Recommendation Systems: Collaborative Filtering using Matrix Factorization

A Presentation by Group 13

Group Members

Name	ASU ID	Email-ID
Visaj Nirav Shah	1225369210	vshah47@asu.edu
Darsh Vaidya	1225514966	dvaidya5@asu.edu
Malvi Mungalpara	1225184103	mmungalp@asu.edu
Meet Kishorbhai Lakhani	1225636854	mlakhan3@asu.edu
Vibavari Gurunathan	1225412305	vnolas25@asu.edu
Rohit Rajesh Rughwani	1223264705	rrughwan@asu.edu

Recommendation Systems

- Endless influx of information and products

- Enter Recommendation Systems
  - Machine Learning-based systems that provide customized suggestions to users by analyzing their interests and mapping it to a large library of content

- Helps discover content/products that a user might be interested in but aware of

Recommendation Systems

- Content Filtering
  - Profile-based
  - Attributes of users (i.e., their interests) and attributes of products (e.g., genre, actors, category) are used
  - Difficult to collect all the information

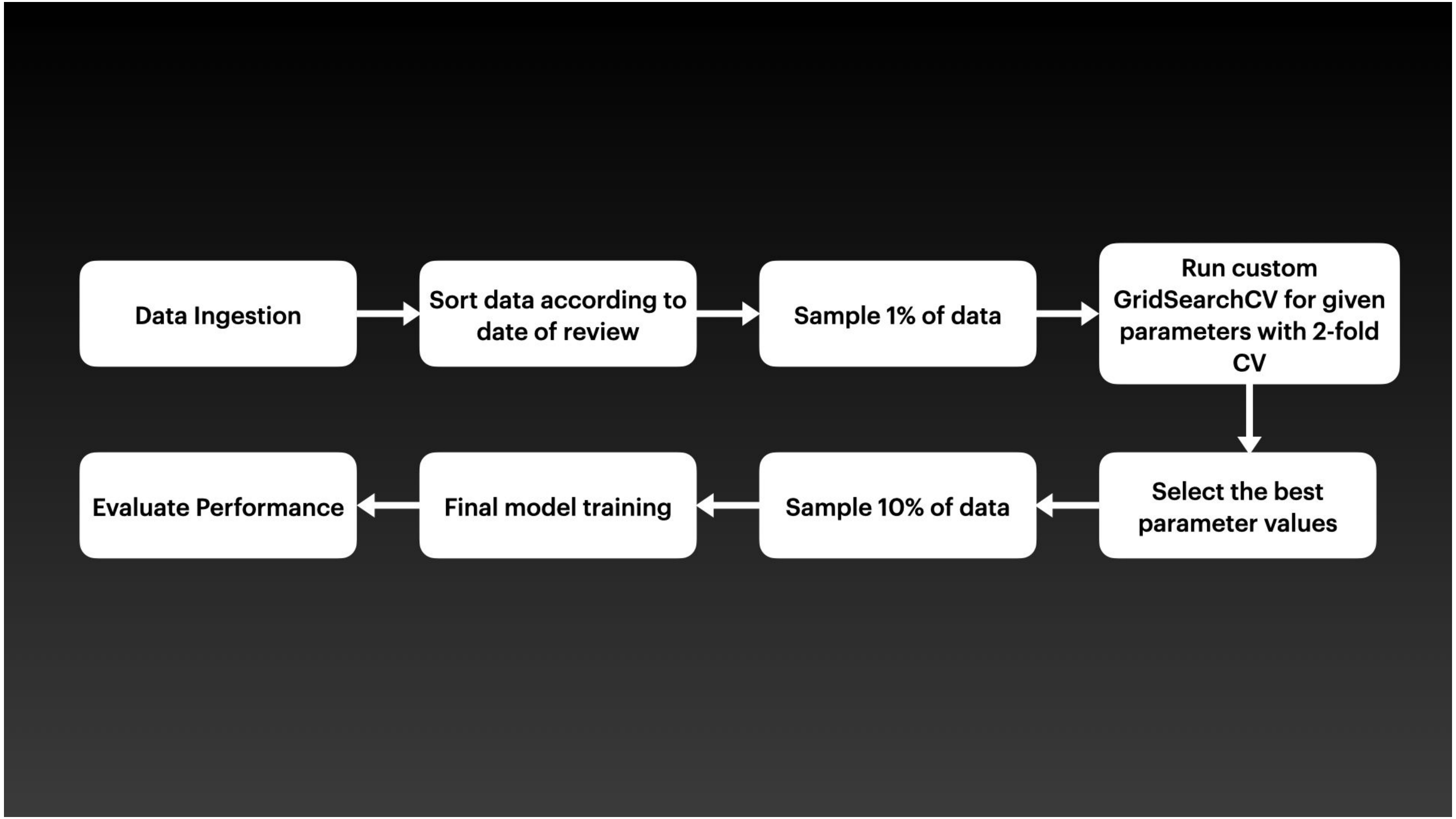
- Collaborative Filtering
  - Relies of past user behavior - no explicit profiling
  - Relationship between users and different product items is used
  - Neighborhood methods and Latent Factor methods

