Book Recommendation System

Mid-Stage Project Report

Viral Shanker
Data Science Department
Stevens Institute of Technology
Hillsborough, NJ
vshanker@stevens.edu

Wei Yang
Data Science Department
Stevens Institute of Technology
Hoboken, NJ
wyang17@stevens.edu

Lei Zhang
School of Business
Stevens Institute of Technology
Hoboken, NJ
lzhang93@stevens.edu

***Abstract*—This project is a an outline for the development of a recommendations system for books, utilizing one of the largest datasets available for book recommendations and user-book interactions: the UCSD book graph dataset. First, to get the user clusters as recommendation features, sequences of reading books for each user are extracted as samples. We convert these sequences into document term of matrix. Then, with dimension reduction methods like T-SNE and PCA, we implemented K-means clustering on the features with shrink dimension. Second, we conducted sentiment analysis with TextBlob and vader tools on the reviews, and combined with TF-IDF matrix, user clusters to predict the ratings of the book by XGBoost and Random Forest algorithms. We think, in some cases, the rating can be of great usefulness for websites when there is no numeric rating but only reviews. Third, with gensim package and sequential book dataset, Cbow and Skipgram model are utilized to get the embedding of each book. We can then make recommendation based on similarity of embedding of the books. Finally, RNNs and CNNs are implemented to figure out the relationships between books.**

All the tasks are implemented in only 400000 sample dataset yet. We will further improve the algorithms, fine-tune the parameters and then try to implement in AWS.

***Index Terms*—sentiment extraction, machine learning, book recommendation**

I. Introduction

Living in modern society, we are surrounded by data in all kinds of forms such as reviews, papers, ads, posts and comments on the internet. past few decades have seen a dramatic increase in the number of people who use Internet. So does the increase of the data online,which make the decision making process and making recommendation quite cumbersome especially in business where informed decisions are required. The recommendation system was introduced in the background to enable business to make informed decision and enable customers to have a great experience.

II. Literature Review

Recommendation system research has become one of the most import focuses in the machine learning field since 1990's. The researches can be classified into three main categories via sentiment analysis,which are collaborative filtering (CF), content based and context based.The most basic and easiest method is Collaborative filtering method. Collaborative filtering(CF) is based on customer's historical data and employ reviews by the customers about an item to predict. The historical data helps make a customer's profile as well as a product profile, which are both used to make the recommendations.Cosine similarity and Pearson correlation similarity are the two main techniques. Unlike Collaboration filtering, the attributes of the products will be the concentration of content based recommendation system. Content based system will make a customer profile based on the reviews as well as the product profile in accordance with the attributes it has and the reviews it received.While the context based recommendation system include contextual data into the system, which enable to abandon the necessity of making customers to fill personal details.

III. Problem Statement

Our goal is to build a book recommendation system for Amazon based on Natural Language Process on using group clustering and LSTM. We built our system based on Goodreads - one of the largest book reviews and rating aggregators in the world. First of all, we cluster users into different clusters based on books that the users read and reviewed. Second, we used extract sentiments from every review content and append with the Term Frequency-Inverse Document Frequency as feature to predict the rating of the book. Third, for every unique user, we generate reading book id sequence for this user based on timestamp. With sequence samples cut into the window size of 10, we will predict the next book the user would probably pick. At last, with training the distributed representation (embedding) for every unique book by skip-gram model, we can recommend the top 10 similar books to the user after a user choose a book. Natural Language Process, Machine Learning algorithm such as Principal Components Analysis, Decision Tree, K-Means Clustering, Long Short-Term Memory, Skip-Gram Model will be employed in the project.

IV. Dataset Description

We will be dealing with 3 primary datasets. First, we have the goodreads-books dataset. This contains descriptive information like title language, author, and publish date. Most importantly, it has the most common "shelves" that the book is a part of. For instance, 10 people have this book on their "fantasy" shelves and so on. It also contains a list of "similar" books. The second dataset, user-book interactions, summarizes every interaction between a user and a book including timestamps including read date, review date, rating date. The final dataset is the review dataset which has time body, user, and book associated with each review.

