Wolfbooks

A Textbook Recommendation Application based on Topic Modeling

Monica Metro
North Carolina State
University
3021 F Dorner Circle
Raleigh, NC
mgmetro@ncsu.edu

Zachery DeLong
North Carolina State
University
2305 Horizon Hike Ct
Raleigh, NC
zpdelong@ncsu.edu

Zhangqi Zha
North Carolina State
University
1800 Vienna Wood Dr
Raleigh, NC
zzha@ncsu.edu

Bikram Singh
North Carolina State
University
Page Hall, Raleigh, NC
bsingh8@ncsu.edu

ABSTRACT

Choosing a textbook for a class is something that many professors do several times a year, but the vast number of available textbooks for any given subject makes choosing one that both meets the needs of the class and meets the budget constraints of today's students is a challenge. In this paper, we propose two methods for automatically learning topics from books, a set of evaluations for those methods, and a system that will use those methods in a web app that attempts to suggest textbooks to a professor balancing covering all topics the course requires while still being economical for students.

1. INTRODUCTION

1.1 Background

One of the most basic tasks that a professor must do is identify what textbook to use for a given class. While some fields have textbooks that have become popular and are clearly the best in their subject matter, many fields (especially emerging ones) have no such exemplar and reading through all possible candidates would be too time consuming to be feasible. It would be a complex enough problem if there were not already a massive number of textbooks on sources such as Amazon, but self aware professors have attempted to use textbooks that are less expensive with hopes of allowing lower income students to attend more easily. This admirable intention serves to increase the already considerable time needed to find appropriate books, and there is no obvious heuristic to apply to searches to limit the field.

In the interests of making students lives less expensive, have expored a system which can identify topics in books automatically, and which, given a set of requirements from a user (our intention is for the system to be used by a professor, but others may find value in it), can suggest options of varying expense and completeness for a given set of topics. To do this, we propose exploring word clusters generated by Doc2Vec, a family of common natural language processing (NLP) algorithms that attempt to cluster similar words, and topics generated by latent dirichlet allocation (LDA), another common NLP algorithm that directly attempts to identify topics in text, to automatically infer the content of a set of textbooks.

We then intend to build a web app that will allow a user to specify a set of topics and which will use the mentioned algorithms to make suggestions while minimizing the cost of said textbooks. The actual web app will be a fairly simple single-page web app developed in Angular4 and using Bootstrap to hasten development of a modern UI.

The team begain this experiment somewhat biased toward LDA as the algorithm expected to perform, and this initial bias turned out to be more or less correct. After spending significant time training and validating a useful Doc2Vec model, the team was unable to build a useful topic mining algorithm on top of Doc2Vec.

1.2 Deviations From Initial Plan

Original Plan.

Table 1: Original Schedule

Week started	Task
1-2	Front-end webapp
1-3	NLP CLI
3-4	Suggestion Engine

The team's first attempt at scheduling out the project was focused on using agile methodology. The overall architecture was broken down into components that could be designed and the work was scheduled for two week sprints.

This schedule is shown in Table 1. The intent was to develop the NLP algorithms, suggestion engine, and web app in parallel as much as was possible, resulting in a fully developed web-app at the end of the project. The original hope was that the algorithms themselves would be fairly straightforward to work with and that it would be possible to apply re-usable algorithms easily and quickly to develop what was needed.

Challenges.

It became apparent very quickly that this schedule was not going to work. The team's experience (summarized in table 2 with data mining generally and machine learning more specifically was not up to par. While the team had some experience in software engineering, the specific NLP tasks that were required took more time and effort than was anticpated, as discussed in Section 1.3.

Table 2: Developer Skills (years)

Name	Data mining	Programming	NLP
Zach DeLong	1	5	0
Monica Metro	0	0	0
Zhangqi Zha	0	2	0
Bikram	?	?	?

