**Name and NetIDs:**

**Hitesh Patel,** [**hlp276@nyu.edu**](mailto:hlp276@nyu.edu)

**Sumedha Rai,** [**sr5387@nyu.edu**](mailto:sr5387@nyu.edu)

**Section 1: Overview**

A recommender system is one of the most successful and widespread applications of machine learning used heavily by companies like Amazon, Netflix, YouTube, and the like. For our project, we built a book recommendation system using Sparks’ alternating least squares (ALS) method to learn latent factor representations for users and items and use those to make book recommendations for users.

We use a large dataset from the book review website Goodreads. The ‘interactions’ dataset has roughly 230 million user-book interactions and the columns encapsulate information on user ID (user\_id), book ID (book\_id), if a book has been read by a user (is\_read), if a book has been reviewed by the user (is\_reviewed) and the final rating given to the book by the user (rating). The ‘is\_read’ column serves as a flag for marking the ‘read status’ of a book. It takes the value 1 if the book has been read and 0 otherwise. The ‘is\_reviewed’ column follows the same format (1 if the book has been reviewed and 0 otherwise). The ratings column takes values in the range of 0-5, with 5 being the highest rating is assigned to a book. For the purpose of this project, we have taken a subsample of the ‘interactions’ dataset. The subsampling process has been described in Section 2 of this paper.

Table 1 illustrates the distribution of the ratings of the entire dataset and the subsampled dataset (compare the following statistics for both datasets)

Brief Statistics

* Distribution of ratings ( + mean, median, max, min)
* Number of distinct users and books
* How many is\_read (0,1)

The recommendation system built as part of this project ensures that we capture and continuously calibrate the reading preferences of the users by learning from the extant data available to us. This enables the website to make better recommendations to its user and increase user interaction on the platform.

**Section 2: Data Processing**

The reader-book interactions dataset consists of ~228.6 million ratings. We removed all the rows with a rating of zero and sampled this dataset to contain only 21.9 million ratings. This constitutes 2.5% of the dataset with no zero ratings. This was done because of resource constraints. We split this sampled dataset into train, validation and test sets such that the training set consists of all the users while the test and validation set each consist of 20% of the users i.e. 60% of the users in the training set have all their interactions and the rest 40% have half of their interactions in the test and validation sets. We ensure that all the books in the validation and the test set are also present in the training set. These datasets were stored in parquet format to be used later. We also fixed a seed for all the random operations so that the results are reproducible.

**Section 3: Model and Experiments**

**Fitting ALS(Alternating Least Squares) model**

ALS is a matrix factorization technique that decomposes a matrix X into 2 matrices, U and V in a parallel fashion. It alternates between optimizing the loss function for both U and V i.e. it optimizes U by treating V as a constant and then optimizes V by assuming U as a constant. For building recommendation engines, one of the algorithms used is matrix factorization. The user-item interaction matrix consists of ratings given by a set of users to a set of items and the goal is to predict ratings given by a user to the items that the user has not interacted with yet. The loss function that ALS optimizes is:

J(U,V)=i,jci,j(Xi,j-UiVj)2+(i||Ui||2+j||Vj||2)

is the regularization parameter and U,V are the latent matrices for users and items respectively.

We used pyspark.ml.recommendation library to fit an ALS model to our dataset. The hyperparameters that we  experimented with are:

1. Rank: 10, 15, 25
2. :10-3,10-2,10-1

These 9 combinations were tested on the validation set and the models were compared using MAP (mean average estimate) estimated on 500 items predicted for each user. The business application behind building a recommendation engine is to predict items to the user. A good recommendation engine is one that predicts items which the users will like/click on. Thus, to assess the performance of any recommendation system, we need to calculate its precision because that captures how many of our predictions made by our model are good. We calculate the precision for the top 500 predicted items predicted for each user. We choose MAP and not precision@k because MAP also takes into account the ranking among the predicted items whereas precision@k just cares about if an item is present in the predicted set or not.

Using the pyspark.ml.recommendation library, we can directly get 500 tuples predicted items for each user along with their corresponding predicted rating. We extract the 500 items from these tuples and put them in a new column using pyspark.sql.functions.explode. Windows function allows us to operate on a group of rows while still returning a single value for each input row. We use this window by partitioning over the user\_id column. We use the pyspark.sql.functions.collect\_list function to combine all predicted books for a user in a single list and make a column out of it. We do the same for the actual books read by the user. Finally, the dataframe with user\_ids, predicted books and actual books column is fed to pyspark.mllib.evaluation.RankingMetrics function as a RDD which calculates MAP of the model.