CSC 510 Software Engineering: Project 1

Group-P

Monica Metro North Carolina State University
1932 Wallamaloo Lane
Wallamaloo, New Zealand
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 that balance hitting all topics the course requires while still being economical for students.

1. INTRODUCTION

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, we propose exploring a system which can identify topics in books automatically, and which, given a set of requirements from a user (usually a professor), 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.

Initially, the team expects LDA to find more appropriate topics than Doc2Vec simply because it is better purpose built. That said, Doc2Vec uses a sophisticated neural-network, which may behave better on shorter passages such as a table of contents or an index, allowing us to parse less of the book while still getting useful topics.

2. LITERATURE REVIEW

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

2.1 LDA

In this section, we describe the Latent Dirichlet allocation(LDA) method for storing topics against books.

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

This method assumes that each book (or books) consists of topics and that each topic has several key words associated with it. The number of topics associated with a book can be variable, and can be changed for better optimization. In this association, words are the measurable variable whereas the topics 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 [**?**]. The number of top-

PLDI '13 June 16–19, 2013, Seattle, WA, USA

ACM ISBN 123-4567-24-567/08/06...$15.00

DOI: 10.475/123_4

ics can initally be chosen as constant for a given book, or as a function of a chosen Poisson distribution. [?] Topic distribution in the book is assumed to be a sparse Dirichlet function.

2.2 Doc2Vec

Doc2Vec is a model that extends an existing model, Word2Vec. Doc2vec mirrors Word2Vec in most ways except that it adds context information. [1]

Doc2Vec is an implementation of the Paragraph Vector (PV) introduced by Mikolov and Le in 2014 - an unsupervised algorithm that generates vector representations of text. [?] The PV utilizes fixed length feature vectors learned from text sources of variable length. 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. This is an advantage over the traditional Bag of Words model which does not account for either resulting in each word having the same amount of weight to each other. Consequently similar words are not mapped close together. 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. Because 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.

2.3 Related Work

3. PROPOSED SOLUTION

To tackle this problem, we propose a three-part system that will go about recommending books via a web app. The first component of this system is a recommendation engine, which will implement the above algorithms in some way and store their results (the topics) in an indexed database which we can summarily search (see section 3.1). The next component will be a recommendation engine which will implement our recommendation algorithm (see section 3.2). The third component will be a web app that will allow the end-user to search the database of topics and organize them into a useful search of books. It will then present a list of suggested books using the infrastructure mentioned earlier.

To host our project, we have begun work on an Ansible automation script that will communicate with NCSU's VCL environment. It is our intention to use this script to provision our needed hosting resources and to tear those resources down when they are not needed.

3.1 Book parser

The first major component of the system is the book parser. We intend to implement this taking advantage of the pre-built libraries available in Python such as SKLearn, Pandas, and others. The goal of the book parser is to implement one of the two algorithms discussed in section 2 and to store the parsed topics in a database for searching later. This will be implemented using the command pattern, which provides a simple interface to implementing different algorithms in code and allows us to easily swap between implementations.

The books being parsed will come from a freely available online database of textbooks. We have found a number of websites such as openstax.org and onlinebooks.library.upenn.edu that offer free libraries of textbooks that we are evaluating for use for this project. It is our intention to use a REST API or something similar to poll for books to analyze.

We are also exploring how much of the books we need to parse. Parsing entire books for topics is infeasible as the number of books grows (and as our introduction pointed out, there are plenty of books to parse) but we need to ensure that we have enough text to encapsulate all of the important topics the books cover, and that we have enough text to reliably train our algorithms on. To test this, the team intends to do testing before we test the actual application to see how much training is required when parsing the table of contents only, the index only, and any available summaries. We also intend to test some combinations such as the table of contents and the index together, to try to isolate what section might be most useful for training.

