AskOski: A University Enrollment Guidance System Based On Deep Course Representation

Zachary A. Pardos
University of California, Berkeley
Berkeley, USA
pardos@berkeley.edu

Yuetian Luo
Renmin University of China
Beijing, China
luoyuetian123@ruc.edu.cn

Zihao Fan
Tsinghua University
Beijing, China
fan-zh13@mails.tsinghua.edu.cn

ABSTRACT
Inefficiencies in college students' choice of courses have been a significant contributing factor to normative graduation rates extending past the conventional 4-year degree. We design an interface and novel recommendation back-end towards addressing course selection inefficiencies with two approaches. Using course enrollment data from 120,000 students over the past seven years at the University of California, Berkeley, we created a course representation using techniques more commonly found in distributed vector representation methods applied to natural language. This representation, which possessed surprising regularities, can be queried by students looking for alternative offerings to courses of interest that may be at capacity. Our second approach was to personalize course suggestions based on a student's unique course history. We use the popular LSTM model, representing multiple courses per semester with a multi-hot encoding to predict students' next semester course enrollments and present this to them as information on what students like them have historically registered for next. Our validation accuracy of both the course similarity representation (27%) and prediction of next courses (19%) are on the same order of accuracy initially achieved in applying LSTM models to the task of text generation.

Author Keywords
Higher Education; Personalization; course recommendation; distributed representation; LSTM

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous;

INTRODUCTION
Normative time to earning a Bachelor's degree has wondered far from the expected four-year target. DeAngelo et al. [8] found that 6 years after matriculation, only 49.5% of students at public colleges had earned their degree compared with 78.2% at private universities. A more recent report on college completion [7] attributes three primary causes for the extended time to degree:

1. Not enough advisors: one for every 400 students
2. Students taking an excess of courses: on average one semester's worth of credits over what is required for graduation
3. Overenrolled courses: Twenty percent of community college students reported being unable to get into courses necessary for their degree and 33% reported being unable to get into electives.

The pervasiveness of overenrolled courses was enough to affect state politics with an ultimately unsuccessful bill introduced (SB-520) [17] which would have compelled the California post-secondary system to begin accepting online course offerings as alternative sources of credit for commonly overenrolled courses. While controversial, it was a proposed solution to an issue that has yet to be mitigated and which has become increasingly costly to the students and to institutions providing financial aid.

In this paper we design a system to address the issue of course enrollment inefficiency. While the long term goal is to reduce normative graduation time, if even by a modest amount, the immediate objective, evaluated in this paper, was to design a system that convincingly addressed aspects of the three causes of extended time to degree, outlined above. Our design involved two key design features. The first was giving students the ability to find alternative courses with deep pedagogical similarities to courses they wanted to take but were at capacity (cause #3). Course descriptions can be difficult for students to parse. They are often outdated, use obscure language, or were simply never an accurate reflection of course material. For this reason, we designed a backend algorithm using a novel application of natural language processing to surface deeper, pedagogical similarities from enrollment sequence data alone. The design second feature was to provide predictions on which courses were most likely to be taken in the next semester given the student's personal course history. We acknowledge that this type of suggestion is not strictly the same as what may be the ideal enrollment given the student's goals but we argue that

it is nevertheless a useful source of information akin to asking a more senior student what they had done. It is also an abstract representation of normative enrollment behavior which, in Universities, can often be tribal knowledge buried deep beneath the administrative substrate. This recommendation feature of the system is intended to surface this knowledge for all students, on-demand. We believe that this information can be useful not only for the student (cause #2) but may aid advisors as well in giving meaningful personalized advice to their many advisees (cause #1).

Related Work on Recommendation and Representation

Several paradigms of approaches have been taken towards improving the course enrollment experience for students. Parameswaran et al. [15] built a recommender based on constraint satisfaction; taking into account institutional breadth and degree requirements as well as scheduling constraints of the student and courses being recommended. Li, Tinapple & Sundaram [12] propose a course to proficiency tagging regime combined with a visual mapping between proficiency components that can guide students through a pathway of post-requisites. While they place the responsibility to decompose proficiencies of a course on the teacher, our representation relies on regularities in the data to surface these relationships. Farzan & Brusilovsky [9] asked students to give their career goals and then rate courses for their workload and relevance to those goals, allowing other students to then select courses based on those characteristics.

A related body of work has focused on predicting students' outcome on courses; using C4.5 decisions trees to help students select courses they're likely to succeed in within the School of Systems Engineering at the University of Lima, Puru (Vialardi, C., Bravo, J., Shafti, L., Ortigosa, A.) [18] and showing students their likelihood of success while taking a course using red, yellow, and green indicators to try to improve engagement at the University of Purdue (Arnold, K. E., & Pistilli, M. D.) [1].

Our current work distinguishes itself from prior art in its distinctly data driven approach to recommendation and similarity representation. While the long-term aims of our system could benefit from complimentary sources of information (e.g. constraints, experts, and surveys), in this paper we focus on a design based on a novel source, large scale course enrollment histories, towards surfacing student actionable information. Our backend course analysis is a novel contribution, serving as a first in a few categories: (1) analysis at this (100k+ students) order of magnitude data scale (2) generalization of a single model over all departments and courses at a college or University, and (3) evaluation of the underlying recommendation algorithm using rigorous validation regimes similar to those employed in word generation models.

Selection of backend algorithm

Our intention was to choose a modeling framework that could satisfy two objectives: (1) allowing students to query the model for similar courses to their desired course and (2) provide suggestions for their next course enrollments based on their course history. Recurrent Neural Networks (RNN) have been used in a recent spate of Natural Language Processing (NLP) applications to predict the next word in a sentence. These methods also include an embedding layer that encodes each word into a vector, which can be queried for similarity. These word vectors exhibit semantic regularities of surprising depth (Mikolov, T., Yih, W. T., & Zweig, G.) [14], such as the now famous vector arthamtic equation of, "[KING] – [MAN] + [WOMAN] = [QUEEN]." In that example, the vector representing [QUEEN] is not the exact match but was the closest in terms of cosine distance, and therefore a hit. In NLP, words are embedded into semantic space and then an RNN is used to learn the temporal traversals across those words (i.e. the structure of English language). In our application, the words synonymous with courses and the RNN is learning the traversal across courses to provide next step recommendations (i.e. learning the structure of degree programs). We quickly discovered that online training of the embedding using an RNN defeated the interesting similarity information of the embedding. So, we instead use word2vec (which we call course2vec) to learn the embedding for similarity queries and then use a freshly trained RNN to power the next course recommendations.

