4/12/2019
BUAN 6346
Project Type 2
Group 1
Topics: Machine Learning & Human buying behavior

Title: How can Machine Learning help in modeling and predicting human buying behavior for Best Buy?
List of group members:

- Yajaira Gonzalez
- Soren Thomsen
- Nirlep Shah
- Mandeep Narang
- Sunny Yadav

Problem statement
We will predict which Best Buy product a mobile web visitor will be most interested in based on their search query or behavior over 2 years.

Questions of interest: Optimizing the mobile app recommendation system in order to increase incremental sales and potentially streamline supply chain by reducing product offerings based on user feedback. Can we use past consumer browsing behavior in order to predict future purchases.

Why? Our team's focus is to gain real world experience on problems companies are having today. Predicting the next occasion or product selection is a question every business wants to know and being able to solve this question in a way that is actionable not only helps build experience but brings a skill set of value to potential employers. By using web data we will be able to find repeated patterns in a users behavior which will then allow us the ability to identify the next product in their purchase history. An analysis on web data is also to our team an universal learning we can apply to many other projects.

Motivation: We hope to predict which Best Buy product a mobile web visitor will be most interested in.

Resources used for the project and the dataset (link to dataset included)
The main data for this project is in the train.csv and test.csv files. These files contain information on what items users clicked on after making a search.
Link: https://www.kaggle.com/c/acm-sf-chapter-hackathon-big/data
Resources:
Best Buy Documentation: https://developer.bestbuy.com/documentation
Best Buy API: https://developer.bestbuy.com/#TableProdRefInfo
and more BigDataR tools: https://github.com/koooee/BigDataR_Examples/tree/master/ACM_comp
https://blog.cloudera.com/blog/2016/05/how-to-build-a-prediction-engine-using-spark-kudu-and-impala/
Other preliminary information:
The analysis was a result of a hackathon in SF in 2014.

The list of tasks each member performed:

Task	Team Lead
Data exploration and visualization ● Find open source data set and download ● Explore the data set in terms of its features, attributes (columns), records (rows), and size, include your findings in your report. ● Visualize the data for a better insight and make it a part of your project report.	**Yajaira Gonzalez**
Data Cleanings ● Missing values, ● Duplications, ● Etc. ● include what you needed to do for data leaning in your report	**Soren Thomsen**
Analysis ● Refer to the problem statement and apply your analysis methods ● Write your queries and illustrate the results, use visualization methods, and include the analysis explanation into your report.	**Lead: Mandeep Narang** **Support:** ● **Nirlep** ● **Soren Thomsen** ● **Yajaira Gonzalez** ● **Sunny Yadav**
Conclusion and future work ● make conclusions based on the analysis (part of the report) ● mention potential future work in your report.	**Lead: Nirlep** **Support: Soren Thomsen**
Report implementation environment ● Computing environment, hardware and software, operating system ● Software tools used in the analysis ● If used, Python version and libraries ● If used, include Python code as an appendix	**Lead: Sunny Yadav** **Support:** ● **Yajaira Gonzalez** ● **Nirlep** ● **Soren Thomsen** ● **Sunny Yadav**

Data Exploration & Visualization

Data set descriptors

Each line of train.csv describes a user's click on a single item and is (1865269 rows, 6 columns). It contains the following fields:

user: A user ID
sku: The stock-keeping unit (item) that the user clicked on
category: The category the sku belongs to
query: The search terms that the user entered

