Recommendation System Developer

Eshita Nandini[1], Janie Neal[2], Taylor Olson[3], and Christiana Prater-Lee[4]

[1]*University of California, Merced*

[2]*Pomona College*

[3]*University of Northern Iowa*

[4]*Vassar College*

Abstract

We construct a user-friendly recommendation system for Harvard University using natural language processing (NLP) techniques that, when given a course, selects similar courses using course syllabus data obtained through Harvard's Vice Provost for Advances in Learning (VPAL). Our Python-based Dash system employs statistical models, such as Term Frequency Inverse Document Frequency (TF-IDF) and Latent Semantic Analysis (LSA), in an ensemble method to rank similar courses using cosine similarity. The system allows the user to upload textual data, select methods for cleaning and statistically modeling the data, and then outputs a ranked list of similar documents. While the system will be used by the Harvard community, it may also be adapted to other universities, as well as generalized to apply to different types of queries.

I. Background

Harvard University offers more than eight thousand courses in over a hundred departments across its fifteen colleges. This allows for a variety of classes, but may be challenging for students and faculty to navigate the current catalog. For example, if a class is filled, students may have trouble finding an alternate course with similar content. In addition, students may wish to avoid classes which have overlapping content, but have a hard time comparing across colleges and departments. Faculty members may also want to know about courses similar to the ones they teach. In the past, Harvard faculty and staff have attempted to compare courses by hand, but this manual method has proved to be inaccurate and tedious. A more efficient approach is to create a recommendation system using natural language processing (NLP). Such systems are used in a variety of applications (Netflix, Amazon, etc.) to suggest similar objects based upon input data.

II. Methods

When "given" a course, the system selects similar courses using course syllabi data. More specifically, the generalized interface allows the user to upload any data set, select how they want to clean given data, statistically model data, and then obtain a ranked list of similar documents based off of the inputted one. NLP is applied to datasets obtained through Harvard's Vice Provost for Advances in Learning (VPAL). We work with 6707 course syllabi from Harvard's College of Arts and Sciences and the Graduate School of Arts and Sciences (GSAS).

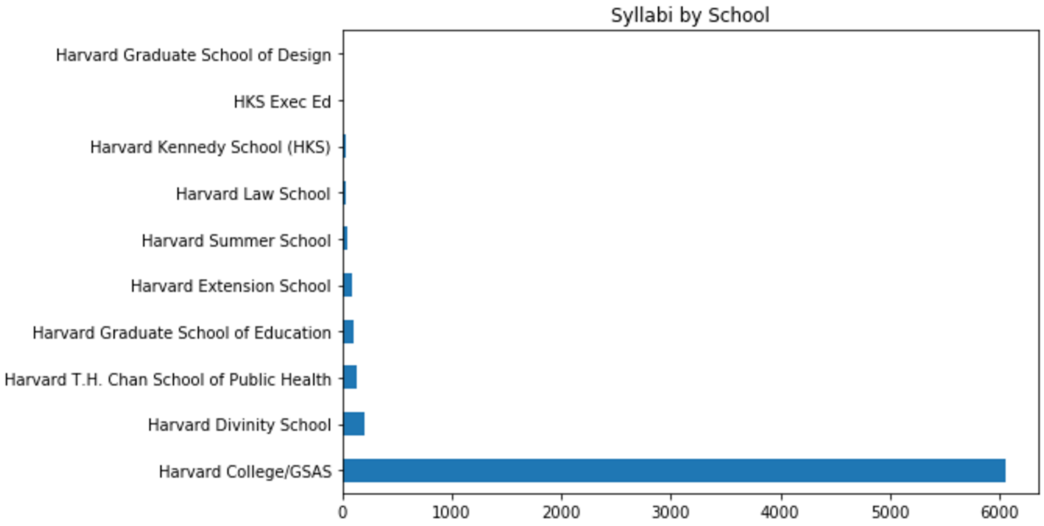


Figure 1: *Distribution of syllabi data across different schools; most of the syllabi were obtained from Harvard College of Arts and Sciences and GSAS.*

We must first convert textual data to a usable form as the scraped data is in an inconsistent and ineffective format (unnecessary words, code, etc). This means tokenizing, removing stop words (frequently occurring words such as "the", "and", "is", etc.), capital letters, punctuation, HTML tags, e-mails, numbers, words that occur once, non-English words, and ASCII characters over 127 [1]. We create functions to either stem or lemmatize the data, but leave these actions as the user's choice in our interface as stemming and lemmatizing are mutually exclusive actions. Stemming truncates the word's root, while lemmatization converts words to their foundational dictionary basis. Each has its benefits and weaknesses. For example, stemming would convert "meeting" to "meet", while lemmatization would be able to tell from context whether "meeting" is a noun or a verb and then convert to the right root, "meeting" or "meet". Lemmatization is therefore more accurate than stemming, but is also computationally more expensive. We also want to void our data of words that occur too frequently or too rarely, as they will not give us much insight into the text and may deter how the models work with the text. For this reason, we have inserted a word-frequency slider object in our interface that will allow the user to pick words to chop off; more explanation is giving in the Implementation and Interface section.

