MO latency SD	2.76	400	24.0	1.00	.163	.283
Help frequency	.002	.001	14.3	.745	.234	.288
***Help* latency average**	2.45	7.28	33.1	1.94	**.030***	.587
***Zoom* frequency**	4e-4	.021	24.1	2.62	**.008***	.741
***Zoom* latency average**	.374	1.98	29.9	2.92	**.009***	**.961***
***Zoom* latency SD**	.700	2.70	22.6	2.24	**.017***	.785
***Zoom* direct SD**	0	.101	24.0	2.41	**.012***	.683

* Significant at p<.05 or d>.8 (feature description and values in bold)

Discussion of Cluster Analysis Results

In this section, we discuss some of the most interesting findings obtained from our pair-wise analysis (presented in Table 5.8).

Some of our findings are consistent with results in [29], as we were hoping. First, there were no statistically significant differences in the frequency of *plot move* or *equation changes* between the HL and LL clusters (see '*PM* frequency' and '*EC* frequency' entries in Table 5.8), consistent with finding in [29] that sheer number of exploratory actions is not a good predictor of learning in this environment. Second, after an *equation change*, the LL students would pause for a significantly shorter duration than the HL students on average (see '*EC* latency average' in Table 5.8). In [29], the authors determined 16 seconds to be an optimal threshold between occurrences of effective reflection on exploration cases and other verbalizations not conducive to learning. Consistent with this result, the second boxplot in Figure 5.14 shows that the average latency by the students in the HL cluster were mostly above this threshold, whereas with the LL cluster the latency averages were centered about the threshold.



Figures 5.14. *Equation Change* boxplots between HL (gray) and LL (white) clusters. From left to right: frequency, latency average, latency standard deviation, indirect average, and indirect standard deviation

Because with clustering we are able to incorporate all interface actions and associated gaze data simply by including them in the multi-dimensional feature vectors, we also found patterns additional to the ones found in [29]. For example, the students in the HL cluster were more varied in how often they would indirectly gaze shift after an *equation change* (see '*EC* indirect SD.' in Table 5.8, and the last boxplot in Figure 5.14). This selective behavior suggests that students need not reflect on the results of every exploratory action in order to learn well so long as they do not consistently refrain from reflection. In addition, the LL students paused less and made significantly fewer indirect gaze shifts after an *equation change* than the HL students (see '*EC* latency average' and '*EC* indirect average' in Table 5.8, and the second and fourth boxplots in Figure 5.14). These results are consistent with less reflection by the LL students compared to the HL students and may account for some of the difference in learning gains. It should be noted that in [29], individual gaze shifts, not multiple gaze shifts, were found to predict student reasoning. In that research, gaze behavior was studied only in the context of *plot moves* and *equation changes* because of the effort of labeling data. The fact that we are using all interface actions and accompanying gaze data may account for this discrepancy in using multiple gaze shifts. We found similar differences in the latency and gaze shifting behaviors of the two clusters when a new function appeared on the screen after a *next exercise* action (see *NE* latency and gaze entries in Table 5.8).

When the Coach suggested that a student spend more time exploring the current exercise, LL students chose to ignore the suggestion and *move on* to another exercise significantly more frequently than HL students (see '*MO* frequency' in Table 5.8). This result is intuitive since the Coach's suggestions are intended to promote effective learning [17] and so ignoring them would be expected to adversely affect students. The frequency of *stay* actions were not found to be relevant by feature selection (see Table 5.4), however when they did occur, HL students paused for significantly longer than LL students (see '*Stay* latency average' in Table 5.8). This is another intuitively good behavior, possible showing that the HL students followed the Coach's advice more carefully by spending additional time pondering over the current exercise before taking additional actions.

While the above patterns are quite intuitive, this approach was also able to identify additional patterns that do not have an obvious relation to learning. For example, the LL students advanced sequentially through the curriculum using the *next exercise* and *step forward* buttons significantly more frequently than the HL group (see '*NE* frequency' and '*SF* frequency' in Table 5.8). Considering that every student examined all three available exercises, intuition would suggest that there should be no differences between the clusters along these dimensions. However, further examination of the clusters reveals that the LL students also made use of both the *step back* feature and the *Lesson Browser* tool to navigate through the curriculum, whereas none of the HL students performed these actions. Since the LL students showed lower learning gains after interacting with ACE, it is probable that these students were moving impulsively back and forth through the curriculum. This hypothesis is substantiated by the fact that the Coach's suggestion to continue exploring the current exercise (computed by combining the frequencies of *move on* and *stay* actions) appeared more frequently ($t(25.66)=1.57$, $p=.063$, $d=.536$) to the LL students than to the HL students. As this pattern involved several interface features (i.e., *next exercise*, *step forward, step back, Lesson Browser, move on* and *stay*) it may have been difficult to observe, even by application experts.

Similarly, there were unintuitive differences in the use of the *zooming* features between the two groups (see '*Zoom'* features in Table 5.8). The LL students zoomed into or out of the plot region significantly more frequently than the HL students. The HL group students paused for a consistently shorter duration after zooming than the LL students on average. Although zooming may not have clear pedagogical benefits, this behavior may suggest confusion on the part of the LL students resulting in the need for more detailed inspection of the plot. This is consistent with the finding related to *help* page exploration. Here, LL students paused for significantly longer after navigating to a *help* page then the HL students (see '*Help* latency average' in Table 5.8) indicating that these students may have felt confused about how to use ACE or about the domain concepts and so required more help than the HL students.

Unsupervised clustering also detected patterns that may reveal the inadequacy of some of ACE's interface tools. First, cluster analysis showed no differences between the groups in their use of the *get hint* feature. And in fact, very few students in either group requested exploration hints from the Coach. This result could suggest that the students preferred to explore independently, or that they had little confidence in the Coach's hints, or that they simply were not aware or did not know how to use this feature. Therefore, this result implies that further investigation is necessary to evaluate the benefits of the Coach's *get hint* feature. Also, the LL students used the *Exploration Assistant* tool significantly more frequently than the HL students (see '*EA* frequency' in Table 5.8), but still had lower learning gains. This suggests that the *Exploration Assistant* had little impact on overall learning contrary to its intended purpose of helping students better plan their exploration.

5.2.4 Supervised Classification for ACE

In this section we discuss the performance of a *k*-means-based online classifier user model (Section 3.2.1), trained with the clusters found in the offline phase, at recognizing students as belonging to either the LL or HL clusters. We performed a 36-fold leave one out cross validation (LOOCV) to evaluate how well the model generalizes to unseen data (see Section 3.3). Recall that we use an LOOCV strategy because of our small sample size. Because we are using an LOOCV strategy, we must estimate the stability cost (see equation 3.6 in Section 3.3) with respect to the clusters detected in the offline phase (as described in Section 3.2.2) before we can draw conclusions about the predictive accuracy of the user model

K-means Classifier Model Evaluation

The estimated stability cost for this *k*-means classifier user model was 0.062 after averaging the costs calculated over the 36 folds of the LOOCV evaluation (recall that 0 is considered perfect stability and 1 is considered maximum instability [51]). This means that the characteristic behaviors of the two clusters identified in the offline phase are reasonably preserved during our LOOCV evaluation.

Figure 5.15 shows the average percentage of correct predictions as a function of the percentage of actions seen by the *k*-means online classifier model (solid line). The accuracy of the model (averaged over all of the students) converges to 97.2% after seeing all of the students' actions. For comparison, the figure also shows the performance of a most-likely class baseline model that always classifies new student actions into the most-likely (or largest) class. In this case, the LL group is the largest class (25 out of 36 students), and therefore the baseline model accuracy is shown by the dashed line straight across the figure at the 69.4% (25/36) accuracy level. The figure shows that the *k*-means classifier model outperforms the baseline model after seeing only 2% of the student actions. The figure also shows the *k*-means classifier model's performance over both the HL and LL clusters (dashed and dotted lines, respectively). The accuracy for the LL group remains relatively stable over time, whereas the performance for the HL group is initially poor but increases to over 80% after seeing about 45% of the actions.

