EECE5698 - Final Report

Chenyang Liu and Jason Booth

Abstract

The purpose of this project is to create a recommender system to present users with suggestions of styles and brands of beer based on their preferences. To accomplish this, a recommender system was trained using collaborative filtering on a dataset of user rankings for several thousand types of beer. Vector representations of users and beers were extracted and used to create recommendations based on vector similarity. Our model was trained and shown to perform marginally better than a popularity based model, showing that we were able to pick up patterns in the data and were effective at creating recommendations.

Background

Recommender systems are a large field of research for advertising and products like Netflix, Spotify, and Amazon. Collaborative filtering is a type of recommender system that is used to predict user's ratings for items based on their previous rating data. This is done by training vector representations of users and items and using a dot product between the user and item vectors to predict a rating. Once user and item matrices are trained, the vectors can be used to calculate ratings on items that they have never rated before, and suggest items that they may like based on their preferences.

There are multiple methods for training matrix representations. Matrix factorization uses gradient descent on each vector to iteratively update both the user and item matrices at the same time. This process can take a large amount of time to optimize and is not guaranteed to converge as the problem is non-convex.

Another approach to training is called Alternating Least Squares. Instead of training both matrices at the same time it will hold one matrix constant while optimizing the parameters of the other matrix. This causes the cost function for Matrix Factorization in Equation 1 to be broken down into a single cost function for each matrix in Equations 2 and 3. When one matrix is held constant, the optimization problem turns into a

regularized version of ordinary least squares. This problem is convex and can be solved in closed form. The matrices are held constant in an alternating fashion until convergence is reached. Because of the closed form approximation used at each step, ALS has been shown to have better performance than Matrix Factorization on similar datasets.

$$Cost(U^k, V^k) = \sum_{i,j}(u_i^T v_j - r_{ij})^2 + \lambda \sum_i ||u_i^k||^2 + \lambda \sum_j ||v_j^k||^2$$

Equation 1: Cost function for Matrix Factorization

$$Cost(V^k) = \sum_{i,j}(u_i^T v_j - r_{ij})^2 + \lambda \sum_j ||v_j^k||^2$$

Equation 2: Cost function for ALS, V matrix

$$Cost(U^k) = \sum_{i,j}(u_i^T v_j - r_{ij})^2 + \lambda \sum_i ||u_i^k||^2$$

Equation 3: Cost function for ALS, U matrix

There is a built-in ALS function in the ml library for apache spark that uses spark Dataframes. Spark Dataframes are an extension to RDDs that include types for columns and more SQL-like functionality. Studies have shown that Dataframes have better performance than RDD's because of type awareness. Spark currently has two libraries for machine learning called MLLib and ML. MLLib uses RDD's while the ml library is based on Dataframes. MLLib will be deprecated in the future and Dataframes are becoming the focus of future development in Apache Spark, which motivated the choice for exploring Dataframes in this project.[1]

Dataset

The dataset for this project was taken from data.world as a suggested dataset for recommender systems. It includes 1,586,614 ratings for 33,387 users on 56,856 beers. The data includes ratings for several attributes including appearance, aroma, taste, and review overall. The data deals with beers and beer styles it is familiar to a college audience.

The dataset is not evenly distributed for beers and users. In fact, the number of users that have only a single rating is 10,000, while the top 100 users have created over 225,000 ratings. The graphs below show the average number of ratings per user and per beer.

