What Do Students Know?

An Outcomes-Based Assessment System

Titus Winters
UC Riverside
Computer Science & Engineering
Riverside, CA, 92521
titus@cs.ucr.edu

Tom Payne
UC Riverside
Computer Science & Engineering
Riverside, CA, 92521
thp@cs.ucr.edu

ABSTRACT

Well-run organizations collect, archive and analyze data relating to the effectiveness of their important processes. Educational institutions discard a wealth of student scores that could be analyzed. Each score contains important information about the student as well as the item (i.e., problem or question). This paper describes our project to develop an outcomes-based assessment system that mines per-item scores to track each student's skills and knowledge. Statistical inference techniques from both educational statistics and data mining will quantitatively determine each student's acquired competency, with minimal input from faculty. The culmination of item-level assessment gives individual faculty feedback on their courses, and gives curriculum committees feedback on which objectives are sufficiently met by their respective curricula.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.2.m [**Computing Methodologies**]: Artificial Intelligence—*miscellaneous*

General Terms

Management

1. INTRODUCTION

One major force behind the push for assessment, course management, introspection, and improvement is the accreditation organization for technology-related curricula in the United States, the Accreditation Board for Engineering and Technology (ABET). ABET's current accreditation requirements, commonly known as "EC 2000"[2], mandate the use of continuous-improvement techniques, similar to "total quality management" and ISO 9000 [1] techniques from the manufacturing and industrial communities, now applied to monitoring and increasing educational effectiveness. The targeted outcomes (student competencies in terms of knowledge and/or skills) of every subsystem (course) and of the overall system (degree program) must be measured, ideally in some direct and quantitative fashion, and corrective actions must be taken to remedy discrepancies between the targeted level of each outcome and the actual levels measured.

This is both a blessing and a curse. Many faculty members found occasional accreditation preparation and visitation to be onerous when undertaken once or twice every decade. Under EC 2000, accreditation is no longer an occasional accounting exercise but a continuous process intended to boost the effectiveness of instruction every year, not just during a visit. Faculty adoption of the process perspective is critical for success under the new requirements.

Most educators have an on-going improvement effort. But EC 2000 is an impetus to move beyond the realm of "folk pedagogy," where gut-feelings and intuition dictate which experimental attempts are considered successful and which are not. Under EC 2000, degree programs must actively and systematically monitor the instructional results.

Can proper quantitative assessment be prescribed and performed in a manner unobtrusive enough to gain wide faculty acceptance? In this paper we will introduce a system that we hope does exactly that. The system described here forms the basis for our own accreditation process.

Throughout the development of this project, two principles have governed our decisions at every step of the way: simplicity and usability. The simplicity of the system can be demonstrated by the fact that it has been assembled primarily by one person over the course of two years. The usability of the system can be seen by the **voluntary** adoption of various components by faculty members, and by the steady increase in the number of users. The project goal is to develop a system that makes it easier for faculty to perform instructional tasks and, as a side-effect, gives them the benefits of comprehensive data analysis and assessment. Because of the difficulty in changing basic usage paradigms, our tools have to *extend* existing methods, like "red-pen grading" or "fill-in-the-bubble" multiple-choice quizzes. Further, since every instructor's requirements are different, the software components should be individually adoptable, with only minimal invariant requirements.

ICER'05, October 1–2, 2005, Seattle, Washington, USA.

2. PROCESS DATA PATH

The ultimate goal of this project is to provide a data-backed, quantitative measurement of how well students are meeting the goals of the curriculum. These goals range from individual course goals, like learning linked-lists in the Data Structures course, to knowledge and skill goals for our students as of the day they graduate. The task is to develop a process, both in terms of deployment and calculation, that will allow us to generate these assessments while placing minimal burden on the instructors.

It is impossible to determine a student's level of ability in any particular topic on the basis of per-course letter grades [12] alone. Those grades represent the aggregation of too many factors, causing the student's ability in any particular topic area to be lost in that aggregation. A finer granularity of information is required.

