The Synthetic Student: A Machine Learning Model to Simulate MOOC Data

by

Michael Wu

Submitted to the Department of Electrical Engineering
and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2015

Author .....
Department of Electrical Engineering
and Computer Science
May 22, 2015

Certified by .....
Kalyan Veeramachaneni
Research Scientist
Thesis Supervisor

Accepted by .....
Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

The Synthetic Student: A Machine Learning Model to Simulate MOOC Data

by

Michael Wu

Submitted to the Department of Electrical Engineering
and Computer Science
on May 22, 2015, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

It's now possible to take all of your favorite courses online. With growing popularity, Massive Open Online Courses (MOOCs) offer a learning opportunity to anyone with a computer - as well as an opportunity for researchers to investigate student learning through the accumulation of data about student-course interactions. Unfortunately, efforts to mine student data for information are currently limited by privacy concerns over how the data can be distributed. In this thesis, we present a generative model that learns from student data at the click-by-click level. When fully trained, this model is able to generate synthetic student data at the click-by-click level that can be released to the public.

To develop a model at such granularity, we had to learn problem submission tendencies, characterize time spent viewing webpages and problem submission grades, and analyze how student activity transitions from week to week. We further developed a novel multi-level time-series model that goes beyond the classic Markov model and HMM methods used by most state-of-the art ML methods for weblogs, and showed that our model performs better than these methods. After training our model on a 6.002x course on edX, we generated synthetic data and found that a classifier that predicts student dropout is 93% as effective (by AUC) when trained on the simulated data as when trained on the real data. Lastly, we found that using features learned by our model improves dropout prediction performance by 9.5%.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Research Scientist

Acknowledgments

Working on this thesis has been a difficult but satisfying journey. Making it this far was the product of a number of influential factors for which I am infinitely grateful.

First, I've had the true pleasure of working with my adviser Kalyan. After working with other mentors on technical projects, I have to say that Kalyan has provided me with unmatched attention, back-and-forth dialogue, and feedback. His care and thoughtfulness to the data and methodology of research has been an eye-opening learning experience. I'm very grateful to have been able to grow and mature as an investigator, a researcher, and as a communicator under Kalyan's guidance.

Next, I thank MIT and the professors I've had along the way for providing me with an excellent education, both in academic content and in personal perseverance. Before arriving here I was a blank slate, and my experiences in the last four years have shaped me into someone with the technical skills and motivation to put together such a deep and widely scoped project.

Finally, I appreciate my parents, my sister, and my friends for their irreplaceable support. My parents have long instilled in me the right values and habits to accomplish greater and greater things. My younger sister Julia never ceases to make me laugh and reminds me to keep an innocent perspective. And my good friends are amazing for setting excellent examples for me, and for keeping me happy, productive, and fulfilled.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Massive Open Online Courses (MOOCs) have recently emerged as potential solutions to meet increasing demands for higher-level education. A number of platforms, such as edX, Coursera, Udacity, and Khan Academy provide courses in a wide range of topics, from New Venture Finance to Statistical Inference and more. This method of education has become quite popular.

In 2012, MIT offered an online version of its Circuits and Electronics class (6.002), which initially drew about 150,000 students, with 7,157 completing and passing the course. Since then, a number of other universities, including Harvard, have joined this effort. Now named edX, the program offers 300 different courses and has more than 1 million unique students enrolled. Course options are often either free or relatively inexpensive, and thus are much more affordable for the general population than a traditional four-year college education. MOOCs therefore have the potential to provide education for millions of students worldwide. It would be tremendously useful to understand how students learn online, to improve courses and create meaningful student learning experiences.

Due to the increasing participation in MOOCs, data on student learning can now be accumulated at an unprecedented scale. MOOCs can track and record every step students take as they engage with course material. The types of student interaction

include clicking on links for viewing study resources, submitting answers to problems and exercises, and posting or asking questions in forums. Scientists looking at this data can conduct research and find answers to a wide variety of questions, such as "Which students are likely to drop out, and why?", "What areas of this course are most challenging for students?", and "What types of study behaviors lead to the best results?" Researchers, educational experts, and professors can mine this data to gain greater understanding about the learning process and improve existing courses.

Hence there emerges the need to make student data open and available to the public. However, with this data, there is a need to protect student privacy. For example, it is reasonable to assume that most students would prefer to not have their identity linked to analytics regarding their educational progress and/or grades. Thus itâĂŹs critical for the success of MOOCs that personal data never becomes identifiable.

The limitations of data anonymization are well documented. For example, the simple act of anonymizing the identity of students is not very effective. Someone looking at course forum posts could deduce student identities, given each student's actions. Since there are many possible ways for third parties to infer identities, it is difficult to reason about which anonymization schemes or techniques offer what types of protection guarantees. One standard for data anonymization is k-anonymity (i.e. given data, a group of k - 1 members described in that data cannot be used to figure out identities of other group members). Applying ideas and standards of anonymity toward MOOC data could lead to safer informational distribution.

1.2 Thesis Goals

In this thesis, we pursue a full machine learning method to model student data gathered on edX. Once the model is trained on a course's data, it is then able to generate synthetic student data. When carefully designed, the computer generated data should conceal individual student contributions, and be able to be freely distributed without loss of student privacy.

The challenging aspect is to design a model that captures the patterns and insights found in course data (e.g., identifying student types and their learning habits or relative usefulness and difficulty of course resources, such as lecture notes or assigned problems). The data set contains important nuances, such as varying behavior of students over the weeks of the course, intrinsic differences between students, and problem submission results. In general, existing machine learning models cannot be directly applied.

Two important design considerations for model development are noted below.

Model granularity. The first consideration is the level of data granularity we wish to model. Most existing research on online student learning has delved into describing student activity at a broad level (e.g., calculating total time spent online, the number of correct submissions, the average length of a forum post, and so on). On the other hand, one can model the data as a time-series sequence of clicks through every course resource (URL) and every problem submission made by a student working through a course. We believe that building a model on a click-by-click level can support a greater variety of studies and lead to more types of conclusions.

Having noted the click-by-click level, the data also presents a unique challenge at a week-by-week level. While most *web log* data mining considers individual sequences drawn from the same distribution, each week in a course usually covers new material. Students therefore interact with a different set of resources each week. A good model must then learn time-series behavior for each week and be able to characterize how individual students transition in behavior over time. This may prove to be useful in understanding which students will dropout.

Generative vs. discriminative. A second design choice insists that the model be generative (learning the distribution over all of the variables) rather than discriminative (learning how to predict certain variables given information about other variables). This is partly because the model aims to be able to create simulated students and not just make individual predictions about the students.

The model also aims to capture the distribution of these sequences and the process behind how students create them. In this case, a generative model provides deeper and fuller characterization of the data than a discriminative one.

1.3 Thesis Contributions

Our approach to modeling time series data can be of interest intrinsically and theoretically. When applied to real student data, however, the model furthers the cause of understanding student learning in two major ways.

1. Our project presents the first complete attempt to model online student data. Covering a number of aspects, such as time-series activity, week-by-week student behavior, trends regarding how long students spend on webpages and how accurately they answer questions, the work is capable of answering many important questions about learning.

   As a generative model, synthetic student data can be generated containing patterns similar to that of real data. We will demonstrate that this synthetic data protects student privacy for public information release, so that our project could have a large impact in opening educational research to more people.

2. Our model extracts useful features about students that provide new value towards understanding student learning. Learned features are model-inferred quantities that cannot be gleaned from the data directly. Once model parameters are learned on a set of given student data, they provide greater insight into individual student behavior than what was previously achievable.

   We will show that these features can be useful in one important area, namely, predicting student dropouts. Predicting which students will drop out of a course and when is a notable question with regard to MOOCs, because it looks at what factors might indicate or influence student decisions. It turns out that using

the inferred features from our model in a logistic regression model to predict weekly dropouts improves prediction accuracy.

1.4 Key Findings

As the focus of our work is to build a general model for the data, and not towards drawing particular conclusions from it, our findings largely involve how well the model fits the data.

- A machine learning model for dropout trained on synthetic data is *only* 7.5% worse than one trained on real data, as measured by Area Under the Curve (AUC).
- Student features learned by the time-series model are valuable. A machine learning model that uses these features to predict student dropout improves upon one that uses only simple features from an average AUC score of .81 to an average of .89.
- Student click-by-click data closely follows a Markov process. However, our novel time-series model, a combination of a Hidden Markov model and a Markov model, improves log-likelihood fit by 6.8%.
- Synthetic data generated by our model preserves relatively how well individual student features contribute to dropout predictions. When a machine learning model is trained using one or two features at a time from a set of five easily computable features, there is a correlation of .72 between the rankings when the training data is synthetic versus real.
- Student activity on a week-by-week basis can be well-characterized by a Gaussian Mixture model. There are about 20 primary types or clusters of users in one edX course.

- Problem submission behavior does not follow the Markov process. Instead, students make problem submissions in groups that are better characterized by the Latent Dirichlet Allocation process.

- Unsurprisingly, the time spent viewing each course resource or the assessments of each problem submission can be better described using a parametric model taking into account both the student and the resource / problem, rather than by considering either only the student or only the resource / problem.

1.5 Related Work

This work draws upon ideas and results from the following three areas of research: time series models, data anonymization methods, and MOOC data analyses.

1. **Discrete time series models.** In this broad area, the most well-known models are Markov chains and Hidden Markov models (HMMs). As a secondary approach, click-stream data of web surfers has been clustered using a mixture of HMMs, trained using an adapted EM (Expectation-Maximization) algorithm [3]. These existing approaches are relevant to modeling student click-stream data. For instance, transitions between resources often manifest as clicking on links between web pages, thus exhibiting clear Markov chain behavior.

   The main novel approach to modeling the identities of the event sequences (webpage IDs and problem IDs) in this thesis is an integrated Markov chain and HMM generative model. In this model, the emission probability of the next event in the sequence is a function of both the current hidden state and the last event in the sequence, thereby combining Markov chains and HMMs. In chapter 6 we explain why this approach specifically fits the data, and we detail the precise model definition and training procedures. Additionally, learning a mixture of these combined Markov chain-HMM models captures some of the differences between student learning behavior.

2. **Data anonymization methods**. While other methods such as clustering or randomization could be useful in this context, they provide no insight into learning actual student behavior. Instead, we propose to ensure privacy by developing a generative model for the students and then sampling event sequences from the model. Steps are taken to ensure anonymity in the training process. One measure is to remove resources that only a few individuals visit, so as to not identify these individuals. All individual student parameters are not explicitly kept or shared, but rather modeled to come from a smooth distribution.

3. **MOOC research** Detailed research has been done on several aspects of edX course data. Other works characterize dropout rates [8] or problem submissions [5]. While neither is the focus of this paper, we borrow simple approaches from both. One aspect of modeling student event sequences involves developing a structure for problem submissions. We also use dropout predictions as a key method for validation of our model.

1.6 Thesis Outline

The rest of the thesis is organized into the following chapters.

- Chapter 2 introduces the data collected on edX courses and the necessary data curation steps.

- Chapter 3 explains the full problem we attempt to solve in the thesis and an overview of the complete machine learning model solution.

- Chapter 4 presents the problem topic model that characterizes how students choose which problems to solve.

- Chapter 5 presents the event descriptions model that learns the distribution of time spent on viewing resources and assessments of problem submissions, across students and across individual resources and problems.

- Chapter 6 presents the Latent-Explicit Markov Model, a novel time-series model, and explains the model's training equations and its application to edX data.
- Chapter 7 presents the course level model of learning how student activity transitions from week to week.
- Chapter 8 describes the methods used to validate the full model and the discovered results.
- Chapter 9 concludes the thesis by summarizing important takeaways and future topics for research.

Chapter 2

MOOC Data

In this thesis we worked with student data from classes offered on EdX, in particular the 6.002 Circuits and Electronics course in fall 2012. The data was maintained in a database outlined by the MOOCdb schema [1].

We present a brief overview of what the course data looks like, from a 6.002x Course Report [7]. The course attracted around 30,000 students who viewed at least 10 resources. Around 18,000 made at least one problem submission and about 3,000 received a certificate for passing the course. Among these students, 87% were men, and as a group were well-educated, with 42% having received a bachelor's degree. The population was diverse by location, with India and US the most well-represented of the 150 countries that had at least one participant. The median student age at 25 years was slightly higher than that of a typical undergraduate student.

Among the course materials, there were approximately 50 distinct HTML webpages, which collectively contained over 400 lecture videos and nearly 200 problems. The EdX tracking system logged around 35 million separate events of student interaction which provide the data stored in the database.

2.1 Data Details

The database consists of 24 tables that detail all of the student actions in the course, including student information, student feedback, and course materials. For example,

it contains tables about each course resource (URL) viewed, each problem answer submitted, and each forum post made by the students, the IP address the actions came from, and the parent / child relationships between all of the resources and problems. Our project focused on the student observation events (where students browse some resource on the course website) and submission events (where a student submits an answer to a course problem), and utilized only those tables. Table 2.1 presents the details of the 6.002x data in our database.

Table Name	Number of entries
observed_events	25,726,044 events across all students
resources	210,085 distinct resources
submissions	7,854,754 events across all students
assessments	7,854,754 assessments, one per submission
problems	1,864 distinct subproblems

Table 2.1: Database Table Summary

Figure 2-1 provides a sense of number of observation (browsing) and submission events on a weekly basis. Observed events peak early in the course as more students view the course at the beginning and subsequently leave, a phenomena commonly referred to as *stopout*. Problem submissions are less frequent at the beginning of the course and pick up around week 7, when the midterm was first released. The number of submissions reaches a peak at weeks 10-11.

In the course database, there were approximately 200,000 distinct labeled resources. Among these resources, about 113,000 of these were purely structural, meant for maintaining parent-child relationships between course components, and these resources were not found on the course website itself. This leaves about 97,000 resources that students interacted with. This set contains every little component of each webpage in the course, from each clip or section of a video to every page of the course textbook. The breakdown of the types of these resources is shown below in Table 2.2.

In our database, each observed event is associated with a *URL* which is of a

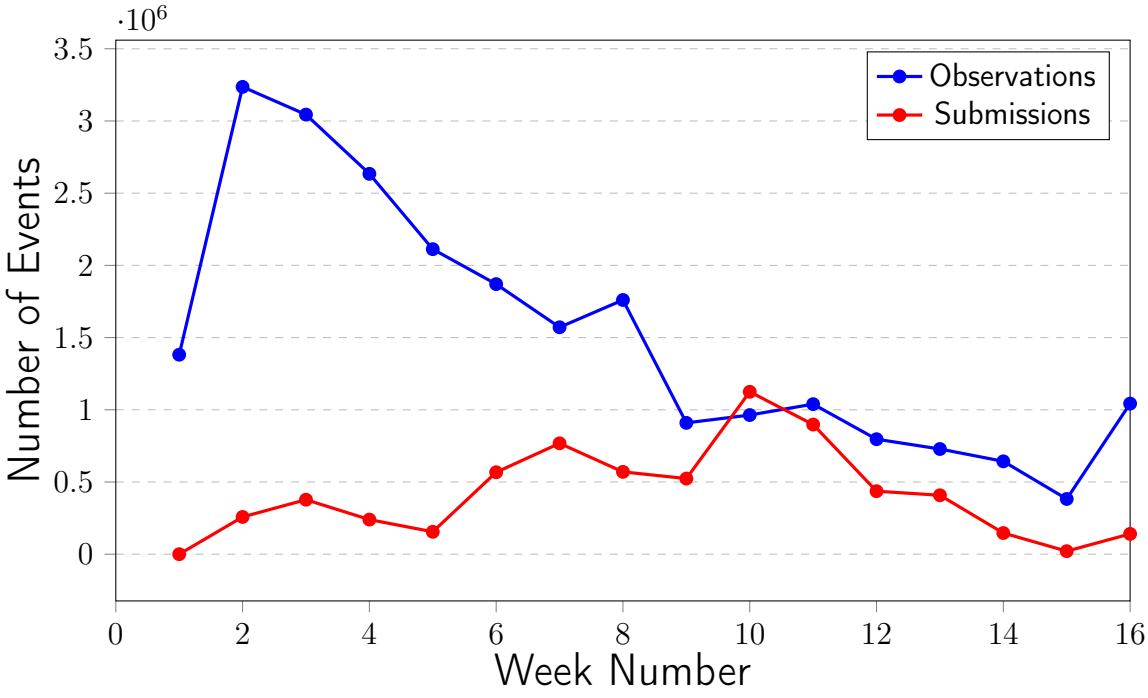


Figure 2-1: Event activity by week. Observations peak early in the course and then decrease as students leave. Submissions spike around week 10, a little after the midterm was released.

Informational	Problem	Lecture video	Book	Forum	Wiki	Profile	None
65	108,900	25,756	1,173	39,739	1,037	2	33,413

Table 2.2: Resource category breakdown.

particular type. Thus we are able categorize the observed events into types. Figure 2-2 shows the frequencies of each type of observed events for each week in the course. The activities are shown for *informational*, *problem*, *lecture video*, *book*,*forum*, and *wiki*.

Once again, the number of problem and book resources spike around week 8 and week 16, soon after the midterm and final exams were released.