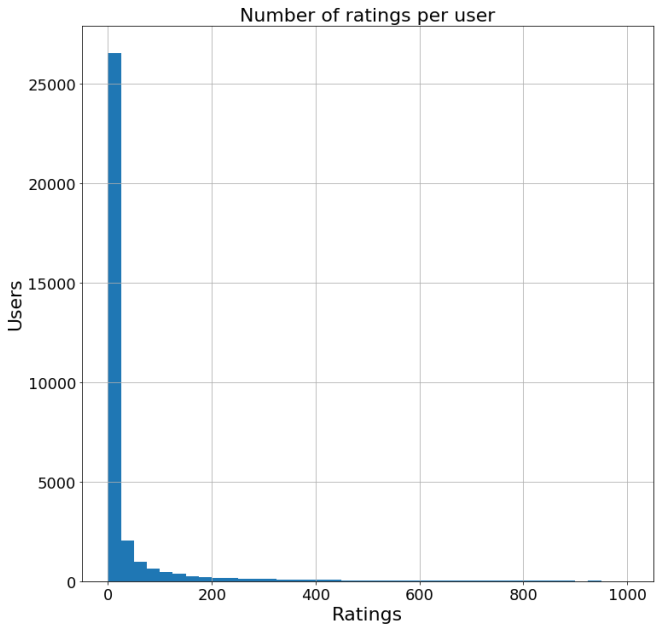


Figure 1: Number of Ratings per User

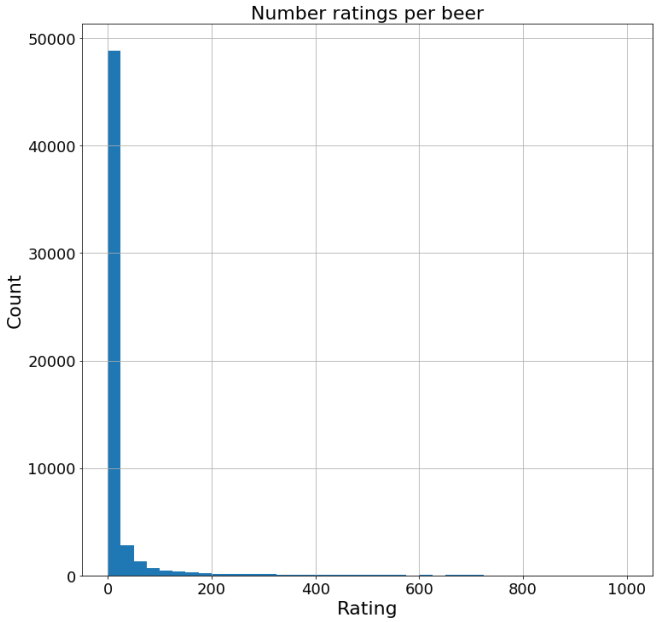


Figure 2: Number of Ratings per Beer

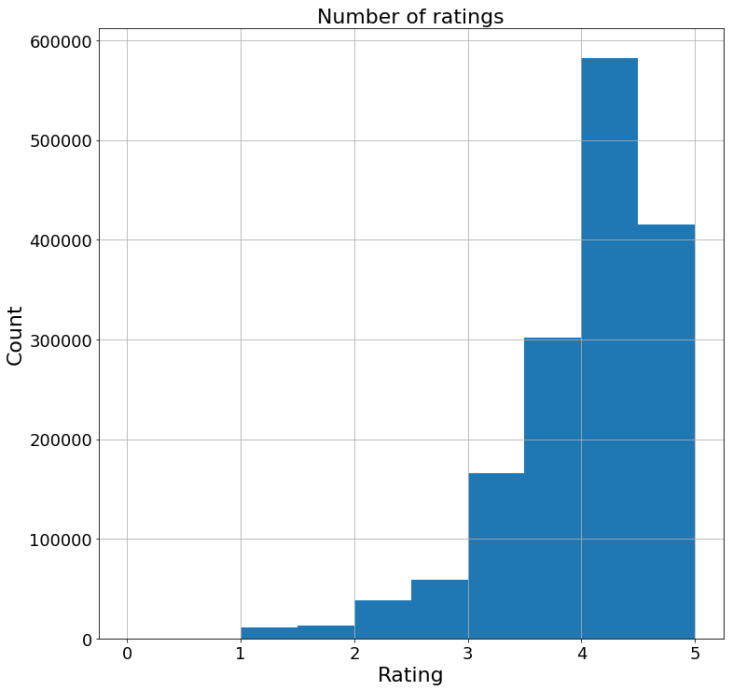


Figure 3: Number of Ratings

The sparsity of these distributions is important in the methods used to train the model and is discussed further in the Methods section. There is also skew in terms of the rating values, with most ratings being around 4 and 5. This means that a model that consistently predicts a single number may end up performing quite well. A further discussion of a comparison of our model to simpler models is included below.

