Final Project Report:

"Guess U Like"

Xiangyun Chu (xc1511)

Elizabeth Combs (eac721)

Amber Wang (yw2115)

New York University

DSGA-1004: Big Data

Professor Brian McFee

May 11, 2020

DSGA 1004
Final Project Report
Professor Brian McFee

Overview

In the 21st century, many brick and mortar stores with limited inventory have been replaced by digital retailers with seemingly endless products and user-bases. Where physical retailers could only store popular items based on average user preferences, digital platforms have built recommender systems, which have become the primary way to interact with these large collections. The Goodreads dataset is one of these large datasets, containing over 800k users with 2.3M possible books to recommend. On average, users are associated with ~260 books, making selecting the next recommended book out of the remaining inventory a daunting task. This project seeks to build and evaluate a recommender system using explicit user feedback, in the form of a rating from zero to five, to recommend up to 500 books to users based on their preferences.

Data Processing

With such a large file, data preparation needed to be performed to build the baseline recommendation model. First, the full, raw, csv-format dataset of interactions was read into memory. It was subsampled to 1% of users based on their user id. Since it can be difficult to recommend to users without a robust history of interactions, users with fewer than 10 interactions were removed. Then, this subset of data was sorted by user id and converted and saved to parquet format for improved storage. Later, we also down-sampled 5% and 25% of the full dataset using the same methodology.

Next, the data was split into training, validation, and test (60/20/20) sets with a user-based approach. First, the dataset was split by user id into 60% (training) and 40% (test-validation) sets. For each user in the test-validation set, half of their interactions were portioned off into a true test-validation set, while the other half were put back into the training set. This way, regardless of splitting, the training set still contained all possible users so that we could make predictions on the users in the test and validation sets for evaluation purposes. Finally, the remaining interactions in the test-validation set were split in half by the user into separate test and validation sets.

The last step of processing was to remove items that were contained in the test and/or validation sets, but not present in the training set for the baseline model; later, we implemented a cold-start to account for these items. Finally, we dropped unnecessary columns, leaving user id, book id, and rating to build the ALS model.

Model and Experiments:

The model method selected for this project was Spark's Alternating Least Squares (ALS) , which learns latent factor representations for users and items simultaneously (*Pyspark.ml Package*). This model has a few important hyper-parameters to tune which helps to optimize performance on the validation set: maxIter, rank, and regParam. MaxIter controls the number of iterations; rank represents the number of dimensions of latent factors; and regParam is the regularization parameter. These parameters help control the complexity of the model to find the right balance between under- and over-fitting.

In the model-tuning process, we explored the impact of all three hyper-parameters on the performance. RMSE and three ranking metrics computed on top 500 recommendations were implemented to evaluate the relevance of our recommendations. We focused on Mean Average Precision (MAP) in choosing the optimal model because it accounts for both the recommendation accuracy and the order of the relevant items (*Evaluation Metrics - Spark 2.4.5 Documentation*). We derived the recommendation's ranking and true ranking from the data set then outputted the MAP score using the built-in ranking metrics function, which computes the MAP over all validation users.

On 1% data, we are able to train the model with multiple ranks and regularizations to explore their effect on all evaluation metrics. The table below shows the results of hyperparameter tuning on the validation set:

1% Data maxIter = 10	regParam= 0.005	regParam = 0.01	regParam = 0.1	regParam = 1

Rank = 5	RMSE = 2.33415
	MAP = 1.103 e-05
	Precision = 0.00021
	NDCG = 0.00064
		RMSE = 2.22377
		MAP = 1.071 e-05
		Precision = 0.00021
		NDCG = 0.00058
			RMSE = 1.94205
			MAP = 1.085 e-05
			Precision = 0.00015
			NDCG = 0.00052
				RMSE = 2.0258
				MAP = 3.088 e-05
				Precision = 0.00012
				NDCG = 0.00057

Rank = 10	RMSE = 2.38184
	MAP = 3.347 e-05
	Precision = 0.00055
	NDCG = 0.00139
		RMSE = 2.26834
		MAP = 2.118 e-05
		Precision = 0.00057
		NDCG = 0.00132
			RMSE = 1.92439
			MAP = 2.529 e-05
			Precision = 0.00051
			NDCG = 0.00136
				RMSE = 2.02567
				MAP = 3.62 e-05
				Precision = 0.00026
				NDCG = 0.00088

