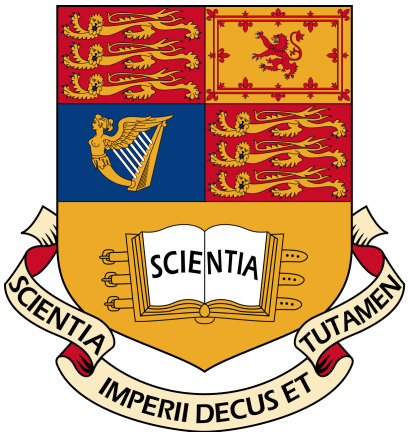
Imperial College London
Department of Electrical and Electronic Engineering



Project Title:	**Machine Learning for Personalising Assistance for Children with Diabetes**
Student:	**Sagar Patel**
CID:	**00842688**
Course:	**EIE4**
Project Supervisor:	**Professor Yiannis Demiris**
Second Marker:	**Dr K.M. Mikolajczyk**

Abstract

This project concerns the research and design of a system, with the specific goal of providing personalized chronic disease education to children with diabetes via task-based learning. Studies have shown that educating diabetic patients could improve patient health outcomes and reduce the cost burden on health services, highlighting the need for a system of this type. There are two underlying objectives of this work: providing a personalized learning strategy specific to each child, allowing them to maximize their learning potential; and rapidly bootstrapping a new user-model for a child using the system for the first time, in order to provide accurate learning strategies immediately. This work focuses on applying recommendation system (RS) technology, commonly used with movie recommendation, to the domain of education. Proposed solutions include the use of Random Forests, Decision Trees, Latent Factor Models and Logistic Regression. It is observed that for both objectives, the proposed solutions could improve the quality of learning strategies suggested. Further to this, experiments show that the system can bootstrap a new user-model accurately, after the completion of approximately 15 tasks.

Acknowledgements

I would like to express my deepest appreciation to Professor Yiannis Demiris, for his help and guidance throughout the course of this project and my time at Imperial College London. I would like to extend my sincere gratitude to Dr. Antoine Cully, for his invaluable advice and technical guidance throughout the past year.

I would also like to thank my family for their unconditional love and support, without whom it would not have been possible to complete this work. Finally I would like to wish all my friends and colleagues at Imperial College the best of luck in their future endeavours.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation & Objectives

Diabetes Mellitus is a group of metabolic diseases characterized by hyperglycemia resulting from defects in insulin secretion and insulin action [1]. Hyperglycemia can often lead to severe damage to the heart, blood vessels, eyes, kidneys and nerves [2].

As of 2016 the number of children living with Diabetes in the UK was estimated at 42,000, with the peak age for diagnosis between 9 and 14 years of age [3]. In addition to the physical effects of diabetes mentioned above, diabetic children are at greater risk of developing emotional and behavioural problems associated with the management of their chronic disease [4].For this reason various organizations have dedicated their resources to providing children with chronic disease management strategies, so that they can continue to lead long and healthy lives and in turn reduce the strain on health care providers

Imperial's Personal Robotics Laboratory is in collaboration with the Personal Assistance for Healthy Lifestyle[1] (PAL) project, with the aim to develop a suite of applications to assist children with diabetes. This project will aim to assist the development of a back-end system currently being developed for PAL, with the specific goal of personalizing the education of diabetes for each child.

[1]PAL project - http://www.pal4u.eu/

1.2 Report Overview

This report starts by introducing the background and context required for this project, discussing diabetes mellitus and highlighting the importance of education in the treatment of chronic disease. This chapter concludes by presenting the required knowledge that is fundamental to the understanding of this work. A description of the current PAL architecture is also provided.

The requirements capture section delineates the project deliverables and introduces the two modules that are developed in this project. The module design chapters, define the problem that is necessary to solve, and provide detailed explanation and analysis of the proposed solutions.

Finally in the results and evaluation sections, description of the testing methodology and baseline comparison measures used to evaluate each model are provided. Analysis of each module is presented and the overall system is evaluated. This report closes by reviewing the project achievements and presenting future work.

Chapter 2

Background & Context

This chapter presents the background knowledge required for this project, starting by discussing Diabetes Mellitus and highlighting the benefit of educating patients about their condition. An overview of the current PAL application is provided, before moving onto developing technical concepts that are relevant to this work, such as Intelligent Tutoring Systems (ITS) and Recommendation Systems (RS). Finally the relationship between ITS and RS is explained and a summary of how these technologies are used to develop an educational back-end system is provided.

2.1 Diabetes Mellitus

A chronic disease as defined by the World Health Organization (WHO) are those which have a 'long duration and slow progression, with no definite cure' [5]. Diabetes Mellitus is recognized as one of four prominent types of chronic illness alongside cardiovascular disease, cancer and chronic respiratory disease.

As of 2016 the number of people living with Diabetes worldwide is 400 million, with this figure expected to rise to around 642 million by 2040. Diabetes is associated with serious complications including heart disease, stroke and blindness, leading to disability and premature mortality [3]. A further consequence of Diabetes is the cost on health services within the UK, with the current burden at £23.7 billion rising to £39.8 billion by 2035 [6].

2.1.1 Diabetes & Patient Understanding

Educating a diabetic patient about their underlying condition involves providing information and teaching technical skills related to the disease. Corbin and Strauss [7] delineate three obstacles faced by people suffering from a chronic illness: the medical management of the condition and dealing with changing diet and exercise regime; having to deal with new responsibilities in everyday situations, regarding work, family and friends; and coping with the emotional & mental burden that comes with having a chronic condition. It is paramount that any education that a patient undertakes handles such obstacles.

Studies have shown that educating diabetic patients can improve health outcomes and reduce the cost burden on health services in some cases [8][9]. A study conducted over a period of two years by the Chronic Disease Self-Management Program (CDSM) [10], found that patients who participated in a CDSM clinical trial compared to those who did not, demonstrated significant improvement in exercise regime, cognitive system management and social/role limitations. There was also a trend that patients who participated in the trial spent fewer days in hospital and had fewer outpatient visits.

This project focuses on educating diabetic children about their underlying condition in the most efficient manner, with the aim of maximizing learning potential. The intention is that an improved understanding on how to handle living with a chronic disease, the better equipped the child is to lead a long and healthy life.

2.2 An Overview of the Current PAL system

PAL, the Personal Assistance for Healthy Lifestyle project supports children with diabetes, their families and any related medical staff. The project aims to help children acquire the knowledge, skills and habits that are essential for them to adhere to their diabetes regimen. The PAL project will aim to provide long-term and personalized self-management support via mobile health applications[1].

This work will assist the development of an existing application in the PAL architecture. The application has the intention of educating a child about their underlying condition through task-based learning (ie.quiz problems & games). At present the sole form of task-based learning is a quiz game, an example of the type of problem in a quiz activity can be seen in figure 2.1.

Problems within a quiz are structured as:

$$topic \supseteq question$$

where each quiz question belongs to a topic category. There are currently 35 different topics that a question can be based upon, each attempting to target a specific development goal that a child is attempting to achieve, these include learning about the disease, and dealing with medical treatment.

Topic	Quiz Question
Carbohydrates	Does fats and proteins influence your blood glucose?
Carbohydrates	What do you do when you want to eat cookies?
Carbohydrates	100 grams of brioche contain more carbohydrates than 100 grams of crackers
Carbohydrates	A tart fruit has more carbohydrates than a cookie
Glucose	You've got a hypo while playing outside and you didn't bring any dextro. You're not far from home. What should you do?
Diabetes general	How does type 1 diabetes develop?
Glucose	Why do you wash your hands or wipe the first drop of blood before you measure your blood glucose?

Figure 2.1: Sample Questions in PAL Quiz Game

[1]PAL project -http://www.pal4u.eu/index.php/project/about/

2.2.1 Current state of the PAL application

At present there are two components that dictate the operation of the current application, the dialogue manager and action selector. The dialogue manager is responsible for the communication across modules within the system, to determine the next action to take the dialogue manager will ask the action selector what to do next. For example upon meeting a child, the action selector will determine whether it should gather information about the child's age, hobbies etc. The dialogue manager also has the responsibility of starting and ending a quiz game.

During each quiz game, the action selector determines the next question to be presented, in the current application state this action is determined randomly, and it is the task of this project to provide a more informative question recommendation.

2.3 Intelligent Tutoring Systems

Intelligent tutoring systems (ITS) are computer based education systems that flexibly present educational content tailored to the idiosyncratic needs of each student, in order to increase learning performance[11]. For instance, Shute et al.[12] showed that students using an economics based ITS, performed as well as students taking a traditional economics course while only requiring half the amount of time to study the course material.

2.3.1 Components of an ITS

Research conducted by Woolf [13] suggests that four components are necessary to build an ITS, each component is summarized in table 2.1.

Student Model	The student model represents the ability of each student within the system. At a minimum it keeps track of student performance in each topic. Additional information can be gathered to represent a students cognitive & affective state. Moreover as a students knowledge and ability is assumed to be increasing [14] the student model can be built to account for this.
Domain Knowledge	The domain knowledge contains the information that is to be taught to a student, it contains the questions, concepts and rules of the domain to be learned.
Tutor Model	This component is responsible for guiding the student through the learning process. It is responsible for determining when to present the next activity or task to each student specific to the student model.
User Interface	This component is responsible for presenting information to each student, it determines how material should be presented to each student in the most effective manner.

Table 2.1: Summary of Intelligent Tutoring Components

The typical function of an ITS is depicted in figure 2.2, the tutor module is responsible for selecting an appropriate task to present to a student, using the learned knowledge in the student model. The response given by the student to this task, is compared with the expected outcome of that task, using the domain knowledge model. The student model is then updated accordingly, incorporating other supplementary data gathered by the system. The process repeats by utilizing the updated student model to choose the next appropriate task.

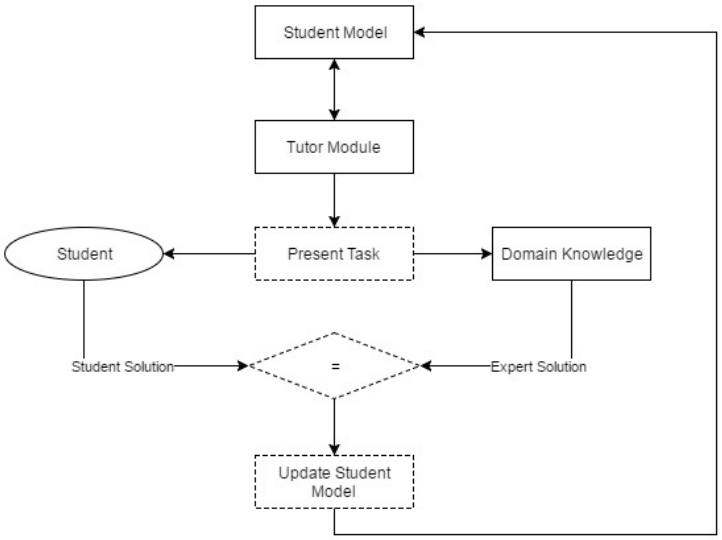


Figure 2.2: ITS: System Overview

The compatibility between recommendation systems (RS) & ITS has been highlighted by Thai-Nghe et al. [15]. This work will focus on building aspects of an intelligent tutoring system for PAL using state-of-the-art RS technologies.

2.4 Recommendation Systems

The function of a recommendation system (RS) is to determine the preference that a single user has for a particular set of items, and then use these preferences to provide recommendation for previously unseen items. Recommendation engines have been successfully utilized in many industries, helping to overcome information overload, while also providing personalized recommendation of content and services to users [16].

RS commonly have two classes of entities, these being users and items, they are generally represented in the form of a utility matrix [17], with each user-item pair r_{ui} representing the degree of preference by a user u for a particular item i (fig. 2.3).

The elicitation of user preference in order to populate a utility matrix has been the topic of much discussion, many of these concepts can be grouped as explicit and implicit methods [18]. Implicit methods infer a user preference, for example the amount

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1i} \\ r_{21} & r_{22} & r_{23} & \dots & r_{2i} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{u1} & r_{u2} & r_{u3} & \dots & r_{ui} \end{bmatrix}$$

Figure 2.3: Utility Matrix for Typical Recommendation System

of time one spends browsing a web-page, could indicate the degree of preference for that web-page. Alternatively explicit methods require active input from a user, often in the form of a rating/review system.

Historically two broad groups have been used to classify the types of recommendation systems: Content Based (CB) RS employ the idea that users will like unseen items that are similar to items they have previously liked; while Collaborative Filters (CF) make the assumption that users will like unseen items that other similar users have previously liked.

An issue with CB methods is that descriptive labels are needed to classify different items, which quite often requires some level of human input [19], as such they are not explored further in this work.

2.4.1 Collaborative Filtering

As aforementioned CF try to predict the preference of an item for a target user, by exploiting the preference of this item by other similar users. The fundamental assumption made in collaborative filtering is that an estimate for a target user's preference can be achieved by aggregating the preference of other similar users [20].

More formally, let U be the set of existing users within a CF system, I be a set of items and $R \to \mathbb{R}$ be the ratings given by U to I, such that:

$$D_{train} \subseteq U \times I \times R$$

are the observed ratings in the CF system, equally D_{test} are the unobserved ratings.

$$D_{test} \subseteq U \times I \times R$$

The prediction of an unobserved rating, given D_{train} then becomes:

$$\hat{r} : U \times I \to \mathbb{R}$$

such that the error between the actual rating r and prediction $\hat{r}$ is minimized. In CF systems this error metric is usually given as root mean square error (RMSE):

$$RMSE = \sqrt{\frac{\sum_{(u,i,r) \in D^{test}} \left(r_{(u,i)} - \hat{r}_{(u,i)}\right)^2}{|D^{test}|}} \tag{2.1}$$

Typically collaborative filtering solutions are categorized as model based *(Latent Factor)* or memory based *(Neighbourhood Based)*. Model based solutions compute offline models on the utility matrix, which can then be used to produce recommendations. In contrast, memory based solutions are computed in real time and operate directly on the utility matrix to produce recommendations.[21]

What follows is an overview of model and memory based methods as they will be useful in the context of this project.