Methods

To develop the beer recommendation system, we investigated three methods: a simple average rating based model, the matrix factorization method built in HW4, and ALS. For cross-validation, we split our data into five equal folds just as in HW4. In our final model, we also processed data by filtering out users with fewer than ten reviews, leaving 1,526,933 out of 1,586,266 total reviews. We then ensured that there were at least two reviews for each user in each fold. This guaranteed that the model will have seen some training samples for a given user during testing instead of using the random initial vector. We used RMSE to quantify the quality of our models.

The popularity model is a simple model used to provide a baseline. This model assumed that every user rates a beer the same as the average rating for that beer. We calculate the RMSE for the popularity model by taking the difference between the

average rating of the beer without the user's rating and the actual user's rating. We also investigated using the matrix factorization method from HW4. We set up a similar environment by searching the parameter space for optimal regularization parameters and number of features. Most of the code here is shared from HW4.To run ALS, we used the implementation provided by Spark in the spark.ml.recommendation module. We also performed parameter search for ALS for regularization and number of features.

We ran our code on the discovery cluster both in local and standalone cluster mode. Ideas were tested in Jupyter Notebooks in local mode but then transferred to scripts that could be deployed to a standalone cluster once ready. Our Spark configuration used 100Gb of memory for each node and at most four nodes per cluster.

Results

We were able to achieve the lowest RMSE in the ALS model with an RMSE of 0.58 across five folds with the regularization parameter set to 0.08, 20 latent features, and 100 iterations. To determine the number of latent features, we first set the regularization parameter to 1, 2, and 3 and varied the number of features from 2 to 20.

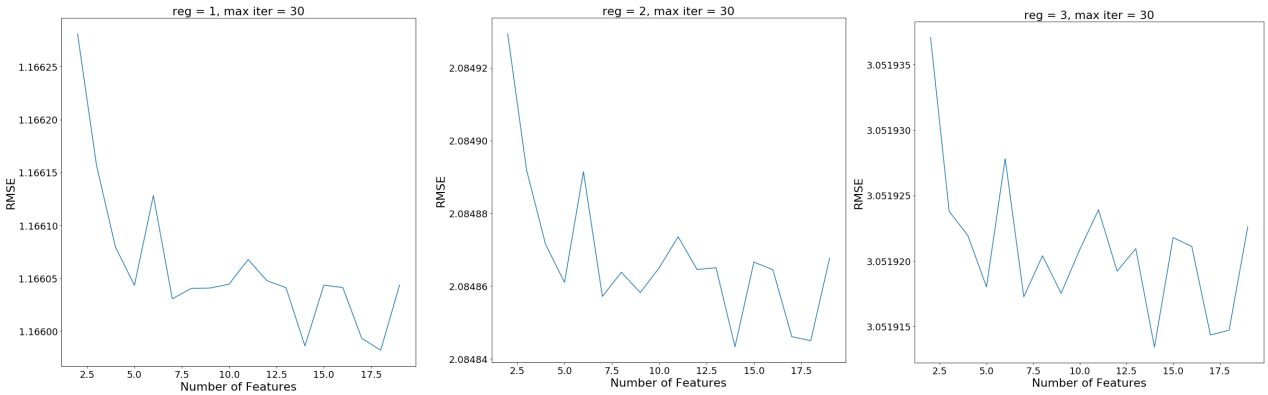


Figure 4: Number of Features for Various Reg Params

We saw that the pattern seemed consistent across different regularization patterns and so we set the regularization parameter to 1 and search the number of features from 2 to 50.