Problem Description

- Build a collaborative filtering-based recommendation system using matrix factorization method that can detect latent factors, and utilize them for improving recommendations

- Why collaborative filtering?
  - Independent of domain

Methodology



Flowchart enlisting the steps followed

Data Ingestion: About Dataset

- Netflix Prize Competition dataset released by Netflix in 2006, currently available on Kaggle

- 100,000,000+ movie reviews (i.e., rows) in total
  - For hyperparameter tuning, we use 1% of the dataset
  - For final model training, we use 10% of the dataset

- 17,770 movies and 480,189 users

- Ratings $\in \{1, 2, 3, 4, 5\}$

Data Aggregation

Features
UserID
MovieID
Ratings
Date
Year

- Making data usable
  - Flatten the hierarchical structure
  - Create custom features based on existing features
    - Extract Year from Date
  - Rearrange the columns

- Data is split across 4 .txt files, so merge all data in a single .csv file

- Sort the rows based on Date of review

The Math

- For each product *i*, we create a vector (*q*) that contains its rating by each of the users *j*. Similarly, we create vectors (*p*) for each users.

- The matrix multiplication $q^T p$ results in the ratings matrix, *r*

- Intuition behind the method: Single Value Decomposition
    - Think about splitting a large number into two prime factors

- The missing values can be imputed, but recent models use regularization to solve the issue

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$$

The Model

- PyTorch implementation of an Embedding model
- Criterion is Mean Squared Error
- Optimized using Stochastic Gradient Descent
- Hyperparameter search space for GridSearchCV:

```
parameters = {
        'batch size': [1024, 2048, 4096, 8192],
        'epoch': [10, 50, 100],
        'lr': [0.01, 0.1]
}
```

Custom GridSearchCV

- GridSearchCV is a standard technique to learn the best hyperparameters for the model
- Hyperparameters to be checked: Batch Size, # epochs, Learning Rate
- Cross validation for temporal data
    - Cannot use regular cross validation - need to create rolling folds
    - Forming folds based on Year
    - 2-fold cross validation