Quantifiable assessment of student knowledge has been possibile using techniques from educational statistics dating back to the 19th century with Classical Test Theory and more recently with Item-Response Theory (IRT), the workhorse behind the computer adaptive testing scheme used in the GRE for the past decade [3]. Both of these techniques have benefits and drawbacks, but both require a relatively large amount of instructor time to feed the system the necessary inputs. In most instructional settings, it is a burdensome chore to record item-level data and to decide which items related to which student competencies. We hope to use advances in automation and data mining to reduce the human oversight needed to extract identical outputs: how much of each competence has each student achieved, and how difficult and/or discriminating is each item of assessment (test question or homework problem) in the curriculum? Much of this is possible using computerized testing, but we do not have sufficient resources to utilize this, and many instructors would prefer to hold on to more traditional assessment methods. In order to *extend* existing methods of instruction, a more complicated assessment system is necessary.

Inspired by and heavily patterned after published outcomes measurement assessment techniques such as those in [12], our overarching process is a multi-stage reduction from concrete numeric data (student scores) through successive layers of abstraction to our final numeric goal. Here we will briefly outline these stages and the terminology behind this process before examining each stage in depth.

Most importantly, given that this is a score-driven process, it is vital to capture and record the scores. In this paper we will represent the scores for a particular course as the matrix S, whose rows are indexed by student and whose columns are indexed by item. In order to facilitate the gathering of student scores at the item level, we have developed a number of tools, described in Section 3. To the extent possible, these tools have been developed to allow behaviors similar to those used when grading by hand, so as to reduce the barriers to adoption.

To assess whether a course offering has successfully met its goals, the instructors for each course have created formal lists of five to ten *course objectives*. While there is likely to be some inter-relation among these objectives, we treat them as independent. It is important to note that identifying objectives is a one-time task: the course objectives do not change from term to term, although different instructors may place heavier emphasis on one or another when they offer the the course. In order to determine the level of coverage and successful knowledge testing on these objectives, we need an $m \times n$ *relevancy matrix* R, which gives the relevance of each of the m questions to each of the n course objectives. Since courses generally have forty to one hundred questions, filling out the entire $m \times n$ matrix would be tedious and frustrating business. However, the student score data itself contains hidden structure that can be extracted using data mining techniques to fill in much of this matrix automatically. These algorithms are discussed in Section 4.2. In order to ensure that the structure that is determined by the course objectives, the instructors are asked to submit partial relevancy information. We have found that identifying $10 - 20\%$ of the entries in R is sufficient to complete R with reasonably good accuracy.

R and the student score information S is sufficient to evaluate a single course. For each student we can now determine a level of coverage and knowledge with respect to each topic. Averaging these across the student population of the course gives a meaningful measurement of the extent to which topics were covered sufficiently for students to understand the material. This is valuable information for the instructor, and can be created easily and automatically given properly formatted gradebooks and relevancy information. If a particular objective is not receiving sufficient coverage, or has been covered but students are consistently unable to answer questions relevant to that objective correctly, the score for that objective will demonstrate this clearly. Similarly, if one objective is covered more than all others, and students perform well on it, scores on that objective will be correspondingly high. By evaluating the scores on those objectives against the relative (subjective) importance of the various objectives, instructors get numeric, verifiable feedback on the workings of the course.

However, there does exist one hidden element of complexity preventing simple matrix multiplication of the score matrix S by R: S will have many missing entries due to students not finishing tests, being absent, not submitting assignments, etc. Assigning a zero value to these is inaccurate: if the students attempted the item, some fraction of them would have answered correctly. Filling with zeros simply drives coverage scores downward and muddles the ability to extract entries of R correctly. In this case, it becomes useful to have a predictive model by which we can predict whether or not a student would have answered a question correctly given their general course performance and some properties of the question based on the responses of the students that did attempt it. A simple solution for this, based on techniques from educational statistics and psychometrics, can predict these values with an accuracy of a 65-to-70 percent. This model also provides feedback to instructors on the quality and difficulty of the questions, providing an extra level of feedback for the system. This technique is discussed in Section 4.1.

Abstracting an additional level beyond course evaluation, we have identified a number of *program objectives*:[1] those general skills and competences that students ought to have on the day they graduate from the program. These are very general proficiencies, such as "Ability to apply mathematics, science, and engineering principles." We can track a

[1]Known in ABET terminology as *program educational outcomes*, a confusing use of the term "outcome" in which it means "goal" rather than "result."

running estimate of how well these are attained by further abstracting from the scores we calculated for each course's objectives. The course objectives for each course are therefore related to the program objectives through a *course matrix*. Given five to ten course objectives and ten to fifteen program outcomes,[2] this will generally have on the order of one hundred entries. Filling these in requires, on average, about ten to twenty minutes, and only needs to be done once, as the course and program objectives do not change, or change very infrequently.[3] Multiplying course objectives scores by the course matrix for that course, and summing across courses gives a final score for each program objective. Just as an instructor can examine the course scores to determine whether or not each objective is being covered sufficiently, the curriculum committee can examine the program scores to see whether or not there is sufficient coverage for each of the program objectives.

