Predicting Student Performance in an ITS using Task-driven Features

Ritu Chaturvedi
Mathematical and Computational Science
University of Toronto, Toronto, Canada
Email: ritu.chaturvedi@utoronto.ca

C. I. Ezeife
School of Computer Science
University of Windsor, Windsor, Canada
Email: cezeife@uwindsor.ca

***Abstract*—Intelligent Tutoring Systems (ITS) are typically designed to offer one-on-one tutoring on a subject to students in an adaptive way so that students can learn the subject at their own pace. The ability to predict student performance enables an ITS to make informed decisions towards meeting the individual needs of students. It is also useful for ITS designers to validate if students are actually able to succeed in learning the subject. Predicting student performance is a function of two complex and dynamic factors: (f1) student learning behavior and (f2) their current knowledge in the subject. Learning behavior is captured from student interaction with the ITS (e.g. time spent on an assigned task) and is stored in the form of web logs. Student knowledge in the subject is represented by the marks they score in assigned tasks and is stored in a specific component of the ITS called student model. In order to build an accurate prediction model, this raw data from student model and web logs must be engineered carefully and transformed into meaningful features. Existing systems such as LON-CAPA predict students performance using their learning behavior alone, without considering their (current) knowledge on the subject. Lack of proper feature engineering is evident from the low values of accuracy of their prediction models. This research proposes a highly accurate model that predicts student success in assigned tasks with a 96% accuracy by using features that are better informed not only about students in terms of the two factors f1 and f2 mentioned above, but also on the assigned task itself (e.g. task's difficulty level). In order to accomplish this, an Example Recommendation System (ERS) is designed with a fine-grained student model (to represent student data) and a fine-grained domain model (to represent domain resources such as tasks).**

I. Introduction

An Intelligent tutoring systems (ITS) is a computer system that tutors students in some domain (e.g. C Programming), without the physical presence of a teacher. Functions of an ITS include adaptation or customization and intelligence. Customization is in terms of presentation of learning materials (e.g. students with current grades greater than 80% are assigned task T4, which is of high difficulty level, as opposed to students with grades less than 50% are assigned task T1, which is of low difficulty level). Intelligence shown by the tutoring system is based on the objectives for which the ITS is designed (e.g. providing support to students through hints or examples in real time). Every ITS system has a domain and a student model (unique for every student using the ITS), in addition to other components (Chaturvedi & Ezeife, 2017). Domain model defines the expertise required on the subject (e.g. correct solutions of every example and task used in the ITS), whereas a student model stores student information (e.g. his / her marks in every task in the ITS's domain).

Example-based ITS (Gog & Rummer, 2010; Renkl, 2014) systems aim to assist students to succeed in assigned tasks by offering them worked-out example solutions that are most relevant to the task and that are customized to the student's current knowledge (student knowledge is typically represented by the marks they score in tasks or tests and is stored in a specific component of the ITS called student model). This study uses an example-based ITS called Example Recommendation System (ERS) (Chaturvedi & Ezeife, 2017) that requires tasks and worked-out examples to be structurally similar so that they can be mined to achieve the desired customization. A task in ERS is defined to be a gradable question or instruction assigned to students (e.g. task T1 for the domain of C programming: 'Write a C program that computes the area of a triangle, given its base and height.'). Similarly, a worked-out example (WE) refers to a complete or partial worked-out solution of a question or instruction (similar to examples in textbooks). For example, figure 1 shows a worked-out example E1, which is essentially the solution to the following instruction: 'Write a program that computes and prints the value of b, given b = a / b * a % b'. A learning unit (LU) in ERS is defined as the smallest basic unit of domain knowledge that a task or worked-out example can be divided into (e.g. "simple arithmetic expressions" is a LU in the domain of C Programming). These LUs belong to the domain model of the ITS and are typically defined by experts. Granularity refers to the level of detail with which an ITS chooses to represent its domain resources such as tasks and worked-out examples (Pardos et al., 2007). Assuming $\supseteq$ is interpreted as "consists of", an ITS may choose to represent its worked-out examples as (choice1: "Lesson $\supseteq$ Examples") or as (choice2: "Lesson $\supseteq$ Examples $\supseteq$ Learning Units"). Choice1 is an example of a coarse-grained system, relative to choice2, which is a fine-grained system. Choice2 can be interpreted as "each lesson consists of worked-out examples, which are further subdivided into basic learning units (LUs)" (e.g. lesson L1 consists of example E1, which has LUs {datatype, printf}). Research has shown that finer the granularity of ITS student models, more accurate is the prediction of student performance (Pardos et al., 2007). In an attempt to design a fine-grained domain and student model, ERS uses the power of regular expression

978-1-5386-0958-3/17 $31.00 © 2017 IEEE
DOI 10.1109/CIT.2017.34

	Solution	Extracted Learning Units

Example E1	#include <stdio.h>
	int main(){
	
	int a = 2, b;
	b = a / 2 * a % 2;
	printf("b = %d\n", b);
	}
		L1: variable declaration
		L2: assignment instruction
		L3: printf-variables
		L4: printf-mixed
		L5: printf-message
		L6: scanf
		L7: arithmetic expression-simple
		L8: arithmetic expression - compound

