

Hotel Recommendation System

By

Aakash Yadav

&

Asif Khan

Post-Graduation Diploma in Statistical Methods and Analytics

2019-2020

Indian Statistical Institute

Chennai Center



In partial fulfilment of the PGDSMA program, Indian Statistical
Institute, Chennai Centre

DECLARATION

We hereby declare that the project work entitled “**Hotel Recommendation System**” submitted to the Indian Statistical Institute Chennai, is a record of an original work and this project work is submitted in the partial fulfillment of the requirements for the award of the Post-Graduation Diploma in Statistical Methods and Analytics. The results embodied in this Project have not been submitted to any other University or Institute for the award of any degree or diploma. This Project is carried out by us and is not copied from other sources. Wherever we have referred to the work from other sources, suitable citations and acknowledgements have been provided.

Aakash Yadav

Asif Khan

Acknowledgement

We would like to take this opportunity to acknowledge and thank Indian Statistical Institute for giving us this great opportunity to undertake this project which has been a great learning experience for both of us.

We would like to especially thank Dr. **Sampangi Raman** for being our supervisor of our project , guiding us and constantly support us even during this pandemic.

We express our special gratitude to all the faculty members, **Dr. S. M. Bendre, Dr. G. Ravindran, Dr. T. Karthick, Dr. A. Venkateswarlu, Dr. Surjit Pal, Dr. Anil Ghosh, Dr. Saurav Ghosh, Dr. Sudheesh Kumar Kattumannil** who gave us so much knowledge and we learned so many new things from them, we are really thankful to that. Lastly, we would like to thank our friends for helping us whenever we required their help.

Aakash Yadav

Asif Khan

Contents

Introduction.....	5
Aim.....	6
Data Description.....	7
Methodologies /Analysis	11
Collaborative Filtering (CF) with Neighbourhood Models	11
User-User Similarity	12
Item-Item Similarity	15
Collaborative Filtering(CF) with Matrix Factorization	16
SVD Factorization	17
Matrix factorization using textual reviews.....	19
Conclusion.....	29
Reference	30
Appendix	31

Introduction

In modern era of advanced web technologies most of the times online businesses have large number of products and customers have no time and patience to go through all the products to search for the particular product that they like. Recommendation systems tackles such problems by prioritizing the products on the basis of past responses given by different users. On the other hand, business organizations are strongly connected with their customers and they want to provide what is best for their customers and ultimate success for any business is satisfaction of their customers. So, the need of a Recommendation System is must in any industry so that the customer will get what they need and the businesses get such customers whom they can take care of best in their own way. Tourists all around the world are always looking for the best hotels where they will be satisfied most and hotels want such customers whom they can provide what customers looking for. A group of customers can like a particular hotel but it does not mean the other group of customers will like that particular hotel also. So, we need to make such a recommendation system which recommends best hotels to every individual tourist on the basis of their historical data.

Aim

Aim of our project is to learn and compare different Collaborative Filtering based recommendation systems to recommend hotels to users.

Data Description

We obtained our data from <https://www.kaggle.com/datafiniti/hotel-reviews>

Column Name	Description
id	Unique id identifying column
dateAdded	The date this business was first added to the business database
dateUpdated	The most recent date this business was updated or seen by the system
address	The physical street address for this hotel location.
categories	A list of category keywords used for this hotel across multiple sources.
primaryCategories	A list of standardized categories to which this hotel belongs
city	The city of this hotel location.
country	The two-letter country code for the hotel location's country.
keys	The keys field is used to merge raw data from individual sources into the master Datafiniti record
latitude	Latitude of hotel location
longitude	Longitude of hotel location
name	Name of hotel
postalCode	The postal or zip code of the hotel' location
province	The province or state for this hotel location
reviews.date	The date the review was posted
reviews.dateSeen	A list of dates when the review was seen
reviews.rating	A 1 to 5 star value for the review.
reviews.sourceURLs	URL of review's source
reviews.text	Textual review
reviews.title	Title of Review
reviews.userCity	The reviewer's city.

reviews.userProvince	The reviewer's province or state.
reviews.username	The reviewer's username
sourceURLs	A list of URLs used to generate data for this hotel location
websites	A list of websites for the hotel

Data Exploration

- Finding sum of nan values in data.

```
pd.concat([df1, df2]).isnull().sum()

Out[7]: id                0
      dateAdded           0
      dateUpdated         0
      address             0
      categories          0
      primaryCategories    0
      city                0
      country             0
      keys                0
      latitude            0
      longitude           0
      name                0
      postalCode          0
      province            0
      reviews.date        0
      reviews.dateAdded   20000
      reviews.dateSeen    0
      reviews.rating      0
      reviews.sourceURLs  0
      reviews.text        1
      reviews.title       2
      reviews.userCity    5836
      reviews.userProvince 7297
      reviews.username    0
      sourceURLs           0
      websites            0
      dtype: int64
```

It can be seen that we have lot of null values in 'reviews.userCity', 'reviews.userProvince' and 'reviews.dateAdded' columns, so we can not use these columns. All the columns except 'reviews.rating', 'reviews.text', 'name', 'reviews.username' have been removed because all other columns have no significance in our models.

- Number of Users, Hotels and Places:

```

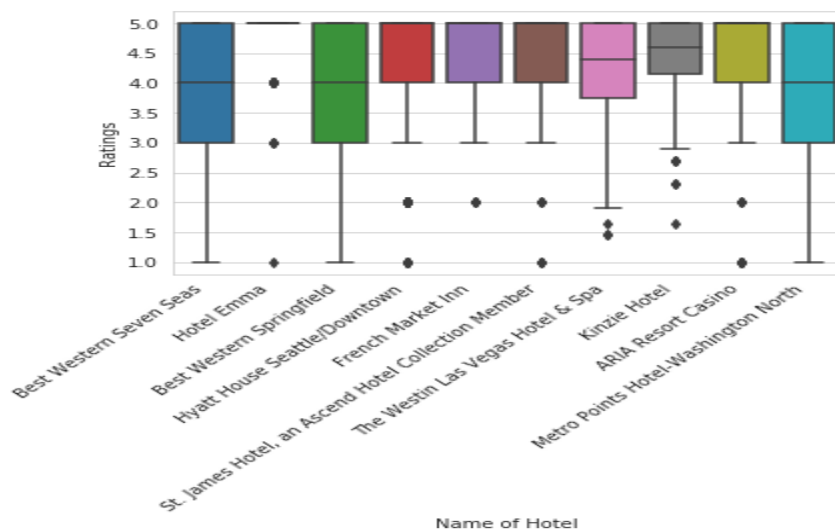
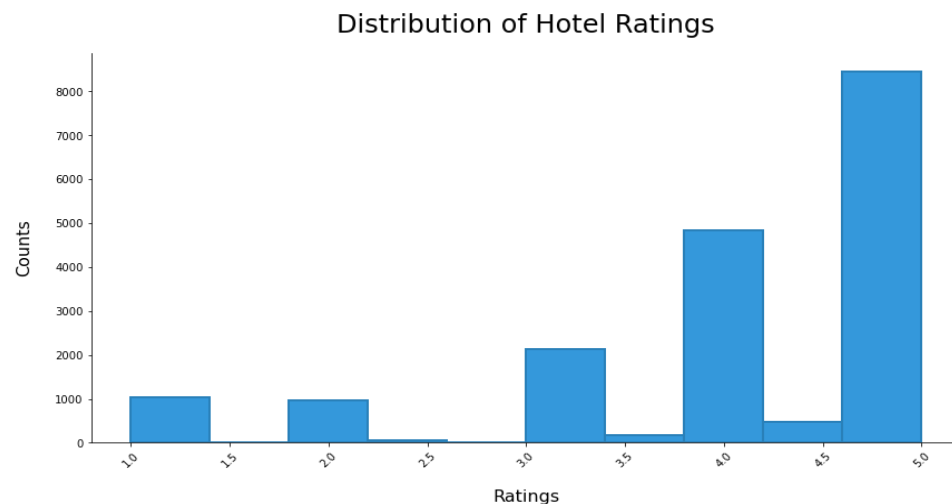
pd.concat([df1, df2])[['name', 'reviews.username', 'country']].nunique()

Out[9]:
name          2764
reviews.username 15587
country         1
dtype: int64

```

There are total of 2764 unique hotels in our dataset and 15587 unique users. Which tells us that most of the users do not give multiple reviews. So, we have to divide our data set in training and testing dataset carefully. Also all of hotels are situated at one Country i.e. 'U.S.A'.

- Distribution Plot of Ratings:



Above box plot is the plot of ratings given to hotels with highest average ratings. It is clear from distribution plot and box plot that users tend to give high ratings more often.

```
a=dfconcat.groupby('Users')['Hotel_Name'].count()
df_reduced=dfconcat[dfconcat['Users'].isin(a[a>=2].index)]
df_1review=dfconcat[dfconcat['Users'].isin(a[a==1].index)]
df_reduced.head()
```

	Users	Hotel_Name	Rating	Reviews
16	103bennier	Days Inn Jackson	4.0	Even though the building is older, it was dev...
17	103bennier	Days Inn-Jackson	4.0	Even though the building is older, it was dev...
26	112traveler47	Aloft Buffalo Airport	5.0	I've reviewed this hotel before as I travel e...
27	112traveler47	The St. Regis New York	5.0	This was our second stay at the St. Regis, an...
648	A	Best Western Twin Islands	3.0	It wasn't our first time at this motel. We li...

