Reado - a contextually relevant and meaningful book recommender

Tejas Dharamsi[1]*, Janak A Jain[1]*, Vijay Balaji[1]*, Abhay S Pawar[1]*

Abstract
Books are an important part of a person's life. They cheer, educate and connect us in ways nothing else can. Readers are often on the lookout for meaningful and relevant recommendations. It is in this spirit of enquiry that we have decided to work on the problem of coming up with a recommender system that resolves this problem while offering to personalize the experience for its users. Our **objective** in this project was to develop several different models of recommendation viz. LSH (Locality Sensitive Hashing) - an ANN (Approximate Nearest Neighbors) approach, Content Based CF etc. **from scratch**, compare them with traditional Collaborative Filtering algorithm (treating it as benchmark) and ultimately blending them together to create a **hybrid** recommendation system. We have merged data from a few sources and enriched the data both using external sources (viz. BookReads API) and pre-trained models such as Word2Vec. The reader is welcome to check our GitHub repository here.

Keywords
recommendation, books, machine learning, LSH, Approximate Nearest Neighbours, content, filtering, word2vec

[1] *Data Science Institute, Columbia University in the City of New York, New York, USA*
*__Emails__: {td2520, jaj2186, vb2428,asp2197 }@columbia.edu

Contents

Introduction

Motivation: Thanks to Gutenberg and now, the digital boom, we now have access to a huge amount of collective intelligence, wisdom and stories. Indeed, humans perish but their voice continues to resonate through humans brains and minds long after they are gone - sometimes provoking us to think, making us parts of revolutions and sometimes confiding in us with their secrets. They have the ability to make us laugh, cry, think - think hard, and most importantly, change our lives the way, perhaps nobody else can. In this sense, books are truly our loyal friends.Can the importance of books as loyal friends ever be overestimated? We think not. Which is why we think that creating just the 'right' recommendations for readers is a noble objective. Consider it a quieter (Shh.. no noise in this library!) Facebook or a classier Tinder for those who like to read and listen, patiently.

Business Case: Milton Keynes lives in England and like almost any other 29 year old, he is finally in a place in his life where he has a job that pays the bills, a nice place to stay and finally some time that he can use to focus

on his hobbies. Milton likes reading -but just not like any other person, he loves reading so much so that he has a small library in his house where he spends about 4 hours on average everyday! He has already read and reviewed about 450 books. Now, that is something you don't expect a typical 29 year old. But Milton is not alone. There are many others like him, who like to have conversations with their favourite books. It wouldn't be an exaggeration to say that they live partly in a different dimension (fiction lovers will like this).

As it so happens, Milton's best friend Jake can't stop thinking about using the data on these ratings to find people similar to Milton and creating a recommendation algorithm that generates a list of books that he should read. This would make a great birthday gift for Milton - for a birthday which is not far! But the problem is much complex than it seems.

Firstly, the data is extremely sparse (99.99%). Secondly, the definition of similarity is very vague. How do we characterize similarity? How do we select similar entities?

In this project, we have developed several models **from scratch** to get a deeper understanding of personalized recommedation systems and develop one such solution to a problem that people like Milton face.

Approach:In this project, we have adopted a hybrid approach of using ratings as well content based features to find nearest neighbours. To reduce runtime, we implemented Tree Based Approximate Nearest Neighbours and Locality Sensitive Hashing from scratch.

Product: The product of our project is a collection of python modules that we developed **from scratch** that implements novel LSH, tree based ANN and Collaborative Filtering approaches to create recommendations which are compared against traditional Item-Item Collaborative Filtering Algorithm.

1. About the Product

1.1 Vision

To provide meaningfully relevant book recommendations for readers based on their readership data.

1.2 Mission

We aim to achieve our vision through a systematic process which is given below:
1. **Meaningful**: The recommendations have to be something that the user is expected to find useful. Usefulness in this context can have several definitions in terms of genre, publisher, author etc. We begin with an assumption that similar people read and rate similar books and hence this similarity becomes a good discriminant for selection.
2. **Relevant**: There has to be a measurable performance that can represent the relevance of a recommendation. On a five-point scale, we would not want the estimated ratings to be "off" by more than naive baseline obtained by assigning mean rating of the dataset to each prediction. i.e. predicting average rating for each book.
3. **Exciting**: We would want out recommendations to have an element of serendipity - indeed, a recommendation is more meaningfully relevant if it excites the user (which is the ultimate objective). We examine this later in detail.

