Shreya Bhatia

Venkat Sai Charan Bandi

Dr. Huzefa Rangwala

CS584-001

Homework 4

Homework 4: Movie Recommender System

For this project, we worked in a group of two. The Miner username we used for our final submission was **Cherry_SLYR**. Our rank **230** was and our score was **0.97**.

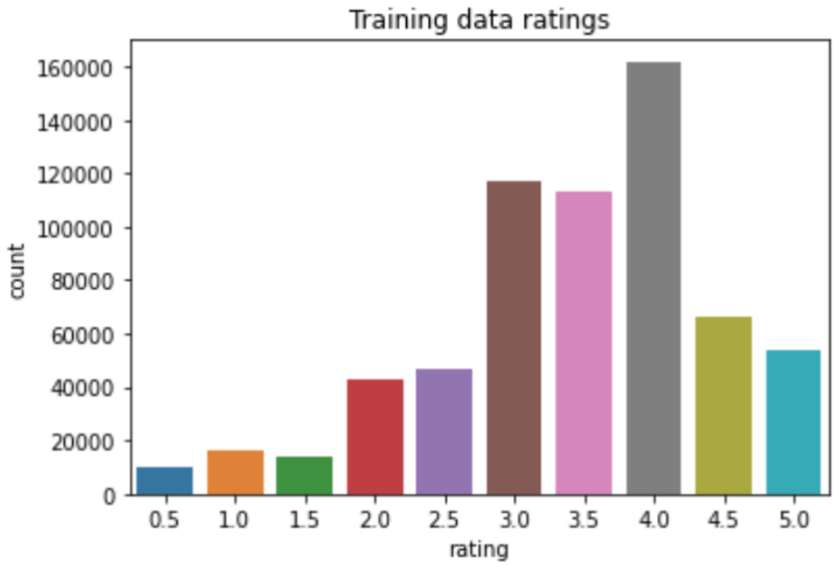
Overall Implementation

Per our knowledge from homework assignment number two, we first tried to put just training data from this assignment through a similar prediction pipeline which leveraged the RandomForestClassifier from sklearn. Upon visually analyzing the data with some graphs as shown below, we multiplied the floating-point ratings by two so they would all be whole numbers. This allowed us to avoid using regression in our pipeline and we decided to stick to the classification approach.

We also used the scikit-learn regressors to replace the classifier to see if that improved the score, even though analyzing the training data seemed to indicate reviews can only be in the range 1-5 with increments of .5.

Data Analysis

Upon analyzing, we noted the frequency of distribution and observed there were 10 total bins ranging from 0.5 to 5 with increments in 0.5. We normalized the bins into multiples of two so we could divide our total sample space into 10 different sets of ratings. We also observed most of the ratings lied between 3 to 4 in the dataset from the included graph we generated attached below, so the data is not balanced which may be a factor affecting our RMSE.



We used Pandas Python library to parse the given data from Miner into multiple data frames each including training data, test data, genres, and other features included within the additional files, we had to use 'latin1' encoding to read some data files that had unrecognized standard encoding. It is possible that names of directors had Latin characters.

After creating the data frames, we knew we wanted to add data to the training set to make the model more robust. We started this process by encoding genres into the training and test data for each movie. We did this encoding by creating a set of all unique genres from the 'movie_genres.dat' file. We obtained 20 different genres that existed and that each movie was tagged with one or more unique genres

Using Categorical Features

To use this genre data to train our model, we manually one hot encoded each genre into our training dataframe. We added columns for each genre to the training dataframe and flipped the genre to 1 if that genre was present for a given movie. All the genres that were not present in a given movie were left to equal 0. At the end of our approach, our training data head looks like the table below.

	userID	movieID	rating	genres	Action	Musical	IMAX	Romance	Comedy	Adventure	...	Animation	Drama	War	Mystery
0	75	3	2.0	Comedy|Romance|	0	0	0	1	1	0	...	0	0	0	0
1	75	32	9.0	Sci-Fi|Thriller|	0	0	0	0	0	0	...	0	0	0	0
2	75	110	8.0	Action|Drama|War|	1	0	0	0	0	0	...	0	1	1	0
3	75	163	8.0	Action|Romance|Thriller|	1	0	0	1	0	0	...	0	0	0	0
4	75	165	9.0	Action|Crime|Thriller|	1	0	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
641694	71534	42900	8.0	Comedy|Crime|Drama|Thriller|	0	0	0	0	1	0	...	0	1	0	0
641695	71534	44555	8.0	Drama|	0	0	0	0	0	0	...	0	1	0	0
641696	71534	46578	8.0	Comedy|Drama|	0	0	0	0	1	0	...	0	1	0	0
641697	71534	61075	10.0	Drama|Romance|	0	0	0	1	0	0	...	0	1	0	0
641698	71534	62049	9.0	Drama|Sci-Fi|	0	0	0	0	0	0	...	0	1	0	0

Feature Selection and Engineering

We initially tried to run the RandomForestClassifier on just the training data included in the data folder for this assignment. This allowed us to ensure we were building the pipeline correctly to predict the ratings. After this, we began to add data from the additional data files we were given to make the model more robust. We started with genre data because it was straightforward to encode since there were only 20 unique genres we were able to gather in the data.

Model Selection

We tried various classifiers from scikit-learn before choosing Random Forest including Decision Tree, Extra Trees, Multi-layer perceptron (neural network), AdaBoost, and HistGradientBoosting classifiers. Theoretical reasoning why RandomForest works so well maybe that it is an ensemble algorithm meaning that it creates multiple models to use for prediction. In the RandomForest case, it is creating multiple decision trees. Decision trees performed better than other types of models I tried for the problem, which is why RandomForestClassifier seemed most appealing to the problem. As we learned in class, this process of building many models and comparing their scoring is known as bagging for minimizing bias by combining low variance models.

Once we got to the hyperparameter tuning stage, RandomForest took a long time because it does not handle sparse data well, so we switched to using the Multinomial Naive Bayes model.

Validation

To validate that we were getting the best possible results from our model, we used RandomizedSearchCV to do efficient K-Fold Cross Validation using our pipeline as a model. The scoring metric used for the validation was RMSE or Root Mean Square Error which is the same metric used to evaluate our submissions on Miner.

Root Mean Square Error is the standard deviation of how far the data points predicted are from the actual regression line or line of best fit generated by a model.

Conclusion

The key findings we gained from the recommender system include:

1. There are lots of different approaches we can use to create the recommender system. We used a basic random forest classifier but we talked about matrix factorization, association rule mining, and user-based KNN in class. We also tried regressors and classifiers for the problem because our graphs showing all ratings were one of 10 different formats.

2. Encoders like OneHotEncoder can help translate string categorical features to numbers.

3. RandomForest performs well for a lot of problems, but not with sparse data.

4. RMSE is a good indicator of the total amount of error in the predictions for models that must predict continuous values.

5. Cross-validation and Grid/Randomized Search are effective for increasing accuracy by tuning hyperparameters for a particular model.

6. MinMaxScaler can help scale numerical data between 0,1 which helps performance around data point outliers. It did not seem that useful for the data we had in this project like userIDs and movieIDs, but the small performance improvements were impactful as this code ran on my local machine for over 30 minutes.

7. We did not end up needing it, but imputers are good for filling in missing data points (both categorical and numerical).