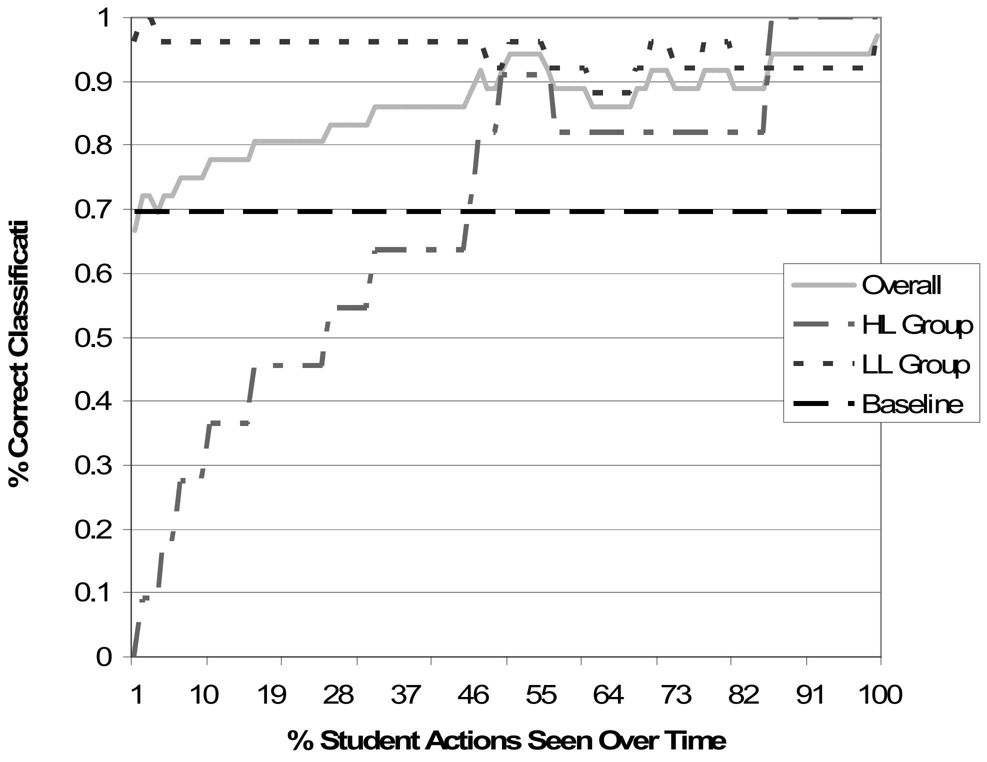


Figure 5.15. Performance of the ACE user models over time

Table 5.9 shows the accuracy, sensitivity, and specificity of the *k*-means classifier model averaged over time, and the accuracy of the baseline model averaged over time. As in our first experiment (see Chapter 4), the results show that the *k*-means based model outperforms the baseline model (86.3% accuracy for the *k*-means classifier model averaged over time compared to 69.4% accuracy for the baseline model). And again, the results indicate that while the *k*-means classifier model would be very effective in detecting behaviors that eventually result in suboptimal learning, it would more often interfere with learners that show these behaviors sporadically but may eventually be successful. As in our first experiment, this imbalance in accuracy is likely the result of the smaller sample of data from the HL cluster compared to the LL cluster. Correcting this imbalance may help increase the specificity rate of the model [82].

Table 5.9. Classification accuracies of the ACE user models averaged over time

	K-means Classifier User Model	Baseline Model
Overall Accuracy	86.3%	69.4%
Sensitivity (True Positive Rate)	94.2%	100%
Specificity (True Negative Rate)	68.3%	0%

Chapter 6

Discussion

One of the goals of this thesis is to show that our proposed modeling framework works on different domains and data sets, therefore, in this chapter, we compare and contrast the experimental results we obtained by applying the framework (Chapter 3) to two different exploratory learning environments (ELEs) and using two different types of input data (Section 6.1). At the end of the chapter, we discuss some of the limitations of our research (Section 6.2), including the limitations of our experiments, modeling framework and evaluation method.

6.1 Comparison of Experiments

6.1.1 Offline Phase

In our first experiment (Chapter 4), we applied our modeling framework to model student interactions with the CIspace CSP Applet, an ELE to support learning of an AI algorithm for constraint satisfaction problems (CSPs). In our second experiment (Chapter 5), we applied the framework to an ELE for learning mathematical functions called the Adaptive Coach for Exploration (ACE). The interfaces of both of these learning environments provide various interaction mechanisms that allow for uninhibited student exploration of the target domain.

The first step of our modeling framework (see 'Data Collection,' Section 3.1.1) involves collecting data from students interacting with the target ELE. For both of our experiments, we used logged interaction and (pre and post) test data collected from previous user studies involving the target ELEs. In our second experiment, we also used eye-tracking data collected during the ACE user study. A total of 24 students participated in the CSP applet user study, and 36 in the ACE user study.

The second step of the framework (see ‘Preprocessing,’ Section 3.1.2) entails processing the collected data to form feature vector (multi-dimensional data point) representations suitable for the subsequent unsupervised machine learning step. In our first experiment, we computed 24 feature vectors corresponding to the 24 CSP Applet user study participants. Each of these 24 feature vectors had 21 feature dimensions pertaining to student interface actions (via the available interaction mechanisms in the target ELE), including the frequencies of each action, and the means and the standard deviations of the latency between actions. In our second experiment with ACE, we computed 36 feature vectors. In this case, we tried two different sets of features (39 features in the first set, and 91 in the second). The first feature set was analogous to the CSP Applet feature set in that it only included interface related features. In the second feature set, we included gaze features extracted from eye-tracking data collected during the ACE user study in addition to the interface features.

In both of our experiments, our sample sizes were relatively small (24 and 36 feature vectors, respectively). In our second experiment, the number of feature dimensions in the two feature sets that we tried (39 and 91 dimensions, respectively), was greater than the number of samples that we had collected. Due to the combination of high-dimensionality and small sample size in this second experiment, the user models we were attempting to build via our modeling framework would have been at a very high risk of over-fitting. Therefore, in this case we applied an automatic entropy-based feature selection algorithm (Section 3.1.2) on both feature sets in order to reduce the dimensionality of the feature vectors. Feature selection reduced the number of feature dimensions to an average of 35 (averaged over the number of features selected for each of the three *k* values we experimented with, i.e., *k* set to 2, 3 and 4) for both the first and second feature sets.

After forming feature vector representations of the data, the next step in our modeling framework is to cluster the feature vectors using an unsupervised machine learning algorithm (see ‘Clustering,’ Section 3.1.3). In both of our experiments, we chose a popular algorithm called *k*-means to cluster the feature vectors. We experimented with *k* values of 2, 3 and 4 in both experiments because our sample sizes were small and, therefore, we only expected to find a few clear clusters. We always ran *k*-means 20 times and then chose the highest quality clusters (i.e., having a maximum between-cluster and minimum within-cluster variance) as the final clusters.

To help guide the developer in designing appropriate interface adaptations (which would be informed by the user model's online predictions), the clusters produced by the clustering step of our user modeling framework must be analyzed in terms of learning outcomes and behavioral characteristics (see 'Cluster Analysis,' Section 3.1.4). Because in both of our experiments we had test results prior and subsequent to students using the target ELE (i.e., the pre and post-tests), we were able to use these to objectively distinguish between effective and ineffective learner groups (i.e., clusters), and to validate the behavioral characteristics we identified by analyzing the differences between the clusters along each of the feature dimensions. If test results were not available, expert intuition would have been necessary to label the clusters in terms of learning outcomes.