Rank = 15	RMSE = 2.42475
	MAP = 4.709 e-05
	Precision = 0.00121
	NDCG = 0.00262
		RMSE = 2.30419
		MAP = 5.335 e-05
		Precision = 0.00121
		NDCG = 0.00275
			RMSE = 1.91677
			MAP = 6.873 e-05
			Precision = 0.00119
			NDCG = 0.00332
				RMSE = 2.02550
				MAP = 5.889 e-05
				Precision = 0.00028
				NDCG = 0.00104

Based on the results above, we observed that large rank leads to the biggest improvement in ranking metrics, while different settings of regParam had an unstable impact on performance. Further experiments indicated that larger maxIter also increases the performance, but the benefit is less obvious compared to rank. Due to limited computation capacity, we focused on rank in tuning the model, and we were able to get our highest validation MAP of 0.0096 on 1% with Rank=200, regParam=0.015, maxIter=10. We also used these metrics when training the model on larger subsamples of the dataset. Considering the limited capacity of the Dumbo cluster, we chose a single, relatively high rank for training and limited our maximum training set to a 25% subsample of the data. With the same hyper-parameter setting, the values of all ranking metrics significantly decreased as the size of the data increased. This implies that more complex models may be needed to achieve high expression power on larger user groups.

	1% data	5% data	25% data

Rank = 30
regParam = 0.015
maxIter = 10
	MAP = 0.00038
	Precision = 0.00335
	NDCG = 0.00933
		MAP = 3.927 e-05
		Precision = 0.00063
		NDCG = 0.00174
			MAP = 1.184 e-05
			Precision = 0.00011
			NDCG = 0.00038

While we were not successful in training the model using the full dataset on the Dumbo cluster due to limited capacity, we were able to find optimal models through raising the rank for each data subset and measured performance on their corresponding test set:

	1% Data	5% Data	25% Data

Best Setting on Training and Validation Sets	Rank = 200
	regParam = 0.015
	maxIter = 10
		Rank = 50
		regParam = 0.015
		maxIter = 10
			Rank = 30
			regParam = 0.015
			maxIter = 10

Evaluation on Test Set	RMSE = 1.90764
	MAP = 0.00836
		RMSE = 2.04944
		MAP = 0.00027
			RMSE = 1.86474
			MAP = N/A*

*MAP metric unavailable for 25% data due to Dumbo cluster congestion.

Extensions:

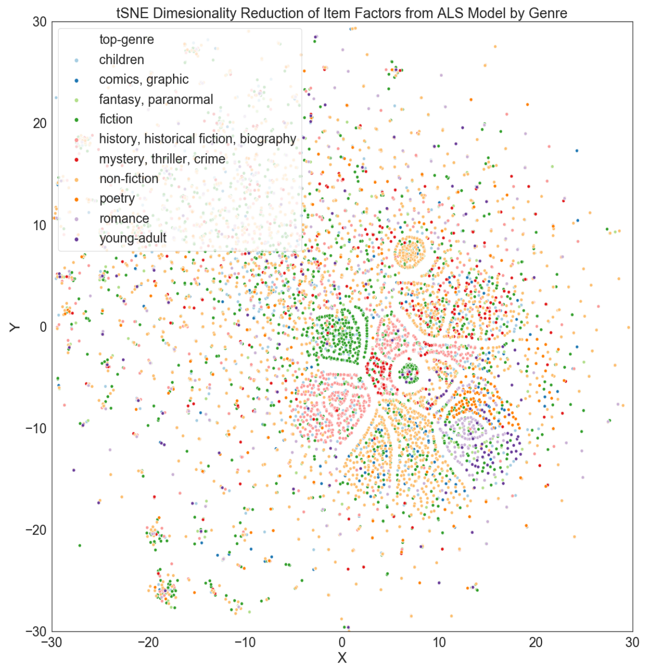
Cold-start: The first extension chosen was the cold-start implementation. The process can be summarized in four phases: 1. Built an attribute matrix for all books; 2. Extracted the latent factor matrix from our model; 3. Built an attribute to latent factor mapping function; 4. Compared performance of cold-start prediction to a full collaborative filter model.