In summary, the inputs to this system are

- the score matrix for each course, which we have built tools to facilitate the recording of,
- the relevancy matrix relating each question in a course with the objectives for that course. This matrix can be provided in only partial form and filled in by data-mining algorithms
- The course matrices for each course, which need to be identified only one time, ever.

Given these inputs and the necessary algorithms to run the system, we can then calculate numeric scores for each course objective and program outcome. While these outputs are dimension-less quantities, they do allow us to perform the necessary task: continuous improvement functions just as well when comparing two quantities or examining derivatives. No absolute scale is necessary for evaluating these numeric outputs, it is sufficient to compare scores between objectives and outcomes ("Why didn't we cover linked lists as well as binary trees?") and corresponding scores between quarters ("Did this quarter's explanation of hash tables go better than last quarter's?")

3. DATA GATHERING

Two major tools have been developed for the gathering of per-item score data, Agar and Marksense. Between these tools, it is generally faster and easier to grade an instrument and get item-level data output and stored in the correct format than it would be to grade by hand. This section will briefly detail each of these tools and their usage.

3.1 Agar

The most ambitious and complex tool is Agar, a robust and generalized framework for computer-aided assessment (CAA). Agar was designed not simply as an "automated grader" like many CAA systems [13, 10, 9, 5], but rather as an *assistant* for human graders. The purpose of Agar is not to replace human grading, which is indispensible for providing the feedback necessary for students to learn from their mistakes. Instead, Agar has been designed to reduce the amount of redundant and automatable work involved in grading. It has proven itself useful not only in grading programming assignments, which can be scripted, but also in grading subjective work such as written homework problems, short-answer test questions, and the style aspects of programming assignments.

Many components work together to make Agar useful, but the most helpful feature is the write-once comment system. While grading a set of submissions, it is common to repeatedly encounter the same small set of student errors. After the first few instances of a frequent error, it is not uncommon for a grader to become frustrated with the students and to grade more harshly for that mistake. When students compare scores and find different penalties for the same mistake, they feel abused, which hurts course morale. Agar's comment system prevents these problems. The first time a grader encounters a particular error, he/she creates a new comment. A comment consists of a score adjustment (bonus or penalty), which item it applies to, a short name to distinguish it from other comments, and some text feedback for the student. Once created, the new comment is added to a list of comments (rubric) that has been generated for that item. It is then a simple matter of drag-and-drop to assign the same penalty and feedback to other submissions deserving the same comment. Not only is this system more fair, it is faster and more convenient for the grader, which enhances the chances of adoption. In conjunction with the user-created, no-programming-necessary automated-testing tools for grading programming assignmnts, Agar saves time and increases the consistency of grading across the spectrum of assignments.

Agar supports the annotation of student work, much like traditional "red-pen" grading, via its PDF annotation facility. Given digital submissions of the student work,[4] Agar allows for comments to be placed on the page and displayed in the Agar interface as a standard red 'X'. In this way, comments can be given a spacial location on the page, just as with manual grading methods. When the grading process is completed, Agar generates annotated PDF files for each student, which contain colored sticky-notes viewable in any compliant PDF reader. Double clicking these notes brings up the comment's text and penalty. Agar also facilitates the automatic mailing of these documents to the students that submitted them, meaning that no more class time needs to be spent on handing back papers. Finally, since digital copies are kept by the grader, there are no longer instances of students bringing altered versions of their submission for regrading.

A non-obvious feature of Agar that is of great importance for matching existing grading paradigms is the ability to merge Agar workspaces. Especially in large classes, grading tasks are often divided among many graders. This division may be accomplished either by having a particular grader grade a particular question for the entire class, or by having each grader grade a subset of the students. In order to facilitate the administration of such group-grading projects, Agar allows for the setup and distribution of an initial grading template, complete with comments, and then a final re-merging of grades into a single workspace composed of all of the individual assessment performed by many graders. This has rapidly become one of the most important features of

[2]The canonical ABET "A through K" are certainly the most common of these.

[3]Although we give instructors the opportunity to update their course matrix each term.