Figure 1. Worked-out example $E1$ and its LUs

analysis (Dubé & Feeley, 2000) to extract individual LUs from its tasks and worked-out examples (Chaturvedi & Ezeife, 2017) and represent them in vector space so that they can be mined to facilitate ITS functionality such as customization. In comparison to ERS, existing ITS that attempt to attain a fine-grained system use extraction methods that are either manual (Li & Chen, 2009), where experts provide the list of LUs for each worked-out example in its ITS or use extremely complicated and resource-intensive automated methods (Yudelson & Brusilovsky, 2005; Mokbel et al., 2013).

Predicting student performance (PSP) in ITS can be very useful to educators in answering questions such as "Are students offered the appropriate resources at the right time" and "What percent of students have difficulty in succeeding in the subject and what are the reasons?". The problem of PSP is also useful to ITS designers in answering questions such as "What is the likelihood that students will succeed in the given tasks using the customized resources that are offered by the ITS?" and "Are the features used for prediction sufficient to predict student success accurately?". Our research attempts to answer the latter 2 questions by designing a predictive model that uses task-driven and objective features such as *average grade of all LUs that belong to a task* and *difficulty level of the task*.

A. Outline of the paper

This paper is organized as follows. Section 2 presents related work and our motivation for this research. Section 3 presents the proposed methodology to predict student success. This section includes data preparation, processing, list of proposed features and algorithms to derive or extract them and the data mining techniques used to build the prediction model. Section 4 presents the dataset used in the proposed method, and the results and analysis of the prediction model built for this study. Section 5 presents conclusions, limitations and future works.

II. Related Works

Predicting academic performance of students has been a challenging problem for intelligent tutoring systems. There are several ITS that do predict student performacne but they either lack in the use of state-of-the-art techniques that can predict student performance accurately or lack in the selection of appropriate features that should be used to predict student performance. Minaei-Bidgoli et al., (2003) in their study of an ITS called LON-CAPA use web-log features to predict student performance in the final exam as Pass/Fail with a prediction accuracy of 87%. Features they use include total number of correct answers that a student has given, student's success at the first attempt for each task, total number of attempts made in each task to get the correct answer and total time spent on each task until solved. Neither of these features are based on student's knowledge and are not good indicators of the student's performance in the final exam. On the contrary, more time spent by students on tasks may allow them to learn that topic better, and therefore may lead to a better performance in the final exam. Another limitation is the absence of student-centric features such as student's current performance on the graded tasks done so far to predict his/her final performance. Thai-Nghe et al., (2010) use the technique used in recommender systems for predicting student performance. A recommender system (RS), in general, is an information filtering method which links users to items. For example, if there are m users and n items, a RS will arrange them as an m * n matrix M, such that $M(i, j) = 1$, if user i likes item j; 0 otherwise. Typically, many entries in M are missing (e.g. when there is a new user who hasn't liked any item yet) and the RS predicts those missing values based on the information of the other existing users using techniques such as clustering and collaborative filtering (Markov & Larose, 2007). The authors in their study map students to users, tasks to items and then assign a rating to the student-task pairs. A limitation of mapping the problem of PSP to recommender systems is that prediction in RS is based on web usage patterns of users, whereas student models in ITS are more concerned with web content usage (e.g. student's knowledge on a given task). Shen et al., (2010) propose a system very similar to that of Minaei-Bidgoli et al., (2003) but they use a finer level of granularity to extract its features specific to each step of a task. Each task is divided into several steps by experts and student's knowledge on each step is predicted. Their system too, like many others, relies on logged features extracted from student's interaction with it, instead of student's current knowledge on the domain. McCuaig and Baldwin(2012) predict student success without using formal assessments such as quiz or exam grades. They use features such as student's mean confidence for the overall semester (by asking students to fill a questionnaire each week), total number of active days spent using their system and the average time spent in doing problem sets (students attempt an ungraded problem set each week) to predict student success using decision trees. For example, student s1 represented by <s1, mean confidence = 4, total days active = 40, mean time spent on problem sets = 5 minutes> is predicted to fail the course. The low value of prediction accuracy for their method (70%) can be attributed to the subjective nature of features such as confidence level (entered by the student) and the lack of task-oriented features. There is no direct measurement of student knowledge either, since the problem sets they use are ungraded. As indicated above, all the existing methods

that claim to predict student performance suffer from a major limitation - improper feature selection. This is due to use of features that are either subjective to student opinion (e.g. asking a student for his / her confidence level on the subject) or rely on interactive features such as 'time to complete a task' that do not directly measure student's knowledge and tend to give misleading results. For example, time to complete a task will be recorded as very high in situations where student starts working on a task but does not logout after completion (in this scenario, 'time to complete the task' will be recorded as high - an (incorrect) indication to the ITS that the student is struggling with the subject). To mitigate these limitations, we propose to predict student performance by extracting or deriving features that are objective, fine-grained, are well-informed about the tasks students are assigned and also well-informed about the resources that assist in the task (such as worked-out examples that assist in completing a task successfully) and then mine these features using existing classification algorithms such as decision trees. Examples of such features include student's performance on a task's LUs, student's current knowledge on the worked-out examples that assist in the task, difficulty level of these examples and so on. This enables us to achieve a higher prediction accuracy for student success in a task and use the model for decision-making.