Phase 1: We used the content-based metadata from three supplement book datasets to build an I*N attribute matrix, where I is the total number of books and N is the number of features. The genres were one-hot encoded, where 1 indicated a positive count each genre-shelf, and 0 otherwise. The author ID was used to retrieve the average author rating as another feature. In total, there are N = 11 features. *Phase 2:* We extracted an I*K latent factor matrix from our model trained on observed training data, where K is the rank, number of latent factors, of the ALS model. *Phase 3:* We used the combination of k-means and k-nearest-neighbors (kNN) to map from a cold-start book's content-based attribute to predicted latent factor features. Inspired by TA Jack Zhu, given the large book dataset, we first ran through a k-means model (k=100) based on attribute features to divide the books into 100 clusters. This step saved a lot of time and memory for later kNN calculation. Then, we used the "weighted kNN regression" method proposed by *Gantner et al. (2010)* to map the attribute vector (R^N) to latent factor vector (R^k). The mapping function is as follows: $\widehat{f_i} = (\sum_{j \in N_{k(i)}} sim(a_i, a_j) * f_j)) \,/\, (\sum_{j \in N_{k(i)}} sim(a_i, a_j))$.

The cosine similarity was used to determine the weight for each neighbor. *Phase 4:* The k-NN based cold-start mapping needs the full dataset to guarantee that the nearest neighbors' latent factors are found in the trained model. An alternative way would be to use the average features of the cluster as the prediction and drop the cold-start books whose neighbors in the same cluster are all not trained in the subsampled data. We held out 1% of the items during the training process to simulate a cold-start scenario. The cold-start model trained on 1% data with Rank=200, regParam=0.015, maxIter=10 achieved RMSE of 2.2074, which was very close to the baseline. The error rate would increase if we held out a larger fraction of data, but the accuracy of k-NN based cold-start predictions would also increase as we trained the model on more data. Furthermore, if the cold-start items were introduced and we received user feedback, the system would correct using new latent factors.

Exploration: The second extension was to explore the intuitive meaning behind the item factors through dimension reduction using t-distributed stochastic neighbor embedding (tSNE) and visualization. To begin, we extracted learned item factors, and subsampled the data to 10k books, allowing for better visualization. Using the supplemental data from cold-start, the top genre by count was extracted for each book and joined to the item factors. This matrix was exported from Spark to Hadoop to a single csv file using coalesce. Once the file was retrieved from Hadoop, tSNE dimensionality reduction and visualization were completed using python. Additional finetuning of important tSNE parameters such as perplexity and number iterations was also explored locally using methodology suggested by Oskolkov (2019).

The figure maps the tSNE components generated using the item factors from the best-performing one-percent, subsampled model with rank=200. While genre does not perfectly explain the groupings within the graph, the patterns do indicate that genres tend to cluster together. The lack of a distinct definition between the genre clusters could be explained by the genre extraction method, which does not account for book ids that map to more than one genre. There is likely additional information beyond genre incorporated to the latent factors. One interesting way this manifests in this plot is that poetry and romance genres fall closely together. It is easy to see why advanced recommender systems can recommend more specific categories that seem built just for you: they are exploiting much more granular information hidden in the latent factors beyond just a single genre.



<u>**Contributions of Team Members:**</u>

Each team member contributed to the research, design, and execution of the project. Team members met together to consult and review each piece of the project together. The contributions below reflect leadership in particular topics:

- Xiangyun Chu: recsys; evaluation; hyper-parameter tuning; data splitting check
- Elizabeth Combs: data preparation/splitting; baseline recys; exploration extension
- Amber Wang: data splitting check; evaluation baseline; cold-start extension

<u>**References:**</u>

"Evaluation Metrics - RDD-Based API." *Evaluation Metrics - RDD-Based API - Spark 2.4.5 Documentation*, spark.apache.org/docs/latest/mllib-evaluation-metrics.html#ranking-systems.

Gantner, Zeno, et al. "Learning Attribute-to-Feature Mappings for Cold-Start Recommendations." *2010 IEEE International Conference on Data Mining*, 2010, doi:10.1109/icdm.2010.129.

Oskolkov, Nikolay. "How to Tune Hyperparameters of TSNE." *Medium*, Towards Data Science, 19 July 2019, towardsdatascience.com/how-to-tune-hyperparameters-of-tsne-7c0596a18868.

"Pyspark.ml Package." *Pyspark.ml Package - PySpark 2.4.5 Documentation*, spark.apache.org/docs/latest/api/python/pyspark.ml.html#module-pyspark.ml.recommendation.