After cleaning the data, we run the data through statistical models. We use a few key `Python` packages to build the models: `Pandas`, `Gensim`, `Sklearn`, and `spaCy`. Using the `Pandas` package, we create a term-frequency (TF) matrix from the course syllabi. The term-frequency matrix is defined as a $D \times W$ matrix that contains the frequency of unique words W in a collection of documents D, in this case, the syllabi, where each row corresponds to a document d and each entry in the column corresponds to a word w in that document. We create prototypes for four common statistical models that build off of the TF matrix and that represent a range of statistical comparison methods: Term Frequency Inverse Document Frequency (TF-IDF), document similarity with `spaCy`, Latent Semantic Allocation (LSA), and Latent Dirichlet Allocation (LDA). Each of these statistical models has a similar structure: input a certain term-frequency matrix, map the documents to a vector space by creating a vector of a document's words (either all given words or keywords) based on the word's associated weight, and perform cosine similarity to compare documents. The formula for cosine similarity is:

$$\cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

where x represents one document and y represents the other document to which x is being compared. The dot product of two vectors represent how close they are, and therefore, how similar. When given a course x, the interface will run the model and compare x to all other courses. Each will receive a cosine similarity score; the scores closest to one represent the most similar courses. The interface returns the most similar courses (per the number selected by the user).

We then conduct some analysis. For example, we can compare similar courses by department to see which departments contain the courses most similar to the inputted course.

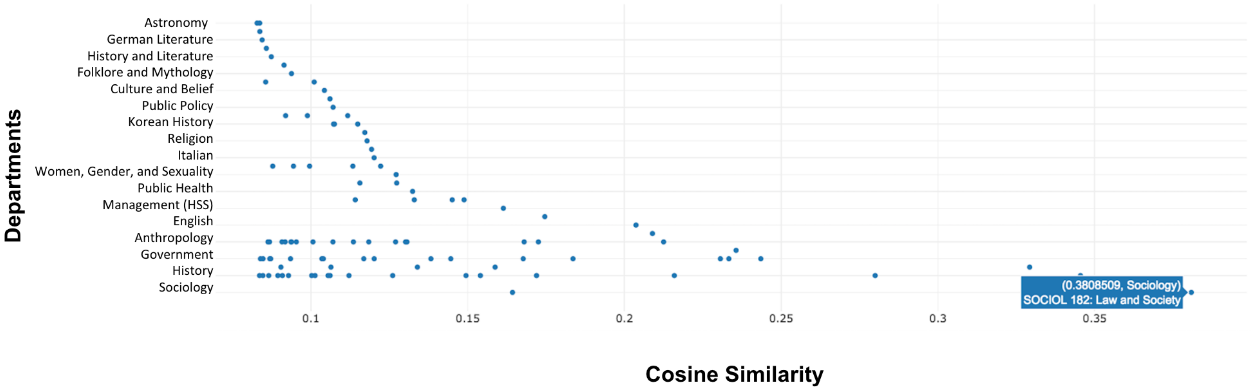


Figure 2: *Similar courses to "ANTHRO 2674: Legal Anthropology and Modern Governance" by department–"SOCIOL 182: Law and Society" has the highest cosine similarity score, therefore the most similar.*

III. Statistical Models

A total of four models are used in the interface. Below is a discussion of the models along with their strengths and weaknesses:

Term Frequency Inverse Document Frequency (TF-IDF): TF-IDF determines how important a word is to a given document based on its frequency in each individual document and the overall corpus [8]. The TF-IDF matrix incorporates the $D \times W$ TF matrix, as defined earlier, and a $W \times W$ inverse document frequency (IDF) matrix. The purpose of the IDF matrix is to dampen the TF matrix and avoid over-representing frequent words.

The $W \times W$ IDF matrix is constructed by summing down the columns of a $D \times W$ binary document frequency matrix where a term is given a score of 1 if it appears in the document and 0 if it does not appear in the document. Each entry is further dampened by taking the natural logarithm and adding 1. The TF and IDF matrix are multiplied together, resulting in the $D \times W$ TF-IDF matrix.

Each entry of the TF-IDF matrix may also be calculated individually. We may calculate the TF and IDF scores for each word in each document. The TF score is calculated by the following formula:

$$\text{TF}(w,d) = \text{ count of } w \text{ in } d$$

where $w \in W$ and $d \in D$.

As mentioned, TF alone over-emphasizes words that occur in many documents. To compensate, the inverse document frequency score is calculated to diminish the weight of words that occur frequently across the corpus, D:

$$\text{IDF}(w) = 1 + \log(\frac{D}{\sum d \text{ that contain } w})$$

Below is the formula for the TF-IDF score for w in d:

$$\text{Value} = \text{ TF } * \text{ IDF}$$

In summary, these calculations result in a $D \times W$ matrix where each row is a vector representation of a document in the corpus. The vectors correspond to values scoring each word, $w \in W$, according their relative importance to the document and in the context of the corpus. Each document vector can be compared with the other documents' vectors using cosine similarity. The documents with the highest cosine similarity score are the most similar.

`Sklearn` provides a package for TF-IDF that calculates the weights for TF-IDF matrix. These individual weights are compared across documents using cosine similarity. While TF-IDF is computationally efficient and serves as the basis for many other statistical models, it does not account for synonyms. For example, the strings "their football captain played well tonight" and "the basketball player had a good game this afternoon" would receive a TF-IDF score of zero although the strings clearly have correlation.

