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.

PLDI '13 June 16–19, 2013, Seattle, WA, USA
ACM ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

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. [6] 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 [4]. The number of topics can initally be chosen as constant for a given book, or as a function of a chosen Poisson distribution. [4] 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 Gensim's implementation of the Paragraph Vector (PV) introduced by Mikolov and Le in 2014 - an un-

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	-

supervised algorithm that generates vector representations of text inspired from research of vector representations of words via neural networks. [3] [7] PV 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. [7]

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. Given a window of text, the vector representing the target word that is in the middle of the text window is formed by either concatenating or averaging all the word vectors around it, along with the paragraph vector for that document. 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, the word vectors are not saved, therefore it does not retain as much information about the context and semantics of the text. This makes DBOW faster and more resource efficient as it requires less memory. Instead of training word vectors, a single paragraph vector is trained for each document and it predicts words via the probability that the target word is in the document.

2.2.3 Application

Mikolov and Le recommend using a combination of PV-DM and PV-DBOW for consistently accurate results by combining the paragraph vectors in each model. [7] For simplicity, our evaluatoins use DM-M (mean), DM-C (concatenate), and DBOW only; not the combinations of DM and DBOW. Gensim's API makes it easy to change parameters that Mikolov and Gensim both determined to significantly affect results. [1] [7] Since we only have access to average computing resources and limited time to test the models, only a few of these parameters will be cross-validated in evaluations: window size (total window text size / 2 such that the target word is in the middle), feature fector size (the amount of 'neurons' in the word vectors), negative for DBOW (controls negative sampling), the epochs (number of iterations to train over the training corpus) and occasionally the minimum count of a word needed to be trained on throughout all the documents.

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.

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.

$$P_i = a \times num\_topics_i + (b \times max(cost) - cost_i) \qquad (1)$$

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

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 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 reuse, 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

Figure 1: Visualization of layered architecture

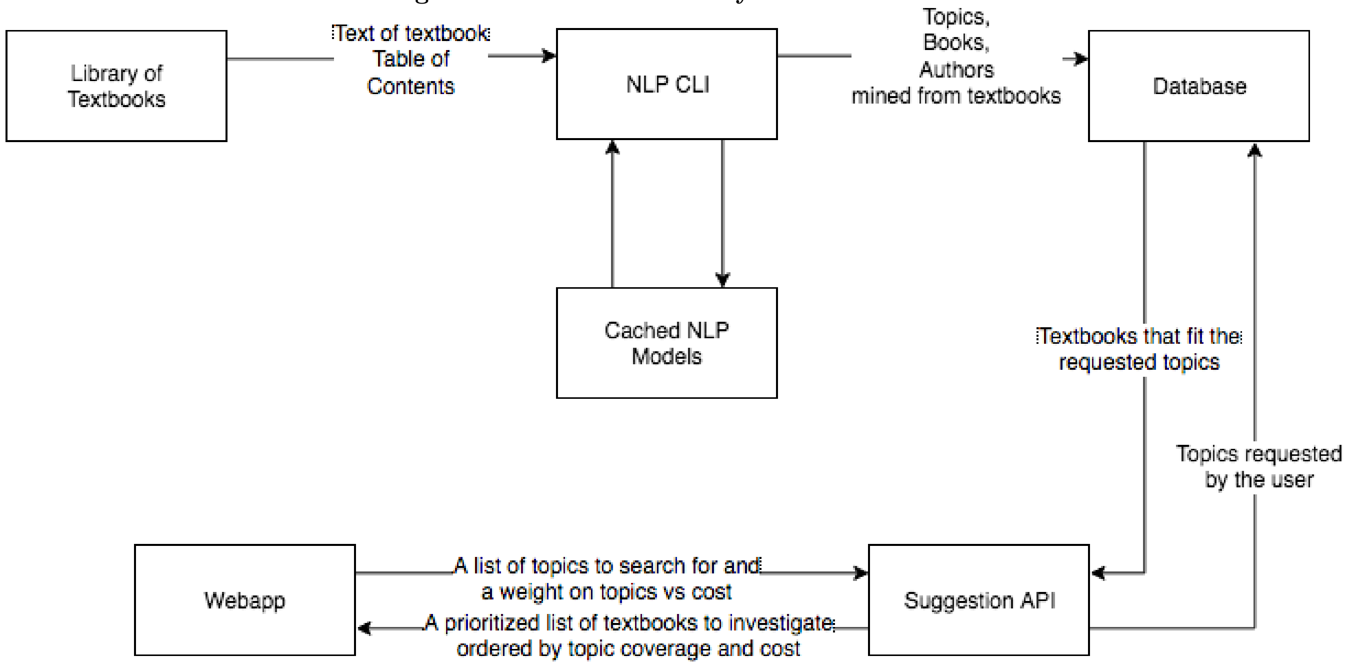
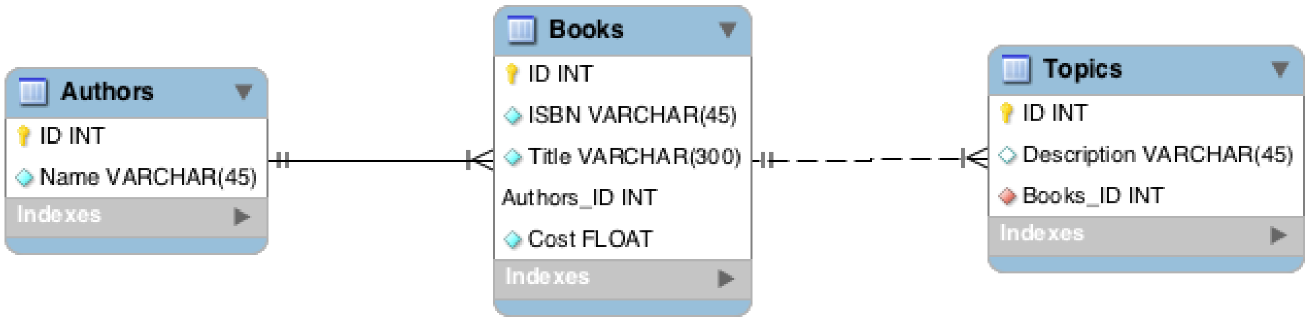