III. Proposed Methodology to predict student success

The problem of predicting student performance in this paper is mapped to the data mining problem of prediction using features that are task-oriented and student-centric. We state our hypothesis H1 as:

Hypothesis H1: A student's performance in online systems such as ITS can be predicted with a high level of accuracy, if its domain and student model are represented using a fine level of granularity such that task-oriented, objective and student-centric features can be extracted and mined for prediction.

By task-oriented features, we mean features that describe that task directly (e.g. difficulty level of the task) or indirectly (e.g. difficulty level of the worked-out examples recommended for a task). A fine level of granularity enables us to use objective measures such as marks scored by students on LUs contained in a task. By student-centric features, we mean those features that describe the current state of a student's student model. Examples of student-centric features are student's current overall performance in ERS's domain and student's marks in the worked-out examples suggested by ERS for a task. The above hypothesis is a sub-hypothesis of a much broader and pertinent question we answer in our ongoing research (Chaturvedi & Ezeife, 2017) on the likelihood of students succeeding in a given task using the worked-out examples offered to them by ERS. Our assumption is that students who use ERS, more specifically, those who study the examples suggested by it have a higher chance of succeeding in an assigned task (where success is reflected by high grades).

In order to validate H1, we prepare our domain and student data to suit the data mining techniques we propose to use for predicting student success. This section describes the task of data preparation for mining, features used and their extraction algorithms. Two predictive mining methods are used on the extracted features : Decision Trees and Naive Bayes methods. The reason for selecting decision tree analysis for this study is that decision trees are easy to explain and interpret. They allow to identify and analyze all possible alternatives for a decision and to generate rules that could be easily applied to new unseen data records. Naive Bayes is selected because it is a simple and an efficient method, and works well with small datasets (Pang-Ning et al., 2005).

A. Preparing ERS data for prediction

ERS breaks down each worked-out example and task solution in the domain into basic learning units (LU) they contain using an algorithm called KERE (Knowledge Extraction using Regular Expressions) (Chaturvedi & Ezeife, 2017). The main inputs to KERE are (a) a set of regular expressions RE (defined by ERS domain experts), one for each LU in the domain of ERS and (2) a worked-out example or task solution wt represented as a string. KERE compares wt with each regular expression in RE and generates an output of those LUs whose regular expressions match wt. For example, figure 1 shows a worked-out example E1 broken down into individual learning units, for a small scope in the domain of C programming (∂) with 8 LUs {L1: variable declaration, L2: assignment instruction, L3: printf-variables, L4: printf-mixed, L5: printf-message, L6: scanf, L7: arithmetic expression-simple, L8: arithmetic expression - compound}. Similarly, KERE breaks down task T1 shown in figure 2 into 3 learning units it contains: {L2, L3, L7}. Worked-out examples and tasks in ERS are designed to share the same structure so that they can be compared and analysed conveniently. After extracting the basic LUs of tasks and examples, KERE stores them as binary vectors of size n, where n is the total number of LUs in the domain and 1/0 at position i in the vecotr indicates the presence/absence of learning unit i in the input task solution or worked-out example wt. For example, task T1 in figure 2 for domain ∂ is stored as a vector [0, 1, 1, 0, 0, 0, 1, 0] and example E1 in figure 1 is stored as a vector [1, 1, 0, 1, 0, 0, 0, 1]. Students receive scores on each task they are assigned by ERS. In order to maintain granularity, student model of each student stores their individual scores in each LU of ERS. ERS captures the student and domain data through a website (Chaturvedi et al., 2015) created using Python and its high-level web framework called Django (Alchin, 2013).

B. Features and their Extraction algorithms

Raw data, usually, is not in a form suitable for prediction algorithms but features from it can be constructed that allow these algorithms to learn (Domingos, 2012). Nine features are carefully chosen for each student Sid and each task Tid in ERS, in order to meet the objectives of the proposed mining model for PSP. We discard the student Ids since they are not

	Task as shown to students	Task solution as stored in the domain model
Task T1	Write statements to find and print the last digit of an integer x, where x = 123.	last_digit = x % 10; printf("%d", last_digit);