click_time: Time the sku was clicked on
query_time: Time the query was run

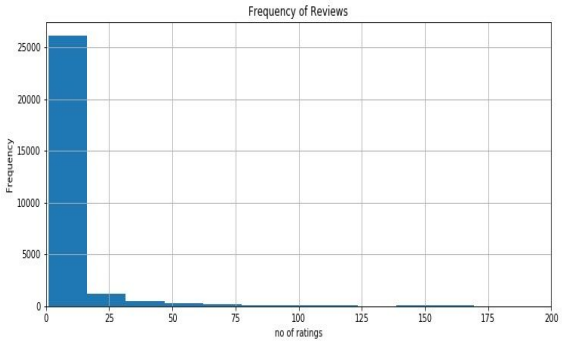
Test.csv contains all of the same fields and is (1865269 rows, 6 columns). The train.csv includes all of the same fields except for sku and is(1865269 rows, 5 columns). We will then estimate which sku's were clicked on in these test queries. Due to the internal structure of Best Buy's databases, there is no guarantee that the user clicks resulted from a search with the given query. What we do know is that the user made a query at query_time, and then, at click_time, they clicked on the sku, but we don't know that the click came from the search results. The click_time is never more than five minutes after the query_time. In addition, there is information about products, product categories, and product reviews in product_data.tar.gz.

Data Cleaning Process:

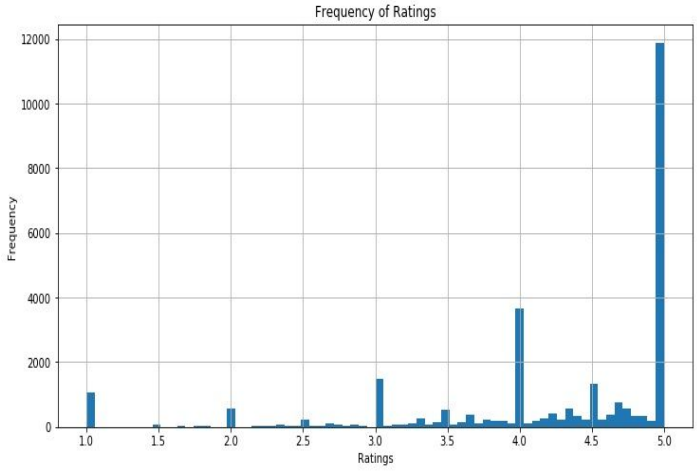
A large part of our initial analysis was focused on getting the data converted from .xml files to csv files. Once this was completed we removed a portion of the data that we did not need for analysis.

Visualizations:

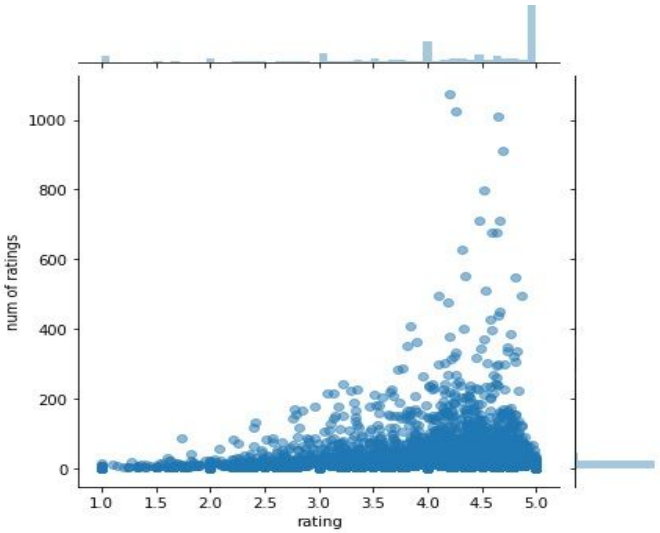
Understand Rating dataset for user-user similarity model (elaborate more on our EDA and why we looked at certain things)