2. Objectives

Our objectives was to gain practical experience with modern personalization frameworks and algorithms.

We have broken down our main objective into sub-objectives as follows:

1. Build a **contextually relevant** and **meaningful** recommender system that is able to use information about users and items and recommend books that are relevant to the user.

2. Experiment with several types of techniques and compare the pros and cons of each to see how they achieve Objective 1.

3. Evaluation Metrics

There are several ways of evaluating recommender systems and in this project we aim to comprehensively evaluate each of our models on the following metrics:
1. **Root Mean Squared Error (RMSE)**: calculated as the root of the mean of the squares of all the error terms for a given dataset and is used a metric to assess the deviation of a model's predicted values from the actual values.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(c_i - \bar{c}_i)^2}$$

2. **Mean Absolute Error (MAE)**: is a measure of difference between two continuous variables. It is possible to split a given MAE value into two components: Allocation disagreement and Quantity disagreement, and this helps in getting a deeper understanding of a given model's performance.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t|$$

3. **Coverage**: Gronoos [5] explains coverage as a measure of the domain of items over which the system can make recommendations. This general definition can be refined further to two specific ones each with a different viewpoint. The first definition describes coverage as the percentage of items for which the recommender system is able to generate a recommendation. The second definition describes coverage as the percentage of the available items which effectively are ever recommended to a user. [5][1]
4. **Novelty**: Novelty is a measure of serendipity. Although novelty is subjective, a generally accepted understanding expresses an item as novel if it is different from what is usually seen. Another definition of novelty is defined in terms of the amount of information its observation conveys and is defined as the negative log (base 2) of its corresponding probability. In this project we evaluate novelty by looking recommendations on a case-by-case basis and judging them qualitatively.
5. **Time**: Running and computation time for a give recommender system is also an important in determining its viability in terms of being used as a practical solution to a given business problem. Therefore, we consider the computation time of each model in evaluating its performance.

4. Dataset

For our objective of book recommendation system, we used two datasets viz. Book-Crossings Dataset [4] and Amazon Books Dataset [2].
Book Crossing Dataset consists of 278,858 users providing 1,149,780 ratings (explicit / implicit) about 271,379 books. After removing the implicit ratings there remained 52,097 users and 129,168 books with sparsity of 99.99995%. Amazon Dataset consists of 8,026,324 users and 2,330,066 books with a sparsity of 99.99999%.

We wanted to work with a dataset which is not as highly sparse as the datasets above and since we were running the recommendation algorithms on our local machines with 8G RAM it was essential to decrease the size of datasets. We did this by combing both the dataset and sampling. Prior combing the dataset, we changed the scale of Book-crossing dataset to 5 similar to Amazon Dataset.

For the purpose of sampling, we made a threshold to keep the books which have been rated by atleast 53 unique users. The sampled dataset consist of 12,059 users and 4,959 books and a sparsity of 99.2%. At an average each item has been rated by 95 users and each user has rated 39 books at average. The rating scale for the dataset is 1-5.

The meta-data associated with books were extracted using Goodreads API [3] using ISBN of the books.

5. Models

5.1 Baseline & Benchmark Models

Naive Baseline:
In order to validate our models, we came up with a naive baseline model which assigns prediction rating for every item in the test dataset as the average rating in the train dataset. MAE for our Naive model was 0.763 and RMSE attained was 0.9444. These values help us to compare the efficiency of our hand-coded models.

Metric	Value
Test MAE	0.763
Test RMSE	0.944

Traditional Item-Item CF for Comparison:
Traditional Item-Item CF is used for comparing training time and evaluation metrics (MAE, RMSE and Coverage) for our implementation of tree based ANN, LSH, Content-Based and Hybrid Model. We used the implementation from part 1 of our project to get results. For a review of collaborative filtering, the reader is suggested to the introductory paper on the topic by Sarwar et. al.[7]

Metric	Value
Training Time	4.1 hours
Best K	15
Test MAE at best K	0.553
Test RMSE at best K	0.759
Coverage at best K	76%

These are the metrics we our going to use for comparing our implementation of models developed in Part 2 of the project.