The database stores rich information about the student activity. To build a statistical model of student browsing and submission activity, we chose a subset of fields. These fields are:

- Event type: an element of {observation, submission}

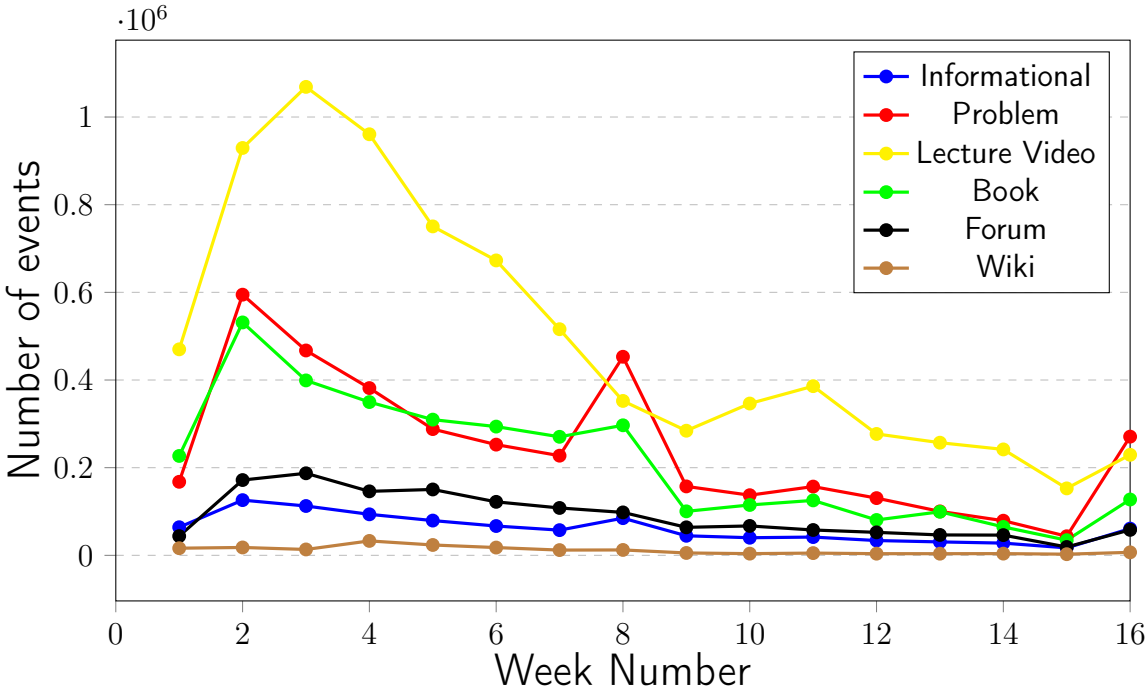


Figure 2-2: Observation event activity by week. The event counts for each observation category tend to follow the trends of the total observations by week, with “lecture videos" being the most browsed and “problem" and “book" the next most browsed.

- URL ID: ID of the observed resource (for observations)
- Problem ID: ID of the problem that was submitted (for submissions)
- Duration: Time spent viewing the resource in seconds (for observations)
- Assessment Grade: Grade assigned to the submission (for submissions)

2.2 Data Extraction and Assembly

We wrote scripts to extract the data from the databases into files, formatted to reflect the sequential aspect of the data. The process involved looking up all of the events associated with each student in the “observed_events" and “submissions" tables, sorting them by time stamp, and then querying the associated duration and grade for them. For further analysis and interpretation of the model results, the “resources" and “problems" tables were useful for looking up the URL and problem

names.

Next, we divided these event sequences by week, with each week designated to start on Monday. Each week's worth of student sequences was written to a separate file. This separation reflects the differences in activity per week. Because new course material is revealed each week, students tend to work with new resources and problem for that week. With such differences in weekly behavior, it makes sense to analyze the activity on a week by week basis. This reasoning is further detailed in the next chapter. In practice, dividing the data into weekly sections also enabled us to train models on a per week basis, making it computationally tractable. A sample of 13 sequential events for a student is shown in Table 2.3.

Event Type	Event Name	Event ID	Duration	Grade
Observation	info/	27	39	
Observation	progress/	6	17	
Observation	courseware/	10	2	
Observation	courseware/Week_12/	11	69	
Observation	courseware/Week_12/The_Operational_-Amplifier_Abstraction/1/	388	16	
Observation	courseware/Week_12/The_Operational_-Amplifier_Abstraction/1/video/S23V1-_Review_previous_amplifier-_abstraction/_cR8XukMGdjk	3291	173	
Observation	courseware/Week_12/The_Operational_-Amplifier_Abstraction/1/	388	143	
Observation	courseware/Week_12/The_Operational_-Amplifier_Abstraction/4/video/S23V4-_Op_Amp-_abstraction/_vLuFuBK5B-g	2077	695	
Submission	S23E1_Non-Inverting_Amplifier/4/1/	40		0
Submission	S23E1_Non-Inverting_Amplifier/3/1/	36		0
Submission	S23E1_Non-Inverting_Amplifier/4/1/	40		0
Submission	S23E1_Non-Inverting_Amplifier/3/1/	36		1
Submission	S23E1_Non-Inverting_Amplifier/4/1/	40		1

Table 2.3: A sample of 13 sequential events for a student.

2.3 Data Curation and Preprocessing

The data we extracted had a few issues that needed careful handling. We curated the original data through the following steps.

Insert session break events: Although the original sequences contain timestamps for observation and submission events, there is no indication of sessions or distinct blocks of time when a student is engaged with the course material. Students are likely to engage with the course in an on and off fashion. We intend to capture these continuous blocks of time when students are engaged with the course as "*sessions*".

To separate the students weekly event sequences into *sessions* we introduce session breaks. We start by inserting a session break event at the beginning of every weekly sequence, at the end of the sequence. Then we insert a break between any consecutive pair of events that are at least one hour apart (evaluated using the difference of timestamps). Thus an hour of no activity by a student is considered as a separation between two distinct activity sessions. This separation of events is useful in modeling sequences because student may engage differently when he revisits the website. There were initially 33,580,798 total events in our extracted data, and this step separated them into 789,907 distinct engagement sessions.

Filter out rarely visited resources and problems: For each week of the course, there are a number of resources and problems visited by the students. To reduce the size of this event set for simplicity, training run time, and to ensure that individual observations or submissions on rare resources or problems do not reveal student identities, we filter out the events corresponding to the visits to less frequented resources. These resources are defined as the ones whose visits account for less than .05% of all visits across all the resources. For problems, we set the threshold as less than .3% for that week. Once we remove such events, the number of observation events drops from 25,726,044 to 17,143,777 (for the entire course) and the number of submissions drops from 7,854,754 to 5,825,201.

Recalculate durations: The duration values in the database were initially incorrect. We recalculated the duration for each observation event using the event timestamps. Each observation event's duration is calculated as the difference between the timestamp of the next event and its own timestamp. If this difference is more than one hour (i.e. the next event is a session break event), then the observation duration is designated to be a fixed 100 seconds.

Remove submissions for a solved problem: Once a student has correctly solved a problem, recordings of subsequent submissions for that problem are an artifact of the UI design. That is, there is only one submit button for multiple subproblems. Hence, any answer submitted later should not be taken as an attempt to solve the problem. When we remove such submissions, the number of total submission events drops from 5,825,201 to 3,602,654.

Summary: By the end of this process, we had examined the data, explored its properties, and extracted the sequences which we wish to analyze. In the next set of chapters we present a series of models that we developed to characterize this data.

Chapter 3

Modeling a Digital Student

Our goal is to model a digital student's engagement patterns on the course website. Student activity however can be defined at different levels of granularity.

At the highest level, one can capture the behavior of students using covariates - for example calculating the number of resources visited in a week, the total time spent engaged in the course, the number of distinct problems answered, the number of forum posts, etc. One can then model students behavior by clustering these covariates on a weekly basis[6], or using a state space model over these covariates [8].

At a more detailed granularity, one might model students' interactions with specific course resources. For example, [9] models social interactions on the forums.

At the finest granularity, one can model the data as it is generated, a time-series sequence of clicks. These clicks capture browsing specific course resources (or URLs) and submissions for problems. This detailed level of granularity is what we are after. Many insights into student learning such as: *which resources are helpful for solving the next problem* and *how successful students are at repeatedly answering a difficult problem*, can be gleaned at the click-by-click level. Thus modeling the student at this granularity is important for generation of synthetic student sequences as we do not a priori limit the purpose of the synthetic student sequences.

In addition, to modeling click-by-click data, we challenge ourselves to characterize each interaction, namely the time spent viewing each resource and the grade assigned to each problem submission. Thus, by modeling every aspect of the student data as

much as possible, the synthetic data generated from the model will be flexible in allowing outside researchers to analyze the data, draw conclusions and build predictive models.

Other modeling work with MOOC students data: We bring up two other studies that attempt to model MOOC students data in order to compare and contrast with our work.

In [8], the authors apply a Hidden Markov model to learn how features describing overall student activity transition from week to week. Their model then predicted whether each student would drop out of the course, as one of the features in their model. We need to build a generative model from which more flexible features can be calculated from the raw click-stream data.

Another study [4] used Latent Dirichlet Allocation to learn groups of course resources that similar students viewed together. While this analysis dives into greater detail at a per-resource analysis, it does not consider the relative time the event occurred at, and cannot uncover some types of insights that a time-series model can.

3.1 Click-by-Click Level Data

Imagine a student as he navigates through an edX course. He might first start at the course homepage, then visit a particular module of the course. Once there, he might view a few lecture videos, or read parts of the online textbook. Then the student might attempt several problems. Depending on whether these are correct, he could attempt some more problems, or go back to the textbook and then retry those problems. These sequences of events define the activity of a digital student on an online platform. Each is represented using the five *tuple* described in the following figure and table.

For each event, we want to model the “event locations" - the identities of the particular URL visited or problem attempted - and the “event descriptors," the attributes that characterize that event. For viewing course resources, this means the amount

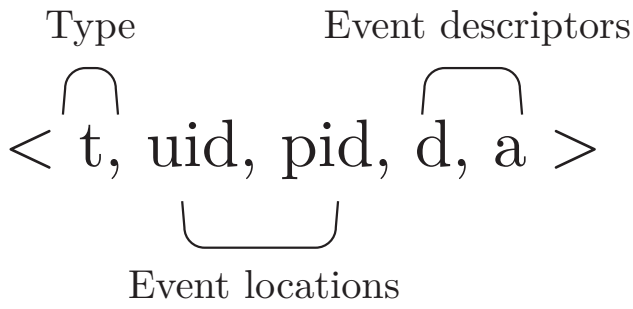


Figure 3-1: A 5-tuple that defines an event in the event sequences. t is the type of the event. u_{id} and p_{id} represent the event locations or the resource locations. d represents the duration of this event and a represents the assessment if a problem submission was part of this event.

Field Name	Field Description
Type (t)	an element of {observation, submission, session break}
URL ID (u_{id})	ID of the observed resource (for observations)
Problem ID (p_{id})	ID of the problem that was submitted (for submissions)
Duration (d)	Time spent viewing the resource in seconds (for observations)
Assessment Grade (a)	Grade assigned to the submission (for submissions)

Table 3.1: Event tuple descriptions

of time a student spent on that resource, measured in seconds. For problem submissions, this means whether the submission was correct or incorrect (in this course there was no partial credit for answers). These considerations sufficiently capture the basic process of how students engage with an online course. These, as we will see offer sufficient complexity and difficulties in the modeling process.

Our goal is to build a statistical model from which we can learn how students navigate through the events in a course, and be able to generate click-by-click samples.

In this problem formulation, there are several factors we **do not** account for.

1. **Timestamp of each event**: In our methodology, we do not capture the exact time in a day or week when the student interacts with the platform. While it would be a interesting challenge to model the times during the day and week each student is active on the online course, we argue that our model

captures most of the temporal aspects of the student activity (just not the exact timestamps). This is because, the events we model are ordered, and the sequence is comprised of sessions separated by session break events. Hence much of the time aspect of student engagement is already captured.

2. **Engagement with forums**: This work does not consider student engagement with forums. While we model the visits to the forums, we do not model exact content of the posts or the nature of the engagement, *that is*, responses or question-asking. This is an aspect we could work on in future.

3. **Exact answer submitted (for open-ended problems)**: For many open-ended questions, there is often a set of common incorrect answers [5]. It may be useful to understand and analyze students sequence of submission of incorrect answers. Currently we only consider whether the student answered correctly or incorrectly.

3.2 Challenges

The task of building a model for these student sequences poses the following key challenges.

Week-to-week variability: Student behavior in an online course changes dramatically as the course progresses. At the beginning of the course, students are just exploring the website and course structure. For courses that follow a weekly schedule, each week new materials appear in the form of notes, lecture videos, homework problems, and labs. The complexity and the nature of the content dictates the way students engage with the course. Thus between each week of a course, there is a huge difference between what students do, the amount of time they spend, and our model needs to account for that. There may also be a midterm and final for the course, and during these periods the serious students would spend more time solving problems. For courses that do not adhere to a weekly schedule, we would need to separate student activity based on modules.

Generating locations for events: The question "Where will this student go next?" is fundamentally the most difficult aspect to model. While information such as the student's previous event(s), his previous activity history, and the week number are useful, the problem of how we can best combine these patterns to make accurate predictions about his next event is challenging. With about 100,000 event locations, the next event location can be hard to determine.

Generating event descriptions: Event descriptions characterize the event by specifying the duration for a observed event and specifying whether or not the student's submissions is correct or wrong for a problem submission event. Learning a very good model for the event descriptors would be akin to learning how engaged a student is in the material or how well he has understood the material, which are both important problems. The event location (that is the particular resource or problem) will have a major effect on the durations and the assessments. For example, resources range from videos to intro pages of a textbook, and problems range from easy to difficult. Additionally, individual students have different tendencies when viewing resources and answering problems. For example, some students spend more time on book vs lecture videos. Through our model we would have to capture inter-resource variability and inter-student variability when generating descriptions for events.

Generating student data for the entire course: As compared to an on-campus course, where most students participate in the course from beginning to the end, edX has a low barrier to entry, where most students are only casually interested in the course and take part for only a few weeks. Our model hopes to capture all types of students, and so must learn how students behavior changes from week to week in the course. This task involves discovering the relationships between many weekly variables, from the number of events in a week to the performance on problem submissions to the average time spent on resources. As these variables exist on a week by week basis, learning how each might affect the other is a challenging problem.

3.3 Multi-Level Modeling

To address the challenges we pointed out above, we defined models at different levels. In the following sections we give an overview of each model, and show how we pull them together to create a composite model of a MOOC student. In subsequent chapters we present each individual model separately.

3.3.1 Week Level Models

These models are trained on the event sequences for each individual week.

Generative model for event location sequences: We developed a model to learn how students transition from one event location to another. Such discrete time-series data is often modeled with an HMM or Markov model. For this application, we propose the *Latent-Explicit Markov Model* (LEMM), which combines the main ideas from both models. While in the Markov model, the next event is dependent on the previous event, and in the HMM the next event is dependent on the hidden state, in the LEMM the next event is dependent on both as depicted in Figure 3-2. The reasoning for this, along with the full model description and learning algorithm, is presented in Chapter 6.

Generative model for capturing problem submission behavior Since submissions in event sequences often appear consecutively, we design a problem topic model to learn groups of problems that are submitted together without a visit to a resource in between. These problem groups are learned by applying a Latent Dirichlet Allocation algorithm, treating a set of consecutive problem submissions as a *document*. The model, its motivations, and the training procedure are presented in Chapter 4.

Generative model for event descriptors: Lastly, a set of parametric models are used to generate how much time students spend on each course resource and how they perform on each problem submission. A model first learns the distribution of grades per problem and duration of time spent per URL, across

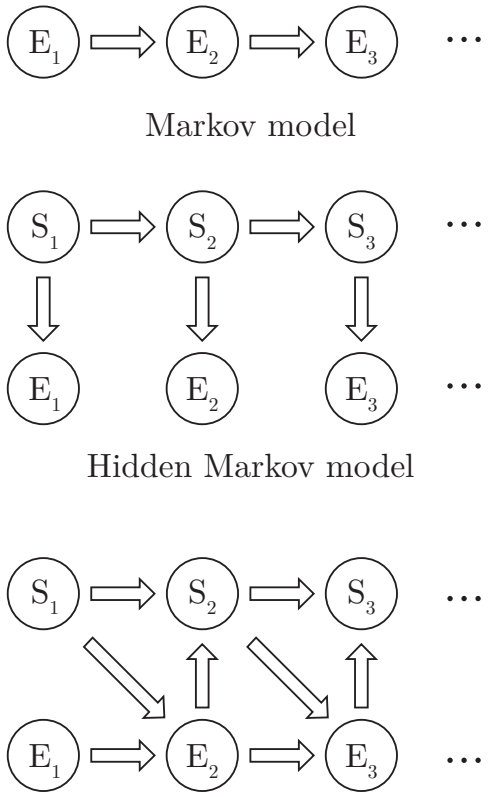


Figure 3-2: Comparison of LEMM to HMM and Markov model, by their directed graphical representation

all students. Then it estimates for each student the average percentile of his submission success rates over the submission events, and the average percentile of his observation durations over the observed events. The models, their definitions, and nuances, and the learning algorithm are presented in Chapter 5.