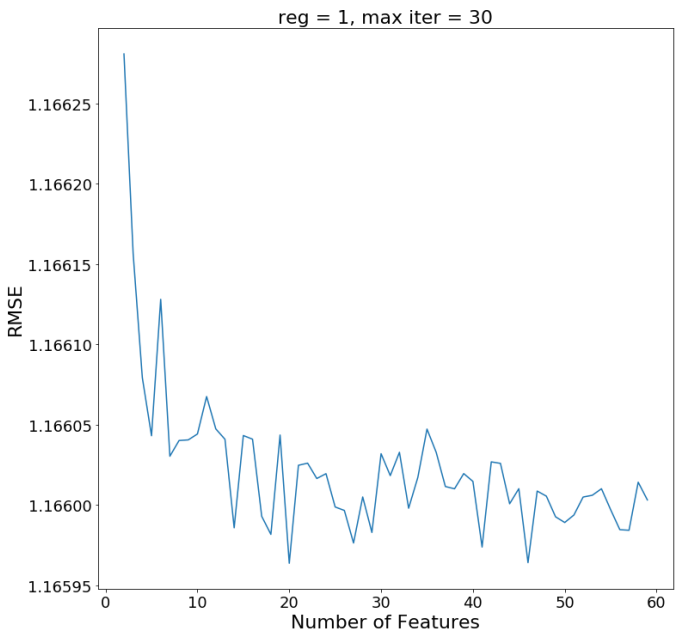


Figure 5: Number of Features vs RMSE

The optimal number of parameters wasn't fully clear. We chose 20 as it was showed a low RMSE and seemed reasonable from the graph.

We then fixed the number of parameters to 20 and ran the regularization search from 0.02 to 1 incrementing by 0.02 every iteration.

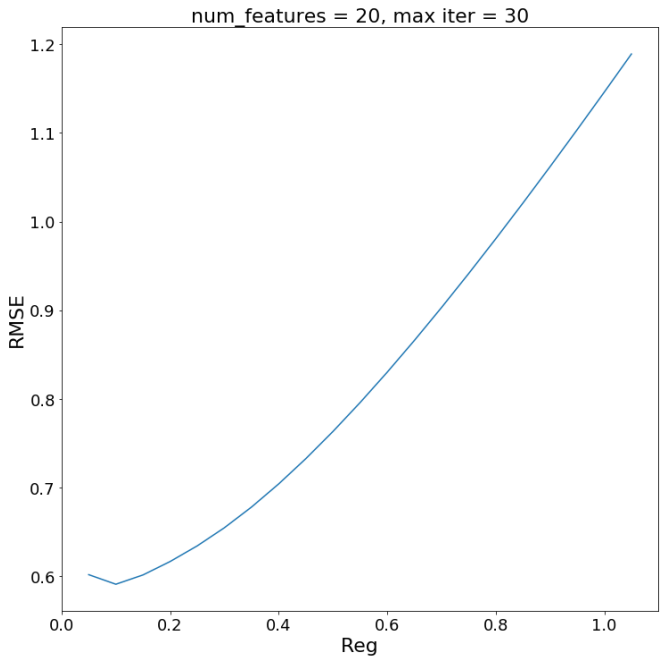


Figure 6: Regularization Param vs RMSE

The minimum RMSE was achieved when the regularization parameter was set to 0.08.

In the matrix factorization model, we attempted to do the same hyperparameter search and compare the results with the ALS method. We first faced difficulty in setting the learning rate and power as the model was susceptible to diverge and have its gradients explode. We found that a gain of 0.00025 and power of 0.18 worked the best. These parameters, however, caused the model to learn very slowly and even after running it for 300 iterations, the model still did not converge in its RMSE. 300 iterations took about 77 minutes and therefore it was not feasible to do a parameter search for matrix factorization. We borrowed the parameters from the ALS model and set the regularization for matrix factorization to 0.08 and the number of features to 20. After 300 iterations, we were able to achieve an RMSE of 0.83. Though it was still decreasing slowly, we decided that it was not worthwhile to continue running the matrix factorization as ALS had produced better results and much quicker.

In the simple popularity model where a prediction for a user's rating is the average rating for a beer, we were able to achieve an RMSE of 0.62. This method produced quite good results due to users' likelihoods to give reviews close to the average rating. In other words, beers have an objective quality that is consistent among users despite different personal tastes. The mean rating across all beers is 3.83 and the RMSE between all users' ratings and the mean rating of 3.83 is 0.72. Though these simple

models provide low RMSEs, no information is actually learned. Therefore, the only recommendation system possible is to recommend the most popular beer for every user despite their personal tastes.

Once the matrices for users and beers are trained, vectors from these matrices are able to be used to determine the similarity between beers or users. If a user likes a specific beer, then the model would predict that they might also like beers with similar vectors. Using this approach, a recommender system was created that allows users to input one or more beers that they like, and then a list of the most similar beers recommended to them is returned. If more than one beer is provided, the beer vectors are averaged together to create the recommendation. Users are also able to input styles, in which case all beers in the given style will be averaged together, and similar beers to that style vector are presented to the user. Visualization of these results is done with plotly, and screenshots are included below.