In both of our experiments, cluster analysis demonstrated that unsupervised clustering was able to identify distinct clusters of students (i.e., clusters of students showing differences in learning outcomes from pre to post-tests). In addition, cluster analysis revealed several characteristic learning behaviors of the distinct clusters. Some of these characteristic behaviors were intuitive and thus reasonably explained either the effective or ineffective learning outcomes. However, as expected, some of the behaviors did not have obvious learning implications, requiring consideration of combinations of dimensions (as *k*-means does to determine its clusters), or knowledge of the student learning outcomes being explained. These latter behaviors would have been difficult to recognize and label by hand, even by application experts.

There are, also, two discrepancies between the results from our experiments that need to be examined:

1. Clustering found distinct clusters when *k* was set to 2 and 3 in our first experiment with the CSP applet, but only found distinct clusters for *k* set to 2 in our second experiment with ACE.
2. Clustering was able to find clusters within the CSP applet data using interface actions alone, whereas clustering only found distinct clusters within the ACE data when using the second feature set that we tried (i.e., that included

interface and eye-tracking data), and not using the first feature set (i.e., that only included interface actions).

Both of these discrepancies could be due to differences in the domains targeted by the two different learning environments, and their consequent interfaces. The AI algorithm that the CSP Applet is designed to demonstrate is more complex compared to the relationship between mathematical functions and their graphs that ACE demonstrates. As a result, the CSP Applet interface includes several mechanisms that allow the student to visualize and reflect on the workings of the AI algorithm, whereas ACE only provides two such mechanisms: plot moves and equation changes. Therefore, there may be a variety of different ways in which students could learn using the CSP applet's interface mechanisms (as is evident from the cluster analysis for this experiment, see Section 4.2.4), whereas there may be fewer such possibilities with ACE's interface, resulting in only two distinct clusters being identified for the latter. Furthermore, considering the variety of possible learning behaviors with the CSP Applet, interface actions alone may better able to capture student learning and reflection during exploration than interface actions alone in ACE. This hypothesis is consistent with the results in [29] showing that gaze patterns, together with action latency, predict student reflection and learning in ACE better than sheer number of actions or action latency alone. Additional data may be necessary [48] to detect distinct clusters of learners with ACE using only this first feature set.

6.1.2 Online Phase

The final step of our proposed framework calls for using the clusters detected by the unsupervised clustering step to directly train a supervised, *k*-means based, online classifier user model (see 'Supervised Classification,' Section 3.2.1). We used the same procedure in both of our experiments to evaluate the classifier user models we built via our modeling framework (see 'Model Evaluation,' Section 3.3). Each user model was evaluated using a leave-one-out cross validation (LOOCV) strategy. Cluster stability over LOOCV was first measured to substantiate the results of the online predictions. In all cases, cluster stability was high (i.e., the stability cost was less than .1 in all cases, where 0 indicates perfect stability and 1 indicates maximum instability), indicating that the characteristic cluster behaviors we identified during the cluster analyses were reasonably preserved using the LOOCV strategy.

Table 6.1 summarizes the accuracy results from the evaluation of each of the *k*-means based classifier user models that we developed via our framework for the CSP Applet and ACE. The table also shows the accuracies of the corresponding baseline models that used most-likely-class classification strategies. In all cases, the *k*-means based user models outperformed the corresponding baseline models on predicting the correct class for new student behaviors.

Table 6.1. Summary of classification accuracies averaged over time

	CSP (*k*=2)	**CSP (*k*=3)**	**ACE**
Overall Accuracy	88.3%	66.2%	86.3%
Sensitivity (True Positive Rate)	93.5%	66.1%	94.2%
Specificity (True Negative Rate)	62.4%	63.3%	68.3%
Baseline Accuracy	83.3%	50.0%	69.4%

In addition, our evaluations show that both of the two-class (*k*=2) *k*-means based classifier user models developed were able to achieve comparably good overall predictive accuracy on new student behaviors (88.3% in the first experiment with the CSP Applet, and 86.3% in the second experiment with ACE). In both cases, the accuracies were higher for predicting ineffective learning behaviors than for predicting effective ones (i.e., the sensitivity rates were higher than the specificity rates). Therefore, the two-class classifier user models built using our modeling framework would be useful in providing adaptive help for students who show ineffective learning behaviors, but may also sometimes interfere with those students who show these behaviors sporadically but eventually learn well. This is likely due to the imbalance in the distribution of the sample data [82] as the number of students clustered in the effective learner groups were fewer than those in the ineffective learner groups in both experiments.

In contrast to the two-class classifiers, the overall predictive accuracy of the three-class (*k*=3) *k*-means based classifier user model built from the three distinct clusters found in our first experiment (with the CSP applet), was only 66.2% despite the stability of the clusters. This is likely attributable to the smaller cluster sizes resulting from the larger *k* value in this case. In this case, the sensitivity rate reported in Table 6.1 for *k*=3 on the CSP Applet data was computed by combining the accuracy results for the two groups that showed ineffective learning behaviors in this case. The individual accuracies for these two groups were 80.9% and 44.9% averaged over time. Therefore, this classifier user model would be most useful for recognizing students behaving in the ineffective ways characterized by the first (larger) group, but not by the second (smaller) group.

6.2 Limitations

6.2.1 Of the Data Collected and Used for Our Experiments

The main limitation of the research presented in this thesis is that both of the data sets we collected and used in our two experiments were small. According to the general rule of thumb for model learning, which suggests between 5 to 10 times as many data samples as feature dimensions [48], the number of feature dimensions we used in each experiment was relatively high in comparison to the number of samples in both of our data sets, even after automatic feature selection in our second experiment. We initially tried to collect additional data for the CIspace CSP Applet experiment, but we had difficulty finding subjects with the appropriate background to participate. Time constraints also prevented us from collecting additional sample data as both of the user studies in our experiments were quite time-consuming (three hours for the CIspace CSP Applet user study and 80 minutes for the ACE user study). The ACE user study would have been especially time-consuming as only one subject could participate in the study at a time due to the use of the eye-tracker. Although in both of our experiments *k*-means clustering was still able to detect clusters of users distinguished by characteristic interaction behaviors and significant differences in learning outcomes, even with our small sample sizes, more data is necessary to better evaluate our framework and substantiate our results. Experimenting with more data would also help verify our hypothesis that more data would indeed improve the performance of the classifier user models, particularly for the smaller clusters such as those corresponding to the students with high learning gains (as in both of our experiments), and those resulting from clustering with a *k* value greater than 2 (as in our first experiment).

6.2.2 Of *K*-means

While the *k*-means algorithm that we chose to use for the unsupervised clustering and supervised classification steps of our user modeling framework is intuitive to understand and easy to implement, it also has some limitations. First, *k*-means clustering makes the assumption that feature dimensions are independent. However, in practice, violation of this assumption will usually not affect the quality of the clusters resulting from unsupervised machine learning [52]. This is also evident in the experiments presented in this thesis as *k*-means clustering was able to discover meaningful interaction behaviors in both cases even though some of the feature dimensions we used (e.g., the action frequency and latency dimensions in both of our experiments) may not have been independent. Feature independence is therefore a common assumption made, especially in high-dimensional data [52, 76]. Nevertheless, when the independence assumption is violated, principal component analysis (PCA) [37] could be used to generate independent features (i.e., the principal components) and reduce the dimensionality of the feature space before performing *k*-means clustering. A drawback of this is that in the Cluster Analysis section of our user modeling framework (Section 3.1.4), we interpret the clusters produced by *k*-means by analyzing them along each of the feature dimensions, but the independent features that PCA produces may not correspond to meaningful quantities [33] making this task difficult. Even if we project the data back to the original feature space before Cluster Analysis, we may not see differences along the original feature dimensions as clustering was done in the reduced feature space.