The next section introduces the dataset, followed by the design for course alternatives interface, model, and results. Next is our design for course recommendation section, which similarly includes its own interface, model, and results sub sections. We end with conclusions and future work.

DATASET

We used a novel dataset from the University of California at Berkley (UCB) which contained student course enrollments from 2008 Fall through 2015 Fall. The dataset consisted of per-semester course enrollment information for 110,334 undergraduate and 38,147 graduate students with a total of 3.6M course enrollment records. A course enrollment meant that the student was still enrolled in the course at the end of the semester. The median course load during students' active semesters was 4 courses. There were 9,739 unique courses, including 9,038 unique primary lecture courses, from 124 different departments hosted in 17 different colleges divisions. Course meta information contained course subject, department name, total enrollment, and max capacity. In all analyses in this paper, we only consider primary courses (lecture) and courses with fewer than 10 enrollments total are filtered out. The unique number of primary courses offered in any given semester can be seen in Figure 1. For both graduate and undergraduate courses, the unique offerings (not including summers) averages around 2,000 with graduate offerings on the down trend and undergraduate offerings on the uptrend. A log scale histogram of total enrollments for each class can be seen in Figure 2 with total number of active students by semester shown in Figure 3.

The raw data were provided in csv format by the UCB Educational Data Warehouse team. Each row of the course

enrollment data contained date stamp information, a student ID, undergrad/grad status, and declared major at each semester. Course information included course name, department, subject, enrollment count, and capacity. For data access simplicity, we preprocessed the data into a dictionary and serialized it into a JSON file as our first step of the data processing pipeline (see in Figure 4). After we populated the dictionary with student ID as key and course enrollment information as value, we randomized the order of courses within each semester for each student, since there should be no discernable order within a semester and we did not want any biases in the given order to affect training of the model.

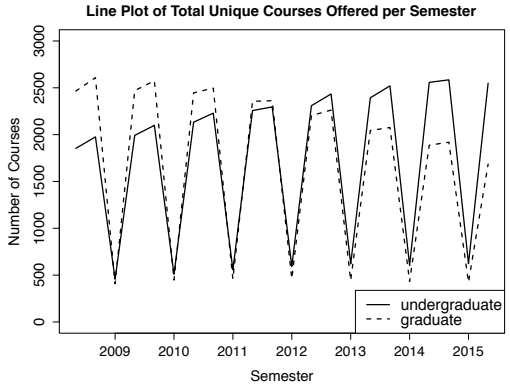

Figure 1. Total unique primary courses offered per semester for undergraduate and graduate students

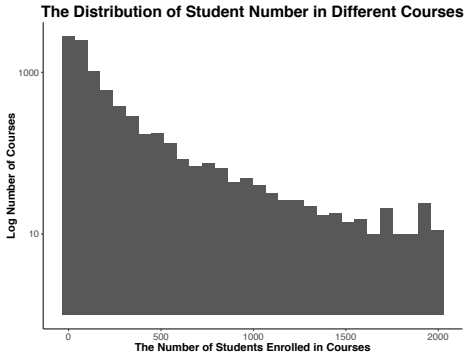

Figure 2. Total number of students enrolled per course across all semesters

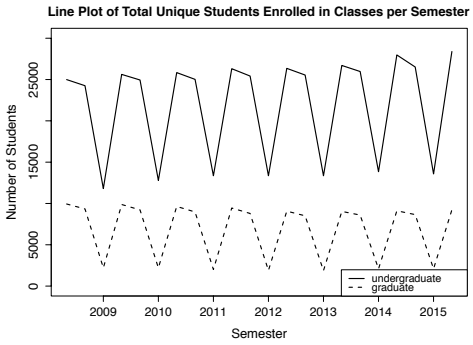

Figure 3. Total number of active grad and undergrad students by year and semester

Semester Year Concat	STU ID(anon)	Undergraduate/graduate	Course Academid Dept	Course Subject Short Name	Course Index
2008 Fall	9984	Graduate	Public Health	Public Health	382
2008 Fall	9984	Graduate	Social Welfare	Social Welfare	200
2009 Spring	9984	Graduate	Public Health	Public Health	219
2014 Spring	282243	Undergraduate	Statistics	Statistics	222
2014 Summer	282243	Undergraduate	Statistics	Statistics	351
2014 Fall	282243	Undergraduate	Sociology	Sociology	103
2014 Fall	282243	Undergraduate	Statistics	Statistics	133

9984: { 20083: [382, 200], 20091: [219] }

282243: { 20141: [222], 20142: [351], 20143: [103, 133] }

Figure 4. Example of raw data and its conversion to JSON

DESIGN FOR ALTERNATIVE COURSE DISCOVERY

As the #3 cause for student extended normative graduation states, 33% of students expressed an inability to register for electives due to them being at capacity. The intention of this design feature was to allow students to find courses which were intellectually, conceptually, and pedagogically similar to the course they were trying to take. If a student were to take a course they were not very interested in, this would be considered a course enrollment inefficiency, which we have set out mitigate.

Interface

In this section we introduce AskOski, the web-based course guidance system whose UCB instance is named after the school's mascot, Oski! Before we discuss the design feature, we will briefly introduce our platform.

Technical Design

One important difference between AskOski and other Berkeley course assistance interfaces such as guide.berkeley.edu[1] and berkeleytime.com[2] is that AskOski provides recommendation and that these recommendations are based on the student's personal course history. It is also the only service providing a non-descriptive, novel source of course relational information. In order to ensure a quick response from server when query comes, AskOski is designed to run on two separate backend servers. The first server is regarded as the main backend server which is used to deal with common logic. The second server is specially designed for running the analysis engine for the two design

[1] http://guide.berkeley.edu/

[2] https://www.berkeleytime.com/

features. We decided to use python flask as the backend for the analysis web service. During the phrase of selecting the first backend server, there were two obvious choices of technologies: Ruby on Rails or Node.js with a backend MVC framework.