Latent Semantic Analysis (LSA): LSA starts with any term document matrix and does dimension reduction using singular value decomposition (SVD). SVD asserts any $m \times n$ matrix, A, can be factored into 3 smaller matrices. In our project, to increase accuracy and by recommendation of the `Gensim` developers, we use the TF-IDF matrix as the input matrix. LSA differs from TF-IDF because, instead of comparing individual words, LSA strips away all but the most important combinations of words or the "topics" of the document set [8].This is especially advantageous for comparing two documents which humans could identify as similar but may not share any exact words. If these documents share many terms with a third document, they will end up being similar in the projected vector space. For example, the problem presented in TF-IDF with the short documents, "their football captain played well tonight" and "the basketball player had a good game this afternoon" would be solved if another document existed in the corpus tying them together, like "Each player, including the captain, is important in team sports like football and basketball". This is able to solve the problem that TF-IDF alone cannot tackle.

Let A be the TF-IDF matrix created from D documents containing W unique terms with rank r. This would mean:

$$A = T\Sigma F^T$$

where T represents an $W \times r$ term-concept matrix, Σ represents a diagonal $r \times r$ matrix with non-negative decreasing singular values on the diagonal, and F^T is an $D \times r$ concept-document vector matrix. Σ describes the relative strengths of the concepts, T describes the relationship between terms and concepts, and F^T describes the relationship between concepts and documents. After finding this factorization, we apply SVD truncation, by first choosing a number of "topics", t, and keeping only the highest entries in Σ. We truncate T and F^T, keeping only the values corresponding to the retained values of Σ. Mathematically, the product of these 3 reduced vectors is the closest approximation of A in a k dimensional space. In application, we have reduced the original matrix to only its most important concepts. The resulting matrix is a topic-document matrix where each row represents what proportion of a document's content is composed of each generated topic. If the reader would like a more in depth explanation and several illustrative examples, they should consider reading the introduction to LSA given at `https://matpalm.com/lsa_via_svd/intro.html`.

`Gensim` provides a package to run the LSA algorithm. This model is computationally more expensive than TF-IDF to implement, but more accurate as well since synonyms may be recognized as the same topic. The expense comes in when adding in more documents; one has to recalculate the truncated matrix each time new words are added.

Latent Dirichlet Allocation: LDA intends to improve upon LSA by allowing for mixed membership of topics [3]. That is, each document d may be assigned multiple topics t, words w that frequently occur together. This allows for a fewer number of topics to be vectorized, further reducing the TF-IDF matrix's dimensions–a more efficient formula. [2].

LDA starts with the TF-IDF matrix as well, then infers topics from the words of a given document based on the prior Dirichlet distribution (a multinomial distribution over possible parameter values of words in each topic for a multinomial distribution of topics in a document). The overarching formula for a document's LDA topic distribution is as follows:

$$Pr(t, \theta_{1:D}, z_{1:D} | w_{1:D}) = \frac{Pr(t, \theta_{1:D}, z_{1:D}, w_{1:D})}{Pr(w_{1:D})}$$

where Pr equals probability, t equals topics, θ equals topic proportions for topics in a document, z equals topic assignments for words in documents, w equals words, and D equals documents. However, this formula is impossible to compute directly if one does not know the probability of a word's appearance in a document, and one must either determine this probability by trial and error or sampling methods.

`Gensim` provides a package that runs the LDA formula using Gibb's sampling–a word from the TF-IDF matrix defined above is randomly assigned a topic based on a Dirichlet distribution for the given number of topics, then these topic distributions are improved through an iterative process. Each word is assigned a new topic t by:

$$Pr(t \text{ generated } w) = \text{proportion}(t|d) * \text{proportion}(w|t)$$

until a coherent topic mixture is reached. The resulting $D \times t$ matrix is a topic-document matrix where each row represents what proportion of a document's content is composed of each generated topic.Each document is then represented as a percentage of different topics which sum to one. The LDA matrix is essentially a further reduced rank version of the matrix from LSA.

LDA is more efficient than LSA. Instead of factoring in more words each time a new course is added, we can just incorporate these new words into the existing topics. However, LDA does not alleviate other issues with LSA, such as determining the optimum number of topics. As discussed, we must determine the optimum number of topics for LSA and LDA. This optimum number of topics will not be the same for LSA and LDA and it will not be the same across data sets. Both LDA and LSA are topic models and while in theory LDA increases accuracy by allowing mixed membership of topics (and therefore requiring less topics), it over-fits the topics on documents with small bodies of text, lending too much emphasis to unnecessary words. Bergamaschi and Sorrentino found that compared to user preferences, LSA is twice as precise as LDA when recommending similar movies based on plot [2]. Their method is extremely similar to ours (they employ movie plot descriptions to compare movies) and thus their findings validate our distrust of LDA.

Note: It is difficult to determine the optimum number of topics t in topic modeling. The method for doing so is highly controversial and the number of topics differs across models and documents

[7]. LDA requires less topics than LSA due to the mixed membership of topics. In addition, more documents require more topics, but at a decreasing non-linear rate as some new document concepts may be incorporated into existing topics. We try many methods (Silhouette Scores, Log Perplexity, and multiple Coherence Measures) and settle on the Cover Coefficient (CC) method of implementation for LSA for its convenience, accuracy and ability to be generalized across a range of documents. We use the CC formula $t = \frac{DW}{p}$, where D equals number of documents, W equals number of terms across documents, and p equals number of non-zero entries in the term document matrix [4]. After experimenting with a number of topics, we decide to use $t = \frac{DW}{3p}$ for LDA. The CC method ignores word distribution and context of words, and therefore has low computational cost. Although rudimentary, the CC method is more accurate than other proposed methods on our data.