3.3.2 Course Level Model

In addition to learning the models on the week-by-week data, we extract week-by-week features for each student. These features are then modeled via a course level model. In particular, from the event descriptor model we will infer for each student a feature derived from his duration model and another feature derived from his assessments

model. From the LEMM model, we can extract for each student a set of counts for the hidden states. These reflect how often a student viewed a location belonging to each of the hidden states learned by the model.

These student features by week are used as a bridge between the course level and week level models. The week level features provide a brief description of a student's activity for that week. When a student's weekly features during the course are joined together, they represent his overall activity for the whole course. The course level model then learns the distribution of these full feature vectors across all the students. We use a Gaussian Mixture model to explain the distribution of these feature vectors.

3.3.3 Training Summary

The overall modeling steps are shown in Figure 3-3. The overall model training procedure is summarized in the following key steps.

1. For each week, train a problem topic model. Replace the p_{id} in the data with the problem groups ID.

2. For each week, train a LEMM on the event sequences (treating the set of URL IDs and problem group IDs as the event locations). The learned LEMM model infers the hidden states for each student and generate for each student the "hidden state counts" as weekly features.

3. For each week, train the event description models. Label each student with a feature generated from his duration model and a feature generated from his assessments model.

4. Train a Gaussian Mixture model over the concatenated feature vectors for all students, for the entire course.

3.3.4 Data generation

To generate synthetic student data, the course level model samples a set of high level features (for all weeks) from the Gaussian mixture model. By capturing the distri-

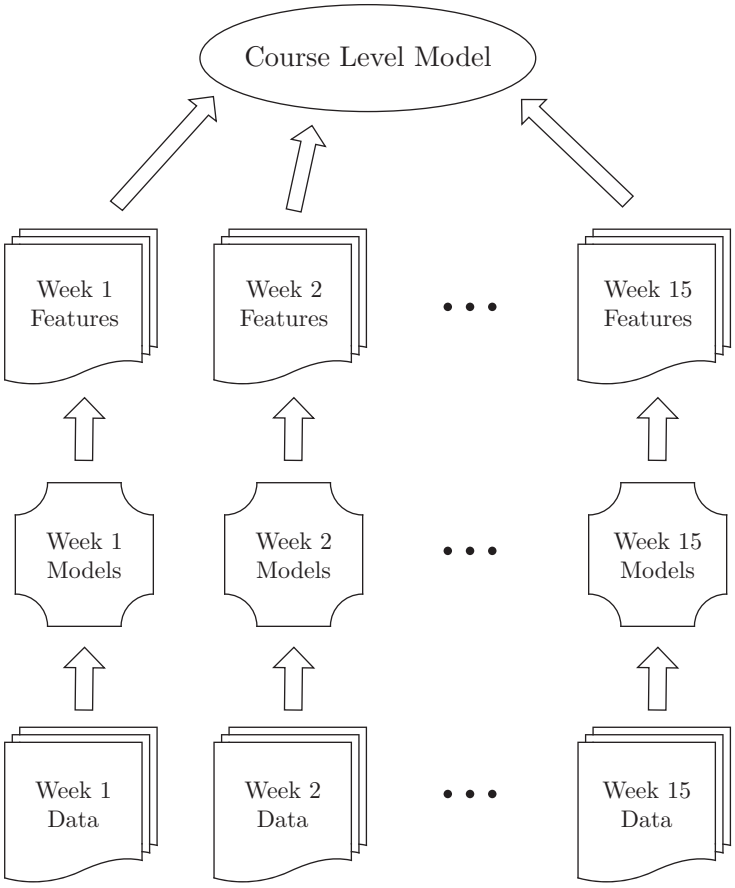


Figure 3-3: Full model training overview

bution of the full features from student to student, the course level model enables a coherent generation of week to week event sequences. These high level features are then given to the week level models for sequence generation. The week level models use these features to sample the full event sequences for that week which are consistent with the desired features.

Summary: In this chapter we presented the overview of our modeling methodology for a digital student. We presented the key challenges and highlights of our approach. In next four chapters we present details about each model separately.

Chapter 4

Problem Topic Model

The problem topic model captures how students navigate through the problems during the week. This captures what problems they submit answers for and whether or not they consult resources in between multiple problems. In particular, we consider sequences of consecutive problem submissions, made without consulting other resources in between as a group. Capturing which problems students answer with/without transitioning to other resources is important because researchers are often interested in analyzing these transitions, for example to figure out which resources were helpful in solving a problem. In order for our model to be able to generate synthetic data that exhibit such behavior, it must correctly capture the sets of problem often submitted together. Due to the way the online platform is designed, we observe three common submission patterns in the course data. These are:

Submitting for a multi-part problem. A student clicks the submit button to submit all parts of a multi-part problem at once. He could have attempted any or all of the subproblems, since there is only one submit button available, there are multiple submission events with the same timestamp.

Repeat submissions. A student clicks on the same submit button two or more times, after getting part(s) of the problem wrong. He can re-attempt the problem *without accessing a course resource in between.*

Submitting multiple problems. A student clicks two or more submit buttons

to submit parts of multiple problems, *without accessing a course resource in between.*

A screen shot of a problem submission page is depicted in Figure 4-1, showing how these three use cases are possible.

The problem submission aspect of the data needs to be separated from the rest of the time-series modeling. Whereas the order of observation events is fundamental, the exact order of the problem submissions is not meaningful when they are all submitted with the same button and have the same timestamp. The model will deduce which groups of problems are often submitted together, either because they belong to the same problem section or because they belong on the same web page and are similar enough that students answer them together. By separating the problem submissions from the rest of the sequence, the model better reflects the process by which students interact with the course - they submit a group of problems with once click, rather than go through events click-by-click as with observation events. To fit this model in our time-series framework, we have the LEMM model learn the problem group ID. The problem topic model, described in this chapter, learns which problems are submitted given the group.

As with the other week level models, we build this model for each week of the course. For the rest of this chapter, the entities involved are specific to one particular week.

4.1 Model Definition

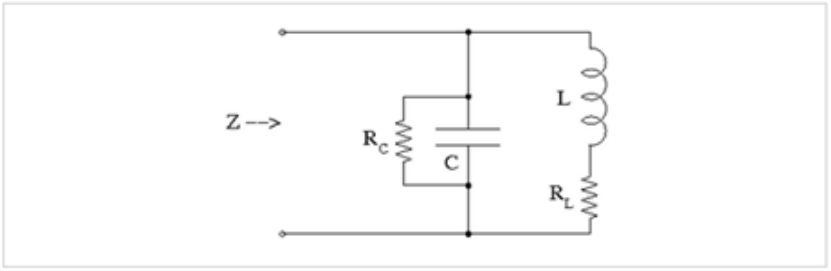
Consider the set of commonly answered problems for the week to be $p \in 1, 2, 3, \ldots P$. Our goal is to learn a set of problem groups, or *problem topics*, that are submitted together. A problem topic could be represented with a P dimensional vector such as:

.21	.0	.31	...	.14	.0

where i^{th} component is the probability that any problem drawn from this topic is

H11P1: LC TANK

Parallel resonant "tank" circuits are common in radio equipment. But unfortunately there is always resistance that prevents them from being perfect: in every real inductor, the wire that makes up the inductance has some resistance, and there may be leakage in the capacitor that can be modeled as a resistance. Also, the Norton resistance of the system connected to the tank circut looks like a leakage through the capacitor. So a realistic model for a tank circuit is the following:



In the space provided below write an algebraic expression in terms of the device parameters for the bandwidth $\Delta\omega$ of the impedance Z looking into this tank circuit.

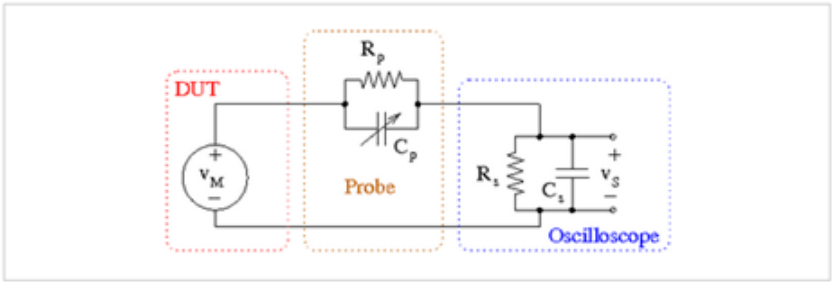
The antenna tank of a Graymark 536 Radio Kit has an inductance $L \approx 0.65$mH. The resistance of the inductor $R_L \approx 4.0\Omega$. The equivalent resistance across the capacitor is $R_C \approx 490.0\text{k}\Omega$. The capacitor is variable, for tuning.

If we tune to a station at $f = 860.0$kHz what is the capacitance, in picoFarads, of the tuning capacitor?

Check

H11P2: SCOPE PROBE

We can model the interaction of an oscilloscope with the device under test (DUT) with the following circuit.



The oscilloscope is used to measure and display waveforms at many frequencies. Suppose that $V_M = 0.04\cos(2\pi\ 120.0 \times 10^6\ t)$ V. What is the value, in picoFarads, that C_p should be set to so that $V_s = 0.004$V?

Suppose now that $V_M = 0.8\cos(2\pi\ 171.0 \times 10^6\ t)$ V. What is the value, in picoFarads, that C_p should be set to so that $V_s = 0.08$V?

Check

Figure 4-1: Snapshot of homework page. The two problems have two subparts. The student may submit answers to H11P1 by clicking on the check button under it. When trying to check, he may or may not have attempted both the subparts.

problem i. Since this vector represents a probability vector, its component sum is 1.

We note that one such problem topic cannot capture all the possible groupings based on the submissions from all the students. We instead model the submissions

as coming from T such groups, where each group j is represented with its own P dimensional probability vector

$$\phi_j \in \mathbb{R}^P, j = 1, 2, ...T$$

where ϕ_j is the probability vector corresponding to the j^{th} topic. These probability vectors are the parameters of the model. The question of when a student makes submissions from each problem group is learned by the event location model, to be discussed in Chapter 7.

To learn these T groupings, we draw a parallel between learning how problem submissions co-occur in groups and the classical methodologies of *topic modeling* that model the co-occurence of words in a document. As a result of this mapping, we are able to use a technique called Latent Dirichlet Allocation (LDA) for learning our model parameters.

In the next section we give an overview of the Latent Dirichlet Allocation algorithm as it is applied in language modeling. We then present how we map our problem to this modeling technique. Section 4.2 presents the learning and inference approach. Section 4.3 presents the results of our learning.

4.1.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation posits that in a set of documents, each document may be viewed as containing a mixture of topics. The distribution of topics θ is a Multinomial distribution represented as $\theta \in \mathbb{R}^J$. Given a corpus of words W, each topic has a word distribution given by ϕ_j, a Multinomial distribution of dimension $|W|$. The model assumes a Dirichlet prior on both θ and ϕ and a particular process by which documents consisting of words are formed [2]. A set of documents each of length n is generated as follows:

1. Sample a distribution over topics $\theta \in \mathbb{R}^J$ for each document from the Dirichlet prior $\text{Dir}(\alpha)$.

2. Sample per-topic word distributions, $\phi_j \; \forall j$ topics from the Dirichlet prior Dir(β). These are shared across documents.

3. To generate the n words for each document, follow these steps:

   For $k = 1, 2, ...n$ do

   $$\text{Sample a topic } j_k \sim Mn(\theta),$$
   $$\text{Sample a word } w_k \sim Mn(\phi_j),$$

   where Mn represents the Multinomial distribution.

This process follows a bag-of-words model where the order of words do not matter.

4.1.2 Application of LDA

So how can we use LDA to learn groups of problems? It turns out that there is a strong parallel between the problem submissions setting and the documents of words setting. If we consider the following mapping

$$\text{words } (W) \rightarrow \text{problems } (P)$$

$$\text{documents} \rightarrow \text{consecutive submissions}$$

$$\text{topics of words} \rightarrow \text{topics of problems}$$

then we have a way to apply LDA. Treating each consecutive sequence of submissions as a document, and each problem as a word, we arrive at the following interpretation of how students select which problems to make submissions for.

First, a student selects a single topic or group of problems (in contrast to the LDA model). Each topic corresponds to a group of problem commonly submitted together (either from the same problem section or not). Then the student repeatedly selects a new problem to answer according the topic probabilities, also following a bag-of-words model. The number of submissions the student makes in this problem group

depends on per-topic parameters to be later defined and learned by the model. This process allows a student to answer the question multiple times in the same group, which reflects the situation where he submits wrong answers. This LDA interpretation yields a process that can reasonably explain how a student makes their selection of submissions.

4.2 Model Training

1. **Assemble the problem documents across all students.** For each student's event sequence, group the submission events that are not separated by any observation event. Each such group of consecutive submission events are considered a document. Index the documents $m = 1, 2, ...M$.

2. **For each choice of the number of topics $T \in [T_{min}, T_{max}]$, do:**

3. **Learn the LDA parameters on the documents D_m.** These parameters are the set of T problem topics

$$\phi_j \in R^P, j \in [T]$$

   There exist established algorithms to optimize the LDA parameters. For this project we use a library that performs collapsed Gibbs sampling.

4. **Evaluate the log likelihood of the model.** The log likelihood score of a document of length n given the topic parameters ϕ_j and α is

$$\int p(\theta|\alpha) \log \left(\prod_{k=1}^{n} \sum_{j=1}^{T} p(w_k|\phi_j) p(\phi_j|\theta) \right) d\theta$$

   such that the prior distribution of θ is $\text{Dir}(\alpha)$.

5. **Select the final choice for T.** Use the best computed log likelihood score to pick an approximately optimal value for T, the number of topics.

6. **Learn the lengths of the documents**. Steps 1 - 4 set up the data and learn the LDA parameters. This step is needed in addition to LDA in order the learn the average lengths of documents, so that the model can generate synthetic data.

   For each topic ϕ_j, define s_j as a length parameter for topic j, interpreted as the average length of a document weighted by its composition of topic j. The LDA algorithm estimates the latent topic distributions per document $\theta_m, m \in [M]$. We then estimate the parameter s_j as

   $$s_j = \frac{\sum_{m=1}^{M} p(j|\theta_m) \cdot |D_m|}{\sum_{m=1}^{M} p(j|\theta_m)}$$

   where $|D|$ is the number of problems in document D.

7. **Sample problems for a new document, given topic** j. Sample a document length l uniformly from the range $[\frac{s_j}{2}, \frac{3s_j}{2}]$. For $i = 1, 2, ... \ l$, sample a problem from $Mn(\theta_j)$.

As a last step we specify how this model interacts with the event location model. For **training the full model**, we perform the following steps.

1. Train the problem topic model by applying LDA as described.

2. For each document D_m, use the learned θ_m to identify the most significant topic j_m by probability.

3. For each original sequence of event locations, replace the sets of consecutive problem submissions (documents D_m) by a single submission event with problem group ID j_m.

The simplification to represent a document by a single topic is demonstrated to be reasonable by the data. When we compute for each document D_m the most significant topic j_m by probability, the average probability of this most significant topic is .92.

4. Train the event location model to learn the sequence of URL IDs and problem group IDs.

   The model makes the assumption that the *topic ID* of the problems answered is important to the time-series data. The particular *problems* submitted are assumed to be not important given their topic ID. Furthermore, the simplification in step 3 is necessary because the event location model needs discrete elements as input data - in this case, the problem topic ID.

To **sample synthetic event location sequences**, we

1. Use the event location model to generate the sequences of event locations and topic IDs.

2. For each proble topic, use the problem topic model to sample problems according to ϕ_j and s_j.

4.3 Model Results

As mentioned, we test our models on the fall 2012 offering of 6.002. For this course, there was an average of 63 commonly solved problems per week that make up the weekly problem corpus for the model. About 59% of students attempted at least one problem, and among these students they averaged 15 submissions per week.

