Meeting Time: 12/11, 4:00pm #3

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.

```
    # search for term x through Google Ngrams using phrasefinder
    result = phrasefinder.search(x)
```

The Phrasefinder API[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:

$$S_t = \big((posScore * ngramScore) + \text{inf}\big) * tf(idf))$$

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.

The posScore is used in the scoring function due to the higher probability of certain parts of speech to appear in an index over others. For example, nouns are the most popular speech tag to appear within an index. The ngramScore is used due to the higher probability that indices with multiple words appear in an index. Informativeness and tf(idf) are also found to be positive contributors toward index possibility. Differing from 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.

[1] https://github.com/mtrenkmann/phrasefinder-client-python

Evaluation

Affinity Propagation Clustering

To narrow down terms further, an Affinity Propagation Clustering algorithm was applied to the dataset. The goal of this clustering process is to find centroids for clusters of similar terms and use those centroids as candidates for the index. Affinity Propagation Clustering is used in favor of the common K-Means Clustering algorithm because the number of clusters is not determined prior to running the algorithm. The Affinity Propagation algorithm builds clusters based on representative elements.

The distance function used to calculate clusters is called the Levenshtein Distance Function, available through the Levenshtein Python Package. Another distance function considered for this clustering algorithm was the Hamming distance function, but this was rejected because the strings would need to be of equal length. Using the Levenshtein distance function, the words of a document were able to be classified by their centroids, which are then used as the potential indices of the document.

Scoring Clusters

Model

With the clusters obtained from Affinity Propagation Clustering, the scoring function can be applied to the centroids to rank them as index candidates. Scoring after clustering allows for additional data cleaning to be executed on a smaller amount of data. This is particularly useful for obtaining Google Ngram data, which requires an HTTP Request to be performed for every term.

Evaluation

Logistic Regression

Model

A Logistic Regression model is implemented to predict the indices among a list of centroids obtained through Affinity Propagation Clustering. This regression model is used to determine binary results; for our purposes, 1 for an index term and 0 for a non-index term. The training features used in this regression model are the part of speech ranking and product of term frequency and inverse document frequency.

Evaluation

Random Forest Classifier

Model

Evaluation

Final Model

Model

Evaluation

Final Index Creation

Automatic Index Builder Script