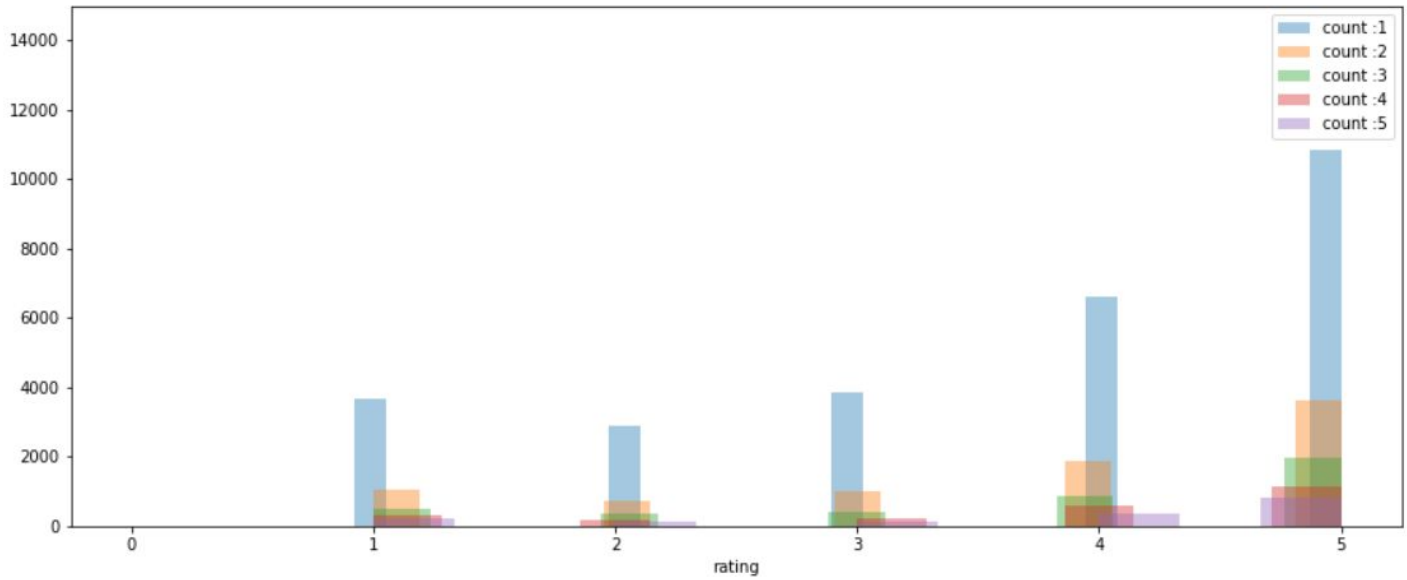
This title shows the frequency of product reviews and clearly shows that most products on the mobile app only have one or two reviews which limited the scope of our analysis



Most product reviews show mainly 5 star ratings

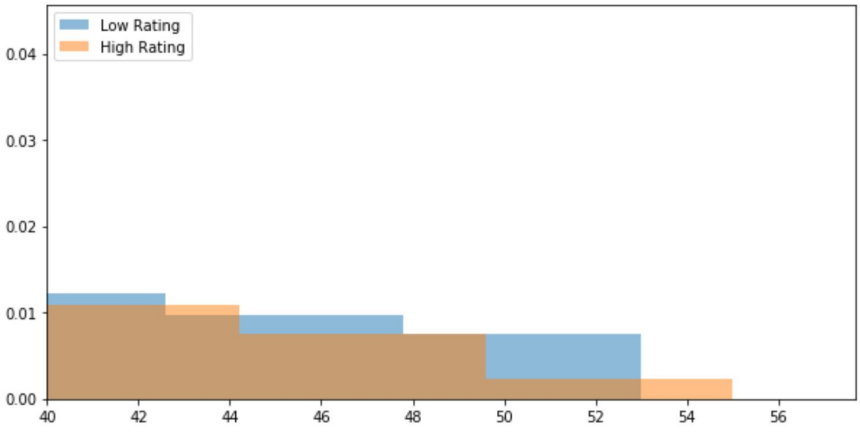


Here the density of Ratings is shown. Where we can see how density of users who have given a rating 1 for a product differs from density of users who have given a rating of 2, 3, 4 and 5



