Brandon Cuadrado, 109237297

Pranavi Venkata Changamma Meda, 111492602

Zenab Bhinderwala, 109897840

**Automatically Building Book Indices: Progress Report**

**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 will be a command line application developed using Python 2.7.

**Reading LaTeX Files**

In the Project Proposal, we suggested TexSoup as the tool to parse LaTeX data from existing LaTeX documents. However, our current parser for reading LaTeX files uses a tool known as pylatexenc. Pylatexenc is capable of outputting the visible text of a compiled LaTeX file, among other functions. Pylatexenc can perform this operation using its LatexNodes2Text class. The LatexWalker class can be used to obtain more detailed information about the document.



The 17 LaTeX documents obtained in preparation for the Project Proposal are used to retrieve data on its words and phrases that are eligible for an index. These same files are used to obtain their current indexes, which will be used to train and test our models as they progress.

**Determining Index Terms**

Index terms are categorized as single words or phrases that contribute to the context of the document. The first model ( baseline ) implements a scoring function using the data collected from the LaTeX file, organized by word or phrase. The success of this model will be determined using precision and recall of the results.

**Obtaining Terms**

**Unigrams**

The potential index terms are obtained by splitting the text variable as obtained above using LatexNodes2Text. Unigrams refer to the single words obtained from the LaTeX file. The text variable is split by various possible characters using the following splitText function.



**Bigrams**

In addition to single words, phrases of two words are included in the dataset. As our model progresses, we intend on including n-long phrases in the dataset. Currently, only phrases consisting of two words are added. NLTK, the Natural Language Toolkit, can extract two word phrases using its bigram function. These are then filtered using the checkWords ( explained below) function on both elements of the bigram.

**Data Cleaning**

Following this operation, the checkWords function is used to filter invalid words from the list of strings. This function is applied to all elements of the words list using the internal filter function. First, checkWords confirms that the word can be converted to a string. Then, it is confirmed that the word is not a stopword or one of the 500 most common English words. The word is then determined to be neither a single character nor a number. Finally, the word is matched against a regular expression such that the string contains at least one letter or number.



**Data Processing**

**N-Gram Score**

Each term is given an n-gram score based on the number of words that appear in the term. Unigrams receive a value of 2, whereas bigrams receive a value of 10, based on their weight in the scoring function. For future models, the score will correspond with the number of words in the phrase. This will allow the weight of this attribute to vary between models.

**Part of Speech Tagging**

Each term is classified using a Part of Speech tag, as defined by NLTK. These tags represent parts of speech such as “Noun”, “Adjective”, and more specific subdivisions such as “Proper Noun”. Each tag has a two or three letter code associated with it. This can be done for words and for entire phrases using the pos\_tag function offered by NLTK.

**Calculating Frequency**

The term frequency is defined by the number of times it appears within the text. Due to words potentially having different POS Tags based on context, the frequency is calculated based on the unique word, n-gram index, and pos tag pair. This can be achieved by grouping a data frame by the paired values and calculating the size of each group.



**Term Frequency \* Inverse Document Frequency**

This feature compares frequency of phrases use in particular document with frequency of that phrase in general use. This is also a metric of informativeness. Term frequency is the frequency of the term in the document. Document frequency is the number of document containing the phrase.

**Informativeness**

*Investigations in Unsupervised Back-of-the-Book Indexing*[[1]](#footnote-1) defines informativeness as the degree to which a key phrase represents the document at hand. It correlates with the amount of information that a specific phrase conveys to the reader. This involves the following procedure.



**Parser Interface**

The Parser Python 2.7 application uses a command line interface. The ‘f’ flag allows the user to input a single LaTeX file as the input and retrieve a single output file as described by the ‘-o’ input flag. Using ‘-d’ provides the same functionality, but for an entire directory of LaTeX files. This allows for successive parsing of multiple LaTeX documents using a single call of the Parser.py script.

**Existing Index Data**

The indices of existing LaTeX files can be parsed using the CreatingIndex.py script. This program runs through all the LaTeX files within a directory and parses the file for all /index tags. It also considers user-defined tags which include the /index command when performing this action. These indexes and data associated with them are outputted to an output csv file for comparison against the general data parsed from the document.

**BASELINE MODEL**

**Scoring Function**

The pos\_tag, term frequency, informativeness, all these are considered as features to evaluate the significance of the term being an index. A function has been developed over these features, to score each term based on its importance.

As nouns and adjectives contribute a lot compared to verbs and other parts of speech, a comparative weighted score has been given to all the terms based on the parts of speech.

Bigrams, trigrams and n-gram( upto 4-5 ) comprise most of the indices compared to the unigrams. So a weighted score is assigned to all terms based upon the degree of n-gram. Unigrams are given lower value ( importance ) compared to bigrams and trigrams.

As of now we have considered only unigrams and bigrams.

St = (( pos\_rank \* n\_gram\_score ) + inf ) \* tf\_idf )

Where; Pos\_rank = rank based on pos\_tag

N\_gram\_score = score based on degree of n-gram

Inf = informativeness

Tf\_idf = term\_freq \* inverse document frequency

The top 50-100 words based on the score are considered for indices.

**Evaluation**

Precision-Recall is a useful measure to evaluate quality or success of prediction. They measure how well an information retrieval system retrieves the relevant documents/features requested by user.

**Precision** = tp / ( tp + fp )

Where; tp = true positives

Fp = false positives

Which can be written as ; Total number of terms retrieved that are relevant/Total number of terms that are retrieved.

**Recall =** tp / ( tp + fn )

Where; tp = true positives

Fn = false negatives

Which can be written as ; Total number of terms retrieved that are relevant/Total number of relevant terms in the database.

Currently we have considered only unigrams and bigrams as a part of indices. With the baseline model (scoring function) to predict the indices, we got a precision of 0.4838 and recall value 0.4921.

**Summary**

1. http://web.eecs.umich.edu/~mihalcea/papers/csomai.flairs07.pdf [↑](#footnote-ref-1)