As described, from the event sequences in each week the algorithm composed the groups of submissions, or documents to be learned by LDA. The LDA training procedure typically found that between 8 and 15 problem topics per week was optimal. Each topic on average consisted of 6 problems that had a probability at least .25 of being sampled when the topic occurred. The log-likelihood per problem of the data

using our model is shown in Figure 4-2, compared to the entropy of the problem distribution. If $n_p, p = 1, 2, ...P$ are the counts of the submissions of problem p, and n is the total count of all problem submissions in a particular week, the entropy is defined as

$$\sum_{j=1}^{P} \frac{n_j}{n} \cdot \log \frac{n_j}{n}$$

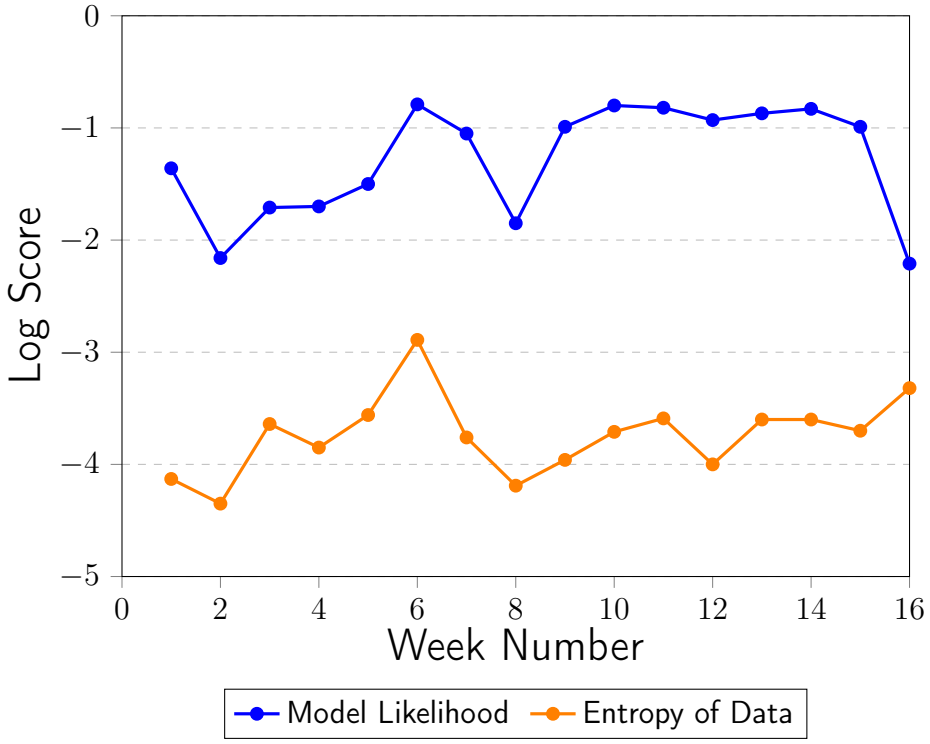


Figure 4-2: Problem topic model performance. For all weeks, the model easily outperforms a baseline model (entropy), on an average by 66%.

The scores suggest that the model performs extremely well in fitting the data. To give an idea of what problem topics are learned, we present the following sample of topics from the 15th week of the 6.002 course in Table 4.1.

In this sample, topics one and three correspond to one section of problems that are submitted with the same button. But topics two and four show groups of problems that some students submit together in one go without making an observation event in between.

Topic	Problem Name	Problem Part
1	S26E3_A_Hot_Processor	1
	S26E3_A_Hot_Processor	2
	S26E3_A_Hot_Processor	3
2	S24E3_Inverting_Amplifier_Generalized	1
	S24E3_Inverting_Amplifier_Generalized	2
	S24E4_Generalization_to_impedances	1
	S24E5_Generalization_to_nonlinear_elements	2
3	S25E1_Positive_Feedback_Gain	1
	S25E1_Positive_Feedback_Gain	2
	S25E1_Positive_Feedback_Gain	3
4	S24E1_Summing_Amplifier	2
	S24E1_Summing_Amplifier	3
	S24E1_Summing_Amplifier	4
	S24E2_Difference_Amplifier	3
	S24E2_Difference_Amplifier	4
	S24E2_Difference_Amplifier	5
	S24E3_Inverting_Amplifier_Generalized	1
	S24E3_Inverting_Amplifier_Generalized	2

Table 4.1: Sample topics from week 15. Note we only display four of the 12 topics learned for that week, and only the main problems in each topic.

Chapter 5

Generating Event Descriptors

Once the event identities are modeled, the remaining information of interest are the durations for the observation events and the assessments, or grades, for the submission events. An event description model learns how these (discrete) fields are distributed across individual problems and URLs and across students. These fields measure or estimate how long students spend on each URL, and how successfully they've learned the material by problem.

Specifically, the data to be modeled here are the complete event sequences, where one sequence $E_j = e_1, e_2, ...e_k$ has elements which contain all five fields described in chapter 3. For each event, given the URL ID or problem ID, we want to predict the durations and grades. Each duration is an integer number of seconds in the range $[0, 1440]$ and each grade is either correct or incorrect.

5.1 Model Considerations

Here are a number of targets to design the model for.

1. Week by week generative model. Again it is sensible to train a model for each week, when different URLs are commonly viewed and different problems are answered. A generative model is needed to sample new durations and grades in addition to predicting them.

2. Captures inter-resource variability. Some URLs have video lectures, others are brief transition webpages. Similarly, there are harder and easier problems. The model should learn the variances in how each URL or problem affects durations and grades.

3. Captures inter-student variability. The model should learn the variances in how long each student spends on viewing resources and how accurately each student submits answers. The model should also be able learn for each student a feature for durations and a feature for assessments, so that student durations and grades can be characterized across weeks using the course level model.

5.2 Modeling Durations Spent on Course Resources

Training the model for labeling durations is outlined below. The algorithm runs on the event sequences from one week at a time.

1. **Assemble data.** For all pairs of students $i = 1, 2, ...S$ and URL IDs $u = 1, 2, ...U$, calculate

   $d_{i,u}$: the average time spent by student i on URL u (across all visits by that student to that resource).

2. **Estimate probability distributions of durations for each resource.** Let $d|u$ denote the random variable of durations spent by students on URL u. Then the cumulative (discrete) probability distribution function is estimated from the observed data as

$$F_{d|u}(d) = \frac{1}{S} \sum_{i=1}^{S} I(d_{i,u} \leq d)$$

   where $I(\cdot)$ is the indicator variable. A non-parametric approximation to this distribution (for memory and runtime purposes) can be attained by computing the deciles of the distribution.

3. **Compute duration percentiles**. For a student i and URL u pair, find the percentile value of the time spent by i on resource u, relative to other students. This percentile $f_{i,u}$ is calculated as

$$f_{i,u} = F_{d|u}(d_{i,u})$$

In practice, look up where $d_{i,u}$ lies on the approximation for $d|u$.

4. **Model the distribution of percentiles by student**. For a student i, assemble the observed duration percentiles $f_{i,u}$ values for all resources u. Suppose that the random variable $f|i$ is drawn from a normal distribution

$$f_{i,u} \sim N(r_i, \sigma)$$

where r_i is the mean of the distribution, a parameter specific to i. The maximum likelihood estimate for r_i is computed as

$$r_i = \frac{1}{|U_i|} \sum_{u \in U_i} f_{i,u}$$

where U_i is the set of URLs that student i visited that week. σ is taken as constant across all students, and is estimated as the average standard deviation among $f|i$.

$$\sigma = \frac{1}{S} \sum_{i=1}^{S} \sqrt{\frac{1}{|U_i|} \sum_{u \in U_i} (f_{i,u} - r_i)^2}$$

5. **Sample a duration for student i on URL u.** Given r_i, the percentile of the duration for the event is sampled as

$$f \sim N(r_i, \sigma)$$

and the duration for the event is

$$F_{d|u}^{-1}(f)$$

In this model, we aim to recognize the effects of both URL ID and student on event duration. First it learns the cumulative probability distribution of durations as $F_{d|u}(d)$ for URL u. Each student i is assumed to view resources for time periods at a similar percentile for each resource. For each URL u the student visits, the duration percentile $f_{i,u}$ is assumed to be drawn from the normal distribution $N(r_i, \sigma)$. Thus his parameter $r_i \in [0, 1]$ describes how long he tends to view URLs relative to other students. The duration is then $F_{d|u}^{-1}(f_{i,u})$.

Once trained, the student parameters r_i are given to the course level model, which learns the distribution for these parameters in relation to the other low level parameters across weeks. To sample new labels, the event description model must be given the resource ID u (by the event ID model) and the student parameter r (by the course level model). To generate a label for each event, the description model samples the duration percentile given r, looks up the distribution $F_{d|u}(d)$, and computes the new duration using the distribution and the desired percentile.

5.3 Modeling Assessments

The training procedure for the assessments model similarly computes the distribution of submission results per problem and the average ranking of each student over his submissions.

1. **Assemble data.** For all pairs of students $i = 1, 2, ...S$ and problem ids $p = 1, 2, ...P$, calculate

$c_{i,p}$: whether i got problem p correct on any of his submissions (1 if yes else 0)

$a_{i,p}$: the number of submission attempts by student i on problem p

The estimated success rate $s_{i,p}$ of student i on problem p is

$$s_{i,p} = \frac{c_{i,p} + 1}{a_{i,p} + 2}$$

which is a Bayesian estimate for the success to be explained later.

2. **Estimate probability distributions of success rates for each problem.** Let $s|p$ denote the random variable of success rates of all students on problem p. Then the cumulative probability distribution is estimated from the observed data as

$$F_{s|p}(s) = \frac{1}{S} \sum_{i=1}^{S} I(s_{i,p} \leq s)$$

A non-parametric approximation to this distribution can again be attained by computing the deciles of the distribution.

3. **Compute success rate percentiles.** For a student i and problem p pair, find the percentile value of the success rate of i relative to other students. The relative ranking $f_{i,p}$ is

$$f_{i,p} = F_{s|p}(s_{i,p})$$

In practice, look up where $s_{i,p}$ lies on the approximation for $s|p$.

4. **Model the distribution of percentiles by student.** For a student i, assemble success rate percentiles $f|i$, the $f_{i,p}$ values for all problems p. Suppose they are drawn from a normal distribution. In particular,

$$f_{i,p} \sim N(g_i, \sigma)$$

where g_i is the mean of the distribution, a parameter specific to i. The maximum likelihood estimate for g_i is computed as

$$g_i = \frac{1}{|P_i|} \sum_{p \in P_i} f_{i,p}$$

where P_i is the set of problems that student i answered that week. σ is a constant across all students, and is estimated as the average standard deviation among $f|i$.

$$\sigma = \frac{1}{S} \sum_{i=1}^{S} \sqrt{\frac{1}{|P_i|} \sum_{p \in P_i} (f_{i,p} - g_i)^2}$$

5. **Sample a grade for student i on problem p.** Given g_i, the percentile of the success rate for the event is sampled as

$$f \sim N(g_i, \sigma)$$

and the success rate for the problem is

$$F_{s|p}^{-1}(f)$$

Finally, the problem submission is labeled correct with probability equal to the success rate.

For modeling assessments, each problem p has a cumulative probability distribution $s|p$ over the success rates of students. A student that has a success rate s of solving a problem will solve the problem correctly on each attempt with probability s. Similar to the duration model, each student i has a parameter $g_i \in [0, 1]$ that

describes his relative accuracy at solving problems. For each problem he attempts to solve, his success rate relative ranking f is assumed to be drawn from the normal distribution $N(g_i, \sigma)$ for some constant σ, and his success rate is $F^{-1}_{s|p}(f)$. Then the parameter g_i can similarly be interpreted as the average percentile that the student correctly answers questions at.

Learning the appropriate parameters for the assessment model is a more challenging task than for the durations because the true success rate r for each student-problem pair is not observed directly. The estimation for success rate $s_{i,p} = \frac{c_{i,p}+1}{a_{i,p}+2}$ comes from the posterior expected value of a student's success rate, given that the success rate is a priori believed to be uniformly likely between 0 and 1. In fact this prior distribution is supported by the training data, that the distribution of success rates over student problem pairs appears quite close to uniform for most problems.

Just as with the duration model, the assessments model labels each student with a parameter that is given to the course level model to learn the relationships between this feature and other student features. In the synthetic data generation process, the assessments model is asked to assign a grade given the problem p (from the event ID model) and the sampled student parameter g (from the course level model). The success rate percentile is sampled using g, and then the success rate is computed using $s|p$.

5.4 Model Discussion

Note that in this model the descriptions (durations or grades) are not assumed to be normally distributed per event ID. Instead, the model assumes that the "relative ranking" of a description is normally distributed with respect to the student's parameters. For each resource or problem, the description distributions are estimated non-parametrically, as enough students have accessed these events to make this estimation meaningful.

One further direction of research for modeling descriptions beyond this paper is to analyze students' tendencies for durations and grades over groups of webpages or

problems. Since the parameters d and g applies for a student across all resources and problems, the model assumes that durations and grades depend only on the student and not on the particular resource or problem, relative to the overall population. That is, students don't spend a longer time viewing any particular group of resources and don't have more success on any group of problems, given individual parameters. This might be an oversimplification, since some students might spend relatively longer on certain topics or answer better on certain problems, according to his interests or academic strengths.

It would be interesting to research how other factors affect how well students do on particular problems or how long they spend on certain resources (for example, how many observation events a student has or which other resources a student visits). However, that may be a challenging problem because per week each student may not visit enough resources or attempt enough submissions to yield useful information on smaller sets of resources or problems.

5.5 Model Results

The model performs reasonably well when trained and tested on the 6.002 course in fall of 2012. To gather experimental results, we train the models on each week of the course, observe the distribution of the durations or grades per event, and compute the errors between predicted durations or grades (the mean of the estimate normal) and the actual fields. The following discusses what the duration and assessment fields look like in the data and the fit of our model.

5.5.1 Durations

The average duration by resource is distributed according to the following figure. Among resources visited by at least 500 students, "Static_Discipline_and_Boolean_Logic /3/video/S4V3_Why_Digital" had the longest average duration with an average of 577 seconds and "Overview/edx_introduction/3/problem/ex_practice" had the shortest at 5 seconds.

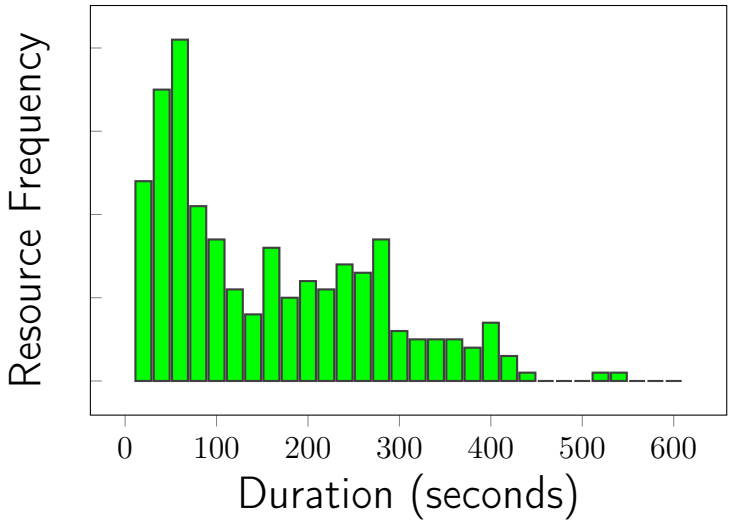


Figure 5-1: Distribution of average durations by resource. The distribution is right skewed with the most common average duration of around 60 seconds.

Resource Name	Duration (s)
Static_Discipline_and_Boolean_Logic/3/video/S4V3_Why_Digital	577
Incremental_Analysis/5/video/S7V5_Incremental_Method_Insight	540
Inside_the_Gate/5/video/S5V5_Switch_Model	522
Dependent_Sources_and_Amplifiers /3/video/S8V3_Example_Dependent_Source_Circuit	508
Inside_the_Gate/19/video/S5V16_Inverters_Based_on_SR_Model	494

Table 5.1: Resources with longest average duration

Resource Name	Duration (s)
Overview/edx_introduction/3/problem/ex_practice_limited_checks	5
Week_2_Tutorials/4	7
Week_2_Tutorials/6	7
Overview/edx_introduction/3/problem/ex_practice_limited_checks_3	8
Circuits_with_Nonlinear_Elements/17	9

Table 5.2: Resources with shortest average duration

We compare the performance of the description model at predicting durations compared to baseline predictors that only consider the identity of the URL or student. The two baseline predictors estimate the mean duration for each resource or for each student. The metric to evaluate the predictors are mean squared error (MSE) and mean absolute error (MAE).

Predictor	MSE	MAE
By Student	318	144
By Resource	301	119
Description Model	267	89

Table 5.3: Duration model performance. In both metrics, our description model beats the baseline (smaller errors are better).

5.5.2 Assessments

We similarly present the results from the assessments model. Problem difficulties (success rates) are displayed below, with "H10P3_An_L_Network/6/1/" being the most challenging problem attempted by at least 500 students at 2.0% accuracy across all submissions and "Q3Final2012/2/1/" the easiest problem at 97.1% accuracy.

Problem Name	Problem Part	Accuracy
H10P3_An_L_Network	5	2.0%
H10P3_An_L_Network	6	2.1%
H5P1_Zero-Offset_Amplifier	3	2.9%
H8P3_Memory	5	3.1%
H10P2_New_Impedances	7	3.3%

Table 5.4: Most difficult problems

The distribution of student assessment parameters is shown below in Figure 5-3. This distribution might be expected to be approximately uniform, because the

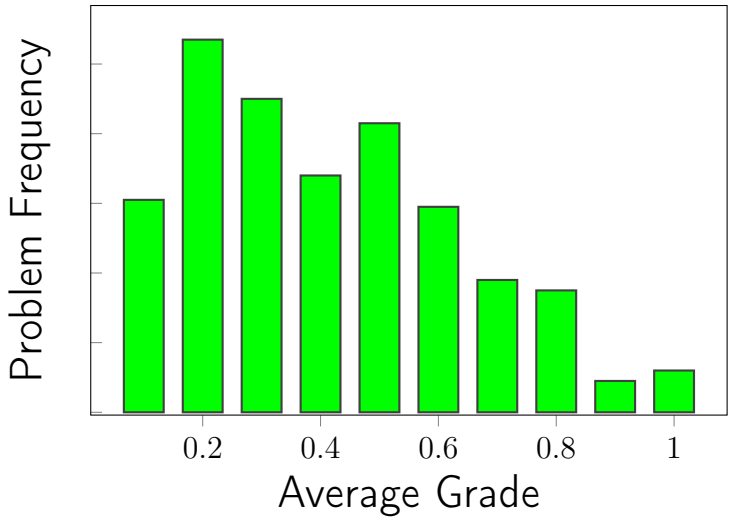


Figure 5-2: Distribution of grades by problem. Average grades are fairly well spread out between 0 and 1, with an overall average a little below .5.