[4]Including scans of written work submitted on paper.

the system. It is continually being used in an ever-increasing number of unforeseen ways.[5]

3.2 MarkSense

To simplify the grading of multiple-choice quizzes and exams, we have developed in house a simple computer-vision system to do optical mark recognition (OMR), which we have dubbed MarkSense. MarkSense so far has been our most successful tool in terms of instructor adoption: with only minimal advertisement, about half of our undergraduate courses utilize it after a year of combined development and deployment time. It has been successfully used on exams that involved over 40,000 grading decisions, and generally requires less than ten minutes of human involvement to grade a typical quiz of 20 questions for a class of 100 students. Like Agar, MarkSense outputs per-item scores directly to a special-format spreadsheet, called a *gradebook*. By simply examining each column of the spreadsheet, an instructor can detect possible errors in their answer keys, identify misleading questions, and more. Additionally, since the pipeline of tools that MarkSense is composed of output OMR results to a text file, re-grading is completely separate from the computationally expensive step of performing the OMR in the first place — thus, it is a trivial matter to fix the effects of an incorrect answer key. MarkSense is by no means a breakthrough tool, but it demonstrates how even a basic application of automation can yield tools that have significant value to educators.

MarkSense integrates perfectly with Agar for tests that are part multiple-choice and part subjective. We have also generalized the algorithms in the MarkSense software to work for our course evaluation forms. Additionally, MarkSense has been used to develop an OMR system for instructors who still prefer to grade by hand, allowing them to bubble-in the scores for students on each question as they grade and then have the grade sheet automatically generated from that stack of forms. While relatively simple in scope, MarkSense has been one of the major components driving adoption of CAA technology within our department.

4. DATA ANALYSIS

4.1 Item Evaluation

Given per-item scores, it is also possible to evaluate the questions themselves. Using this technique, it is possible to identify question types or individual questions that are particularly good or bad *given the instructor's teaching and grading style*. The concepts behind this method are a simplification of a well-known statistical method from psychometrics[6] known as Item-Response Theory (IRT).

IRT is a test-analysis technique developed and refined by a number of psychometric researchers, with much credit going to Frederic Lord [8], a research scientist with Educational Testing Service (ETS), the organization that develops and administers the SAT and GRE exams. The fundamental idea behind IRT is that each item of assessment corresponds to a single competency. Each student is presumed to have some numerically quantifiable amount of that competency (a.k.a., latent ability) to be measured via IRT. Each item has a "characteristic curve", which is a plot of the probability that a given student will get that item correct as a function of the amount θ of the corresponding competence that the student possesses. The process of assessing or measuring student competencies is therefore the process of administering these items, and then using student performance to derive the parameters governing these characteristic curves. Using regression techniques, one can then estimate each student's θ, i.e., where each student falls on the θ axis. The most common model for an item's characteristic curve is a sigmoid curve or logistic function, $(1 + e^{\alpha(\theta-\beta)})^{-1}$, which has two parameters:

- a difficulty, β, which is the value of θ at which the probability is 50% and
- a discrimination factor, α, which corresponds to how sharply the probability transitions from almost zero to almost one.

An example plot is shown in Figure 1.

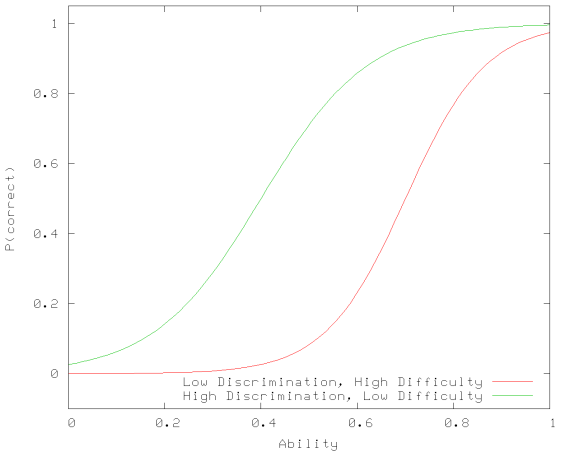

Figure 1: Example Characteristic Curves

IRT is powerful but has some limitations. Chief among them is the need for larger student populations than most courses have, which makes the actual mechanics of IRT inappropriate for our purposes. Also, within IRT there is no good way of expressing that a single item may involve more than one competency, or that the required competencies are not needed in equal amounts. A simple thought experiment can demonstrate that most if not all questions are testing multiple competencies, although in general one is tested more explicitly. For example, questions written in English require that the student have a minimum amount of the "Reads-English" competency to even attempt the item as intended. While this is an extreme example, multiple competency questions are quite common, especially in a discipline like computer science where the majority of knowledge is cumulative and interconnected. While IRT itself is not directly usable for this task, the concepts of question difficulty and discrimination are very valuable.