Library of Textbooks
Text of textbook Table of Contents
NLP CLI
Topics, Books, Authors mined from textbooks
Database
Cached NLP Models
Textbooks that fit the requested topics
Topics requested by the user
Webapp
A list of topics to search for and a weight on topics vs cost
A prioritized list of textbooks to investigate ordered by topic coverage and cost
Suggestion API

Figure 2: ERD of Proposed Database



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

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

Table 4: Use Cases

USER	TASKS
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

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.

5.2 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. As we can see from Table 5, the average likelihood of all the test files has been converged as we increased the epoch to 100 passes.

We also used Jaccard and Cosine similarity to independently measure our test results. For each of our test file, the matched topic with the highest likelihood will contribute to the final average measurement. Table 6 summarized the test results. These test results confirmed and agreed with the likelihood measure.

Table 5: Test Result of LDA Model

Epoch	Likelihood	Topic words
1	0.4350	table, sql, like, code, javascript
10	0.5206	sql, table, database, like, java
100	0.5137	table, database, sql, java, javascript

Table 6: Similarity Measure of Test Result

Epoch	Likelihood	Jaccard	Cosine
1	0.4350	0.9709	0.0900
10	0.5206	0.9568	0.1358
100	0.5137	0.9430	0.1731

5.3 Evaluating Doc2Vec

While LDA produces keywords predicted to be the main topics of the document, Doc2Vec produces vectors representing the content. To convert these vectors in potential important topics, first the accuracy of the model must be established to ensure that it can distinguish between one topic and another. Seocondly, there needs to be a sense of precision because if the models are only accurate a fraction of the time then they won't be of much use. The three models tested in this experiment were Distributed-Memory by taking the mean of the vectors (DM-M), Distributed-Memory by concatenating (DM-C), and Distributed Bag of Words

(DBOW). Two tests were used to evalutate the accuracy and precision of the three Doc2Vec models: information retrieval and vector prediction. These two tests were based off the tests Le and Mikolov utilized when analyzing their implementation of the paragraph vector. [7]

The information retrieval test involved taking 1 million popular search engine queries and creating three paragraphs several sentences long from the 10 most popular results for each query. Two of the paragraphs would be from the same source while the third would be from a different source. Their implementation of the paragraph vector was able to correctly idenity the paragraphs from the same source as closer in distance within the vector space with only a 7.42% error.

