Project 3: Building user-based recommendation model for Amazon Report

By: Kureishi Shivanand

Date: January 18, 2020.

Course: Data Science with Python

Background

The dataset provided contains movie reviews given by Amazon customers. Reviews were given between May 1996 and July 2014.

Problem Statement

Explore the dataset and determine certain statistics. In addition, Netflix would like to build a machine learning algorithm which provide ratings for movies that haven't been seen for each of the users.

Remedies Applied for Problematic Dataset

Since the majority of the dataset contained missing values (represented as NaN), they needed to be replaced with a meaningful value in order for the algorithm to work. I couldn't have just drop the rows which contained missing values since basically none of the dataset would be left, hence this option was declined. So my only two options were left was replace the missing value with a value that was meaningful. I connsidered using an explicit value of 0 to act as a baseline in the algorithm, but since the rating scale was from -1 to 10 (even though ratings recorded were only from 1-5), a rating of 0 would have an actual effect on the prediction model. Hence, I decided to go with imputting the mean of each column (average rating recorded for each movie) as a replacement to the missing values (Figure 4). I believe this method gives a better indication of the movie through verified ratings.

Theory on Prediction Model Chosen

The goal of the prediction model is to determine the movies that a user would rate highly so Netflix could then recommend such movies to them. The better the model is at predicting user ratings, the greater the movie recommendations are to the customer, and thus overall better for the company as well. The provided data set gives User ID as the observations/row headers and Movie ID as the features/column headers, with the rating being the value. To predict ratings, matrix factorization (more specifically, SVD (Singular Value Decomposition)) would be used. It groups items from the original matrix into abstract concepts. It breaks matrix elements into single factors, removing all 'non-math' information to attain pure mathmatical results. With the factorization done, the system tries to predict the rating by combining user preferences with movie summaries [6]. To achieve this in Python, the Surprise library will be utilized (this should be fine since no restrictions on libraries were defined in the assignment). This library is made specifically for recommendation models [6].

Note: Due to the sparsity of the dataset (overwhelming amount of NaN values when compared to actual ratings given by users), the dataset is very noisy. Ideally, most of this noise would have been removed by deleting affecting movies, but since there is very little information as it is, the decision was made to keep all movies as to get a rating for each (as required in the Problem Statement). Hence, it was opted that the imputted mean values were the best way to work around this issue and produce a reasonably accurate prediction model. Also, a numeric (more specifically, a float) had to be used to fill the data set since each column in a dataframe can only contain one data type. Hence, it could not have just been replaced with None. This is also the reason np.nan is used for missing data in numeric dataframes instead of None.

Code and Results

Exploratory Data Analysis

```
# Exploratory Data Analysis: Question 1
print(data_df.notnull().sum().idxmax()) # get the index/id of the maximum of the entries per column that are numeric values
print(data_df.notnull().sum().max())    # get the maximum of the entries per column that are numeric values (not NaN)
```

```
Movie127
2313
```

Figure 1.

```
# Exploratory Data Analysis: Question 2
data_df.mean().sort_values(ascending=False) # get mean for each movie, then sort them from highest rating to lowest
```

```
Movie1      5.0
Movie55     5.0
Movie131    5.0
Movie132    5.0
Movie133    5.0
           ...
Movie60     1.0
Movie58     1.0
Movie45     1.0
Movie67     1.0
Movie144    1.0
Length: 206, dtype: float64
```

Figure 2.

```
# Exploratory Data Analysis: Question 3
data_df.notnull().sum().sort_values() # get number of non-null entries in each column and soft ascending order
```

```
Movie1         1
Movie71        1
Movie145       1
Movie69        1
Movie68        1
             ...
Movie29      243
Movie103     272
Movie16      320
Movie140     578
Movie127    2313
Length: 206, dtype: int64
```

Figure 3.

Brief Explanation of Exploratory Analysis Coding and Results

Figure 1 answers the first question of which movie has the maximum view/ratings (if there is a rating, the user had to first view the movie). Movie 127 has the maximum number of views (most users watched this movie) with 2313 ratings. The sum method was used to count how many numeric (not null) values each movie had, then decide on the maximum (using max method) value of the series and its id (using idxmax method) [1].

Figure 2 answers the second question by selecting the movies with the top 5 average recorded ratings. There are multiple movies with top rating (highest recorded rating: 5), but the method displayed the first it found. The average rating for each movie was found using the mean method then the series was sorted in

descending order (highest to lowest) using sort_values(ascending=False) [2]. According to this: Movie 1, Movie 55, Movie 131, Movie 132 and Movie 133 are the top 5 rated movies.