We decided to build off an existing codebase (Pardos & Kao) [16] which used Node.js with an MVC framework versus using Rails, citing the following reasons:

- Rails is built on Ruby. Node.js is built on JavaScript, which (often times) has comparable or faster benchmark times than Ruby due to the V8 Engine3.Try to avoid long or complex sentence structures.
- Rails is an opinionated framework, forcing a developer to adhere to its culture and stigmas. Node.js is the opposite, more allowing for a step- by-step build of various components.
- Rails has a steep learning curve with numerous Ruby intricacies to consider. Learning JavaScript is rather quick with experience in any major language like Java or C.

To make up for what Rails provides out of the box, we used a popular Node.js MVC framework: Sails.js. This allowed us to quickly develop a web application with basic functionality in a Rails manner while still maintaining the ability to customize the application as we desired.

Alternative Course Discovery Interface

When selecting the "Alternatives" screen, the student is prompted to enter a department, followed by subject, and then course. Each one of these entries are searchable from within the pull-down box and the subsequent potential values are loaded dynamically to avoid the full 9k list of courses from loading at start. Once the student has submitted their course, the corresponding top 10 most similar courses in terms of cosine similarity are displayed along with the course ID, name, subject, department, and similarity (see Figure 5).

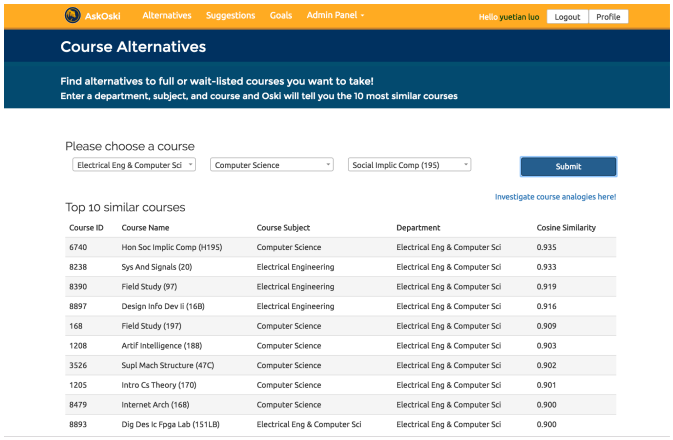

Figure 5. Course alternatives feature. Showing the top 10 similar courses to "CS: Social Implications of Computing"

Course Analogies Interface

Not only does the course representation help us to find alternative course, it also displays pretty nice regularities among course vectors. For example vector("Econ 101A")-vector("Econ 100A")+vector("Econ 100B")≈vector("Econ 101B"), "Econ 101A" and "Econ101B" are the math version of "Econ 100A" and "Econ 100B", so it means that course vectors can represent the rigorousness of course's mathematical level; also vector("ARTH 34") – vector("CHI 1A") + vector("JAP 1A") ≈ vector("ARTH 35"), "ARTH 34" is the art history of China, "CHI 1A" is a foreign language class for introductory Chinese, "JAP 1A" is the foreign language class for introductory Japanese, and "ARTH 35" is the art history of Japan, from this example, we can find that the course vectors can even represent culture. These nice regularities are quite common in course vectors. Because of its nice property, we are going to use these regularities in our user analogies to evaluate the quality of course vectors.

Validation

In order to test the validity of the similarity representation, we could (a) conduct a user study, asking students to pre-register their hypotheses about similar courses and see how those hypotheses are born out from the model or (b) utilize existing hypotheses about courses to validate similarity. We used both, which are described below.

Cross-list Accuracy

Ideally, two classes that teach the same topic should have a cosine similarity close to 1. At UCB, there are around 2k courses which have different course numbers, often in different departments, but are actually referring to the same course. These are cross-listed courses and represent the exact same course and thus an ideal set of similarity hypotheses to test our model against. We also include the reverse of each pair since, while the cosine distance will be reciprocal, the rank of the distance may not be. It is worth noting that semantic accuracy, as computed by (Mikolov, T., Chen, K., Corrado, G., & Dean, J.) [13] relied on evaluating analogies (A is to B as C is to ?), whereas in this cross-listed accuracy validation, we are validating based on rank of cosine distance between a single cross-listed pair. In our user study, we where we solicit course analogies, which we evaluate with the analogy method.

The procedure to calculate semantic accuracy based on cross listed courses is shown below.

1. Denote N cross-list course pairs as $p_1,p_2\ldots..p_N$. We denote the first course in course pair pi as $p_i[1]$, the second course in course pair as $p_i[2]$.
2. For the first course in each course pair, calculate the most nearest course, top5 nearest courses, and top10 nearest courses. Denote these three sets as S_1, S_5, S_{10}.
3. Then define the cross-list accuracy for each set j as:

$$\mathrm{Pr}_j = \frac{\sum_{i=1}^{N}(\mathbf{1}_{S_j}(p_i[2]))}{N}$$

where $\mathbf{1}(\cdot)$ is the indicator function.

So, we assume that a course representation is better if it has higher average cross-list accuracy.

Cross-list accuracy is an effective way to validate the quality of our course vectors for surfacing similarity but we may also like to know how far away the cross-list pairs were from each other, on average. We use the median rank of the cosine distances among the cross-listed pairs to compute this.

User Analogies

In addition to sourcing the cross-listed pairs as a validation set, we also solicited a fourth year undergraduate Economics student and a University staff subject matter expert to put together a list of course analogies that they expected to observe. The list totaled 58 course analogies after they completed their task. We regarded the first and third courses as positive courses and the second course as the negative course. By simple arithmetic calculation vector("course1")-vector("course2") + vector("course3"), we find the rank of cosine distance of the fourth vector to the computed vector. We defined similar metrics for evaluating the accuracy of this user analogies validation.