After the project was broken into the above sprints, the team realized they lacked the expertise to split the work up this way and the decision was made to organize the project differently with the intention of producing a smaller subset of the original goal with more fully fleshed out functionality.

The Spiral Model.

To attack the unforseen complexity of this project, the team chose the spiral model of software development. The spiral model is a form of risk-driven development where the most risky parts of a project are identified early and planned into prototypes that can inform the final product. See the breakdown of the prototypes the team chose in section 1.3.

This afforded the team the ability to have regular check-ins and to measure progress against the prototypes. The team convened two to three times a week to do pair programming and to check progress against the prototypes. When combined with regular check-ins with the TAs, the team was able to build prototype programs to perform NLP on a set of training data and to build various measures of effectiveness of those models.

1.3 Planning

Prototypes.

The prototypes the team identified for this project are listed in table 3. The team identified the learning algorithms as the most risky part of the project, primarily because of the team's lack of experience with natural language processing algorithms.

As discussed before, these algorithms have succesfully been used to mine topics for some time, but the particular problem of identifying the topics in textbooks for search recommendations is fairly novel. This process requires a data set that matches up well with the desired topics, which will come into play in Section 5.

One other major point of risk is developing the Doc2Vec model. The Gensim API contains a Doc2Vec model but it does not expose parts of the API to make the clusters directly available, which is expanded on in section 5

Timeline.

The team completely had to re-evalaute the sechedule to accomodate the prototypes desired in table 3. The team intended to iterate every week, attempting to finish one prototype each week on the way around the spiral model. This often worked, and prototypes 1, 2, and 3 were finished in the first three weeks of the project. Prototypes 3 and 4 were develoed in parallel over the last several weeks of the project and were followed immedaitely by implementing the evaluation plan in section 5.

2. LITERATURE REVIEW

We propose two methods for analyzing and storing topics from books automatically: LDA and Doc2vec.

2.1 LDA

This section provides a brief overview of Latent Dirichlet Allocation and how the team intends to use it.

LDA uses a Bayesian clustering approach to find topics relevant to each document. [3] The input to the algorithm is a book (or later, a series of books) which consists of a collection of words.

LDA's core assumption is that a given book (or set of books) contains a number of topics that can be represented as keywords. The number of topics associated with a book can be variable, and can be changed for better optimization. In this association, the words comprising the text are thrown into a "bag-of-words" model such that grammer and word order are disregarded, but multiplicity is measurable. The calculated topics found from this bag-of-words are "latent" variables, that is they are not directly measurable but indirectly observed.

The performance of the method depends on the initial assumptions made. For example, some have assumed that books are random mixes of topics [2]. The number of topics can initally be chosen as constant for a given book, or as a function of a chosen Poisson distribution. [2] Topic distribution in the book is assumed to be a sparse Dirichlet function.

A sparse Dirichlet function implies that we assume that a topic will be discussed only in a small breadth of pages in the book, and also that words relating to this topic will predominantly figure in these pages.

The algorithm looks to identify unique words to identify as topic names. Words like "the" and "a" are common and will occur with equal probability throughout the text. However, in say one chapter of the book some words may occur much more frequently in that particular chapter than the rest of the book, which means that the probability of these words in that chapter is more, which then the algorithm can use to choose a topic name and also a list of words related to that topic.

2.2 Doc2Vec

2.2.1 Introduction

Doc2Vec is an implementation of the Paragraph Vector (PV) introduced by Mikolov and Le in 2014 - an unsuper-