Figure 3 basically uses the logic of Figure 1 to answer question 3 of defining the top 5 movies with the least audiences. The numeric values (ratings) were summed for each movie then the values were sorted in ascending order (lowest to highest). Again, as with Question 2, there are multiple movies with only 1 view, but the ones recorded as the top were the first seen by the sort_values method. The top 5 movies with the least audiences are: Movie 1, Movie 71, Movie 145, Movie 69 and Movie 68.

Recommendation Model

```
# Preprocessing: replace NaNs with mean of each movie
from sklearn.preprocessing import Imputer
replaceNan = Imputer(missing_values=np.nan, strategy='mean', axis=0)
imputed_df = pd.DataFrame(replaceNan.fit_transform(data_df))
imputed_df.columns = data_df.columns
imputed_df.index = data_df.index
imputed_df
```

```
D:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:66: DeprecationWarning: Class Imputer is deprecated; Im
puter was deprecated in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198	Movie199	Movie200	Mov
A3R5OBKS7OM2IR	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
AH3QC2PC1VTGP	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
A3LKP6WPMP9UKX	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
AVIY68KEPQ5ZD	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
A1CV1WROP5KTTW	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
A1IMQ9WMFYKWH5	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
A1KLIKPUF5E88I	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
A5HG6WFZLO10D	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
A3UU690TWXCG1X	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3
AI4J762YI6S06	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.8	5.0	5.0	4.625	4.3

4848 rows × 206 columns

Figure 4.

```
from surprise import SVD
from surprise import accuracy
from surprise.model_selection import train_test_split
```

```
imputed_df.reset_index(level=0, inplace=True)
```

```
data_df_modified = pd.melt(imputed_df, id_vars=['user_id'], var_name='movie_id', value_name='rating')
```

```
from surprise import Dataset
from surprise import Reader
reader = Reader(rating_scale=(-1, 10))
data_df_v2 = Dataset.load_from_df(data_df_modified[['user_id', 'movie_id', 'rating']], reader)
```

```
trainSet, testSet = train_test_split(data_df_v2, test_size=.25)
```

```
algo = SVD()
```

```
algo.fit(trainSet)
prediction = algo.test(testSet)
prediction
```

```
[Prediction(uid='A355CL74L20X4N', iid='Movie35', r_ui=5.0, est=4.990919356099508, details={'was_impossible': False}),
 Prediction(uid='ALRVAO6XBSPV8', iid='Movie201', r_ui=4.333333333333333, est=4.341635465599857, details={'was_impossible': Fa
lse}),
 Prediction(uid='A3T2JX1MWYOCSE', iid='Movie192', r_ui=4.5, est=4.501140218369829, details={'was_impossible': False}),
 Prediction(uid='A2I2JJVP2I4KGC', iid='Movie200', r_ui=4.625, est=4.620078351800815, details={'was_impossible': False}),
 Prediction(uid='ALYPKUG1S4DYJ', iid='Movie151', r_ui=4.5, est=4.501555263161036, details={'was_impossible': False}),
 Prediction(uid='A160AB64G2E949', iid='Movie102', r_ui=4.0, est=4.011061763105762, details={'was_impossible': False}),
 Prediction(uid='A2KT3M7HFAOSKR', iid='Movie161', r_ui=4.6, est=4.596701244310747, details={'was_impossible': False}),
 Prediction(uid='A310SB7OWCFKQ2', iid='Movie180', r_ui=5.0, est=4.9836513322986224, details={'was_impossible': False}),
 Prediction(uid='A1E5L728SLU0TG', iid='Movie142', r_ui=5.0, est=4.993618557701867, details={'was_impossible': False}),
 Prediction(uid='A3FOJHZF01BKXW', iid='Movie140', r_ui=4.833910034602076, est=4.795673181382421, details={'was_impossible': F
alse}),
```

Figure 5.

```
accuracy.rmse(prediction)
```

```
RMSE: 0.0851

0.08510341715244817
```

Figure 6.