Model

In this section we describe the mathematical model behind the similarity representation. While there are several sources of information that could be brought to bear to the same end, we use only the sequence of course enrollments for every student in order to surface more general similarities between all courses, globally. One common technique to achieve such purpose in a classical language model is to represent words with continuous vectors. And among the most popular word representation methods is distributed word embeddings (Mikolov, T., Chen, K., Corrado, G., & Dean, J.) [13]. Although the sequence of courses do not have the exact same features of sentences, such as grammar and morphology, it does have a similar "semantic theory" or structure to sentences in language. For example, if two courses are often taken as alternatives to one another and students who take them otherwise share a similar context of courses, those two courses will be mapped to a similar vector space. The mapping of a course to a vector, much like with language, is a combination of the effect of a degree program structure and student deliberate self selection.

Course2Vec (C2V)

Our Course2Vec model uses skip-gram or alternatively continuous-bag-of-word models (CBOW) models to generate course representations. In these model the chronological sequence of all courses one student has taken is viewed as one sentence in our Course2Vec model. The architecture of skip-gram and CBOW are shown in Figure 6 and Figure 7.

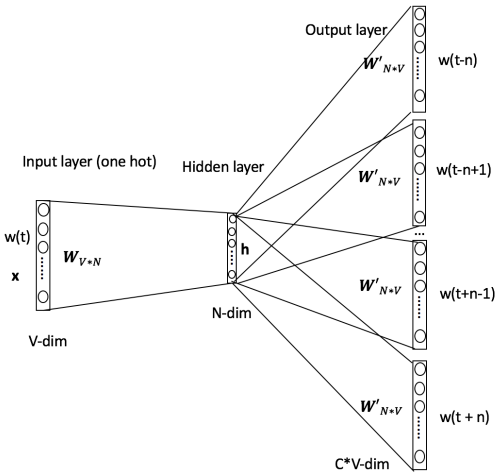

Figure 6. The Skip-gram model architecture

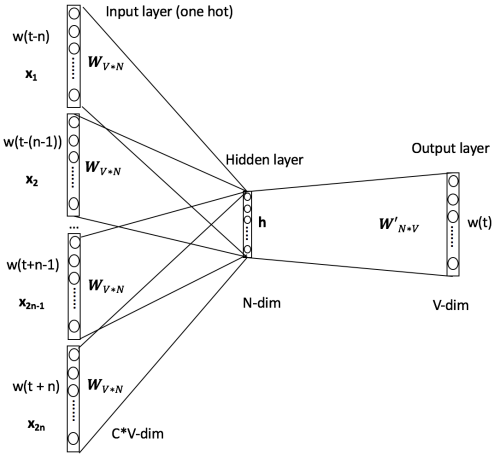

Figure 7. The CBOW model architecture

The internal mechanism of skip-gram and CBOW are given below:

For skip gram: The vector of hidden layer is

$$\mathbf{h} = \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{x}$$

By using softmax, a log-linear classification model, we get the posterior distribution of words

$$\mathrm{p}\left(\boldsymbol{w}_{\boldsymbol{c,j}} = \boldsymbol{w}_{\boldsymbol{c}} \middle| \mathbf{w}(\mathbf{t})\right) = \boldsymbol{y}_{\boldsymbol{c,j}} = \frac{\exp\left(\boldsymbol{u}_{\boldsymbol{c,j}}\right)}{\sum_{\boldsymbol{j'=1}}^{\boldsymbol{V}} \exp\left(\boldsymbol{u}_{\boldsymbol{j'}}\right)}$$

$w_{c,j}$ is the j-th word on the c-th panel of output layer; w_c is the actual c-th word in the output context words; $y_{c,j}$ is the output of the j-th unit on the c-th panel of the output layer; $u_{c,j}$ is the net input of the j-th unit on the c-th panel of the output layer

Because the output layer panels share the same weights, thus

$$u_{c,j} = \boldsymbol{v}'^{\,T}_{w_j}\mathbf{h}$$

where $\boldsymbol{v}'_{w_j}$ is the j-th column of the matrix $\boldsymbol{W}'$.

The training object of skip-gram is to maximize the conditional probability of observing the actual nearby words given observed word. So the loss function is

$$E_{skip-gram} = -\log\left(w(t-n), w(t-n+1), \dots, w(t+n-1), w(t+n) | w(t)\right)$$

where w(t) is the input word; w(t-n),w(t-n+1),…,w(t+n-1),w(t+n) are output words.

For CBOW: The vector of hidden layer is averaged by all $\boldsymbol{W^T x_i}$

$$\boldsymbol{h} = \frac{1}{2n}\boldsymbol{W}^T(\boldsymbol{x_1} + \boldsymbol{x_2} + \cdots + \boldsymbol{x_{2n}})$$

The posterior distribution of words is

$$\mathrm{p}\left(w_j \middle| \mathrm{w(t)}\right) = y_j = \frac{\exp(u_j)}{\sum_{j'}^{V} \exp(u_{j'})}$$

Where y_j is the output of the j-the unit in the output layer; w_j is j-th unit in the output layer; u_j is the score for j-th word in the vocabulary; $u_j = \boldsymbol{v'_{w_j}}^T \boldsymbol{h}$, $\boldsymbol{v'_{w_j}}$ is the j-th column of the matrix $\boldsymbol{W'}$

Also we can get the loss function of CBOW:

$$E_{CBOW} = -logp(w(t)|w(t-n), w(t-n+1), \dots, w(t+n-1), w(t+n))$$

where w(t) is the output word; w(t-n),w(t-n+1),w(t+n-1),w(n) are input words.

Supervised Course2Vec (supervised C2V)

In our base Course2VecC model, all word relationships were learned without any input information about word morphology, which is a kind of weak unsupervised learning. Although, it is mentioned in Mikolov et al. [13] that further progress can be achieved by incorporating information about structure of words, it is not easy to achieve that in a language model. We hypothesized that if we collapsed the cross-listed pairs of courses into a single course, this would bias the training towards modeling this particular relationship. While this is not strictly supervised, we give this model which combines cross-listings the nickname of "supervised C2V." In our results section, this supervised variant is created by collapsing only a random half of the cross-listed pairs. The other half are used as the validation pairs. The same half are used to validate both the supervised and unsupervised versions of C2V.

