Group Members:
Vedanta Dhobley: vjd41@scarletmail.rutgers.edu
Avani Bhardwaj: ab1572@scarletmail.rutgers.edu
Shazidul Islam: si194@scarletmail.rutgers.edu
Akshat Shah: avs91@scarletmail.rutgers.edu
Kutay Kerimoglu: kk851@scarletmail.rutgers.edu
Alan Patel: akp122@scarletmail.rutgers.edu
Anthony Matos: amm720@scarletmail.rutgers.edu
Joel Cruz: jc2125@scarletmail.rutgers.edu

Project Title: Recommending Alpha
Group Number: 16
Date: February 10, 2019

Problem Statement

A current problem today for many avid readers is that after reading a book, people come back to try and find books with a similar taste. They want to read books with specific similarities to the ones they have already read. However, the feedback they are getting from current automated searches at these bookstores and libraries are not up to par with what they want to read. They may go with what the recommended search gives them, and then return as unsatisfied or disappointed customers. While there are software programs that can find books the same way, they lack the more specific tags that the customer is looking for. There are times where the genre may match well but the age recommendation does not work with the customer. If someone were to read an adult fantasy novel that has a very intense vocabulary, and wanted to find books of the same type, some recommendations may give them much simpler books that are not appropriate for their maturity level and they may not want to finish reading their recommended book because it was nothing similar to what they had previously read.

For example, a bad recommendation is being recommended to ‘Twilight’ or ‘Lord of the Rings’ to someone who wanted to read something much more relatable to ‘Harry Potter.’ The most general similarity between these three books is that they all fall under the ‘fantasy’ genre. Other than that, there is probably little to no similarity between these three series at all. The goal is to identify more specific details about books and give recommendations for books that are much more similar in taste. ‘Percy Jackson’ would be a relatively good recommendation for someone who has read ‘Harry Potter’ because it also applies the concept of magic being a major factor in both series, along with the ‘fantasy’ genre. The algorithm being developed should be able to give recommendation based on more than one specific tag that a book has in common with others. In this example, there were two tags that were similar but there could be many more after all of the tags from both books are matched.

Another bad recommendation would be when differentiating between what categorizes as ‘historical fiction’ and a simple history documentary. For example, a student who is trying to write a history paper on the roaring twenties prior to the Great Depression, they would most likely want to find articles and books with facts about this time period, instead of receiving a

search result of ‘The Great Gatsby.’ Although the Great Gatsby was written in this time period, the novel is not focused on the general historical value. It was written as a fictional novel during that time. A student doing a report on American history would probably not want to waste time reading ‘The Great Gatsby’ because of the fictional aspect. The implemented algorithm would be able to differentiate between ‘history’ and historical fiction’ in order to give the user a better search result with books of the appropriate tags.

For the program, there will be two algorithms, which are ‘Recommendation,’ ‘Analyze,’ and a string search being implemented. In order to implement the first algorithm, this search must pass correctly. The string search does not assume that the user will type the book name in correctly. The data, which are the books, will be kept in a database. This search ensures that if the book is in the database, there will be a dropdown menu allowing the user to choose from the options that are in the database. If the book does not exist in the specific database, there will be no search results available. For the tags, the way the search function will work is by using a breadth first search. For this to work properly and most efficient, it needs to be made so that the tags of every book is in the same order For example, Age → Genre → Author → Length → etc. Creating them in a link list formation makes it easier to actually go through the heap and find everything needed. As the developer runs through the tags, it starts tallying the total number of points and then record that next to the book. The breadth first search will be used for optimized time. The tags will be organized in a link list type of fashion that allows for easier more organized traversing. Account for all distinct tags that associate with all the books entered

This also works well for people who are consistent at misspelling words. If words are misspelled, they will not be able to find the book that is actually available in the database. Therefore, having a dropdown menu that starts giving the user recommendations as they start typing the book will make it more convenient for the user to find the book rather than be unhappy when a search is unavailable due to a spelling error. Essentially, there will also be an assumption that the user is searching for books that are already in the database, for the books that are selected from the dropdown menu. For example. if someone were to try to find a book in a different language, this search would return no result because that specific book would not be able to be located in the database.