```
data_df_zero.reset_index(level=0, inplace=True)
data_df_modified2 = pd.melt(data_df_zero, id_vars=['user_id'], var_name='movie_id', value_name='rating')

data_df_v3 = Dataset.load_from_df(data_df_modified2[['user_id', 'movie_id', 'rating']], reader)
trainSet2, testSet2 = train_test_split(data_df_v3, test_size=.25)
algo2 = SVD()
algo2.fit(trainSet2)
prediction2 = algo2.test(testSet2)
prediction2
```

```
[Prediction(uid='A3QABD7O3UGPE9', iid='Movie9', r_ui=0.0, est=-0.006101569090192906, details={'was_impossible': False}),
 Prediction(uid='A28Q3IONFX1KD8', iid='Movie175', r_ui=0.0, est=0.006867085181395996, details={'was_impossible': False}),
 Prediction(uid='A3LXKO2S6G2DBI', iid='Movie192', r_ui=0.0, est=-0.006647028200268901, details={'was_impossible': False}),
 Prediction(uid='A1004AX2J2HXGL', iid='Movie146', r_ui=0.0, est=-0.007103434575813477, details={'was_impossible': False}),
 Prediction(uid='A3JQZHU3CYS60O', iid='Movie29', r_ui=0.0, est=0.49627292605392254, details={'was_impossible': False}),
 Prediction(uid='A1KI6L82M5RAHQ', iid='Movie165', r_ui=0.0, est=-0.0005140934710450157, details={'was_impossible': False}),
 Prediction(uid='A130DD6CVD40F3', iid='Movie31', r_ui=0.0, est=-0.009713277979505861, details={'was_impossible': False}),
 Prediction(uid='A3AP6H48QHX8BR', iid='Movie127', r_ui=4.0, est=1.711568370221345, details={'was_impossible': False}),
 Prediction(uid='A3ADK3ZTJ87915', iid='Movie47', r_ui=0.0, est=0.0016271917808932611, details={'was_impossible': False}),
 Prediction(uid='A1K3ANBJNUWEV', iid='Movie123', r_ui=0.0, est=-0.002030674147238983, details={'was_impossible': False}),
 Prediction(uid='A21ERFG7H737MF', iid='Movie190', r_ui=0.0, est=-0.0005431958603872881, details={'was_impossible': False}),
 Prediction(uid='A1PBUABYGNCHUZ', iid='Movie98', r_ui=0.0, est=0.002306134466281721, details={'was_impossible': False}),
 Prediction(uid='A2SC1NB60TVAKU', iid='Movie141', r_ui=0.0, est=0.04872778099962093, details={'was_impossible': False}),
 Prediction(uid='A1XKFLOE7YJQAR', iid='Movie80', r_ui=0.0, est=-0.018228099381737747, details={'was_impossible': False}),
 Prediction(uid='A16IIQA8V9IQS5', iid='Movie191', r_ui=0.0, est=0.018558271301122628, details={'was_impossible': False}),
 Prediction(uid='A1K3ANBJNUWEV', iid='Movie21', r_ui=0.0, est=-0.0059586862538616526, details={'was_impossible': False}),
 Prediction(uid='A3S7Z9PRDH2C7L', iid='Movie188', r_ui=0.0, est=0.01163549613425547, details={'was_impossible': False}),
 Prediction(uid='A36J3RGUT0XYK8', iid='Movie11', r_ui=0.0, est=0.0005715234990918711, details={'was_impossible': False}),
 Prediction(uid='AOGUS6N3KP2O5', iid='Movie137', r_ui=0.0, est=0.0037151881190712307, details={'was_impossible': False}),
 Prediction(uid='A2OOI90Y1HNTP9', iid='Movie2', r_ui=0.0, est=-7.074796141577568e-05, details={'was_impossible': False})
```

