For any individual making a choice, be it their next show to watch, who they should date, or what kind of cereal should they buy, there can be hundreds, if not thousands, and possibly even millions of options for the individual to choose from, often times overwhelming the person. This phenomenon is referred to as the paradox of choice in decision making. Given information on these individuals and their preferences, we can assist this choice by recommending items that our user may enjoy, with the overall goal being to make the process of decision making easier. The choice we decided to focus on for this project was on what to read next. Based upon your preferences, can we a recommend a book (or set of books) that you would enjoy and that may overall enhance your experience?

Even though the principles behind our recommender systems can be utilized for any individual or company looking to recommend books, such as Barnes and Nobles or Amazon, we decided to focus our efforts on the users of the Goodreads website. Goodreads is a platform in which users can archive and rate the books they have read, receive recommendations on what to read next, and share and discuss what they are currently reading with their friends. The algorithms we developed look to enhance the user’s experience by recommending that the users enjoy and therefore rate generally high.

For this project, we used the Goodreads data set available on Kaggle (<https://www.kaggle.com/zygmunt/goodbooks-10k/home>). This data set has information on 10,000 books and over 50,000 users. However, we were forced to drop about 80 books, primarily due to issues when reading in books of a particular language. For each book, we pulled the top tree tags associated with that book after deleting unnecessary tags such as “to-read”, ‘have read”, “favorites”, etc. We decided to supplement this data set with additional metadata on the books which we pulled from the Google Books API such as genre, number of pages, maturity, google rating, etc. Once we have our final data set of Google Book variables and Goodreads variables (book\_features.csv), we can move forward in creating our feature space for the content-based recommendation system.

We now create dummy variables for each of the features we want in our feature space from our variables. For certain variables such as the genre or tag, we have hundreds, possibly a thousand potential options, so we decide to select from the roughly 50 most popular genres and roughly 50 most popular tags. For the book feature space, we have 9915 books on nearly 130 features which we now need to normalize so that each variable is between 0 and 1 as to not overweight a particular feature. We also decide that the author of the book would probably be a very crucial feature in recommending a future book, so we decide to create an author feature space from the same features and normalize in a similar fashion. We have 3840 unique authors on about the same number of features. We want to embed the non-normalized author feature space into the normalize book feature space, so we first decompose the author feature space into a reduced number of features and normalize so that values are between 0 and 1. We can now compute the recommendations based on nearest distance and different metrics.

We were able to create five separate recommendation algorithms based on our feature space. The first oddly enough does not use any of the features that we have discussed, rather we use matrix factorization to impute the missing ratings to create a dense user-book rating matrix. From this matrix, we can select the user and order the books with the highest ratings to recommend a book. This is by far the least complex model we employ since we use the matrix factorization as a sort of black box approach.

The second model we employ is the short of the average distances. Based on the book feature space, we compute the distances between each vector (book). From the books a given user has rated, we then average the distances to all other books and select the n number of books that have the shortest distances.

The third model builds upon the previous model. In our feature space, averaging the distances between all the books the user has rated may convolute the recommendation. For example, we have a user who enjoys classic fiction and books about birds. By averaging across all books, we are simply splitting the distance; we might get lucky and recommend a book that is classical fiction and about birds, but most likely, we are not capturing this nuance. For each book we could possibly recommend for a user, we select some number of observations with the smallest distances that the user has previously rated and average those distances. We are taking n number of minimum distance and averaging those and recommending based on smallest average minimum distance we observe.

The fourth model we use builds a vector for the individual user and recommends based upon the books nearest that user vector. For this model, we are trying to create an individualized vector, like a fingerprint, for that user. We take the observations the user has positively rated and average the features of those observations to create this user vector. Obviously, we run into the problem we outlined previously about differentiating between a user’s different interests. However, we then select the books that are nearest this user vector, and we actually find to have the same results for our example as we did in the second model.

The final model that we explore builds upon the third and fourth model, making it the most complex model that we employ. Rather than creating a vector for the user’s book preferences, we decided to create a vector for the user’s author preferences. We take the authors the user has rated positively and we select some number of observations that have the smallest distances to the other authors and average those distances. We select the authors that have the smallest of the average of minimums as the recommended authors. From these recommended authors, we filter out any books not written by them and again select some number of minimal values and compute the average on these books. From these books, we select some number of books with the smallest of the average of minimum values as the recommend books.

For recommender systems, there is a level of creativity that has to be utilized in thinking of how we wish to construct our algorithm. The above five can be further modified, tuned, or combined to create any number of recommender algorithms, and we could further utilize some ensemble method between each of the algorithms to solidify the recommendations. Finally, we need some method or metric to be able to evaluate the results from our recommendation system. In the business world, we can compare the recommendations that we generated against what the user actually thought. However, this requires new data and possibly some hypothesis testing. We could employ a train-test split idea to see if our models would recommend books that we excluded from the user’s observations. This could be computationally expensive. Rather, we could look at our example user and examine which algorithm recommends something in one of the series that he or she has rated.