Document Similarity with spaCy: While topic models efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating an unsatisfactory vector space structure. `spaCy`, by default, uses the Global Vectors (GloVe) algorithm to produce word embeddings. GloVe provides word representations that outperform other models on word analogy, word similarity, and NER tasks. What GloVe essentially does is create a matrix of word-word co-occurrences–how many times a word occurs in the context of another word [5]. This matrix is mapped to a vector space; GloVe uses a least squares regression model to minimize the dot product between word vectors. `spaCy` trains GloVe on Common Crawl corpus by default, which offers free web page data. Using these word embeddings, we are able to map documents from our corpus. `spaCy` averages the term vectors for each documents to represent it, and then cosine similarity is done to come up with similar document suggestions.

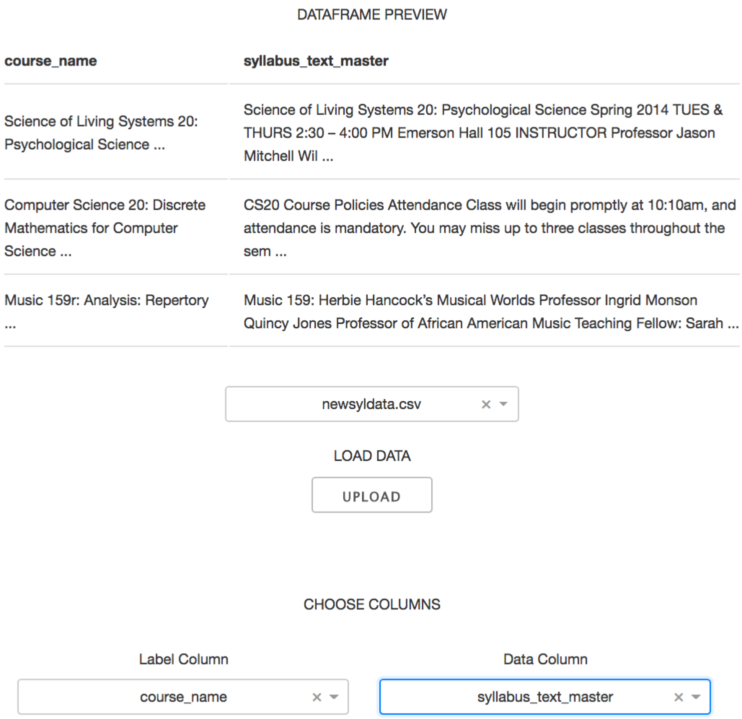
Ensemble Methods: We employ an ensemble based voting system to our recommendation system. Each model (TF-IDF, spaCy, LDA, LSA) receives equal weight in selecting similar documents. The user picks the number of similar documents they desire for the inputted document, and each model generates a ranked list of similar documents. The most similar document is given the highest score and the least similar document is given a score of 1. For example, if the user wants the top ten similar documents, the most similar is given a score of 10 and the tenth most similar document is given a score of 1. Then each similar document is summed across all models (for our example, the highest theoretical score is 40) and a new list is obtained where the highest score is the most similar document.

IV. Implementation and Interface

Using Plotly's newly developed Dash framework, we designed a locally hosted web interface that implements the previous steps (upload, clean, model, test, and analyze data) [6]. The Dash framework allows interactive Python web applications to be built with little to no setup. The front-end of the application was developed using HTML and Dash components, while the back-end was built using Python and various packages. Rather than moving between multiple HTML pages, our application works as a single page. As the user navigates through the various steps, the content of the page is simply refreshed with the necessary front-end components. Dash makes this easy to do with *callbacks*, event handlers that allow for the capture of user actions such as clicking, moving sliders, and loading data. When an event is fired, the Python methods following the event return the necessary HTML components and/or data needed to update the content of the application.

Our interface contains the following steps:

- **Load Data**: The user uploads a csv data file of documents that contain textual data with the minimum columns: name and description. The user then selects two columns (name and description) to be analyzed. For the course syllabi data, these columns are course name and syllabi body. The interface creates a data frame of these columns to be cleaned and allows the user to preview these columns.



DATAFRAME PREVIEW

course_name	syllabus_text_master
Science of Living Systems 20: Psychological Science ...	Science of Living Systems 20: Psychological Science Spring 2014 TUES & THURS 2:30 – 4:00 PM Emerson Hall 105 INSTRUCTOR Professor Jason Mitchell Wil ...
Computer Science 20: Discrete Mathematics for Computer Science ...	CS20 Course Policies Attendance Class will begin promptly at 10:10am, and attendance is mandatory. You may miss up to three classes throughout the sem ...
Music 159r: Analysis: Repertory ...	Music 159: Herbie Hancock's Musical Worlds Professor Ingrid Monson Quincy Jones Professor of African American Music Teaching Fellow: Sarah ...