Hyper Parameter Search

For each similarity model, we sweep through various hyper parameters of the models including the method (skip-gram or CBOW), window size [1,2,3,4,6,8], vector size [2,8,16,32,64,96,128,160,192], iterations [5,10,15,20,25], fixing the minimum required data points for a course at 50.

Results

Comparison of C2V and supervised C2V on Cross-listings

In these C2V results, shown in Table 1, we find that with 13.2% of the cross-listed pairs, the pair was the closest course and in 37.2% the pair was among the top 10 closest courses. Supervised C2V does only marginally better.

Cross-listing Accuracy	Top 1	Top 5	Top 10	Median Rank
C2V	13.2%	28.0%	37.2%	21
Supervised C2V	**13.7%**	**30.3%**	**41.7%**	**17**
RNN embedding	1.2%	5.1%	7.2%	1960
RNN+pre-trained vectors	1.5%	5.7%	10.0%	710

Table 1. Comparison of cross-listing validation accuracy using four sources of course vector representation

Comparison of C2V and supervised C2V on User Analogies

In Table 3, we find that C2V performs better on user analogies (27.5%) than on cross-listings (13.2%). We also find that, unlike cross-listings, supervised C2V provides a very substantial improvement in accuracy over C2V on the user analogies validation; improving accuracy by nearly 50%.

User Analogies Accuracy	Top 1	Top 5	Top 10
C2V	27.5%	39.2%	47.1%
Supervised C2V	**40.5%**	**51.4%**	**56.8%**

Table 3. Comparison of user analogies accuracy using C2V and supervised C2V

Comparison of Cross-listed accuracy among Different Students Groups (Based on supervised C2V)

If the focus is on undergraduate education, are we better off training the model only on undergraduate students or do the graduate student enrollments provide useful information that improves the cross-listing accuracy? To investigate this, we keep only the cross-listed course pairs where each course has at least 10 undergrads enrolled and 10 grads enrolled total, throughout all of its offerings. This step leaves us with only 105 cross-listed pairs left but was necessary so we would be comparing the same pairs with each dataset. What we found, shown in Table 3, is that training on undergraduates only gives the best accuracy on the overlapping 105 cross-listed pairs. While interesting, these 105 pairs are special in that they represent courses that both grads and undergrads take and are therefore not necessarily representative.

	Measurements			
Accuracy	Top 1	Top 5	Top10	Median Rank

Undergraduate	**25%**	**40.4%**	**44.2%**	**19**
Graduate	15.2%	19.6%	21.7%	40.5
Both	16%	21%	25.9%	64

Table 3. Comparison of Different Student Groups

DESIGN FOR NEXT COURSE RECOMMENDATION

Cause #1 of extended graduation time was oversubscribed advisers and #2 was students taking more credits than needed to graduate. We designed a course recommendation feature to help advisors give personalize recommendations to students and for students to directly receive personalized recommendations that may aid them in their course selection.

Interface

The second function of AskOski helps students to find courses they are likely to take next based on their course history. The system will display the top 10 next course recommendations upon clicking on the "Suggestions" tab (see Figure 9). If the student wants to view the top 10 within a specific department, she can specify that using the department and subject selection drop-down (see Figure 10).

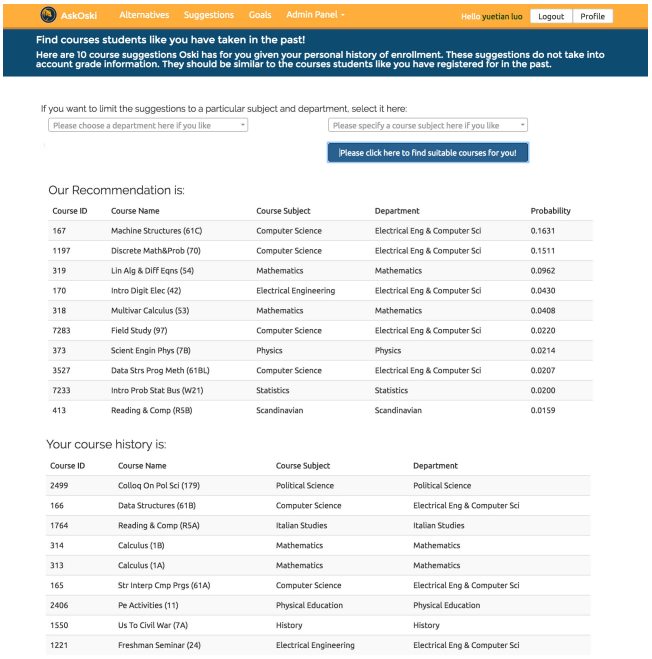

Figure 8. Course Suggestions: the top 10 suggested next courses predicted for the student given their enrollment history

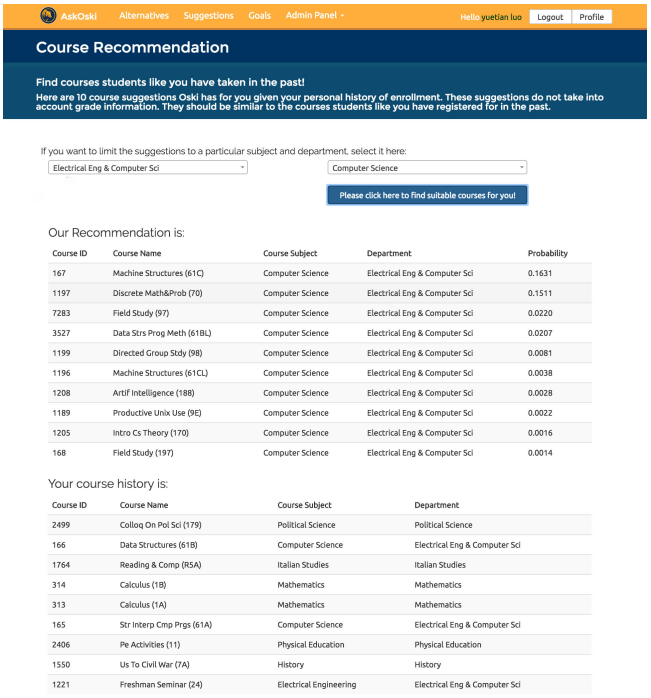

Figure 9. Course Suggestions: the top 10 suggested next courses within the subject area of Computer Science