V. Data Pre-Processing

We deal with 3 main files. All files are huge, for a total of 45 GB of data, so efficient loading and storing techniques are loaded. In the case of each file, we use pandas' "chunksize" argument when reading the files. This allows us to read the data in parts that fit in memory. Each chunk is then processed into a format we use for our algorithms. Then we explore the dataset and plot the book id distribution. To run the model with limited computational resource, we set 95 percent of higher frequent bookid that appear in this dataset to decide the bookid that we consider. We only consider 18 thousand books in our experiment. Note that with sparse optimizations, we were able to use the enitre 6 million book dataset for clustering and SVD.

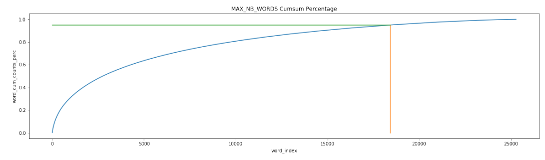


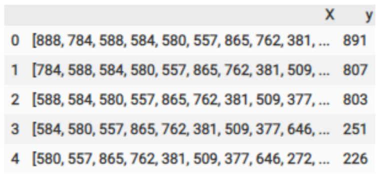
Fig. 1. bookid count distribution from Keras Tokenizer

A. Books Data

Here we have each book and its associated tags and the number of people that assigned that tag. For instance, the row corresponding to Book A may have entries as follows: count: 32, name: "Fiction",count:15, name:"funny" and so on. We turn this data into a BxT matrix where B is the number of books and T is the number of tags. Let's call this matrix D. D_{ij} = The number of people who marked book i with tag j. We take a very NLP approach with this matrix. We treat it as a word frequency approach. Each book is a document and each tag a word. Since TF-IDF does not care for order of words, only their frequency, this is the perfect candidate for doing this.

B. Interactions Data

Similar to Books. We construct a matrix D such that $D_{ij} =$ the rating user i gave to book j. This is a sparse matrix since its dimensions are the number of users times the number of books. This dataset is extremely large. However, some of the interactions had given 0 ratings to some books where the user has not actually read the book. This is most likely an error since otherwise it makes 0 ratings by and far the most often given rating. However, most books in Goodreads have a rating between 3 and 4. So we drop these ratings. Also from a practical point of view it makes sense to count the ratings of people who legitimately read the book since they would be able to give a much more accurate impression of the book, and hence their rating is more trustworthy than that of someone who has not even read the book.



	X	y
0	[888, 784, 588, 584, 580, 557, 865, 762, 381, ...	891
1	[784, 588, 584, 580, 557, 865, 762, 381, 509, ...	807
2	[588, 584, 580, 557, 865, 762, 381, 509, 377, ...	803
3	[584, 580, 557, 865, 762, 381, 509, 377, 646, ...	251
4	[580, 557, 865, 762, 381, 509, 377, 646, 272, ...	226

Fig. 2. Sequence of bookid with 20 window size

C. Reviews

We are concerned primarily with the reviews. While traditionally, in fields like sentiment analysis, a single review would be treated as a document on its own, however, here we want to develop features for each book as a whole. Again, however, we run into issues.

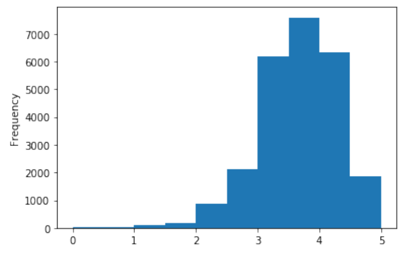


Fig. 3. Rating Distribution

The distribution for the amount of reviews a book has is extremely skewed. With books like *Harry Potter* having, understandably, thousands of reviews, whereas some obscure title may only have two. To avoid significantly skewed reviews, the idea is to get at most 10 reviews for each book. We do not sample randomly, however. We are given the number of votes each review receives, and so we prioritize those. The rationale being that higher rated reviews are more useful and hence contain more useful features. Now we have a dataframe with the book, and each book has at most 10 reviews all concatenated together. From here we can use the sklearn TFIDF vectorizer to get a sparse TFIDF matrix as is traditionally done in NLP problems.