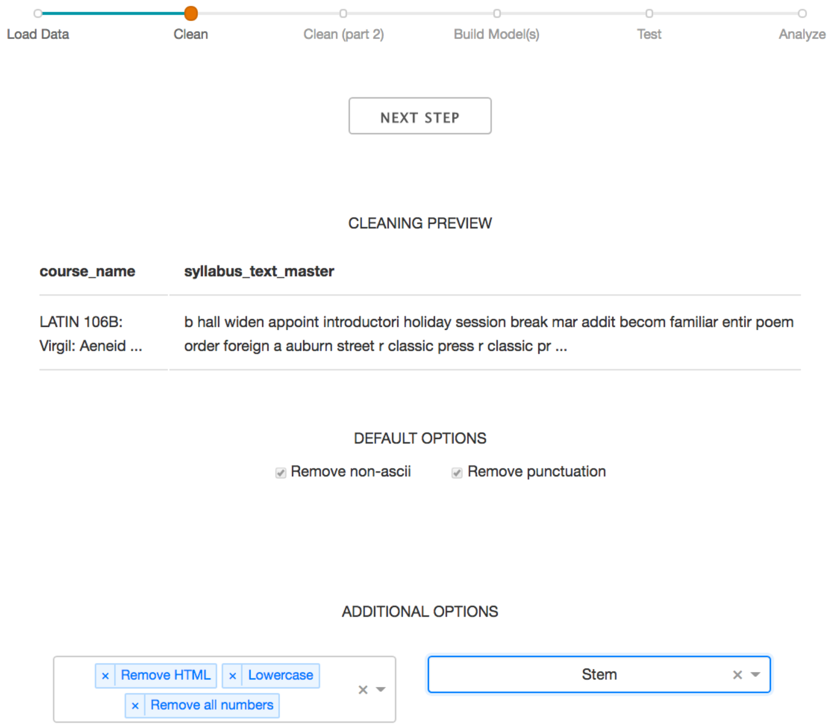
newsyldata.csv

Label Column: course_name

Data Column: syllabus_text_master

Figure 3: *Load Data Page*

- **Clean**: Non-ASCII characters and punctuation are automatically removed from the data frame; these are required in order for the model to work. The user can select additional cleaning options (remove HTML, e-mail, stop words, non-English words, and/or numbers and lowercase the data) based on the nature of the data they are using. The user may also choose to either stem or lemmatize the data.



CLEANING PREVIEW

course_name	syllabus_text_master
LATIN 106B: Virgil: Aeneid ...	b hall widen appoint introductori holiday session break mar addit becom familiar entir poem order foreign a auburn street r classic press r classic pr ...

DEFAULT OPTIONS

☑ Remove non-ascii ☑ Remove punctuation

ADDITIONAL OPTIONS

× Remove HTML × Lowercase × Remove all numbers

Stem

Figure 4: *The first cleaning section. The user is presented with a few default options, plus other options which they can choose or leave out from the cleaning process.*

- **Clean (part 2)**: The user may employ the cutoff slider to to remove high and low frequency words. The y-axis contains indexes for each term in the corpus, and the x-axis contains the documents that these terms occur in. After removing these words, the user may download the data. One may use our interface as solely a cleaning tool rather than a recommendation system.

