CSC 510 Software Engineering: Project 1

Group-P

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 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. [3] 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, 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 "la-

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

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

DOI: 10.475/123_4

tent" 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 unsupervised 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. 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.

2.2.3 Application

-talk about different parameters and reults of gensim / Mikolov

-talk about using all three models : one of each and one that uses both

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. 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

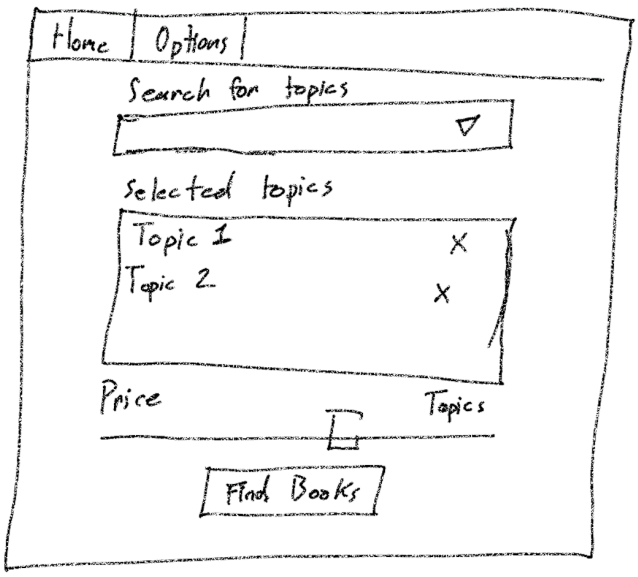
The next step in our problem is to use the topic analysis data outlined in the previous section (3.1) to suggest books based on the topics covered and the cost. This algorithm will be written in Python as well.

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. 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.

Figure 1: Mock up of search setup screen



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

Need to fix this section...

4.1 Evaluating LDA

4.2 Evaluating Doc2Vec

TO DO

4.2.1 Sentiment Prediction Accuracy

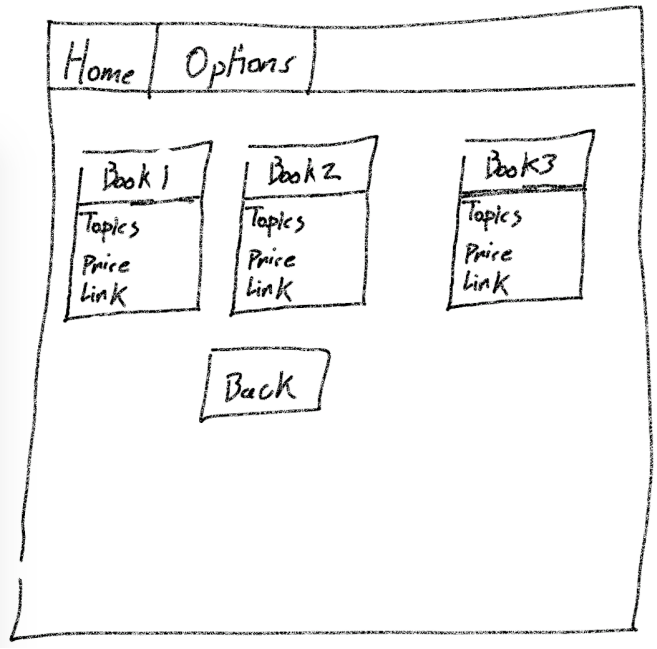
Need to figure out how to test the accuracy of doc2vec when it predicts future text

-gensim has a infer vector function that "infers a vector for given post-bulk trainng document"

-calling model.docvec[doc tag/id] should pull up the vectors that were trained already -I think gensm uses most similar function to find a good trained vector (good because it would predict the given document well) - so if the most similar trained vector is very different than the infered vector, the model is not good?

-I suppose we should be infering documents that were not used in training

Figure 2: Mock up of search results screen



-Error rate would be how far away best vectors are, lower the better?

4.2.2 *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.

-Seperate content such that documents used for testing were not trained on

- 3 folders: two with the same topic, a third folder of different topics that don't include the picked special topic

-Report results (use DM, DBOW, and both together - follow gensim example at https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-IMDB.ipynb

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.

5.3 Timeline

It is our team's goal to parallel path as much of the development of this application as possible. The front-end will be developed in parallel with the learning algorithms, and we plan to have those complete by the 14th of February, pending any problems requiring us to move things around. Once the algorithms are functional, we will spend the next sprint validating them and developing the recommendation engine. Once those tasks are complete, we will begin the final validation phase.

6. ADDITIONAL AUTHORS

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).