Neighbourhood Based CF

The first stage in any memory based method is to calculate a user neighbourhood N for the target user u_t, such that $N \subset U$ represents all users that are most similar to the target user. Similarity between two users is determined using a similarity function, in practice Pearson's Correlation (eqn. 2.2) is most commonly used [22].

$$s(u, u_t) = \frac{\sum_{i \epsilon I_u \cap I_{u_t}} (r_{u,i} - \overline{r_u})(r_{u_t,i} - \overline{r_{u_t}})}{\sqrt{\sum_{i \epsilon I_u \cap I_{u_t}} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{i \epsilon I_u \cap I_{u_t}} (r_{u_t,i} - \overline{r_{u_t}})^2}} \tag{2.2}$$

Where $r_{x,i}$ represents the rating of user x for item i. $\overline{r_x}$ is average rating given to all items by user x. After determining a user neighbourhood we can generate predictions for the target user by aggregating the preferences of the users in N using eqn. 2.3.

$$\hat{r}_{u_t,i} = \frac{\sum_{u \in N} s(u, u_t)(r_{u,i})}{\sum_{u \in N} |s(u, u_t)|} \tag{2.3}$$

Latent Factor Model CF

Latent factor models operate under the assumption that ratings are heavily influenced by a set of factors that are specific to the problem domain. While these factors can be described (eg. the amount of action in movies), it is not feasible to determine every factor manually. Latent factor models aim to infer factors from a utility matrix using mathematical techniques.

Arguably the most common technique for latent factor analysis is Matrix Factorization (MF) [23]. MF aims to approximate the matrix $\mathbf{A} \in \mathbb{R}^{|U| \times |I|}$ by two matrices, $\mathbf{P}$ & $\mathbf{Q}$ such that their product approximates $\mathbf{A}$:

$$\mathbf{A} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{A}} \tag{2.4}$$

The matrix $\mathbf{P}$ represents the strength of association between users and factors, and $\mathbf{Q}$ represents the strength of association between items and factors. To determine optimal values for $\mathbf{P}$ and $\mathbf{Q}$, one could use stochastic gradient descent or alternating

least squares optimization.

Finally to generate a rating prediction for an item i made by user u, the dot product is calculated:

$$\hat{r_{ui}} = \mathbf{p}_u^T \mathbf{q}_i \tag{2.5}$$

2.4.2 User Cold Start Problem

The cold start problem arises when a new user enters a recommendation system and as the system is oblivious to the new user's item preference, it is unable to present this user with personalized recommendations. Failing to provide tailored content from an early stage of interaction between a new user and a recommendation system, may lead to users becoming disengaged with the system. Thus it is paramount that new users are rapidly modelled as they enter the system.[24]

Tackling the cold start problem has been the face of much research into CF in recent years, the proposed solutions can be categorized as being, **External** or **Internal** [25].

External solutions make use of external data-sources such as demographic data.

- Vozalis and Margaritis.[26] present a hybrid CF algorithm by combining traditional neighbourhood models with a demographic matrix representing all user to produce a more informative measure of similarity among users. This technique performed considerably well in the domain of movie recommendation, where demographic data (*age, gender & location*) would naturally provide useful information about a users preference for movies. However it is not clear the benefit of such a system where trends in demographics are not as clear cut.
- Rosli et al.[27] combine data mined from a user's Facebook profile with movie rating data to determine more accurate similarity measures between groups of users. This method relies on sufficient external data being readily available (existence of Facebook profiles), which may not be the case for all users.

Internal Solutions improve upon existing CF methods to determine similar users without requiring additional sources of data.

- Ahn.[28] described the limitations of existing similarity metrics and proposed the PIPS measure, a heuristic similarity measure which combines, the arithmetic distance between two ratings (proximity), the extent to which the item is preferred or disliked (impact) and to what degree two ratings deviate from the average rating of an item (popularity).
- Zhou et al.[29] develop a novel algorithm, Functional Matrix Factorization (FMF). FMF constructs an initial interview process using decision trees, with each node representing a question. The idea is that a new user will be queried adaptively on their preferences to common items and depending on their response, latent factors are associated with the user.

Interview based methods, present an interesting solution to the cold start problem, especially in this project for their parallelism to an examination in the educational environment. What follows is an introduction to two interview based algorithms, that are used to some extent in this work (*Information Gain via Clustered Neighbours* and *Adaptive Bootstrapping using Decision Trees*).

2.4.3 Information Gain via Clustered Neighbours

Information Gain via Clustered Neighbours (IGCN) is an information theoretic algorithm developed by Rashid et al.[30] The algorithm develops a dynamic interview process, such that a list of items to be rated is presented to a new-user, the order of items to be rated adapt according to the response given by the user after each question in the interview. Taking concepts from collaborative filtering, existing users of the system are considered to be grouped into user-neighbourhoods, IGCN assumes the goal of modelling a new-user becomes finding the best user-neighbourhood for the new-user.

IGCN works by repeatedly computing the information gain (IG) [31] of each item at every stage in the interview process, where necessary data to calculate IG is considered from the user neighbourhoods that best match the new user's model thus far. The subsequent item to be rated is one that has the largest information gain.

$$IG(a_{item}) = H(C) - \sum_{r} \frac{|C^r_{a_{item}}|}{|C|} H(C^r_{a_{item}}) \tag{2.6}$$

$$H(Z) = -\sum_{i=1}^{n} P(z_i) log_2 P(z_i) \tag{2.7}$$

Where $H(Z)$ denotes the entropy [31] of a discrete random variable Z. C denotes the distribution of users in neighbourhoods. $C^r_{a_{item}}$ signifies the distribution of users in each neighbourhood who rated item a_{item} with r .

$IG(a_{item})$ essentially expresses the reduction in required information toward the goal of finding the right neighbourhood by rating item a_{item}.

The IGCN algorithm (1) works in two stages:

1. At the first stage interview questions[2] are asked with the goal of building a general user model. Data from all users is considered when calculating IG.
2. At the second stage, the general user model is enhanced by asking tailored questions. IG in this case is calculated using data from user neighbourhoods that best match the general user model thus far.

[2] A question represents an item to be rated.

Algorithm 1 IGCN Algorithm

```
Create C user neighbourhoods
Compute information gain (IG) of items using data from all users.
repeat
    Ask the next question on the list of items ordered by IG score
    Update user profile
until user has rated I items
repeat
    Find the best L neighbourshood based on the profile so far
    Recompute IG based on the data of users in the best L neighbourhoods
    Ask the next K questions based on the list of items ordered by IG score
    Update user profile
until best L neighbours do not change
```

2.4.4 Adaptive Bootstrapping using Decision Trees

Decision trees (DT) are well recognized as useful prediction and classification tools [32]. Decision trees have been used by [24] & [33] to bootstrap a new user model, as they very clearly resemble an adaptive interview process. In general each node in a DT evaluates a preference for a certain item. Depending on a user's response the tree directs the user to a sub-tree, eventually leading to the leaf node that best characterizes them. The interview process equates to a new-user following a path from root to leaf rating different items.

Each node in the tree represents a group of existing users, the root node encompasses all users and each leaf node represents a small subset of these users. Each node can then be thought of as a neighbourhood, with the neighbourhood progressively refining as tree depth increases.

2.5 Recommendation Systems In Educational Environments

The most popular use of RS have been in the field of e-commerce and movie recommendation, however recently a shift in attitude in applying technology to education has seen the application of RS extend to educational environments.

Manouselis et al.[34] discuss Educational Recommendation Systems (ERS). These systems predominantly focus on recommending learning materials (*books*, *papers* and *courses*) to students. The group argue that unlike users in traditional recommendation systems which have a fixed time span (eg. when a user buys a recommended product to when they own it), users in an ERS achieve different levels of competency at different time spans. Thus an ERS should consider an individual's ability level when providing recommendations.

Thai-Nghe et al.[35] explore the temporal effect in task-based educational recommender systems to build a context aware recommendation system (CARS) [36]. This

system considers that the more a student studies the better they become at solving problems. The same group also attempt to predict performance levels of students for any given problem using traditional RS technology [15].

2.5.1 PAL Recommendation System

This project aims to create an Intelligent Tutoring System than can provide task based learning to children with diabetes, such that they can achieve their targeted learning objectives and competence levels.

The largest determinant of tasks selected in an ITS is the student model, as aforementioned the student model encompasses all information regarding how a student learns and how they adapt to new problems [37]. The quality of tasks provided by an ITS is therefore heavily dependant on the accuracy of the student model. Recall that quiz problems in the current PAL system are characterized as questions that belong to a topic category. The Student model in the current PAL system is therefore composed of a Topic and Question model.

- Topic model: represents the average performance across each topic in a quiz game by a single student.
- Question model: describes the specific performance in each question within a topic achieved by each student. There is a question model associated with each topic.

According to Vygotsky's, Zone of Proximate Development [38], when providing students with learning tasks, the difficulty of these tasks should be at a level slightly higher than the learners current competency level. Naturally competency level of a student can be determined through their performance results, thus being able to predict the performance of a student across all quiz problems, one can determine the most appropriate task to recommend.

Predicting student performance can be mapped to a recommendation system problem. Rather than predicting the rating given by a user to an item, the performance of a student in completing a quiz problem is predicted instead.

$$\begin{aligned} Student\,(S) &\mapsto User \\ LearningTask\,(T) &\mapsto Item \\ Performance\,(P) &\mapsto Rating \end{aligned}$$

The utility matrix for this problem is shown in figure 2.4, where p_{st} represents the observed performance for student s in task t. By utilizing CF techniques discussed earlier in this chapter, one can predict the unobserved performances.

Please note that by using CF techniques, the assumption made is that students experience the same developmental steps when learning, and two students in the same

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1t} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{s1} & p_{s2} & p_{s3} & \cdots & p_{st} \end{bmatrix}$$

Figure 2.4: Utility matrix in Educational Context

step have similar performance. Alternatively this can be thought as, two students with similar ability in some tasks will have similar ability in other tasks.

Finally it is important to consider the relationship between traditional RS and educational RS, especially when attempting to predict the student performance. A summary of these concepts taken from [15][34], is given below:

1. In a traditional RS it is assumed that a user rates an item once, this is not the case in an educational based RS as students have the possibility of repeating activities multiple times. The more they perform an activity the better they become at that activity and other similar activities. Thus temporal effects have to be considered when predicting performance

2. The difficulty of a task and the competence of a student when compared to other tasks/students have to be modelled, otherwise some students may receive inaccurate estimates for performance. These concepts are referred to as item-bias & user-bias in literature [39].

3. Unlike traditional RS where ratings are deterministic, task-based ERS suffer from 'slip' and 'guess' factors [40]. A slip factor is defined as the probability that a student knows the solution but accidentally answers incorrectly. A guess factor is the probability that the student guessed the answer and is correct.

Chapter 3

Requirements Capture

This project assists the development of a task-based learning application in the PAL architecture, more specifically this work shall concentrate on providing tailored task recommendation to new and active users of the PAL application. There are two primary features that this project specifically focuses on:

- The system is able to provide bespoke task recommendation to children using the knowledge present in the student model. Task recommendation should include both tailored topic suggestion and individual problem selection. As student performance increases over-time in most educational settings, the recommendation system should adapt its suggestion accordingly.
- The system is able to effectively handle the problem of providing valuable task recommendation to a new user of the PAL application. The system should consider the quality of recommendation and minimizing the amount of effort a new user undergoes before being provided with quality recommendation.

The overall deliverable of this project is a software implementation of the proposed recommendation system. As the PAL application is a real-world project less priority is placed on developing components that exist within PAL, such as the action selection and the dialogue manager. Instead emphasis is given to the components that solve the new-user problem and on-going task recommendation problem. In the following sections the design and analysis of two modules that have been developed for the PAL application are discussed, **New User Estimation Module** and **Task Recommendation Module** which aim to meet the objectives highlighted above.

Chapter 4

New-User Estimation Module

4.1 Problem Formulation

Intelligent Tutoring Systems are mechanisms designed to enrich education, by assisting each student through the learning process. A crucial element of an ITS is the student model, being able to accurately determine individual student models allows an ITS to select the right blend of exercises and choose an appropriate level of difficulty for each student [41].

The purpose of the New-User Estimation module would be to rapidly determine a student model for a new user to the PAL system. Once a suitable student model has been achieved, task recommendation can be provided as normal, the problem of task recommendation is discussed in later sections. Recall that a student model in the PAL system consists of a topic model and question model, as such the problem of estimating a new-student model becomes predicting the performance for each model.

When modelling a new-user the choice to predict the topic model over a question model is made, the motivation behind this choice shall become evident shortly. As aforementioned the topic model is the average performance made by a student across all questions within each topic, the performance in a single question is indicated by 0 (incorrect) or 1 (correct). Thus the performance $p_{s,t}$ of a student for a given topic is given by eqn. 4.1, where $q \cap s \in t$ represents the questions q that have been answered by student s in topic t.

$$p_{(s,t)} = \frac{\sum_{q \cap s \in t} p_q}{|(q \cap s) \in t|} \tag{4.1}$$

For this project, student performance is predicted using recommendation system technology, the problem of building a new user topic model directly translates to alleviating the cold start problem in a recommendation system. Mapping this problem to that of a recommender system, the utility matrix of existing student topic models is shown in figure 4.1a. When a new student interacts with the system the performance for each task in the utility matrix must be determined, forming a new-user performance model as shown in figure 4.1b

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1t} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{s1} & p_{s2} & p_{s3} & \cdots & p_{st} \end{bmatrix} \qquad \begin{bmatrix} \hat{p}_{u1} & \hat{p}_{u2} & \hat{p}_{u3} & \cdots & \hat{p}_{uj} \end{bmatrix}$$

(a) (b)

Figure 4.1: Utility Matrix for Cold Start Problem

According to [42], the elicitation strategy to obtain information about a new user must consider two factors: a) **User Effort**: The process of elicitation must not be burdensome otherwise the user may become disengaged with the system; b) **Model Accuracy**: If the elicitation strategy doesn't result in an accurate user performance model, this could hinder the objectives of an ITS.