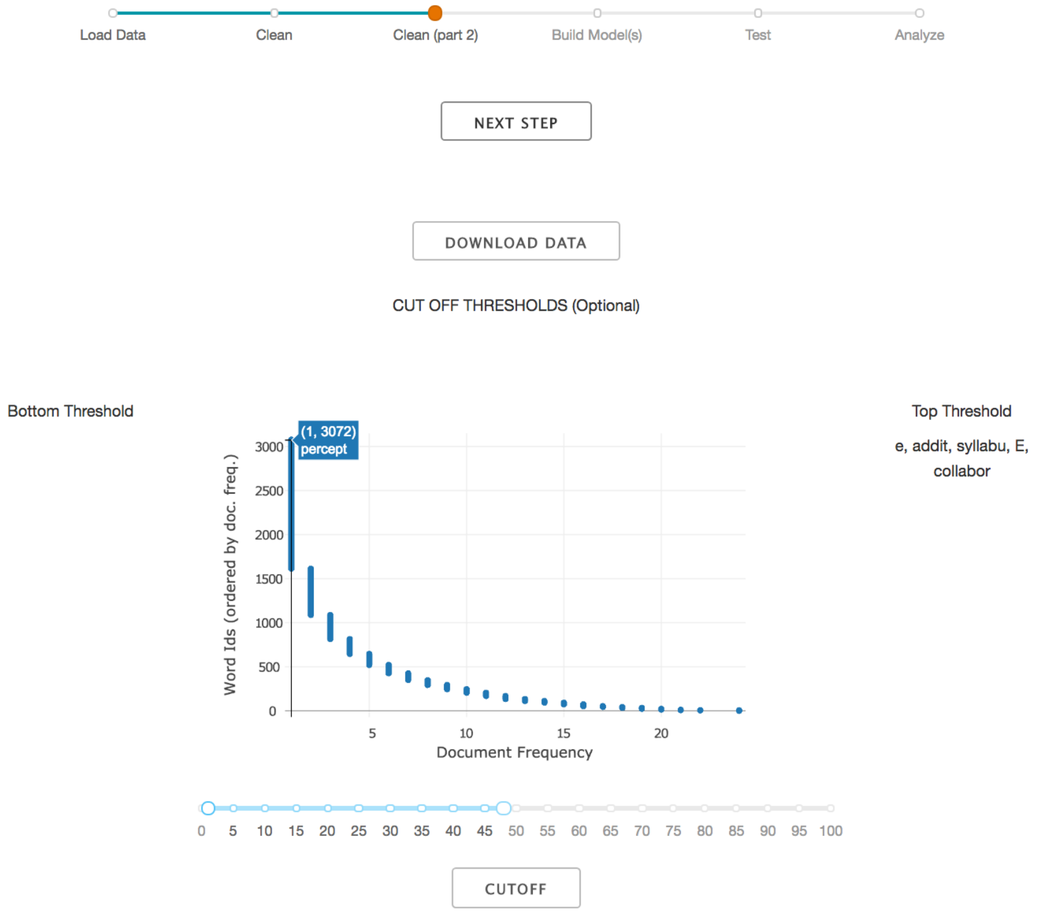
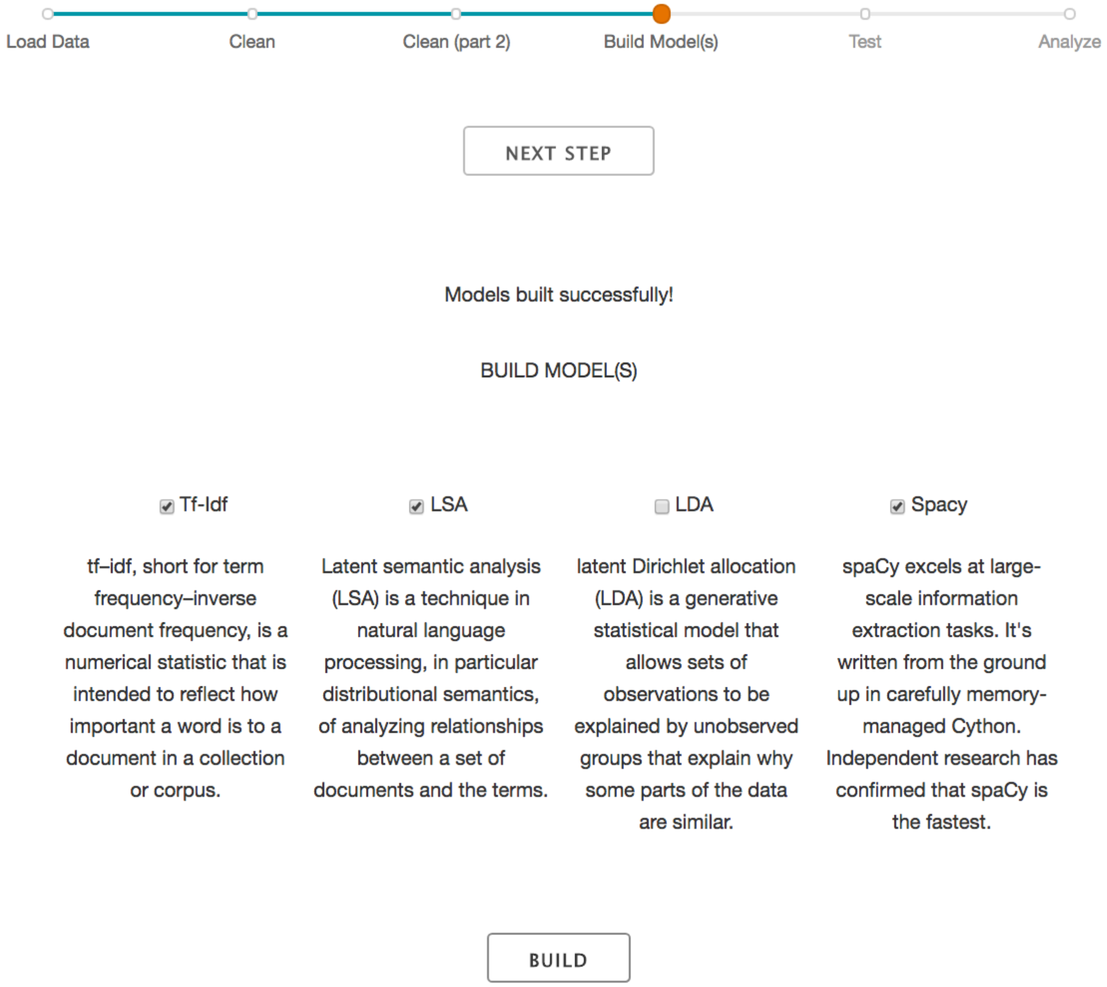


Figure 5: *Clean(part 2) section; in this example, the user has not picked a bottom threshold percentage, which means words in the bottom frequency are not truncated. The user chose to omit words that occur in about 50% or more documents. The right side of the object shows what words were removed from the top threshold.*

- **Build Model(s)**: The user selects which statistical models they want to use for comparison (TF-IDF, spaCy, LSA, LDA). The models are then built. If multiple models are selected, the interface employs the aforementioned ensemble method to output recommendations.



Models built successfully!

BUILD MODEL(S)

☑ Tf-Idf

tf–idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

☑ LSA

Latent semantic analysis (LSA) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms.