Validation of Course Prediction Models

We validate the next course prediction models by calculating the accuracy with which they predict the next courses students enroll in. At the beginning of this analysis we split the data into a training set, consisting of all data from Fall 2008 through Summer 2015. The test was Fall 2015, which was only tested against once using each of the three predictive model variants, discussed later. Within the training set we specified Fall 2014 to be our hill-climbing validation set used to tune hyper-parameters, with Fall 2008 through Summer 2014 as the internal training set. After the best of each model type was identified based on the validation set, the hyper-parameters for each were used to train on data from Fall 2008 through Summer 2015 in order to make one-time predictions on the Fall 2015 test set. This regime was used in order to mirror the expected accuracy on predicting the next semester if the system were deployed during the previous semester.

Prediction Accuracy

The prediction accuracy is the most straightforward way the evaluate course prediction models. In the course prediction experiment, the prediction accuracy is defined as:

$$\Pr = \frac{\sum_{i=1}^{M} \#m_i}{\sum_{i=1}^{M} m_i}$$

M is total number of students, m_i denotes the number of courses student i take in the evaluation semester. $\#m_i$ denotes the number of courses we predict right in that semester. If the students take M courses in the evaluation semester, then we calculate what percentage of the top M course predictions

match the M courses taken by the student. All students with data in the evaluation semester are predicted, including freshman students with no previous data and students with data from previous semesters that could be leveraged to make personalized predictions.

Perplexity

Another suitable metric for our task is perplexity, which is a measurement of how well a probability model predicts a sample given the number of potential classes. The equation for perplexity is

$$Perplexity(Student) = \frac{1}{\#Student} \sum_{i \in Student} \frac{1}{\#X_i} \sum_{x \in X_i} \sqrt[k]{\frac{1}{P_i(x_k x_{k-1} \dots x_2 x_1)}}$$

Where X_i denotes our predictions for student I and Student is our evaluation set.

Model

In this section, we detail our course prediction models which take advantage of students' course enrollment history and give them suggestions for the coming semester. Since our models were trained with a dataset composed of actual enrollment data from Berkeley students, the predictions provide a peer-like suggestion representing the common next enrollments by students like them. Our task is similar to text generation tasks in natural language processing. However, instead of using words or characters as modeling units, we used course enrollment records for whole semesters as modeling units. In our models, each semester is considered a time slice and since there is no discernable order of course within a semester, we aimed to provide a representation of courses that acknowledges this.

We imposed that, in order to be involved in training, the student had to have at least two semesters of data in the training set. We also ruled out all students with more than 12 semesters (students who attended 4 years, including all 4 summers). This filtering allowed the recommendation to be slightly biased towards the enrollment behaviors of students who graduated approximately on-time.

We use two primary modeling techniques to conduct recommendation; a neural network with LSTM units, and an N-gram model as our baseline.

LSTM Recurrent Neural Network

A popular variant of the RNN is the Long Short-Term Memory (Hochreiter and Schmidhuber) [10] architecture, which helps RNNs train by the addition of several "gates" which learn to retain and forget the information from latent states. It has been proved that LSTM networks could be train more effectively than the simple RNN networks (Bengio, Simard and Frasconi; Gers, Schmidhuber and Cummins) [3,6] by mitigating the vanishing gradient phenomenon. The mathematical expression of LSTM network are:

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \quad (1)$$

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \quad (2)$$

$$\widetilde{C}_t = \tanh(W_{Cx}x_t + W_{Ch}h_{t-1} + b_C) \quad (3)$$

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C}_t \quad (4)$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \quad (5)$$

$$h_t = o_t \times \tanh(C_t) \quad (6)$$

Figure 11 illustrates the inner structure of a LSTM unit. f_t, i_t and o_t represent the gating mechanisms used in LSTM to determine which information to "forget", what to input to the next cell state and what to output from the current cell state. C_t represents the latent cell state and $\widetilde{C}_t$ represents a candidate cell state that is to update the next cell state。

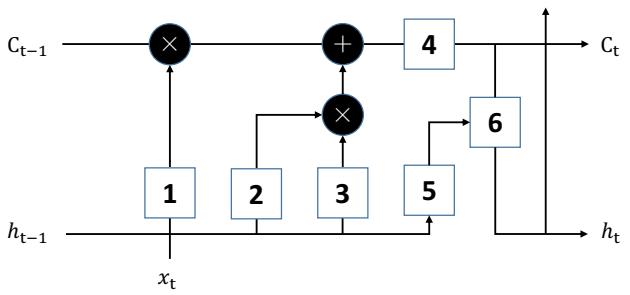

Figure 10. An illustration of an LSTM layer.An illustration of a LSTM cell where numbers correspond to previous numbered equations.

Our neural network architecture and optimization

The LSTM next course prediction models used in this paper were implemented using Keras (Chollet and Francois) [5] with Theano backend (Bergstra et al.; Bastien et al.) [2,4]. We implemented a basic single layer LSTM course prediction model using only course enrollments, a single layer LSTM using both enrollments and student major information, and a stacked, multi-layer LSTM model using both enrollments and major.