Given these two elicitation strategies, it is now clear the advantage of predicting the performance of a student in an individual topic rather than each question. As there are generally far fewer topics than questions, it would require significantly less user-effort to predict the performance in each topic.

For this project new-user estimation is achieved via an initial interview process (*Information Gain via Clustered Neighbours & Adaptive Bootstrapping using Decision Trees*), the choice to use this technique is due to the similarity between an examination process not to dissimilar to those seen in education. The interview strategy must therefore strike a balance between the number of questions asked and user model accuracy.

Note than an interview question in this problem domain requires students to answer questions that belong to a specific topic category. The individual questions within a topic category are randomly chosen from within a topic once the category of the next question is determined.

4.2 Information Gain via Clustered Neighbours

Information Gain via Clustered Neighbours (IGCN) [30] is an adaptive interview process that assumes the goal of building a new user model is dependant on finding the best user neighbourhood of existing users for each new user. The final IGCN

implementation is shown in algorithm 2.

Algorithm 2 IGCN Algorithm for PAL system

Smooth utility matrix U using eqn. 4.3
Partition U into C clusters/neighbourhoods
Compute information gain (IG) of each topic in U using eqn. 4.4

▷ The next stages occur when new user enters the system
▷ Conducting general interview process
repeat
    Ask the next question based on the list of topics ordered by IG score
    Update initial new user model (u_t)
until user has answered NPQ questions

▷ Conducting tailored interview process
repeat
    Find the best L clusters to u_t thus far using eqn.4.5
    Recompute IG based on the users in the L clusters
    Ask the next question based on the list of topics ordered by IG score
    Update u_t
until best clusters are constant or questions asked $\leq$ PQ

Determine final user model using eqn 4.6

4.2.1 Identifying Neighbourhoods

The first stage of IGCN involves identifying neighbourhoods of existing users, a useful algorithm in this case is K-means clustering. K-means aims to partition the data into K clusters, a cluster is a group of users that share similar traits [43], in a sense clusters can be regarded as neighbourhoods.

K-means Algorithm

The objective of this algorithm is to obtain K cluster centers, where each cluster center maximizes intra-cluster similarities and minimizes inter-cluster similarities between all users and their corresponding cluster center (figure 4.2).

$$
\begin{array}{c} \\ i \text{ Users} \end{array}
\begin{array}{c} j \text{ Topics} \\ \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1j} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{i1} & p_{i2} & p_{i3} & \cdots & p_{ij} \end{bmatrix} \end{array}
\rightarrow
\begin{array}{c} j \text{ Topics} \\ \begin{bmatrix} c_{11} & c_{12} & c_{13} & \cdots & c_{1j} \\ c_{21} & c_{22} & c_{23} & \cdots & c_{2j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{k1} & c_{k2} & c_{k3} & \cdots & c_{kj} \end{bmatrix} \end{array}
\begin{array}{c} \\ K \text{ Clusters} \end{array}
$$

Figure 4.2: K-means applied to Utility Matrix

K-means attempts to minimize the objective function:

$$\mathcal{O}^{KM} = \sum_{i=1}^{K} \sum_{j=1}^{U} \left\| \mathbf{x}_j^{(i)} - \mathbf{c}_i \right\|^2 \tag{4.2}$$

where $\left\| \mathbf{x}_j^{(i)} - \mathbf{c}_i \right\|$ is the distance between each user $\mathbf{x}_j$ and corresponding cluster center $\mathbf{c}$.

Although a variety of distance measures in the domain of collaborative filtering (CF) exist, including Pearson's Correlation and Cosine distance, research into clustering for CF has shown that euclidean distance often outperforms these traditional CF metrics [44][45].

Improved Neighbourhoods via Smoothing

In CF it is not uncommon for the utility matrix to be sparse, especially when the number of users & tasks within the system are numerous. Data sparsity can lead to K-means clustering identifying sub-optimal user-neighbourhoods, as a result of insufficient information describing users [46]. To ease the effect of sparsity on the utility matrix, the matrix is often smoothed before applying K-means clustering.

Smoothing is achieved by pre-populating unobserved values in the utility matrix to form $\hat{\mathbf{A}}$. As discussed by [47] dimensionality reduction can be applied to the matrix to effectively achieve smoothing. Smoothing starts by replacing values that have no recorded topic performance in the utility matrix $\mathbf{A}$, with the average performance of all users for that topic. Next singular value decomposition (SVD) is applied to the adjusted utility matrix.

$$\mathbf{A} = \mathbf{U} \times \mathbf{S} \times \mathbf{V}^T$$

After the matrix is decomposed it is reduced to a dimension d, this dimension can be determined via brute-force during experimental analysis.

$$\bar{\mathbf{U}} = \mathbf{U}_d \quad \bar{\mathbf{S}} = \mathbf{S}_d \quad \bar{\mathbf{V}} = \mathbf{V}_d$$

Finally smoothed utility matrix $\hat{\mathbf{A}}$ can be determined:

$$\hat{\mathbf{A}} = \bar{\mathbf{U}}\sqrt{\bar{\mathbf{S}}} \times \sqrt{\bar{\mathbf{S}}}\bar{\mathbf{V}}^T \tag{4.3}$$

This smoothed utility matrix is used with the regular k-means process discussed earlier to determine user neighbourhoods.

Determining Optimal Neighbourhoods

The problem still remains of how to determine the optimal number of cluster centres to use. A common solution is the elbow method, which looks at the amount of variance explained (ie. ratio between cluster variance and total variance) by adding an extra cluster. This is illustrated in figure 4.3, initially by increasing the number of clusters there is a significant increase in the variance explained, however as the number of clusters increases, this effect plateaus. The elbow criterion is the point where increasing the number of clusters does not yield significant accuracy advantages.

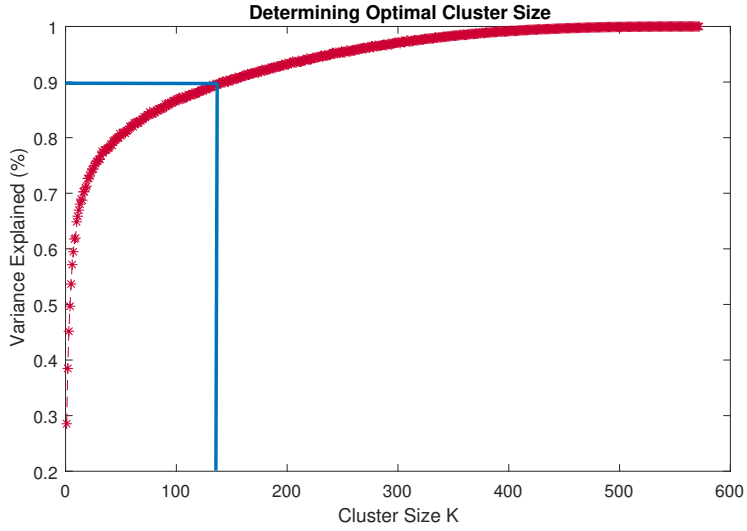


Figure 4.3: Variance Explained vs. Cluster Size

4.2.2 Interview Process via Information Gain

The second stage of IGCN computes the information gain (IG) of each topic at every stage in the interview process. The subsequent question comes from a topic that has the largest IG.

$$IG(a_{topic}) = H(C) - \sum_{p} \frac{|C^{p}_{a_{topic}}|}{|C|} H(C^{p}_{a_{topic}}) \tag{4.4}$$

As mentioned earlier, there are two stages to this algorithm. At the first stage IGCN builds a general user model using data from all users when calculating IG. At the second stage a customized model is built, with IG calculated using data from users in the most similar clusters to the target user.

To determine the most similar clusters to the target user, the auto correlation (4.5) between the target user model $\mathbf{u}_t$ and cluster centres $\mathbf{c}$ is calculated. The centres with the highest cross correlations are regarded as the best neighbourhoods.

$$s(\mathbf{u}_t, \mathbf{c}) = (\mathbf{u}_t * \mathbf{c})[0] = \sum_{m=-\infty}^{\infty} u_t[m]c[m] \tag{4.5}$$

4.2.3 Determining New User Topic Model

Once the interview process concludes the new user topic model resembles something similar to figure 4.4, moreover with each new user model there will an accompanying neighbourhood of users. The final step of this implementation involves predicting a final new user model by combining the two data sources.

$$\begin{bmatrix} 1 & 0.5 & 0.33 & 0 & \dots & 0.75 \end{bmatrix}$$

Figure 4.4: Initial New User Topic Model

The goal of finding one best user cluster (neighbourhood) for a new user may be insufficient, as a new user may share traits with users across multiple clusters. For this reason to build the final new user model, a comparison between the initial user model $\mathbf{u}_t$ and all users in the best neighbourhoods of $\mathbf{u}_t$ is made. The users with a similarity (eqn. 4.5) greater than some threshold form a final user neighbourhood, N_f. The final estimation of a new user topic model $\mathbf{u}$ is then given by:

$$\mathbf{u} = \frac{\sum_{i \in N_f} \mathbf{u}_i s(\mathbf{u}_t, \mathbf{u}_i)}{\sum_{i \in N_f} s(\mathbf{u}_t, \mathbf{u}_i)} \tag{4.6}$$

4.2.4 Determining New User Question Model

Recall that the choice to determine a topic model over a question model was motivated by user-effort, the advantage of using interview methods which identify optimal user-neighbourhoods enables a question model to be determined indirectly. By aggregating the performance of each user in N_f for each question in a question model, one can determine an estimation for a new-user question model.

4.3 Adaptive Bootstrapping using Decision Trees

Decision trees (DT) have been previously used in the field of recommendation systems to bootstrap a new user model [24][33] as they very clearly resemble an adaptive interview process. Decision tree construction occurs prior to a new user entering a system, as a result the topics of interview questions are predetermined, however the interview process is dynamic as this list of topics is refined by the response to each question given by a new user.

Each node in a decision tree corresponds to a single topic that a question is based upon, depending on the new user's response to this question they are directed to

a resulting sub-tree. The objective of the interview process is to direct a new user from a root node to a leaf node, at which point they have hopefully reached the best possible classification.

Each tree node represents a set of existing users within the system, as tree depth increases each node progressively refines this set of existing users. An example of a decision tree specific to this implementation is shown in figure 4.5, each node represents a topic with associated user set U. The DT in this specific problem is binary, such that a response of a new user at each node can be correct or incorrect.

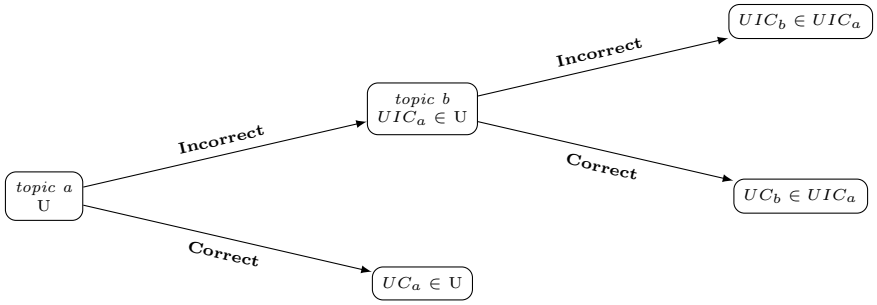


Figure 4.5: Example of Decision Tree Structure

4.3.1 Constructing the Decision Tree

As with all CF mechanisms a DT is built using performance data from existing users in a utility matrix. Each node represents a set of existing users, the prediction for performance in a topic can be obtained by taking the average performance within this topic for all users at this node. Construction of the DT then begins at the root node, which encompasses all users in the system. At each node the best splitting topic is chosen, that is the topic that best partitions users into two sub-trees (correct & incorrect), which minimizes the overall prediction error (4.8). Note that as the performance results in each topic range from $0\% - 100\%$, users with a performance $p \geq 50\%$ are mapped as correct and $p < 50\%$ as incorrect.

More formally for a tree node n and topic s, the splitter is defined as:

$$splitter(n) = \arg\min_{s} E_n(s) \tag{4.7}$$

$$\begin{aligned} E_n(s) &= e^2(L_n(s)) + e^2(R_n(s)) \\ e^2(D) &= \sum_{t \in T} \sum_{i \in D \cap t} (p_{it} - \frac{\sum_{i \in D \cap t} p_{it}}{|D \cap t|})^2 \end{aligned} \tag{4.8}$$

where $L_n(s)$ & $R_n(s)$ defines the subset of users at node n that have a correct performance score or incorrect performance score in topic s respectively. p_{it} is the performance by user i in topic t The algorithm to generate a single decision tree can

be found in algorithm 3.

Algorithm 3 Decision Tree Algorithm

function GROWDECISIONTREE(n)
▷ All nodes are associated with a user set D

    splitter = splitFunction(D)
    if not ready to terminate **then**
        Initialise node $L = L_n(splitter)$
        Initialise node $R = R_n(splitter)$
        growDecisionTree(L);
        growDecisionTree(R);
    end if
end function

function SPLITFUNCTION(D)
▷ Loop over all possible splitters
    for all s **do**
▷ Loop over all users in D that have a performance score for s
        for all $u \in D \cap s$ **do**
            $u \leftarrow \begin{cases} L_n(s) & p_{us} > 0.5 \\ R_n(s) & p_{us} \leq 0.5 \end{cases}$ ▷ Assign u to either $L_n(s)$ or $R_n(s)$
        end for
        Calculate $Err_n(s)$ using eqn.(4.8)
    end for
return $splitter = \arg\min Err_n(s)$
end function

The DT construction is terminated by two factors:

1. **Tree Depth**: As the tree grows larger the users in each subset progressively reduce. By increasing the depth, we increase the complexity of the tree, which leads to over-fitting of the training data, over-fitting generally results in poor prediction performance of a decision tree.

2. **Error Rate**: $\min_s Err_n(s) \geq e^2(n)$, that is the error in prediction accuracy once splitting of users at node n is greater that the error before the split. This translates to the split yielding no accuracy benefit in determining an accurate user classification.

4.3.2 Enhancement via Decision Forests

Decision forests (DF) are regarded as one of the best classification algorithms, able to classify data with a high degree of accuracy. DF are ensemble learning methods that draw parallels with nearest neighbour algorithms [48] making them suitable solutions in CF.