The sentiment prediction accuracy test corpus was taken from a collection of sentences from Rotten Tomatoes reviews that were labeled by MTurk human workers as positive or negative to determine if the paragraph vector model could dinstinguish between the sentiment of positive or negative. The models preformed well with being able to tell if a sentence was negative or positive, but not with five fine-tuned ratings - if it was very positive vs. slightly positive, etc. The paragraph vector percent error for the fine-tuned ratings was a high 51.3%. Since the stackoverflow data does not necessarily differ by sentiment like the positive or negative movie reviews, this test was adapted to instead to determine the precision of the models on the diverse stackoverflow data that is contributed to by 8.5 million different users. [2]

Doc2Vec was tested on parsed stackoverflow posts. Datasets of different tags were created such that training no post in a set for one tag had any posts that contained other tags that were being tested listed. For exaple, a post tagged 'python' would not also have 'javascript' as a tag. This was done to decrease topic blending and increase the semantic importance of the tags. Tags included python, javascript, database, c#, .net, and java. Decisions on what tags to tests, what parameters to use, etc. where decided in response to test results along the way with the goal of the models being able to distinguish a semantic difference between tags so that the models could determine which posts belonged to which tags, e.g. python post was indeed a python post and a javascript post.

5.3.1 Information Retrieval

To verify that our implementation of Doc2Vec is correctly mapping word vectors to other semantically similar word vectors the trained model was tested on an information retrieval task. 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 into separate datasets containing documents (posts to stackoverflow). If the stackoverflow tags possess a semantic correlation, the Doc2Vec model should report two posts that pertain to the same tag as less distant to each other than a third document that focuses on a different tag.

Dataset Reasoning.

Three datasets were used based off total post count and the results of an initial vector prediction analysis: a set trained on just python, a set trained on both python and javascript, and a set trained on python and database. Python and javascript were chosen because there were more posts for these tags than the others. They were trained together to simulate our goal of using Doc2Vec on multiple topics to filter data into the correct topic like LDA. Database and the just python dataset were added because the vector prediction analysis testing suggested that both the python and database tagged posts were more semantically similar than javascript such that the python and javascript dataset was expected to perform worse. Javascript was not tested by itself due to the hardware and time limitations as the models took between 10 minutes and an entire day to train depending on parameters on a household quad core computer that was available.

The python and javascript training data contained 25,000 randomly selected documents from each tag. Another 15,000 documetns for each tag that were not used for training were selected to be tested. Out of these 15,000 documents, two javascript posts were shown with one python post to determine if the model could accuratlely determine a semantic difference between the two tags allowing the model to be used for topic prediction like LDA. The same was one with the python and database set, except only 2,000 database posts were tested against 4,000 python posts shown two at a time. The rest of the python and database posts were used for training. The number of database (14,000) posts was far less than both python (50,000)and javascript (78,000), but the database dataset scored high on the vector prediction analysis indicationg a possible semantic advantage over python and javascript. The set trained on just python was tested the same way as the python and database trained set.

Measuring Distance in Vector Space.