5.2 Tree Based ANN

Various tree based ANN models have been tried and tested. For our project, we built something very similar to what was discussed in class. First, we tried to build randomized trees on full matrix. But, due to sparsity of the matrix, the leaves formed typically had very few items. Hence, we reduced the matrix to dense lower dimension matrix using randomized projections.

Randomized projections solved the problem of sparse leaves in trees. Randomized projections are known to maintain similar distances between items as is seen in full matrix. Unfortunately, the ratings matrix has missing values and distance is not calculated using the complete vector, but only using non-missing values. To make sure that similar distances are maintained, we compared distances calculated from full and reduced matrix. We got a correlation of 0.85 and largely similar values.

The code implemented avoids recalculating item pair distance by checking if the distance has already been calculated while processing a previous leaf. Based on how the randomized tree was formed, we observed that only around 30% to 60% of total item pair similarities were needed to be calculated.

We decided to refrain from building the randomized trees by ourselves due the complexity involved and decided to use an existing implementation. We used sklearn's RandomTreesEmbedding to create the randomized tree. For getting randomized projections we again used sklearn's implementation. Rest of the code was built on top of existing CF using KNN code.

We also faced issues with respect to getting the right trees. Some of the randomized trees formed contained a huge chunk of items in single leaf. We tried tuning the min_samples_split and other similar parameters to get the number of items in each leaf to be not too high or too low. Also, the code has massive opportunities to be parallelized and hence, improve runtime further. Some ways to parallelize would be to search the leaves

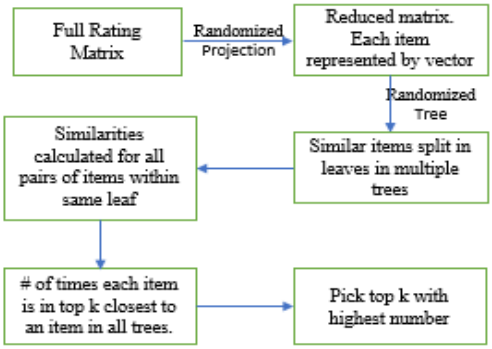

Figure 1. Steps to find closest neighbours using tree based ANN

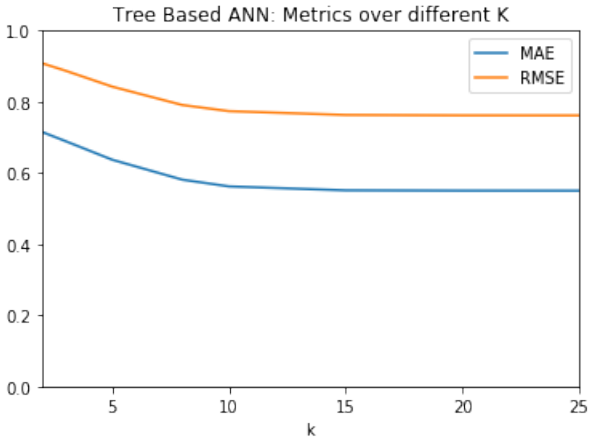

Figure 2. MAE/RMSE: Grid Search results for Tree based ANN

in parallel and parallelize similarity calculation.

The performance metrics We also toyed around the idea of using the reduced matrix from randomized projections for similarity calculation. Reduced matrix was used only for building trees until then. We saw promising results on smaller dataset. Runtime was reduced to around one third for size=10 projection and similar RMSE metric. We couldn't replicate the results on our larger dataset due to time constraints.