<u>**Data Sources & Software:**</u>

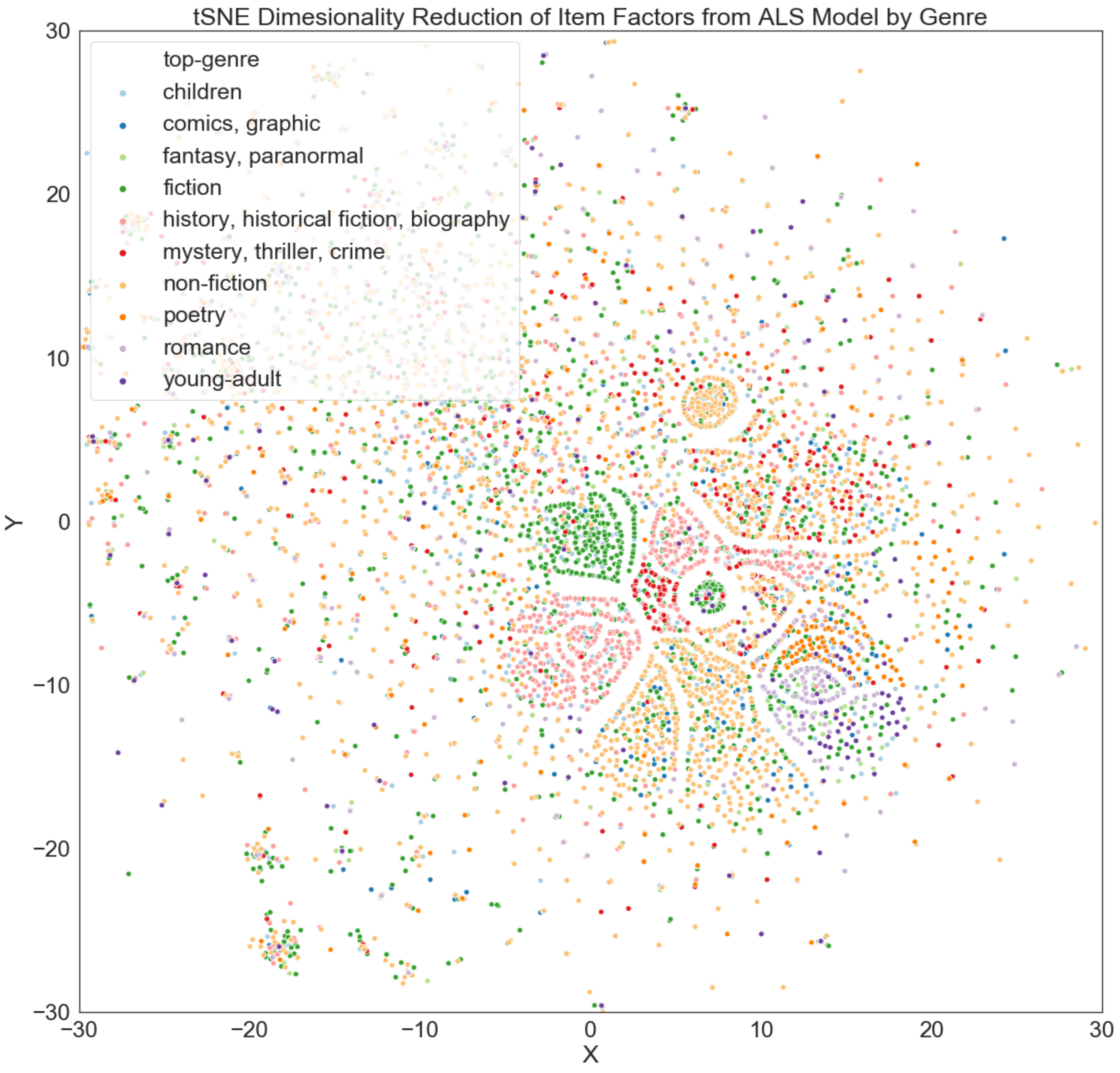
- Goodreads dataset is available online, but the version used in this project was extracted from `hdfs:/user/bm106/pub/goodreads`.
- Supplemental data was collected from this link.
- Software used includes: NYU's Dumbo cluster, Hadoop, pyspark, python.
- All code developed for this project is available on GitHub.

<u>**Code Overview:**</u>

- `data_prep.py:` data preparation, subsampling, and splitting
- `recsys.py:` modeling, hyperparameter tuning
- `evaluation.py:` evaluation metrics
- `cold_start.py:` supplemental data treatment, cold start implementation
- `viz/tsne_prep.py & viz.py:` supplemental data treatment, visualization

DSGA 1004
Final Project Report
Professor Brian McFee

Appendix:


tSNE Dimesionality Reduction of Item Factors from ALS Model by Genre
top-genre
children
comics, graphic
fantasy, paranormal
fiction
history, historical fiction, biography
mystery, thriller, crime
non-fiction
poetry
romance
young-adult
Y
X