Decision trees are regarded as weak learners, such that they are a type of classifier that produce a result that is only slightly correlated with the true classification of data [49]. Decision forests build on the notion that a group of weak learners can be combined to form a strong learner. By employing a decision forest method to construct the interview process the problem high bias often associated with single decision tree can be mitigated, to produce a more accurate classification for a new user.

A potential avenue that was explored to generate an ensemble of trees was Bootstrap aggregating (Bagging) [50], this method involves training multiple decision trees using randomly sampled subsets of the training data. It became evident that in the context of a RS, utilizing the entire utility matrix would yield better performance results.

Multiple decision trees are generated by randomizing the choice of splitting topic. This process employs a roulette selection algorithm such that at each node n, the splitting topic s has a probability proportional to $E_n(s)^{-\alpha}$ of being selected, alpha determines the spread of E_n.

4.3.3 Determining New User Model

The general principle for building a final new user topic model is similar to the method discussed in subsection 4.2.3. Every new user to the system is classified by a leaf node in the DT, in the case of DF this becomes multiple leaf nodes. Each leaf node has a subset of users associated with it, thus the user neighbourhood is equivalent to a leaf node(s).

Chapter 5

Task Recommendation Module

5.1 Problem Formulation

Vygotsky's, Zone of Proximate Development [38] states that in an educational environment, when providing a student with learning tasks, the difficulty of the task should be slightly higher than the student's current competency level. This notion is used to determine the most appropriate tasks to recommend to a student.

A task in the PAL system is structured as follows:

$$topic \supseteq question$$

This task classification results in two stages of recommendation that is necessary to determine the next task. First a topic category for the next task must be provided and then a more specific question from within that task[1]. The student model plays an important role in determining the next task as mentioned earlier. The student model in the proposed PAL system consists of two entities: the topic model and question model.

The remainder of this section details how task recommendation in the PAL system could be achieved, discussing specifically how to choose the most suitable topic and question.

5.2 Providing Topic Recommendation

Topic recommendation is determined by the topic model (fig. 5.1), this model represents the average performance for a single student across each topic.

$$\begin{matrix} & & & & & (1..t)\text{ Topics} & \\ \big[0 & 0.7 & 0.1 & \dots & p_{t-1} & p_t \big] \end{matrix}$$

Figure 5.1: Topic Model Example

[1] In the context of a quiz game, the next task is a quiz problem.

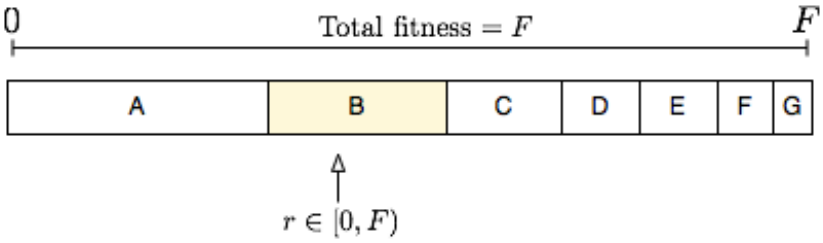


Figure 5.2: Fitness Proportionate Selection

According to the Zone of Proximate Development the choice for the next topic should be the topic that has a difficulty level slightly higher than the students current competency level. For this specific problem this translates to selecting a topic that has a performance score slightly below the average performance of a student across all topics. What follows is a discussion of possible action selection techniques to determine the next topic category.

5.2.1 Fitness Proportionate Selector

Selecting the next topic in a deterministic manner could lead to unfavourable effects within the system. Most importantly it does not introduce variability in the learning strategies, which may force a student to complete a number of questions in a single topic before being able to move onto another topic.

A better suited topic selection strategy would allow all topics to be chosen, but the more appropriate suggestions (ie. those slightly lower than the users current average performance level) have a greater probability of selection. Fitness proportionate selection (FPS) is an algorithm that can achieve this desired outcome[2].

The algorithm (4) works by assigning a fitness value to each topic, where topics that are closer to the desired difficulty level receive a higher weighting. A topic then has the probability proportional to the fitness score of being chosen (fig. 5.2). FPS highlights the exploitation-exploration trade off, while we could make the best possible decision for topic selection using information available to us, it is important to realize that by introducing a random factor it allows the system to learn further about a students ability.

5.2.2 Temporal Fitness Proportionate Selector

The problem with using the average student performance as in the topic model to provide task selection is highlighted by figure 5.3. The graph depicts a typical student's performance aggregated over each day from within a single topic.

Thus far it is assumed that student ability remains constant across time, it is evident from fig. 5.3 that this is not the case. In an educational setting it is safe to assume that a trend exists between a students past learning and current performance. Figure

[2]https:en.wikipedia.orgwikiFitness_proportionate_selection

Algorithm 4 Fitness proportionate selection

```
1: function FPS(topic_model)
   ▷ topic_model has K topics with values indicating a student performance in that topic
   ▷ Determine fitness weights
2:     mean = avg(topic_model)
3:     for t ∈ topic_model do
4:         Fitness(t) = 1/|(topic_model(t)−mean)|
5:     end for
6:     F = sum(Fitness)
7:     Start ← RandUniform() × F
8:     for t ∈ topic_model do
9:         Start = Start - Fitness(t)
10:        if Start ≤ 0 then
11:            return t
12:        end if
13:    end for
14: end function
```

5.4 shows the typical sequential trend.

To account for the temporal effects in student performance, logistic regression can be applied to model an individual student's performance overtime in each topic. Using this model the probability that a student will perform the next question correctly in each topic can be estimated. This performance estimation can be used to provide a more accurate representation of student ability at the current time in each topic. Fitness proportionate selection is then applied on the adjusted student topic performance model to determine the next topic to recommend.

Logistic Regression

Logistic regression allows one to predict the probability of a discrete variable (ie. probability a student answers the next question correctly or incorrectly) using a linear function [51]:

$$y = h_\theta(x) = \theta^T \mathbf{x} \tag{5.1}$$

where the outcome y is determined using the regressor x that best describes the outcome. An equation in this form is unbounded and therefore unsuitable for predicting a probabilistic outcome. Logistic regression maps $y \in \{0, 1\}$ using a sigmoid function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{5.2}$$

The outcome now takes the form:

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}} \tag{5.3}$$

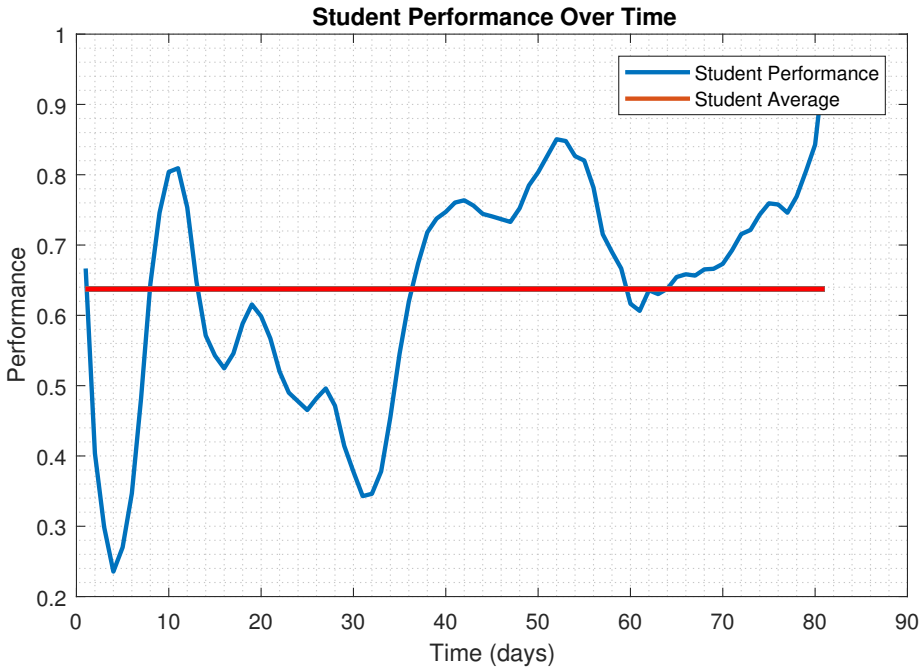


Figure 5.3: Student Performance vs Average Performance over time

In short the aim of logistic regression is to determine the coefficients θ, that best fit the observed outcomes and their corresponding features. For this work $\theta\mathbf{x}$ is a function of time t. Intuitively the next outcome of a question would also be dependant on the N most previous performance outcomes, recall that performance of a question is indicated by a 0 (incorrect) or 1 (correct). These performance outcomes are referred to as lag terms (x_i). To determine the optimal number of lag terms one must observe the time series data of each student, calculating the auto-correlation of the time-series will yield the number of optimal value number of terms to use.

$$\theta^T \mathbf{x} = \theta_0 + \theta_1 t + \sum_{i=2}^{N} \theta_i x_i \tag{5.4}$$

Coefficients are determined using maximum likelihood estimation, this is achieved using gradient ascent in this work. The objective function that is maximized:

$$l(\theta) = \sum_{i=1} y_i \log(h_\theta(x^i)) + (1 - y_i) \log(1 - h_\theta(x^i)) \tag{5.5}$$

Gradient ascent is an algorithm, which iteratively optimizes the parameter θ, by taking the differential of $l(\theta)$ with respect to each θ_j term (eqn. 5.6). The differential gives an indication of the direction to update the values of θ.

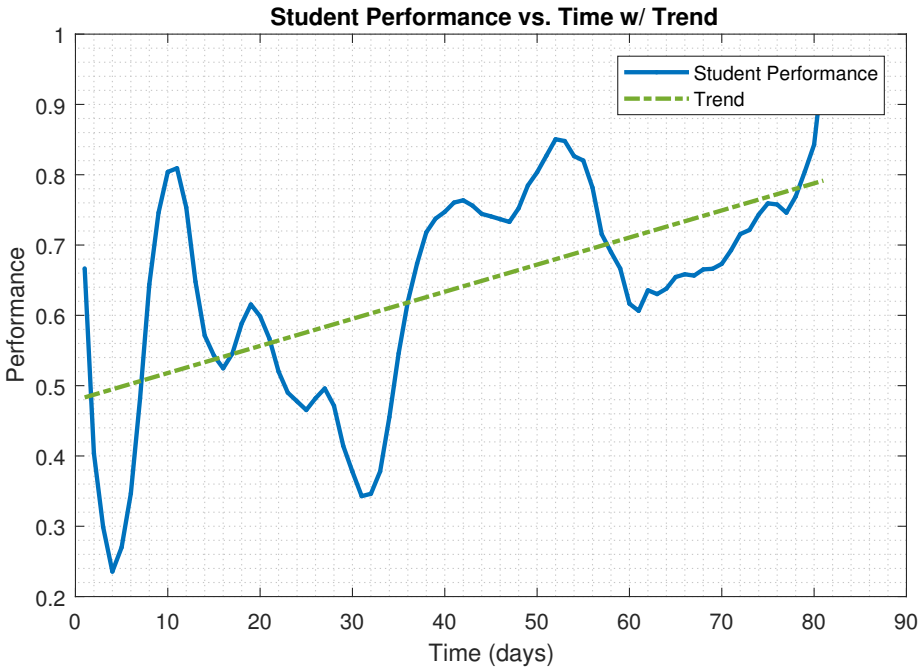


Figure 5.4: Student Performance Performance over time w/ Trend Line

$$\frac{\partial l(\theta)}{\partial \theta_j} = (y - h_\theta(x))x_j \tag{5.6}$$

Algorithm 5 details the implementation used in this work to determine the coefficients of logistic regression.

5.3 Providing Question Recommendation

Once an appropriate topic has been selected, a question within this topic must be provided. The problem in the PAL system is that there no explicit data indicating the difficulty of each question nor do we have the performance of each student in every question. As a result collaborative filtering (CF) techniques can be utilized to derive relationships between students and questions. It is hoped that by utilizing these relationships the performance of every student in each question can be predicted and thus questions that students do not find too easy/difficult can be recommended.

Recall that the three factors that are important to consider when predicting student performance, **Temporal effects**, **Bias** and **Slip & Guess factors**.

There is a question model associated with each topic in the PAL system, it describes the performance for each question answered by every student. The performance metric for each question is denoted by a 0 (incorrect) or 1 (correct). At this point it is important to discuss the significant difference between traditional CF and educational CF. Typically CF assume that users rate an item once and this preference describes

Algorithm 5 Learning Coefficients for Logistic Regression

function LOGISTIC REGRESSION(D_{train},α, $StoppingCond$)
▷ D_{train} contains $(Y, t, X_2..X_N)$
  Initialize $coef \leftarrow rand(N+1)$
  while $StoppingCond$ **do**
    for $i \in D_{train}$ **do**
      $\hat{y} = coef[0]$
      for j = 1..N **do**
        $\hat{y} = \hat{y} + coef[j] * X[j]$
      end for
      $\hat{y} = \frac{1}{1+e^{-\hat{y}}}$
      $error = Y[i] - \hat{y}$
      $coef[0] = coef[0] + \alpha(Y(i) - h_\theta(X))$
      for j = 1..N **do**
        $coef[j] = coef[0] + \frac{\partial l(\theta)}{\partial \theta_j}$
      end for
    end for
  end while
return $coef$
end function

a user across time, this is not the case in educational environments where it is often encouraged to repeat a problem for a better grasp on the concepts being taught. For this reason the temporal effect of students performing problems multiple times must be considered. To this end context aware recommendation systems (CARS) are introduced.

5.3.1 Context Aware Recommendation Systems

CARS take context (eg. time) into account when providing recommendations, such that the utility matrix of performance is shown in figure 5.5. A users performance is now modelled as:

$$P : User \times Task \times Context \mapsto Performance$$

The context in this problem domain, is the number of attempts a student has made on a particular task, this representation of context was shown to yield good accuracy in [35]. Now at each attempt the performance is denoted as $p_{(u,t,c)} \in \{0, 1\}$.

Adomavicius et al.[16] detail the most common approaches to contextual recommendation systems:

- **Contextual Pre-filtering**: The context in this case is used to filter the data specific to the context. After filtering occurs typical recommendation techniques can be implemented as normal.