D. A Note on Dimensionality Reduction

All 3 of our matrices are sparse. As such, it is normal to use dimensionality reduction on them for any sort of future processing. Generally we use sklearn's TruncatedSVD since this has out-of-the-box support for sparse matrices. For tags and Reviews we reduce dimeensionality to 100 (from 100000+). We choose 100 because this is the typical size needed for Latent Semantic Analysis, which is what we hope to get from our book and review dataset (as described above). This makes most ML algorithms run much quicker and because of the sparse nature of the initial matrices, such a dramatic reduction does not represent too big of an information loss.

For the future, we will use Word2Vec, or Tensorflow Autoencoders to get better extraction, especially for the review dataset.

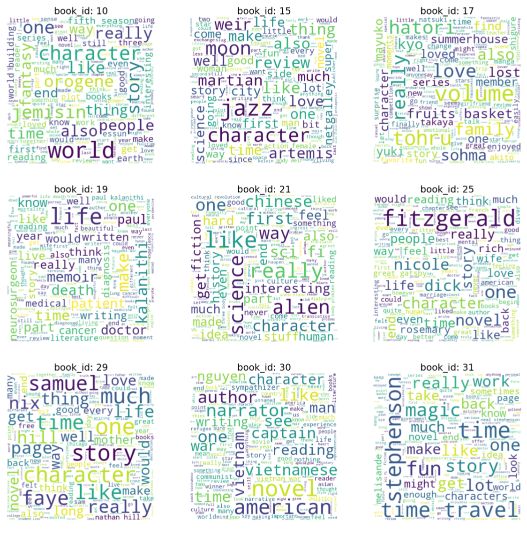


Fig. 4. Word Cloud of reviews

Finally, a final note on interactions. Ratings can actually be zero on Goodreads, however this can cause issues with sparse matrices. So we convert 0 ratings to 0.1.

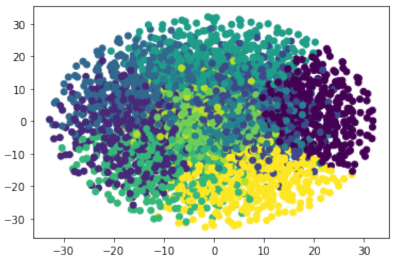


Fig. 5. TSNE dimension reduction

VI. Methods

The first and most naive algorithm we implement for the purposes of the project is Clustering, but with 3 different files, there are several different approaches to take, and each must be verified differently. Please refer to the pre-Processing section for details, but in each instance, we use NLP concepts to guide our data processing.

A. Evaluating Performance: True Similarity

This is one of the metrics we use to evaluate our content based similarities. The ultimate goal for content filtering is to recommend books, given a highly rated book, is to return books that a reader will rate similarly highly. So we can evaluate the veracity of this statement using historical interaction data.

Given a book A and suggested book B and I interaction matrix I when row i and column j tell us the rating user i gave to book j.

True similarity is calculated as follows:

Select users from I that have rated both books A and B and then calculate correlation between the ratings of book A and B in this filtered dataset.

This is quite simple but tells us exactly how similarly readers of book A feel toward book B. This is a basic collaborative filtering metric used to evaluate a content filtering metric. See below the distribution of True Similarity estimated using bootstrap: sampling with replacement.

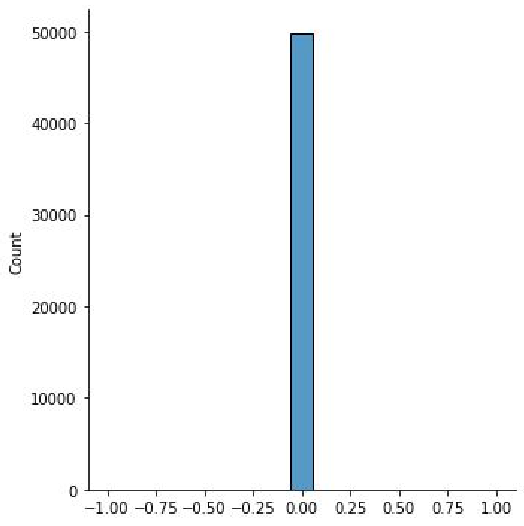


Fig. 6. Random True Sim distribution. The mean was 0.000414

B. Clustering from the Books Data

This is the most straightforward of the lot, but it sets the tone for most of our clustering. We have our TFIDF matrix of tags for each book in sparse form. From here we simply use the standard K-Means to cluster books together, assigning them some group X.