Table 3: Prototypes and Schedules

Prototype	Requirements	Risk (1-5)	Complete
1	Implement CLI to make testing of models easier. Implement LDA algorithm using Gensim on static text files.	5	+
2	Prototype 1 Parse Stack Overflow data to text files for input to prototype 1 Save/Load model so multiple experiments can be run more quickly Adapt Prototype 1 to operate on Stack Overflow data.	4	+
3	Prototype 2 Prototype database models/access code. Prototype similarity measures to be used with learning algorithms	4	+
4	Implement Doc2Vec Implement Doc2Vec with stack overflow data	5	+
5	Implement code to parse PDFs of textbooks Implement connection to library (either REST API or a folder of PDFs Apply trained models to textbooks Save modeled topics in the database	4	-
6	Write algorithm to parse saved topics Write REST API to serve above algorithm	2	-
7	Build web-app front end to the above suggestion service	2	-

vised algorithm that generates vector representations of text inspired from research of vector representations of words via neural networks. [5] By concatenationg or averaging vector represenations of words with other word vectors, the resulting vector can be used to predict future text. This technique offers one distinct advantage to bag-of-words models like LDA: the ability to record semantics due to the mapping of the word vectors into a vector space that allocates words of similar meaning together. For example, the words "powerful, strong and Paris" are equally distant when considered by a bag-of-words model, however "powerful" and "strong" are semantically close and should therefore be less distant from one another. [5]

PV is "unsupervised", which means that it predicts words in a paragraph and then uses these predictions to attempt to form structures of related words without any knowing of accuracy of the structure(s). The PV utilizes fixed length feature vectors learned from text sources of variable length.

2.2.2 Models

Mikolov offers two models of PV that are based on the implementation of Word2Vec: Distributed Memory Model (PV-DM) and Distributed Bag of Words (PV-DBOW).

PV-DM is similar to the Continuous Bag of Words (CBOW) model in Word2Vec such that is predicts the next word based on the given context. Paragraph vectors are concatenated with word vectors from that paragraph. Each word vector is representing a word from the given context and a new vector is generated by concatenating all of those word vectors together to predict other words. These word vectors are able to remember context and semantics. The paragraph vector allows the algorithm to remember the topic or 'label' of the paragraph, whether it is a sentence or a long document.

PV-DBOW follows the skip-gram model of Word2Vec. In this model, instead of concatenating the paragraph vector with the word vectors formed in close proximity in order to predict the next word, random text from the paragraph is selected and a random word is selected from that text. Be-

cause this model does not save the word vectors and therefore does not retain as much information about the context and semantics of the text, it requires less storage.

2.2.3 Application

Mikolov and Le recommend using a combination of PV-DM and PV-DBOW for consistently accurate results. [5]

2.3 Related Work

Doc2Vec was recently tested as the algorithm behind a recommender system with the goal of recommending unseen Twitter messages (tweets) relative to a user's typical activity and interests. A graph based link prediction method was used to infer which unseen tweets the users would like by "predicting the presence or absence of edges between nodes" on the graph. [1] K values were tested in intervals divisible by 5 ranging from 5 to 35. The system preformed the best when K=30.

An LDA approach of a tagging system was implemented to improve tag recommendations. [4] Tagging systems are often used for organizaing a user or organization's data. When this data is shared, tagging can be used to find or search for relative content.This system is similar to our proposed book recommendation system that uses subject topics to search for relative material. The initial tag data set was comprised of a large sample creating a diverse tag set. This set was used to elicit latent tags of sources that did not have that many tags to describe the document. Often, the tags recommended were more specifc. By increasing the number of tags and increasing specific tags, their approach contributed to the usefulness of searching for new content.

3. ARCHITECTURE

3.1 System Design

Overall System.

We propose building a layered system where each piece of the funcionality of the overall application is built into one layer of the overall design.

This design gives us the ability to build the riskiest, most difficult pieces of functionality early, while slowly building up to an application taht is usable. It also allows us to parallel path much of the development of the overall system. One person can easily be working on one layer of the system without impacting other layers, as long as interfaces between layers are well designed and documented.

The other primary advantage of the layered model detailed in the following sections is that it makes validation possible at each layer of the model. As we will get into in the following sections, the overall application will need to be validated as it is built, for the accuracy of its NLP algorithms, the usefullness of its suggestions, and the user-friendlyness of it's UI

See figure 3.1 for a detailed breakdown of the overall system design. The arrows in this diagram represent the flow of information from one layer to the next. Notice how no layer communicates with anything past its surrounding layers.