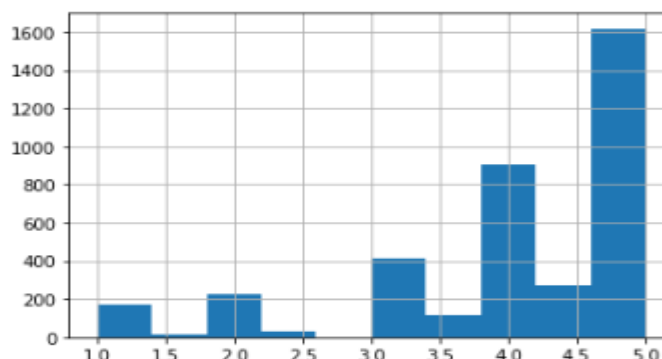
```
#IMP
df=df_reduced
print(len(df_reduced))
print(len(df_1review))
```

```
3769
14398
```

We have removed users who have only 1 rating(14398). It has reduced the rows in our dataset to 3769. We will be using our dataset of just one review later in the project for now we will continue with df_reduced dataset.

```
In [100]: df['Rating'].hist()
```

```
Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x248c4286>
```



Above is the histogram of rating in reduced dataset. Most of the rating are greater than equal to 3 with majority of them equal to 5.

```
from sklearn.model_selection import train_test_split
train0,test0=train_test_split(df,test_size=0.4,random_state=42)
test0=test0[test0['Users'].isin(train0['Users'])]
test0=test0[test0['Hotel_Name'].isin(train0['Hotel_Name'])]
```

We have divided our dataset into train(train0) and test (test0)

Methodologies /Analysis

In this project we have focussed on collaborative methods of recommender systems. Under collaborative methods there are majorly 2 methodologies neighbourhood models and matrix factorization. We used user-user and item-item neighbourhood models. We have used 4 type of algorithms. First 3 algorithms just use user and hotel ratings. Last algorithm uses both rating and textual reviews given by user. The algorithms are:

- User-User neighborhood model
- Item -Item neighborhood model
- SVD Factorization
- Textual reviews factorization.

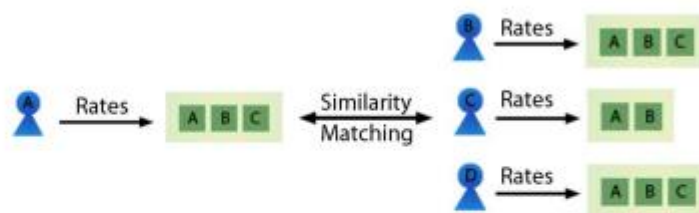
If r is the actual rating and \hat{r} is the predicted rating then evaluation metric used is RMSE(Root Mean Square Error).

$$RMSE = \{\Sigma[(\hat{r} - r)^2/n]\}^{0.5}$$

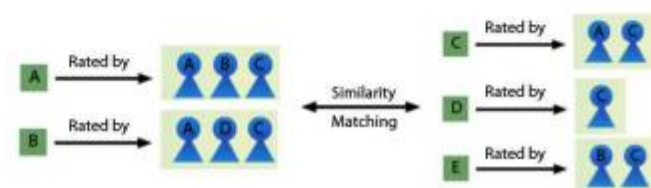
Collaborative Filtering (CF) with Neighbourhood Models

The key idea behind CF with neighbour models is that similar users that share the same interest and that similar items are liked by a user.

User based approach:



Item based approach:



User-User Similarity

Assume there are m users and n items, we use a matrix with size $m \times n$ to denote the past behaviour of users. Each cell in the matrix represents the associated opinion that a user holds. For instance, $r(i, j)$ denotes how user i likes item j . Such matrix is called **utility matrix**. CF is like filling the blank (cell) in the utility matrix that a user has not seen/rated before based on the similarity between users or items.

Let s_{ij} denotes the similarity between user i and user j and r_{ij} denotes the rating that user i gives to item j .

$$\cos(u_i, u_j) = (\sum r_{jx} \cdot r_{ix}) / (\sum r_{jx}^2 \cdot \sum r_{ix}^2)$$

But there's a problem with this formula of cosine similarity . Below is an example :

Consider a utility matrix with 4 users and 7 items.

User/Item	H1	H2	H3	H4	H5	H6	H7
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

To calculate similarity matrix we replace missing values with 0

User/Item	H1	H2	H3	H4	H5	H6	H7
A	4	0	0	5	1	0	0
B	5	5	4	0	0	0	0
C	0	0	0	2	4	5	0
D	0	3	0	0	0	0	3

Similarity between A and B is $S(A,B)=0.38$

Similarity between A and B is $S(A,C)=0.32$

On observing the data we see that A and B have quite similar taste's as compared to B and C. But if calculate rating user A would give to H2 is 0 although both A and B give both high rating to H1 which shows they are quite similar and A has given 5 rating to H2.

Also similarity between A and B is greater than similarity between A and C but not by that much as were expecting. The reason is because missing's are acting as negative's.

To improve this we centered the rating of each user around it's mean. We subtract each entry in a row with it's row mean. Then replace the missing values with 0. We have made replacing missing data with 0 as neutral.

User/Item	H1	H2	H3	H4	H5	H6	H7
A	2/3			5/3	-7/3		
B	1/3	1/3	-2/3				
C				2	1/3	4/3	
D		0					0

Similarity between A and B is $S(A,B)=0.09$

Similarity between A and B is $S(A,C)=-0.56$

Now similarity between A and B is greater than similarity between A and C by good margin. Positive rating indicates user like the item more than average and negative rating indicates user like the item more than average and magnitude tells the magnitude or like or dislike.

Now, we can predict the users' opinion on the unrated items with the below equation. We use the ratings of K most similar users to user i.

$$r_{xi} = (\sum s_{ij} \cdot r_{xj}) / (\sum s_{ij})$$

Some users are very hard raters whereas some very soft. Some items are very popular so for these items we get a popularity bias. To overcome this issue we use Global Baseline method. Below is the formula.

$$r_{xi} = b_{xi} + (\sum s_{ij} \cdot r_{xj}) / (\sum s_{ij})$$

b_{xi} is the baseline estimate for r_{xi} , u is overall average rating, b_x is rating deviation of user x (avg rating of user x - u) and b_i is rating deviation of item i

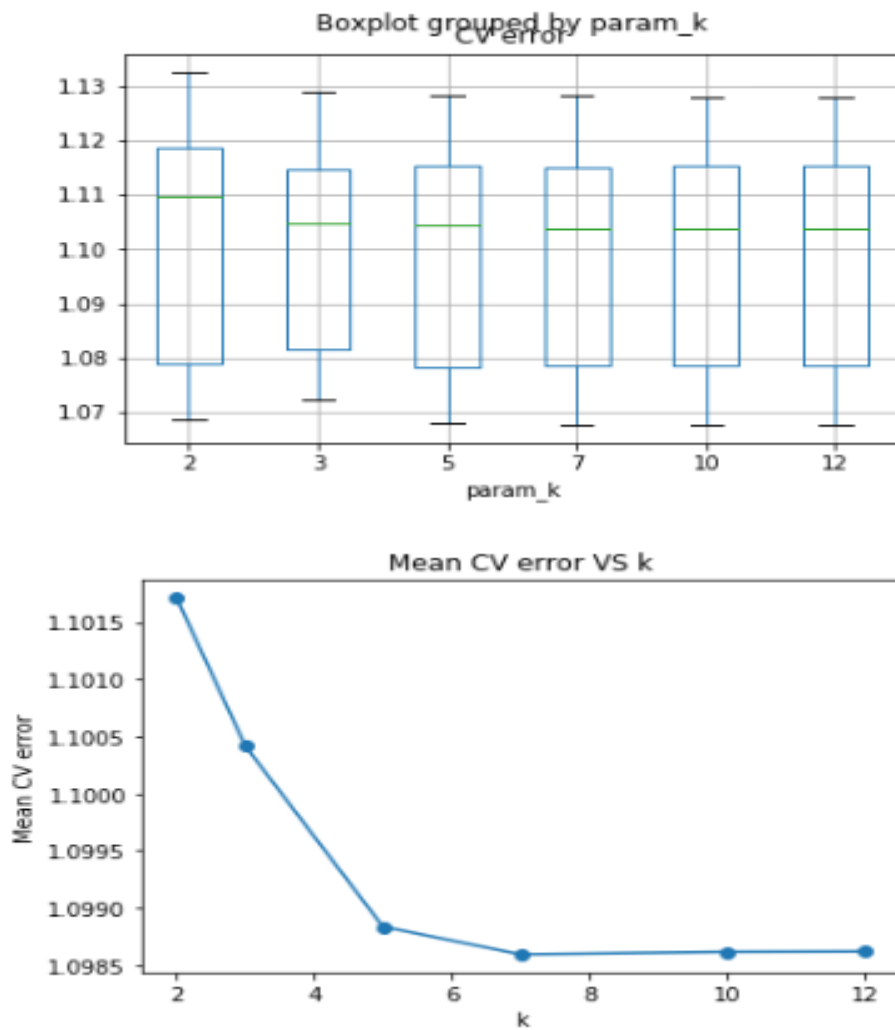
$$b_{xi} = u + b_x + b_i$$

```

: #Parameters
param_grid = {'k': [2,3,5,7,10,12], 'sim_options':{'name': ['cosine'], 'user_based': [True]}}
grid_search = GridSearchCV(KNNBaseline, param_grid, measures=['rmse'], cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_dataframe = pd.DataFrame(grid_search.cv_results_)

```

We have done hyper parameter tuning of our model with CV=5



Variation in CV error is almost same we select k=7 and we have selected min_k as 1.

```
#Training model
algo.fit(trainset)

#Accuracy training predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error user-user ', end=' ')
accuracy.rmse(training_predictions)

#Testing error
pred_user_user=algo.test(testset_surprise.build_testset())
print('Testing error user-user ', end=' ')
accuracy.rmse(pred_user_user)
print('Best Parameters for user-user ', end=' ')
print(grid_search.best_params['rmse'])

Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Training error user-user      RMSE: 0.6830
Testing error user-user      RMSE: 1.0914
Best Parameters for user-user  {'k': 7, 'sim_options': {'name': 'cosine', 'user_based': True}}
```

Training and testing error are 0.683 and 1.0914

Item-Item Similarity

It is just dual of user-user similarity algorithm here we compute similarities between items and then predict the rating.