We can evaluate these clusters fairly well, actually. Our dataset comes with a similar books column, which lists a variable amount of books that are similar to a given book. There was not any information on what made it so, but it gave us a decent metric to measure by. We selected the optimal K using the elbow method and found that around 55 percentage of the similar books were in he same cluster as the reference book. Furthermore, most of the time, the ones in a different cluster were all in the same, different cluster.

Note that this metric does have its drawbacks. For instance, picking just two clusters gives us a 70 percent match rate! That is, 70 percent of the similar books show up in the same cluster as the base book, but obviously two clusters is not a very useful measurement. More clusters allow for more granular recommendations. And more clusters clearly do help a lot:

We also tried cosine similarities, but we got very limited success. While trying popular books showed similar books - for instance fantasy books would show other books in the same series, but we didn't come up with any books in the given similar indices. Although in terms of true similarity we

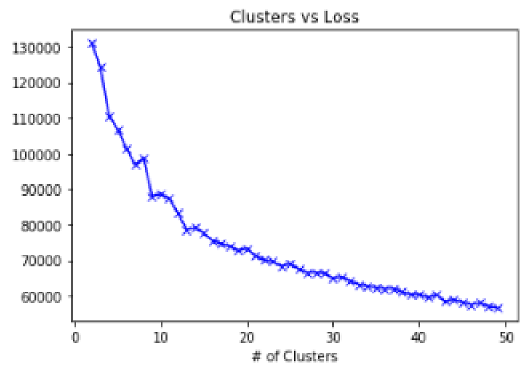


Fig. 7. Loss as a function of increasing clusters. There needs to be some work done here clearly, since upwards of 50 clusters is quite a lot.

did marginally better. Our mean, while still very close to 0, was an order of magnitude better than the random one.

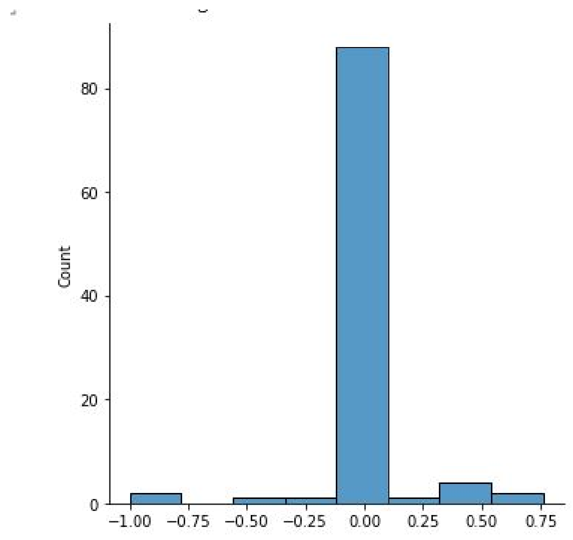


Fig. 8. Tag Based True Sim distribution. The mean was 0.0.007355 which seems poor, but is 50 times better than the random. Still far from ideal, but it speaks to the difficulty of this

C. Interactions Data

With our data, we can cluster using mini batch KMeans just as above. We can evaluate by making copies of known users, and eliminating one or two ratings from it. Then we predict put that in the predict function of our cluster classifier. Once assigned a cluster K, we will compare the average rating of the withheld movie for people in that cluster with the actual rating we withheld.

D. Clustering from Review Data

This is textbook NLP. We have a TF-IDF matrix corresponding to each book. From here, it is just a matter of reducing dimensions using Truncated SVD, to make kmeans quicker and choosing the best cluster! From here, we can evaluate our clusters the same way we did in the books data.

Finally, we also do cosine similarity here, and this works extremely well compared to tags clustering. Once more we use true similarity and we do around 50 times better than the tag clusters and 2500 times improvement over random recommendations!