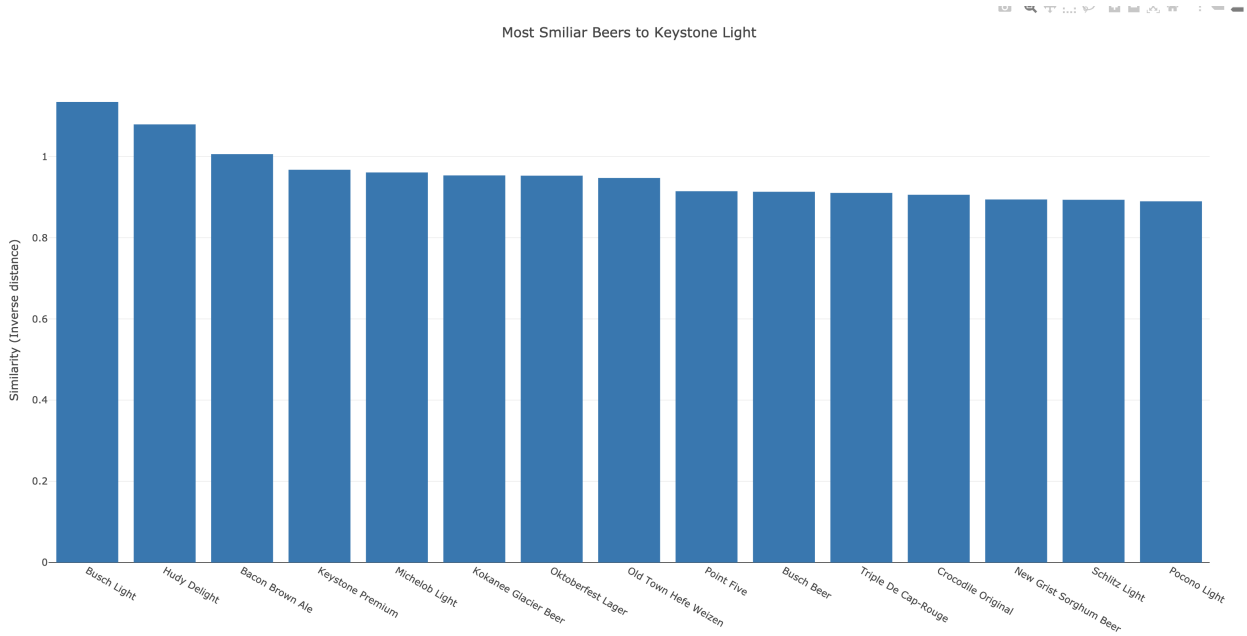


Figure 7: Suggested Similar Beers to Keystone Light

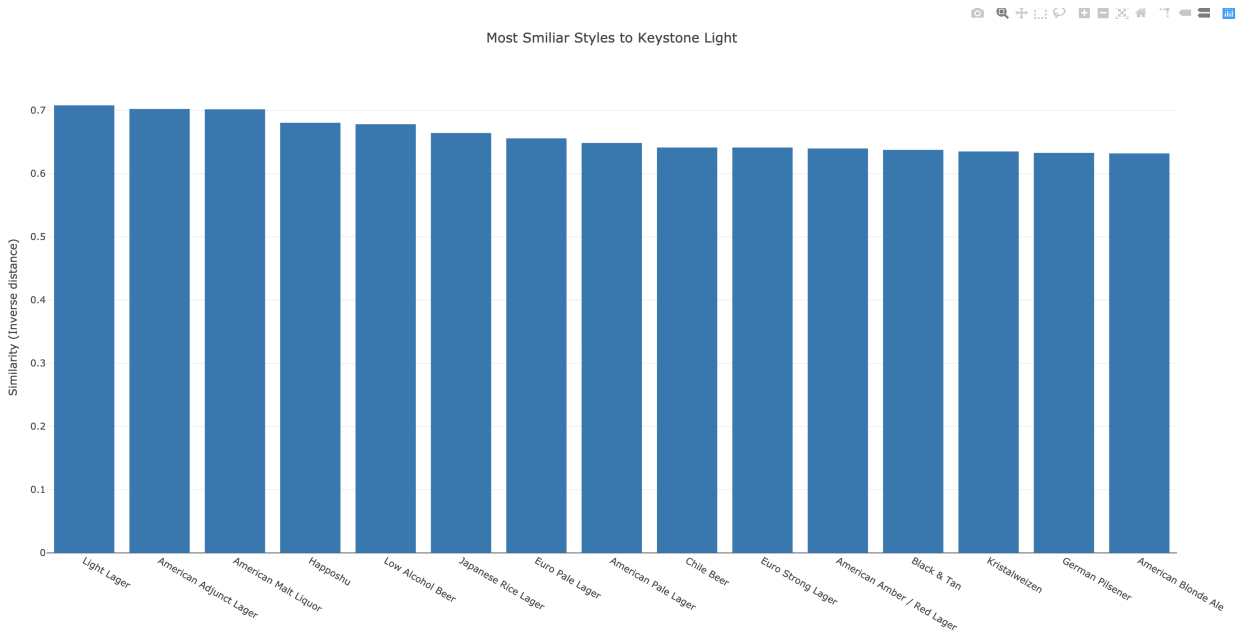


Figure 8: Suggested Similar Styles for Keystone Light

We also used visualization techniques to see if there are clusters in the data based on the style of beers. We wanted to see a specific type of beer like a stout is tightly grouped together, and also to see if different styles are separable. To do this we ran PCA on the beer matrix and graphed it in 2 and 3 dimensions. We also performed a T-SNE visualization, which is intended to preserve more separability information during downward projection. The following images include data from American Malt Liquor, American Barleywine, and Belgian Pale Ale styles.