ITEM ITEM

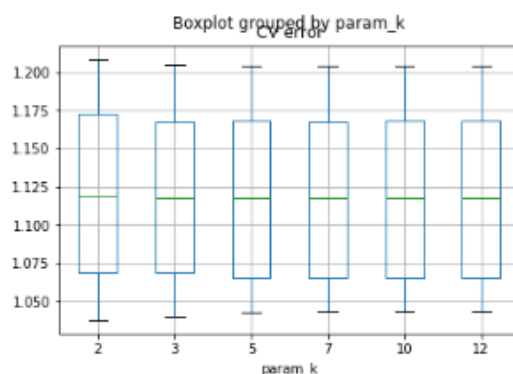
```
#Parameters
param_grid = {'k': [2,3,5,7,10,12], 'sim_options': {'name': ['cosine'], 'user_based': [False]}}
grid_search = GridSearchCV(KNNBaseline, param_grid, measures=['rmse'], cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_dataframe=pd.DataFrame(grid_search.cv_results_)
```

We have done hyper parameter tuning of model with CV=5

```
Done computing similarity matrix.

In [75]: a=param_dataframe[matches].set_index('param_k').stack().reset_index().rename(columns={0:'CV error'})
a.boxplot(column='CV error',by='param_k')

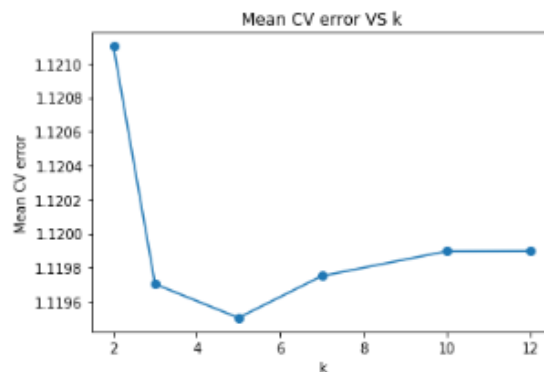
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x1b659c5f788>
```



Variation of CV error is almost same with all k values but mean error is minimum at k=5 and we have selected min_k as 1.

```
param_k

In [76]: b=param_dataframe[['mean_test_rmse','param_k']]
plt.plot('param_k','mean_test_rmse',data=b,marker="o")
plt.title("Mean CV error VS k")
plt.xlabel("k")
plt.ylabel("Mean CV error")
plt.show()
```



```
In [77]: #Trainingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error item-item ', end=' ')
accuracy.rmse(training_predictions)

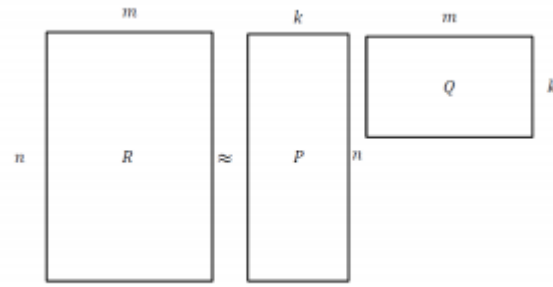
#Testing error
pred_item_item=algo.test(testset_surprise.build_testset())
print('Testing error item-item ', end=' ')
accuracy.rmse(pred_item_item)
print('Best Parameters for item-item ', end=' ')
print(grid_search.best_params['rmse'])

Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Training error item-item    RMSE: 0.7214
Testing error item-item    RMSE: 1.1175
Best Parameters for item-item    {'k': 5, 'sim_options': {'name': 'cosine', 'user_based': False}}
```

Training and testing error are 0.7214 and 1.1175

Collaborative Filtering(CF) with Matrix Factorization

One method for predicting user ratings based on past rating is through matrix factorization. This model assumes that user ratings are the result of a few factors. For example, we assume that there are only a handful of reasons that account for a customer rating a hotel . Perhaps the service, the quality of the food in hotel restraunt , spa, gym etc to account for all user ratings. That is, as these factors differ from business to business, the ratings of customers differ accordingly.



Matrix factorization methods assume that all the information in the utility matrix can be accounted for by some k (latent) factors. If n is the number of users and m is the number of items, the $n \times m$ utility matrix can be factored as $R = PQ$, where P is $n \times k$ and Q is $k \times m$. To find such a P and Q is to find the matrices that when multiplied together are as close to rating R as possible. Specifically, if $S = PQ$, then the root mean squared error, the error that should be minimized

SVD Factorization

SVD is a in which any matrix can be decomposed into three matrices.

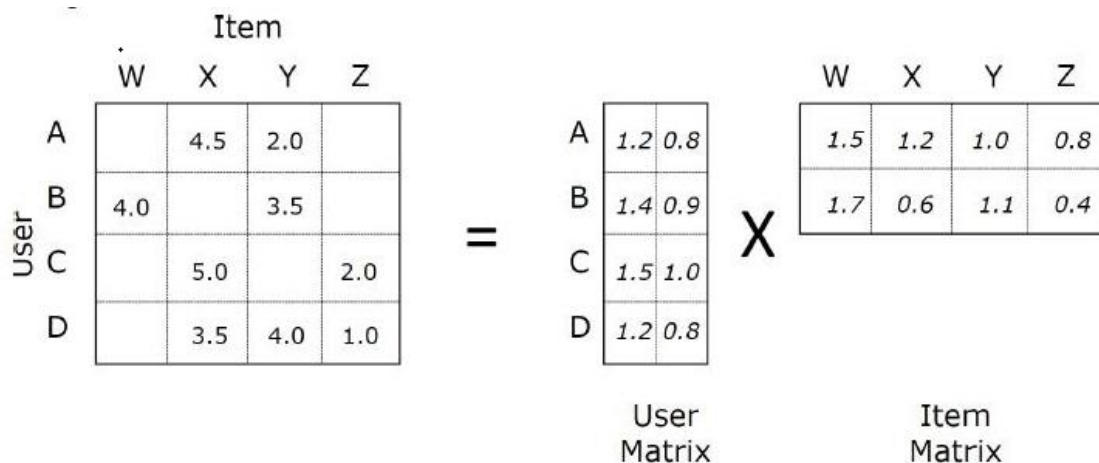
$$\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}_{m \times n} \approx \begin{pmatrix} u_{11} & \dots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}_{m \times r} \begin{pmatrix} s_{11} & 0 & \dots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}_{r \times r} \begin{pmatrix} v_{11} & \dots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}_{r \times n}$$

If $X(m \times n)$ is a utility matrix, $U(m \times r)$ is a orthogonal left singular matrix, representing the relationship between users and latent factors. S is a diagonal matrix describing the strength of each latent factor, while $V(r \times n)$ transpose is a orthogonal right singular matrix, indicating the similarity between items and latent factors.

SVD maps each user and item into r -dimensional latent space. This mapping facilitates a representation of relationship between users and items.

SVD has a very nice property that it minimizes RMSE. But it does so when it is completely filled. Since our utility matrix has missing values we replace those by 0 and calculate U, S, T, P and Q and can be calculated as.

$$P = U * S^{1/2} \quad \& \quad Q = S^{1/2} * V^T$$



Suppose, M is our Utility Matrix, P is our User Matrix and Q is our Hotel Matrix. If we subtract the real ratings(R) with the predicted ratings (P * Q) we will get error. So, we need to minimize the error function. And to avoid over fitting we add regularization . Here lambda is penalizing parameter and optimize using stochastic gradient.

$$\min_{P \in \mathbb{R}^{m \times k}, Q \in \mathbb{R}^{k \times n}} \sum_{(i,j) \in K} (M_{i,j} - (PQ)_{i,j})^2 + \lambda (\|P\|^2 + \|Q\|^2).$$

```
#Grid Search and finding best parameters for SVD
param_grid = {'n_factors':[10,12,15], 'n_epochs':[10,15], 'reg_all':[0.02,0.05], 'lr_all':[0.01,0.02]}
#param_grid={}
grid_search = GridSearchCV(SVD,param_grid,measures=['rmse'],cv=5)
grid_search .fit(data)
algo = grid_search.best_estimator['rmse']
param_df=pd.DataFrame.from_dict(grid_search.cv_results)
param_df.head()
```

```
#Trainingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error SVD ', end=' ')
accuracy.rmse(training_predictions)

#Testing error
pred_svd=algo.test(testset_surprise.build_testset())
print('Testing error SVD ', end=' ')
accuracy.rmse(pred_svd)
print('Best Parameters for SVD ', end=' ')
print(grid_search.best_params['rmse'])

Training error SVD    RMSE: 0.8161
Testing error SVD    RMSE: 1.0437
Best Parameters for SVD    {'n_factors': 12, 'n_epochs': 15, 'reg_all': 0.05, 'lr_all': 0.01}
```

After hyper parameter tuning

latent factor= 12

Regularization parameter is 0.05

Learning rate is 0.01

Max iteration for stochastic gradient is 15

Matrix factorization using textual reviews

Till now we have used just ratings of users and hotels. Now we will use ratings and reviews both.