For the simplest course prediction model, we considered different semesters as different time steps for the LSTM layer. Our model takes all the courses within a semester as input, each course is represented by an index number and therefore correspond to a one-hot representation. Our input layer directly takes in the summation of all the one-hot representation of courses and we call this kind of course representation a multi-hot representation, reflecting the multiple courses present in a single semester. The model then converts the multi-hot representation to a low-dimension embedded vector using a fully-connected layer with linear activation. The use of this kind of embedded representation is common in natural language processing tasks and language modeling (Levy and Goldberg) [11]. Then the LSTM layer takes in the embedded semester representation at each time step and generates a representation of student course enrollment history. Finally, we use a fully-connected

layer with softmax activation to converts this representation into next course probability distributions.

$$P(y = j|x) = \frac{e^{xW_j}}{\sum_{k=1}^{K} e^{xW_k}}$$

At each time step we feed in the multi-hot representation to the neural network and use the next semester's multi-hot representation as the output labels to calculate loss for the back propagation algorithm. We use categorical cross entropy as our loss function.

$$H(p, q) = -\sum_{x} p(x) log q(x)$$

Since students have an unfixed number of courses for each semester, we built six output layers from the softmax output layer, each of them will be assigned with a corresponding label picked from the next semesters' course list and they will calculate the cross entropy independently but with shared weights. The back propagated loss will be the arithmetic average of these six losses. If a student has more than six courses in one semester, we will randomly choose six of the courses as the labels and remove the remainder; however, if a student has less than six courses, we will pad the unused labels with zeroes and specify that the loss for those unobserved outputs be weighted at 0 using sample weighting. To optimize our objective function, we use RMSProp optimizer with learning rate of 0.01, $\rho = 0.9$ and a gradient norm value of 0.3.

The two improvements to our simplest model are incorporating major information and stacking the LSTM layers. To incorporate major information, we generate a major one-hot representation vector and concatenate it with the course multi-hot representation. For the stacked LSTM model, we tried 2, 3 and 4 layers with dropout of 0.20 between LSTM layers to help prevent overfitting. The structure of the LSTM course prediction model is shown in Figure 12.

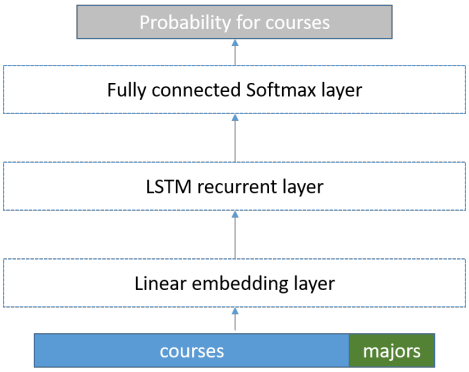

Figure 11. The structure of LSTM course prediction model

N-gram model as baseline

N-gram models are simple yet powerful on language modeling tasks. They break sequences from the training set into small sub-sequences with length N as grams. For a given size n, it can approximate the probability distribution of $P(x_n|x_{n-1}x_{n-2} \dots x_1)$ by building lookup tables for each sub-sequence of length n-1 and their following elements. The number of parameters of N-gram models grows respectively with the number n. We expect these kinds of models can be competitive course prediction models to our LSTM models.

We have two ways of generating the training set for N-gram models. The first way of generating training samples for each student is to concatenate all their course enrollment record according to the order of the semesters, just like our training set for our Course2Vec model. Each student has a semester number of n and our training set size will grow linearly with n. The second way to generate our training set, is to calculate the Cartesian product of all the semesters of each student, which means that we will get all the possible course enrollment sequences and a potentially more even distribution of course grams observed. In such away, our training set size will grow exponentially with the semester number n.

The first kind of generation has a reasonable space complexity but the second one has a better representation of students' course enrollment behavior. We did experiments on both and found that the N-gram trained on the Cartesian product generated dataset produced accuracies 3 percentage points better on average. We chose this N-gram version for all experiments hereafter.

Hyper Parameter Search

In the course prediction experiment, we tried various hyper parameters for LSTM model, including embedding dim [32, 64, 128, 256], number of LSTM nodes [64, 128, 256, 512, 1024], number of layers [1,2,3,4] and different number of grams [2-20] in N-gram model.

Results

Comparison of Prediction Accuracy and Perplexity among Different Predictive Models

In Table 4, we compare the N-gram model with three different LSTM models. These results suggests that the stacked LSTM model (with major information) performs best on prediction accuracy, with around 20%, which equates to showing 10 prediction results in the interface and having students, on average, find two courses of interest. Also, we find that incorporating major information of students can improve prediction results by around 3 percentage points. Surprisingly, the smallest perplexity is achieved in single layer LSTM models without involving major information. One possible reason is this model is relatively simpler than other models, so there is less variance in prediction.

measurements	prediction accuracy	perplexity
n-gram	11.9%	**3.8**
single layer LSTM (without major)	15.5%	11

single layer LSTM (with major)	18.1%	16.7
stack LSTM (with major)	**19.1%**	16.3

Table 4. Comparison of different methods on courses prediction

Comparison of Different Hyper Parameters of LSTM model

From Table 5, we try different hyper parameters in the stacked LSTM model. For the number of LSTM layers, we find that the prediction accuracy will first increase and then decrease, which might be suggested simple model may lack representation ability but deeper models are harder to be fine-tuned and have more tendency to overfit. For the number of LSTM nodes, more lead to better prediction results but larger perplexity. And for the dimension of embedding layers, it seems that in this case dimension 32 had better performance than dimension 64.

	The number of LSTM layers				**The number of LSTM nodes**			**Embedding Dim**	
	1	2	3	4	64	128	256	32	64
Prediction Accuracy	14.4%	**17.5%**	16.8%	16.1%	16.2%	17.0%	**17.2%**	**16.9%**	16.7%
Perplexity	22.6	15.6	18.0	**15.0**	**14.5**	16.9	17.2	**15.5**	16.9

Results on test set

After evaluation on semester 2014 Fall, we found the best hyper-parameters for our four course prediction models. To test their performances, we re-trained our models with the fixed hyper-parameter settings, and training set which contains course enrollment information from 2008 Fall through 2015 Summer and predicted our 2015 Fall test set. The results are shown in Table 6. The same order of model performance and magnitude was achieved on the test set as on the validation set, with the stacked major model on top and the n-gram baseline at the back.

measurements	**prediction accuracy**	**perplexity**
n-gram	10.8%	47
single layer LSTM (without major)	15.3%	**10**
single layer LSTM (with major)	17.6%	15.8
stack LSTM (with major)	**18.6%**	13.2