```
accuracy.rmse(prediction2)
```

```
RMSE: 0.2757

0.27574076983596013
```

Figure 7.

Brief Explanation of Recommendation Model Coding and Results

Figure 4 gets the dataset ready for the SVD prediction model by imputting missing value (np.nan) with the mean along the column (average recorded rating for each movie). This new dataframe was named imputted_df as opposed to the original dataframe (data_df) [3].

Figure 5 deals with transforming the dataframe to a suitable format for the SVD method to create a prediction model, as well as training the model with 75% of the data and test its accuracy with 25% of the remaining data. The first part of deals with importing required modules from the Surprise library, then transforming the index column (user_id) to an actual column to use in the creation of the prediction model (data_df_modified) [4]. To create a SVD model, Reader and Dataset modules need to be imported from Surprise, in order to use a dataset in training and testing [5]. To instantiate the Dataset object, the 3 columns of user id, movie id and rating must be assigned to their relative columns. The reader object sets the rating scale which was defined from -1 to 10. The formatted dataset is named data_df_v2 [5]. Next, the training (75%) and testing sets (25%) of data_df_v2 are defined using the train_test_split module of surprise.model_selection class [5]. The SVD object is also instantiated and named algo [5]. In the next cell, the training (fiiting training set to model) takes place, then test the model with testing data. The testing results are assigned to variable: prediction [5]. This is a list that contains certain elements for each user's rating of the different movies. The main fields of interest are: uid (user whom predictions carried out for), iid (movie id defined as item id), and est (expected rating user expected to give a movie) [6].

Figure 6 defines the RMSE (root mean squared error) of the model (further insight into this value given in the 'Analysis/Insight' section). Losely, the lower the RMSE leads to better model accuracy [7]. The RMSE of this model is 0.0851, which is extremely good. This shouldn't be a surprise however since the NaN values were replaced with column means, which relate much better with the column/movie ratings

than an arbitrary 0 value. Note that the estimator contains floats since the data type are floats and should also be allowed since the rating scale is also interpreted as continuous since it was defined of [-1,10].

Figure 7 follows the same coding semantics as in Figure 5 and 6. The only difference being the dataset used was filled with 0 (data_df_zero= data_df.fillna(0)) instead of column averages. This was only included as a comparison of the RMSE values to visualize the difference the value used to fill NaN has on the accuracy. The RMSE value for this model was 0.2757 which is much higher than the previous model, meaning this model is a worse fit/less accurate model as compared to the previous model.

Analysis/Insight

The major insight that can be obtained from the Exploratory Analysis is from the first question. Because many movies have so few ratings, it was interesting to see that Movie 127 stood out to most audiences. It can be inferred that this movie had mass appeal with 2313 people viewing it. The other questions defined the maximum rating as 5 (even though the scale's max was 10). This proved that the variety of movie recorded were not particularly good. The analysis also showed the minimum viewer of a movie is 1 (makes sense since at least one person needed to view the movie for it to have a rating, so it could have qualified for the prediction model).

The major insight attained from the Recommendation Model is in the preprocessing phase. The biggest factor that would affect my prediction model metrics is the value chosen to replace NaN in the dataset. The reasons why column means were chosen were discussed heavily in detail in 'Theory on Prediction Model Chosen', so I won't go over them here. However, it should be noted that it is important to choose replacement values that will relate to your 'real recorded' value and not 'trick your model' (as would have been the case if 0 was used).

Lower RMSE indicates better fit. RMSE is the standard deviation of residuals (prediction error or unexplained variance). Residuals are measure of how far from regression line data points are, therefore the RMSE is a measure of how spread out the residuals are [7]. RMSE is an absolute measure of fit, whereas R-squared is a relative measure of fit [8].

The model which used the imputed_df (column means) dataset produced a more accurate model when compared to when only 0s were used to fill in missing data (data_df_zero). This makes sense since the average of ratings of a particular movie is more relative to that movie than an arbitrary value such as 0.

Overall, the SVD model in the Surprise library might not be the best way to go about creating a prediction model, but it is very logical and straight-forward. The results and parameters required of the model also relate heavily to the dataset to be analyzed. In addition, due to the heavy noise (from NaN), the model might not be the most optimal. However, given the data I was, I believe the accuracy attained from the first model (Figure 5, 6) using imputed_df dataset was quite high and the model fits the data quite well.

References:

[1] pandas.Series.idxmax (January 18, 2020). *pandas 0.25.3 documentation.* Retrieved from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.idxmax.html

[2] Python | Pandas Series.sort_value() (January 18, 2020). *GeeksforGeeks*. Retrieved from https://www.geeksforgeeks.org/python-pandas-series-sort_values/

[3] Impute entire DataFrame (all columns) using Scikit-learn (sklearn) without iterating over columns (January 18, 2020). *Stack Overflow*. Retrieved from https://stackoverflow.com/questions/33660836/impute-entire-dataframe-all-columns-using-scikit-learn-sklearn-without-itera

[4] pandas.melt (January 18, 2020). *pandas 0.25.3 documentation.* Retrieved from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.melt.html

[5] Getting Started (January 18, 2020). *Surprise 1 Documentation*. Retrieved from https://surprise.readthedocs.io/en/v1.1.0/getting_started.html

[6] Building Python Recommendation Systems that Work – Blog Jakub Cwynar (January 18, 2020). *Mirumee*. Retrieved from https://blog.mirumee.com/building-python-recommendation-systems-that-work-8d8d218c1464

[7] RMSE: Root Mean Square Error (January 18, 2020). *Statistics How To*. Retrieved from https://www.statisticshowto.datasciencecentral.com/rmse/

[8] Assessing the Fit of Regression Models (January 18, 2020). *The Analysis Factor*. Retrieved from https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/