We break down the large matrix of ratings from users and hotels into two smaller matrix of User-Feature and Hotel-Feature. In our recommendation system we find these User-Feature and Hotel-Feature by using reviews given by users and reviews received by hotels.

In order to use reviews, we first convert raw textual reviews into structured form to extract opinion-based feature i.e. we need to clean up the reviews first. This can be done by applying natural language processing.

We have performed following steps to process a natural language text review as follows:

- **Tokenizing:** Tokenization is the process of breaking a document down into words, punctuation marks, numeric digits, etc. All of the reviews are available in the form of paragraph which contains different characters, white-spaces, strings etc. These strings are tokenized into tokens. These tokens are usually words but can also be numbers or symbols. For this we have used `spacy.load('en_core_web_sm')` and removed all whitespace, numbers and symbols from the sentences. spaCy is free publicly available library and it comes with a bunch of prebuilt models where the 'en' we used is one of the standard ones for english. spaCy tokenizes the text, i.e. segments it into words, punctuation and so on. This is done by applying rules specific to each language. For example, punctuation at the end of a sentence should be split off, whereas "U.K." or "U.S.A." should remain one token. First, the raw text is split on whitespace characters. Then, the tokenizer processes the text from left to right. On each substring, it performs two checks, Does the substring match a tokenizer exception rule? For example, "don't" does not contain whitespace, but should be split into two tokens, "do" and "n't", while "U.K." should always remain one token. Can a prefix, suffix or infix be split off? For example punctuation like commas, periods, hyphens or quotes. If there's a match, the rule is applied and the tokenizer continues its loop, starting with the newly split substrings.

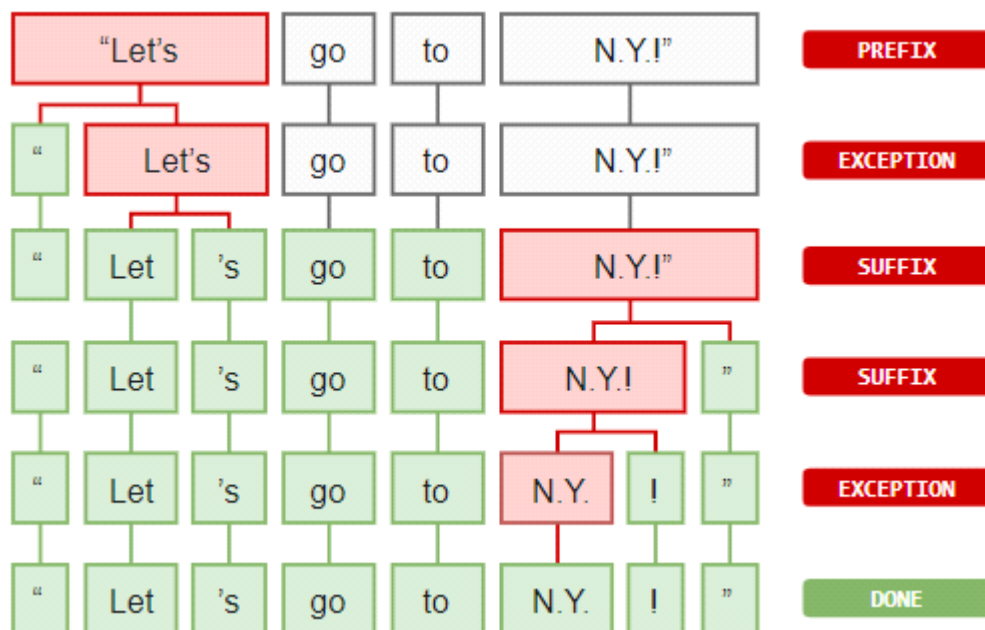


image credits spacy.io

This way, spaCy can split complex, nested tokens like combinations of abbreviations and multiple punctuation marks.

For further reads use spacy documentation(<https://spacy.io/usage/spacy-101>)

- **Lemmatization:** Lemmatization is the process of converting a word to its base form. It considers the context and converts the word to its meaningful base form, For example, lemmatization would correctly identify the base form of 'eating' to 'eat'. But sometimes, the same word can have multiple different 'lemma's. In such case we should identify the 'part-of-speech' (POS) tag for the word in that specific context and extract the appropriate lemma. spaCy comes with in built class for Lemmatization. spaCy determines the part-of-speech tag by default and assigns the corresponding lemma.
- **Stop-words removal:** This step involves removal all the stop-words (List of most common words of a language that are often useful to filter out, for example "have", "do", "I", "you", "he", etc). There is no such stop-words library which is globally approved but we are using the library from `spacy.lang.en.stop_words` for the stop-words. After the processing of each textual review and removing unnecessary stop words we join these processed tokens to build up the string again for our next step.

reviews before processing

```

4 Cool place in downtown! I've been to all of t...
5 The Roosevelt seeps a cool vibe that can only ...
6 Glad to say this place is still going strong a...

```

reviews after processing

```

4 cool place downtown iteration restaurant prett...
5 Roosevelt seep cool vibe attain convert histor...
6 glad place strong little year visit review Wan...

```

In order to use the processed reviews, for each user, combine all the processed reviews to form a single paragraph. The similar approach for each Hotel. This way we get two Data Frames, one having all users with all of their processed reviews calling it "User Reviews", second data frame having all Hotels with all of their processed reviews calling it "Hotel Reviews".

```

userid_df = train1[['username', 'cleaned']]
business_df = train1[['name', 'cleaned']]
userid_review = userid_df.groupby('username').agg({'cleaned': ' '.join})
hotel_review = business_df.groupby('name').agg({'cleaned': ' '.join})
userid_review

```

Out[12]:

username	cleaned
103bennier	building old devorate nicely holiday room clea...
112traveler47	review hotel travel extensively season ONLY ai...
A	terrible staff hateful air condition work room...
A S	quiet restful helpful service friendly people ...
A Traveler	stay night day touring driving staff friendly ...
...	...
susan	weekend trip attend Gonzaga hotel great offer ...
taysiewhua	draw BB pic hall website stay hall open event ...
vincent	Interior Renovations design nice comfortable
yuliangelc	awesome place gladly surprise staff nice frien...
z	clean smell good breakfast ok stay overnight f...

1057 rows x 1 columns

```

userid_df = train1[['username', 'cleaned']]
business_df = train1[['name', 'cleaned']]
userid_review = userid_df.groupby('username').agg({'cleaned': ' '.join})
hotel_review = business_df.groupby('name').agg({'cleaned': ' '.join})
hotel_review

```

Out[13]:

name	cleaned
250 Main Hotel	stay day past weekend jewel Rockland maine sta...
AC Hotel Chicago Downtown	bad need security pool bad experience young gi...
AC Hotel by Marriott Boston Downtown	bad change thing good staff accommodate share ...
APPLE Inn	place great outside look like old style hotel ...
ARIA Resort Casino	come weekend getaway friend time Aria try prop...
...	...
Wyndham Houston - Medical Center Hotel and Suites	stay great restful night plan stay month ok Ho...
Wyndham Santa Monica At The Pier	MoreMore
XV Beacon	thoroughly enjoy stay XV Beacon staff friendly...
dana hotel and spa	bad rooftop bar style substance enjoy view sur...
hampton inn Springfield southeast	excellent accommodation excellent staff super ...

943 rows x 1 columns

- TF-IDF Generation: Term Frequency (TF): The number of times a word appears in a document divided by the total number of words in the document. Every document has its own term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Data Frequency (IDF): The log of the number of documents divided by the number of documents that contain the word w . Inverse data frequency determines the weight of rare words across all documents in the corpus.

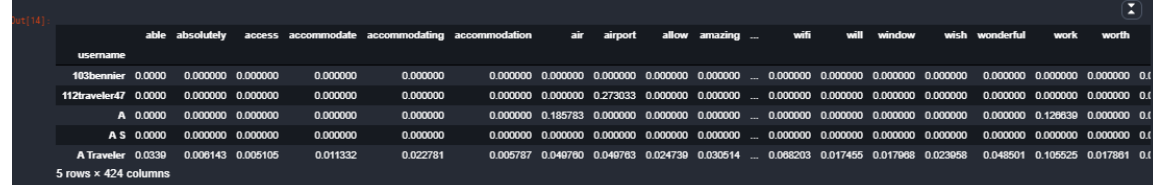
$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

The TF-IDF is simply the TF multiplied by IDF.