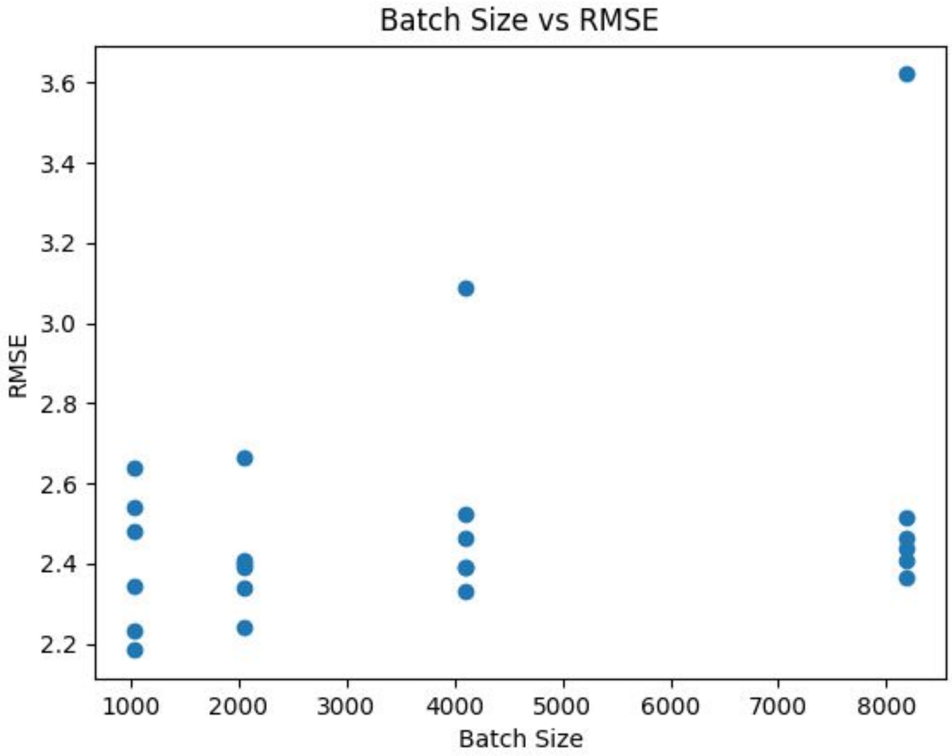
Results and Analysis

Root Mean Square Error (RMSE)

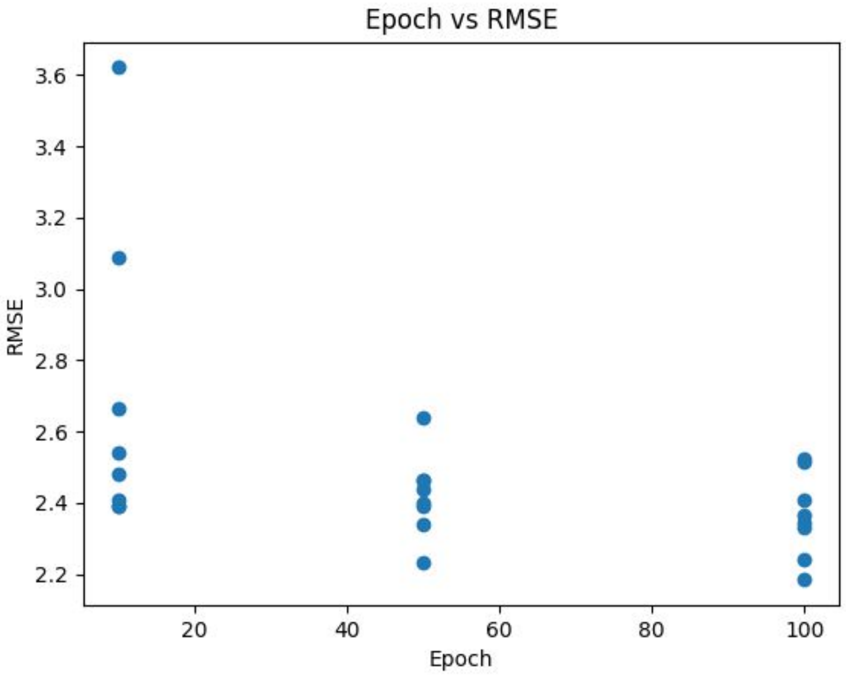
- RMSE: Metric for evaluating model performance

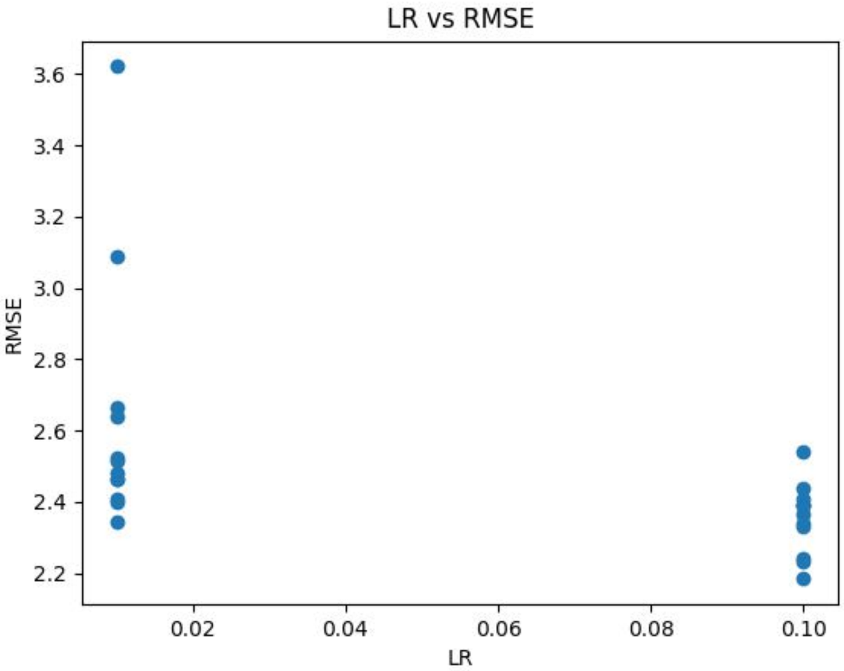
Netflix's RMSE	0.9514
Required RMSE for Grand Prize	0.8563
Authors' RMSE	0.8614
Our RMSE	**1.3339**