Problem Name	Problem Part	Accuracy
Q3Final2012	1	97.1%
Sample_Numeric_Problem	1	96.5%
Sample_Algebraic_Problem	1	92.5%
S2E1_Circuit_Topology	1	92.4%
H7P1_Series_and_Parallel_Inductors	1	92.3%

Table 5.5: Least difficult problems

interpretation of the parameter is the typical ranking of each student by problem. However, it turns out that more students consistently answer questions at a lower ranking than at a higher ranking, leading to average performances that have fewer top students.

Finally, the assessment model also compares well to baseline predictors that only predict the mean grade for each problem or for mean grade for each student. The log-likelihood metrics are calculated from the probability of seeing the grade per submission attempt, according to the predicted success rate for that student on that problem. The comparison of the assessment with the baseline predictors is shown in Table 5.6.

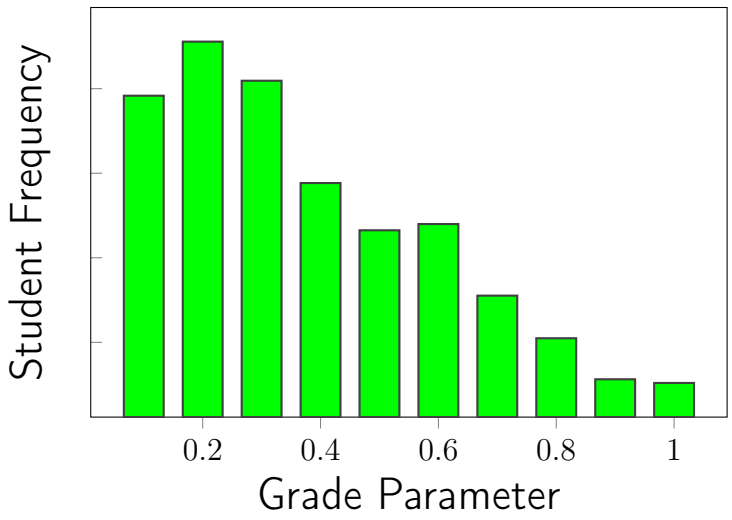


Figure 5-3: Distribution of grades by student. The distribution is somewhat right skewed, with more students having a low grade parameter.

Predictor	Log-Likelihood	Cross Validation
By Student	-.41	-.42
By Problem	-.43	-.43
Description Model	-.36	-.37

Table 5.6: Assessment model performance. Our model beats the baseline in both metrics as expected.

Chapter 6

Event Location Model (Latent-Explicit Markov Model)

6.1 Model Motivations

Having detailed how we handle issues such as grouping problems and describing events, we turn to the core problem of modeling the sequences of event locations, the URL IDs (the u_{id}) and the problem groups (groups of p_{id}). Intuitively, student click-stream data contains both local and global properties. At a local level, students typically transition from one event to another by clicking on the available links, or by interacting with different components of a webpage in a certain order. This behavior is precisely described by a Markov model,, which are commonly used in web-based systems.

On a global level, student behavior is affected by his own interests, background, goals, and previous interaction with the course. That is, students tend to focus on certain groups of links; for example, one student might read a lot of forum posts, another may frequently watch video lectures, while yet another may read lectures notes and make many problem submissions. So there exist factors that determine the event locations at a higher level.

Once again, we can draw a parallel between the student event data and text documents. Whereas documents contain sequences of words, student click-stream

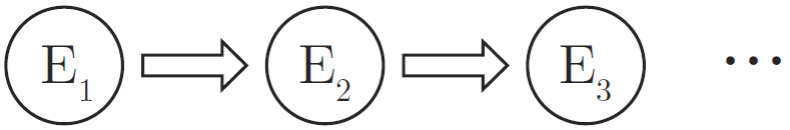


Figure 6-1: Markov model directed graph

data contain sequences of event locations. Documents also have similar *local* and *global* properties, with local word patterns largely determined by grammar and phrase sequences, and global word patterns influenced by what topic(s) the document is written about. It turns out that two major concepts from document modeling apply to this data - n-grams and topic modeling.

N-grams in documents provide a statistical count of observed short sequences, and in fact 2-gram models are essentially Markov models. For click stream data, 2-grams, or Markov models, seem appropriate as a student's next action depends largely on what resource on the website he previously observed. A directed graphical diagram for Markov models can be drawn as in Figure 6-1.

Topic modeling (for example by LSA, LDA, or NMF), the second major idea in document modeling, reveals groups of words that often appear together, or in the case of student data, resources or problems that appear together. However, topic modeling usually assumes a bag-of-words model where the order of the words is ignored. Conversely, event sequences provide an order of event locations that can aid in the discovery of topics, as students tend to observe one topic at a time before switching to another. A new approach to topic modeling for our data could be effective.

Hence we propose using a Hidden Markov Model (HMM) to capture the topic transitions. We introduce hidden states in the graphical model, such that each hidden state represents a topic of events, as shown in Figure 6-2.

The emission probabilities at each state correspond to the probabilities of the associated problem topic. Instead of topics being drawn independently for each new word or event, the next topic is drawn from a distribution or topic, conditional upon the previous topic (in general, more likely to stay the same topic). In addition, the

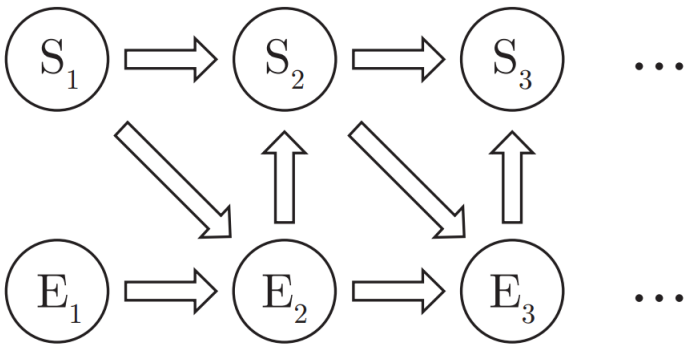


Figure 6-2: Proposed directed graphical model

topic the student views next (may be the same as the current) depends on both the previous topic and which event location he is currently viewing.

Under this interpretation, a student, depending on their topic preferences, views topics in groups, and the exact event locations they select depend on both the topic and the previous event location. To relate back to the original works on topic modeling, we can use an existing topic modeling technique to provide a good first estimate for the event topics, or the hidden states in the HMM. Then the HMM training procedure can refine the topic distributions.

It turns out that this merged Markov model-HMM structure we investigate is only slightly better than a simple Markov model. However, another major contribution of the merged model is that it can infer a distribution over the hidden states for each event sequence. This provides valuable feature descriptions for each sequence that makes week by week continuations of sequences coherent.

6.2 Model Definition

A fully descriptive model of the event location sequences combines a Markov model with an HMM. We call this model, the key new construct of the thesis, a *Latent-Explicit Markov Model* (LEMM). It is a variant in the family of Markov models where the output depends on both a *latent* factor (the hidden state) and an *explicit* factor (the previous observed output). The data modeled in this section is the set of

student sequences where sequence $e_1, e_2, ...e_n$ is made up of event locations. The set of event locations L is the union of the set of URL IDs, the set of problem topic IDs, and the singleton set of the session break event. The LEMM is a directed graphical model defined by the following components:

1. L, set of output event locations
2. S, set of hidden states or topics
3. T, a transition function. $T(s_i, e_{i+1}, s_{i+1}) = Pr[s_{i+1}|s_i, e_{i+1})]$
4. O, an output function. $O(s_i, e_i, e_{i+1}) = Pr[e_{i+1}|s_i, e_i)]$
5. P_0, the initial joint state, event location distribution over $S \times L$

The hidden states in S correspond to the topics or sets of resources / problem groups that tend to be observed together.

An instance of a student sequence is initialized with the first output event and the first hidden topic. The next event is generated from the current event and hidden topic, and then the next hidden topic is generated from the new event and current topic. The process repeats to produce the entire event sequence. Mathematically, the probability of a sequence of events and states is given by

$$\begin{aligned}
&Pr[e_1, e_2, ...e_n, s_1, s_2, ...s_n] \\
&\quad = Pr[e_1, s_1] \cdot \prod_{i=2}^{n} Pr[e_i, s_i|e_{i-1}, s_{i-1}] \\
&\quad = Pr[e_1, s_1] \cdot \prod_{i=2}^{n} Pr[e_i|e_{i-1}, s_{i-1}] \cdot Pr[s_i|e_i, s_{i-1}] \\
&\quad = P_0(e_1, s_1) \cdot \prod_{i=2}^{n} O(s_{i-1}, e_{i-1}, e_i) \cdot T(s_{i-1}, e_i, s_i)
\end{aligned}$$

where we use the definition of O and T as the output and transition functions,

respectively. The probability of the sequence itself is

$$Pr[e_1, e_2, ...e_n] = \sum_{s_1, s_2, ...s_n} Pr[e_1, e_2, ...e_n, s_1, s_2, ...s_n]$$

taking the sum over all possibilities of the hidden states. This directed graphical model is similar to that of an HMM, but now the output emissions and state transitions are dependent on both the most recent emission and state. This is justified by the Markov principle that the next output is generated strongly conditioned on the last output.

The transition and output functions are structured to reduce the number of parameters, by decomposing them into independent functions over the current state and over the current event. That is, we assume

$$T(s_i, e_{i+1}, s_{i+1}) \sim T_1(s_i, s_{i+1}) \cdot T_2(e_{i+1}, s_{i+1})$$

$$O(s_i, e_i, e_{i+1}) \sim O_1(s_i, e_{i+1}) \cdot O_2(e_i, e_{i+1})$$

Further, the functions T_1 and O_1 map to exactly the HMM transition and output functions, which specify the distribution of transitions to s_1 and the emissions of e_1 from a given state s_0, for all combinations of s_0, s_1, and e_1. They can be any joint discrete probability functions over the spaces $S \times S$ and $S \times L$ respectively.

The function O_2 is exactly the Markov transition function between events, calculated as the ratio of the number of transitions from e_0 to e_1 to the number of appearances of e_0 for all pairs e_0, e_1. Lastly, the function T_2 outputs the conditional probability of transitioning to s_0 given event e_0. By Bayes rule, we derive

$$\begin{aligned} Pr[s_0|e_0] &= \frac{Pr[s_0] \cdot Pr[e_0|s_0]}{Pr[e_0]} \\ &\sim Pr[e_0|s_0) \\ &= O_1(s_0, e_0) \end{aligned}$$

In the second step we assume no prior knowledge of states. Therefore, learning O_1 immediately yields T_2.

6.3 Model Training

The Latent-Explicit Markov Model is constructed so that it can be trained similarly as an HMM, via an Expectation Maximization (EM) algorithm. There are four sets of parameters to train for - T_1, O_1, T_2, O_2, given a set of training sequences

$$\{E_k = [e_{k,1}, e_{k,2}, ...e_{k,n_k}] | k = 1, 2, ...m\}$$

In this notation there are m sequences, one per student for that week. The k^{th} sequence has n_k events and the i^{th} event of sequence k is $e_{k,i}$.

O_2 is learned by counting transitions just as with a Markov model. T_1, O_1 (and therefore T_2) are optimized using the EM algorithm as with an HMM. The algorithm estimates the distribution of hidden states, $Pr[S_{k,i} = s|E_k]$, and uses these to find maximum likelihood estimators for the transition function T_1 and emission function O_1. Finally, T_2 is easy to deduce given O_1.

As with the other models, the LEMM is trained on a week-by-week basis.

6.3.1 Markov Training

A training algorithm should first estimate the Markov component parameters, as these don't change during the HMM training process. Let $C(e_0)$ denote the number of times event e_0 appears in the set of sequences $\{E_k\}$ and $C(e_0, e_1)$ denote the number of times the consecutive pair e_0, e_1 appears in $\{E_k\}$, for any pair of event locations e_0, e_1. Then a smoothed Markov function would be

$$O_2(e_0, e_1) = \frac{C(e_0, e_1) + \gamma}{C(e_0) + \gamma \cdot |L|}$$

where γ is some small constant. This Markov formulation turns out to overly favor the most common transitions among the student sequences. That is, sequences generated by this model would generally output events that are common to many students. However, in reality the majority of students investigate some less common events according to his own preferences. Hence we use a modified Markov model that redistributes transition probabilities toward less popular events, to better capture each student's diversity of events. The modified output function is

$$O_2(e_0, e_1) = \frac{\frac{C(e_0,e_1)}{C(e_0)} + \gamma}{\left[\sum_{e \in L} \frac{C(e_0,e)}{C(e)}\right] + \gamma \cdot |L|} \sim \frac{C(e_0, e_1)}{C(e_0)} + \gamma$$

The value for γ is chosen by optimizing for log-likelihood on a held-out test set of data.

6.3.2 HMM Training

Next, we detail an EM algorithm to learn the parameters for the overall model. The HMM training outline is as follows, which is essentially the same as for the standard HMM. Only the exact form of each step differs in implementation. For simplicity, we consider from this point on only a single sequence $e_1, e_2, ...e_n$ as the given data, but the process applies naturally to multiple sequences.

1. Initialize HMM parameters $T_1 : S \times S \to R$ and $O_1 : S \times L \to R$ using LDA, as described below.
2. (E Step) Compute the probability distributions for the hidden states

$$Pr[s_i = s | T_1, O_1]$$

for $s \in S, i = 1, 2, ...n$ (using a forward-backward algorithm).

3. (M Step) Recompute the parameters T_1 and O_1 given the hidden state probability distributions to optimize the log-likelihood

$$Pr[e_1, e_2, ...e_n] = \sum_{s_1, s_2, ...s_n} Pr[e_1, e_2, ...e_n, s_1, s_2, ...s_n]$$

4. If the new parameters are "close" to the previous parameters, stop. Else go back to step 2.

Each step is explained in detail below.

Initialization of HMM parameters.

As mentioned, the hidden states for the model correspond to the topics that students might be interested in. Therefore, training an LDA model on the sequences provides a good estimate of the topics or hidden states in the model. Specifically, we take each sequence to be a document, and the event locations to be the words in the LDA model. Learning the topics over the event locations, given all of the events sequences, yields the initial hidden state emissions for the HMM. This provides a reasonable first guess for the hidden states in the model, and makes it likely that the HMM training will converge to a solution that agrees somewhat with the LDA inference of the topic contents.

E Step.

In the estimation step, the distribution of hidden states for each step is deduced as the following modification to the forward-backward algorithm. Let

$$\alpha[i, s] = Pr[e_1, e_2, ...e_t, S_i = s|\hat{\theta}]$$

for all states s where $\hat{\theta}$ denotes the estimated parameters so far. The algorithm calculates all values of α using the "forward" algorithm or a dynamic programming algorithm. We initialize for all s

$$\alpha[1, s] = P_0(s)$$

Then, for $i = 2, 3, ...n$ and all s the values for α are calculated as

$$\alpha[i, s] = \sum_{s'} \alpha[i-1, s'] \cdot O(s', e_{i-1}, e_i) \cdot T(s', e_i, s)$$

which is a sum over all choices of the previous state s'. Similarly, let

$$\beta[i, s] = Pr[e_{i+1}, e_{i+2}, ...e_n | S_i = s, \hat{\theta}]$$

All values for β are calculated by initializing values for $\beta[n, s]$ and iterating as

$$\beta[n, s] = 1$$

$$\beta[i, s] = \sum_{s'} \beta[i+1, s'] \cdot O(s, e_i, e_{i+1}) \cdot T(s, e_{i+1}, s')$$

for all s, and this time summing over possible choices of the next state s'. Both sequences of calculations take $O(n \cdot |S|^2)$ runtime.

The probability of any sequence is given by

$$Pr[e_1, e_2, ...e_n | \hat{\theta}] = \sum_s \alpha[n, s]$$

Finally, the probability of having state s at time i is

$$\begin{aligned} Pr[S_i = s | e_1, e_2, ...e_n, \hat{\theta}] &= \frac{Pr[e_1, e_2, ...e_n, S_i = s | \hat{\theta}]}{Pr[e_1, e_2, ...e_n | \hat{\theta}]} \\ &= \frac{\alpha[i, s] \cdot \beta[i, s]}{\sum_s \alpha[n, s]} \end{aligned}$$

M Step.