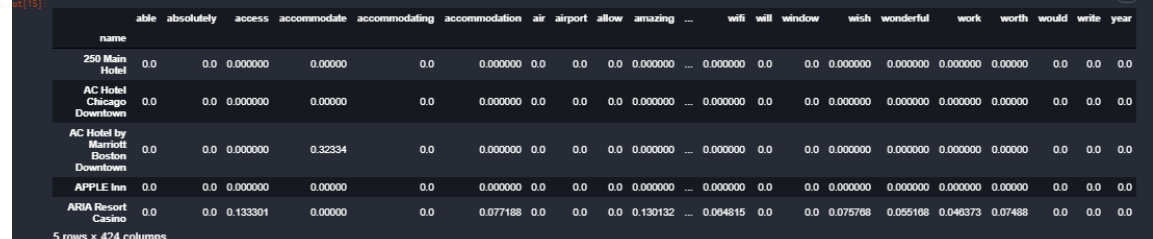
Tf-Idf (term frequency -inverse document frequency) is generated for all reviews by each guest/user. We apply TFIDF Vectorizer on both of the data frames ("User Reviews", "Hotel Reviews") to extract the features from the processed reviews.

Rather than manually implementing TF-IDF ourselves, we could use the class provided by sklearn.

```
from nltk.tokenize import WordPunctTokenizer
user_vectorizer=TfidfVectorizer(tokenizer = WordPunctTokenizer().tokenize, ngram_range=(1,3), max_df= 0.9, min_df= 0.03, max_features=5000)
user_vectorized=user_vectorizer.fit_transform(user_review['cleaned'])
user_vectorized_df=pd.DataFrame(user_vectorized.toarray(), index=user_review.index, columns=user_vectorizer.get_feature_names())
user_vectorized_df=user_vectorized_df.drop(labels=['sincerely', 'river', 'san diego', 'french quarter', 'french', 'best', 'lounge', 'resort', 'quarter', 'hy
user_vectorized_df.head()
```



```
hotel_vectorizer=TfidfVectorizer(tokenizer = WordPunctTokenizer().tokenize, ngram_range=(1,3), max_df= 0.9, min_df= 0.03, max_features=5000)
hotel_vectorized=hotel_vectorizer.fit_transform(hotel_review['cleaned'])
hotel_vectorized_df=pd.DataFrame(hotel_vectorized.toarray(), index=hotel_review.index, columns=hotel_vectorizer.get_feature_names())
feat=user_vectorized_df.columns
hotel_vectorized_df=hotel_vectorized_df[feat]
hotel_vectorized_df.head()
```



Parameters of TFIDF vectorizer:

tokenizer: With the help of nltk.tokenize.WordPunctTokenizer() method, we are able to extract the tokens from string of reviews.

ngram_range: With this we can also provide n-gram range i.e. the lower and upper boundary of the range of n-values for different n-grams to be extracted from each text or review. For example an ngram_range of (1, 1) means only unigrams i.e. we take only one word as a feature for nstance 'food', (1, 2) means unigrams and bigrams i.e. we can have one word as well as two words as a feature for instance 'delicious food'. We have used ngram range of (1,3) for both data frames.

max_df: When building the vocabulary this helps in ignoring terms that have a document frequency strictly higher than the given threshold and helps in dealing with high frequency words that were not in stopwords list. It is in range of [0.0, 1.0]. We have taken it to be 0.95

min_df: When building the vocabulary this helps in ignoring terms that have a document frequency strictly lower than the given threshold and helps in dealing with very low frequency words. It is in range of [0.0, 1.0]. We have taken it to be 0.03

max_features: This helps in building a vocabulary that only consider the top features ordered by term frequency across the corpus this also helps us to match the dimensions of the matrixes created by applying TFIDF on "User Reviews" and "Hotel Reviews". We have taken its value to be 5000.

Let the matrix that we obtain from applying TFIDF vectorizer on "User_vec" be "user_vectorized", where rows represent the Users Id and columns represent the vocabulary/words generated by TFIDF vectorizer and the matrix we obtain from applying TFIDF vectorizer on "Hotel Reviews" be "hotel_vectorized", where rows represent the Hotels Name and columns represent the vocabulary/words generated by TFIDF vectorizer. Both of the matrix have same columns.

Predicting Ratings of train data: Now, we can multiply "user_vectorized" matrix with transpose of "hotel_vectorized" matrix to predict the ratings that a user gives to a restaurant and calling it "Predicted Rating" matrix.

Original Ratings: The "Original Ratings" matrix can be obtained by pivoting the training data frame by "User Id" as rows and "Hotel Name" as columns and taking ratings as values. We also set values to take mean value, in case if user has given multiple ratings to a particular hotel. For this purpose we use pivot_table class in Pandas library. Pandas is also a very useful library and available for free.

Error Calculation: Error can be calculated by taking difference of "Original Ratings" matrix and "Predicted Rating" matrix and then taking square of it, we get the LSE(Least Square Error) as our loss function.

Error Minimizing: Now we need to update the values in the "user_vectorized" and "hotel_vectorized" matrix in such a manner that it minimizes the Error. For error minimization we have used Gradient Descent Algorithm. To overcome the over fitting in our model we penalize our model by adding regularization to our loss function. At each epoch of our optimization algorithm, we are updating values in "User Features" and "Hotel Features" matrix. We need to do hyper-parameter tuning by updating changing number of epochs, learning rate of gradient descent (gamma) and regularization rate (lambda). Suppose A is our 'user_vectorized' matrix, B transpose of our 'hotel_vectorized' matrix and R is the original rating martix of train set. Error is minimized by using:

P has dimensions (i,k)

Q has dimensions (k,j)

R has dimension (i,j)

i is number of users, k is number of features and j is number of hotels.

Least square error is calculated as

$$\text{Error} = \sum \sum [(R_{ij} - P_i \cdot Q_j)^2 + \lambda[||P_i||^2 + ||Q_j||^2]]$$

Optimization is done by looping the below function

$$E_{ij} = R_{ij} - P_i \cdot Q_j$$

$$P_{i \text{ new}} = P_i + \gamma(E_{ij} \cdot Q_j - \lambda P_i)$$

$$Q_{j \text{ new}} = Q_j + \gamma(E_{ij} \cdot P_i - \lambda Q_j)$$

Loop is run for several epochs.

```
def matrix_factorization(R, P, Q, steps=20, gamma=0.001, lamda=0.02):
    for step in range(steps):
        print(step)
        m=0
        for i in R.index:
            m+=1
            if m%10==0:
                print('=', end = "")
            for j in R.columns:
                if R.loc[i,j]>0:
                    eij=R.loc[i,j]-np.dot(P.loc[i],Q.loc[j])
                    P.loc[i]=P.loc[i]+gamma*(eij*Q.loc[j]-lamda*P.loc[i])
                    Q.loc[j]=Q.loc[j]+gamma*(eij*P.loc[i]-lamda*Q.loc[j])
        return P,Q
```

This algorithm will give out updated P,Q matrix and we save it our hard disk to use it later.

Testing Model: In our test set we have such users and hotels which were available in our training set of data. Our predicted values can be obtained by taking matrix multiplication of updater P and Q matrix. For checking the accuracy, we use RMSE (Root Mean Squared Error).

First we check our training error

```
[32]: ##calculating train error
org_rate = pd.pivot_table(train1, values='reviews.rating', index=['username'], columns=['name'])
pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,columns=hotel_vectorized_df.index)
pred_vec=pred.values.flatten()
org_vec=org_rate.values.flatten()
train_error_df=pd.DataFrame({'org':org_vec,'pred':pred_vec})
train_error_df=train_error_df.dropna(axis=0)
from sklearn.metrics import mean_squared_error as mse
mse(train_error_df['org'],train_error_df['pred'])

Out[32]: 1.7806177040081177
```

Then we check our test error.

```
(mse(pred_org_df['org_ratings'],pred_org_df['predicted']))**(1/2)

Out[76]: 1.9616142777873424
```

Recommendation: If RMSE is sufficiently low, then we can use our model to predict how much rating a user will give to a hotel and on that basis we can recommend the user those hotels which he/she is predicted to give higher ratings.

```
[60]: def recommend(username):
      predict_rate=P.dot(Q.T)
      return (predict_rate.loc[username].sort_values(ascending=False)[0:5])

recommend('susan')

Out[61]: name
Hotel Emma          4.660224
ARIA Resort Casino  4.288770
Mandarin Oriental, Miami  4.272781
Hampton Inn & Suites Orlando at SeaWorld  4.218053
Grand Hyatt Seattle  4.203754
Name: susan, dtype: float64
```

Here top five hotels are recommended to a user called 'susan' along with the predicted rating.

Below is the summary of using all the algorithms on our data

Algo	Train error(RMSE)	Test error(RMSE)
User-User Similarity	0.6885	1.0912
Item-Item Similarity	0.7199	1.1178

Matrix Factorization(SVD)	0.7615	1.0436
Matrix Factorization(Textual Reviews)	0.1	2.55

All the algorithms have low train error and very high test error. Especially Matrix factorization using textual reviews.

We have updated the train dataset by adding those rows which were removed as there were a high number of users which were removed as they had reviewed only once.

```
#Now training with bigdata
train_small=train0
train0=pd.concat([train0,df_1review])
```

Note : We are just adding data in our training set we have not changed our test set.

We have tried to see if the users which were removed earlier (removed because those users had only one rating in whole dataset) are added into the training dataset would it change our test error.

After using our new large training data set the test error for matrix factorization using textual reviews was reduced to 1.959.