Given the matrix of student scores for every question and the vector of their overall course scores, we can easily evaluate which questions are actually correlated with high ability

[5]It is perhaps the hallmark of successful software development that users can develop usage patterns and expertise that the developers never expected or planned for.

[6]The area of psychology and educational statistics that governs measurement, data analysis, and test valididty

in the course, and estimate the difficulty of each question. Instructors can then begin to learn which types of questions are most meaningful (have a high α). This refinement of questions is based entirely on the course score vector, and thus the grading and course management policies of the individual instructor. Questions that are perfect for one instructor may not be perfect for another, although in general they are likely to be correlated.

Knowing the β for the questions can provide insight into what the students are really understanding. If, after evaluating a question, the β seems too high, it indicates that the topic is not well understood by the students. Generally one can assume that a question with a high α and a β in the useful range (the range of course scores that would pass the course) is a "good question."

4.1.1 Calculating α and β

To see how calculating the difficulty and discrimination parameters can be accomplished easily, first assume β is known for each question. It is then possible to find the empirical discrimination for a question by evaluating how well that β separates students that got the question right from those that got it wrong.

There are a number of ways of evaluating this, and in general they give similar results in most cases. We have evaluated complex techniques like entropy-based measures, but in general the simplest technique works well, with near-identical results. That technique is to calculate the percentage of scores that would be guessed correctly under the assumption that every student with ability under β got the question wrong and every other student got it right:

$$\alpha = (\sum_{i:S_i<\beta} \delta(M_{i,j}) + \sum_{i:S_i\geq\beta} \delta(M_{i,j}-1))/n$$

where i ranges over all students, j ranges over all items, M is the matrix of per-item scores, and S is the vector of overall course scores. δ is the Kronecker delta function, which returns 1 if its argument is 0 and 0 otherwise.

If it really is the case that β is a perfect split point, then α will be 1, a perfect score. If correct and incorrect scores are evenly distributed on both sides of β then α will be .5, and if somehow the question were completely backward (only students with ability less than β answered correctly) this results in a score of 0. Nearly any similarity measure developed for data-mining / machine-learning "decision tree" algorithms can be adapted to work here. Additional techniques can be found in [11].

Given this ability to produce α given the question scores, ability scores (course scores), and β, it is now easy to find the actual β. The best estimate of β is the value of β that maximizes α. Since α depends only on which scores were guessed correctly, it is sufficient to only loop through each distinct course score and evaluate on those split points, avoiding any gradient optimization methods. The procedure to produce α and β for each question can be implemented in about 100 lines of code and is available from the first author's website.

4.1.2 Assumptions

In order for the above to be usable, several assumptions must be made. First of all, we are assuming that the question has some relevance to the general subject matter of the course so far. Secondly, nobody can track the true ability level of the students; the use of course scores is only an estimate. If those scores are highly uncertain (e.g., at the beginning of the course) or have nothing to do with the question, then this is not a valid assumption. This presumes that each course involves a single proficiency, which is generally untrue. But, for many courses where this is a particularly bad assumption (for example, a course composed of two half-courses on different topics), it is possible to use the scores from only the appropriate portion of the course.

The estimates of both parameters rely heavily on the distribution of the course scores. Student grades fluctuate significantly at the beginning of the course before the law of large numbers begins to stabilize each student's grade toward its final value. Therefore, the values of α and β are going to be most accurate at the end of the course, and a final evaluation of which questions are worth keeping for possible reuse is best done after the course has completed. It is useful to evaluate questions immediately after grading the instrument, especially to find topics that were tested but not fully understood, but such evaluations should be recognized as less-accurate estimates.

Independence of scores is also a concern. If the estimate of ability (course score) includes the score on the given question, then by definition there is some correlation between the two. To get the best estimates of α and β, it is best to provide ability estimates that do not include the score on that question or instrument. In practice, if each individual question has a very small effect on the total grade, then this effect is negligible and no questions need to be evaluated separately.