Textbook Library.

NLP Algorithm.

The "lowest" level of the stack we have designed is the suggestion algorithms. This layer has the responsibility of identifying the topis found in textbooks using trained models. Because all modern topic modeling algorithms are extremely slow to run, analysis can not be done in real-time. To work around that, we propose buildign a command-line application that could be scheduled on a server to run on some kind of regular interval and set against some external textbook parsing API. See section 3.2 for a more detailed breakdown than this overview.

Database.

The database sits above the NLP algorithms, but below the rest of the stack. Its purpose is to hold the topics, textbooks, and author information mined from our external API. To house this information, we have assembled a database using the Python library Peewee as an ORM. When run, Peewee will build out the tables listed in figure 3.1. The app proposed will use the database as a repository from which to make suggestions rather than natively running the NLP algorithms, which should help to improve the performance of the overall app.

Suggestion Algorithm.

The suggestion layer is the last layer of the application before the interface itself. To make suggestions, the team intends to build out a REST API built on top of the database previously discussed. The API should accept a list of topics to search for along with two weights: one for the wieght of topic coverage and one for weight of cost.

The weights serve as tuning parameters and may or may not be displayed to the end-users. The weights are them multiplied by their respective parameters and summed to come to a final priority number, which can then be considered a measure of the "fitness" of a particular textbook for use, where larger numbers are more fit. The fitness numbers should also be normalized to a scale between 0 and 1,

so that the "fitness" of different searches could be compared in a meaningful way.

The proposed formula for calculating the priority of a suggestion is detailed in equation 1.

$$Priority_i = a \times num\_topics_i + (b \times max(cost) - cost_i) \tag{1}$$

Web App.

The final componenet in this proposed architecture is the user interface. For the purpose of this project, the team proposes building a simple webapp using AngularJS. In this design, the actual web app needs to do very little; it is merely acting as an interface to the previously mentioned API. It does not need to directly connect to a database or to perform complex calculations, it simply needs to provide a simple, easy way to input a set of search topics and to display the results of the REST API to the users.

AngularJS is well positioned to be useful in this situation. It provides convenient APIs for the display and manipulation of complex interfaces, without providing significant bloat to the application stack duplicating the features of the previous components. It is also structured as a model-view-controller system, so our manipulation of the REST api can be well and cleanly separated from the presentation, allowing us to develop other applications easily in the future.

3.2 CLI Design

The NLP algorithms are the focus of most of the rest of this paper. To make it easier to test and evaluate these algorithms, the team designed a CLI using the Python package Click. Once the CLI was up and running, the learning algorithms were implemented using the package Gensim. Wrapper classes around the Gensim models were built using a variant of the strategy pattern, so that our implementations could easily be swapped back and forth during testing.

3.3 Use cases

The team envisions fairly few use cases for the proposed system, as the system's scope is fairly well defined. The system recommends textbooks to users based on the textbook cost and the topics covered in that textbook. With that very limited scope in mind, the webapp needs to support three use-cases documented in table 4

4. PROGRESS

Prototype Progress.

The original plan for implementing our system did not adequately address the issues we had with implementing our learning algorithms, so the team quickly pivoted to a more risk driven approach as outlined in section 1.2.

The team currently has implemented prototypes 1-4, which means that the lion's share of NLP processing has been done. We now have a script that will transform Stack Overflow dumps into something that our learning algorithms can more easily and efficently process, as well as a NLP implementations of both LDA and Doc2Vec that are able to act on the processed data and which implement various accuracy measures that we have used to detail their usefulness. We also have save and load functionality worked into our models so