Algo	Train error(RMSE)	Test error(RMSE)
User-User Similarity	0.5532	1.3404
Item-Item Similarity	0.2636	1.0768
Matrix Factorization (SVD)	0.743	1.0121
Matrix Factorization (Textual Reviews)	0.2645	1.959

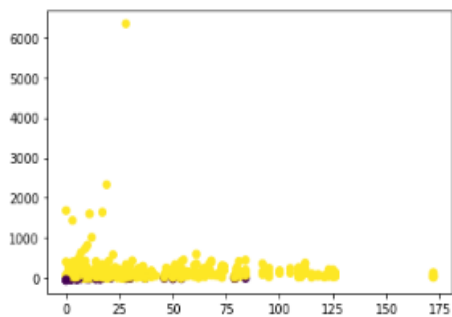
Below is a scatter plot in which each bubble represents rating in test set. When we increased our training data then the no of hotel reviews increased for some hotels. X axis shows the increase in hotel reviews for every data point in test set. Y axis shows the percentage change in estimated rating from small data set and large data set for algorithm number 4(matrix factorization using reviews). Colour represents whether the change is towards actual rating or not. As it is clear from image estimated rating has improved for most of the bubbles (test set data point). For some data points estimated ratings have improved significantly.

```
In [90]: forscatter=Analysistable[['Users','Hotel_Name','pct chage in rating','Pos/Neg','Increase in hotels','Increase in Users']]
forscatter.head()
```

```
Out[90]:
```

	Users	Hotel_Name	pct chage in rating	Pos/Neg	Increase in hotels	Increase in Users
0	Richard M	Luana Waikiki Hotel & Suites	52.427402	1.0	118	0
1	Richard M	Hyatt Regency Grand Cypress	-4.578784	-1.0	17	0
2	Richard M	Best Western Plus St Charles Inn	58.376555	1.0	58	0
3	A TripAdvisor Member	The Orchard Garden Hotel	81.295088	1.0	69	0
4	A TripAdvisor Member	TownePlace Suites Anaheim Maingate Near Angel ...	77.408854	1.0	80	0

```
In [94]: #Scatter plot Increase in hotels vs pct chage in rating
plt.scatter(forscatter['Increase in hotels'],forscatter['pct chage in rating'],c=forscatter['Pos/Neg'])
plt.xlabel="Increase in hotels"
plt.show()
```



```
: #Number of Predictions improved and got
forscatter[['Pos/Neg']].groupby('Pos/Neg')['Pos/Neg'].count()
```

```
: Pos/Neg
-1.0    132
 0.0     1
 1.0   822
Name: Pos/Neg, dtype: int64
```

For 132 test cases estimated ratings have gone further away from actual rating . One remained same and 822 have move towards actual rating

Conclusion

Below is the overall summary using all the methods on both training datasets.

Training Dataset	Algo	Train error(RMSE)	Test error(RMSE)
Small (2261 rows)	User-User Similarity	0.6885	1.0912
Small (2261 rows)	Item-Item Similarity	0.7199	1.1178
Small (2261 rows)	Matrix Factorization(SVD)	0.7615	1.0436
Small (2261 rows)	Matrix Factorization (Textual Reviews)	0.1	2.55
Large(16659 rows)	User-User Similarity	0.5532	1.3404
Large(16659 rows)	Item-Item Similarity	0.2636	1.0768
Large(16659 rows)	Matrix Factorization (SVD)	0.743	1.0121
Large(16659 rows)	Matrix Factorization (Textual Reviews)	0.2645	1.959

All algorithms for this dataset are suffering from over fitting. In small dataset user-user is performing slightly better than SVD in terms of total error but test error of user-user is very high as compared to SVD.

On large dataset item-item have very less train error which is quite expected as training data especially for hotels has increased and testing error has also decreased. For user-user train error has decreased but test error has increased surprisingly. One possible explanation can be that as large train data set has lot of users which are not present in test data set due to which it might be over fitting. SVD's performance has improved

There's a significant improvement for matrix factorization using textual reviews as text reviews have increased significantly in train data set. This shows that as textual reviews increase this algorithm performs better.

SVD and matrix factorization using text reviews both use stochastic gradient descent to optimize. But have different starting points. SVD is sort of like a black box. We don't know what those latent factors represent but with textual reviews we get an idea that what user has preference for and which factors affect rating of hotels. Overall for this hotel reviews data set we can say that SVD is the best option as in both extreme cases of data it is performing pretty well with a slightly better balance between test error and training error than all other algorithms.

Reference

- Mining of Massive Datasets by Anand Rajaraman and Jeffrey Ullman
- Mining of massive datasets lecture series:
https://www.youtube.com/watch?v=xoA5v9A07S0&list=PLLssT5z_DsK9JDLcT8T62VtzwyW9LNepV
- An Intelligent Data Analysis for Hotel Recommendation Systems using Machine Learning: Bushra Ramzan, Imran Sarwar Bajwa , Noreen Jamil, Farhaan Mirza
- Surprise python package-<https://surprise.readthedocs.io/en/stable/>
- <https://towardsdatascience.com/how-to-build-a-restaurant-recommendation-system-using-latent-factor-collaborative-filtering-ffe08dd57dca>
- Kaggle dataset <https://www.kaggle.com/datafiniti/hotel-reviews>

Appendix

```
df1=pd.read_csv('Datafiniti_Hotel_Reviews_Jun19.csv')
df2=pd.read_csv('Datafiniti_Hotel_Reviews.csv')
features=['reviews.username','categories','city','country','name','province',
'reviews.rating','reviews.text']
df1=df1[features]
df2=df2[features]
df_concat=pd.concat([df1,df2])
column_to_keep=['reviews.username','name','reviews.rating','reviews.text']
df_concat=df_concat[column_to_keep]
df_concat=df_concat.rename(columns={"reviews.username": "Users",
"name":
"Hotel_Name","reviews.rating":"Rating","reviews.text":"Reviews"})

#
a=df_concat.groupby('Users')['Hotel_Name'].count()
df_reduced=df_concat[df_concat['Users'].isin(a[a>=2].index)]
df_1review=df_concat[df_concat['Users'].isin(a[a==1].index)]
df_reduced.head()

df=df_reduced

from sklearn.model_selection import train_test_split
train0,test0=train_test_split(df,test_size=0.4,random_state=42)
test0=test0[test0['Users'].isin(train0['Users'])]
test0=test0[test0['Hotel_Name'].isin(train0['Hotel_Name'])]

from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import GridSearchCV
from surprise.model_selection import cross_validate
from surprise import BaselineOnly
from surprise import KNNBaseline
from surprise import KNNBasic
from surprise import SVD

#Data for CV and testing
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(train0[['Users', 'Hotel_Name', 'Rating']],
reader)
```

```

trainset_surprise=data.build_full_trainset()
trainset = data.build_full_trainset()
data2 = Dataset.load_from_df(test0[['Users', 'Hotel_Name', 'Rating']],
reader)
testset_surprise=data2.build_full_trainset()

#USER-USER STAARTS

#Parameters
param_grid = {'k': [2,3,5,7,10,12], 'sim_options':{'name':
['cosine'], 'user_based': [True]}}
grid_search
=GridSearchCV(KNNBaseline,param_grid,measures=['rmse'],cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_dataframe=pd.DataFrame(grid_search.cv_results)

import re
'split'
pattern_test_split = re.compile(r'^split..test_rmse$')
#pattern_train_split = re.compile(r'^split..test_rmse$')
matches = [x for x in param_dataframe.columns.tolist() if
pattern_test_split.match(x)]
matches.append('param_k')
a=param_dataframe[matches].set_index('param_k').stack().reset_index().
rename(columns={0:'CV error'})
a.boxplot(column='CV error',by='param_k')

b=param_dataframe[['mean_test_rmse','param_k']]
plt.plot('param_k','mean_test_rmse',data=b,marker="o")
plt.title("Mean CV error VS k")
plt.xlabel("k")
plt.ylabel("Mean CV error")
plt.show()

##User User Testing and traing error

#Traingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error user-user ', end=' ')
accuracy.rmse(training_predictions)

#Testing error

```



```

pred_user_user=algo.test(testset_surprise.build_testset())
print('Testing error user-user ', end=' ')
accuracy.rmse(pred_user_user)
print('Best Parameters for user-user ', end=' ')
print(grid_search.best_params['rmse'])

#USER-USER Ends

#ITEM ITEM Starts

#Parametrs
param_grid = {'k': [2,3,5,7,10,12], 'sim_options':{'name':
['cosine'], 'user_based': [False]}}
grid_search
=GridSearchCV(KNNBaseline,param_grid,measures=['rmse'],cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_dataframe=pd.DataFrame(grid_search.cv_results)

a=param_dataframe[matches].set_index('param_k').stack().reset_index().
rename(columns={0:'CV error'})
a.boxplot(column='CV error',by='param_k')

b=param_dataframe[['mean_test_rmse','param_k']]
plt.plot('param_k','mean_test_rmse',data=b,marker="o")
plt.title("Mean CV error VS k")
plt.xlabel("k")
plt.ylabel("Mean CV error")
plt.show()

# Item-Item training and testing

#Traingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error item-item ', end=' ')
accuracy.rmse(training_predictions)

#Testing error
pred_item_item=algo.test(testset_surprise.build_testset())
print('Testing error item-item ', end=' ')
accuracy.rmse(pred_item_item)
print('Best Parameters for item-item ', end=' ')
print(grid_search.best_params['rmse'])

