Fadi Tadros and Anthony Monaco

B490 Final Project

A Simple Tool for Finding "Like" Books

Individuals all over the world are constantly searching for which book to start reading next. The process of using K-grams, Jaccard distance, and min-hashing over a set of books will produce recommend books along similar patterns. This process can be extremely helpful for a searching reader, as well as researchers and students who are looking for data for their analysis. For example, if a user inputs a book the process will recommend numerous other books that you would enjoy.

Project Gutenberg and Data

Project Gutenberg is the oldest digital library in the World Wide Web today. Majority of the books and audiobooks in its site are the full texts. The project tries to make these as free as possible, in long-lasting, open formats that can be used on almost any computer. Early March 2014 Project Gutenberg stated 45,000+ books available for use. This process enabled the design of a remarkable efficient program to help the searching reader in researching "like" books using K-grams, Jaccard distance, and min-hashing algorithms.

This process and algorithm utilizes 1,270 free books from Project Gutenberg for the data set. These books have genres from history to horror. However, in the future implementation this process will include "like genres". The code will be using each book

as a text file, and comparing them for recommendations. This process also shows users each step involved in the process of finding “like books”.

Storage

The data is being stored in multiple formats. As the whole process of finding the Jaccard distance and min-hash between certain books requires the data to be first stored as k-grams. Next, the code stores each document as a series of k-grams based on 2 and 3 words as well as the number of grams selected. After the k-gram is stored, those files will be computed as a hash set, and stored for further analysis.

Implementation

The projects implementation includes, not only the algorithms for k-gram, Jaccard distance, and min-hash, but also a command prompt to allows users to easily initialize the entire library of books (running the algorithms listed above over the entire library of books), add a book to an existing library, or retrieve information about an existing book in an existing library. The k-gram algorithm was adopted from our assignment 2 to be able to handle an ever-increasing collection of books, while using 2 or 3 words as its parameters for the k-gram algorithm. K-grammed books will be stored in individual text files, giving the users a concrete idea of how the algorithm actually works. All books that have been K-grammed will have their title stored in a text file called library.txt. This allows for the users to quickly find distances between books, by reducing the amount of times our k-gram program is run. New books will, thusly, only have to be K-grammed once as the user imports said book. Initially, once a book has been K-grammed, the user will then be able to find the “closest” 10-15 books by either

using the Jaccard distance algorithm, or a min-hash algorithm. Both of these algorithms were taken from assignment 2, but have been updated, debugged, and changed, to work with the new k-gram program. Once all the distancing algorithms have been run, our program stores each individual book's Jaccard distance and min-hash in a text file in their own specific file. The users will be shown only the top closest books, but they will also be able to find more by finding the respective distance text file. The distances are then stored in a unique text file until new books are added. This process allows for repeated searches to be completed much quicker, as the information is "cached", and stored for as long as the readers pleases.

Algorithms

K-gram

The ordinary keyword information retrieval has limitations on processing phrases and sequences of words that have equivalent semantics. However, have different notations including: Author's writing style, individual synonym use, phrasal, and numerous word variations. Using k-gram matching, the process compares sets of k neighboring units, which can be characters, word, and phrases of one string to those of another string ("N-gram" Wiki, 1). However, sometimes this can be a problem due to k-grams fixed pattern ("N-gram" Wiki, 1).

Jaccard Distance

This process includes the Jaccard similarity, which is a process that compares the similarity and differences of n sets. The process measures the similarities between

finite sets. This is processed by the n size of the intersection divided by the n size of the union of sets.

All while, the process in the algorithm for the Jaccard distance shows the intersection between two sets. For example sets 1 and 2 show all books in the collection that appear in both sets. The union between sets 1 and 2 show the books in the collection that appear in either set.

In the algorithm conducted the Jaccard distance records the items between the sets. This process produces the distance by dividing the variance of the n sizes of the union and the intersection of both sets by the size of the union (Phillips 2).

Min Hash

This process of min-hash is able to quickly find the Jaccard distance between two sets by sacrificing accuracy for speed. A great example is as follows:

> "The simplest version of the min-hash scheme uses k different hash functions, where k is a fixed integer parameter, and represents each set S by the k values of $h_{\min}(S)$ for these k functions ("MinHash" Wiki 1). To estimate $J(A,B)$ using this version of the scheme, let y be the number of hash functions for which $h_{\min}(A) = h_{\min}(B)$, and use y/k as the estimate" ("MinHash" Wiki 1).

