Brandon Cuadrado, 109237297

Pravani Venkata Changamma Meda, 111492602

Zenab Bhinderwala, 109897840

**Automatically Building Book Indices: Final Project**

**Introduction**

An index is an alphabetical listing of words or phrases (usually key words) with references to the places/page numbers where they occur. The goal of this project is to develop an automatic index builder; which takes a LaTeX document and the desired index size as input and outputs an index in a new LaTeX document. The application will use a model learned from existing LaTeX indices to predict the appropriate content for the generated index. The automatic index builder is a command line application developed using Python 2.7.

**Parsing LaTeX Files**

As discussed in the Progress Report, Pylatexenc is used to parse text from a LaTeX file and for cleaning and processing. To refine the process of determining terms, the Parser has been updated to filter out words with less than 2 characters, words containing numbers, and words containing algebraic symbols. While this shortens the scope of the Parser to files with English indices, it allows for a more accurate analysis of the words and phrases making up a document.

To aid in making a versatile Parsing process, the Parser program has been expanded to accept various arguments which allow it to be used in different ways. This is done using the argparse Python 2.7 package. Existing input variables have been refactored to use the “-f” (file) and “-o” (output) flags to specify the input LaTeX file and desired output location.

**Parsing Entire Directories**

The flag -d (directory) has been added to the Parser script which parses data for an entire directory of LaTeX files, and outputs the parsed data to one large CSV file. The source attribute is added to the CSV to indicate the source file. This function is particularly useful during training, as it allows multiple files to be parsed with a single call to the Parser script. Multiple files can then be used to train, with their predicted indices having a reference to the associated file.

**Google Ngram Phrase Finder**

To have a more robust understanding of English language beyond our dataset, Google Ngrams is queried for the usage of each term in the dataset. Some models use this data before predicting an index, while others narrow down the number of terms then use Google Ngrams in creating the final list of indices. To accommodate this, the Parser script uses the flag “-n” (ngram), which is a boolean that toggles the function of querying Google Ngrams for term usage within their database.



The Phrasefinder API[[1]](#footnote-1) made available by Github user Martin Trenkmann (mtrenkmann) provides a simple way to send an HTTP Request to the Phrasefinder search engine. This utility queries Google Ngrams and returns easily-parsed usage data. Multiple HTTP requests can take a while when gathering data for an entire list of terms. To address this, some models will narrow useful terms before gathering data from Google Ngrams, to cut the time it takes to predict an index without sacrificing accuracy.

**Scoring Function**

**Model**

The baseline scoring function was expanded upon from the previous model to include Google Ngram data in its prediction. The scoring model uses the same formula from the baseline model, but with improvements based on a broader scope of data:

In the above function, posScore represents a value determined by a term’s part of speech, ngramScore represents a value determined by the number of words in the term, inf represents a term’s informativeness, and tf(idf) represents the term frequency multiplied by the inverse document frequency.

As opposed the previous model, term frequency and document frequency are calculated using the term frequency among all documents in Google Ngrams and the number of files within Google Ngrams that contain the term, respectively.

**Evaluation**

**Affinity Propagation Clustering**

**Model**

Levenshtein Distance Function. Instead of hamming distance. Hamming is only for strings of equal length.

Levenshtein Python Package

**Evaluation**

**Affinity Propagation Clustering + Scoring Function**

**Model**

**Evaluation**

**Linear Regression**

**Model**

**Evaluation**

**Logistic Regression**

**Model**

**Evaluation**

**Random Forest Classifier**

**Model**

**Evaluation**

**Final Model**

**Model**

**Evaluation**

**Output Index**

**Final Index Creation**

**Automatic Index Builder Script**

1. https://github.com/mtrenkmann/phrasefinder-client-python [↑](#footnote-ref-1)