Table 6. Comparison of different models on the test set

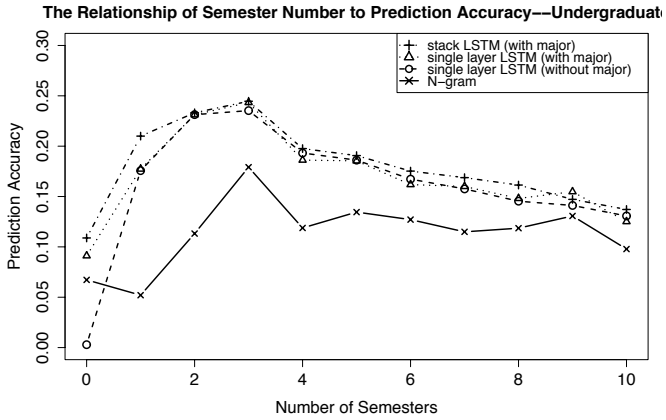

Figure 12. Plot showing the number of semesters of data an undergraduate student in the Test set had versus the prediction accuracy for each of our four prediction models.

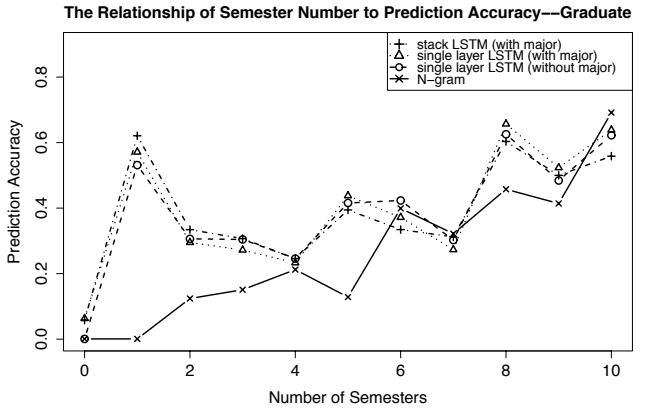

Figure 13. Plot showing the number of semesters of data a graduate student in the Test set had versus the prediction accuracy for each of our four prediction models.

We drew a line plot between the number of semesters for one student and course prediction accuracy for undergraduate and graduate students (see Figure 13 and Figure 14) in order to show how the accuracy of recommendation changes given the student is a freshman (0 semesters), sophomore (2-3 semesters), and so on. Looking at the undergraduate plot, we can see that prediction accuracy peaked around semester

three before going down. A hypothesis for this is that students' enrollment is mostly filled with breadth requirements in the first 3 semesters, which are more easily predicted than their electives taken in subsequent years.

CONCLUSIONS

We used a dataset made up purely of sequences of course enrollments and found that this behavioral data could surface interesting relationships between courses much the same way as word vector representations could in natural language. This rich representation served as a convenient model to query to allow students to find courses similar to overenrolled courses of interest. Using an LSTM on top of this representation, we were able to recommend courses to students with an accuracy of 2 out of 10 on average, a score not unlike those achieved in natural language prediction. The AskOski UI provides a portal into novel and personalized information about courses that students can utilize towards better informed and therefore more efficient course enrollment decisions. The system can be stood up at any college or University with non-grade course enrollment data and minimal resources.

A demo is available here[3], which any user can sign-up for (bottom of the page) with open sourcing not far behind.

REFERENCES

1. Arnold, K. E., & Pistilli, M. D. (2012) Course signals at Purdue: using learning analytics to increase student success. In Proceedings of the 2nd international conference on learning analytics and knowledge (pp. 267-270). ACM.
2. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., ... & Bengio, Y. (2012). Theano: new features and speed improvements. arXiv preprint arXiv:1211.5590.
3. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.
4. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., ... & Bengio, Y. (2010, June). Theano: A CPU and GPU math compiler in Python. In Proc. 9th Python in Science Conf (pp. 1-7).
5. Chollet and Francois. (2015). Keras. GitHub. https://github.com/fchollet/keras
6. Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471.
7. Complete College America (2014). Four-Year Myth: Make College more Affordable. Downloaded from http://completecollege.org/wp-content/uploads/2014/11/4-Year-Myth.pdf.
8. DeAngelo, L., Franke, R., Hurtado, S., Pryor, J. H., & Tran, S. (2011). Completing college: Assessing graduation rates at four-year institutions. Los Angeles: Higher Education Research Institute, UCLA.
9. Farzan, R., & Brusilovsky, P. (2011). Encouraging user participation in a course recommender system: An impact on user behavior. Computers in Human Behavior, 27(1), 276-284.
10. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
11. Levy, O., & Goldberg, Y. (2014). Dependency-Based Word Embeddings. In ACL (2) (pp. 302-308).
12. Li, Z., Tinapple, D., & Sundaram, H. (2012). Visual planner: beyond prerequisites, designing an interactive course planner for a 21st century flexible curriculum. In CHI'12 Extended Abstracts on Human Factors in Computing Systems (pp. 1613-1618). ACM.
13. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
14. Mikolov, T., Yih, W. T., & Zweig, G. (2013, June). Linguistic Regularities in Continuous Space Word Representations. In HLT-NAACL (Vol. 13, pp. 746-751).
15. Parameswaran, A., Venetis, P., and Garcia-Molina, H. (2011). Recommendation systems with complex constraints: A course recommendation perspective. ACM Transactions on Information Systems (TOIS), 29(4), 20.
16. Pardos, Z. A., & Kao, K. (2015, March). moocRP: An open-source analytics platform. In Proceedings of the Second (2015) ACM conference on learning@ scale (pp. 103-110). ACM.
17. SB-520 Student instruction: California Online Student Incentive Grant programs (2013) http://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_id=201320140SB520
18. Vialardi, C., Bravo, J., Shafti, L., Ortigosa, A. (2009). Recommendation in higher education using data mining techniques. In International Conference on Educational Conference, Cordoba, Spain, 190-198.

[3] http://maxwell.ischool.berkeley.edu:1338