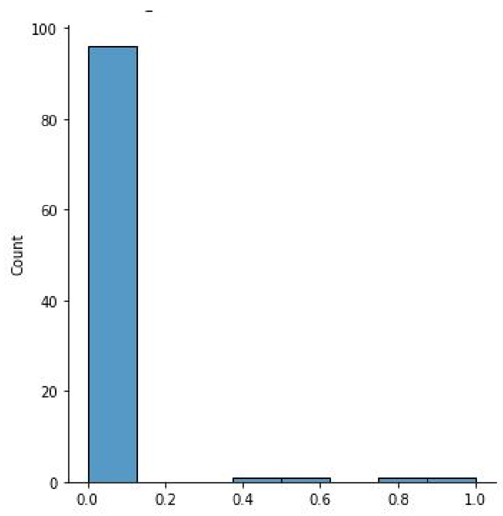


Fig. 9. Review Based True Sim distribution. The mean was 0.029 which seems poor, but is 2500 times better than the random. Still far from ideal, but it speaks to the difficulty of this

E. Supervised Model

As we can see, our evaluation methods definitely leave something to be desired. We can recommend similar books to a given book, but we have no metric of distinguishing between the clusters. Furthermore, we are not leveraging across clusters. We have features extracted from reviews, from book metadata, and from users' histories. From here we don't just have to cluster. For a given user we can put as an input, user history, book review features, and book meta features - all in our reduced form - and train a model on it.

Specifically we will assume we have 3 matrices available (see pre-processing for details).

- Matrix B from Books data. A MxN matrix where M is the number of books and N is the number of unique tags. B_{ij} tells us the tf-idf weight of tag j in book i, which in turn is derived from the number of people who assigned tag j to book i.
- Matrix I from Interactions Data. A MxN matrix where M is the number of users and N is the number of books. I_{ij} tells us the rating user i gave to book j.
- Matrix R from Interactions Data. A MxN matrix where M is the number of books and N is the unique vocab. R_{ij} is the tf-idf weight of word j in document i.

Considering the sparse nature of these matrices, we will reduce the dimensionality of matrices B and R to 100. These are static since they are features of books, which do not change. Now we can construct our data.

For Row i in matrix I, we make a duplicate of Row i with one of the non-zero ratings as zero. We store the book ID and actual rating in a different structure. So each row has N copies where n is the number of books the user has rated. The rows are identical the original one except one rating is zeroes (i.e. withheld).

So now we have a NxM matrix S where N is the total number of interactions and M is the number of books. We essentially have multiple versions of each user. We use PCA or TruncatedSVD on this to get dimensionality reduction. Now for each such user we add the row in R and the row in B associated with the book that we withheld from the user in matrix S. Now we have our training X which has features of the tags, the reviews, and the user's own history. And our y is the rating the user actually gave. The machine will predict the rating the user would give. Naturally we will use training and testing data for evaluating the model (as well as dimensionality reduction for the I matrix).

The end product is a machine that, given a user's history and a book, can predict the rating the user would give to it. Our machine can run a user against every book and suggest X books with the highest predicted ratings.

Unfortunately, in actual implementation, this model is a complete failure. The model tries fitting noise and no amount of tuning actually helps us. We conclude that dimensionality reduction in 3 separate fields make this too difficult to accomplish. Even in actuality a model like this has never been implemented in practice, despite in theory appearing quite sound.

F. *SVD Predictions*

This is our best performing model.

SVD is the state-of-the-art non-deep solution to the recommendation problem. Netflix uses this exact system to make recommendations, and we can leverage this powerful technique to make great recommendations.

The first thing to do is to get our interaction data in the correct format. We are given a dataframe like so:

What we need is a pivot table of this data. A matrix I that MxN, where M is the number of users and N is the number of movies. The ith row and jth column tell us the rating user I gave to movie J.

Naturally, this gives us very sparse data, as the majority of users have rated a fraction of the whole universe of books. And this is where we make an optimization of the standard SVD algorithm, inspired by Simon Funk's Netflix solution. This allows us to use the entirety of the data and process it in a reasonable time. This solution is orders of magnitude faster.