- **Contextual Modelling**: Rather than manipulating the data prior to use with the RS, the utility matrix is treated as is. A model is trained to represent the 3-D utility matrix, similar to the latent factor methods described earlier.

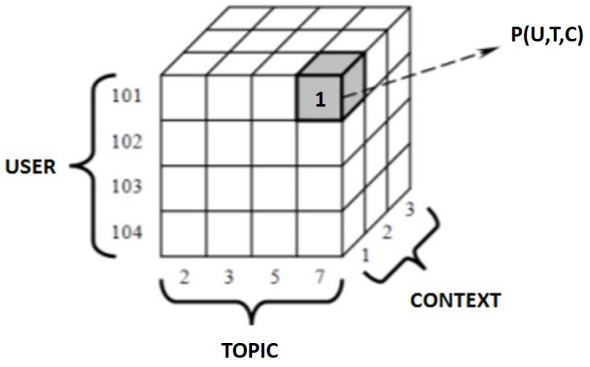


Figure 5.5: Context Aware Utility Matrix

For its simplicity, this project adopts a pre-filtering based approach. The most appropriate pre-filter would be to average a students performance across multiple repetitions of a question. Such that for a specific observation (u, t, c, p), the pre-filtered observation becomes $(u, t, \bar{p})$.

An example of the question utility matrix available for each topic is shown in figure 5.6, where $\bar{p}_{uq} \in [0, 1]$ is the observed performance of a user u for question q across a particular context. The predicted performance $\hat{p}_{uq}$ of unobserved questions can be regarded as the probability that a student answers a question correctly. Given this perspective the next question that will be suggested should have $\hat{p} \approx 0.5$, such that the student will find this question neither too difficult or too easy. To ensure that question recommendation does not become stale as in topic recommendation, FPS is again utilized where the questions with performance centered around a difficulty of 0.5 have a greater probability of selection.

$$
\begin{array}{c}
q \text{ Questions} \\
\begin{bmatrix}
\bar{p}_{11} & \bar{p}_{12} & \bar{p}_{13} & \cdots & \bar{p}_{1q} \\
\bar{p}_{21} & \bar{p}_{22} & \bar{p}_{23} & \cdots & \bar{p}_{2q} \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
\bar{p}_{u1} & \bar{p}_{u2} & \bar{p}_{u3} & \cdots & \bar{p}_{uq}
\end{bmatrix}
\end{array}
\; u \text{ users}
$$

Figure 5.6: Question Utility Matrix for Individual Topics

In the remainder of this section we provide possible solutions to predicting the performance of each student for a question given this updated utility matrix.

5.3.2 Prediction using Neighbourhood Models

A typical CF approach relies on a neighbourhood of users being found as discussed in section 2.4.1, a neighbourhood is determined by calculating similarity between the target user u_t and other users within the system.

$$s(u, u_t) = \frac{\sum_{q \epsilon Q_u \cap Q_{u_t}} (p_{u,q} - \overline{p_u})(p_{u_t,q} - \overline{p_{u_t}})}{\sqrt{\sum_{q \epsilon Q_u \cap Q_{u_t}} (p_{u,q} - \overline{p_u})^2} \sqrt{\sum_{q \epsilon Q_u \cap Q_{u_t}} (p_{u_t,q} - \overline{p_{u_t}})^2}} \tag{5.7}$$

where $Q_u \cap Q_{u_t}$ is the set of questions that have observed performances by user u and the target user u_t. $\overline{p_u}$ is the average observed performance of user u.

A prediction for a question q can then be made using:

$$\hat{p}_{u_t,q} = \frac{\sum_{u \in N} s(u, u_t)(p_{u,q})}{\sum_{u \in N} |s(u, u_t)|} \tag{5.8}$$

where N is the neighbourhood of most similar users. To determine the size of N we threshold the similarity function such that the only users which are chosenm=, are those that have a similarity above a certain threshold. The threshold is determined by:

$$\alpha \times \max s(u, u_t), \quad 0 < \alpha \leq 1$$

Accounting for Bias

Typical recommendation systems observe two phenomena: user-bias, the systematic tendencies for a user to give higher ratings than others; item-bias, the tendency for an item to receive higher ratings that others.[39]

In the context of education this translates to student-bias b_u, the ability of a student compared to other students and task-bias b_t, the difficulty of a task compared to other tasks. According to [39] the student and task bias can be modelled as:

$$b_{u,t} = b_u + b_t + \mu \tag{5.9}$$

where b_u and b_t indicate the deviations of observed performance of u and i respectively, from the average. The mathematical formulas for μ, b_u and b_t can be found in (5.21), (5.22), (5.23) respectively.

The similarity function does not take the student/task bias into account but instead attempts to identify users with similar trends of data[3]. Simply taking the weighted average as in eqn. 5.8 would yield sub optimal accuracy. Adjusting eqn. 5.8 to account for bias as in eqn. 5.10 should provide a more accurate prediction.

[3]This observation is a result of the similarity function being correlation measure

$$\hat{p}_{(u_t,q)} = b_{(u_t,q)} + \frac{\sum_{u \in N} s(u, u_t)(p_{u,q} - b_{u,q})}{\sum_{u \in N} |s(u, u_t)|} \tag{5.10}$$

5.3.3 Prediction using Latent Factor Analysis

Latent factor models as discussed earlier, discover latent features in the data which help model underlying characteristics of student and task interactions. For this work we propose using the most common technique for latent factor analysis, matrix factorization.

Matrix Factorization (MF)

Matrix factorization attempts to represents the utility matrix $\mathbf{A} \in \mathbb{R}^{|U| \times |Q|}$ by two smaller matrices $\mathbf{R} \in \mathbb{R}^{|U| \times |K|}$& $\mathbf{S} \in \mathbb{R}^{|Q| \times |K|}$, where K are the number of latent factors chosen to represent the data. The value of K can be determined by brute force.

An approximation of the utility matrix is given by:

$$\mathbf{A} \approx \mathbf{R} \times \mathbf{S}^T = \hat{\mathbf{A}} \tag{5.11}$$

Finally to generate performance predictions $\hat{p}_{u,q}$ for user u in a specific question q, the dot product between the factor vector representing the user and factor vector representing the question is taken.

$$\hat{p}_{u,q} = \mathbf{r}_u^T \mathbf{s}_q = \sum_{j=1}^{K} r_{uj} s_{jq} \tag{5.12}$$

Determining MF Model

While the prediction stage is straightforward, we must first determine optimal values for the matrices $\mathbf{R}$ and $\mathbf{S}$. For this project Stochastic Gradient Descent (SGD) is used to train the CF model [52].

This method begins by initializing $\mathbf{R}$ and $\mathbf{S}$ as randomly generated matrices, we can then determine a prediction for the utility matrix $\mathbf{A}$ using eqn. 5.11. The goal would be to iteratively optimize the values of $\mathbf{R}$ and $\mathbf{S}$, such that error between $\mathbf{A}$ and $\hat{\mathbf{A}}$ is minimized.

The error between the estimated performance (eqn. 5.12) and actual performance for a user-question pair (u,q) is given by:

$$e_{u,q}^2 = (p_{u,q} - \hat{p_{u,q}})^2 = (p_{u,q} - \sum_{j=1}^{K} r_{uj} s_{jq})^2 \tag{5.13}$$

To minimize the error, we take the differential of the error with respect to r_{uj} & s_{jq} separately, this gives us an indication of the direction to change each value.

$$\begin{aligned}\frac{\partial e_{u,q}^2}{\partial r_{uj}} &= -2(p_{u,q} - \hat{p_{u,q}})(s_{jq}) = -2(e_{u,q})(s_{jq}) \\ \frac{\partial e_{u,q}^2}{\partial s_{jq}} &= -2(p_{u,q} - \hat{p_{u,q}})(r_{uj}) = -2(e_{u,q})(r_{uj})\end{aligned} \tag{5.14}$$

We can then update the values of r_{uj} & s_{jq} accordingly, where α determines the rate at which we incrementally approach the minimum:

$$\begin{aligned}r'_{uj} &= r_{uj} + \alpha\frac{\partial e_{u,q}^2}{\partial r_{uj}} = r_{uj} + 2\alpha(e_{u,q})(s_{jq}) \\ s'_{jq} &= s_{jq} + \alpha\frac{\partial e_{u,q}^2}{\partial s_{jq}} = s_{jq} + 2\alpha(e_{u,q})(r_{uj})\end{aligned} \tag{5.15}$$

Evidently this iterative refinement of the two matrices will be dependant on the values in the training data-set D_{train} as we are comparing observed ratings against predicted. The overall error is therefore determined by:

$$E^{MF} = \sum_{(u,q)\in D_{train}} (p_{u,q} - \hat{p_{u,q}})^2 \tag{5.16}$$

Consequently the choice to terminate the operation of this algorithm is dependant on whether we achieve a maximum number of iterations or the error falls below a defined rate.

Regularization

Regularization is a common technique used to reduce the effect of over-fitting the training data to parameters **R** and **S** [23]. After applying regularization the new error function becomes:

$$e_{u,q}^2 = (p_{u,q} - \sum_{j=1}^{K} r_{uj}s_{jq})^2 + \beta(\|R\|^2 + \|S\|^2) \tag{5.17}$$

where $\|X\|$ is the Frobenius norm. The β term is used to control the magnitude of the factorization matrices, essentially it means that we get good approximations for **R** & **S** without using large data specific values to represent the matrices.

The updated values of r_{uj} & s_{jq} now become:

$$\begin{aligned}r'_{uj} &= r_{uj} + 2\alpha(e_{u,q}s_{jq} - \beta r_{uj}) \\ s'_{jq} &= s_{jq} + 2\alpha(e_{u,q}r_{uj} - \beta s_{jq})\end{aligned} \tag{5.18}$$

Accounting for Bias

The added benefit of using matrix factorization techniques is that it allows us to incorporate bias into the model easily. As described by [23]:

$$\hat{p}_{u,q} = b_{u,q} + \sum_{j=1}^{K} r_{uj} s_{jq} \tag{5.19}$$

where $b_{u,q}$ is the combination of task bias and student bias:

$$b_{u,q} = \mu + b_u + b_q \tag{5.20}$$

μ is the global average:

$$\mu = \frac{\sum_{(u',q) \in D_{train}} p_{u,q}}{|D_{train}|} \tag{5.21}$$

The task and student bias:

$$b_u = \frac{\sum_{(u',q) \in D_{train} | u'=u} (p_{u',q} - \mu)}{|\{(u',q) \in D_{train} | u' = u\}|} \tag{5.22}$$

$$b_q = \frac{\sum_{(u,q') \in D_{train} | q'=q} (p_{u,q'} - \mu)}{|\{(u,q') \in D_{train} | q' = q\}|} \tag{5.23}$$

The updated error function for SGD:

$$e_{u,q}^2 = (p_{u,q} - b_{u,q} - \sum_{j=1}^{K} r_{uj} s_{jq})^2 + \beta(\|R\|^2 + \|S\|^2 + b_u^2 + b_q^2) \tag{5.24}$$

Finally the algorithm to determine the latent factor matrices **R** and **S**, is shown in algorithm 6. The performance prediction for a question can then be made using eqn.5.12.

There are two advantages of using latent factor models over typical neighbourhood models:

1. The heavy computation of latent factors occurs offline, the overhead of computing a prediction in real time is significantly lower than that of neighbourhood models especially when a number of users are using the system. The benefit of this method to our system is that we can provide recommendations for questions almost instantaneously in real time.

2. Latent factor models implicitly model slip and guess factors, where in neighbourhood models there is no defined way to account for this [53].

Algorithm 6 Learning Latent Factor Models

function LatentFactorModels(D_{train}, α, β, K)
  Initialize $\mathbf{R}$ and $\mathbf{S} \leftarrow$ rand()
  Determine μ using eqn.5.21
  Determine Student-Bias for each student using eqn.5.22
  Determine Task-Bias for each question using eqn.5.23

  while $(iter < maxiter)$ **or** $(E^{MF}_{iter-1} - E^{MF}_{iter}) \leq thresh$ **do**
    $D'_{train} \leftarrow$ Randomize D_{train} order
    for $(u, q) \in D'_{train}$ **do**
      Set $\hat{p}_{u,q}$ equal to eqn.5.19
      $e_{u,q} = p_{u,q} - \hat{p}_{u,q}$
      $\mu \leftarrow \mu + \alpha \times e_{u,q}$
      $b_u \leftarrow b_u + \alpha \times (e_{u,q} - \beta b_u)$
      $b_q \leftarrow b_q + \alpha \times (e_{u,q} - \beta b_q)$

      for $j = 1 : K$ **do**
        $r_{uj} \leftarrow r_{uj} + 2\alpha(e_{u,q} s_{jq} - \beta r_{uj})$
        $s_{jq} \leftarrow s_{jq} + 2\alpha(e_{u,q} r_{uj} - \beta s_{jq})$
      end for
    end for

    $E^{MF} \leftarrow 0$
    for $(u, q) \in D'_{train}$ **do**
      $E^{MF} \leftarrow E^{MF} + \mu + b_u + b_q + \sum_{j=1}^{K} r_{uj} s_{jq}$
    end for
  end while
return μ, $\mathbf{R}$, $\mathbf{S}$, b_u, b_q
end function

Chapter 6

Testing and Results

This chapter details the experiments conducted on the new-user estimation module and task recommendation module. We begin by outlining the use of different datasets before moving on to the testing methodology, design options and baseline measures used to evaluate each module, we conclude by discussing the results obtained from experiments.

6.1 Datasets

The testing of the proposed new-user profiling module and task recommendation module occurs on the following data-sets, KDD Cup Algebra data-set [54] and data gathered from the current PAL application. Henceforth the two datasets will be denoted as **KDD** & **PAL**.

6.1.1 KDD Data-set

The KDD data is gathered from a mathematics based ITS, it contains data from 575 students (aged 9-14 years) and 813,661 individual problems [54]. The structure of each problem within KDD is as follows:

$$ProblemHierarchy \supseteq ProblemName \supseteq StepName$$

A step refers to an individual activity in a problem and a collection of steps is a single problem. For the purposes of this work, we consider the problem hierarchy to represents the general overarching topic. The problem name indicates the specific question that constitutes a single topic. The individual steps are aggregated such that if a student answers incorrectly in a single step they also answer the problem incorrectly.