Because analyzing entire libraries of books is not something that we can do in real time, we plan to cache the topics parsed from these algorithms in a database to be referenced in the future. We are evaluating MySQL, Postgress, and an object-oriented DB such as Mongo for this task. This will allow us to train the algorithms on the books ahead of time and will make searching for books based on topics much more efficient. It also allows us to do routine similarity analysis on the topics of books to identify potential synonyms between topics. In other words, we could store the topics from two different books in the same database tables we are exploring algorithms that might then allow us to identify similarities in topics mined and then link the books to the same topics for future analysis. That said, this is a very lofty goal in an already complex project, and similarity analysis may need to be cut for time constraints.

3.2 Recommendation Engine

The next step in our problem is to use the topic analysis data outlined in the previous section (3.1) to suggest books based on the topics covered and the cost. This algorithm will be written in Python as well.

Conceptually, this algorithm will need to take in a set of required topics and weighting for topics vs price. This value will be used to favor inexpensive books over complete topic coverage in our algorithm. It will then search its algorithms for books that meet the required topics. The algorithm should then rank the books using the weighting mentioned earlier, then it will sort some number of the top results and give them back to the UI.

3.3 Web App

The web app is the final component of this architecture and it serves to tie the previous two components together.

Figure 1: Mock up of search setup screen

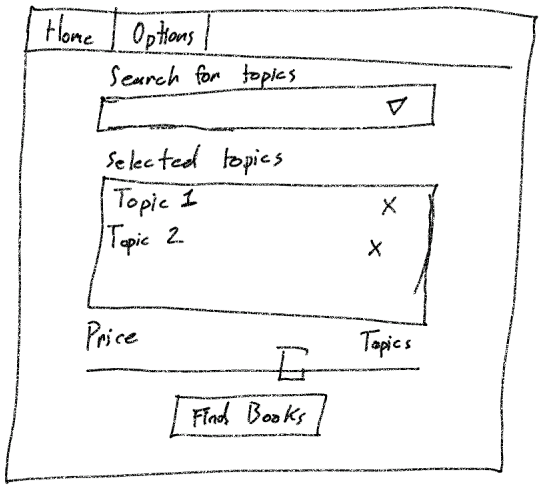
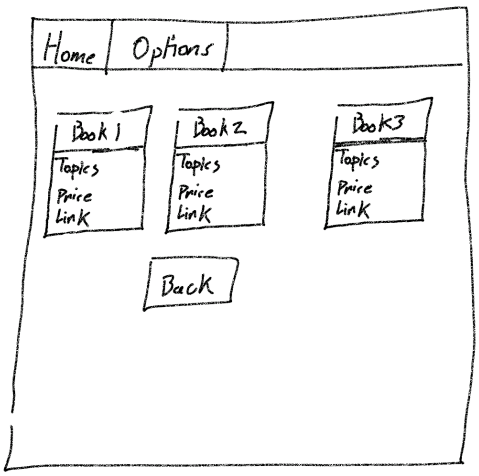


Figure 2: Mock up of search results screen



The app provides the UI to the algorithms, allowing professors to log in and select a set of subjects they wish to teach on, then it should present them with a UI for searching for topics in our database. It should then allow them to create a list of topics to suggest books for. Once the list has been generated, the tool should search against the book databases it has access to using the algorithms outlined in the previous sections.

We will develop this app in either the MEAN stack or some variant thereof. (Does it make more sense to just use Django since we are writing learning algorithms in Python?) The front-end will be developed using Bootstrap to save time in developing a useful, mobile friendly UI. In the end, the app itself will be a single-page web app that includes minimal authentication features. The real complexity of this app is in the learning algorithms and the suggestion engine, so the front end of the app itself needs only to be a client to reach out to those algorithms.

4. EVALUATION PLAN

In this section, we will talk about methods to evaluate our solutions. This including two parts: methods used to evaluate our topic learning model and methods used to evaluating textbook suggestions which is our final product.