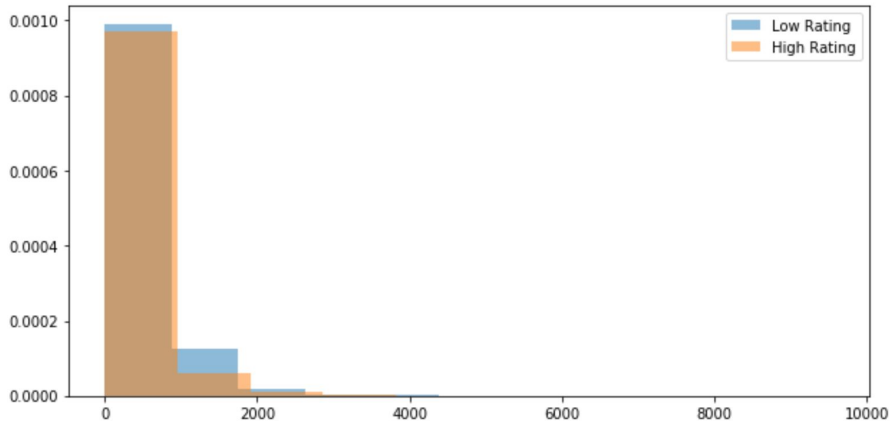
Here we see the frequency of votes for each ratings i.e 1, 2, 3, 4, 5
We see that most users have rated a product only once.

Title length exploration



Here we can see that there is no direct obvious relation between the Title length of a product and the ratings given by user

Comment length exploration

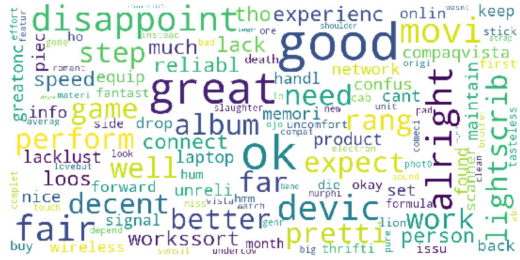


Even here we can see that there is no direct obvious relation between the Title length of a product and the ratings given by user

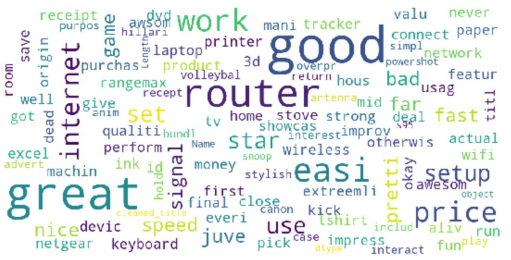
Rating: 1

Rating: 2

Rating: 3



Rating: 4



Rating: 5

Here is the word cloud of all ratings and the size of each word indicates its frequency or importance. That is to see what words were more frequently used to rate a certain product.

Analysis

To help predict the best product for a consumer based on past online history we must create a recommendation model. Our research pointed us to 3 implementable strategies based on existing best practices for recommendation models. Below is a summary of the queries and methods we employed for each strategy.

Data Preparation:
To prepare our data for the analysis we filtered products that were unpopular and focused on the 'sku', 'user', 'name', 'flag' resulting in a 'merged' data frame of 329148 rows, 5 columns. We use this df in our analysis below to cross reference recommendations with product name, category based on user & sku.

Collaborative Filtering:
A method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). There are two categories of CF, user-based & item-based.

Summary: This was an unsupervised model based on two frameworks, pearson correlation and cosine similarity. We dropped the users from the matrix and used item/user columns in order to check the relations between each item/user column. We then sort each items/user based on correlations coefficients in order to find similarities and take top n items to be used for recommending to new users. Generally we used between 10 and 20 items.

The first part of the CF method is to create a matrix.

Steps in Matrix development:

1. Create User & Item Sparse matrix; User set as index and product sku set as columns (for user based we used users for the columns), the values in the matrix belong to existing ratings

sku	7897001	7903227	7918177	7933686	7934658	7942845	7945361	7956599	7960986	7968773	...	18850273
user												
00003cb3f85244c652f22c1daf11aed35d5ab7f6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
0000776d7bf35b984ca8e3671327a7ac1d07a86c	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
000126150c17acb66664a42c47d5dba399311783	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
00014aac7ef84270dab58af1c92b79685b89b9b9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
00016ac345cf3220c4f56855d7021693290796c2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

5 rows × 976 columns

R_df.shape
271,937 rows x 976 columns

Understanding the quality of the matrix:

```
print('Shape of Sparse Matrix: ', R_df.shape)
print('Amount of Zero occurrences: ', (R_df == 0).sum(axis=1).sum())
# Percentage of zero values
density = (100.0 * (1-((R_df == 0).sum(axis=1).sum()) / (R_df.shape[0] * R_df.shape[1])))
density
print('density: {}'.format(density))
```

```
Shape of Sparse Matrix:  (271937, 976)
Amount of Zero occurrences:  265081364
density: 0.1240146810763898
```

User/Item Based, Using Sparse matrix - Using Pearson Correlation

User similarity works well if number of items is much smaller than the number of users and if the items change frequently. Item similarity works best when user base is smaller.