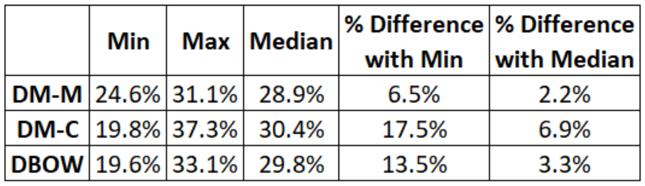
Similar to the fine-tuned Rotten Tomatotes experiment, attemps to implement Gensim's Doc2Vec were met with high percent errors with the parsed stackoverflow data. The error rate was determined by two measurements: the Euclidean distance and cosine similarity. Euclidean distance is the ordinary, straight-line distance between two points. This measurement was used initially with poor results. Training parameters were experimented with in effort to better results, but high error rates wre consistent. The only variable that increased performance was making the number of feature vectors smaller. Upon further research, it was discovered that within a vector space, Euclidean distance is not always an accurate measurement because it focuses on quantity of words and not the semantic meaning behind them. [5] This can generate problems, such as two similar vectors being far apart because of varying document length. For example, a document multiplied several times that is semantically the same would possess a larger magnitude and be calculated as further away. Cosine Similarity, the measurement Gensim uses to compare inferred vectors to the trained vectors within the models, was adopted to solve this problem.

Cosine similarity is a more efficient measurement for comparing semantic relationships as it focuses on the orientation of the vectors by using the cosine angle between them to correlate similarity. [5] It is also independent of document length, thus the magnitude problem is avoided; a document multiplied several times would possess a cosine similarity of 1, which corresponds to being the same. A value of -1 corresponds to being the opposite.

Results.

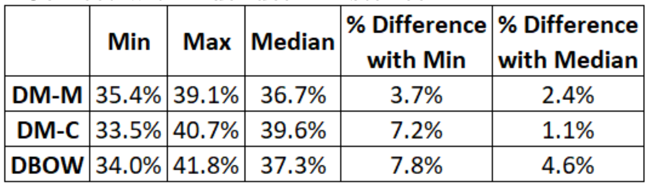
Using cosine similarity to determine if the two posts from

Table 7: Summary of Information Retrieval Results - Correct with Cosine Similarity



	Min	Max	Median	% Difference with Min	% Difference with Median
DM-M	24.6%	31.1%	28.9%	6.5%	2.2%
DM-C	19.8%	37.3%	30.4%	17.5%	6.9%
DBOW	19.6%	33.1%	29.8%	13.5%	3.3%

Table 8: Summary of Information Retrieval Results - Correct with Euclidean Distance



	Min	Max	Median	% Difference with Min	% Difference with Median
DM-M	35.4%	39.1%	36.7%	3.7%	2.4%
DM-C	33.5%	40.7%	39.6%	7.2%	1.1%
DBOW	34.0%	41.8%	37.3%	7.8%	4.6%

the same tag were more similar in vector space than the other was expected to produce results with a lower rate of error, however, the Euclidean distance consistently outperformed. Tables 7 and 8 display the summaries of the results for cosine similarity and Euclidean distance respectively among all the datasets. The percentages were calculated from the fraction of correctly categorized posts such that the two posts from the same tag were deamed closer together in vector space, out of the total amount of times the model was tested. Highlights of the tests are as follows:

- DM-M was the most precise with both measurements among all three models.
- DBOW was the least accurate, but the fastest model to train. With proper optimization, it could be an alternative to datasets that are too large for DM.
- Not every dataset generated the same correlation between parameter changes. Optimization depends on the individual dataset as well as the model type being used.

Overall, the default model of 300 feature vector size, 20 epochs, minimum word count of 1, total window size of 10, and negative set to 10 in DBOW performed well. DBOW and DM-C exhibited the ability to perform more accurately with optimizing of parameters, but not every dataset was impacted in the same way for one parameter change. For example, the highest percentage of correctly categorized posts in DM-C was brought on by increasing the 300 vector size model from 20 epochs to 50 in the just trained on python dataset, but the other two datasets suffered.

5.3.2 Vector Prediction Analysis

The poor results produced by the informaton retrieval testing prompted testing to determine both how accurately and precisely the models were representing the datasets by generating an inferred vector from the training corpus and comparing it to the most similar learned vector from the model. If a document (stackoverflow post) is inferred to be the most similar to its trained self, then the model is an accurate representation of itself. If the model is largely accurate, but demonstrates a wide distribution of cosine similarity values, then the data may not be precise such that the posts differ greatly from one another.

Dataset Reasoning.