Figure 2. A task solution in ERS designed to have the same structure as its examples

	f1	f2	f3	f4	f5	f6	f7	f8	f9
TASK_ID	COP	GSE1	GSE2	VE1	VE2	DURATIONSE1	DURATIONSE2	DIFFICULTY_LEVELSE1	DIFFICULTY_LEVELSE2
T14	0.52	16	14	Y	Y	261.2	1035.9	Easy	Easy
T14	0.95	30	26	Y	Y	437.2	1771.28	Easy	Easy
T14	0.89	28	24	Y	Y	214.1	653.59	Easy	Easy
T14	0.89	25	21	Y	Y	1021.74	821.7	Easy	Easy
T14	0.84	22	18	Y	Y	533.9	370.72	Easy	Easy
T14	0.95	30	26	Y	Y	1021.74	2381.66	Easy	Easy
T14	0.87	26	22	Y	Y	6815.65	1583.1	Easy	Easy
T14	0.95	30	26	Y	Y	50689.56	12961.4	Easy	Easy

Figure 3. Partial dataset prepared for PSP - 8 rows shown correspond to marks scored by 8 students in task T14

required for prediction. Figure 3 shows the first 8 rows of the dataset prepared for PSP. Features include {feature f1: Sid's current overall performance (COP), features f2 and f3: grades in the worked-out examples suggested by ERS (GSE1 and GSE2) (we use the top 2 suggested examples in the current research), features f4 and f5: whether student has visited the suggested examples (VE1 and VE2), features f6 and f7: time spent on the suggested examples (DurationSE1 and DurationSE2), features f8 and f9: difficulty level of the suggested examples (Difficulty_LevelSE1 and Difficulty_LevelSE2).

Features and algorithms used to derive them are listed next.

1) Feature f1: COP (Current Overall Performance) : Student model of each student in ERS stores the marks they achieve in each learning unit (LU) in ERS's domain. COP is derived by finding the average performance of a student in all the LUs learnt so far.
2) Feature f2 : GSE1 (Grade in the Suggested Example 1): Algorithm 1 (called GSE) explains the steps required to derive this feature. ERS students are graded on assigned tasks and these grades are distributed among the tasks's LUs. Features 2 and 3 compute the grades students score in worked-out examples using the grades of their LUs. GSE takes as input (1) the current task Tid's represented as a binary vector of n LUs, (2) each worked-out example, also represented as a binary vector of n LUs (GSE stores these vectors as a binary matrix of size m * n, were m is the total number of worked-out examples) and (3) student Sid's current scores in each LU. Step 1 of GSE searches for those worked-out examples that are closest to task Tid (called as $List_{relevant}$) using a well-known data mining algorithm called k-nearest neighbors (Pang-Ning et al., 2005). K-nearest-neighbor (k-nn) algorithm finds the k nearest neighbors of a test sample t in a given set S. The core of k-nn algorithm is the similarity function that it uses to compare t with each data sample in S. Then, it sorts these similarity values and picks the top k samples – these are the k nearest neighbors of test sample t. The choice of an appropriate similarity function is driven by the nature of the attributes of the sample dataset. ERS domain and student model data is chosen to be asymmetric and binary (Chaturvedi & Ezeife, 2014, 2017). An asymmetric binary attribute regards only the presence of 1 as significant. In ERS, it is the presence of a LU in a worked-out example that is significant and therefore the binary values representing LUs in each worked-out example or task solution in ERS are chosen to be asymmetric. We demonstrate this with the following example. Let E1 = [1, 0, 1, 0, 0, 0, 0] and E2 = [0, 1, 1, 0, 0, 0, 0], where E1 and E2 are worked-out examples. If both values 1 and 0 are given equal importance, then E1 and E2 will be considered as similar since 5 out of their7 LUs match. But this is misleading. E1 and E2 are dissimilar (and rightly so) if zeros are ignored when matching them (only 2 out of 7 LUs match). With this rationale, all worked-out examples and task solutions in ERS are represented as vectors of binary and asymmetric values. The similarity function that GSE chooses to use for ERS is Jaccard's coefficient of similarity (JC), since it ignores the matching zeros in its formula and therefore caters best to asymmetric and binary data (such as ERS's) (Pang-Ning et al., 2005). JC between two binary vectors x and y is measured as

$$JC(x, y) = \frac{f_{11}}{f_{11} + f_{01+}f_{01}} \quad (1)$$

where f_{11} is the frequency of occcurence of 1 and 1 in the corresponding bits of x and y, f_{01} is the frequency of occcurence of 0 and 1 in the corresponding bits of x and y and f_{10} is the frequency of occcurence of 1 and 0 in the corresponding bits of x and y. Terms f_{01} and f_{10} in equation 1 represent the non-matching attribute pairs. For example, $JC(E1, E2)$ = 1 / 3. Step 2 of GSE fetches the LUs of each of the k worked-out examples in $List_{relevant}$ and stores them in E1_LU. For example, if $List_{relevant}$ for a given task TP1 is [EP2, EP4], then E1_LU = [LU1, LU2, LU5], assuming that EP2 consists of these 3 LUs. Step 3 computes the sum of grades student Sid scores in worked-out example 1 stored in $List_{relevant}$. For example, GSE1 for student Sid = grade[LU1] + grade[LU2 + grade[LU5].
3) Feature f3: GSE 2 (Grade in the Suggested Example 2) is extracted using the same algorithm GSE and steps used for feature 2. Using the same $List_{relevant}$ in step 2, E2_LU = [LU3, LU5] and GSE2 in step 3 computed for student Sid = grade[LU3] + grade[LU5].
4) Feature f4: VE1 (Visited Example 1) is extracted from the weblogs generated by student interaction with ERS for worked-out example1 in $List_{relevant}$. It indicates