Another assumption that *k*-means makes is that the actual clusters in the data are elliptical, compact and well-separated [47]. Unlike with the independence assumption, if this assumption is violated, then *k*-means clustering will fail to discover meaningful clusters in the data. For example, Figure 6.1 shows the two distinct clusters of students found by *k*-means clustering (with *k* set to 2) on student interaction behaviors (as represented by the feature vectors) in our experiment with the CIspace CSP Applet (see Section 4.2.3). These clusters are visibly elliptical and well-separated, as dictated by *k*-means. And, as it turns out, one of these clusters of students (marked by crosses, '+', in Figure 6.1) did indeed show significantly higher learning gains (from pre to post-test) than the other cluster (marked by asterisks , '*'). However, if we were to divide the data points by considering learning gains and not interaction behaviors, then the data would take on a more complex terrain in the feature space, as shown in Figure 6.2. In this figure we used a threshold value of 5.75 marks to divide between 'high learners' and 'low learners' since this was the average learning gain observed. Clearly, these two clusters of learners are no longer well-separated in the feature space and, consequently, *k*-means clustering would not have been able to find them. However, our modeling framework is not restricted to using *k*-means for clustering. It could instead use a more flexible, but computationally more expensive, clustering algorithm such as hierarchical clustering [47] to discover meaningful clusters.

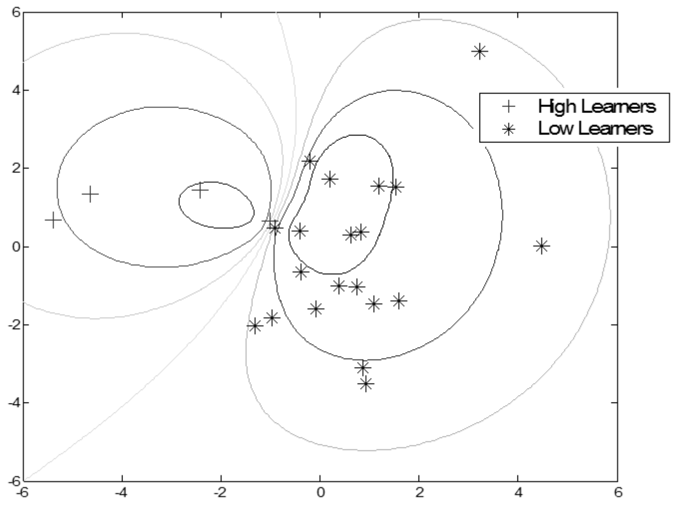


Figure 6.1. *K*-means clusters (*k*=2) from CIspace CSP Applet data

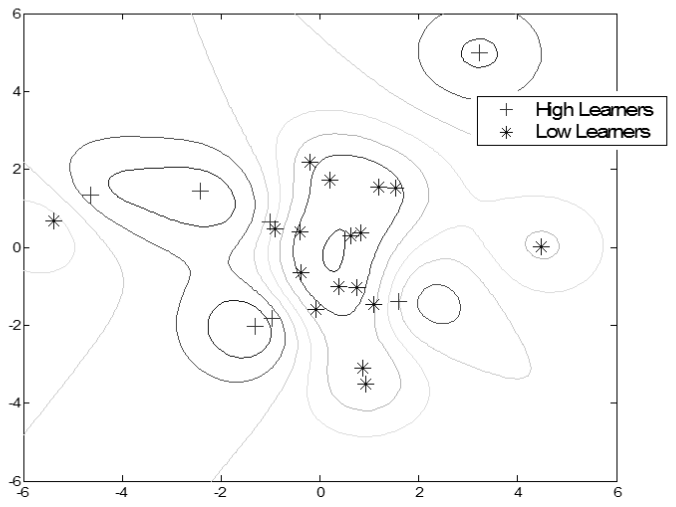


Figure 6.2. Division of CIspace CSP Applet data points by thresholding learning gains

Unfortunately, using complex or irregularly shaped clusters, such as those shown in Figure 6.2, to train our *k*-means based online classifier, poses a different problem. Our *k*-means based online classifier makes its classifications by assigning incoming data points to the cluster whose centroid is nearest, as measured by Euclidean distances (see Section 3.2.1). However, centroids are poor representations of clusters when their shapes are complex or irregular, and therefore making classifications this way may be ineffective. A possible solution to this problem is discussed in detail in the following chapter on Future Work (Chapter 7).

Another limitation of *k*-means is that a *k*-means-based classifier user model can only make hard classifications, whereas when making decisions about how to provide adaptive support we may like to take into account the certainty of our predictions. To do this we could use a probabilistic variant of *k*-means called Expectation Maximization (EM) [37] to determine the membership of every data point in each cluster.

6.2.3 Of Our Cluster Analysis Method

The main bottleneck in our user modeling framework is the Cluster Analysis step (Section 3.1.4), as this step requires manual analysis of the different clusters on each feature dimension. An alternative approach would be to automatically mine for characteristic cluster behaviors in the form of association rules [68] using an unsupervised rule learning algorithm such as the popular Apriori algorithm [1]. Association rules are of the form $[x_1, x_2, \ldots, x_n] \Rightarrow x_m$, where the x_i's in the body of the rule (left hand side) and the head of the rule (right hand side) are feature dimensions. Intuitively, association rules are 'if-then' correlations in the data. An example association rule for the high learning (HL) cluster found by *k*-means with *k* set to 2 in our first experiment with the CIspace CSP applet (see Section 4.2.4) would be "if a student is observed using the *Auto AC* feature very frequently, then the student is likely to use the *Stop* feature frequently." Automatically learning association rules such as this could help reduce the time and effort required to manually analyze and characterize the clusters in the Cluster Analysis step of our modeling framework. Furthermore, the algorithms for automatically learning these rules were designed for sparse data sets (i.e., that have high-dimensional spaces and few data points), such as the data sets that we used in both of our experiments.

One drawback of using rule learning algorithms is that most of them require that continuous feature dimensions, such as the ones that we use in our experiments (e.g., action frequencies), be manually discretized before rules can be discovered. Discretizing continuous attributes can result in information loss [6]. Our approach of manually analyzing the clusters returned by *k*-means clustering does not require discretization of the data. Another, more major, drawback to rule learning is that association rules are descriptive in nature [59] (i.e., they try to summarize information and extract patterns), whereas we are trying to do a predictive task (i.e., we are trying to build user models for online predictions). There is no obvious way of using the learned association rules for online predictions of student learning outcomes.

6.2.4 Of Any Classifier User Model

One of the drawbacks of developing any user model for classifying students as either effective or ineffective learners, is that it does not allow isolation of the specific suboptimal behaviors that are causing the student to be classified in a specific class of learners at any given time. Thus, an adaptive ELE informed by a classifier user model would not be able to generate precise interventions targeting the suboptimal behavior that the student is currently showing. However, an adaptive ELE could use the classifier's results for general hints and interface adaptations to promote more effective learning behavior. Such adaptations are discussed further in the following chapter on Future Work (see Chapter 7).

6.2.5 Of Our Model Evaluation Method

A caveat discussed prior to describing our model evaluation method in Section 3.3 is that we cannot ensure the effectiveness of the models built via our framework in a real world setting without performing live adaptations based on those models for new students interacting with the ELEs. This would require developing an adaptive support facility that uses the classification information derived from our models in a meaningful way. This is indeed the long term goal of this research. However, we took efforts to validate the results of our evaluation strategy by measuring the stability of the clusters used to train the online classifiers prior to assessing each user model's predictive accuracy.