Metric	Value
Training Time	1.927 hours
Best K	20
Test MAE at best K	0.55
Test RMSE at best K	0.76

5.3 Locality Sensitive Hashing (LSH)

LSH [6] is one of the methods for Approximate Nearest Neighbors (ANN) approach. In Nearest Neigbours ap-

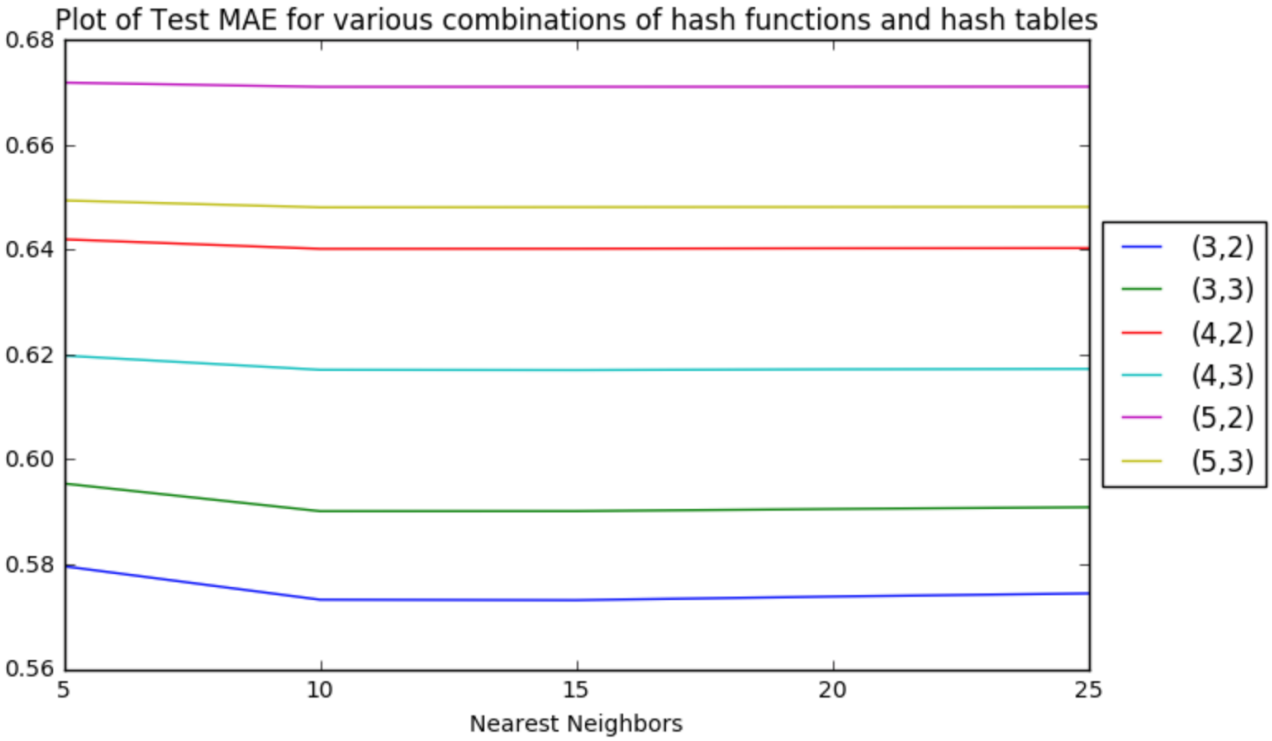

Figure 3. Test MAE: Grid Search results for LSH

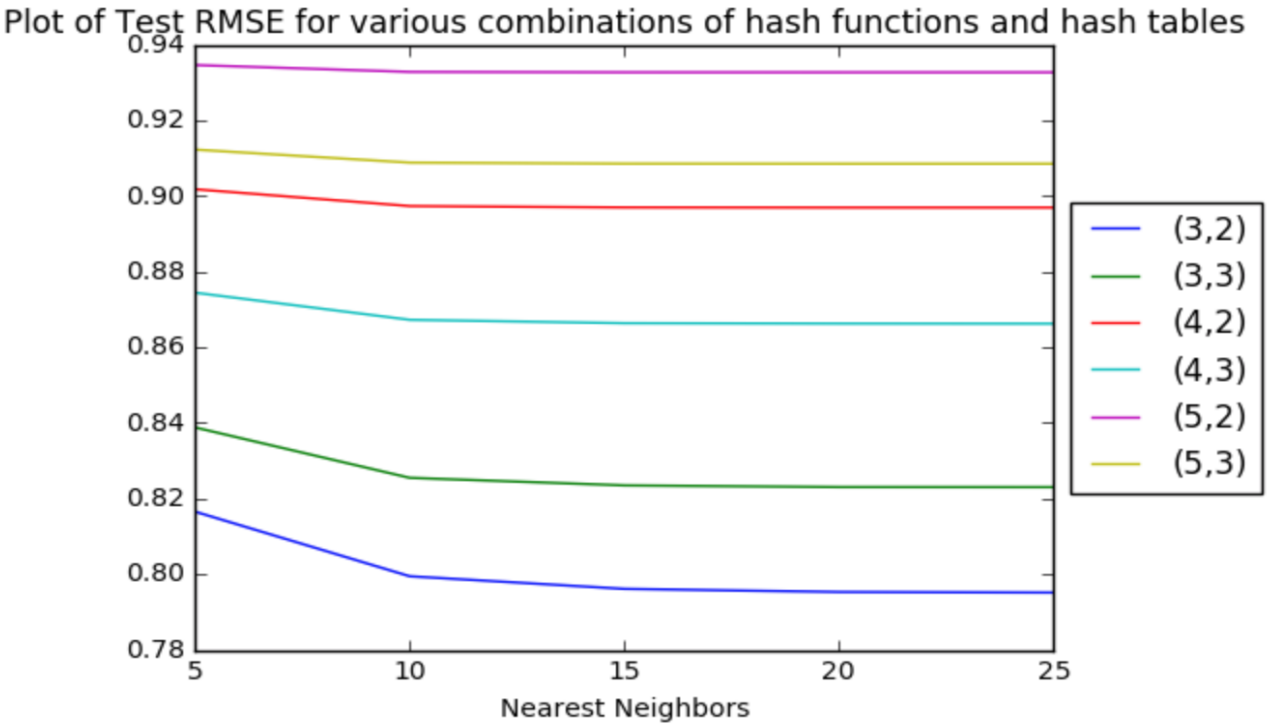

Figure 4. Test RMSE: Grid Search resuls for LSH

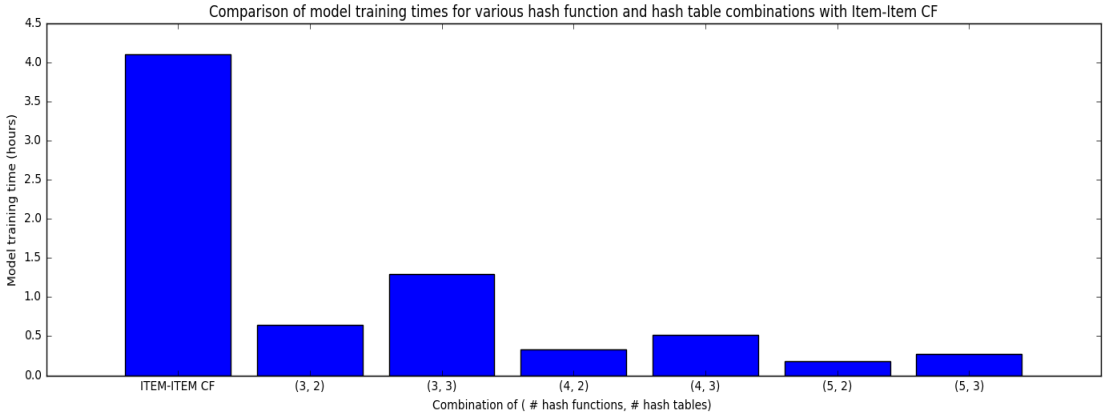

Figure 5. Model Training Time Comparison between CF and various LSH

proach to compute most similar items to a given item i under consideration, time complexity is O(n) to compute the similarity between each other item with item i and find the most the similar. To create a offline model of all-pairs similar items, time complexity will be $O(n^2)$, which is the case in traditional Item-Item Collaborative Filtering. The time to create the similarity matrix will be considerably longer when the number of entities under consideration are more. To overcome this time barrier, approximate nearest neighbours try to generate candidates set (small subset of all items) which have higher likelihood of being similar to the item under consideration. The broader idea of ANN using hashing is that items which hash to similar buckets have more probability of being similar.