Algorithm 1 GSE (Grade in the Suggested Example)

Input: 1. task Tid as a binary vector of size n, 1/0 indicating presence/absence of LU (where n = number of LUs in the ITS)
2.LU_EX: binary matrix of size m examples * n LUs
3. $grades_{Sid}$: vector of size n that holds student Sid's current scores in each LU
Output: grade in suggested examples 1 and 2: $GSE1$ and $GSE2$
Other variables: $List_{relevant}$: list of k examples most relevant to Tid
Method:
***begin of GSE
1. Find worked-out examples that are closest to task Tid and store in $List_{relevant}$.

1.1. Compute similarity between task Tid and each example i in LU_EX using Jaccard's coefficient JC (equation 1).

1.2. Sort JC values computed in step 1.1 in ascending order and store the corresponding examples of top k (=2) of them in $List_{relevant}$(lets call them E1 and E2)
2. Find E1's LUs
$E1\_LU = LU\_EX[E1]$
3. Find Sid 's grade in example E1, n = number of LUs

$$GSE1 = \sum_{i=1}^{n} grade_{Sid}[E1\_LU[i]],\ \forall E1\_LU[i] = 1$$

4. Repeat step 3 for example E2 to find $GSE2$
***end of GSE

whether a student has visited the suggested example or not.

5) Feature f5: VE1 (Visited Example 2): same as feature f4 but for example 2 in $List_{relevant}$.
6) Feature f6 : DurationSE1 is derived by finding the sum of the time spent (in seconds) on the worked-out example E1 in $List_{relevant}$.
7) Feature f7 : DurationSE2 is derived in the same way as feature 6 for worked-out example E2 in $List_{relevant}$.
8) Feature f8: Difficulty_LevelSE1 (Difficulty level of suggested example 1): example1 in $List_{relevant}$ is assigned a difficulty level (E (easy) or D (difficult)) using algorithm findDL (this algorithm was proposed in an ongoing research (Chaturvedi & Ezeife, 2017)).
9) Feature f9: Difficulty_LevelSE2 (Difficulty level of suggested example 2): example1 in $List_{relevant}$ is assigned a difficulty level (E (easy) or D (difficult)) (same as feature f8).
10) Target attribute SUCCESS_IN_TASK_YN: SUCCESS_IN_TASK_YN is assigned a yes, if a students succeeds in the assigned task and no otherwise. It is assumed that a student succeeds in a given task if he/she has achieved a grade of 75 or higher in it.

C. Data Mining model and techniques used for predicting student performance

Given a set of data samples, each represented as a tuple (x,y) where $x = (x_1, x_2, ..x_n)$, (each x_i is a feature or attribute in x), prediction is the task of mapping the attribute set x into attribute y, where y is termed as its class label (or target attribute). A predictive data model divides its data into 2 subsets: $training$ and $test$. $Training$ dataset is used to learn the model, whereas $test$ dataset is used to test the model. First the model is built by applying a mining algorithm (such as decision trees) on the $training$ dataset. Then, this model is applied to the $test$ dataset and their actual class labels are compared to the predicted ones to evaluate the model's performance. Thereafter, this model can be used to classify unseen records.

The mining algorithm chosen in this study to build ERS's prediction model is decision tree analysis (although we experimented with other prediction algorithms). The reason why decision tree analysis is selected for this study is that it allows us to identify and analyze all possible alternatives for a decision and to generate rules that could be easily applied to new unseen data records. They are easy to explain and interpret as well.

IV. Results and analysis

In this section, we first describe the dataset used by ERS for predicting success (section IV-A). Results of predicting student performance using mining techniques are presented in section III-C.

A. Dataset used for prediction

The dataset for this study consists of the following from the domain of C programming: 13 graded tasks, 200 worked-out examples and 26 learning units (LUs). It also consists of a small set of 10 students who were registered in Fall 2015 at the University of Windsor and who consented to participating in this study. This generates a dataset (D) with 130 records (one for each student per task); each record has the 10 features (including the target attribute success) as described in section III-B.

In order to justify our selection of features further, we compute the correlation between them using Pearson correlation coefficient. Pearson correlation coefficient (ρ) between 2 features x_i and y_i is a measure that quantifies the strength and direction (positive or negative) of the relationship between them, provided x and y share a linear relationship (Pang-Ning et al., 2005) and is defined by equation 2. In this equation, $\bar{x}$ is the mean of all values of x and $\bar{y}$ is the mean of all values of y. For example, (example adapted from Pang-Ning et al. (2005)), if x = (1, 1, 0, 1, 0, 1) and y = (1, 1, 1, 0, 0, 1), then $\bar{x}$ = 4/6 = 0.667; $\bar{y} = 4/6$= 0.667; $\sum_i (x_i - \bar{x})(y_i - \bar{y})$ = 0.333; $\sum_i (x_i - \bar{x})^2 = 1.333$; $\sum_i (x_i - \bar{x})^2 = 1.333$; $\rho = 0.333/\sqrt{(1.333 * \sqrt{1.333}}$ = 0.25. A value of $\rho = 0.25$ indicates that the correlation between x and y in this example is a weak relation. In general, ρ is a value between -1 and +1. Closer the value to 0, weaker is the