For our user and item based CF model we looked at similarity by employing the use of Pearson's correlation coefficient. To help illustrate how **Person correlation** works, let $r_{u,i}$ be the rating of the ith item under user u, $\overline{r_u}$ be the average rating of user u, and $com(u, v)$ be the set of items rated by both user u and user v. The similarity between user u and user v is given by Pearson's correlation coefficient.

$$sim(u,v) = \frac{\sum_{i \in com(u,v)} (r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum_{i \in com(u,v)} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{i \in com(u,v)} (r_{v,i} - \overline{r_v})^2}}$$

For our analysis and application of PC in our python code we first selected 2 users. We then defined the PC function to find correlations of user 1 & 2 based on existing ratings that matched those users behavior in the developed matrix. The table below shows what users are closely related to user 1. Please reference code for item correlation table.

In [186]:
```
def Similar_users_pearson (user_rating):
    users_similer_to_user = R_df_trans.corrwith(user_rating)
    corr_tab = pd.DataFrame(users_similer_to_user,columns=['Correlation'])
    corr_tab.dropna(inplace=True)
    corr_tab=corr_tab.sort_values('Correlation',ascending=False).reset_index()
    return corr_tab
```

In [187]:
```
users_similer_to_user1 = Similar_users_pearson(user1_rat)
users_similer_to_user1.head()
```

Out[187]:

	user	Correlation
0	889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898	1.000000
1	050ea239d38e11d097eb62835baa1e9622b4e06c	0.665639
2	3c6d2056a593a944ca9170c34feca7e0efe38d88	0.576758
3	e29eb6d24e001125625913e58149ff2e264d8abd	0.576758
4	04f5e5a20bcf5f203c035a7c5aa3409715e7f3c6	0.576758

This information from the PC table of correlations then helped take our analysis a step further by being able to cross reference the full 'merged' data frame and understand what products have been already

viewed, viewed by others similar to the user that are unique and provide a list of recommended items. The code below helped generate the more detailed recommendation output.

In [167]:
```
def recommendations(n_near_users,user,similar_to_user_data):
    alreadyviewd = merged[merged['user'] == user].name.unique()
    rec_dict={}
    for i in range(1,n_near_users+1):
        similati = similar_to_user_data['user'][i]
        reci =  merged[merged['user'] == similati].name.unique()
        for rec in reci:
            if rec in rec_dict:
                rec_dict[rec] = rec_dict[rec]+1
            else:
                rec_dict[rec] = 1
    rec_list=[]
    for i in rec_dict.keys():
        if i not in alreadyviewd:
            rec_list.append(i)
    return rec_list,alreadyviewd,rec_dict
```

In [44]:
```
rec_list,alreadyviewd,rec_dict = recommendations(250,user1,users_similer_to_user1)
```

In [97]:
```
print("Already viewed items: "+ "\n"+ str(pd.DataFrame(alreadyviewd)) + "\n")
print("Viewed by other users - unique items: "+"\n"+ str(pd.DataFrame(list(rec_dict.keys())))+'\n')
print("Recommended list of items: "+"\n" + str(pd.DataFrame(rec_list))+"\n")
```

Final Recommendation from User based, Sparse Matrix using PC:
(Based only on User1)

```
Already viewed items:
                                                                                0
0  Bose® - SoundDock® Series II Digital Music System for Apple® iPod® - Black
1  Klipsch - Image S4 Earbud Headphones
2  Altec Lansing - Mix Digital Boombox for Apple® iPhone and iPod® - Black

Viewed by other users - unique items:
                                                                                0
0  Bose® - SoundDock® Series II Digital Music System for Apple® iPod® - Black
1  Altec Lansing - Mix Digital Boombox for Apple® iPhone and iPod® - Black
2  Klipsch - Image S4 Earbud Headphones
3  Bose® - SoundDock® Portable Digital Music System for Apple® iPod® - Black

Recommended list of items:
                                                                                0
0  Bose® - SoundDock® Portable Digital Music System for Apple® iPod® - Black
```