Additionally the KDD data-set (fig. 6.1) tracks the number of times a student has seen a problem as indicated by the problem view. Supporting time information (problem duration, problem start time etc.) is also provided but has not been utilized for all aspects of this work. Finally the field, correct first attempt signals whether a student answered a problem correctly (indicated by either a 1 or 0).

Student Id	Problem Hierarchy	Problem Name	Problem View	Step Name	Correct First Attempt
075kbY53c1	Unit ES_07, Section ES_07-2	LIT53A	1	INNERv2	1
075kbY53c1	Unit ES_07, Section ES_07-4	LIT66A	1	INNERx2	0
075kbY53c1	Unit ES_07, Section ES_07-4	LIT68A	1	h1R1	1
075kbY53c1	Unit ES_07, Section ES_07-4	LIT68A	2	h1R1	1
0GlR30c2Mt	Unit ES_07, Section ES_07-2	LIT53A	1	INNERv2	0
0GlR30c2Mt	Unit ES_07, Section ES_07-4	LIT66A	1	INNERx2	1
0GlR30c2Mt	Unit ES_07, Section ES_07-4	LIT66A	2	INNERx2	1

Figure 6.1: Snapshot of KDD dataset

Educational Collaborative Filters (ECF) assume that users with similar performance in some topics will share similar performance in other topics. The KDD data-set in our opinion is a suitable test data-set for this project, while it does not represent diabetic children it does map well for the problem we aim to solve. KDD represents educational data from children ages 9 to 14 years old, similarly the PAL project supports children from ages 7 to 14. We are able to obtain trends in learning for children and if our ECF assumption holds, we are able to determine possible outcomes.

6.1.2 PAL Dataset

This data has been collected from the current PAL application, as such it is already in a form suitable for collaborative filtering. the data-set is considerably smaller than **KDD** with 17 users and 9 topics, it is a complete data-set with all users having recorded performance scores for each activity (ie. sparsity is 0%). A snapshot of this data is shown in figure 6.2.

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9
User 1	0.875	0.5	0.846154	0.7	1	0.914286	1	0.75	0.583333
User 2	1	1	0.75	0.791667	0.7	0.84	0.75	0.5	0.896552
User 3	0.666667	1	0.857143	0.733333	0.666667	0.777778	1	1	1
User 4	0.833333	0.5	0.942857	0.884615	0.818182	0.896552	1	0.9	0.842105
User 5	0.8	0.5	0.777778	0.933333	1	0.880952	1	0.888889	0.769231
User 6	1	0	0.869565	0.8	0.75	0.75	0.6	0.5	0.875

Figure 6.2: Snapshot of PAL dataset

6.2 New User Model Estimation

6.2.1 Experimental Setting

The testing of this module occurs on the KDD and PAL data-sets. Recall that the new user model estimation shall focus on predicting the performance of a student for each topic rather than a single question. As such we can map the data in KDD to a topic utility matrix (fig.4.1a) where:

$$\begin{aligned} Student\,Id\,(S) &\mapsto User \\ Problem\,Hierarchy\,(T) &\mapsto Topic \\ Avg.Correct \in S \times T &\mapsto Performance \end{aligned}$$

This transformation of data results in a utility matrix with 574 users and 138 topics. Data sparsity of this data-set is 79.4%.

6.2.2 Testing Methodology

Testing is performed using k-fold cross-validation, in summary the sample data is partitioned into k equal subsamples, of the K samples a single partition becomes the test data (D_{test}) and the remaining k-1 samples is used as training data (D_{train}). The cross-validation process repeats k times with each sample being used a test set. Each sample in the test data acts as a new user to the PAL system, with the training data being used to determine the interview process structure.

Two separate criteria shall be used to deduce the effectiveness of the proposed system.

1. User Effort: This criteria shall be governed by the number of questions proposed to each user, ideally the upper limit to this metric is 20 questions.
2. Model Accuracy: As with most CF evaluation, this criteria shall be determined by RMSE (6.1), that is the error between the new user model prediction $\hat{p}$ and the actual new user model p.

$$RMSE = \sqrt{\frac{\sum^{D_{test}}(p - \hat{p})^2}{|D_{test}|}} \tag{6.1}$$

Please note that for interview based solutions, we consider the actual new user model as a probability that they answer a problem correctly, for example if a user has performance score 0.65 in a topic they have a 65% chance of getting the next question in the interview process correct. As this determination of correctness is uniform random we repeat the experiments a further n times, thus the result across k cross-validation observations are then averaged over n to produce a single result.

6.2.3 Baseline Comparison

For the sake of comparison the proposed solutions are compared with baselines, *global average* & *biased student-task*.

Global average assigns the user model estimation as the average performance of the training data, formally given as:

$$\mu = \frac{\sum^{p \in D_{train}} p}{|D_{train}|} \tag{6.2}$$

Biased user-task is a baseline estimator suggested in [55], where the performance of each student-task pair is predicted by eqn. 6.3. The significance of biased predictors was explained earlier in this work. As the user model is non-existent in the cold

start case $b_u = 0$, b_t is given by eqn. (6.4).

$$\hat{p_{st}} = \mu + b_t + b_u \tag{6.3}$$

$$b_t = \frac{\sum_{\{(s,t',p) \in D_{train} | t'=t\}} (p_{s,t'} - \mu)}{|\{(s,t') \in D_{train} | t' = t\}|} \tag{6.4}$$

Moreover the proposed solutions are also compared against a static interview method, these are interviews which do not adapt as the interview progresses.

Popularity: The static method uses popularity to generate an initial list of topics, which questions are based upon. The list of topics are ranked based on the number of students who have performed a question within this topic.

Randomize: Question topics are chosen at random as the interview progresses, this method is representative of the solution in the current PAL system.

Note that final user models are built in the same manner as described in subsection 4.2.3, but with all current users becoming the target user neighbourhood.

6.2.4 Experimental Results

Performance criteria of proposed solutions are *user effort* and *model accuracy*, we first observe the effects of the solutions on accuracy, and then we discuss the effect of user effort on accuracy.

Model Accuracy

Table 6.1 shows the accuracy (RMSE) of the baseline and interview based methods on the **KDD** & **PAL** datasets. Naturally for interview based methods the accuracy increases with the number of questions asked, thus we fix the number of questions to 10 for **KDD** and 5 for **PAL** to give a fair representation of accuracy. Finally as in IGCN we smooth the data used to generate the DF in Adaptive Bootsrapping.

We observe that on the **KDD** data set Adaptive Bootstrapping using Decision Forests performs significantly better that other implementations after asking 10 questions. Moreover the benefit of using decision forests (DF) over a single decision tree is highlighted. The increased accuracy can be attributed to DF ability to mitigate higher variance and bias in results, often associated with single trees. Further to this by smoothing the data used to construct the DT, we observe a slight increase in accuracy.

The static based interview implementations (*Popularity* & *Random*) have the highest error compared to baseline and proposed implementations. This low accuracy is a result of constraining the maximum of interview questions, as will be shown later this accuracy considerably improves as the number of questions asked increases.

Method	KDD	PAL
Global Average	0.328	0.200
Biased User-Task	0.314	0.198
Popularity	0.338	0.230
Random	0.342	0.227
IGCN	0.340	0.229
IGCN w/ Smoothing	0.289	N/A
Adaptive Bootstrapping DT	0.294	0.237
Adaptive Bootstrapping DF	0.254	0.203
Adaptive Bootstrapping Smoothed	0.252	N/A

Table 6.1: RMSE Comparison of Solutions for KDD & PAL

Finally, we see that performance of IGCN increases as smoothing is applied. The **KDD** dataset is almost 80% sparse, decreasing the ability of k-means clustering to find accurate user-neighbourhoods, smoothing the data-set has helped to alleviate this problem, evident by the increased performance.

From results obtained for the **PAL** data-set we conclude them to be inclusive, as each solution failed to perform better than baseline methods. This conclusion could be put down to ineffective algorithms, however we are confident that the **PAL** data-set is too small to capture any useful trends that are necessary for these methods to work. Please note that $IGCN\ w/Smoothing$ observes no result as the data-set is full.

User Effort

The amount of effort a user must exert before obtaining an accurate user model is determined by the number of interview questions they face. Figure 6.3 shows an error vs. questions asked comparison between proposed interview methods on the **KDD** data-set.

For static based methods, the observed behaviour is as expected, accuracy of the user model increases with the number of questions asked. Moreover *IGCN* & *IGCN Smoothed* experience similar behavior. The maximum number of questions asked by *IGCN* was 30, after this point information gain of the 31st question was deemed irrelevant to finding the best user neighbourhood.

Looking further at DF (fig.6.4) we observe a steady decrease in error as questions asked increases, after asking 15 questions the error increases once again. In the decision forest implementation the number of questions asked is determined by the tree depth. Figure 6.5 shows the effect of depth on error, increasing the depth past 4 results in worsened error. This observation is a result of over-fitting, in short training the DF has led the model to fit the training set too precisely, reducing its ability to generalize for unknown observations. To avoid this we limit the tree depth in the final solution.

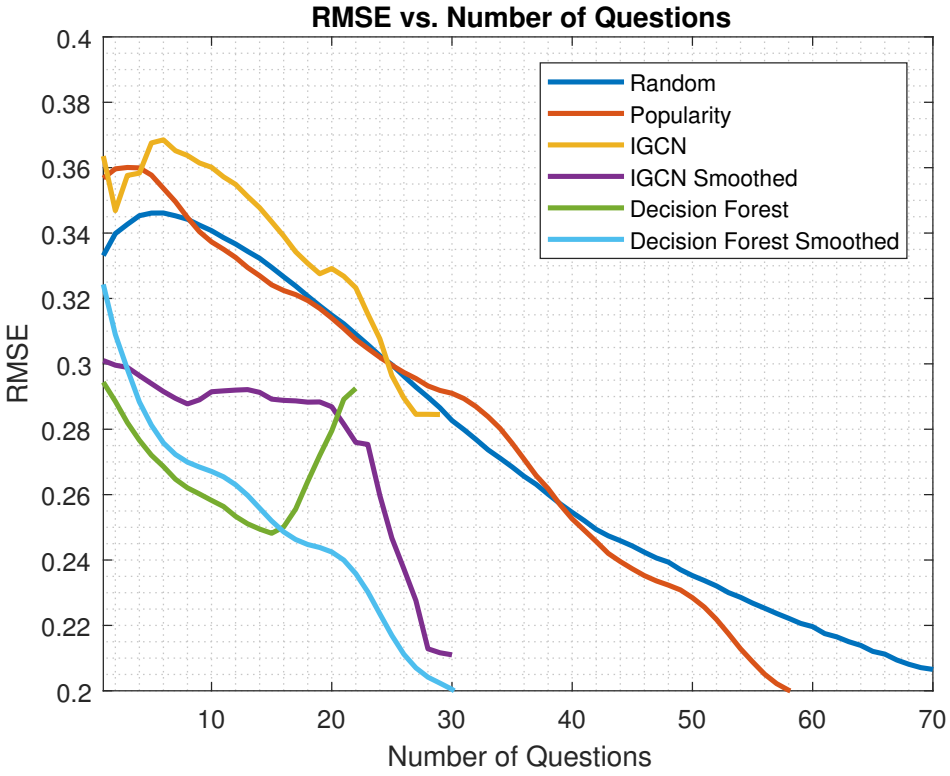


Figure 6.3: RMSE vs. Questions Asked Comparison

Finally observing the DF implementation after smoothing was applied we notice that increasing the number of questions does not have the same effect as for DF. This phenomenon can be explained by the fact that there is more training data to construct the DT, this allows us to build a tree with greater depth without worrying about over-fitting the data at an early stage in tree construction.

As the results for **PAL** were inconclusive we have decided to omit the results for user-effort analysis.

Method	Questions Asked	RMSE
Popularity	138	0.1452
Random	163	0.1715
IGCN	30	0.2845
IGCN Smoothed	30	0.2110
Adaptive DF	15	0.2482
Adaptive DF Smoothed	65	0.1325

Table 6.2: Best RMSE Values and Corresponding Number of Questions

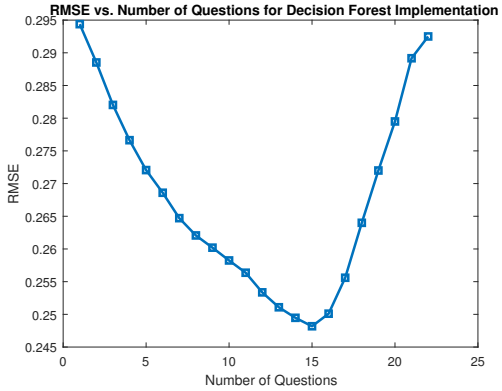


Figure 6.4: RMSE vs. Questions DF

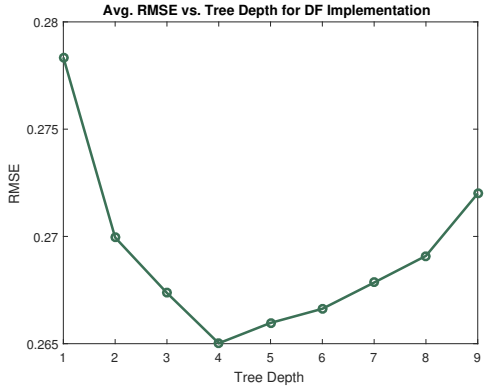


Figure 6.5: RMSE vs. Tree Depth

Parameters

For completeness we report the best parameters used in each implementation 6.3. The majority of the parameters were determined via brute force. Optimal cluster size was determined using the elbow method, whereby the cluster size is the number of clusters corresponding to 90% of variance explained (ie. ratio between group variance and total variance).