relationship. A 1 / -1 indicates a perfect positive / negative relationship. Experiments indicate that the value of ρ for the feature set used in this study is always higher than +0.62, implying a moderately strong positive correlation between them. For example, $\rho(durationSE1, COP) = 0.70$ indicates that there is a positive correlation between these 2 features and that there is a tendancy for feature COP to go high when $durationSE1$ goes high. Similarly, $\rho(GSE1, COP) = 0.62$;$\rho(durationSE2, COP) = 0.71$; $\rho(GSE2, COP) = 0.69$.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} \quad (2)$$

There are two concerns with dataset D - first, D obviously is a small dataset; second, the class label attribute (success) is imbalanced because it contains more of class label SUCCESS_IN_TASK_YN = 'yes' (113 instances) as compared to SUCCESS_IN_TASK_YN = 'no' (17 instances). Although this is a good indication for a tutoring system (more students succeeding in tasks), it creates an undesirable bias for classification algorithms used to predict student performance. In order to overcome these two concerns (small and imbalanced dataset), this study uses an existing over-sampling algorithm called SMOTE (Chawla et al., 2002) proposed by Chawla et al. SMOTE, (Synthetic Minority Oversampling Technique) is an over-sampling technique used to overcome the issues of imbalanced datasets by adding volume to the minority class label, so that the instances of majority and minority class labels are equally distributed (Chawla et al., 2002). Although the objective of SMOTE is to balance existing class labels, it does so by adding more samples or records and therefore allow us to generate simulated data using the original dataset. SMOTE adds samples by taking 5 nearest neighbors (NN) of a minority class sample X. It finds the difference between feature vector of X and feature vectors of the NN of X, multiplies this difference by a random number between 0 and 1 and then adds the resulting value to the original value of X. Equation 3 represents this idea where X is the original instance/sample; X_{new} is the newly generated sample, X_{NN_i} is one of the 5 NN of sample X; δ represents a random number between 0 and 1. For example, if feature vector of $X = (6, 4)$ and one of its 5 NN is $X_{NN_i} = (4, 3)$, then one of new samples generated is $X_{new} = (6, 4) + (2, 1) * 0.5 = (6, 4) + (1, 0.5) = (7, 4.5)$.

$$X_{new} = X + (X_{NN_i} - X) * \delta \quad (3)$$

This study uses a data mining tool called Weka (Hall et al., 2009), to implement SMOTE on its original dataset D. The original dataset D has 113 (out of 130 samples) that have a target attribute SUCCESS_IN_TASK_YN with a value of 'yes', whereas only 17 (out of 113) have a value of 'no'. The result of applying SMOTE to D not only met the main objective of balancing D towards the target attribute, it also increased the volume of D considerably. The new dataset, after applying SMOTE to D, had a total of 520 samples, 264 instances of them had a value of 'yes' for its target attribute and 256 instances (out of the 520) had a value of 'no'. We call this set as DS.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

$$precision = \frac{Number\ of\ actual\ positives\ correctly\ predicted}{number\ of\ predicted\ positives}$$

$$recall = \frac{Number\ of\ actual\ positives\ correctly\ predicted}{number\ of\ actual\ positives}$$

$$f_{score} = \frac{2}{\frac{1}{r}+\frac{1}{p}} = \frac{2 * r * p}{r + p}$$

Figure 4. Performance measures for classification algorithms used to predict student performance

B. Results of predicting student performance

We perform experiments for predicting student performance using decision trees, using both, the original dataset (D) of 130 instances and the original+simulated (DS) dataset of 520 instances. Tables 1 and 2 show the performance of decision tree analysis on datasets D and DS in terms of the confusion matrix and measures such as accuracy, precision, recall and f_score. A confusion matrix is a table that allows visualization of the performance of a classification algorithm. It gives a count of data records or instances that are correctly and incorrectly predicted by the algorithm. In general, for a 2-class problem (such as the one used in this study), labels are termed as positive / negative; the original class labels are referred to as actual and those determined by the classification algorithm are termed as predicted. The classification algorithm then, assigns to each instance one of the following : True Positive (TP: actually positive - predicted as positive), True Negative (TN: actually negative - predicted as negative), False Positive (FP: actually negative - predicted incorrectly as positive) and False Negative (FN: actually positive - but predicted incorrectly as negative). The number of FP (false positives) is of concern here as it gives misleading information to students and teachers. FP in this study measures the number of students whose success is wrongly predicted to be yes (where actually they do not succeed). Performance measures used for evaluating the classifiers for decision tree and naive bayes are accuracy, recall and f_measure and are shown in figure 4.