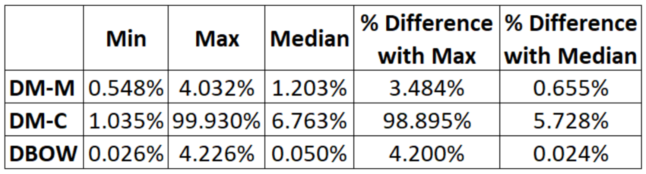
Initially, testing was done on a dataset containing both python and javascript. To avoid taking the time to create special training datasets to be used against multiple combinations of parameters, data was not chosen randomly. All of the posts in a given training corpus were used during testing. The python and javascript dataset was the training dataset created randomly for information retrieval testing that contained 25,000 posts each from python and javascript. The results from the information retrieval testing suggested that there was not a semantic differece between python and javascript posts so python was also tested alone with all parsed python posts (slightly less than 50,000). Since training the Doc2Vec models was resource and time intensive, the database tag dataset was not tested as throughly as the python and javascript and just python datasets. The vector prediction analysis results ofjava, c#, and .net indicated they were not much different than just python so they were not tested further.

Results.

Error rates were calculated as the percent error of inferred posts that were not determined to be the most similar to their trained selves. The best models per each dataset displayed in Table 9 revealed several interesting possibilities regarding the stackoverflow data:

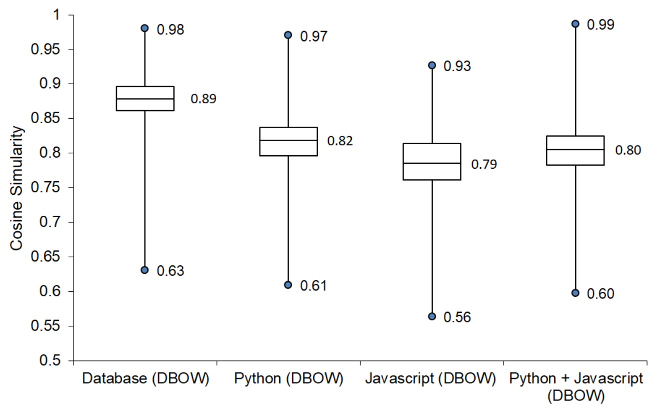
- Again, DM-M was the most precise, but DBOW was close behind.
- Optimized DBOW was the most accurate.
- Setting the total window size to 16 caued DM-C to generate 99+% error with python, javascript, and both python and javascript datasets. Over 97% for the database dataset.
- Increasing epochs from 20 to 50 on a 300 feature vector greatly decreased error rates in DM-C, slightly in DM-M and did not significantly change in DBOW.
- Increasing epochs from 20 to 50 on a 100 feature vector slightly lowered error rates in DM-M and slightly more in DM-C. Only javascript was benefitted in DBOW.
- Raising minimum word count from 1 to 10 greatly reduce error rates in a 300 vector size at 20 epochs for DM-M and DM-C.

Table 9: Summary of Vector Prediction Analysis Percent Errors



	Min	Max	Median	% Difference with Max	% Difference with Median
DM-M	0.548%	4.032%	1.203%	3.484%	0.655%
DM-C	1.035%	99.930%	6.763%	98.895%	5.728%
DBOW	0.026%	4.226%	0.050%	4.200%	0.024%

Figure 3: Most Accurate Vector Prediction Models



Dataset Diversity Affecting Semantic Correlations.

The python and javascript dataset in information retrieval testing consistently performed worse than the python and python and database datasets while using cosine similarity. The complete collection of parsed stackoverflow posts pertaining to each individual tag was tested via vector prediction in addition to the training python and javascript corpus to determine how similar the content was. Javascript was expected to reveal more post diversity than the python and database tags to explain the higher information retrieval error rates. Figure 3 is a box plot visualization of the most accurate model per the datasets tested. Javascript did indeed have the lowest minimum, median, and maximum cosine similarity values suggesting greater post vocabulary and semantic diversity. Surprisingly, the python and javascript training corpus performed about the same as the complete collection of python posts. The fact that javascript was the largest dataset with over 78,000 different posts and the python and javascript training corpus only contained 25,000 posts from each tag could be a contributing factor to this. It is likely that the greater the amount of posts, the greater the amount of different users with different language semantics resulting in the Doc2Vec model struggling to find semantic correlations.