The estimation is calculated based on the average of k different random variables ranging from 0 to 1, where any variable equals 1 when $h_{\min}(A) = h_{\min}(B)$ and zero when they do not. Each of which is an unbiased estimator of $J(A,B)$ (Wiki Min-Hash). A great example is as follows:

> "Therefore, their average is also an unbiased estimator, and by standard Chernoff bounds for sums of 0-1 random variables, its expected error is $O(1/\sqrt{k})$. Therefore, for any constant $\varepsilon > 0$ there is a constant $k = O(1/\varepsilon^2)$ such that the expected error of the estimate is at most ε" ("MinHash" Wiki 1).

The time that this process can be conducted is in $O(k)$ from both given sets. While When both ε and k are constants, the time to compute the min-hash similarity should also remain constant (Wiki Min-Hash). However, the set 1 of the hash variant takes $O(n\ k)$ time. This process proves that $O(n\ k)$ actually can actually be computed in faster time, requiring $O(n \log k)$time to maintain the sorted list of minimal when using the single-hash variant of the algorithm.

Results

Book	**Jaccard Distance**
1. the Story of a Patriot - Sinclair, Upton.txt **2. the Gadsbys, The - Kipling, Rudyard.txt**	**0.9931741**
1. (Leaves of Grass).mobi - Unknown.txt **2. (Walt Whitman_ Leaves of Grass).mobi - Unknown.txt**	**0.013513505**

1. Twain, Mark.txt **2. Thackeray, William.Makepeace.txt**	
	0.9932203
1. Aaron Trow - Trollope, Anthony.txt **2. Disobedience - Thoreau, Henry.David.txt**	**0.993266**

Future Improvements

Character Analysis and Stopwords:

The current implementation and algorithm allows for all ASCII characters in the data to be processed. Which makes it difficult to match two k-gram documents as grammar is more complex with punctuation (Ex: The Dog. != The Dog). A future implementation of our k-gram algorithm could allow for the omission of every character except a-z, A-Z, and 0-9. All other characters would be considered Stopwords. This would allow for our distancing algorithms to become more accurate and efficient.

Algorithmic Improvements

Although the current implementation of this project's algorithm are completely correct, the algorithms have not yet been optimized to run efficiently, thusly they can be improved upon drastically. An algorithmic improvement would be to reduce the amount of calls the code makes while writing information for the user to text files. The current java files that are affected by this inefficiency are as follows: kGram.java, jaccardDistance.java, and minHash.java. In the code of all of these files, the program is forced to open the document it is writing to every time information is received, and closes the document after. If the information received were to be stored in some format until the entire algorithm is ran before writing the information collected to a text file, the algorithms affected would save copious amounts of computation time.

Dynamic k-grams

Under the projects current implementation, the k-gram algorithm is only limited to handle k-grams of 2 or 3 words. To be able to make this project more user friendly, the k-gram algorithm should be able to dynamically handle any number for k that the future users want. This would allow for users to be able to customize this project more to their research needs.

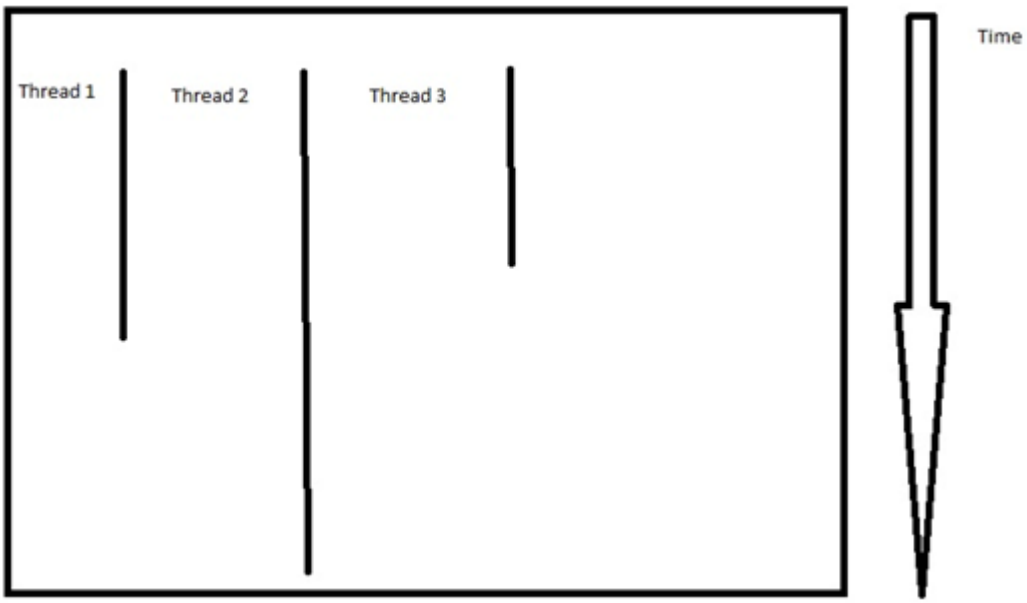
Dynamic Hashing

The process of the min-hashing algorithm is used to get a general approximation of the Jaccard distance between any two books. Thus, sacrificing accuracy for speed. The current implementation uses 40,127 as its number for k. However, if this were to be

changed to a number chosen by the user, the user could gather data on how changing that number can affect the speed of the min-hash algorithm.