User Based, Using Sparse matrix - Using Cosine Similarity
Same approach was produced for the item-based category

After assessing our recommendation based off of PC, we wanted to take a look at different methods, focusing on cosine similarity in this section. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

Step 1. Create a copy of the ratings matrix with all null values imputed as 0; r_matrix_dummy
Step 2.. Create a matrix based off of user 1 & 2, we named this user_rat in our code. [2 rows x 976 columns]

sku	7897001	7903227	7918177	7933686	7934658	7942845	7945361	7956599	7960986	7968773	...	18850273
user												
889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
e93d81ed71a34bf33b61a4bc0cd2a486d73da2d2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

Step 3. Compute cosine similarity matrix using the ratings matrix and the user matrix

Step 4. Review qualities of the matrix

In [53]:
```
#Convert into pandas dataframe
cosine_sim = pd.DataFrame(cosine_sim, index=r_matrix_dummy.index, columns=user_rat.index)
print(cosine_sim.sum())
cosine_sim.head()
```

```
user
889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898    465.464611
e93d81ed71a34bf33b61a4bc0cd2a486d73da2d2    656.213823
dtype: float64
```

Out[53]:

user	889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898	e93d81ed71a34bf33b61a4bc0cd2a486d73da2d2
user		
00003cb3f85244c652f22c1daf11aed35d5ab7f6	0.0	0.0
0000776d7bf35b984ca8e3671327a7ac1d07a86c	0.0	0.0
000126150c17acb66664a42c47d5dba399311783	0.0	0.0
00014aac7ef84270dab58af1c92b79685b89b9b9	0.0	0.0
00016ac345cf3220c4f56855d7021693290796c2	0.0	0.0

Now that we have created this matrix, we can edit the table to showcase the user by user recommendation. In the table below you can see we only looked at user 1 and we ranked the similarity of users for better viewability.

In [54]:
```
similar_to_user1 = cosine_sim[user1].sort_values(ascending = False).reset_index()
```

In [55]:
```
similar_to_user1.head()
```

Out[55]:

	user	889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898
0	889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898	1.000000
1	050ea239d38e11d097eb62835baa1e9622b4e06c	0.666667
2	0cb341bb3eed19536367fa7c88dd20c82c40e550	0.577350
3	9e3d25835122bab061cc5c683bd9ed8387503828	0.577350
4	d8ac7d22e57e03ca6322aac5d751127918c534d1	0.577350

This information is now ready to be run into our recommendation function for better understanding of what the cosine similarity approach produced for user 1.

In [57]:
```
rec_list,alreadyviewd,rec_dict = recommendations(10,user1,similar_to_user1)
print("Already viewed items: "+ "\n"+ str(alreadyviewd) + "\n\n")
print("Viewed by other users - unique items: "+"\n"+ str(rec_dict)+"\n\n")
print("Recommended list of items: "+"\n" + str(rec_list)+"\n")
```

```
Already viewed items:
['Best Buy GC - $100 Gift Card' 'SanDisk - Cruzer 4GB USB 2.0 Flash Drive'
 'Lowepro - Navi Plus Camera Case and GPS Pouch - Black']


Viewed by other users - unique items:
{'Best Buy GC - $100 Gift Card': 2, 'Smart Choice - Stainless-Steel Refrigerator Water Line Kit - Silver': 1, 'SanDisk - Cru
zer 4GB USB 2.0 Flash Drive': 5, 'Lowepro - Navi Plus Camera Case and GPS Pouch - Black': 4}


Recommended list of items:
['Smart Choice - Stainless-Steel Refrigerator Water Line Kit - Silver']
```

Model Based Collaborative Filtering

Based on the item & user data set, a utility matrix was composed using SVDpp, SVD and NMF found in the python Surprise library. This is a library that was created specifically for collaborative filtering of data.