Chapter 7

Future Work

The next step of this research is to address the limitations discussed in the previous chapter (see Chapter 6). In particular, we would like to collect more data to strengthen the results of our experiments. In addition, we would like to experiment with alternative methods for the clustering and cluster analysis steps of our user modeling framework. For instance, we plan to use hierarchical clustering [47] or expectation maximization [37] instead of *k*-means clustering, and automatically learn association rules [68] instead of manually analyzing the clusters.

One of the limitations of *k*-means discussed in Chapter 6 (see Section 6.2.2) is that *k*-means clustering will fail when the actual clusters in the data are irregularly shaped and not well-separated. Computationally more expensive clustering algorithms, such as hierarchical clustering [47], could be used in these cases, however, a *k*-means based classifier trained with irregularly shaped clusters would poorly predict the classes of incoming data points (i.e., new students) as it uses distances to cluster centroids to make its predictions (see Section 3.2.1).

An alternative to using *k*-means for classification in our user modeling framework, particularly when clusters are complex or not well-separated (as can occur when using algorithms such as hierarchical clustering for example), is to use Gaussian Processes (GPs) [67]. We now briefly describe how to build a GP-based online classifier [83] user model. We then present preliminary results of building and testing a GP-based online classifier for the CIspace CSP Applet ELE (see Chapter 4). For simplicity, we restrict our GP user model to handle only binary classifications.

Consider a fully connected graph over the training data, where the graph nodes are the feature vectors. The edges of the graph correspond to similarity weights w_{ij} (given by a radial basis function) between feature vectors i and j, much like how *k*-

means clustering measures similarities between data points in a Euclidean feature space (see Section 3.1.3). We can then define an error function over the graph:

$$E(\boldsymbol{f}) = \tfrac{1}{2} \Sigma_{ij} \; w_{ij} \, (\boldsymbol{f}(i) - \boldsymbol{f}(j))^2 \tag{7.1}$$

such that minimization of this error function will result in a GP-based classifier user model, $\boldsymbol{f}$, that can assign similar labels to similar feature vectors. Thus, like the *k*-means based classifier user model that uses cluster distances to make predictions, the GP user model also exploits the structure in the training data to make its online classifications. However, unlike the *k*-means based classifier, the GP model does not require that the training data (i.e., the clusters) be compact or well-separated.

To demonstrate that a GP-based user model can work with well-separated training data, as well as complex training data, we developed and evaluated two GP user models with the CSP Applet data. Like the *k*-means based classifier we developed for the CSP Applet in this research (see Section 4.2.5), our first GP-based user model was trained with the elliptical and well-separated clusters found by *k*-means clustering (with *k* set to 2) on the CSP Applet user study data (see Figure 6.1 in Chapter 6). Our second GP classifier model was trained with the complex clustering obtained by thresholding the learning gains of the students represented in the data (see Figure 6.2 in Chapter 6).

To evaluate each of these GP-based user models, we performed a 24-fold leave-one-out-cross-validation over the training data since we only had 24 data points (see Section 3.3). Figure 7.1 shows that the performance of our first GP-based user model (solid line) in predicting the correct classes of new students as they interact with the CSP Applet, is comparable to the performance (two-class) *k*-means based user model (dashed line), as would be expected since *k*-means can also handle well-separated data. The accuracy of the GP-based user model, averaged over time, is 91.5%, whereas that of the *k*-means user model is 88.3% (see Section 4.2.5).

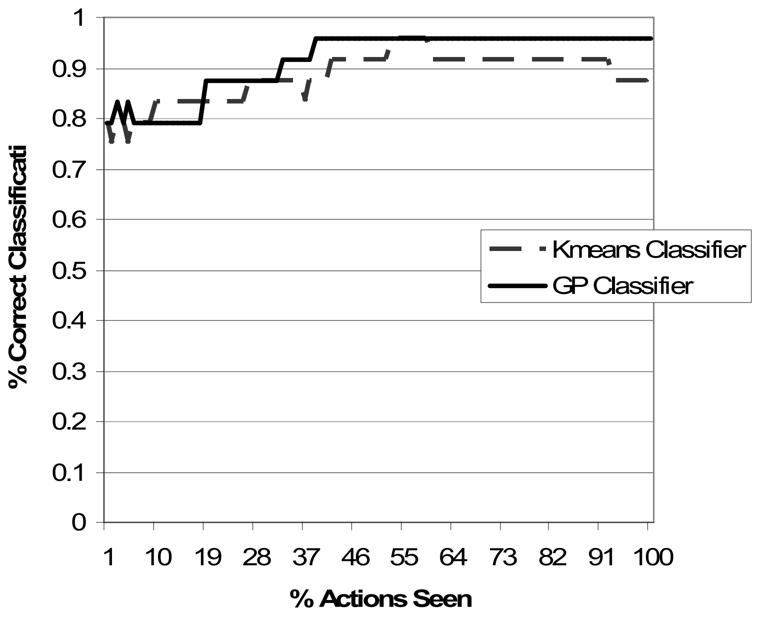


Figure 7.1. Performance of *k*-means and GP-based user models trained with the *k*-means clusters

In contrast, when using the clusters obtained by thresholding the students' learning gains (see Figure 7.2), the *k*-means based classifier (dashed line) is no better than chance at predicting the classes of incoming data points, whereas that GP-based classifier performs at least moderately (solid line). The accuracy of this GP user model averaged over time is 63.4%, whereas the *k*-means classifier model is only 42.6%.

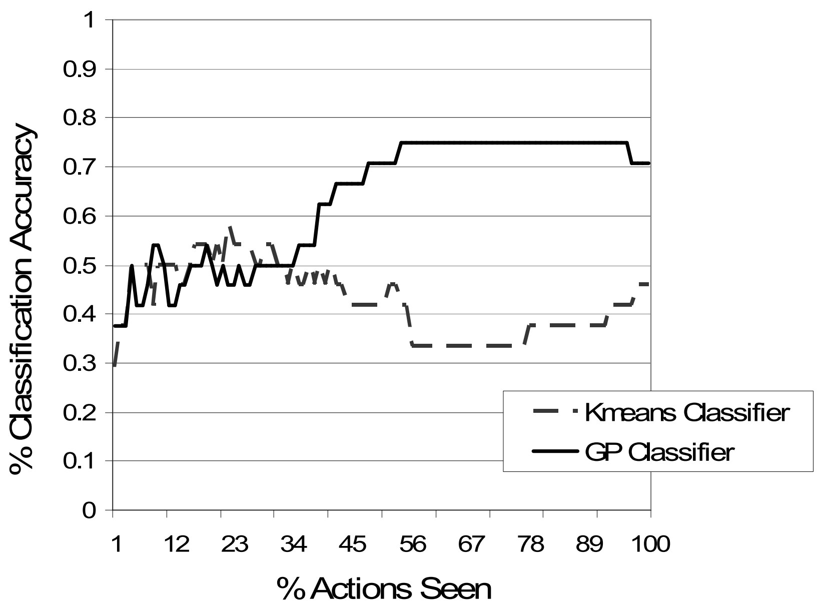


Figure 7.2. Performance of the *k*-means and GP-based user models trained with the complex clustering obtained by thresholding learning gains

Therefore, since the GP-based classifier user model can handle compact and well-separated clusters just as well as the *k*-means classifier and can also handle complex or irregularly shaped clusters, the GP-based classifier could be substituted for the *k*-means based classifier in the 'Supervised Classification' step of our user modeling framework (see Section 3.2.1).