4.1.3 Cross-Term Consistency

This technique allows databanks of questions to be assembled along with an estimate of the difficulty associated with those questions. There are a few caveats here: the difficulties are likely very dependent on instructor, are obviously dependent on course, and are hopefully dependent on what time during the course the question was asked. For example, if one instructor uses a question on a quiz early in the term in one term and on the final exam in the next term, it is hopefully the case that the students will have solidified the skills necessary to answer that question, so the empirical difficulty will decrease. (This is a deviation from IRT, where question parameters are absolute but student abilities are generally increasing throughout the term.)

To demonstrate the validity of this statement, we have identified 20 questions that were repeated between Fall 2004 and Winter 2005. Of these, eight were dramatically too easy, with difficulty levels in the D- range or lower (in this range, the density of student scores is too low for predictions to be very accurate, without enormous class sizes). After ignoring anything with a difficulty less than 65%, we are left with 12 questions ranging from difficulties of 66% to 95%. On average the difference between difficulties between terms is 5%, with a standard deviation of 4%, meaning that more than two thirds of the time, a repeated question will have a difficulty within one letter grade when used under similar circumstances.

4.1.4 Item Assessment Results

Table 2 are empirical difficulty and discrimination values for several particularly good or bad questions from various courses.

Question	Difficulty	Discrimination
How many values can be represented by a 4 byte binary word?	95%	.81
What is the result of NOT(1000 AND (1100 OR 0101))?	67%	.81
The Unix filesystem is a true tree/hierarchy. There are no loops or cycles. (T/F)	73%	.83
Using *pthreads* on a dual processor system can result in a single process utilizing both processors fully. (T/F)	77%	.82
Write a simplified *wc(1)*: a program that reads from standard input until it reaches EOF and prints to standard output the number of newline characters it reads.	67%	.83
Virtual memory and memory mapping allow multiple processes to share memory under the illusion of being the only process in memory. (T/F)	93%	.60
Thrashing may not occur on a system using a two-level scheduler. (T/F)	100%	.58

Table 1: Sample Difficulty and Discrimination Scores

Method	Avg. Squared Error
Diff+Disc	0.309%
Wrong	0.632%
Right	0.321%
Student Avg	0.416%

Table 2: Comparison of data-filling methods

4.1.5 Estimate Missing Data.

As mentioned in Section 2, it is advantageous if the student score matrix S for each course has no missing entries. In reality, this is a very rare event: every student answering every item during the entire course offering. Developing a simple predictive model to fill in the missing entries is an important step. Given the already-discovered α, β parameterizations of each question, we can predict individual item scores with an accuracy of $65 - 70\%$ on binary items. For each missing data value, set it to 0 if the course grade for that student is less than β and 1 otherwise. As seen in Table 1, this technique outperforms all others we have seen mentioned in the literature on dealing with missing entries in student data, in some cases by as much as a factor of 2.

4.2 Objective Identification

The final algorithmic hurdle to overcome on the data path from item-score data to program objective scores is to generate R. Manually identifying topic information for each question is tedious, and may not match the statistical model supported by the data itself. An ability to automatically extract topic information from student scores would avoid such tedium and additionally allow instructors to:

- determine which topic a student is struggling most with,
- give targeted study suggestions,
- build a question-bank of questions with known difficulty levels and known relevancy to each topic in the course, or
- automatically generate a test or quiz covering specific topics with a known expected average score.

The broad class of algorithms most suitable for topic extraction are called "collaborative filtering algorithms."

4.2.1 Collaborative Filtering

Collaborative filtering was initially developed as a method for filtering arbitrary information based on user preferences [4], for use in systems such as Amazon.com's recommendation engine. At a high level, a pool of users assign ratings to a various products (books, movies, CDs) and the system infers the underlying structure: what factors govern whether a user will like a product or not, how much does each factor affect a given user, and how much of each factor is present in each product. User preferences may drift over time, but most filtering algorithms ignore this temporal aspect.

The overall concept of collaborative filtering can be viewed in a number of ways. Collaborative filtering can be viewed as a system for predicting missing ratings based on a user's similarity to other users that rated the missing item. It can also be viewed as identifying the latent factors that influence each rating and estimating the influence of those factors on each user and each product.

This concept generalizes very easily into the educational domain, although the input matrices are generally denser and noisier in educational datasets. Each course covers a number of "topics," which are the latent factors in this context. Each item in a course pertains to one or more of these topics, and each student has some ability in each topic. Students may gain or lose ability in each topic over the course of the term, just as user preferences may drift. Here we assume that drift is a negligible effect since testing on a topic occurs *after* that topic has been covered in the course.