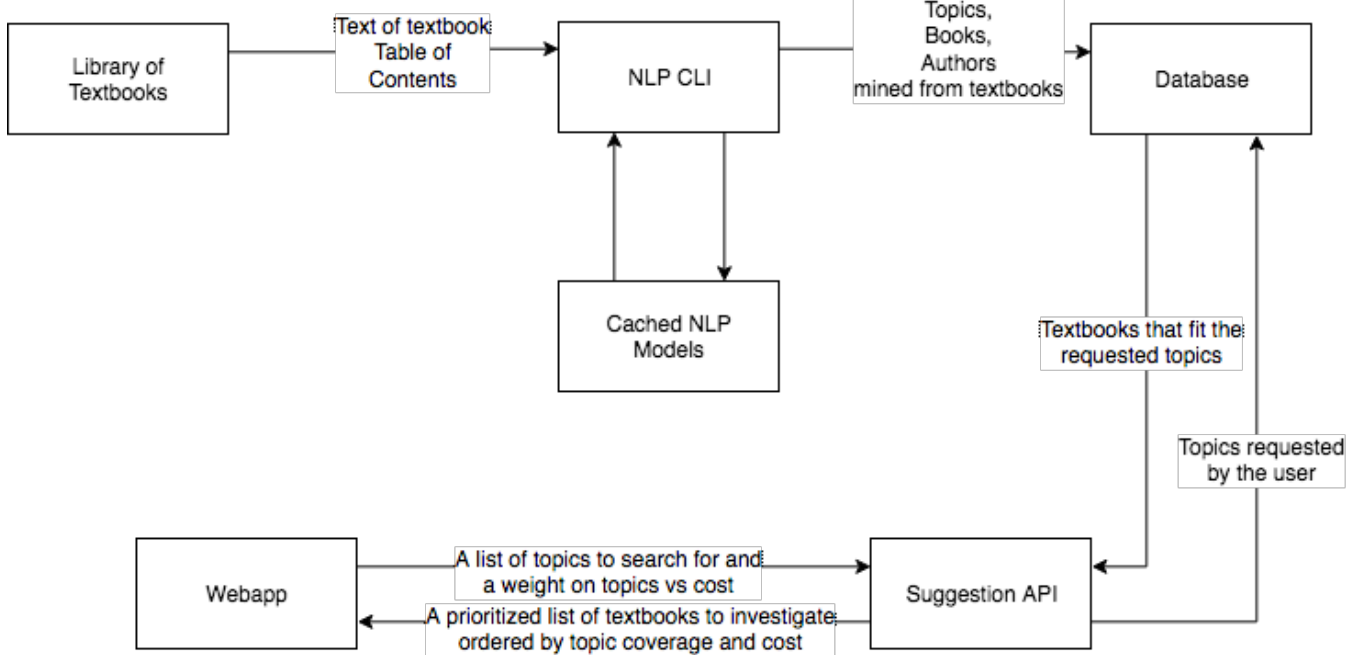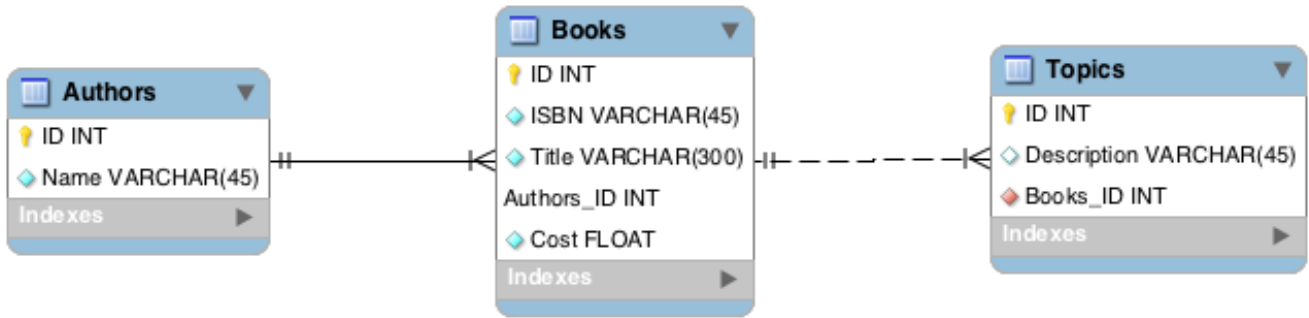Figure 1: Visualization of layered architecture

Figure 2: ERD of Proposed Database

that the models can be reused without the need to train the model on every run, which has proven to be one of the most useful features during testing.