In the future, we would also like to experiment with a variety of other features than the ones we included in the experiments presented in this thesis. In both of our experiments, we included features pertaining to interaction behaviors (i.e., action frequencies and latencies) in the feature vector representations of students (see Section 3.1.2). However, other features could also be distilled from data logs. The Educational Data Mining Working Group at Intelligent Tutoring Systems 2006 produced a list of such features [44], some of which we could include in future applications of our framework (e.g., action sequences). In our second experiment, we also included specific eye-gaze features (i.e., the mean and standard deviation of the number of indirect and direct gaze shifts) in the feature vector representations of students (see Section 5.2.2), but other possible eye-gaze features could also reveal interesting learning behaviors. For example, we could experiment with removing the distinction between indirect and direct gaze shifts when computing the eye-gaze features. We could also use binary gaze shift features, comparable to the approach taken in [56], where every interface action is either accompanied by a gaze shift or not. And, in Section 5.2.2, we discussed the possibility of experimenting with manually adding back certain eye-gaze features (i.e., gaze shifts after *plot move* actions) that were removed by automatic feature selection but that we believed were relevant.

Our long term goal is to design an adaptive support facility that takes as input the online classification information from the user models build via our framework. For example, an adaptive ELE could employ a multi-layered interface design [69], where each layer's mechanisms and help resources are tailored to facilitate learning for a given learner group. Then, based on a new learner's classification, the ELE could select the most appropriate interface layer for that learner. For instance, the CIspace CSP Applet ELE (Chapter 4) may select a layer with *Fine Step* disabled or with a subsequent delay to encourage careful thought for those students classified as ineffective learners (see Section 4.2.4) by the two-class *k*-means based classifier user model developed using our modeling framework in our first experiment. Similarly, for the three-class case, the CSP Applet ELE could disable or introduce a delay after *Fine Step* for students classified into either of the ineffective learner groups. Additionally in this case, the Applet could also include a delay after *Domain Splitting* for students classified into the LL2 (low learning 2) group as these students were consistently hasty in using this feature (see Section 4.2.4). The other ineffective learner group, LL1 (low learning 1), discovered by our framework in this experiment was characterized by lengthy pauses after *Domain Splitting* as well as *Backtracking* indicating confusion about these Applet mechanisms or concepts (see Section 4.2.4). Therefore, general tips about *Domain Splitting* and *Backtracking* could be made more prominent for these particular students for clarification purposes.

After developing adaptive systems that can use the classifier user models' predictions, we can then empirically evaluate the effectiveness of the user models in a real world setting. To give credence to our approach, we would also like to compare the performance of adaptive ELEs that use models constructed via our modeling framework against those built by traditional knowledge-based or supervised methods with hand-labeled data.

Chapter 8

Conclusions

Traditional approaches to user modeling for intelligent learning environments (ILEs) rely heavily on expert knowledge and intuition about the target application and domain. This type of knowledge can be difficult and time-consuming to elicit, especially when the elements being modeled (such as effective or ineffective learning behaviors) are complex or hard to determine through standard observations, as is often the case with exploratory learning environments (ELEs) because of the highly unstructured and unconstrained learning paradigm that they support. Some data-based approaches to user modeling that use supervised machine learning techniques have been able to reduce these development costs, but mostly when labels for the data are readily available from the application (e.g., student answers in more controlled, problem-solving learning environments). However, when labels are not readily available, they must be manually supplied, often through video or live observations and laborious data analyses. In this thesis, we devised a data-based user modeling framework for intelligent learning environments that utilizes both unsupervised and supervised machine learning to address some of the practical difficulties of constructing user models, particularly for unstructured ELEs.

8.1 Satisfaction of Thesis Goals

8.1.1 Presentation of User Modeling Framework

The first goal of this thesis was to outline a data-based framework for user modeling that addressed the practical challenges of building user models for learning environments, and especially for ELEs. These challenges include (*i*) the unavailability of explicit evidence of learning in ELEs (such as the correctness of student answers), making supervised machine learning approaches that use system-provided labels unsuitable, (*ii*) the difficulty of recognizing meaningful learning behaviors in highly unstructured exploratory environments, even for application experts, making knowledge-based or supervised approaches that require hand-labeled data costly and unappealing, (*iii*) the novelty of the exploratory learning paradigm, making experts hard to even find.

In Chapter 3, we presented a framework that takes a statistical pattern recognition approach to user modeling, making use of both unsupervised and supervised machine learning in order to address these aforementioned challenges. The framework advocates using unsupervised machine learning to automatically detect distinct behavioral patterns of users interacting with a learning environment. Therefore, instead of experts having to observe and search for meaningful learning behaviors, they are automatically presented with a picture of common patterns which they can then analyze in terms of their effects on learning. In addition, the behavioral patterns identified by unsupervised learning can be used directly to train a classifier user model by using a supervised machine learning technique. The online classifications by this user model can then inform the adaptations of an intelligent learning environment. Therefore, by maintaining a data-based design, and employing unsupervised as well as supervised machine learning, this framework is intended to reduce some of the costs faced by user model developers.

8.1.2 Framework Evaluation

Our second thesis goal was to evaluate our user modeling framework by answering the following three questions:

i. How well does the framework automatically identify meaningful learning behaviors?
ii. How well does a user model built via the framework perform at classifying new users online?
iii. How well does the framework transfer across different applications and data types?

In order to answer these questions, we implemented the techniques prescribed in the framework, applied them to two different ELEs and using two different data sources (interaction logs and eye-tracking data), and then analyzed the experimental results.

The answer to the first question can be determined by the results of completing the 'Cluster Analysis' step of our user modeling framework during our two experiments (see Section 4.2.4 for the CSP Applet, and Section 5.2.3 for ACE). In both experiments, we show that unsupervised machine learning (i.e., clustering) was able to identify several behaviors either beneficial or detrimental for learning out of the multi-dimensional sets of behaviors related to exploration. Some of these behaviors easily explained either successful or poor learning outcomes. Others (such as those pertaining to eye-gaze behaviors in our second experiment with ACE) confirmed the results of previous research. And finally, some of these required more complicated analyses of several interrelated behaviors to accurately explain the learning outcomes. Discovering these complex behaviors would have been a difficult task to perform manually, even by application experts. Thus, the Cluster Analysis results from our two experiments demonstrate the utility of using our framework to automatically identify meaningful learning behaviors.

Answering the second question requires evaluation of the predictive accuracies our classifier user models. As discussed earlier (see Section 6.2.5), the best way to assess the effectiveness of any user model would be to implement an intelligent learning environment (ILE) that could use the model's predictions to make live adaptations and then test the ILE in a real world setting. However, our less than ideal evaluation method (i.e., using leave-one-out-cross-validation (LOOCV) due to our limited sample sizes), still provides us with initial evidence of the predictive performance of our classifiers. In addition, we took efforts to ensure that we did not misrepresent the predictive accuracies of our classifiers by using a LOOCV evaluation strategy. We did this by measuring the stability of the training data (i.e., the clusters) over the folds of LOOCV when evaluating each of our models. In all cases, our classifier user models were determined to be reasonably stable. Also, all of the classifier user models outperformed baseline user models that used most-likely class classification strategies for making predictions. Both of the two-class classifier user models that we developed (one for each ELE) achieved good overall predictive accuracy on new user data, especially in detecting behaviors detrimental for learning. However, the predictive accuracy of the three-class user model developed for the CSP

Applet ELE in our first experiment was noticeably lower. This was likely attributable to the smaller clusters used to train this user model, and therefore, we hypothesize that the performance of this classifier will improve with additional data.

The comparable results of applying our framework to two different ELEs, for two different instructional domains, and using different types of input data, answers the third question. Specifically, we were able to follow our framework to build a user model for both the CIspace Constraint Satisfaction Problem (CSP) Applet, in Chapter 4, which uses interactive visualizations to help students understand a constraint satisfaction algorithm in artificial intelligence, and the Adaptive Coach for Exploration (ACE) in Chapter 5 which supports the exploration of mathematical functions. Furthermore, we were able to apply the framework to both logged interaction data and eye-tracking data. This, and the answers to our first two questions, show that the framework is flexible and domain/application independent.