That summarizes the estimation procedure. Given these estimates, the maximization step optimizes the parameters T_1 and O_1 by selecting the maximum likelihood

values. As with the standard HMM training algorithm, the transition between states is considered to be a Markov model. Therefore, the best estimate for the probability of transition from state s_0 to s_1 is estimate for all pairs s_0, s_1 as

$$T_1(s_0, s_1) = \frac{\text{count}(s_0, s_1)}{\text{count}(s_0)}$$

where $\text{count}(s_0, s_1)$ is the (expected) count of consecutive appearances of states s_0 and s_1 in the sequence, and $\text{count}(s_0)$ is the (expected) count of states s_0. Specifically,

$$\begin{aligned}
\text{count}(s_0, s_1) &= \sum_{i=1}^{n-1} Pr[S_i = s_1 | S_{i-1} = s_0] \\
&= \sum_{i=1}^{n-1} \frac{\alpha[i-1, s_0] \cdot O(s_0, e_{i-1}, e_i) \cdot T(s_0, e_i, s_1) \cdot \beta[t, s_1]}{\sum_s \alpha[n, s]} \\
\text{count}(s_0) &= \sum_{i=1}^{n} Pr[S_i = s_0] \\
&= \sum_{i=1}^{n} \frac{\alpha[t, s_0] \cdot \beta[t, s_0]}{\sum_s \alpha[n, s]}
\end{aligned}$$

Finally, to estimate the output function O_1, we want to find the MLE of O_1 for the overall probability of observing the data

$$\log Pr[e_1, e_2, ...] = \sum_{i=1}^{n} \sum_{s \in S} \Pr[s_i = s] \cdot \log \left[\frac{O_1(s, e_{i+1}) \cdot O_2(e_i, e_{i+1})}{\sum_{e \in E} O_1(s, e) \cdot O_2(e_i, e)} \right]$$

given O_2 as fixed. For each initial state s, the target function is differentiable analytically with respect to the values of $O_1(s, e)$ for all pairs of s and e. Hence it's feasible to use gradient descent to select the parameters for O_1 to optimize $Pr[e_1, e_2, ...|s_1, s_2, ...]$.

O_1 represents the updated estimation of what the hidden topics over the events are. Altogether, the EM algorithm alternates estimating the hidden states and optimizing parameters until converges, at which point it has learned the best LEMM.

6.4 Model Results

We test for the accuracy of the model again on the same 6.002 course. Our metric for evaluating the model is *average log-likelihood per event*. Given a trained model and test data, we can compute the probability of observing the data as

$$Pr[e_1, e_2, ...e_n] = \sum_s \alpha[n, s]$$

as detailed in the previous section. We compute the average log-likelihood per event by dividing the sum of the log-likelihood values over all event sequences by the total number of events across all sequences. This provides an intuitively understandable metric for how well the model fits the data on a event by event basis. When we refer to cross validation log-likelihood scores, we mean 5-fold cross validation. To compute this metric, we randomly partition the data into five sections. For each of the five sections, we train our model on the other four sections and test for the log-likelihood score on the fifth, and take average score over the five sections.

The following chart depicts the learning curve for training the LEMM, that is the cross validation log likelihood score of the model, when given certain numbers of training samples.

Furthermore, we calculate the weekly cross validation scores for the log-likelihood fit of the data for the LEMM, in comparison to the simpler Markov chain or HMM predictors. The scores in Figure 6-4 depict a week by week comparison between the LEMM and the Markov model.

We omit showing the comparison to HMMs because they perform much worse than either the LEMM or the Markov model. For EdX students, our model improves upon these scores by 6.8% over the Markov model, and by 90% over the HMM. From the cross validation scores, our model is a better fit for the data than either of the models alone. As an interpretation of the actual per-event log-likelihood values, the model assigns an (geometric) average of $e^{-2.13} \approx .12$ probability for each event. While this seems reasonable, Figure 6-5 puts the distribution of the next-event predictions in perspective.

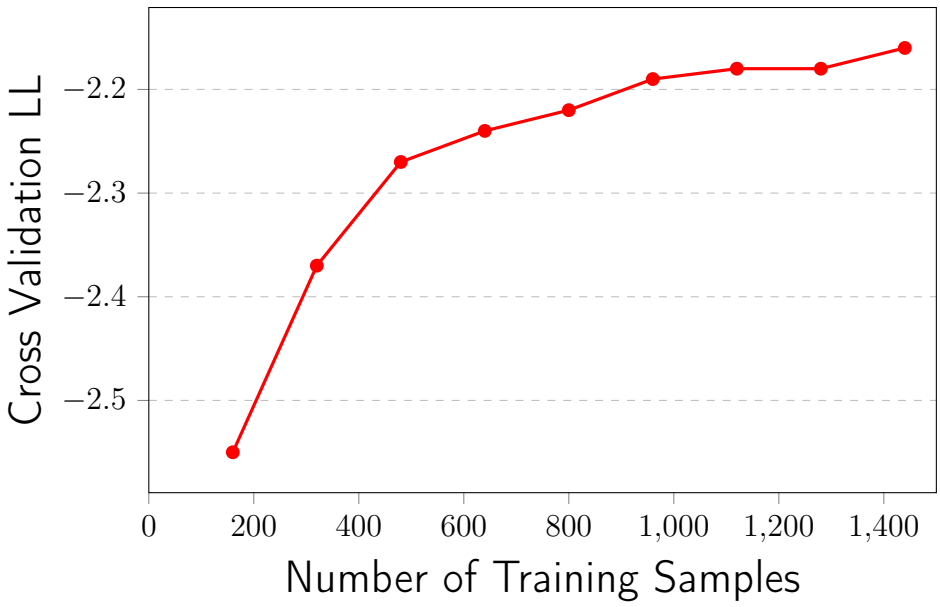


Figure 6-3: LEMM learning curve. The training makes rapid progress under about 500 training samples and approximately reaches a plateau near 1200 samples.

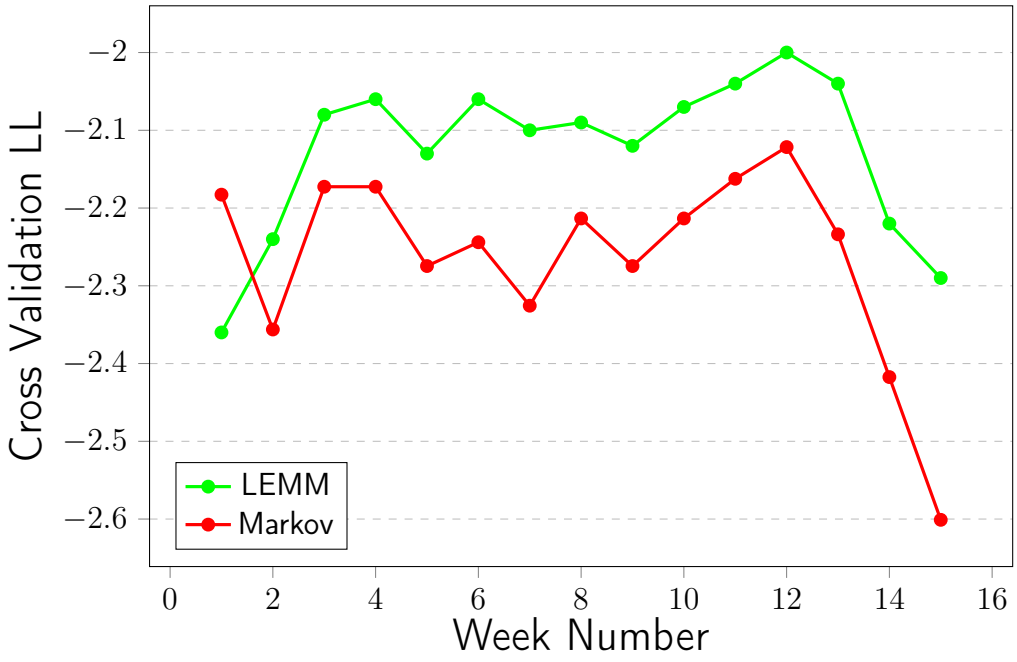


Figure 6-4: Comparison of LEMM and Markov models. The LEMM performs better in every week except for the first, although usually by only a small amount.

Over the course of all the event sequences, most events are easily predictable, but there is a definite tail for surprise events that skew the mean scores higher than the median.

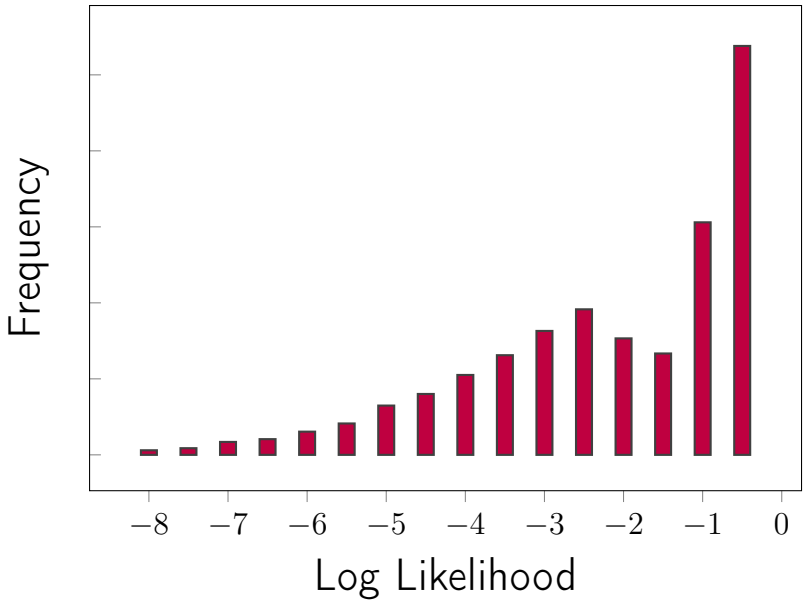


Figure 6-5: Distribution of LEMM log likelihoods by event. Most events are easy to predict, but there is a clear tail of hard-to-predict events.

6.5 Training Considerations

A number of issues in the training step arose when writing the software. Here we discuss some of the details of how we implemented the algorithm and the reasons behind the design choices.

Initializing hidden state transitions. Since the hypothesis is that students tend to stay on the same topic, the state transitions are initialized such that each state transition back to state with some probability ρ and to all other states with equal probability. We choose $\rho = .8$.

Selecting Markov smoothing constant γ. We tested various levels of γ using cross validation, and found that $\gamma = .0001$ is approximately optimal.

Learning initial hidden topics. We considered several topic learning approaches, such as training LDA or LSA models treating sequences as documents, or clus-

tering resources based upon their URL pathname. Although they perform reasonably similarly, LDA proved as least as good as the others and sufficiently fast.

Selecting number of hidden topics. Again we tested several values for the number of hidden topics, between 5 and 30, and found that 10 topics was generally effective each week.

Caching probabilities for faster runtime. A naive implementation of the algorithm required a very long time to run. We optimized the training process by precomputing all probabilities

$$T(s_0, e_1, s_1) \quad \forall s_0, e_1, s_1$$

$$O(s_0, e_1, e_1) \quad \forall s_0, e_0, e_1$$

during each iteration of the EM algorithm, before the hidden state estimation step. Computing each value of T requires iterating over all possible next states, taking $O(|S|)$ time. Computing each value of O requires iterating over all possible next events, taking $O(|L|)$ time.

Therefore, computing all the of the probabilities during the E step would require $O(N \cdot |S|^2 \cdot (|S| + |L|))$ time, where N is the total number of events in all of the student sequences. Precomputing the T and O values takes $O(|S|^2 \cdot |L|)$ and $O(|S| \cdot |L|^2)$ time, for a total runtime of $O(N \cdot |S|^2 + |S|^2 \cdot |L| + |S| \cdot |L|^2)$.

As $N \sim 10^6, |S| \sim 10^1, |L| \sim 10^3$, the runtime for the cached version is mostly dominated by the first term. This term is dramatically less than the runtime for the naive implementation.

6.6 Topic Features

As previously mentioned, the LEMM learns particular features for each sequence that can be useful for the course level model. In the training procedure for an LEMM,

the forward-backward algorithm infers for each sequence the probability distribution over the set of hidden states for each event $Pr[S_i = s]$, $i = 1, 2, ... \quad n, s \in S$. Over the entire sequence, the total counts of the hidden states or topics f_s are the features and are computed as

$$f_s = \sum_{i=1}^{n} Pr[S_i = s]$$

The loose interpretation of the total counts is how inclined each student is to visit resources or answer problems belonging to each hidden topic. In that light, our LEMM is also an algorithm for learning topics over events that utilizes the time-series structure, unlike traditional topic modeling.

In addition to being essential to training the parameters of the LEMM, the distribution of hidden topics are useful features for the course level model to capture topic trends across week by week sequences, and can also separately be used to improve dropout predictions. Learning these features for students makes the LEMM especially valuable, beyond the small improvement in log-likelihood over Markov models. The course level model learns the distribution of these topics week by week, and specifies in the data generation process the hidden state distribution for each synthetic student.

6.7 Mixture of Latent-Explicit Markov Models

6.7.1 Model Description and Training

The LEMM is successful at explaining how students transition from event to event. However, we hypothesize that students differ from one another enough that their behavior might be better described by multiple LEMMs. Students may differ in the topics that they pursue or in their direct link to link transition tendencies. A mixture of LEMMs is a collection of LEMMs that describe disjoint subsets of the student sequences. For example, the sequences $\{E_k\}$ can be partitioned into g clusters $C_1, C_2, ...C_g$ and then a LEMM can be trained on each cluster of sequences. This

would enable students to be grouped with others with similar behavior and characteristics, and provide a more accurate and fine-grained model.

The clustering procedure works similarly to other mixture models, for example a mixture of Gaussians. From an initial clustering, the algorithm first trains a LEMM on each cluster. Then it reestimates the (soft) membership of each sequence to clusters, based upon the log-likelihood scores assigned by each LEMM to the sequence. The process is carried out until the clusters converge. This training method is another instance of an EM algorithm. We describe the algorithm with the following pseudocode.

1. Randomly initialize the membership matrix M, where $M_{k,j} = Pr[E_k \text{ belongs to cluster } j]$.
2. For each $j = 1, 2, ...g$, train $LEMM_j$ for cluster j where sequences k are weighted proportional to $M_{k,j}$.
3. Calculate for each k, j the probabilities $Pr[E_k|LEMM_j]$. Set memberships
$$M_{k,j} = \frac{Pr[E_k|LEMM_j]}{\sum_{j'} Pr[E_k|LEMM_{j'}]}$$
4. Repeat until M converges.

In this formulation, there a few different ways we can choose to share some parameters across the different LEMM mixtures. The two sets of parameters that can be shared are the topics (states) and the Markov model.

To make topics global, and not specific to individual mixtures, we can reestimate the topic contents only once per iteration. The topics are fixed and not reestimated during the individual LEMM training, and so the resulting mixture of LEMMs will share a single set of topics or hidden states. This has the nice interpretation that there is a single set of topics across the events in the course, and the clusters differ

in their distribution and transitions over the topics.

Similarly, it is possible to estimate the Markov model only once using all of the student sequences, and not for each cluster of sequences. This supposes that student transitions are the same across clusters given the topics they look for.

Making either set of parameters (topics or Markov model) global reduces the number of parameters in the mixture model and increases stability at the possible expense of losing individual behavior. Note that the two components can be treated as global or individual to clusters independent of each other. For example, in the formulation where both topics and transition behavior are deemed global parameters, the clusters differ only in their distributions over the hidden states or topics.

6.7.2 Model Results

With many more parameters than the single LEMM, the LEMM mixture tended to perform better by log-likelihood fit on the training data. Typically, around 8 - 12 mixtures of students were appropriate per week in the course. However, the likelihood results on a held-out test set were more inconclusive. With cross validation scores that are quite similar to those from the single LEMM, the mixture model does not appear to fit the data much better. Table 6.1 compares the four possible LEMM mixtures by likelihood fit on both the training and test data.

Model	Global Params	LLH Per Event	Cross Validation Per Event
Single LEMM	NA	-2.07	-2.13
LEMM Mixture	Topics, Markov	-1.95	-2.08
LEMM Mixture	Topics	-1.93	-2.10
LEMM Mixture	Markov	-1.89	-2.13
LEMM Mixture	None	-1.88	-2.12

Table 6.1: LEMM mixture performance. The mixtures have a significantly better log-likelihood score than the single LEMM, with better scores for variants with more parameters. However all of the models perform similarly on held-out test data.

In practice, it may be more reasonable to use a single LEMM due to simplicity and a much faster training procedure. Training a single LEMM is already expensive both in runtime (approximately 3 hours per week's data) and in RAM (approximately 4 GB). Implementing a feasibly fast LEMM mixture model required leveraging one core per mixture, running in parallel. We used an Amazon EC2 machine with 16 cores and 64 GB of RAM. Even with these configurations, the LEMM Mixture model still takes on the order of 15 hours to train on each week's data, as each iteration of the EM algorithm requires retraining an LEMM.

Chapter 7

Course Level Model

Recall that some of the week level models learned particular features about each student in the data each week. Additionally, it is easy to compute some simple features about each student's sequence to help describe his activity for the week. The simple features we choose characterize the level of engagement for each student in three different ways: the number of events he had, his number of distinct sessions, and the number of submissions he made. The full set of feature vectors for each student in one week is outlined in Table 7.1.