The total number of FP (false positives) and FN (false negatives) in a confusion matrix indicate erroneous results. For example, FP in table I measures the total number of students whose success is wrongly predicted by PSP to be yes (although in reality, those students do not succeed). Similarly, FN measures the total number of students who actually succeed but the model incorrectly predicts them as failures. With decision trees, the number of FP in the original dataset D is found to be 6 out 130 instances (4.6%) , whereas it is less than 1% (4 / 520) on the simulated dataset DS. PSP's decision

	Predicted = Yes	Predicted = no
Actual = Yes	112	1
Actual = No	6	11

(a) Confusion matrix for D

	Predicted = Yes	Predicted = no
Actual = Yes	245	19
Actual = No	4	252

(b) Confusion matrix for DS

Table I
CONFUSION MATRIX GENERATED BY DECISION TREE MODEL WITH ORIGINAL DATASET D (130 INSTANCES) AND (ORIGINAL+SIMULATED = 520 INSTANCES) DATASET DS

tree model, when applied to the simulated and larger dataset DS achieves much higher values of accuracy and f_score, as compared to the original dataset, as shown in table II. Both accuracy and f_score are as high as 96% when class labels are predicted using the simulated dataset DS with 520 instances as compared to 91% and 89% for dataset D with 130 instances. Evidently, the reason for misclassification in D for the minority class (SUCCESS_IN_TASK = no) is imbalance in distribution of the two class labels (SUCCESS_IN_TASK = yes and SUCCESS_IN_TASK = no) and too few training instances of the minority class for the model to learn accurately. Such high values of performance measures such as accuracy and f_score validate our hypothesis H1 that a prediction model can be built with a high f_score and accuracy by selecting features that have proper knowledge on the assigned tasks and those that measure student knowledge objectively. Figure 5 shows a decision tree generated from DS. As an example, one of the rules generated by this tree is:

if the average current grade of a student is > 86%
  success = yes
else if the average current grade of a student is > 81%
  if average grade in LUs of example SE2 > 49%
    success = yes
  else if time spent of SE2 < 0.009 (9 minutes)
    if average grade in LUs of example SE1 > 68%
      success = yes
    else success = no

These rules indicate that if a student s has not performed well in the learning units of suggested examples and has not spent enough time on them, then s is less likely to succeed in the task. Even if student s's current performance is average or above average, the rules indicate that the student still has to achieve a certain level of grades in the LUs of the suggested worked-out examples, which yet again asserts the importance of students using and understanding of the worked-out examples suggested for each task by ERS.

V. CONCLUSIONS AND FUTURE WORK

The main objective of building the PSP model is to accurately predict student success for assigned tasks in a fine-grained ITS system by proposing features that are focused on the task's resources such as similar worked-out examples suggested by the ITS and student's knowledge on these resources.

	Dataset D (size=130)	Dataset DS (size=520)
Accuracy	91 %	96 %
Recall	90 %	96 %
Precision	91 %	96 %
F_Score	89 %	96 %

Table II
PERFORMANCE MEASURES: DECISION TREE USING DATASETS D AND DS

Our proposed method is able to extract meaningful task-based features and implement them to predict student performance using decision trees with accuracy and f_score values as high as 96%. Existing ITS that predict student performance surveyed in section II have much lower values of accuracy, which can be attributed to their improper selection of student and domain model features. This validates our hypothesis that student performance is predicted with a high value of accuracy if features used for prediction are well-informed about the assigned tasks and are measured objectively. The rules generated by decision trees allow us to analyze and take informed decisions on ERS's future students.

REFERENCES

Alchin, M. (2013). Django is python. In *Pro Django* (pp. 11–40). Springer.

Chaturvedi, R., Donais, J., & Ezeife, C. I. (2015). Ers at https://ritu106.cs.uwindsor.ca/.

Chaturvedi, R. & Ezeife, C. (2014). Mining relevant examples for learning in its student models. In *2014 IEEE International Conference on Computer and Information Technology* (pp. 743–750).

Chaturvedi, R. & Ezeife, C. I. (2017). Task-based example mining for learning in an its. Submitted to Knowledge and Information Systems.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, (pp. 321–357).

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10), 78–87.

Dubé, D. & Feeley, M. (2000). Efficiently building a parse tree from a regular expression. *Acta Informatica*, 37(2), 121–144.

Gog, T. & Rummer, N. (2010). Example-based learning: Integrating cognitive and social-cognitive research perspectives. In *Edu. Psych. Rev., 22* (pp. 155–174).

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 10–18.

Li, L. & Chen, G. (2009). A coursework support system for offering challenges and assistance by analyzing students web portfolios. *Educational Technology & Society*, 12,2, 205–221.