We have evaluated more than ten algorithms that could perform some form of this type of filtering [14], and in the end the most suitable is also in many ways one of the easiest: Non-Negative Matrix Factorization.

4.2.2 Non-Negative Matrix Factorization

Non-Negative Matrix Factorization[7] (NMF) approximates the matrix of interest M by the product of two non-negative

matrices $UH \approx M$ where M is $m \times n$ and U is $m \times f$ and H is $f \times n$. The parameter f is the number of factors assumed to be present in the data. In the idealized educational view of the algorithm, the factors are topics, i.e., $U_{i,j}$ is the ability of student i in topic j, and $H_{j,k}$ is the relevance of topic j to question k. In practice, the factors that are identified may correspond to non-topical factors ranging from whether the student was having a good day to how good the student is at reading trick questions.[7]

The non-negativity of NMF gives the output of the algorithm a more intuitive interpretation than does principle-component analysis (PCA) or singular-value decomposition (SVD), two commonly used algorithms that perform a similar function. In these more common algorithms, the matrix is broken down into possibly-negative values. With NMF, any non-zero entry is an additive weight for the final output. The assumption that a student's score on an item is the inner product of their ability on f topics and the relevancy of the item to those f topics is certainly a gross simplification, but it does capture some of the structure of the process.

A straightforward technique presented in [7] is a gradient descent approach to discovering the U and H whose product best approximates M. At each iteration, every entry in U and H is updated multiplicatively in a way guaranteed to reduce the component-wise Euclidean distance between M and UH. On all gradebook datasets we have evaluated, this algorithm converges to within 1% of M in under 300 iterations, each of which can be calculated rapidly. Additionally, it is generally very insensitive to initial conditions. Of the 1000 NMF runs we have performed to date, only once has there been a noticeable effect of the algorithm being trapped in a local minima. The simplest algorithm presented gives multiplicative update rules that perform a gradient descent minimization of the Frobenius norm of $M - UH$ (that is, the element-wise Euclidean distance between M and UH):

$$U_{i,a} \leftarrow U_{i,a} \frac{[MH^T]_{i,a}}{[UHH^T]_{i,a}}$$

$$H_{a,u} \leftarrow H_{a,u} \frac{[U^T M]_{a,u}}{[U^T UH]_{a,u}}$$

In practice, squared error is not necessarily the best metric to minimize because it penalizes just as heavily for estimating a score to be 30% instead of 10% as it does for 80% instead of 100%. An alternative to optimizing the Euclidean distance metric is presented in [6] where the objective function is

$$F = \sum_{i=1}^{m} \sum_{u=1}^{n} [M_{iu} \log(UH_{iu}) - UH_{iu}]$$

This objective function is derived by interpreting the NMF process as a method for constructing a probabilistic model where value $M_{i,j}$ is generated by adding Poisson noise to $UH_{i,j}$ and then finding the maximal likelihood U and H for generating the known M.

The benefit of using this objective function is that its corresponding update rules

$$U_{i,a} \leftarrow U_{i,a} \sum_{u} \frac{M_{i,u}}{[UH]_{i,u}} H_{a,u}$$

$$H_{a,u} \leftarrow H_{a,u} \sum_{i} \frac{M_{i,u}}{[UH]_{i,u}} U_{i,a}$$

no longer contain matrix multiplication operations on M, thus allowing the missing values to be dealt with only in the multiplication of S by R, rather than repeatedly through the process.

In the summations, simply skipping those entries that are undefined because of missing values in M is equivalent to assigning 0 to the missing entries. An option that works better is to normalize each summation by the number of missing entries in that summation (thus, multiply by 1.5 if only 2 of 3 values was present.) This is actually equivalent to pre-processing two versions of M: one where missing values are initialized to the average value for the row (a student is predicted to do as well on missing items as they did on average for existing items) and one similarly for columns (a student is predicted to do as well on a missing item as the average of those classmates that attempted the item.)

4.2.3 Known Relevance

One benefit that NMF has over other algorithms that we could have chosen is that it is trivial to enhance it with known entries from R. In order to encode a known value, it is sufficient to set that value at the beginning of every gradient descent update. The question is, what percentage of the values in R must be supplied by the faculty in order for meaningful results to be extracted? It may be the case that the factors that are mathematically most relevant are not topic at all, but are instead question type (short answer vs. multiple choice), whether the student was having a good day, or whether the question is a "trick" question.