Feature Category	Directly Observable	From LEMM	From Event Desc Model
Feature Names	Number of events	Hidden state 1 count	Duration parameter r
	Number of submissions	...	Assessment parameter g
	Number of sessions	Hidden state S count	

Table 7.1: Student features, computed for each week

The course level model learns how these feature vectors transition from week to week for most students. This model provides the backbone of the broader patterns and trends in the student data, such as which students tend to drop out or the

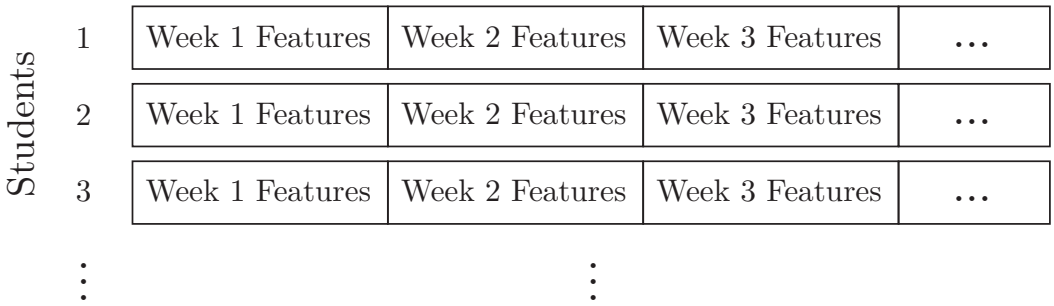


Figure 7-1: Concatenated student feature vectors

trajectories of how well different types of students do in the course.

The data the model is trained over is the set of full feature vectors that are the concatenation of each student's weekly feature vectors. This full feature vector has size $W \cdot F$ where W is the number of weeks in the course and F is the number of features compiled about the student's activity each week. The goal is to learn a probability distribution over these high-dimensional vectors, which would mean discovering the common types of students and their paths as their navigate through the course.

7.1 Model Definition

We fit a Gaussian Mixture Model on the set of the normalized full feature vectors. To normalize the input vectors, we subtract from each vector the global mean and then divide each component by the component-wise standard deviations.

We choose this model because it is a widely established model for unsupervised learning. It is able to learn a distribution of data points when they are hypothesized to come from different clusters (in this case, different types of students).

With M mixtures, the model parameters are

$$\{\mu_j, \Sigma_j, w_j\}, j = 1, 2, ...M$$

Each mixture is defined by the mixture mean μ_j, covariance matrix Σ_j, and weight

w_j. The likelihood of observing a data point x from this mixture model would then be

$$\sum_{j=1}^{M} w_j \cdot N(x; \mu_j, \Sigma_j)$$

The standard training procedure for a Gaussian Mixture model is an EM algorithm, that alternately estimates which cluster each data point belongs to given a guess for the model parameters, and optimizes the parameters given the estimate for cluster memberships. The number of mixtures M is chosen based on the Bayes Information Criterion (BIC) of the model fit.

Each Gaussian mixture can be interpreted as one group or type of student, with the means corresponding to the week by week average student. This model makes the assumption that the distributions of the features by week are approximately multivariate normal distributions in each cluster, which is generally reasonable.

One spot where the true distribution deviates from this assumption is that for the features "number of events", "number of sessions", and "number of submissions", there is a high density of students with value of 0 at these features. These 0 values are in fact important to the data because they indicate when students stop being involved in the course or stop trying to solve problems. A mixture of Gaussians when learned on this distribution will still generate a solid mass of students with 0 values in this features when negative feature values get rounded up to 0. However, a more sophisticated training algorithm to handle these edge values could be used and be more effective.

7.2 Application of Model

We train and test our model on same 6.002 course in the fall of 2012, which took place over approximately 16 weeks. Using 10 hidden topic features learned by the LEMM, there were 240 features in the vector space. Of the approximately 30,000 students who viewed at least 10 webpages, 20 mixtures of students were empirically

appropriate according to the BIC.

Figure 7-2 gives some idea of the general activity of the students in the course, by clusters. The largest 5 clusters of students are shown, when the Gaussian mixture model is trained with 10 clusters. While one cluster participated during the entire course, the other clusters in the data set only looked more casually at the course material for approximately 2, 3, 7, and 10 weeks respectively before dropping out.

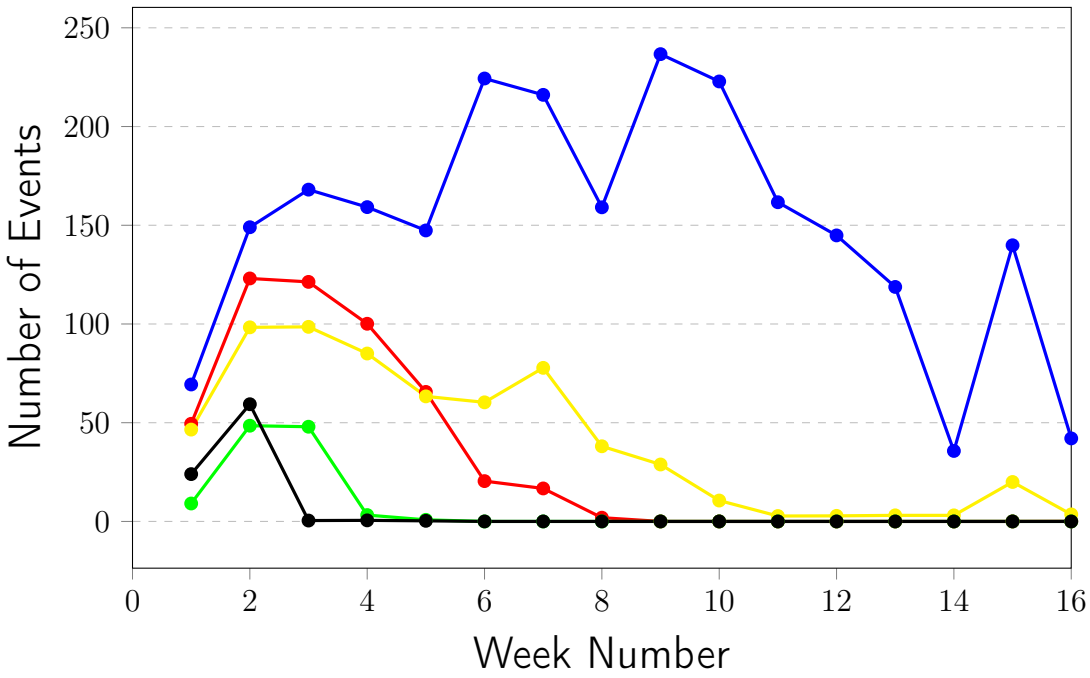


Figure 7-2: Clusters of student activity by week. Of these five largest clusters of students, one group participates through the whole course, and the rest drop out at differnt weeks.

Another look at the students can be demonstrated by looking at the relationships between features of the clusters, instead of only considering number of events. After model training, the following plots show the distributions of the cluster means. The plots are two-dimensional slices of the means, with each of the shown dimensions being an aggregate value over all of the weeks.

The graphs indicate positive relationships between the a cluster's number of events and number of submissions, and between the number of events and the assessment parameter. Neither of these trends are unexpected.

One easy-to-grasp measure of fit for the model is the average distance each feature

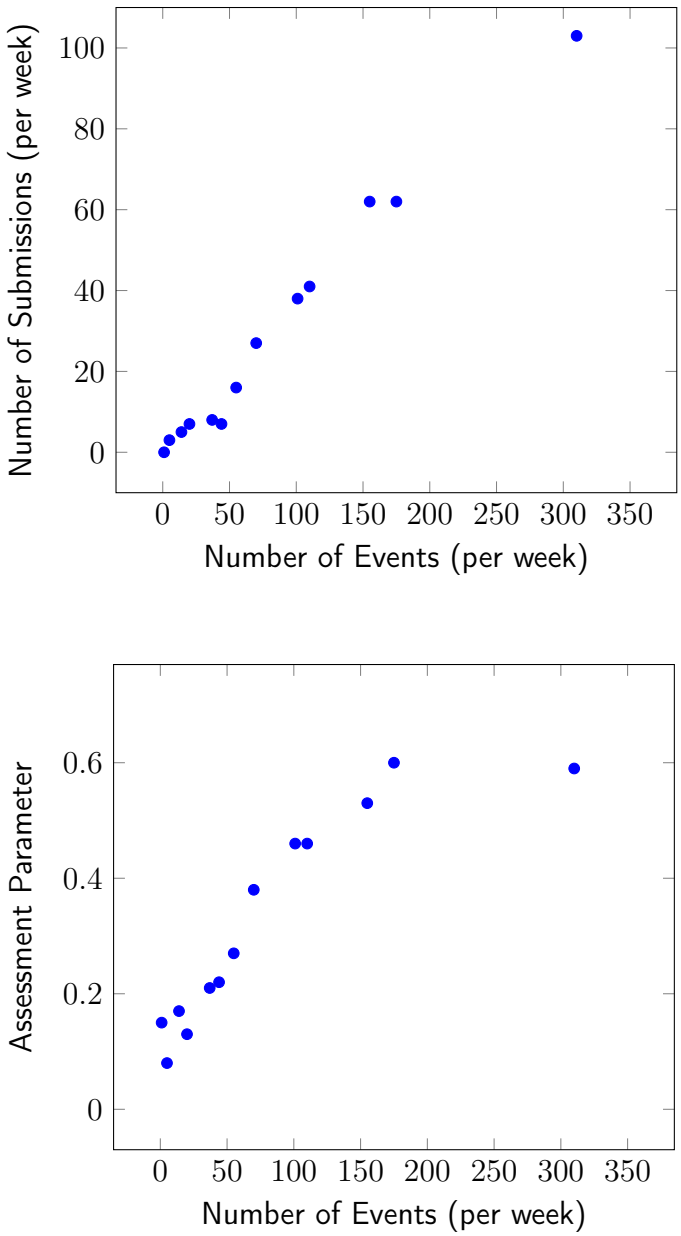


Figure 7-3: 2-D slices of cluster means. The more active clusters (as measured by more events) tend to make more submissions and get better grades.

vector to the nearest cluster mean. In units of standard deviation per feature, the average distance over all students to the nearest cluster mean was 9.09, or a distance of .038 per feature. Among the students who attempted at least 100 submissions, the average distances were 18.4 and .977 per feature. Students who have little overall activity tend to be close to a nearest cluster whereas there is naturally more variety among students who work through the entire course. Given the high dimensionality

of the feature vectors, being able to fit 20 clusters to 30,000 students at this level of accuracy is a good fit.

Chapter 8

Experimental Results

This chapter examines our full model developed for the Circuits and Electronics (6.002) course offering in the fall of 2012. This was the second semester of the first major course offered on edX. Approximately 30,000 students participated in this course, with 18,000 submitting at least one answer and nearly 3,000 receiving a certificate for passing the course.

The experimental procedure consisted of the following:

- Cleaning and reformatting data from the MOOCdb database
- Training week level models on each week's data as presented in Chapter 4, Chapter 5 and Chapter 6
- Training a course level model over per-student concatenated features as presented in Chapter 7
- Using the course-level model to generate synthetic student features
- Using week-level models to generate synthetic student sequences

8.1 How Good Are Our Models?

8.1.1 How Well Do Our Models Fit?

To validate the first goal, we tested the accuracy of all three-week-level models and the course-level model, usually in terms of log-likelihood. These results have already been discussed in the earlier chapters that detailed each model.

Table 8.1 summarizes the average log-likelihood scores for each model component on both training and test data, showing the overall fit of the multi-level model.

Component	LLH Per Event	Cross Validation Per Event
LEMM	-2.07	-2.13
Prob Topics	-1.29	-1.36
Durations	NA	NA
Assessments	-.36	-.37
Course Level	321	294

Table 8.1: Model component likelihood scores

8.1.2 Do Our Models Produce Synthetic Data Valuable For Data Mining?

Number of Events	Number of Submissions	Number of sessions	Grade Param	Duration Param

Table 8.2: Basic features. Used as a baseline set of features that should be preserved in the synthetic data.

To test whether we are able to generate synthetic sequences or synthetic features that have value in data mining, we performed the following experiments:

Build a predictive model over the real data: In this experiment, we extract features (given in Table 8.2) from the real student sequences. We then build a dropout prediction model using logistic regression and report cross-validated accuracy of this model on the real data. We call this experiment **PRD**.

Build a predictive model over synthetic features: In this experiment, we generate feature vectors for 10,000 synthetic students using the course level model. We then build a dropout predictive model via logistic regression and test the model on the real data. We call this experiment **PSF**.

Build a predictive model over synthetic sequences: In this experiment, we generate 10,000 synthetic student sequences from our trained models. We then extract features (given in Table 8.2) from these sequences and build a dropout prediction model via logistic regression. We then test model on the real student data. We call this experiment **PSS**.

In all of our experiments, we chose logistic regression for faster computation and stability of results. It turns out that using a Support Vector Machine yields similar results. For the weekly prediction problem, features are calculated for data up to that week and then weighted exponentially with respect to time. A logistic regression model is trained on these features, and the output variable is whether each student dropped out of the course by the next week. The definition of "*dropout*", how we assemble covariates for the logistic regression from the weekly feature data and the way we normalize data, is presented in Appendix 9.3.

The results from these three experiments are presented in Figure 8-1.

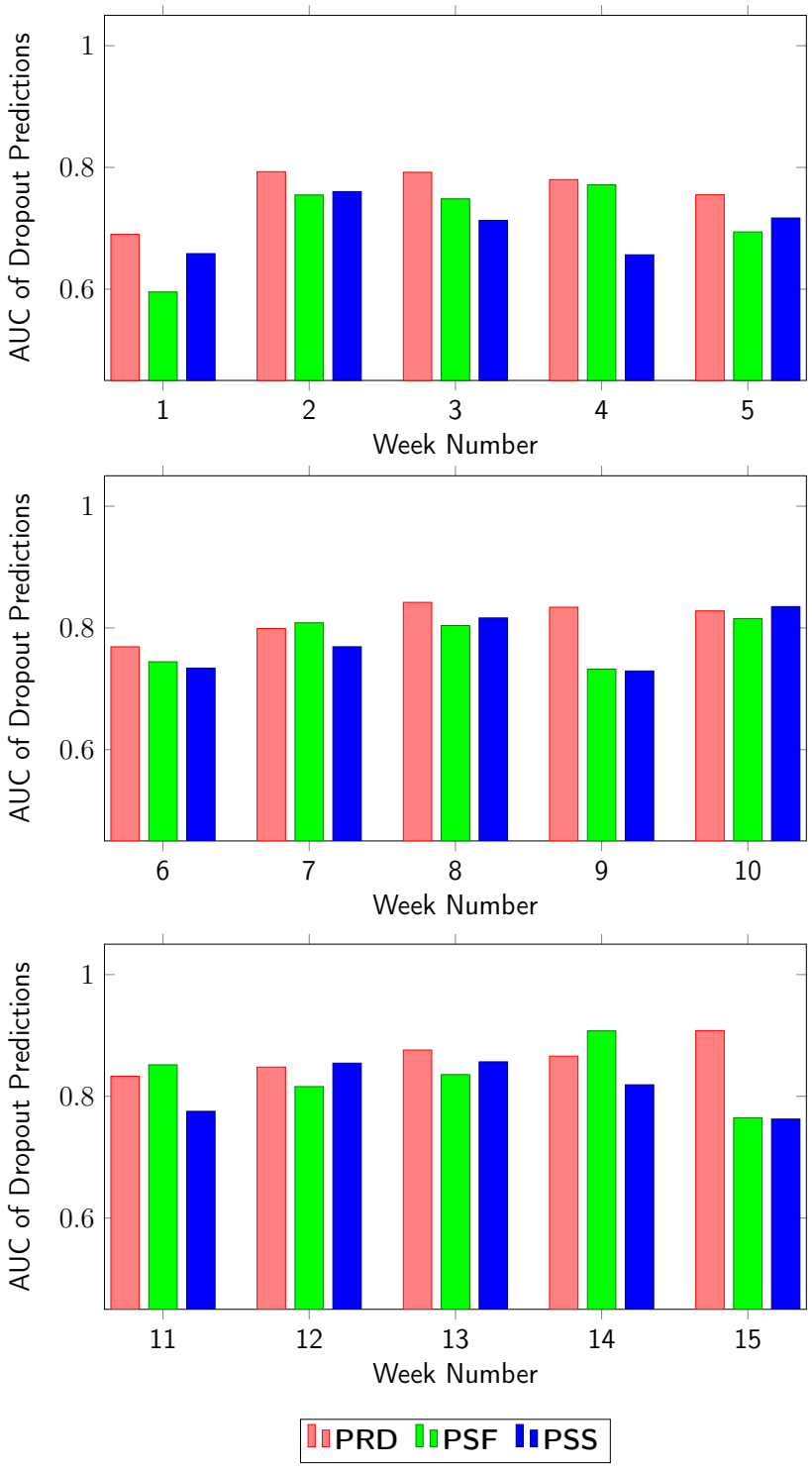


Figure 8-1: **Retention of dropout-predictive information in synthetic data**. In most weeks, there is small drop in score when **PRD** (models relying on real data) are compared to **PSF** (models developed on top of synthetic features) and a slightly larger drop when **PRD** is compared to **PSS** (models developed on top of the synthetic sequences).