Markov, Z. & Larose, D. (2007). *Data mining the web -*

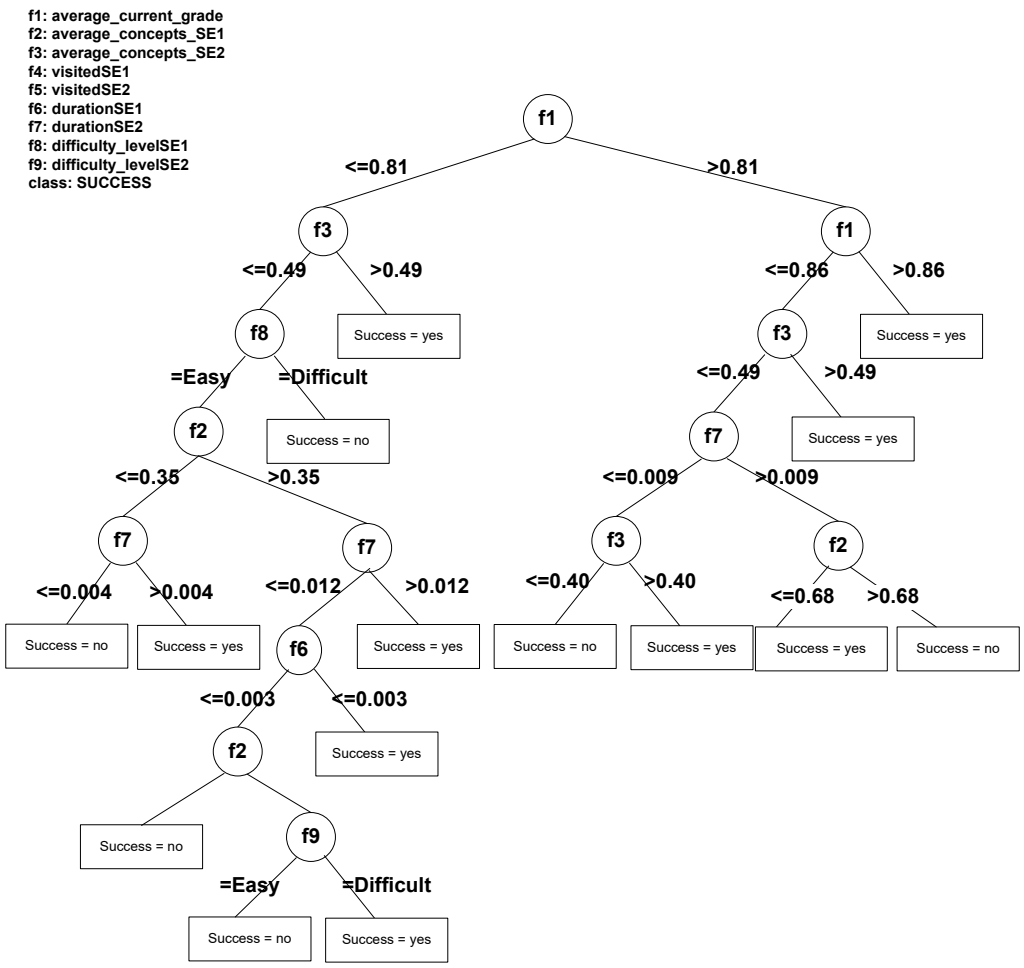

Figure 5. Decision tree generated by Weka's J48 for student dataset with 520 instances

Uncovering Patterns in Web Content, Structure and Usage. John Wiley.

McCuaig, J. & Baldwin, J. (2012). : (pp. 160–163).: ERIC.

Minaei-Bidgoli, B., Kashy, D. A., Kortemeyer, G., & Punch, W. (2003). Predicting student performance: an application of data mining methods with an educational web-based system. In *Frontiers in education, 2003. FIE 2003 33rd annual*, volume 1 (pp. T2A–13).: IEEE.

Mokbel, B., Gross, S., Paassen, B., Pinkwart, N., & Hammer, B. (2013). Domain-independent proximity measures in its. In *Sixth ACM International Conference on Educational Data Mining-EDM 2013, July 6 to 9, 2013, Tennessee,USA* (pp. 334–335).

Pang-Ning, T., Steinbach, M., & Kumar, V. (2005). *Introduction to Data Mining*. Addison-Wesley.

Pardos, Z. A., Heffernan, N. T., Anderson, B., & Heffernan, C. L. (2007). The effect of model granularity on student performance prediction using bayesian networks. In *User Modeling 2007* (pp. 435–439). Springer.

Renkl, A. (2014). Toward an instructionally oriented theory of example-based learning. *Cognitive Science*, 38(1), 1–37.

Shen, Y., Chen, Q., Fang, M., Yang, Q., Wu, T., Zheng, L., & Cai, Z. (2010). Predicting student performance: A solution for the kdd cup 2010 challenge. In *Proceedings of the KDD Cup 2010 Workshop held as part of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Thai-Nghe, N., Drumond, L., Krohn-Grimberghe, A., & Schmidt-Thieme, L. (2010). Recommender system for predicting student performance. *Procedia Computer Science*, 1(2), 2811–2819.

Yudelson, M. & Brusilovsky, P. (2005). Navex: Providing navigation support for adaptive browsing of annotated code examples. In *In Proceedings of 12th International Conference on Artificial Intelligence in Education, AIED.* (pp. 18–22).