Challenges.

On the way to getting those prototypes running, we ran into a number of challenges, chief among them the fact that the team had no experience in this field, as outlined in seciton 1.2. Another major complication we ran into was the simple complexity of these algorithms. The Gensim library provides amazingly powerful implementations for re-use, but that power comes at the cost of complexity which their reasonably high quality documentation does not entirely weed out. The third major stumbling block was the fact that we intended to use Doc2Vec to mine topics out of text—something it does not natively do! The team came up with a somewhat novel approach to this as outlined in seciton 5 after some time, but this was a major challenge of our research and was met with mixed results. The last major challenge was one of time. Not time to finish the projec, though we could easily have listed that, but the execution time of the NLP algorithms. These algorithms are both extremely expensive, and to run them on the data sets required can take significant portions of time.

To respond to those challenges, the team spent a lot of time in research and in documentation. We spent a lot of time studying the research and trawling through Gensim documentation To solve doc2vec we came up with a somewhat novel approach to modeling topics. We tried to optomize LDA and implemented caching of models to make retraining unnecessary once a reasonable model was trained.

5. VALIDATION

In this section, we present the evaluations that been performed on our solutions. This including two parts: 1) methods that used to evaluate our core topic learning models LDA and Doc2Vec and 2) methods used to evaluate the whole solution as the final product.

5.1 Evaluating Learning Models

Table 4: Use Cases

USER	TASK
Professor	Search for textbooks with a given set of topics
Professor	Search for textbooks with a given set of topics using an increased weight on providing an inexpensive textbook
Professor	Search for textbooks with a given set of topics using an increased weight on matching all topics requested

The performance measure is a typical way to evaluate machine learning models. It is the measurement that will make of the predictions made by a trained model on the test dataset. Performance measures are typically specialized to the class of problem, for example in this paper, while we using LDA to deal with NLP topic modeling, we will use likelihood score and clustering result to evaluate our LDA model.

Based on how to split the data into training and testing, evaluate machine learning models usually involves hold out, cross validation(CV) and leave-one-out.

Hold out method is a simple split of dataset, for example, 70% of the instances are used for training the models, the rest 30% for testing the model. In the CV method, it first involves separating the dataset into a number of equally sized groups of instances (called folds). The model is then trained on all folds exception one that was left out and the prepared model is tested on that left out fold. The process is repeated so that each fold get's an opportunity at being left out and acting as the test dataset. Finally, the performance measures are averaged across all folds to estimate the capability of the algorithm on the problem. Leave-one-out is a special case of CV, where folds is equal to data points.

In our original plan, we may need CV to run our model evaluation, because our dataset is not large enough and the computational difficult is not critical. But with using only stackoverflow data, we have plenty of dataset. So the decision is using hold out method. We tested our models using different size of dataset.

Dataset Description.

Ultimately, we plan to use actual open source books as our primary dataset, which including over 100 books in computer science discipline. As our work went through, we plan to use some other dataset that have pre-labeled topics to train our models as the risk reduction process. With this in mind, we download stackoverflow posts data. The original posts data is in xml format, about 62G Bytes. Each post contains post body, post tag, post time and other attributes.

Data Preparation.

StackOverFlow dataset is a signal file in xml format. In order to use these data to train our models, we need to do some data preprocessing to clean up the data. In the topic modeling, the input data is text itself, which is post body in the case of stackoverflow dataset. After training, during the test phase, the extracted topics from the test dataset using the trained model, will be compared with the actual topics, which are the post tags. In summary, the first step of data preparation is to extract post body and tags from raw data. Normally, with the text corpus ready, the next data preprocessing would be removing punctuations in the text corpus, and changing words to lower cases, but these steps have been included in our models. So the input to models are the parsed post body text files.