Our Implementation: In LSH, we consider a random hyperplane and find the projection of the input query data-point on the hyperplane. If the value obtained after the projection is greater than zero, make it one else it will be zero. Here the random hyperplane can be considered as a hash function emiting values either zero or one. We considered more hash function (random hyperplane) and concatenate the results from each of them. This creates a bit representation (or signature) of the data-point using the multiple random hyper-plane. The bit representation is used to find the index associated with the data point when storing in the hast table L. For example, 1001 becomes 9, binary to decimal transformation. This is performed for each input data-point and their corresponding index in the hash table is found. With LSH, there are chances of false positives and True Negatives, so more Hash Tables can be generated using different g(v) i.e. different group of random hyperplanes to improve the chances of finding similar items. Candidate pairs are those that hash to the same bucket atleast once.

Overall the LSH can be divided into two phases: indexing and search. In the first phase each data-point is mapped to a single bucket in each L_i hash table using $g_i(v)$ hash function. In phase two, the query point q is also indexed to buckets using the same $g_i(v)$ hash function and other items in the same buckets as q are selected as candidates. Similarity of these points is computed using cosine similarity and k closest are selected.

LSH Evaluation: We followed ref [1] for our LSH implementation and varied two hyperparameters **k** : number of random hyperplanes in the hash function and **M**: number of hash tables. Index phase can be termed as training phase to generate item similarity matrix. Since there are approximately 5000 items we decided to use hash functions of size 3, 4, 5 which would result in 8, 16, 32 buckets respectively per hash table suggesting at 612, 313, 156 respectively items in a bucket of a hash table. We tried either 2 and 3 hash tables in the grid search.

Testing was performed using 5-fold cross validation. The Test MAE and Test RMSE for our dataset can be seen in Figure 3 and Figure 4. One can observe two trends from the figure, 1) better results with more hash tables, 2) results deteriorate with increase in size of hash functions. Reason of one, is number of candidates increase with increase in hash tables. Reason for two is the daat-points get scattered two different bins as number of bins increase with increase in size of hash functions. **In our case, best MAE of 0.573 and RMSE 0.796 at K=15 for three hash functions and three hash tables.**

The coverage for best model produced by LSH for recommeding 10, 15, 20, 25 books is 52.5%, 65.6%, 73.8%, 79.5% respectively.

From Figure 5, we can observe that LSH models take very less time to train/ compute similarities with other items. Our best performing model with three hash functions and three hash tables take 1.29 hrs for training.

Metric	Value
Training Time	1.29 hours
Best K	15
Test MAE at best K	0.573
Test RMSE at best K	0.796
Coverage for 15 books	65.6%

We can observe that with small trade-offs in errors MAE and RMSE, our LSH trains the model in 1/3 time as compared to traditional Item-Item model discussed in section 5.1.

5.4 Content Based Model

We obtained descriptions, title and other meta data associated with the books from the Goodreads API using the ISBN values of the books.

Our idea was to extract the elements of genres from the book descriptions and use them as features. We did some pre-processing of the book description prior to getting features from it. All English stop words were removed from the descriptions. This is performed to get rid of extraneous information in the data, which does not contribute much to the description of the book. We essentially want to deal with keywords in the description which strongly indicate genre related to the book.

The intuition behind our method is that given a genre, words from the description of a book will have good similarity scores with the name of that genre. We have considered 14 genres of books: Science, Satire, Drama, Romance, Mystery, Action, History, Religion, Horror, Travel, Fantasy, Children, Biography and Autobiography.

Now how does the model know the context in which these words have been used? The only way to do this is to feed the model with lots of data and many contexts. This is exactly why we have used Google's word2vec [8] model to extract features from descriptions of books. Google's word2vec model has considered context of these words over millions of sentences.

Word2vec is a neural network model which transforms input words to vector embeddings of those words. Each genre words are vectorized, we can compare similarities between words in the context provided by the input. Now we compute similarities of each word in this description with each of the 14 genres. Using this approach we get proportion of each genre in our book. So in the end we are left with 14 genre scores, which is our feature set.