The program will allow multiple users. To allow this, the program will create a new spreadsheet for a new user. That spreadsheet will be used to store the information about the books entered as wells as the books that we pulled from the database that has the other books. This will help with sorting later down the line. As the user is inputting the names of the book, the system will being to predict what the use is trying to input. This will save the user time and thus increase satisfaction. Not only this but because the data is case sensitive, this will minimize errors and discrepancies

For the second part, there will be a ‘Recommendation’ algorithm. This algorithm will work with the tags we put on the books. Each tag will have a set multiplier (1x, 5x, 10x, etc) added to the books. For example, the ‘age’ tag will have a x50 or so multiplier to ensure that the recommendations that are coming back are in the same age limit of the user/reader. ‘The Lord of

the Rings' would definitely not be a good recommendation for a 5- year old with a very limited vocabulary. Another example is a tag for books written during certain time periods. A modern millennial may find a book with a few similar tags, but when he or she sees the tag that determines what time period it was written in, he or she may not want to read a book that was written such a long time ago because he or she may not be familiar with the vernacular of the time. This is also a useful tool for parents who do not want their children to read books with a much higher maturity level than they can comprehend. A big concern for parents these days are whether or not a book is fit for a child, especially if they have no knowledge about what the book is about. A kid may want a book on monsters, but would probably not be to happy reading a book that would give them nightmares because of a book that was very explicit and too gruesome for even the most avid young readers. This recommendation algorithm will recommend books to users without typing in specific keywords, which is essentially the whole point of the algorithm.

Once the search is done, the results will be displayed in tiers. These tiers will be based on a statistical analysis of each book's tags. The more similar the tag, the more closely related the books will be. This would be considered as the second algorithm being implemented in the program. This 'Analyzing' algorithm uses the points allocated to different books based on our Search Function, using the data for statistical analysis of the results. Using this analysis, we can group our results into tiers (S, A, B, C from highest to lowest) based on standard deviations. This allows the user to see not only which books are recommended to them based on their inputs, but also how how closely related each result is, giving a more organized and focused list of results. The S tier books will contain books in the top 0.5% , A tier books will contain books in the next 2%, and so on and so forth. Using a standard deviation to split the tiers, we will decide a baseline score required for each book to earn a spot on the list of recommended books. This is the most optimal way for organizing the books for the user because it is easy to implement on a user interface and will be convenient for the user to see the tiers and maybe just have to decide between 5 or 10 books instead of 50+ books.

The most important feature in this program would be the recommendation algorithm. This algorithm needs to identify the tags on each book in the database in order to return the tiers. It will group the books that have the most tags in common, and list them after the user selects the book from the dropdown menu. The tags can include things such as genre, age level, "Disney", number of pages, difficulty of vocabulary, etc. The database is required to have these books with these tags in order for the search to go through. The database that will be used will be an open source library with about at least 50,000 books that will be searched through for their tags.

For the user interface, there will be an easily accessible website to use. So every input, whether it be from a input device or a smartphone screen, will be handled through features of the user interface and will essentially allow the program to perform its job through user demands. The list of books that have already been read will be displayed horizontally. Below this list of books that are already read, there will be a section called "Top Picks for You." This section will be based off all the recommended books that the user has displayed interest in previously. There will be a trending section which lists all the new books that are based off popularity and reviews. When scrolling through each book, there should be a brief summary of the books, the ratings,

and the genres. Any additional lists will have other genres for easy access to a book that is **not** part of the user's preferred genre. For example, these lists may include: comedy, horror, romance, drama, etc. However, only a few genres should have their own list at this point because if not, users would have to keep scrolling to find their preferences which may end up ruining their experience on the platform. As previously mentioned in the search string implementation, the drop down menu can also be used to have users explore further. In any list, the image of each book will be its cover so that the user can easily identify the book without any difficulty. every input, whether it be from a input device or a smartphone screen, will be handled through our features of the user interface and will essentially allow the program to perform its job through user demands. There will need to be a way to check the algorithm consistently after it is implemented. The database will constantly need to be refreshed as new books are added to the system. As the database is updated, the tier list will change for every input that is added in. If more books are added, more tags will also be added. This algorithm should be able to take this into account when the search is being done by the user.

The on-screen appearance aspect of this project is where most of the front-end design and development will be hosted. Along with the front-end work, this is also where the user interface aspects of this project will reside. The front-end design and development will drive how the visuals and website aesthetics will not only look, but also operate accordingly to the functional demands of the backend partition. All of the visuals of the website and user input will be held and driven through the on-screen requirements. The on-screen appearance aspect of the project will also be where the user-interface will strive. So every input, whether it be from a input device or a smartphone screen, will be handled through the features of the user interface and will essentially allow the program to perform its job through user demands.

There are a few main goals that need to be accomplished by this project. First, there needs to be a well made interactive website for the user to connect with the database. This database will need to be managed by a developer who updates the database every two weeks or so to make sure that the latest trending novels are included in the database. The website should be user friendly, and look the part. It should be easy to navigate through and easy to read. Next, there should be random checks to make sure that the tier list that was created is up to date with the database. There should not be books left out when more are added into the system, and each books' tags should be registered into the system as well. As more books are added, the searching string needs to make sure that it takes into account the added books. Adding more books into the database should not cause the algorithm to run slower. It should be optimized at the same speed that it was running on prior to the adding of new books into the database. The biggest challenge will be optimizing the runtime for this algorithm to run efficiently and quickly. Since there will be more than 50,000 entries, it might take a while to get a recommendation from the database which requires optimization. Getting the runtime to be efficient will require a lot of testing on a public database which will give good feedback on how efficient the algorithm is. Even after the program is launched, it should be checked on occasionally to make sure that the runtime is not affected badly at any point in time.