Conclusions.

The distribution of cosine similarity values in vector prediction analysis was thought to affect accuracy in information retrieval results, but that was not always the case. Generally, the models with smaller ranges, less distributed quartiles, and higher means and medians performed better than models that were in the opposite end of the spectrum for boh vector prediction and information retrieval. However, The models with the best vector prediction distributions were not always the best at the information retrieval task.

Information retrieval of the just python dataset generally produced lower error rates than the others, therefore, training a Doc2Vec model on one topic at a time and filtering for that topic might be better for predicting topics like LDA. However, the error rates in information retrieval were too high to recommend using Doc2Vec over a technique like LDA to predict text content topics. The Doc2Vec DM models depend on semantic relationships to distinguish between input. A high error rate suggests that there may be no actual semantic relationship between the stackoverflow data. Part of the problem might be the topics themselves. Python and javascript, for example, are both computer programming languages. Although they have different uses and purposes, the concepts surrounding them are more similar than say a programming language and more distance subject such as biology. The stackoverflow data could also be parsed more effectively. The data consisted of the 'posts' from stackoverflow. These posts did not contain the post titles or the comments. Both the titles and comment so often contain the most helpful information relevant to the topic at hand. Alternatively, content with more specific information that is better suited for parsing could also be adopted instead, such as typo-less information or cleaned up code from scholarly resources to allow for more efficient tokenization of relevant words or symbols like parathensis and brackets which are more prevalent in some programming languages than others. It is possible, however, that because the data from stackoverflow comes from a substantial amount of users, a semantic relationship may simply not exist.

5.4 Evaluating Application

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 10 is the breakdown of the survey questions.

Table 10: 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

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 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. But since this is a relatively small project, the issues have been solved one by one cases.

6.2 LDA

As expected, purely based on the accuracy (topics likelihood) 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.

Noticed that our independent Jaccard and Cosine measurements were not as good as topics likelihood, it could be the way that we calculated the measurements which only used certained labeled words, these words could be less similar to those words in the mined topics.

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, for instance, by refactoring the algorithms to run on distrbuted system, but it is something that seems to be inherent in how to train the algorithm.

6.3 Doc2Vec

Doc2Vec has some advantages over LDA that make it a desirable alternative. It can compare documents quickly if the model is already trained, eliminating LDA's wait time. In addition, there is no need to specify the number of topics to mine. However, Doc2Vec was unsuccessful at accurately generating topics as it could not distinguish between the different stackoverflow topics within an acceptable precision. If Doc2Vec has been able to distiguish between different topics, it could have been trained on pre-labeled documents such that an untrained, un-labeled document could have been suggested the labels from the most similar labeled document in the model. A better use for Doc2Vec might be as an extra feature of the Web Application to suggest text that is semantically similar to given text, i.e. finding a book author that writes in a similar syntax to the given author or text.

6.4 Future enhancement

The trained learning algorithms have not peroformed as well as expected. This might be due to a few different potential causes. First off, in Stack Overflow posts, topics are not always explicitly mentioned, which would prevent LDA from properly parsing topics. For example, many posts tagged C++ do not actually mentione C++ specifically. A tweak to the input data set might help improve algorithm accuracy significantly. One approach the team considered but was unable to finish in time was manually parsing Stack Overflow posts for posts that did include their tags in the body and running the example on those.

The next obvious goal would be to start building the rest of the system. Once the problem of topic modeling is more appropriately solved, the suggestion algorithm discussed in 3.1. Once Equation 1 is implemented in a REST API, the actual webapp could easily be built quickly.

7. CHITS

- FNL
- VVR
- AFJ
- GVO
- YLV
- RBC
- VZV
- WLB
- XYU
- BLM

8. REFERENCES

[1] Gensim doc2vec tutorial on the imdb sentiment dataset, Dec 18, 2017.

[2] All sites - stackexchange<br>, 2018. stackoverflow 8.5 million users.

[3] models.doc2vec âĂŞ deep learning with paragraph2vec, March 1, 2018. API.

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

[5] A. Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

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

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