Figure 9: 2D PCA for Beer Matrix

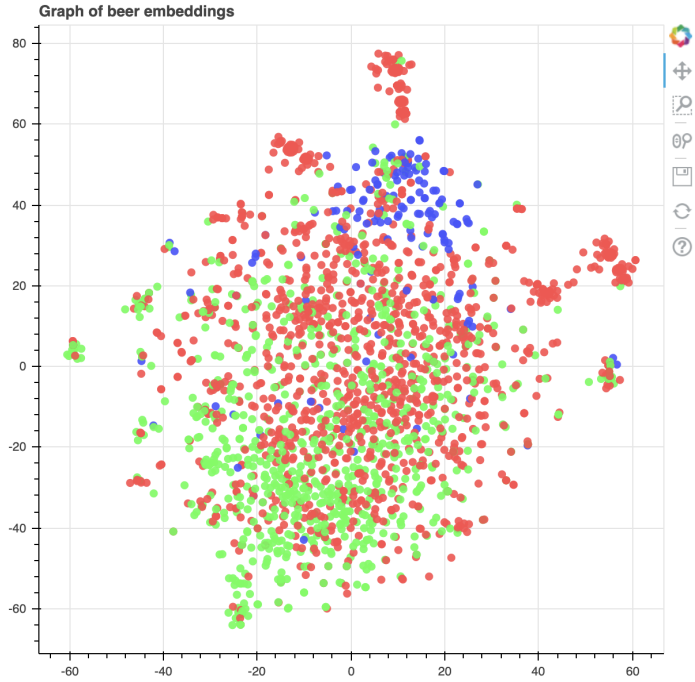


Figure 10: T-SNE Visualization for Beer Matrix

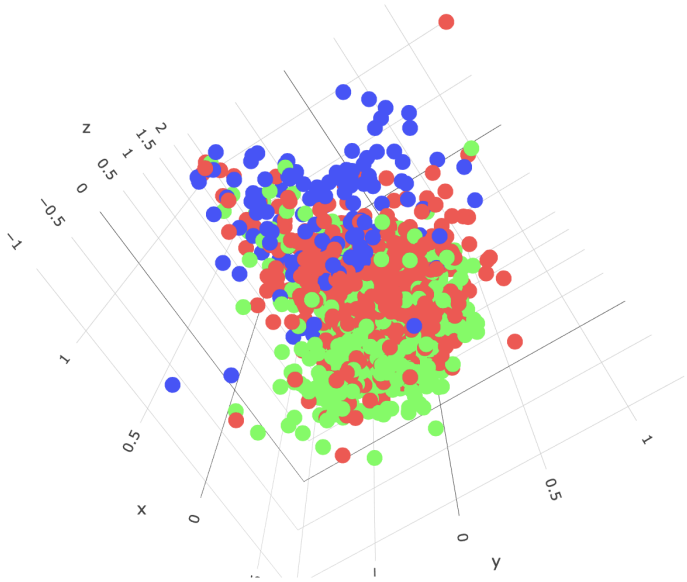


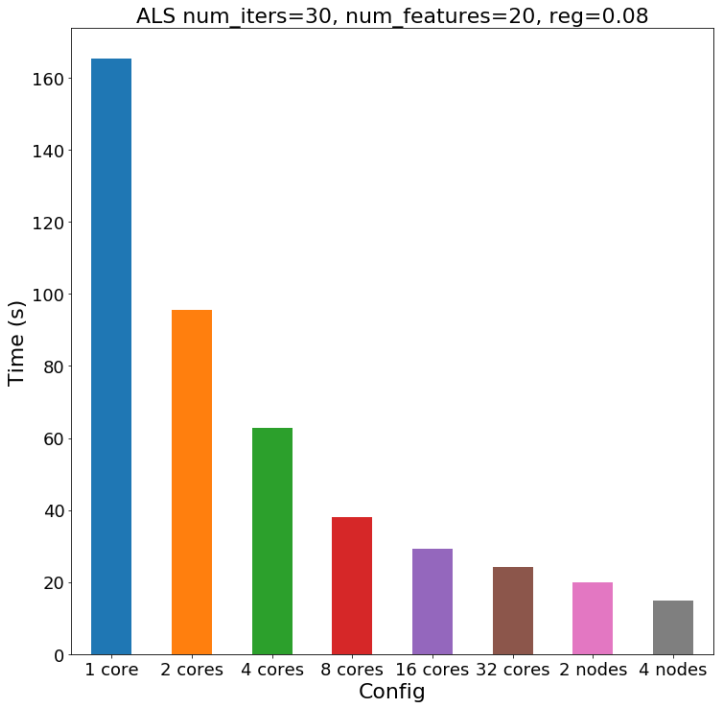
Figure 11: 3D PCA for Beer Matrix

Some separation can be seen between groups. The 3D visualization is able to capture more of the cluster separability because it uses more principal components. The presence of clusters shows that the model was able to learn representations for styles of beer effectively.

Performance Analysis

Running our code on multiple machines significantly sped up our ability to determine optimal parameters. All of our code is based in Spark and saw large speedups running it with more cores on more machines. There were difficulties in scaling our solution at first due to jobs failing and running out of memory. To fix such issues, we had to checkpoint our calculations sometimes have to manually save results and restore after a crash.

Once we tested our ALS code locally on 1, 2, 4, 8, 16, and 32 cores as well as across 2 and 4 worker nodes with 20 latent features, 30 iterations, and across the entire pruned dataset with 1,526,933 training samples averaged across 5 runs each.



As we increased the number of cores, the speedup became less apparent due to the increased communication cost between cores. In the end, we were able to achieve an 11.1x speedup between running on 1 core vs running on 4 nodes.

Conclusion

Compared to the popularity model with an RMSE of 0.62, our ALS model performed slightly better with an RMSE of 0.58. This validates our model as we were able to perform better than the simple approach. Using the latest features we learned during training, we continued to build out a recommendation engine that allows users to select beers they enjoy and suggest similar beers and beer styles.

Code

The code for this project can be found at
https://github.com/jaybooth4/ParallelProcessingFinal

Contributions

Jason: data preprocessing, wrote ALS Spark file with Dataframes, created recommender interface for similar beers/styles, data visualization with PCA/T-SNE

Chenyang: deployed ALS and MF models, optimal parameter search, accuracy/performance analysis, created a popularity-based model for comparison

Citations

Machine Learning Library (MLlib) Guide. (n.d.). Retrieved April 17, 2019, from https://spark.apache.org/docs/latest/ml-guide.html