We use these scores to compute neighbors as part of content based model.Since LSH is more suitable for sparse input, we chose to generate context based similarity matrix using collaborative filtering. Computational time complexity is $O(n^2)$, however this time there are only 14 feature vectors as compared 12053 sparse vectors in form of the users.

Content Based Evaluation

Testing was performed using 5-fold cross validation. The Test MAE and Test RMSE for our dataset can be seen in Figure 6. Best value of nearest neigbour **K** is 25. **In our case, best MAE of 0.593 and RMSE 0.803 at K=25**

The coverage for best model produced by LSH for recommeding 10, 15, 20, 25 books is 26.31%, 31.5%, 35.4.5% and 38.57% respectively.

Metric	Value
Training time	0.6 (approx.)
Best K	25
Test MAE at best K	0.593
Test RMSE at best K	0.803
Coverage for 15 books	31.5%

5.5 Hybrid Model: LSH + Content Based

It can be observed that coverage for our content based model is relatively low even though the errors are very close to the item-item CF. In order to improve the coverage and relevant results from Content Based Model, we decided to come up with a Hybrid model leveraging best of both Content-Based and LSH. Main aim is to evolve a model with variety of LSH and relevant recommendations of Content-Based to make recommendations that

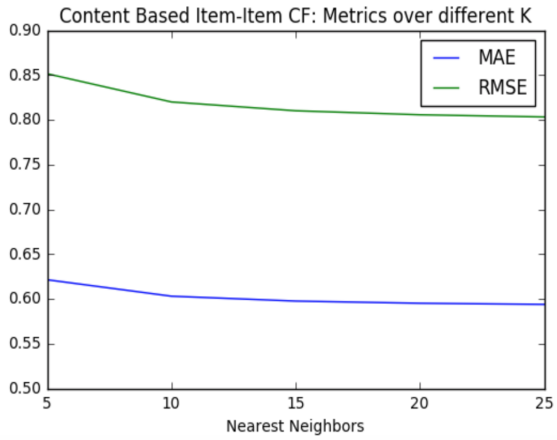

Figure 6. MAE and RMSE for Content Based CF over different K

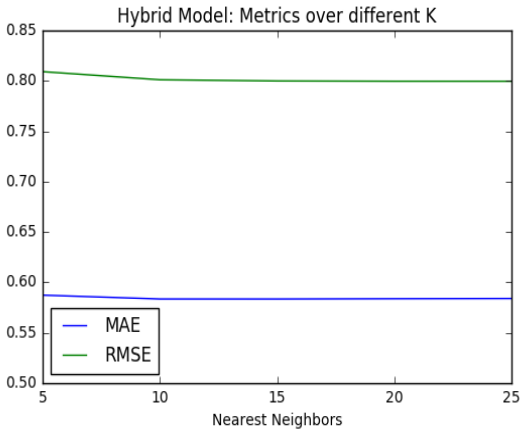

Figure 7. MAE and RMSE for Hybrid Model: LSH + Content Based over different K

would be meaningful as well as novel.

After developing the Hybrid model using LSH and Content-Based Model, the next step for us was to evaluate it different values of its hyper-parameter - the distribution of weights on the two underlying models. Given below is a summary of the MAE and RMSE metrics for the Hybrid model for various combinations of these weights:

W_{LSH}	$W_{Content}$	MAE	RMSE
0.9	0.1	0.587	0.813
0.8	0.2	0.585	0.806
0.7	0.3	0.583	0.799
0.6	0.4	0.583	0.796
0.5	0.5	0.583	0.792

To improve the coverage of Content Based Model, we selected weights 0.7 for LSH and 0.3 for Content Based. The coverage offered by this model for recommending 10, 15, 20, 25 books is 38.9.%, 46.54%, 52.63% and 57.85% respectively.

The coverage has improved with respect to Content Based Model, however, it is not up to the mark with respect to best LSH model. As a result, we still prefer to used LSH model owing to similar lower error metrics, better coverage and smaller training time.

6. Model Comparisons

Please refer to Table 1. In this table we have compared all the developed models on various evaluation metrics for a comprehensive review. We can see that our implementation of models are comparable to benchmark item-item CF model.