Method	**Dataset**	**Parameters**
IGCN	KDD	Cluster Size = 163, L = 10, PQ = 15, NPQ = 15
IGCN Smoothed	KDD	Cluster Size = 143, L = 10, PQ = 20, NPQ = 10 , SVD Dimension = 25
Decision Forest	KDD	Max Depth = 4, Ensemble Size = 10

Table 6.3: Parameters for KDD dataset

6.3 Task Recommendation

6.3.1 Experimental Setting

The testing of the task recommendation module occurs on the **KDD** data set. We opt not to test the system on the PAL data set due to its limited size. Recall that there are two stages to task recommendation, topic and question recommendation. To test the topic recommendation we utilize the same structure of data as described in the new-user model estimation testing. To test the question recommendation we can map the data in the KDD data-set to a question utility matrix (fig 5.6) as follows:

$$\begin{aligned} Student\,Id\,(U) &\mapsto User \\ Problem\,Hierarchy\,(T) &\mapsto Topic \\ Problem\,Name\,(Q) &\mapsto Question \\ Correct \in S \times T \times Q &\mapsto Performance \end{aligned}$$

This transformation of data for a single topic results in a utility matrix with 574 users and 1206 questions. Data sparsity of this data-set is 90.8%.

6.3.2 Topic Recommendation

To determine the accuracy in the action selection we compare the effects of the two proposed selectors, fitness proportionate selector and temporal selector. More specifically how accurate each solution is at determining the performance of each topic over time. The significance of having accurate performance measures over time is that it allows us to recommend the most suitable tasks for a student at that time. For this comparison we use the data for a single topic and compare for each student the performance at time $t + 1$ against the average performance (as in the fitness proportional selector) and the predicted performance (as in the temporal selector) at time t. The time t represents the performance on a single day.

For the temporal fitness proportional selector we must determine the most appropriate regressors $\mathbf{x}$ to use for the logistic regression function. Naturally the first regressor is time t, such that the outcome at $t + 1$ is dependant on the previous time. The remaining regressors as aforementioned are *lag* terms, we calculate the auto-correlation function to determine the best number of *lag* terms to use. It was found that the average number of lag terms that achieved the best performance across all students was 3.

Figure 6.6 shows a comparison between the temporal and average performance selector for a single student in the KDD data-set. Initially we can see that the average out performs the temporal prediction, this is expected as the data available to train the logistic function is minimal. As the number of observations increases, so to does the performance of the temporal average prediction, eventually outperforming the average. There is a case to be made that while there is insufficient data to provide accurate temporal prediction, we should combine the average performance when performing FPS.

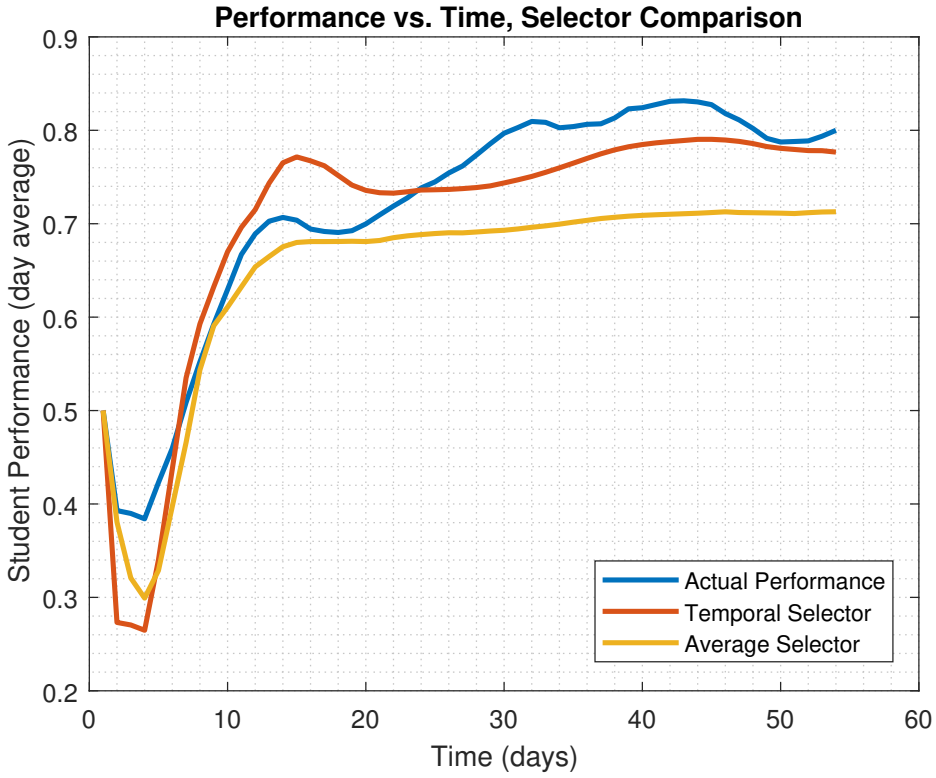


Figure 6.6: Student Performance vs. Time - Comparison of Selectors.

Method	**RMSE**
Average Performance	0.2139
Temporal Avg. Performance	0.1998

Table 6.4: RMSE Comparison between Average and Temporal Average against Actual Observed Performance

Table 6.4 summarizes the accuracy comparison between the two methods across all student in the test data-set.

6.3.3 Question Recommendation

To provide question recommendation we suggest first predicting the estimated performance $\hat{p}$ of each student in every question within a topic. Question recommendation is determined based on the problems that students find challenging such that $\hat{p} \approx 0.5$. In order to evaluate the quality of question recommendation we measure the accuracy of the performance prediction.

Testing is performed using k-fold cross-validation, the details of which have been previously discussed. Each sample in D_{test} represents the performance p of a subset of questions from a current user of the PAL system. The samples in D_{train} are used to predict the performance $\hat{p}$ of the questions in the test data-set. The accuracy of

the prediction is determined by the RMSE (6.1).

The proposed solutions (*Neighbourhood Model & Latent Factor Analysis*) are compared against two baselines discussed earlier in this report, *Global average* and *Biased user-task*.

Table 6.5 shows the accuracy comparison of the baseline and proposed methods, we observe that Latent Factor analysis performs significantly better at predicting student performance than any other solution. We see that by incorporating student and task bias the performance of the algorithm also improves. Moreover the effect of slip & guess factors being implicitly encoded in latent factor models can also be seen, with every latent factor method outperforming neighbourhood models.

Method	RMSE
Global Average	0.419
Biased User-Task	0.410
Neighbourhood Model	0.402
Neighbourhood Model w/Bias	0.374
Latent Factors	0.389
Latent Factors w/Regualarization	0.364
Latent Factors w/Bias	0.356

Table 6.5: RMSE Comparison of Question Performance Prediction Methods

The effect of adjusting the number of latent factors and maximum iterations etc. used to train the model can be found in Appendix A

Chapter 7

Software Implementation

This section details the back-end software implementation of the PAL system. The final software was implemented in MATLAB. The choice to use MATLAB over more conventional programming languages such as JAVA is a result of the current application being in a development phase. The lack of data available to provide suitable analysis of the new-user estimation module and task recommendation module, meant that there was a desire to produce a software implementation that made testing and analysis of the system easier when the final system was ready to be implemented. As such the MATLAB implementation supports all the algorithms suggested and incorporates testing and analysis functions allowing optimal model parameters to be determined when sufficient data is available.

Additional effort was taken when designing the architecture of the MATLAB implementation to account for the possibility that the system will eventually be developed on a more conventional platform like JAVA. As such aspects of the software are partitioned as they would be in such a system. The system flow diagram in figure 7.1 depicts the operation of the new user estimation module. A background process is run prior to a new user entering the system to generate the ML learning models, depending on the model the corresponding data is saved to a database. For example for Adaptive Bootstrapping using Decision Trees (ABDT), algorithm 3 is run for multiple decision trees, the resulting decision forest is stored in memory. The advantage of storing models in this way allows for real time computation for when a new-user arrives to be minimised.

After the interview process is complete, using either the ABDT or IGCN model, the initial user model and new-user neighbourhood are combined to produce a final new-user model, the details of which have been discussed in chapter 4.

Similarly the flow of the system for a current user requiring a task recommendation is shown in figure 7.2. As aforementioned there are two stages to task recommendation, topic suggestion and question suggestion. Topic suggestion is provided using fitness proportionate selection, with fitness weights being determined either by the topic model average or temporal average. The final stage in task recommendation is to provide a specific question, this is done using either the latent factor model or neighbourhood model methods described in section 5.3

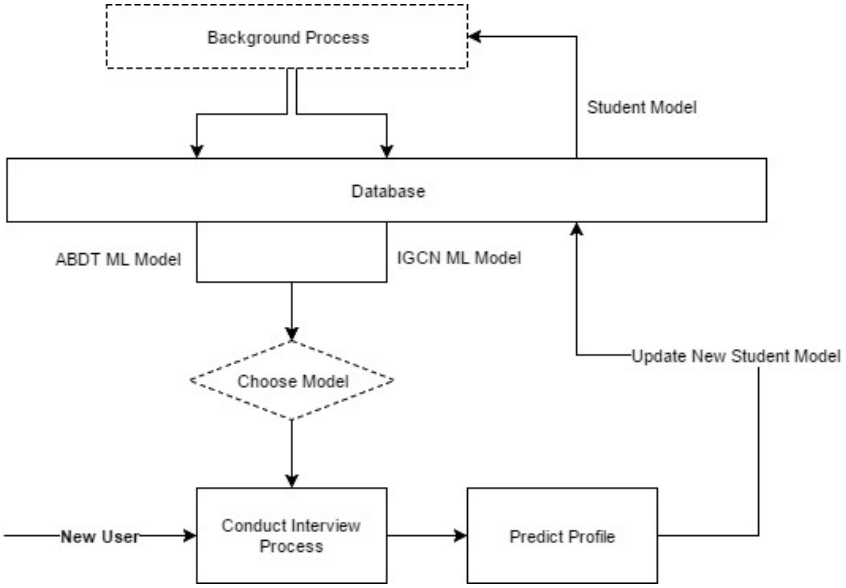


Figure 7.1: PAL System Diagram for New User Bootstrapping

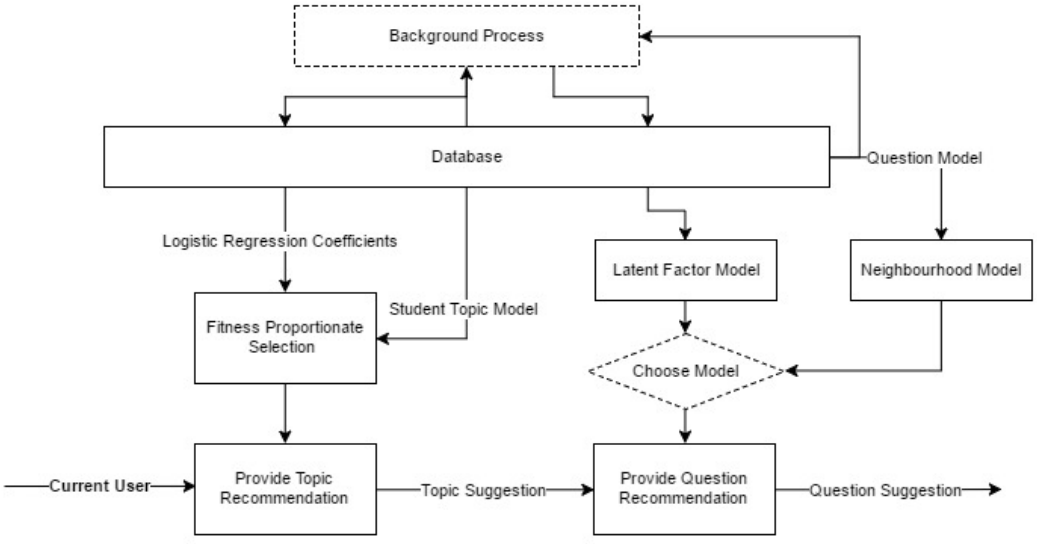


Figure 7.2: PAL System Diagram for Providing Task Recommendation

Chapter 8

Evaluation

This work has proposed a task-based educational recommendation system that is intended to compliment an existing application in the PAL architecture. The proposed recommendation system is able to provide task recommendation to existing users of the system, accounting for the temporal effects often seen in educational environments. Moreover the proposed system is able to bootstrap a new user model effectively requiring on average 15 questions on the test-data.

While the initial objectives have been met, there are a few drawbacks to the proposed implementation. Firstly as the PAL architecture is currently in development, there was a lack of experimental data that the system could be tested with, as such the true effects of our proposed system could mean that critical analysis is still required. To mitigate some of these effects, we test the proposed solutions using a data-set not unlike the one expected in the PAL system.

Due to time constraints it was not possible to test the entire system as a unit, arguably the only accurate means of testing the system would be in a live environment with actual user's of the system. As this was not possible we are unable to completely validate the effectiveness of task recommendation. However under the assumption that when presenting a task to a student, the difficulty of the task should be slightly greater than the students current performance level, we have shown that the proposed system should work in practice, as we are able to predict student performance quite accurately.

Finally, at present we have only considered student performance as the determining factor for task recommendation, in reality there may be other equally useful factors, such as a students preference or emotional response to certain types of activities. The reason these factors were not considered in the analysis of the proposed system was due to the absence of such data. We do however provide alternate solutions that may be considered, when discussing future work.

Chapter 9

Conclusion & Future Work

The proposed task-based recommendation system is a prototype that has shown promising results in utilizing student performance to provide task recommendation. Although it was not possible to test the effect of task recommendation on children in a live environment, if the hypothesis that the most suitable task to present is one which has a difficulty slightly over the students current competency level is correct, we have shown that the task-based recommendation system should work in practice. Moreover the performance of the new-user estimation model showed promising results, allowing for a new-user to be bootstrapped after the completion of only 15 tasks.

9.1 Future Work

9.1.1 Incorporating Professional Knowledge

Recall that when bootstrapping a new-user profile, specific questions were provided at random. The problem with this method is that two different questions may have different difficulty levels, as such a two similar student may be classified incorrectly if one is presented with a more difficult learning task. By incorporating professional knowledge, indicating the difficulty of each question to each child (eg. for a child who has been diagnosed with diabetes for 2 months, the difficulty of this question is 4/10) this mis-classification situation could potentially be alleviated. This solution could also be explored for providing on-going question recommendation, such that each question performance prediction is enriched with supplementary professional knowledge.