Evaluating LDA.

One typical method to evaluate LDA model is to calculate the likelihood probability of the testing data(hold out data in our case). We used perplexity to calculate and estimate the likelihood. Perplexity is defined as the reciprocal geometric mean of the token likelihoods in the test text corpus given the trained model. Lower values of perplexity means lower misrepresentation of the words of the test files by the trained topics.

We first parsed the stackoverflow data into several groups of data. Each group contains different number of text files. We would like to evaluate the model performance with the increasing of data size.

As we found out from the first experiment, the likelihood is not increasing significantly, but the train time is extremely longer. After we analyzed the results, it is possible that the training data has too many topics which has less common with the test data. Therefore, we designed the second test method.

Secondly, we parsed the stackoverflow data with filter out those posts do not contain any tags in a five topics set. The set contains tags: python, java, javascript, database and sql. With this less sparsity dataset, we have been able to train the model and test it with a better result.

Table 5: Test Result of LDA Model

Epoch	LIKELIHOOD
1	0.5443
10	0.6584
100	0.9189

Evaluating Doc2Vec.

TO DO

Sentiment Prediction Accuracy.

Information Retrieval.
One way to verify that our implementation of Doc2Vec is correctly mapping word vectors to other semantically similar word vectors is to test the trained model on an information retrieval task . [5] The data collected from stackoverflow is tagged with different topics that pertain to the content. The tags can then be used to organize the content via topic. The Doc2Vec model should report two documents that pertain to the same topic as less distant to each other than a third document that focuses on a different topic.

5.2 Evaluating Application

Pre-Surveying.
To evaluate our project idea, we conducted a pre-surveying to see what the problem we are facing and how people reacting about our proposed solution. We asked our participants how long they will spend on find the right textbook, what aspect do they think is most important when choosing a textbook (except the content, our learning model will do the best of this part). See Table 6 for a breakdown of the questions to ask. Among the 29 responses, we found that

Table 6: Questions in Pre-Survey

NO	QUESTIONS
1	Are you a student or professor?
2	How much time do you spend on choosing a textbook?
3	How do you choose a textbook?
4	What aspect do you think is most important when choosing a textbook (except the content)?
5	Would you consider an app that will give you recommended textbooks base on your course syllabus and your preference?

73.3% responded as they interested to our application to recommend the textbooks. The answer to most important aspect during the textbook chosen, review/ratings and sell price were the highest concern. These preliminary results give us the guidance to design the application and to improve the core learning models. See Figure 3 for detailed results.

Experiments and Post-survey.
After we build the application with the command line interface and core learning models, we ran the experiments with different users in a focus group to evaluate the solution. We present our work to our peers to evaluate, after they try our application, we used google form to collect their opinions and comments on our application. Table 7 is the breakdown of the survey questions.

6. CONCLUSION

6.1 Process Conclusions

This project is the first time anyone in the team applied the principles of risk-driven development to their work, and
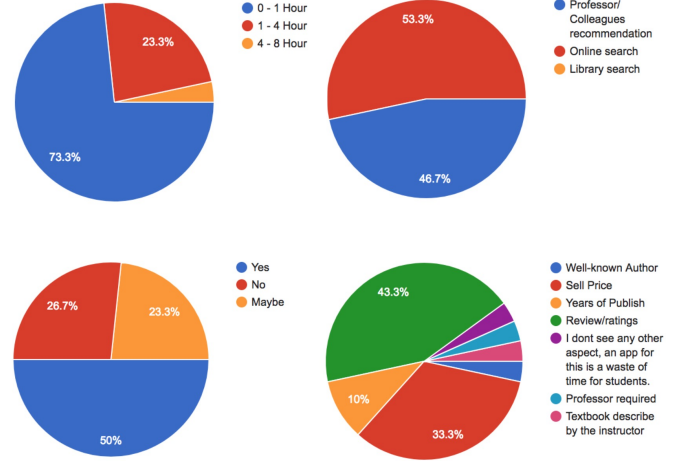
Figure 3: Summary of Pre-survey