☐ LDA

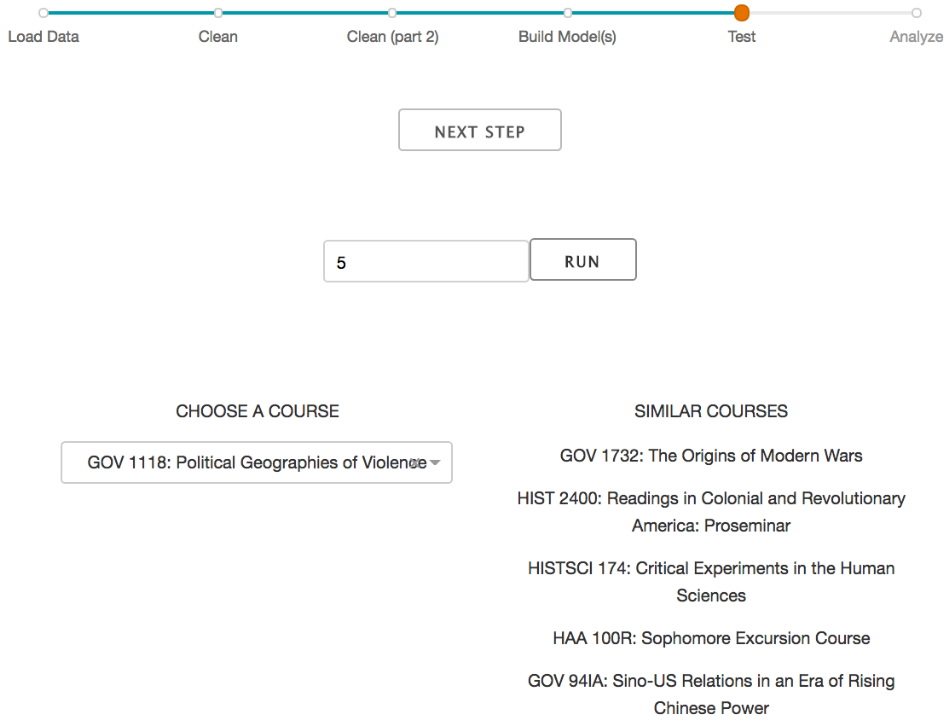
latent Dirichlet allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar.

☑ Spacy

spaCy excels at large-scale information extraction tasks. It's written from the ground up in carefully memory-managed Cython. Independent research has confirmed that spaCy is the fastest.

Figure 6: *Build Model(s) page. The user is able to pick as many models as they would like. Each model comes with a short description of how it behaves with the data. In this instance, the user picked all but the LDA model; the interface will do an ensemble voting to generate recommendations.*

- **Test**: The user selects a document and the number of similar documents they desire. The interface generates a ranked list of similar documents. In Figure 7, the user has selected the course "GOV 1118: Political Geographies of Violence". The interface returns "GOV 1732: The Origins of Modern War" as most similar.



5

CHOOSE A COURSE

GOV 1118: Political Geographies of Violence

SIMILAR COURSES

GOV 1732: The Origins of Modern Wars

HIST 2400: Readings in Colonial and Revolutionary America: Proseminar

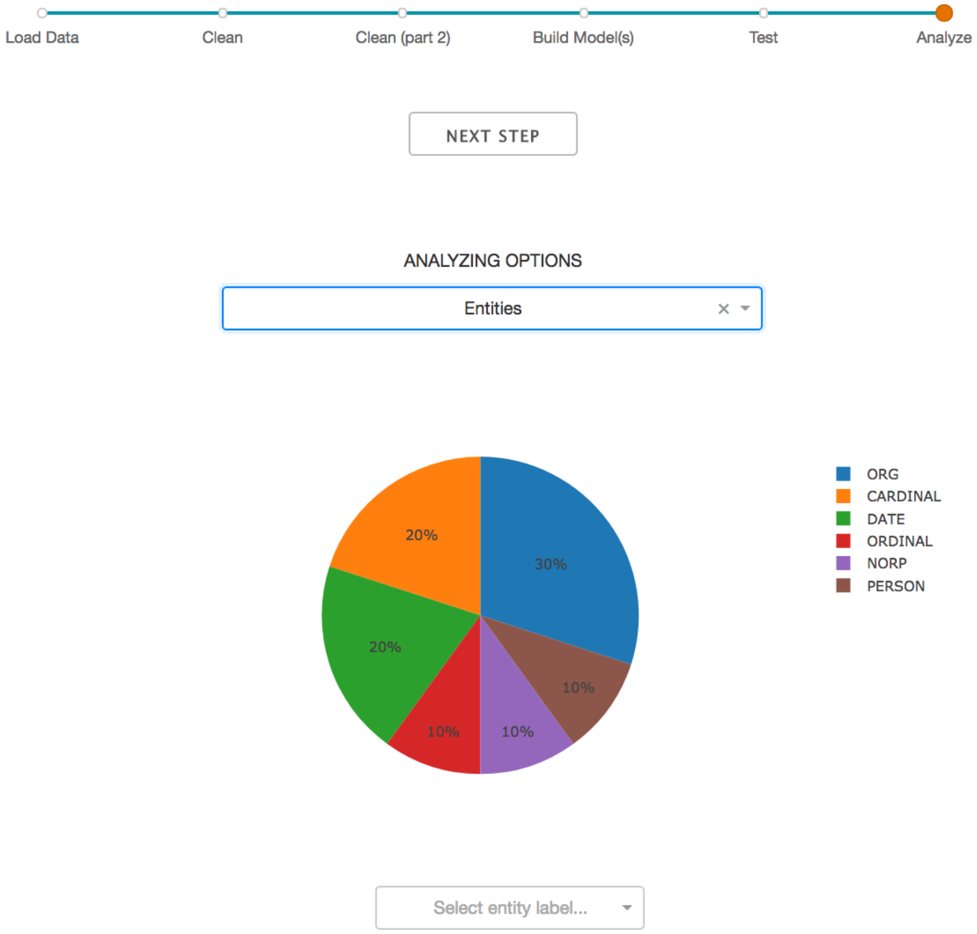
HISTSCI 174: Critical Experiments in the Human Sciences