```

```

#Item-Item Ends

###SVD STARTS

#Grid Search and finding best parameters for SVD
param_grid =
{'n_factors':[10,12,15],'n_epochs':[10,15],'reg_all':[0.02,0.05],'lr_all':[0.01,
0.02]}
#param_grid={}
grid_search =GridSearchCV(SVD,param_grid,measures=['rmse'],cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_df=pd.DataFrame.from_dict(grid_search.cv_results)
param_df.head()

#Traingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error SVD ', end=' ')
accuracy.rmse(training_predictions)

#Testing error
pred_svd=algo.test(testset_surprise.build_testset())
print('Testing error SVD ', end=' ')
accuracy.rmse(pred_svd)
print('Best Parameters for SVD ', end=' ')
print(grid_search.best_params['rmse'])

###SVD END

import pandas as pd
import numpy as np

#####Textual Factorization starts(small data)
import math
import swifter

```

```

from sklearn.feature_extraction.text import TfidfVectorizer
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from nltk.tokenize import WordPunctTokenizer
from sklearn.metrics import mean_squared_error as mse
import seaborn as sns
import matplotlib.pyplot as plt

test1=test0
train1=test0

## cleaning textual reviews(tokenization,lemmatizing,stopwords
removing)
stopwords=list(STOP_WORDS)
stopwords.remove('no')
nlp=spacy.load('en_core_web_sm')
def clean(text):
    doc=nlp(text)
    tokens=[token.lemma_ for token in doc]
    tokens=[lemma for lemma in tokens if lemma.isalpha() and lemma not
in stopwords]
    texts=(' '.join(tokens))
    return(texts)
train1['cleaned']=train1['reviews.text'].swifter.set_npartitions(20).apply(
lambda x : clean(x),axis=1)
print(train1.head())

userid_df = train1[['username','cleaned']]
business_df = train1[['name', 'cleaned']]
userid_review = userid_df.groupby('username').agg({'cleaned': ' '.join})
hotel_review = business_df.groupby('name').agg({'cleaned': ' '.join})
hotel_review.head()

##applying tfidf on user reviews
from nltk.tokenize import WordPunctTokenizer
user_vectorizer=TfidfVectorizer(tokenizer =
WordPunctTokenizer().tokenize,ngram_range=(1,3),max_df=.95,min_df=.
03,max_features=5000)
user_vectorized=user_vectorizer.fit_transform(userid_review['cleaned'])
user_vectorized_df=pd.DataFrame(user_vectorized.toarray(),index=useri
d_review.index,columns=user_vectorizer.get_feature_names())
##removing words which were not in businesss_vectorized after applying
tfidf

```

```

user_vectorized_df=user_vectorized_df.drop(labels=['sincerely', 'river',
'san diego', 'french quarter', 'french', 'best', 'lounge', 'resort', 'quarter',
'hyatt', 'marriott', 'delighted', 'apologize', 'recent stay', 'seattle', 'garden',
'good great', 'diego'],axis=1)
user_vectorized_df.head()

```

```

##applying tfidf on hotel_reviews
hotel_vectorizer=TfidfVectorizer(tokenizer =
WordPunctTokenizer().tokenize,ngram_range=(1,3),max_df=.95,min_df=.
03,max_features=5000)
hotel_vectorized=hotel_vectorizer.fit_transform(hotel_review['cleaned'])
hotel_vectorized_df=pd.DataFrame(hotel_vectorized.toarray(),index=hote
l_review.index,columns=hotel_vectorizer.get_feature_names())
##set columns same as that of user_vectorized
feat=user_vectorized_df.columns
hotel_vectorized_df=hotel_vectorized_df[feat]
hotel_vectorized_df.head()

```

```

##predicting rating of train set
predict_rate=user_vectorized_df.dot(hotel_vectorized_df.T)
predict_rate

```

```

##original rating matrix of train set
org_rate = pd.pivot_table(train1, values='reviews.rating',
index=['username'], columns=['name'])
org_rate

```

```

org_vec=org_rate.values.flatten()##original rating values in train set

```

```

##function of optimization algorithm on loss function
def matrix_factorization(R,P,Q,steps=20,gamma=0.001,lamda=0.02):
    for step in range(steps):
        print(step)
        m=0
        for i in R.index:
            m+=1
            if m%10==0:
                print('=',end='')
            for j in R.columns:
                if R.loc[i,j]>0:
                    eij=R.loc[i,j]-np.dot(P.loc[i],Q.loc[j])
                    P.loc[i]=P.loc[i]+gamma*(eij*Q.loc[j]-lamda*P.loc[i])
                    Q.loc[j]=Q.loc[j]+gamma*(eij*P.loc[i]-lamda*Q.loc[j])

```

```

return P,Q

%%time
P, Q = matrix_factorization(org_rate, user_vectorized_df,
hotel_vectorized_df,steps=1, gamma=0.02,lamda=0.02)

##calculating train error
org_rate = pd.pivot_table(train1, values='reviews.rating',
index=['username'], columns=['name'])
pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,column
ns=hotel_vectorized_df.index)
pred_vec=pred.values.flatten()
org_vec=org_rate.values.flatten()
train_error_df=pd.DataFrame({'org':org_vec,'pred':pred_vec})
train_error_df=train_error_df.dropna(axis=0)
from sklearn.metrics import mean_squared_error as mse
mse(train_error_df['org'],train_error_df['pred'])

## finding test set error
import seaborn as sns
pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,column
ns=hotel_vectorized_df.index)
org_test = pd.pivot_table(test1, values='reviews.rating',
index=['username'], columns=['name'])
org_test_vec=org_test.values.flatten()
pred_df=pred.loc[org_test.index,org_test.columns]
pred_df_vec=pred_df.values.flatten()
df_test=pd.DataFrame({'org':org_test_vec,'pred':pred_df_vec})
df_test=df_test.dropna(axis=0)
print((mse(df_test['org'],df_test['pred']))**(1/2))

## getting test data predicted rating predicted rating against original
rating
flat_org=pd.pivot_table(test1, values='reviews.rating',
index=['username'], columns=['name']).stack().reset_index()
flat_org.columns = ['username','name','org_ratings']
org_test = pd.pivot_table(test1, values='reviews.rating',
index=['username'], columns=['name'])
flat_pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,columns=hotel_vectorized_df.index).loc[org_test.index,org_test.columns].stack()

```

```

predicted=[]
for i in range(len(flat_org)):
    a=flat_org['username'][i]
    b=flat_org['name'][i]
    predicted.append(flat_pred.loc[a][b])
pred_org_df=flat_org
pred_org_df['predicted']=predicted
pred_org_df_small=pred_org_df

def recommend(username):
    predict_rate=P.dot(Q.T)
    return (predict_rate.loc[username].sort_values(ascending=False)[0:5])

recommend('susan')

#####Textual Factorization ends(small data)

#Now testing and traing with bigdata
trainsmall=train0
train0=pd.concat([train0,df_1review])

#USER-USER STAARTS

#Parametrs
param_grid = {'k': [2,3,5,7,10,12], 'sim_options':{'name':
['cosine'],'user_based': [True]}}
grid_search
=GridSearchCV(KNNBaseline,param_grid,measures=['rmse'],cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_dataframe=pd.DataFrame(grid_search.cv_results)

import re
'split'
pattern_test_split = re.compile(r'^split..test_rmse$')
#pattern_train_split = re.compile(r'^split..test_rmse$')
matches = [x for x in param_dataframe.columns.tolist() if
pattern_test_split.match(x)]
matches.append('param_k')
a=param_dataframe[matches].set_index('param_k').stack().reset_index().
rename(columns={0:'CV error'})

```

```

a.boxplot(column='CV error',by='param_k')

b=param_dataframe[['mean_test_rmse','param_k']]
plt.plot('param_k','mean_test_rmse',data=b,marker="o")
plt.title("Mean CV error VS k")
plt.xlabel("k")
plt.ylabel("Mean CV error")
plt.show()

##User User Testing and traing error

#Traingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error user-user ', end=' ')
accuracy.rmse(training_predictions)

#Testing error
pred_user_user=algo.test(testset_surprise.build_testset())
print('Testing error user-user ', end=' ')
accuracy.rmse(pred_user_user)
print('Best Parameters for user-user ', end=' ')
print(grid_search.best_params['rmse'])

#USER-USER Ends

#ITEM ITEM Starts

#Parametrs
param_grid = {'k': [2,3,5,7,10,12], 'sim_options':{'name':
['cosine'],'user_based': [False]}}
grid_search
=GridSearchCV(KNNBaseline,param_grid,measures=['rmse'],cv=5)
grid_search.fit(data)
algo = grid_search.best_estimator['rmse']
param_dataframe=pd.DataFrame(grid_search.cv_results)

a=param_dataframe[matches].set_index('param_k').stack().reset_index().
rename(columns={0:'CV error'})
a.boxplot(column='CV error',by='param_k')

b=param_dataframe[['mean_test_rmse','param_k']]
plt.plot('param_k','mean_test_rmse',data=b,marker="o")
plt.title("Mean CV error VS k")