Table 7: Questions in Post-Survey

NO	QUESTIONS
1	I was able to run the test code (python analysis.py –verbose) with the documentation provided.
2	I was unable to understand the structure of the code.
3	What would you change about this CLI?
4	What would scare you about working on this program or the resulting webapp?
5	What do you like about this CLI?
6	Commnets

the results were fairly satisfying. After the initial ambitions gave way to the true complexity of the project, the team was unsure of how to proceed, and this process provided a prescription to help dig out of the problem and into a useful series of prototypes. The prototypes themselves also served to illuminate the unknown risks quite well. For example, it was more or less assumed at the outset of the project that third party libraries would make the implementation of learning algorithms trivial and only a few days worth of work, but the process of implementing and tweaking those algorithms actually consumed the majority of the team's time.

The group's meetings were a somewhat sensetive topic as well. For the first two weeks of the project, the group met only once a week, which was not often enough. After that period, the team met twice a week for at least half an hour (and most oftem more) to do some pair programming and to brainstorm our solutions to problems encountered. This proved to be closer to optimal and significant progress started being made after that change.

The team also made significant use of the chat app Slack. Since the entire class was already using this tool, the team created a private channel in the course's Slack server and used that as the primary touchpoint for communication. Random questions, brainstorming, research, and general coordination was quite straightforward in Slack.

One more negative organizational conclusion was that the team could have made significantly better use of an issue-tracking system. Peer review revealed some platform specific issues and it would have been easier to coordinate the response to those challenges if there was one central repository to track progress.

6.2 LDA

As expeted, purely based on the accuracy of the topics, LDA is the clear winner in our battle of learning algorithms based purely on the measures presented. That said, it has some serious downsides. While it should be considered a positive that the algorithm is capable of learning topics unsupervised, the need to specify the number of topics mined limits the flexibility.

It is also worth noting that LDA can take significant portions of time to train in a casual computing environment like a laptop. There are some configurations of parameters that performed well in evaluation but took upwards of 20 minutes simply to train on a few thousand short Stack Overflow posts. This can be mitigated somewhat with proper tuning and the algorithm could be optomized for a high-performance environment but it is something that seems to be inherent in how to train the algorithm.

6.3 Doc2Vec

6.4 Future enhancement

The problem of textbook suggestion is fairly general and lends itself well to a number of different directions. We would like to start this app off with a fairly limited set of functionality, just the suggestions and a link to the FOSS library holding the book in question, but enhancements such as amazon integration would be One major enhancement would be the introduction of a rating system to the suggestion engine. Another major enhancement would be including links in the UI to buy the books. A third major enhancement would be evaluating other topic modeling algorithms to see how they stack up, such as IDA. Another enhancement would be to test an even more rudimentary topic modeling method, such as simply representing topics as lines in a table of contents. Essentially asking the question of whether or not fancy algorithms are even more useful that simple text search for this domain.

7. REFERENCES

[1] M. Z. Amiri and A. Shobi. A link prediction strategy for personalized tweet recommendation through doc2vec approach. *Research in Economics and Management*, 2(4):14, Jul 13, 2017. Doc2vec works better than other âĂŸapproachesâĂŹ with tweets.

[2] A. Y. N. David M. Blei and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2000.

[3] M. S. Jonathan K Pritchard and P. Donnellyu. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, June 1, 2000.

[4] R. Krestel, P. Fankhauser, and W. Nejdl. Latent dirichlet allocation for tag recommendation. RecSys '09, pages 61–68. ACM, Oct 23, 2009.

[5] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. May 16, 2014. Paragraph vector (doc2vec).