Glossary of Terms

Algorithm: A process or set of rules to be followed in calculations or other problem-solving operations.

Analyze: The algorithm that is used to separate the returned books into a specific tiers.

Books: The entries in the database.

Database: Hosted on the website, used to store the entries (books) and their tags.

Entries: Every single book in the database is an entry itself.

Feedback: Information about reactions to a product, a person's performance of a task which is used as a basis for improvement. For this project, the feedback would come from how well it works for users.

Tag: Words and short phrases that are used to describe a book.

Tier: Subsections which identify how close in relationship two books are. It includes a list of books that are related to the searched book.

Search: The algorithm that is used to find the user input from the database.

Recommendation: The algorithm that is used to recommend a book based on the tags that is associated with it.

User Interface: How the user and the computer system interact with each other. In this case, it will be a website that the user uses to search for their recommendations.

String Search:

Platform: The basic hardware (computer) and software (operating system) on which the applications are run.

Optimization: Bettering the algorithm so its runtime is faster and more efficient.

Multiplier: Associated with the tags. Each tag has a multiplier. They vary based on the importance of the tag.

Vernacular: The language or dialect that is spoken by people in a specific region or country

Enumerated Functional Requirements

Identifier	P.W.	Definition
REQ-1	**2**	Use a breadth for search for optimized time
REQ-2	**5**	The tags will be organized in a link list type of fashion that allows for easier more organized traversing.
REQ-3	**4**	Account for all distinct tags that associate with all the books entered
REQ-4	**4**	Create a new table every time a new user enters the list of books (New user just means a new list. Users are not recorded)
REQ-5	**5**	Spreadsheet will hold a list of books that have tags that are matching the distinct tags list
REQ-6	**5**	The list will include all books that have at least Age and Genre matching.
REQ-7	**1**	(Optional) Delete the last sheet to save memory allocation
REQ-8	**4**	System will begin predicting user input
REQ-9	**2**	Update prediction with ever letter input
REQ-10	**4**	The spreadsheet list will be organized via point system to rank them
REQ-11	**4**	Ranking will be broken via percentage tile of the highest point in that list 95%+ =S, 90%-94%=A, 80%-89% =B, 70%-80%=C
REQ-12	**2**	System will have a short description about the tier in the box located next to the letter
REQ-13	**4**	List should hold tags about the books

Enumerated Nonfunctional Requirements

Identifier	P.W.	Description
REQ-1	**3**	Database should be updated once week to account for new books and other updates
REQ-2	**5**	System must be able to handle infinite amount of inputs and display finite amount of outputs for accurate representations. This should not take longer than 5 minutes
REQ-3	**4**	System must display suggestions for spelling mistakes or if the inputs are not written exactly as the actual book name (even if few words are written, or should read 50 as the word “Fifty” in the book and vice-versa)
REQ-4	**5**	System should display an error message for invalid inputs, even if there are few valid list of inputs followed by an invalid on (“Try again” button that refreshes the system.)
REQ-5	**5**	System should prompt an option to start anew after displaying output
REQ-6	**4**	System must be able output the recommended book using the algorithm in a quick fashion (Again limit is 5 min)
REQ-8	**3**	Program should be able to handle multiple clients accessing the program
REQ-9	**5**	Application must be straightforward to use
REQ-10	**5**	System code must be broken down into manageable and maintainable parts
REQ-11	**5**	Must be able to run on web browsers without error
REQ-12	**5**	Output should be displayed only after all desired inputs are submitted

On-Screen Appearance Requirements

Identifier	P.W.	Definition
REQ-1		
REQ-2		
REQ-3		
REQ-4	**1**	A list of books already read should be displayed horizontally.
REQ-5	**5**	Below the list of books already read, a “Top Picks For You” section should be dedicated to all the recommended books based on the user’s interests
REQ-6	**2**	A trending section must list all books based on popularity and reviews
REQ-7	**5**	When scrolling through each book, there needs to be a brief summary of the book, the ratings, and it’s genre.
REQ-8	**4**	Additional lists should have other genres for easy access to a book that’s not part of the user’s preferred genre. For example, such lists are comedy, horror, romance, drama etc. However, only a few genres should have their own list at this point. Otherwise, the user would have to scroll further down to see all genres which can ruin the user experience
REQ-9	**2**	A browse drop down menu should list all possible genres for users to further explore.
REQ-10	**4**	In any list, the image of each book will be its cover so that the user can easily identify it.

References

https://blog.statsbot.co/recommendation-system-algorithms-ba67f39ac9a3

https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine

https://www.ibm.com/developerworks/library/os-recommender1/index.html