Some dropoff in prediction accuracy is to be expected when the predictor is trained on synthetic data, because almost any synthetic data will lose some information or have some deviation from reality. The average AUC scores drop from .827 to .774 when the ML model is trained on the real data (**PRD**) to when trained on the synthetic features (**PSF**). The average AUC score when trained on the synthetic sequences is still .764 (**PSS**). As the performance difference is small (around 7.5% between real sequences and synthetic sequences), these synthetic sequences are seemingly still valuable for performing data mining. Thus, outside and independent researchers could develop meaningful machine learning/data mining models, if this synthetic data were to be released.

Predicting dropout for the second week of the course is the most difficult, as can be seen in all three experiments. The prediction problem difficulty appears to decrease later in the course. Possibly, students remaining in the later stages of the course are the more motivated and therefore less likely to drop out.

8.1.3 Do Our Models Preserve Data Characteristics?

In our second experiment, we compared the ranking of features, in terms of their predictive power, when the ranking was derived on the synthetic dataset versus when the ranking was derived on the real dataset.

Researchers perform studies to identify valuable features from data. To see how feasible this would be with our synthetic data, we proceeded with the following experiment:

Evaluate features on both real and synthetic data: We compute the 5 features listed in section 8.2 for each student in the real student data and the synthetic data.

Compose feature sets: For the 5 features we listed in Table 8.2, we compose a single feature set by picking one feature at a time (5 members in this set) and double features set by picking 2 features at a time (10 members in this set).

Build models on synthetic sequences: Using each member of the single set, we build a dropout prediction model and report its AUC performance. We do the same for the members of the double features set. We obtain 15 AUC values.

Build models on real sequences: Using each member of the single feature set, we build a dropout model on the real data and report its performance. We do the same for the double features set. We obtain 15 AUC values.

Figure 8-2 shows the relationship between the ranking of the two sets of 15 AUC values. We calculated the correlation between AUCs of models built on top of synthetic sequences and real sequences as well as the correlation between the rankings of the AUCs. Finally, we calculated these metrics under the single feature set and the double features set individually.

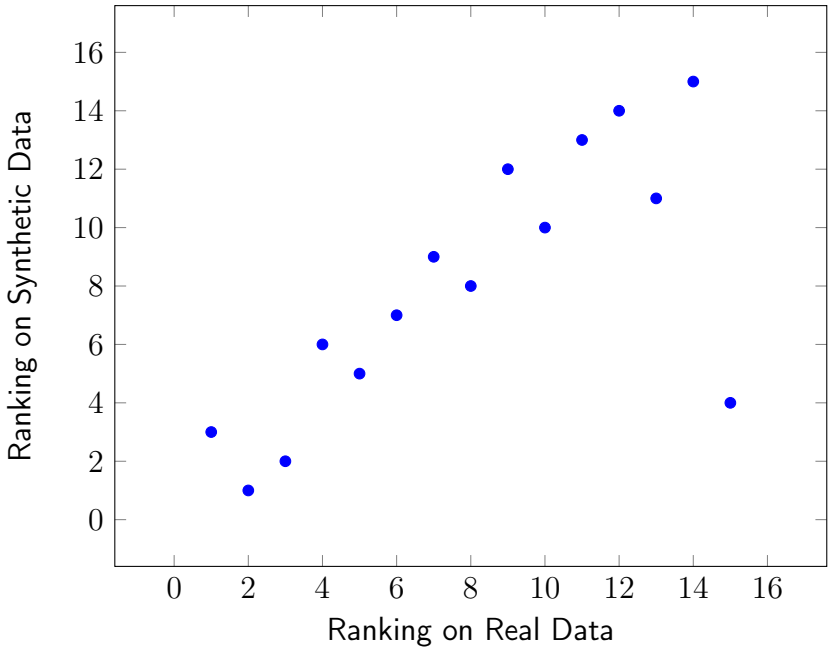


Figure 8-2: Comparison of relative feature performance. The trend appears roughly linear along the x = y line.

8.1.4 Can Our Models Protect Privacy?

We argue that our models successfully ensure privacy. First, none of the model parameters depend heavily on any particular sequence. Second, the data was originally

Feature Set	AUC Correlation	Rank Correlation
Single Feature	.91	.90
Double Features	.97	.49
Single + Double	.95	.72

Table 8.3: Relative Feature Performance Statistics

cleaned to remove URLs that only a few students visited and problem submissions that only a few students made. Additionally, timestamp information is not considered in our models. Finally, in each of our models, the parameters are learned across a number of students together. We further explain intricacies involved in each of the models below:

- In the problem topic model, problem topics are computed across all student submissions and cannot be traced back to any one student.
- In the event description models, distribution of durations and grades are computed only for *non-rare* URLs and problems. The individual students' duration and assessment parameters are not released but instead are given to the course level model.
- In the LEMM model, the Markov transition model contains aggregate transition counts across all students. The hidden state emissions and transitions are again based on cumulative (hidden) behaviors.
- In the course-level model, while the computed student features reveal individual activity, the learned parameters only capture a smooth distribution over the features. As each cluster has around 600 or more students, the Gaussian means and variances do not reveal anything about individual students.

8.2 Can Our Models Produce Predictive Features?

To answer this question, we evaluated our models to see if they provide valuable "features" for machine learning. We examined the feature value for the same dropout prediction problem defined in section 8.1.2 by performing the following experiments.

1. **Use behavioral features as predictors**: For all the students, we extract the basic features listed in Table 8.2. These can be directly computed from the real data. We then train a logistic regression model on the real data using these features. We report the cross-validation accuracy and call this experiment **BF**.

2. **Use behavioral features + PCA features as predictors**: We extract the basic features listed in Table 8.2 and PCA features learned on the event counts. To derive these PCA features, we compute the vector of counts of how often a student visited each resource or answered each problem. PCA solves a dimensionality reduction problem on these vectors to get a lower dimension description of event counts. We then train a logistic regression model on the real data using these features. We report the cross-validation accuracy and call this experiment **BF-PCA**.

3. **Use hidden state features from LEMM**: For each student, we infer the hidden state for each event in his sequence using the learned LEMM model. We then aggregate the total hidden state counts for each hidden state. We train a logistic regression model on the real data using these features. We report the cross-validation accuracy and call this experiment **LEMM-F**.

Leveraging LEMM-learned topic features improves dropout prediction accuracy, according to AUC, by 9.5% over the basic features and 7.0% over the PCA-learned features. This experiment confirms the claim that LEMM captures information beyond what simple features directly observable from event sequences can capture.

Furthermore, features learned by the PCA algorithm on basic event counts can be considered as latent topic features. These features, while improving upon prediction

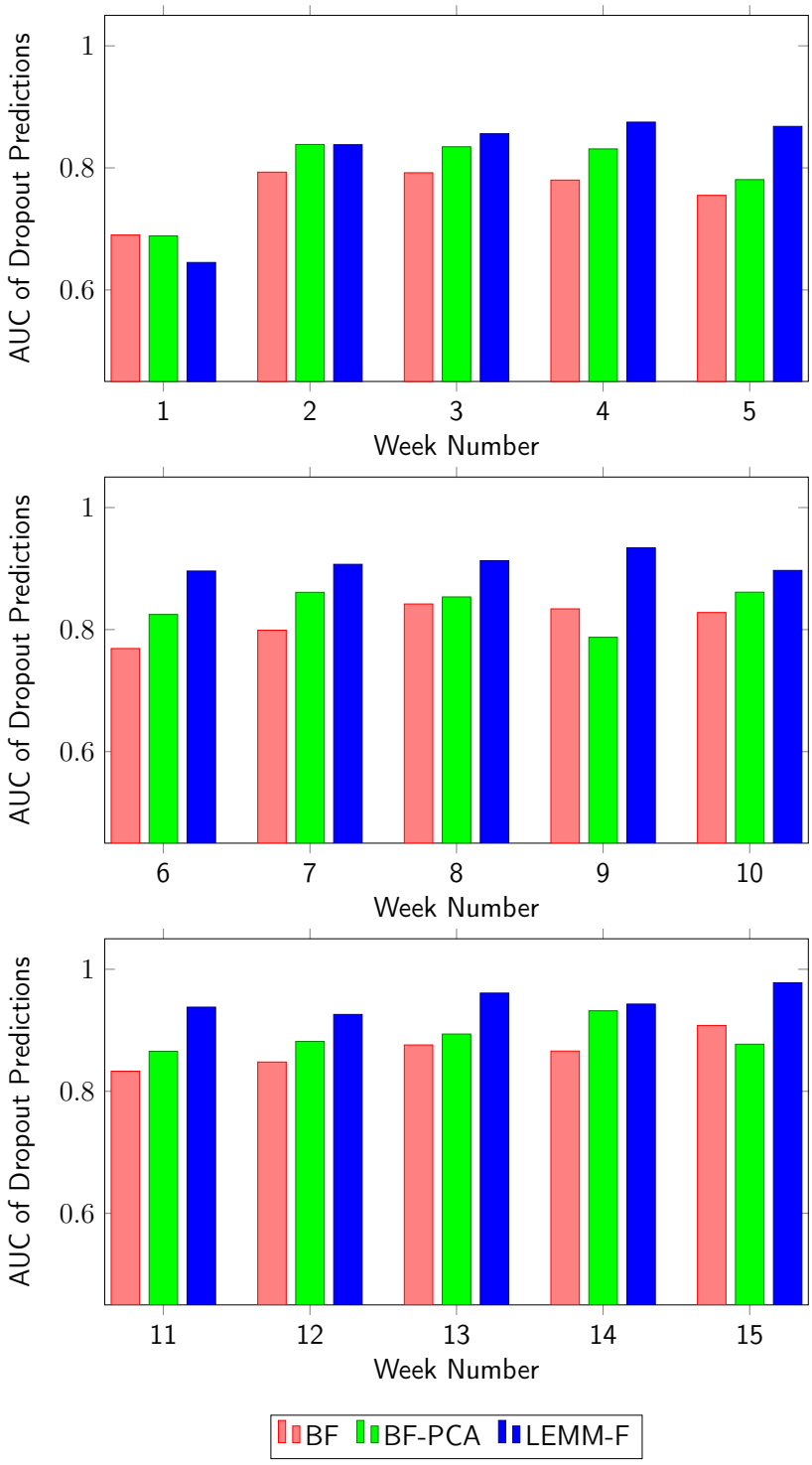


Figure 8-3: **LEMM-aided dropout predictions**. Here are the results of the three different experiments: **BF**, **BF-PCA** and **LEMM-F**. For almost all the weeks, the **LEMM-F** experiment does better than **BF** and **BF-PCA**. Thus we can say that features generated from LEMM provide good predictive value.

performance when compared to basic features, are still not as valuable as the topics learned by LEMM.

The general dropout prediction performance is similar to the results mentioned in [8], as a basis of comparison. Some of the differences in experimental setup include how "dropout" is defined and how to utilize features from all of the previous weeks. In this brief exploration into dropout prediction, it appears promising that LEMM topic features can make a strong impact on dropout prediction accuracy.

Chapter 9

Conclusion

9.1 Contributions

This thesis project involved many steps. We iterated over several versions of data curation, problem formulation, general approaches and models, and training algorithms. For each idea, we implemented and experimented with the course data. In the end, the main contributions of this thesis are both conceptual and practical, as follows:

1. Constructed the first complete model for analyzing student learning in online courses. Our generative model is able to generate synthetic student data that can be freely distributed.

2. Designed a novel generative model, LEMM, to explain student time-series data. This application is a better fit than the common HMM or Markov models. Also, we derived equations to train model parameters.

3. Applied LDA to model problem topics. This application has further potential to contribute to understanding how students choose which problems to consecutively attempt.

4. Designed a model for generating event descriptions, presenting a way to understand how long students spend on web pages and how well they solve problems.

5. Validated the full student model by comparing how machine learning models developed with synthetic data perform on the real data.

6. Improved upon simple features for predicting dropout by augmenting them with LEMM-learned features.

9.2 Key Results

The project successfully provides a method of generating synthetic student data for MOOCs. In the process, we obtained the following key outcomes:

1. There were approximately 20 main clusters or types of students in the fall 2012 6.002 course, when we fit the best Gaussian Mixture model on simple week-by-week features of the students.

2. LEMMs improved upon the cross validation log-likelihoods of the students' event ID sequences by 6.8% and 90% over Markov models and Hidden Markov models, respectively.

3. Applying LDA to consecutive problem submissions in students' event sequences yielded a log-likelihood fit that is 66% better than the entropy of simple problem distribution.

4. An events description model that accounts for variations among events and among students performed between 17% and 12% better than models that account for only one of the variances on duration and assessment predictions, respectively.

5. A logistic regression predictor for dropout performed only 7.5% worse when trained on the model-generated *synthetic* data than when trained on the real data, in terms of AUC score.

6. Using LEMM-learned features improved dropout predictions by 9.0% in terms of AUC score, compared to using only directly computable student features.

7. When training a logistic regression predictor on one or two directly computable features at a time (out of 5 possible) and ranking their usefulness in predicting dropout, there was a .72 correlation between the rankings on the synthetic data and on the real data.

9.3 Areas of Further Research

Such a broad and open-ended project allows us many further directions for continued work.

The structure of our method, to develop a modular course-level model and three week level models, allows for further development of individual components. In particular, there are likely easy improvements to be made to the course level model and to the event descriptions model. The course level model applies the common Gaussian Mixture model to the sets of student features, but as explained earlier, these features may not be normally distributed. In addition, other choices of interface features between the course level model and week level models could be potentially more effective. For generating descriptions, our model only assumes one duration and one assessment parameter per student; however, these descriptions are probably also influenced by which topics the student is more interested in or which materials he has already viewed.

Additionally, our project omitted information collected by edX that could be useful in developing a complete learning model. For example, a possible model could incorporate student information, such as gender, age, or country to aid the understanding of how these demographics may relate to online learning. We also did not use course information, such as resource types, student interaction types, problem types, or the structure of problems by parent/child relationships. For some courses, this information may not be available, but further investigation into these fields would lead to more conclusions in the future.

Appendix: Predict Dropout

Given a training set of student sequences and a test set of sequences,

1. **For each week $m = 1, 2, ...k$ perform on the training set:**

2. **Calculate the set of students I who have not yet dropped out by week m.** Define a student to have dropped out by week m if he doesn't have at least 10 events in any week beyond week i.

3. **For students $i \in I$, calculate the explanatory variables f_i.** These are the five features listed in Table 8.2. The feature vectors are computed for the student i's sequences in weeks $1, 2, ...m$ as $f_{i,1}, f_{i,2}, ...f_{i,m}$ and then weighted exponentially by time as

$$f_i = \sum_{j=1}^{m} \gamma^{m-j} \cdot f_{i,j}$$

for some constant $\gamma \in (0, 1)$. This means that more recent weeks are weighted more heavily.

4. **Normalize the explanatory variables.** For features 1, 2, and 3, the normalization involves a standard computation of the sample mean and variance. For features $s = 4, 5$, the mean and variance are estimated as

$$\overline{f[s]} = \frac{\sum_i w_i[s] \cdot f_i[s]}{\sum_i w_i[s]}$$

$$\sigma^2_{f_s} = \frac{\sum_i w_i[s] \cdot (f_i[s] - \overline{f_s})^2}{\sum_i w_i[s]}$$

This is a weighted estimate, where the weights are

$$w_i[4] = f_i[2]$$

$$w_i[5] = f_i[1] - f_i[2]$$

5. **For students in I, compute the output variables** y_i, dropout of student i for week $m + 1$.

6. **Train a logistic regression on the training set.** The training set is the set of normalized input-output pairs

$$\frac{f_i - \overline{f}}{\sigma_f}, y_i$$

7. **Compute the ROC curve and the AUC score for the predictor on the test set.** For each sequence in the test set, compute the same features, normalized by the already-computed means and variances. Evaluate the predictions of the logistic regression on these data points.

Bibliography

[1] MIT ALFA. Moocdb. Available at http://moocdb.csail.mit.edu/wiki/index.php ?title=MOOCdb, (December 2013).

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 2003.

[3] Igor V. Cadez, Scott Gaffney, and Padhraic Smyth. A general probabilistic framework for clustering individuals and objects. *ACM KDD*, page 140, 2000.

[4] Cody A. Coleman, Daniel T. Seaton, and Isaac Chuang. Probabilistic use cases: Discovering behavioral patterns for predicting certification. *ACM*, pages 141+, 2015.

[5] Fang Han. Modeling problem solving in massive open online courses. Master's project, MIT, 2014.

[6] RenÃľ F. Kizilcec, Chris Piech, and Emily Schneider. Deconstructing disengagement: Analyzing learner subpopulations in massive open online courses. *ACM*, 2013.

[7] Daniel Thomas Seaton, Justin Reich, Sergiy O Nesterko, and et al. 6.002x circuits and electronics mitx on edx course report - 2012 fall. Technical report, MIT Office of Digital Learning and the HarvardX Research Committee, 2014.

[8] Colin Taylor. Stopout prediction in massive open online courses. Master's project, MIT, 2014.

[9] Diyi Yang, Miaomiao Wen, Abhimanu Kumar, Eric P. Xing, and Carolyn Penstein RosÃľ. Towards an integration of text and graph clustering methods as a lens for studying social interaction in moocs. *The International Review of Research in Open and Distributed Learning*, 2014.