8.2 Conclusion

Detecting learner groups and understanding their characteristic behaviors is important to provide adaptive support for those students who tend to not learn well with unstructured and unguided exploratory learning environments, but it is also very difficult. This is due to the very openness of the interaction that ELEs are designed to support, making it hard to foresee which of the many possible student behaviors may be detrimental to learning. The few existing approaches to this problem have been very knowledge intensive, relying on time-consuming, detailed analysis of the target system, instructional domain and learning processes. Since these approaches are so domain/application specific, they are difficult to generalize to other domains and applications.

In this thesis we have outlined a data-based user modeling framework that uses both unsupervised and supervised machine learning. We also experimented with applying the framework to build user models for two different ELEs. The results we obtained allow us to argue that unsupervised machine learning is valuable in reducing the practical difficulties of user modeling, especially in highly unstructured learning environments. In addition, we have proposed some ideas for future research in this area.

Bibliography

1. Agrawal, R., Imilinski, T., and Swami, A.N. Mining Association Rules between Sets of Items in Large Databases. In *ACM SIGMOD International Conference on Management of Data*, 1993, pp. 207-216.
2. Aleven, V., et al. Toward Tutoring Help Seeking: Applying Cognitive Modeling to Meta-Cognitive Skills. In *Intelligent Tutoring Systems*, 2004, pp. 227-239.
3. Amershi, S., et al. Designing CIspace: Pedagogy and Usability in a Learning Environment for AI. In *SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2005, pp. 178-182.
4. Anderson, J.R., Conrad, F.G., and Corbett, A.T., Skill Acquisition and the LISP Tutor. *Cognitive Science 13,* (1989), 467-505.
5. Arroyo, I., et al. Analyzing Students' Response to Help Provision in an Elementary Mathematics Intelligent Tutoring System. In *AIED Workshop on Help Provision and Help Seeking in Interactive Learning Environments*, 2001, pp. 34-46.
6. Aumann, Y. and Lindell, Y., A Statistical Theory For Quantitative Association Rules. *Journal of Intelligent Information Systems 20,* 3 (2005), 255-283.
7. Baker, R.S., Corbett, A.T., and Koedinger, K.R. Detecting Student Misuse of ITSs. In *ITS*, 2004, pp. 531-540.
8. Baker, R.S., Corbett, A.T., and Wagner, A.Z. Human Classification of Low-Fidelity Replays of Student Actions. In *Workshop on Educational Data Mining, Intelligent Tutoring Systems*, 2006, pp. 29-36.
9. Baker, R.S., et al. Off-task Behavior in the Cognitive Tutor Classroom: When Students "Game the System". In *SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 383-390.
10. Bareiss, R., Porter, B.W., and Murray, K.S., Supporting Start-to-Finish Development of Knowledge Bases. *Machine Learning 4,* (1989), 259-283.
11. Beck, J. Engagement Tracing: Using Response Times to Model Student Disengagement. In *Artificial Intelligence in Education*, 2005, pp. 88-95.

12. Beck, J. and Woolf, B.P. High-Level Student Modeling with Machine Learning. In *Intelligent Tutoring Systems*, 2000, pp. 584-593.

13. Beck, J., Stern, M., and Haugsjaa, E., Applications of AI in Education. *ACM Crossroads 3,* 1 (1996), 11-16.

14. Bellmann, R., *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

15. Ben-Ari, M. Constructivism in Computer Science Education. In *SIGSCE*, 1998, pp. 257-261.

16. Buchanan, B.G. and Shortliffe, E.H., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

17. Bunt, A. and Conati, C., Probabilistic Student Modeling to Improve Exploratory Behavior. *UMUAI 13,* 3 (2003), 269-309.

18. Bunt, A., Conati, C., and Muldner, K. Scaffolding Self-Explanation to Improve Learning in Exploratory Learning Environments. In *ITS*, 2004, pp. 656-667.

19. Bunt, A., et al. On Improving the Effectiveness of Open Learning Environments through Tailored Support for Exploration. In *AIED*, 2001, pp. 365-376.

20. Burton, R.R. DEBUGGY: Diagnosis of Errors in Basic Mathematical Skills. In *ITS*, 1982, pp. 157-183.

21. Burton, R.R. and Brown, J.S., A Tutoring and Student Modeling Paradigm for Gaming Environments. *SIGCSE Bulletin 8,* 1 (1976), 236-246.

22. Carbonetto, P., et al. Bayesian Feature Weighting for Unsupervised Learning with Application to Object Recognition. In *International Workshop on Artificial Intelligence and Statistics*, 2003, pp. 122-128.

23. Castillo, G., Gama, J., and Breda, A.M. Adaptive Bayes for a Student Modeling Prediction Task Based on Learning Styles. In *User Modeling*, 2003, pp. 328-332.

24. Chi, M.T.H., et al., Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science 13,* (1989), 145-182.

25. Cohen, J., *Statistical Power Analysis for the Behavioral Sciences*. 2 ed. Lawrence Earlbaum Associates, Hillsdale, 1988.

26. Conati, C. and VanLehn, K., Toward Computer-Based Support of Meta-Cognitive Skills: A Computational Framework to Coach Self-Explanation. *Artificial Intelligence in Education 11,* (2000), 398-415.

27. Conati, C. and VanLehn, K. Providing Adaptive Support to the Understanding of Instructional Material. In *International Conference on Intelligent User Interfaces*, 2001, pp. 41-47.

28. Conati, C. and VanLehn, K., Using Bayesian Networks to Manage Uncertainty in Student Modeling. *User Modeling and User-Adapted Interaction 12,* 4 (2002), 371-417.

29. Conati, C. and Merten, C., Gaze-Tracking for User Modeling in Intelligent Learning Environments: an Empirical Evaluation. *Knowledge Based Systems* Techniques and Advances in IUIs (To Appear).

30. Conati, C., et al. On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks. In *User Modeling*, 1997, pp. 231-242.

31. Conati, C., et al. Exploring Eye Tracking to Increase Bandwidth in User Modeling. In *UM*, 2005, pp. 357-366.

32. Dash, M. and Liu, H. Feature Selection for Clustering. In *PACKDDM*, 2000, pp. 110-121.

33. Dash, M., Liu, H., and Yao, J. Dimensionality Reduction for Unsupervised Data. In *IEEE International Conference on Tools with Artificial Intelligence*, 1997, pp. 532-539.

34. Dash, M., et al. Feature Selection for Clustering - A Filter Solution. In *IEEE International Conference on Data Mining*, 2002, pp. 115-122.

35. de Vicente, A. and Pain, H. Informing the Detection of the Students' Motivational State: An Empirical Study. In *ITS*, 2002, pp. 933-943.

36. du Boulay, B. and Luckin, R., Modeling Human Teaching Tactics and Strategies for Tutoring Systems. *Artificial Intelligence in Education 12,* (2001), 235-256.

37. Duda, R.O., Hart, P.E., and Stork, D.G., *Pattern Classification*. 2 ed. Wiley-Interscience, New York, 2001.

38. Faraway, J.J., *Practical Regression and ANOVA using R*, 2002.

39. Ferguson-Hessler, M. and Jong, T.d., Studying Physics Texts: Differences in Study Processes Between Good and Poor Performers. *Cognition and Instruction 7,* 1 (1990), 41-54.

40. Fisher, R.A., The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics 7,* 2 (1936), 179-188.

41. Friedman, J.H. and Meulman, J.J., Clustering Objects on Subsets of Attributes. *Journal of the Royal Statistical Society, Series B 66,* (2004), 815-849.