But first, we consider what SVD does. It decomposes a matrix into 3 parts A = UDV. A in our case is our interactions data, U would represent our user space and V represents our movie space. Obviously, a perfect one doesn't exist, so SVD aims to minimize the MSE of this fitting operation.

Note that the right part is a dot product between row i and column of matrices U and V respectively.

With millions of dimensions, this can take quite long. But using our sparse nature, we only consider non-zero dimensions. A simple but effective speedup.

This is a supervised learning algorithm, actually, with training, validation, and testing data. So we split our interaction data BY USER. This is critical, as we do not want part of one user to be in another sample, as this would corrupt our test set. After training, we predict, and our results are excellent. When predicting the ratings of users the machine had never seen before, we get a mean absolute error of 0.89 and an MSE of 1.46. This tells us that the machine can predict what a user will rate a movie within 0.89 points. With this, we can recommend movies quite easily, as the rating user I will give movie J is given by multiplying row I of U by column J of V scaled by the diagonal elements.

We selected the parameters by using cross validation but had to limit the data we did it on to make this practical. We tested various n-factors and regularization terms and settled on reg: 0.005 and factors 15. Though various different combinations did not actually change performance too much.

G. *Skip-Gram and CBOW*

As what we have done in previous part, we transform the books that each user have read into a sequence. It is similar with the text paragraph concept in NLP. For example, If one person read 22 books continuously in a year. Then it will be recorded as the one sample for our model. We set the size of sliding window with 20, which means the twenty first books that the person read is the label of the previous 20 books. So, in this year, we can generate two samples for our deep learning model. Finally, we generate 254 thousand training samples for the partial dataset.

With training the distributed representation (embedding) for bookid by Skip-gram or CBOW model, we recommend the top 10 similar books to the them after they pick up a book. Both models are implemented by Gensim.

Finally, it can give the recommendation based on the similarity of the bookid. We shows ten samples in this method. But for evaluation part, we did not implement a reasonable evaluation method. We might further check the similar book that marked by the book store.

H. *Multi-channel CNN*

With sequence samples cut into the window size of 20, we predict the next bookid that the user would probably pick. The model structures we used are the figures as follows:

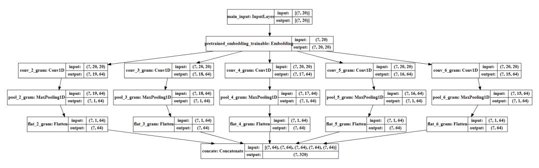


Fig. 10. multichannel CNN

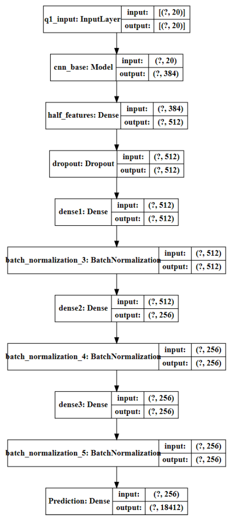


Fig. 11. multichanneltextCNNbasedClassifier

Hyper-parameter setting for Multichannel-CNN: Filter size is [2,3,4,5,6] (Five channels to detect unigram, bigram, trigram... sequence combination); Parameter size is 5,754,288; Embedding size is 20; Batch size is 128; Optimizer is Adam with learning rate 0.001

We use sparse cross entropy loss as our loss function. The Training history plot is shows as follows:

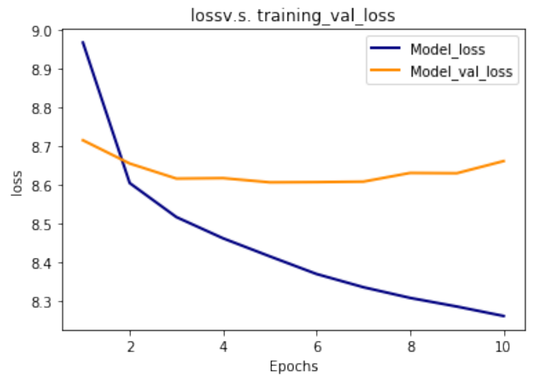


Fig. 12. Training history for CNN 1

For evaluation result, due to large parameter size and computational resource limitation, we only trained our model with 20 epochs. On the evaluation part, we only achieve 4.22 percent of accuracy. One reason is due to the large prediction target, since we consider 31406 prediction bookid. For this part, we should filter more unpopular books, which are not that useful for recommendation. Most of user are not interested with such books. Also, we may need to change the evaluation method. we can make our prediction target as the clusters from the task 1 and 2. Then, model can make recommendation for specific clusters that user may love.