4.1 Evaluating Topic Modeling

In order to carry out a comparative evaluation for a set of candidate recommendation algorithms, evaluation metrics will be used to measure some quality and performance features, in our case, we will use prediction accuracy to evaluate the models. The most popular metric that is used for evaluating prediction accuracy is Root Mean Squared Error (RMSE). RMSE is used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed. We will need to run this RMSE analysis on a data set with labels, and we are currently evaluating available data sets on Kaggle for fitness for this task.

$$RMSE = \sqrt{\frac{\sum_1^n (\widehat{y}_i - y_i)^2}{n}} \tag{1}$$

4.2 Evaluating textbook suggestions

4.2.1 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 1 for a breakdown of the questions to ask.

Among the 29 responses, we found that 73.3% responded

Table 1: Questions in Pre-survey

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?

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 **??** for detail results.

4.2.2 *Experiments and Post-survey*

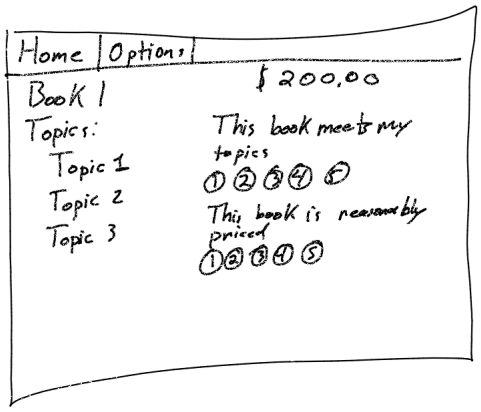
After we build the application UI and core learning model, we plan to run a few experiments with different groups of users to evaluate the solution.

We have briefly discussed how we intend to validate that our model suggests books well and the usefulness of our application overall, but we have not yet discussed how we intend on validating whether or not the textbook recommendations that our algorithm make are topical (cover the topics) and are frugal (are less expensive than other options). To do this, we intend to do some focus grouping easier by some custom UI in our web app.

We intend to sit down a group of volunteers with our app and to give them access to our app with an anonymized user. They will be asked to find an appropriate textbook for a class using a provided list of topics. After choosing the appropriate topics and setting their own weighting on frugality and topic coverage, they will be asked to parse the results and rank their fitness for the particular purpose on a scale from 1 to 5 (with 5 being perfect fit) and rate their cost from 1 to 5 (with 5 being perfectly inexpensive compared to the alternatives). After rating the books that were suggested, the user will then be presented with the top 5 books that did not make the suggestion cut, and will be asked if any of these books should have replaced books that were presented and why they should have been on the list (for example, was the book expensive, but did it offer better topic coverage, or was there something unique about it that we should factor into our analysis). The test can then be repeated with the other algorithm.

This test will answer several questions. First, it allows us to evaluate if the topic mining is finding semantically appropriate items. By presenting the user with the top choices, then choices that were not considered optimal, we are attempting to see if the algorithm should have found a different book more relevant. We are also attempting to see if a book that was more or less expensive should have been on the list. More importantly, though, by rating suggestions on a 1 to 5 scale, we can easily compare the ratings of suggestions made by LDA to the ratings of suggestions made by Doc2Vec to see if there is a significant difference between them.

Figure 3: Mock up of book details view with survey question radio-buttons



5. CONCLUSION

5.1 Anticipated challenges

There are a few potential major pitfalls that we are trying to plan our way around in this project. The first and most glaring problem is that the team's staff has very little experience with NLP in general, so we will be relying on third party libraries that implement algorithms (Pandas, SKLearn, and Gensim specifically) as much as possible. The other major potential problem is extracting useful clusters out of Doc2Vec. Doc2Vec is not actually designed for this kind of topic modeling, and our application of it here is experimental in nature. It is our goal to see if Doc2Vec can match or outperform the more conventional LDA, and it may not.

5.2 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.

6. ADDITIONAL AUTHORS

7. REFERENCES

[1] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. Oct 16, 2013. Skip-gram model, Negative Sampling.