Multi-Threading:

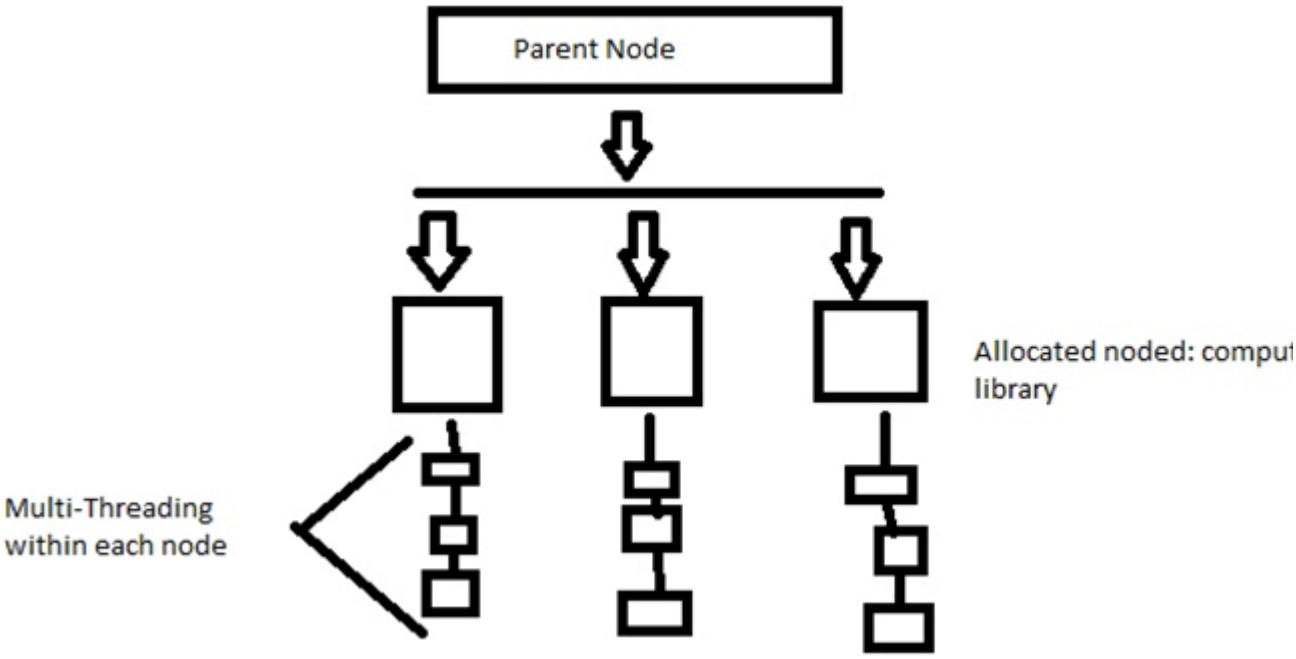
After designing the code and running through the entire collection of books, it was determined that it is not wholly cost effective. Thusly, reducing the accuracy and efficiency. However, if the project included multi-threading in all the algorithms, it would process multiple books concurrently. This should show a drastic decrease in running time based on the number of threads used.



Hadoop

With Hadoop, the code should be written so that the parent node can handle a randomly allocated number of nodes (n). Then, the parent node processes a portion of the data to be initialized to each one of these nodes. Which is defined as the 1/n. Each

node multi-threads its portion of the data until the process is finished. If the future users were to have access to a cloud to implement the algorithms with Hadoop, the user would be able to handle a data collection magnitudes larger than the one this project has processed in its code.



Works Cited

"Jaccard Index." Wikipedia.
Wikimedia Foundation, 25 Apr. 2014. Web. 28 Apr. 2014.

"MinHash." Wikipedia.
Wikimedia Foundation, 13 Apr. 2014. Web. 28 Apr. 2014.

"N-gram." Wikipedia.

Wikimedia Foundation, 27 Apr. 2014. Web. 28 Apr. 2014.

Phillips, Jeff. "Jaccard Distance ." University of Utah. Jeff Phillips. University of Utah, Utah. 29 Apr. 2014. Reading.