9.1.2 Accounting for Student Preference

While it is sufficient to assume that the most appropriate task to recommend to a student is governed by student performance data, incorporating external information could enrich the student model, allowing for even better task recommendation. This notion was explored in the early stages of this project, however due to limited data availability in the current PAL application, these ideas were not pursued.

By considering the notion that a child will learn more effectively if provided with exercises they enjoy, a potential avenue for future work could be to incorporate emotional response into the task-recommendation engine. To this end an Android application

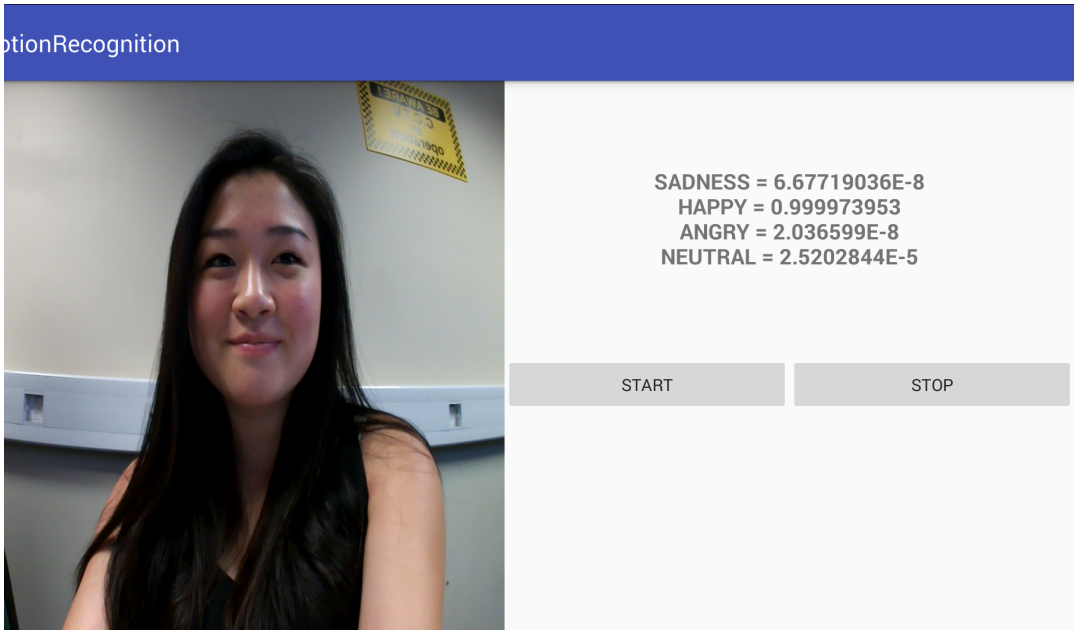


Figure 9.1: Example of Emotional Response Android Application

has been developed which is able to record the emotional response of each child at time intervals. Figure 9.1 shows the operation of this application, the data recorded is a confidence score indicating the probability that the emotion expressed is either happy, sad, angry or neutral. The cognitive recognition of this application is provided by the Microsoft Emotion API[1].

[1]https://azure.microsoft.com/en-us/services/cognitive-services/

Appendices

Appendix A

Supplementary Results

When training a latent factor model for providing question recommendation, we have to perform analysis to determine the optimal number of parameters and maximum number of iterations to train the model on. Figure A.1 depicts the effect of adjusting the number of factors against the error rate (RMSE). The effect of increasing the number of iterations (epochs) the stochastic gradient descent algorithm completes is shown in figure A.2.

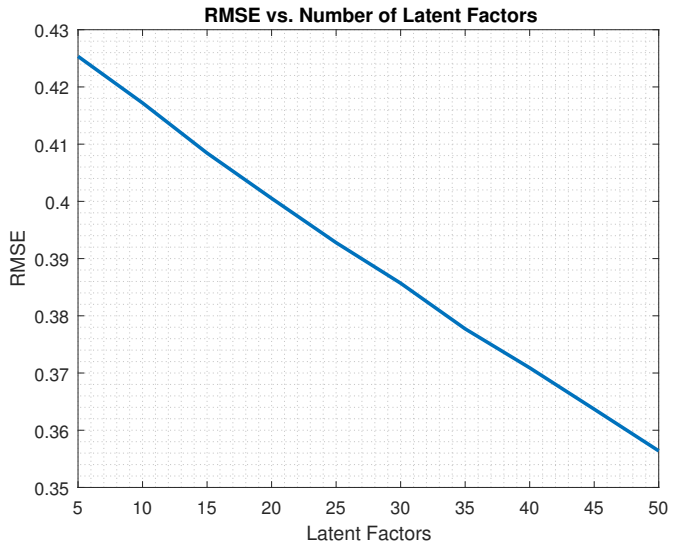


Figure A.1: Effect of Increasing the number of Latent Factors on RMSE

The neighbour-hood model was suggested as an alternative to latent factor models to provide question recommendation. The parameter α was used to determine the size of the user neighbour-hood, the effect of which can be seen in figure A.3.

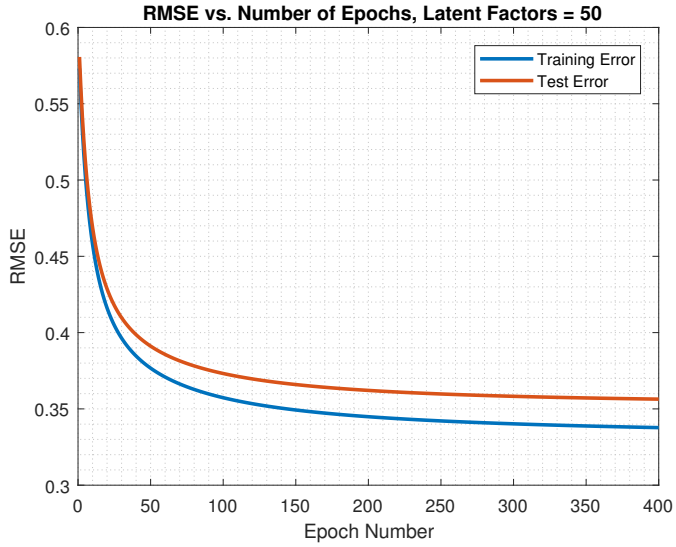


Figure A.2: Effect of Increasing the number of Epochs on RMSE

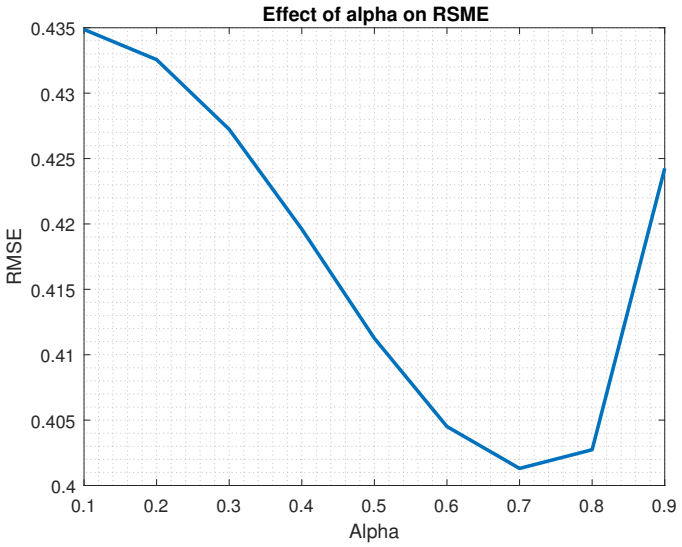


Figure A.3: Effect of Adjusting Alpha RMSE

Bibliography

[1] American Diabetes Association. Diagnosis and classification of diabetes mellitus. *Diabetes care*, 33(Supplement 1):S62–S69, 2010.

[2] World Health Organization et al. *Global report on diabetes*. World Health Organization, 2016.

[3] Diabetes UK. Facts and stats. 2016.

[4] Suhel Ashraff, Muhammad A Siddiqui, and Thomas E Carline. The psychosocial impact of diabetes in adolescents: A review. *Oman medical journal*, 28(3):159–162, 2013.

[5] World Health Organization et al. *Global status report on noncommunicable diseases 2010*. Geneva: World Health Organization, 2011.

[6] N Hex, C Bartlett, D Wright, M Taylor, and D Varley. Estimating the current and future costs of type 1 and type 2 diabetes in the uk, including direct health costs and indirect societal and productivity costs. *Diabetic Medicine*, 29(7):855–862, 2012.

[7] Juliet M Corbin and Anselm Strauss. *Unending work and care: Managing chronic illness at home*. Jossey-Bass, 1988.

[8] Robert John Adams. Improving health outcomes with better patient understanding and education. *Risk Management and Healthcare Policy*, 3(1):61–72, 2010.

[9] Thomas Bodenheimer, Kate Lorig, Halsted Holman, and Kevin Grumbach. Patient self-management of chronic disease in primary care. *Jama*, 288(19):2469–2475, 2002.

[10] Kate R Lorig, Philip Ritter, Anita L Stewart, David S Sobel, Byron William Brown Jr, Albert Bandura, Virginia M Gonzalez, Diana D Laurent, Halsted R Holman, et al. Chronic disease self-management program: 2-year health status and health care utilization outcomes. *Medical care*, 39(11):1217–1223, 2001.

[11] Joseph Beck, Mia Stern, and Erik Haugsjaa. Applications of ai in education. *Crossroads*, 3(1):11–15, 1996.

[12] Valerie Shute, Robert Glaser, and Kalyani Raghavan. Inference and discovery in an exploratory laboratory. Technical report, DTIC Document, 1988.

[13] Beverly Park Woolf. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann, 2010.

[14] Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[15] Nguyen Thai-Nghe, Lucas Drumond, Artus Krohn-Grimberghe, and Lars Schmidt-Thieme. Recommender system for predicting student performance. *Procedia Computer Science*, 1(2):2811–2819, 2010.

[16] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[17] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[18] Robert Kass and Tim Finin. Modeling the user in natural language systems. *Computational Linguistics*, 14(3):5–22, 1988.

[19] Michael Pazzani and Daniel Billsus. Content-based recommendation systems. *The adaptive web*, pages 325–341, 2007.

[20] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

[21] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[22] Alejandro Bellogín and Arjen P De Vries. Understanding similarity metrics in neighbour-based recommender systems. In *Proceedings of the 2013 Conference on the Theory of Information Retrieval*, page 13. ACM, 2013.

[23] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[24] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 595–604. ACM, 2011.

[25] Le Hoang Son. Dealing with the new user cold-start problem in recommender systems: A comparative review. *Information Systems*, 58:87–104, 2016.

[26] Manolis Vozalis and Konstantinos G Margaritis. Collaborative filtering enhanced by demographic correlation. In *AIAI Symposium on Professional Practice in AI, of the 18th world Computer Congress*, 2004.

[27] Ahmad Nurzid Rosli, Tithrottanak You, Inay Ha, Kyung-Yong Chung, and Geun-Sik Jo. Alleviating the cold-start problem by incorporating movies facebook pages. *Cluster Computing*, 18(1):187–197, 2015.

[28] Hyung Jun Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008.

[29] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 315–324. ACM, 2011.

[30] Al Mamunur Rashid, George Karypis, and John Riedl. Learning preferences of new users in recommender systems: An information theoretic approach. *SIGKDD Explor. Newsl.*, 10(2):90–100, December 2008.

[31] Tom M Mitchell et al. Machine learning. wcb. 1997.

[32] Tibshirani Hastie and R Tibshirani. & friedman, j.(2008). the elements of statistical learning; data mining, inference and prediction.

[33] Mingxuan Sun, Fuxin Li, Joonseok Lee, Ke Zhou, Guy Lebanon, and Hongyuan Zha. Learning multiple-question decision trees for cold-start recommendation. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 445–454. ACM, 2013.

[34] Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel, and Rob Koper. Recommender systems in technology enhanced learning. In *Recommender systems handbook*, pages 387–415. Springer, 2011.

[35] Nguyen Thai-Nghe, Tomáš Horváth, and Lars Schmidt-Thieme. Context-aware factorization models for student's task recommendation. *Proceedings of UMAP*, 2011.

[36] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 191–226. Springer, 2015.

[37] Kenneth R Koedinger, Ryan SJd Baker, Kyle Cunningham, Alida Skogsholm, Brett Leber, and John Stamper. A data repository for the edm community: The pslc datashop. *Handbook of educational data mining*, 43, 2010.

[38] VK Zaretskii. The zone of proximal development: What vygotsky did not have time to write. *Journal of Russian & East European Psychology*, 47(6):70–93, 2009.

[39] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[40] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207, 1995.

[41] Joseph Psotka, Leonard Daniel Massey, and Sharon A Mutter. *Intelligent tutoring systems: Lessons learned*. Psychology Press, 1988.

[42] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K Lam, Sean M McNee, Joseph A Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.

[43] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Data mining cluster analysis: basic concepts and algorithms. *Introduction to data mining*, 2013.

[44] Manh Cuong Pham, Yiwei Cao, Ralf Klamma, and Matthias Jarke. A clustering approach for collaborative filtering recommendation using social network analysis. *J. UCS*, 17(4):583–604, 2011.

[45] Arnd Kohrs-Bernard Merialdo. Clustering for collaborative filtering applications. *Intelligent Image Processing, Data Analysis & Information Retrieval*, 3:199, 1999.

[46] Gilda Moradi Dakhel and Mehregan Mahdavi. A new collaborative filtering algorithm using k-means clustering and neighbors' voting. In *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*, pages 179–184. IEEE, 2011.

[47] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[48] Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[49] J Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[50] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[51] College of Computer and Information Science, Northeastern University, 2014.

[52] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[53] Nguyen Thai-Nghe, Lucas Drumond, Tomáš Horváth, Artus Krohn-Grimberghe, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Factorization techniques for predicting student performance. *Educational Recommender Systems and Technologies: Practices and Challenges*, pages 129–153, 2011.

[54] J Stamper, A Niculescu-Mizil, S Ritter, GJ Gordon, and KR Koedinger. Algebra 2005–06. *Challenge Dataset from KDD Cup*, 2010.

[55] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.