We then computed similar browsing trends between users by comparing rows of the user matrix. Accuracy of the model is based on the RMSE scores after running model on the test set.

Matrix Factorization-based algorithms

Class of collaborative filtering algorithms used in recommender systems. The essence is that MF helps represent users & items in a lower dimensional latent space

SVD: algorithm is equivalent to Probabilistic Matrix Factorization

- Decomposes a matrix A into the best lower rank (i.e. smaller/simpler) approximation of the original matrix A. To get the lower rank approximation, we take these matrices and only keep the top k features, which can be thought of as the underlying tastes and preferences vectors.

SVD++: algorithm is an extension of SVD that takes into account implicit ratings.

NMF: collaborative filtering algorithm based on Non-negative Matrix Factorization. It is very similar with SVD.

The data we are using is the merged data frame we created in our data prep with the columns ‘user’, ’name’, ‘flag’. Below is the code we implemented to run these algorithms and understand the RMSE.

In [772]:
```
benchmark = []
# Iterate over all algorithms
for algorithm in [SVD(), SVDpp(), NMF()]:
    # Perform cross validation
    results = cross_validate(algorithm, data, measures=['RMSE'], cv=3, verbose=False)

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], index=['Algorithm']))
    benchmark.append(tmp)
```

In [773]:
```
surprise_results = pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

In [774]:
```
surprise_results
```

Out[774]:

Algorithm	test_rmse	fit_time	test_time
SVDpp	0.008484	39.694466	1.526751
SVD	0.016601	21.993636	1.511943
NMF	0.097915	52.416549	1.346666

Based on these results we opted to proceed further with the SVD++ algorithm.

We then used the scikit-learn library to split the dataset into testing and training. Cross_validation.train_test_split shuffles and splits the data into two datasets according to the percentage of test examples, which in this case is 0.25. We will also use the accuracy metric of rmse. We’ll then use the fit() method which will train the algorithm on the train set, and the test() method which will return the predictions made from the testset

In [104]:
```
trainset, testset = train_test_split(data, test_size=0.25,random_state=0)
```

In [105]:
```
algo = SVDpp(n_factors=20,
    n_epochs=20,
    init_mean=0,
    init_std_dev=0.1,
    lr_all=0.007,
    reg_all=0.02,random_state = 0)
predictions = algo.fit(trainset).test(testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.0077
```

Out[105]:
```
0.007740230315223599
```

From this we can see that the model is generalized and the rmse score is also good.

In our code we conducted a deeper dive of our worst and best predictors to further inspect results. For additional details please reference line 108-109.

The new predictions code following this new matrix is below:

In [111]:
```
from collections import defaultdict
def get_top_n(predictions, n=10):
    '''Return the top-N recommendation for each user from a set of predictions.

    Args:
        predictions(list of Prediction objects): The list of predictions, as
            returned by the test method of an algorithm.
        n(int): The number of recommendation to output for each user. Default
            is 10.

    Returns:
    A dict where keys are user (raw) ids and values are lists of tuples:
        [(raw item id, rating estimation), ...] of size n.
    '''

    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n
```

For his method when we look for user1 we see the recommendation below

In [127]:
```
pd.set_option('display.max_colwidth', -1)
rec_df.loc[[user1]]
```

Out[127]:

	0	1	2	3	4	5
889dea4aeb8f72eb0a1f6c0c89f42f4f3d107898	(Best Buy GC - $100 Gift Card, 0.9974162176903099)	None	None	None	None	None

Collaborative Filtering using ALS on pySpark

Alternating least squares model:

Supervised model implemented in Apache Spark traditionally used for large scale collaborative filtering problems. The model works by holding the user matrix fixed and then running gradient descent with the user matrix. We chose to implement this model because it helped us solve issues with scalability as well as the spareness of the ratings data we uncovered within the data set. We also chose to evaluate this model using RMSE.