42. Gama, C. Megacognition in Interactive Learning Environments: The Reflection Assistant Model. In *Intelligent Tutoring Systems*, Springer, 2004, pp. 668-677.

43. Gorniak, P.J. and Poole, D. Building a Stochastic Dynamic Model of Application Use. In *UAI*, 2000, pp. 230-237.

44. Heiner, C., Baker, R.S., and Yacef, K., Data Distilled by Educational Data Miners - Notes from Discussion at Educational Data Mining Workshop at ITS 2006.

45. Hundhausen, C.D., Douglas, S.A., and Stasko, J.T., A Meta-Study of Algorithm Visualization Effectiveness. *Visual Languages and Computing 13,* 3 (2002), 259-290.

46. Hunt, E. and Madhyastha, T. Data Mining Patterns of Thought. In *AAAI Workshop on Ed. Data Mining*, 2005, pp. 31-38.

47. Jain, A.K., Murty, M.N., and Flynn, P.J., Data Clustering: A Review. *ACM Computing Surveys 31,* 3 (1999), 264-323.

48. Jain, A.K., Duin, R.P.W., and Mao, J., Statistical Pattern Recognition: A Review. *IEEE Trans. on Pattern Analysis and Machine Intelligence 22,* 1 (2000), 4-37.

49. Kearns, M. and Ron, D. Algorithmic Stability and Sanity-Check Bounds for Leave-One-Out Cross-Validation. In *Computational Learning Theory*, 1997, pp. 152-162.

50. Kushmerick, N. and Lau, T. Automated Email Activity Management: An Unsupervised Learning Approach. In *IUI*, 2005, pp. 67-74.

51. Lange, T., et al. Stability-Based Model Selection. In *NIPS*, 2003, pp. 617-624.

52. Law, M., Figueiredo, M., and Jain, A.K., Simultaneous Feature Selection and Clustering Using Mixture Models. *IEEE Trans. on Pattern Analysis and Machine Intelligence 26,* 9 (2004), 1154-1166.

53. Lieberman, H. Letizia: An Agent That Assists Web Browsing. In *International Joint Conference on Artificial Intelligence*, 1995, pp. 924-929.

54. Luckin, R. and du Boulay, B., Ecolab: The Development and Evaluation of a Vygotskian Design Framework. *Artificial Intelligence in Education 10,* (1999), 198-220.

55. Mayo, M. and Mitrovic, A., Optimising ITS Behavior with Bayesian Networks and Decision Theory. *Artificial Intelligence in Education 12,* (2001), 124-153.

56. Merten, C. and Conati, C. Eye-Tracking to Model and Adapt to User Meta-Cognition in Intelligent Learning Environments. In *IUI*, 2006, pp. 39-46.

57. Mislevy, R.J. and Gitomer, D.H. The Role of Probability-based Inference in an Intelligent Tutoring System. In *User-Mediated and User-Adapted Interaction*, 1996, pp. 253-282.

58. Murray, T. and Woolf, B.P. Results of Encoding Knowledge with Tutor Construction Tools. In *AAAI*, 1992, pp. 17-23.

59. Mutter, S., Classification Using Association Rules. Master's Thesis. Computer Science Department, Freidburg im Breisgau, Germany, 2004.

60. Naps, T.L., et al., Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bulletin 35,* 2 (2003), 131-152.

61. Olejnik, S. and Algina, J., Measures of Effect Size for Comparative Studies: Applications, Interpretations, and Limitations. *Contemporary Educational Psychology 25,* (2000), 241-286.

62. Paliouras, G., et al., Discovering User Communities on the Internet Using Unsupervised Machine Learning Techniques. *Interacting with Computers 14,* 6 (2002), 761-791.

63. Pazzani, M.J. and Billsus, D., Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning 27,* (1997), 313-331.

64. Perkowitz, M. and Etzioni, O., Towards Adaptive Web Sites: Conceptual Framework and Case Study. *AI 118,* 1-2 (2000), 245-275.

65. Piaget, J., *The Construction of Reality in the Child*. Basic Books, New York, 1954.

66. Poole, D., Mackworth, A., and Goebel, R., *Computational Intelligence: A Logical Approach*. Oxford University Press, New York, 1998.

67. Rasmussen, C.E. and Williams, C.K.I., *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

68. Robardet, C., Cremillieux, B., and Boulicaut, J. Characterization of Unsupervised Clustering with the Simplest Association Rules: Application for Child's Meningitis. In *International Intelligent Workshop on Data Analysis in Biomedicine and Pharmacology, Co-located with the European Conference on Artificial Intelligence*, 2002, pp. 61-66.

69. Schneiderman, B. Promoting Universal Usability with Multi-Layer Interface Design. In *ACM Conference on Universal Usability*, 2003, pp. 1-8.

70. Segal, R. and Kephart, J.O. MailCat: An Intelligent Assistant for Organizing E-Mail. In *International Conference on Autonomous Agents*, 1999, pp. 276-282.

71. Self, J., *Bypassing the Intractable Problem of Student Modeling*, in *Intelligent Tutoring Systems: At the Crossroad of Artificial Intelligence and Education*, C. Frasson and G. Gauthier, Editors. 1990, Ablex Publishing Corporation: Norwood, N. J. p. 107-123.

72. Shute, V.J., *A Comparison of Learning Environments: All That Glitters...* in *Computers as Cognitive Tools*, S.P. Lajoie and S.J. Derry, Editors. 1993, Lawrence Erlbaum Associates: Hillsdale, NJ.

73. Sison, R. and Shimura, M., Student Modeling and Machine Learning. *AIED 9,* (1998), 128-158.

74. Sison, R., Numao, M., and Shimura, M., Multistrategy Discovery and Detection of Novice Programmer Errors. *ML 38,* (2000), 157-180.

75. Stern, L., Markham, S., and Hanewald, R. You Can Lead a Horse to Water: How Students Really Use Pedagogical Software. In *SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2005, pp. 246-250.

76. Talavera, L. and Gaudioso, E. Mining Student Data to Characterize Similar Behavior Groups in Unstructured Collaboration Spaces. In *Workshop on AI in CSCL, European Conf. on AI*, 2004, pp. 17-23.

77. VanLehn, K., *Student Modeling*, in *Foundations of Intelligent Tutoring Systems*, M.C. Polson and J.J. Richardson, Editors. 1988, Erlbaum: Hillsdale, NJ. p. 55-78.

78. Verleysen, M. and Francois, D. The Curse of Dimensionality in Data Mining and Time Series Prediction. In *International Workshop on Artificial Neural Networks*, 2005, pp. 758-770.

79. Vicente, K.J. and Torenvliet, G.L., The Earth is Spherical (p<0.05): Alternative Methods of Statistical Inference. *Theoretical Issues in Ergonomics Science 1,* 3 (2000), 248-271.

80. von Glasersfeld, E. Learning as Constructive Activity. In *North American Group of Psychology in Mathematics Education*, 1983, pp. 41-101.

81. Webb, G.I., Pazzani, M.J., and Billsus, D. Machine Learning for User Modeling. In *User Modeling and User-Adapted Interaction*, 2001, pp. 19-29.

82. Weiss, G.M. and Provost, F., The Effect of Class Distribution on Classifier Learning: An Empirical Study. Technical Report ML-TR-44. Department of Computer Science, Rutgers University, 2001.

83. Zhu, X., Ghahramani, Z., and Lafferty, J. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *International Conference on Machine Learning*, 2003, pp. 912-919.

84. Zukerman, I. and Albrecht, D.W. Predictive Statistical Models for User Modeling. In *User Modeling and User-Adapted Interaction*, 2001, pp. 5-18.