```

```

plt.xlabel("k")
plt.ylabel("Mean CV error")
plt.show()

# Item-Item training and testing

#Traingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error item-item ', end=' ')
accuracy.rmse(training_predictions)

#Testing error
pred_item_item=algo.test(testset_surprise.build_testset())
print('Testing error item-item ', end=' ')
accuracy.rmse(pred_item_item)
print('Best Parameters for item-item ', end=' ')
print(grid_search.best_params['rmse'])

#Item-Item Ends

###SVD STARTS

#Grid Search and finding best parameters for SVD
param_grid =
{'n_factors':[10,12,15], 'n_epochs':[10,15], 'reg_all':[0.02,0.05], 'lr_all':[0.01,
0.02]}
#param_grid={}
grid_search =GridSearchCV(SVD,param_grid,measures=['rmse'],cv=5)
grid_search .fit(data)
algo = grid_search.best_estimator['rmse']
param_df=pd.DataFrame.from_dict(grid_search.cv_results)
param_df.head()

#Traingmodel
algo.fit(trainset)

#Accuracy training_predictions
training_predictions=algo.test(trainset_surprise.build_testset())
print('Training error SVD ', end=' ')
accuracy.rmse(training_predictions)

```



```

#Testing error
pred_svd=algo.test(testset_surprise.build_testset())
print('Testing error SVD ', end=' ')
accuracy.rmse(pred_svd)
print('Best Parameters for SVD ', end=' ')
print(grid_search.best_params['rmse'])

###SVD END

#####Textual Factorization starts(large data)

test1=test0
train1=test0

## cleaning textual reviews(tokenization,lemmatizing,stopwords
removing)
stopwords=list(STOP_WORDS)
stopwords.remove('no')
nlp=spacy.load('en_core_web_sm')
def clean(text):
    doc=nlp(text)
    tokens=[token.lemma_ for token in doc]
    tokens=[lemma for lemma in tokens if lemma.isalpha() and lemma not
in stopwords]
    texts=(' '.join(tokens))
    return(texts)
train1['cleaned']=train1['reviews.text'].swifter.set_npartitions(20).apply(
lambda x : clean(x),axis=1)
print(train1.head())

userid_df = train1[['username','cleaned']]
business_df = train1[['name', 'cleaned']]
userid_review = userid_df.groupby('username').agg({'cleaned': ' '.join})
hotel_review = business_df.groupby('name').agg({'cleaned': ' '.join})
hotel_review.head()

##applying tfidf on user reviews
from nltk.tokenize import WordPunctTokenizer
user_vectorizer=TfidfVectorizer(tokenizer =
WordPunctTokenizer().tokenize, ngram_range=(1,3), max_df=.95, min_df=.
03, max_features=5000)

```

```

user_vectorized=user_vectorizer.fit_transform(userid_review['cleaned'])
user_vectorized_df=pd.DataFrame(user_vectorized.toarray(),index=useri
d_review.index,columns=user_vectorizer.get_feature_names())
##removing words which were not in businesss_vectorized after applying
tfidf
user_vectorized_df=user_vectorized_df.drop(labels=['sincerely', 'river',
'san diego', 'french quarter', 'french', 'best', 'lounge', 'resort', 'quarter',
'hyatt', 'marriott', 'delighted', 'apologize', 'recent stay', 'seattle', 'garden',
'good great', 'diego'],axis=1)
user_vectorized_df.head()

```

```

##applying tfidf on hotel_reviews
hotel_vectorizer=TfidfVectorizer(tokenizer =
WordPunctTokenizer().tokenize,ngram_range=(1,3),max_df=.95,min_df=.
03,max_features=5000)
hotel_vectorized=hotel_vectorizer.fit_transform(hotel_review['cleaned'])
hotel_vectorized_df=pd.DataFrame(hotel_vectorized.toarray(),index=hote
l_review.index,columns=hotel_vectorizer.get_feature_names())
##set columns same as that of user_vectorized
feat=user_vectorized_df.columns
hotel_vectorized_df=hotel_vectorized_df[feat]
hotel_vectorized_df.head()

```

```

##predicting rating of train set
predict_rate=user_vectorized_df.dot(hotel_vectorized_df.T)
predict_rate

```

```

##original rating matrix of train set
org_rate = pd.pivot_table(train1, values='reviews.rating',
index=['username'], columns=['name'])
org_rate

```

```

org_vec=org_rate.values.flatten()##original rating values in train set

```

```

##function of optimization algorithm on loss function
def matrix_factorization(R,P,Q,steps=20,gamma=0.001,lamda=0.02):
    for step in range(steps):
        print(step)
        m=0
        for i in R.index:
            m+=1
            if m%10==0:
                print('=',end='')

```

```

        for j in R.columns:
            if R.loc[i,j]>0:
                eij=R.loc[i,j]-np.dot(P.loc[i],Q.loc[j])
                P.loc[i]=P.loc[i]+gamma*(eij*Q.loc[j]-lamda*P.loc[i])
                Q.loc[j]=Q.loc[j]+gamma*(eij*P.loc[i]-lamda*Q.loc[j])

    return P,Q

%%time
P, Q = matrix_factorization(org_rate, user_vectorized_df,
hotel_vectorized_df,steps=1, gamma=0.02,lamda=0.02)

##calculating train error
org_rate = pd.pivot_table(train1, values='reviews.rating',
index=['username'], columns=['name'])
pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,column
ns=hotel_vectorized_df.index)
pred_vec=pred.values.flatten()
org_vec=org_rate.values.flatten()
train_error_df=pd.DataFrame({'org':org_vec,'pred':pred_vec})
train_error_df=train_error_df.dropna(axis=0)
from sklearn.metrics import mean_squared_error as mse
mse(train_error_df['org'],train_error_df['pred'])

## finding test set error
import seaborn as sns
pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,column
ns=hotel_vectorized_df.index)
org_test = pd.pivot_table(test1, values='reviews.rating',
index=['username'], columns=['name'])
org_test_vec=org_test.values.flatten()
pred_df=pred.loc[org_test.index,org_test.columns]
pred_df_vec=pred_df.values.flatten()
df_test=pd.DataFrame({'org':org_test_vec,'pred':pred_df_vec})
df_test=df_test.dropna(axis=0)
print((mse(df_test['org'],df_test['pred']))**(1/2))

## getting test data predicted rating predicted rating against original
rating
flat_org=pd.pivot_table(test1, values='reviews.rating',
index=['username'], columns=['name']).stack().reset_index()
flat_org.columns = ['username','name','org_ratings']

```

```

org_test = pd.pivot_table(test1, values='reviews.rating',
index=['username'], columns=['name'])
flat_pred=pd.DataFrame(np.dot(P,Q.T),index=user_vectorized_df.index,columns=hotel_vectorized_df.index).loc[org_test.index,org_test.columns].stack()
predicted=[]
for i in range(len(flat_org)):
    a=flat_org['username'][i]
    b=flat_org['name'][i]
    predicted.append(flat_pred.loc[a][b])
pred_org_df=flat_org
pred_org_df['predicted']=predicted
pred_org_df

```

```

def recommend(username):
    predict_rate=P.dot(Q.T)
    return (predict_rate.loc[username].sort_values(ascending=False)[0:5])

```

```

recommend('susan')

```

```

#####Textual Factorization ends(large data)

```

```

Analysistable=pd.merge(pd.DataFrame(pred_svd)[['uid','iid','r_ui','est']],Users_in_big_Data,left_on=['uid'], right_on = ['Users'],
how='left').drop('Users',axis=1)
Analysistable=pd.merge(Analysistable,Hotels_in_big_Data,left_on=['iid'],
right_on = ['Hotel_Name'],
how='left').drop('Hotel_Name',axis=1).rename(columns={'uid':'Users','iid':'Hotel_Name','r_ui':'Actual Rating','est':'SVD_est'})
Analysistable=pd.merge(Analysistable,Hotels_in_small_Data,left_on=['Hotel_Name'], right_on = ['Hotel_Name'], how='left')
Analysistable=pd.merge(Analysistable,Users_in_small_Data,left_on=['Users'], right_on = ['Users'], how='left')
Analysistable=pd.merge(Analysistable,pred_org_df_small[['username','name','predicted using small data']]).rename(columns={'predicted using small data':'Reviews_smalldata_est'}),left_on=['Users','Hotel_Name'],
right_on = ['username','name'],
how='left').drop(['username','name'],axis=1)
Analysistable=pd.merge(Analysistable,pred_org_df[['username','name','predicted using big data']]).rename(columns={'predicted using big data':'Reviews_bigdata_est'}),left_on=['Users','Hotel_Name'], right_on = ['username','name'], how='left').drop(['username','name'],axis=1)

```

```

Analysistable['pct chage in
rating']=((Analysistable['Reviews_bigdata_est']-
Analysistable['Reviews_smallldata_est'])/Analysistable['Reviews_smalldat
a_est'])*100
Analysistable['Pos/Neg']=np.sign((Analysistable['Actual Rating']-
Analysistable['Reviews_smallldata_est'])-(Analysistable['Actual Rating']-
Analysistable['Reviews_bigdata_est'])) #-
Analysistable['Reviews_bigdata_est'])
Analysistable['Increase in hotels']=Analysistable['No of Hotels in big Data
train']-Analysistable['No of Hotels in small Data train']
Analysistable['Increase in Users']=Analysistable['No of Users in big Data
train']-Analysistable['No of Users in small Data train']
Analysistable.head()

forscatter=Analysistable[['Users','Hotel_Name','pct chage in
rating','Pos/Neg','Increase in hotels','Increase in Users']]
forscatter.head()

#Scatter plot Increase in hotels vs pct chage in rating
plt.scatter(forscatter['Increase in hotels'],forscatter['pct chage in
rating'],c=forscatter['Pos/Neg'])
plt.xlabel="Increase in hotels"
plt.show()

#Number of Predictions improved and got
forscatter[['Pos/Neg']].groupby('Pos/Neg')['Pos/Neg'].count()

```