```
: for i in [0.01,0.03,0.1,0.3,1,3,10,100]:
    als = ALS(maxIter=5, regParam=i, userCol="new_user_id", itemCol="sku", ratingCol="flag",
          coldStartStrategy="drop")
    model = als.fit(training)
    predictions = model.transform(test)
    # Evaluate the model by computing the RMSE on the test data
    evaluator = RegressionEvaluator(metricName="rmse", labelCol="flag",
                                    predictionCol="prediction")
    rmse = evaluator.evaluate(predictions)
    print("Root-mean-square error = " + str(rmse))
```

```
Root-mean-square error = 0.8039689878472729
Root-mean-square error = 0.5660920786197627
Root-mean-square error = 0.5590686336163447
Root-mean-square error = 0.7992666118718785
Root-mean-square error = 0.9999438596107291
Root-mean-square error = 0.9999999999999419
Root-mean-square error = 1.0
Root-mean-square error = 1.0
```

From these results we used the best parameter (Root-mean-square error = 0.5590686336163447) to generate recommendations for each user. Below is the table of recommendations once we applied the code.

```
In [750]: user1_recommedations
```

Out[750]:

	sku	rating	name
0	9908156	0.937659	Best Buy GC - $50 It's Better To Receive Gift Card
480	9322814	0.923787	Best Buy GC - $100 Thank-You Gift Card
811	9322752	0.921582	Best Buy GC - $25 Thank-You Gift Card
1469	8234118	0.921043	Best Buy GC - $100 Gift Card
2491	8234092	0.918713	Best Buy GC - $50 Gift Card
3605	9759691	0.907641	Samsung - 24" Tall Tub Built-In Dishwasher - Stainless-Steel
3751	9759258	0.899840	Mad Men: Season Three [4 Discs] - Widescreen Subtitle AC3 Dolby - DVD
3859	8178063	0.895708	Best Buy GC - $150 Gift Card
4059	8234127	0.894972	Best Buy GC - $200 Gift Card
5115	9383115	0.894121	Best Buy GC - $20 Military Gift Card

To reiterate the steps we followed can be broken out below and can be observed in our code from the matrix factorization method utilizing the ALS.

Step 1. Data Splitting: Stratified Sampling
Step 2. Stochastic Gradient Descent; Starting with random numbers
Step 3. Predicting and optimizing/minimizing RMSE for already present ratings/flag in the matrix on Training Data
Step. 4 Prediction for the Test Data; can be seen from the resulting matrix of predictions

Conclusion & Future Work

Due to the challenges within the data sets provided we conducted several different types of analysis to build out our recommendation system. The following was used, collaborative filtering, using item and user-based similarity with the use of cosine similarity and pearson correlation coefficients. The use of model collaborative filtering, using SVDpp, SVD and NMF. Lastly collaborative filtering with MF specific to ALS. What we learned is that we cannot rely on matrix factorization.

Insights of Matrix created:

```
In [800]: print('Shape of Sparse Matrix: ', R_df.shape)
          print('Amount of Zero occurrences: ', (R_df == 0).sum(axis=1).sum())
          # Percentage of zero values
          density = (100.0 * (1-((R_df == 0).sum(axis=1).sum()) / (R_df.shape[0] * R_df.shape[1])))
          density
          print('density: {}'.format(density))

          Shape of Sparse Matrix:  (271937, 976)
          Amount of Zero occurrences:  265081364
          density: 0.1240146810763898
```

Due to the size of zero occurrences in the Best Buy dataset we moved on from implementing a specific modeling CF strategy. In an ideal situation we would advise any analysis using matrix factorization to have observable values for more than 25% of the data. We only have flags in our data, not any true ratings, therefore these, binary recommendation models can't work.

Our approach would be to deploy the model and capture users response which will then lead us to update and optimize the model, we cannot rely on just numbers we need to rely on the users response to help us flesh out a fully working model.

Implementation Platform

Stack

- Python
- Spark
- Etc

Modeling/Machine Learning

- Scikit-learn
- Pandas
- Numpy
- Spark ML - Lib
- Surprise
- Thinkstat
- Thinkplot

Data Visualization

- Plotly
- Interactive
- Matplotlib
- Seaborn