HAA 100R: Sophomore Excursion Course

GOV 94IA: Sino-US Relations in an Era of Rising Chinese Power

Figure 7: *Testing the model and viewing the recommendations outputted.*

- **Analyze**: The user can choose to further analyze the contents of their data. If the user selects "Entities", they can view the named entities (books, people, languages, etc) within their data set. The pie chart represents entities that appear in the corpus. The percentages represent the proportion of that entity to all the entities in the documents. If the user clicks "Select An Entity", they may view those entities in the data set. For example, if the user clicks "Work of Art", they will obtain the most common books in the data set. Cardinal and Ordinal refer to numbers.



ANALYZING OPTIONS

Entities

Figure 8: *Pie-chart of entities appearing in user inputted data set.*

In the analyze section, the viewer may also select "Visualize Topics" to view the topics that are contained their data set. Each dot represents a different document and each color represents a different topic; the three words that represent each topic are listed on the right. The color of the text matches the color of the cluster (topic). Closer dots mean that those documents are more similar.

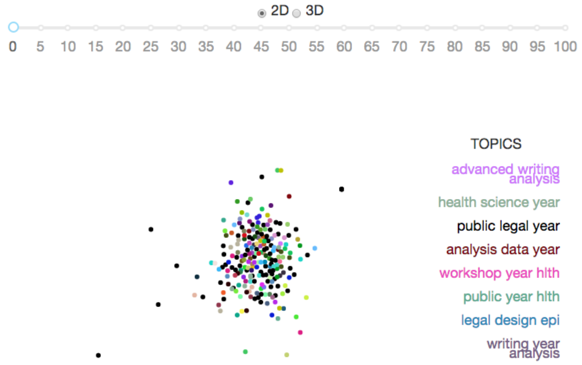


Figure 9: *Topic cluster visualization, with topic names on the right list.*

V. Discussion

i. Generalized Interface

We construct our recommendation system to be generalizable; that is, it can load any textual dataset and, based on a text column, output similar documents. For example, when "given" a database of Wikipedia book summaries, our recommendation system will suggest similar books to the inputted book title. We have applied our system to political social media (Trump tweets obtained from *www.reddit.com*) to suggest similar tweets by date tweeted. We also brought our system to the law clinics at Harvard for suggestions and have implemented our system on court data (obtained from *www.mass.gov*). The user inputs a criminal offense and based on the charges' descriptions, receives similar offenses. A lawyer may use our recommendation system to find an alternative, more appropriate sentence that has a lesser charge. Our system has many useful possibilities, from resource management to career suggestions.

ii. Future Work

Our cleaning and modeling processes will be refined to make the system more accurate. Ideally, our system would preserve entities (book titles, events, etc). However, this is computationally expensive for each entity has to be preserved as a token during cleaning then added back to the data after other words are tokenized and cleaned. Therefore, named entity extraction is not efficient to implement in our model prototype.

We will add additional analysis features into our interface. For example, we want to filter our recommendation system by department, so the user may search for similar courses within one's department.

Our interface will be integrated into Harvard's existing server to increase accessibility and efficiency. Harvard currently has a website where one may input a subject and obtain a list of Harvard resources about that subject. VPAL plans to create a similar website for our recommendation system accessible to Harvard students and faculty.

Acknowledgements

Sponsors: Dustin Tingley (VPAL Faculty Director) and Daniel Seaton (VPAL Senior Research Scientist)

Mentor: Margo Levine (Harvard Associate Director of Undergraduate Studies in Applied Mathematics)

Funding: NSF REU Site: Team Research in Computational and Applied Mathematics, NSF DMS-1460870

References

[1] S. Bansal, *Beginners guide to topic modeling in python*. obtained from Analytics Vidhya, 2016.

[2] S. Bergamaschi, L. Po, and S. Sorrentino, *Comparing lda and lsa topic models for content-based movie recommendation systems*, December 2015.

[3] D. Blei, *Probabilistic topic models*, Communications of the ACM, 55 (2012), pp. 77–84.

[4] F. Can and E. Ozkarahan, *Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases*, ACM Transactions on Database Systems, 15 (1990).

[5] C. Manning, J. Pennington, and R. Socher, *Glove: Global vectors for word representation*. Stanford University Computer Science Department, 2014.

[6] Plotly, *Dash by plotly*. user guide and github code used as references, 2017.

[7] J. Tang, Z. Meng, X. Nguyen, X. Mei, and M. Zhang, *Understanding the limiting factors of topic modeling via posterior contraction analysis*, Proceedings of the 31st International Conference on Machine Learning (ICML-14), (2014), pp. 190–198.

[8] X. Tang, T. Yoshida, and W. Zhang, *Tfidf, lsi and multi-word in information retrieval and text categorization*. presented at 2008 IEEE International Conference on Systems, Man and Cybernetics, 2008.