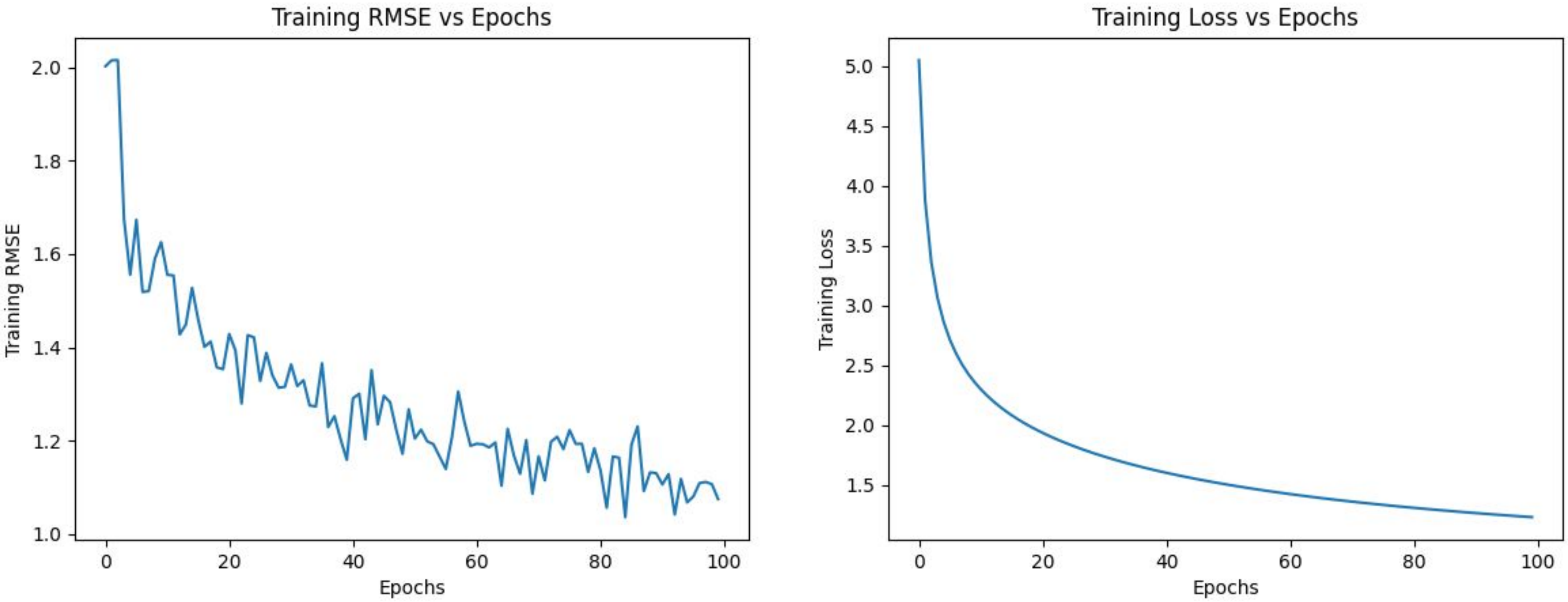
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$



Variation of RMSE vs. Hyperparameters


Epoch vs RMSE
RMSE
Epoch
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.2
20
40
60
80
100


LR vs RMSE
RMSE
LR
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.2
0.02
0.04
0.06
0.08
0.10



Variation of RMSE and Loss vs. Epochs in the final model

Conclusion and Future Works

Conclusion

- Matrix factorization is an efficient method for building a recommendation system, especially for such large datasets.
  - Easy scalability
  - Sparsity of data is not a problem

- RMSE as a metric is a good judge of model performance

- Recommendation systems create immense value for a variety of businesses because of their satisfactory predictions

Future Works

- Parallelize code
  - Our custom GridSearchCV function is slow
  - Creating workers which run in parallel can increase performance
- Using GPUs for working with a larger dataset
  - Currently, we are only using 10% of the dataset
    - Limitations of personal computers
  - With GPUs, we can train our model on the full dataset which will further reduce the RMSE
- Integrate metadata in matrix factorization
  - More information, better recommendations
- Explore using reinforcement learning

Thank you