7. A look at Serendipity from our preferred model: LSH

Our best model is LSH - which has comparable values of MAE and RMSE versus the traditional item-based CF model. Moreover, LSH trains in about a third of the time taken to train the item-based CF model. Another evaluation metric is serendity or novelty of recommendations.

An example of recommendation is shown in Figures 8 and 9. An interesting recommendation that can be observed from Figure 9 is ”Don Quixote”. It belongs to a genre that is not currently present in the user’s rapport of genres. What’s more is that Don Quixote is considered one of the most influential works from the Spanish Golden Age.

Upon closer observation, we find that Don Quixote contains several thematic plots and stylistic elements which are very similar to other books that the user has read. Moreover, such a serendipitous result is also likely to be liked by the user given the higher chances of similarity in stylistic and thematic patterns.

8. Conclusion and future scope of work

In this project, we have created a hybrid model of personalized recommendations by combining content-based collaborative filtering approach and ANN based approaches like LSH. The hybrid model performs reasonably well,

Table 1. A table comparing several models on various evaluation metrics

Model	Training time (hours)	Best K	Test MAE	Test RMSE	Coverage
Naïve Model	N/A	N/A	0.763	0.944	N/A%
Tree based ANN*	1.93	20	0.550	0.761	xx%
LSH	1.29	15	0.573	0.796	65.6%
Item-Item CF	4.1	15	0.553	0.759	76%
Content based	0.6 (approx.)	25	0.593	0.803	31.5%
Hybrid	1.89	15	0.583	0.799	46.54%

* indicates evaluation done on the whole dataset and not on the test dataset

	Books Read	Books Recommended
Horror_Score	19.0	5.0
Biography_Score	13.0	2.0
Mystery_Score	6.0	7.0
Romance_Score	4.0	1.0
Fantasy_Score	2.0	1.0
Children_Score	2.0	0.0
History_Score	1.0	3.0
Autobiography_Score	1.0	0.0
Drama_Score	0.0	1.0

Figure 8. Comparison of Genre scores for Books Read vs. Books Recommended

	Recommended Books
0	For Whom the Bell Tolls
1	Open: An Autobiography
2	Night
3	Deeply Odd (Odd Thomas, #6)
4	Unbroken
5	Eye of the Needle
6	Don Quixote
7	The Graveyard Book
8	To Kill a Mockingbird
9	The Lion, the Witch and the Wardrobe (Chronicl...

Figure 9. Recommended books

however its coverage is significantly lower than the LSH model - which is why we recommend the use of the LSH model for making recommendations.

Given the challenges mentioned in earlier sections, we were constrained in our attempts to develop and include models with high computation and space requirements. We would like to to extend our current work to include such models.

In the future, we would also like to extend this study to convert our code into a Python package over pip. Hence, we invite members of the larger academic community to contribute to this project.

Acknowledgments

We would like to thank Prof. Brett Vintch for his guidance and support. We also sincerely wish to thank Amine Rajafillah for his consistent efforts and help through his office hours and accommodating our concerns despite his understandably busy schedule.

References

[1] Shani G;Gunawardana A. *Evaluating Recommendation Systems*. Technical report.

[2] *Amazon review data*. `http://jmcauley.ucsd.edu/data/amazon/`. (Accessed on 12/15/2017).

[3] *API — Goodreads*. `https://www.goodreads.com/api`. (Accessed on 12/15/2017).

[4] *Book-Crossing Dataset*. `http://www2.informatik.uni-freiburg.de/~cziegler/BX/`. (Accessed on 12/15/2017).

[5] Christian Gronroos. *Strategic management and marketing in the service sector in SearchWorks catalog*. `https://searchworks.stanford.edu/view/9984529`. (Accessed on 12/15/2017). 1983.

[6] Aristides Gionis; Piotr Indyk; Rajeev Motwani. "Similarity search in high dimensions via hashing". In: 1999, pp. 518–529.

[7] Badrul Sarwar; George Karypis; Joseph Konstan; John Riedl. *Item-Based Collaborative Filtering Recommendation Algorithms*. 2001.

[8] *Word2Vec*. `https://code.google.com/archive/p/word2vec/`. (Accessed on 12/15/2017).