4.2.4 Topic Identification Results

To determine the accuracy of NMF in reconstructing R, we measure the average value of the Frobenius norm of a fully-provided R subtracted off from the R recovered by NMF. A plot of this as a function of how much of R is provided is shown in Figure 4.2.4.

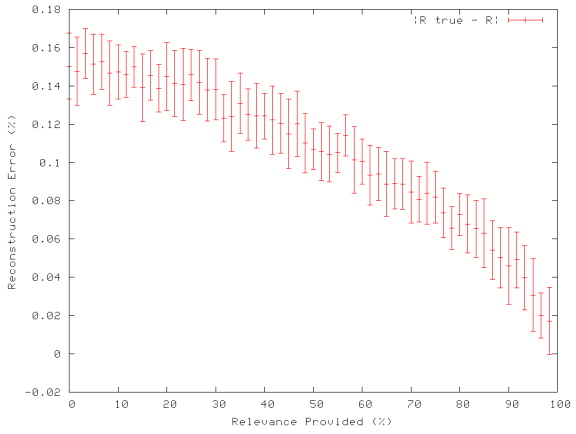

Figure 2: Error vs. Data present

While the error decreases continuously as a function of how much data is provided, anything below the 15% error range can be viewed as acceptable error, misclassifying only one question in 7 on average. In our informal experiments

[7]We have seen and identified both of these situations in practice.

with having the same instructor generate a new R for the same data, the error rate remained at least that high. For this plot, taken from a course with four course objectives and only 15 items, on average each question would need to be linked with one objective to provide this level of accuracy. It is important to note that although some of the identifications made here should be positive (this question relates to this objective), it is just as useful to identify which objectives an item does not relate to. In practice, this makes the task much faster.

5. CONCLUSIONS

We have identified a method for performing quantitative outcome assessment based primarily on data generated during every course: student score data. The tools and algorithms presented here, the domain knowledge of the instructors, and a moderate amount of computational time are sufficient to generate a detailed quantitative evaluation of how much graded exposure the students were given to each topic, and how successful they were on those items. Additionally, as a side-effect of performing this analysis, items that were particularly good or particularly bad can be identified for the instructor, giving them feedback that will hopefully train them to ask better questions more consistently.

It is important to note that none of this system is theoretical. As of this year, this system has been deployed, and every step presented here has been performed in our department's improvement process. Furthermore, our department has seen ever-increasing adoption and buy-in from the faculty.

Given this infrastructure, we are now capable for the first time of assigning a numeric impact on experimental curricula changes or changes in teaching method for a given course. The effects of this are just beginning to be seen in our department as our continuous-improvement process takes firmer hold and begins to be explored more fully by the faculty. We expect that in-addition to assessing the effectiveness of our curriculum, our process and the data it gathers will allow for significant future research into how students understand computer science.

6. ACKNOWLEDGMENTS

The authors would like to thank Dr. Christian Shelton for his continued advice and input on the data mining aspects of this project, and the lecturers of our department for their support during the development of our data gathering tools.

7. REFERENCES

[1] Quality management principles. http://www.iso.org/iso/en/iso9000-14000/iso9000/qmp.html.

[2] Accreditation policy and procedure manual. http://www.abet.org/policies.html, November 2002.

[3] W. J. V. der Linden and C. A. Glas, editors. *Computerized Adaptive Testing: Theory and Practice*. Kluwer Academic Publishers, July 2000.

[4] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[5] C. Higgins, P. Symeonidis, and A. Tsintsifas. The marking system for coursemaster. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 46–50. ACM Press, 2002.

[6] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[7] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.

[8] F. Lord. *Applications of Item Response to Theory to Practical Testing Problems*. Lawrence Erlbaum Associates Inc, 1980.

[9] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 55–59. ACM Press, 2002.

[10] A. Pardo. A multi-agent platform for automatic assignment management. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 60–64. ACM Press, 2002.

[11] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 11(1):81–106, 1986.

[12] G. Rogers. Do grades make the grade for program assessment. *ABET Quarterly News Source*, (Fall/Winter):8–9, 2003.

[13] urs von Matt. Kassandra: the automatic grading system. *SIGCUE Outlook*, 22(1):26–40, 1994.

[14] T. Winters, C. R. Shelton, T. Payne, and G. Mei. Topic extraction from item-level grades. In *AAAI-05 Workshop on Educational Data Mining*, 2005.