I. RNNs

We also implemented the RNNs model to make the comparsion with the textCNN. Due to limited computational resource, we only stack two LSTM unit and a dense layer to make the prediction. We use sparse cross entropy loss as our loss function. The same, we use 20 embedding size. we use 20 unit in the hidden layer. But the model can perform better than multichannel-CNN in such case. However, it is hard to say which model structure is better, since more fine tuning works should be done there.

The RNN model structure shows as follows:

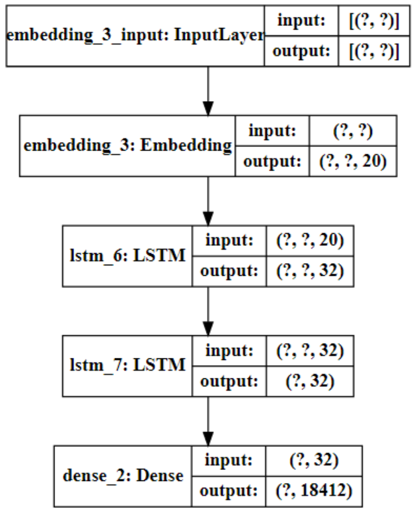


Fig. 13. LSTM model structure

We only train our model with 20 epochs.

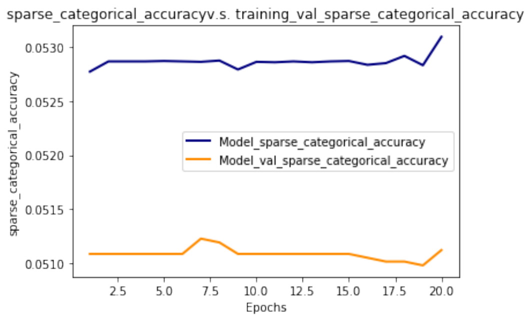


Fig. 14. Training history for RNN

VII. Comparison and Future Directions

In the end we had quite a lot of failed models. The best model by far was the SVD model. Able to predict a rating within 0.9 points is very impressive considering that the ratings themselves are at 1 point intervals (4.5 is not possible rating). The state of the art rating for similar problems (Movie recommendation) is 0.82. But goodreads is far more expansive in terms of users and number of books. There are many, many books.

The sheer number of books is what brought down our content based methods. Though our evaluation metric of True Similarity showed huge improvement over random, it was still very close to 0 when it came to our ratings. Furthermore, cosine similarity was very slow but only constrained by disk constraints. With a lot more disk, we could make a lookup table, showing distances from every to every book, but with 6 million books, such a task would take terabytes of data. For the future, of course, we would like to improve our deep learning models, which were not able to converge well and be tuned well. Autoencoders could even be used to improve upon the SVD model.

VIII. Conclusion

Machine Learning along with Natural Language Process is a scientific way to achieve precision and efficiency when it comes to products recommendation system. In this project, we have implemented both supervised learning and unsupervised learning including Decision Tree, PCA,Clustering and we are going to implement LSTM and Skip-Gram.Statistical Packages including Scikit-learn, Pandas, Matplotlib were employed in the analysis.

In the end, our SVD model was able to give excellent predictions and could definitely be put in the field to make actual, real recs for books.

IX. Reference

Mengting Wan, Rishabh Misra, Ndapa Nakashole, Julian McAuley, ”Fine-Grained Spoiler Detection from Large-Scale Review Corpora”, in ACL’19.

Grbovic, Mihajlo, and Haibin Cheng. ”Real-time personalization using embeddings for search ranking at airbnb.” Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining. 2018.