Machine Learning Engineer Nanodegree

Capstone Project

Kanja Saha
January 15, 2018

I. Definition

Project Overview

Build a Product Recommendation Engine with item to item Collaborative Filtering technique using Matrix Factorization

Some say customer is God and others say customer is King. Some say, "Listen to your customer!" while others say "Know thy customer!". The latter is my personal marketing philosophy!

A company exists because of its customers; to be precise, its loyal customers. A loyal customer is a satisfied customer whose trust you have earned. And the word "trust" carries a lot of weight and is only earned through consistent positive value of product, service and experience and show that you "know your customer" . To reflect that you indeed know your customer, it is imperative that you recommend only the products that you believe will benefit the customer.

Hence a product recommendation engine is crucial for every company's success. In fact it is necessary for success of every department of the company. A recommender system is already in place for large market place such as Amazon, Google and other organizations, small or large, are inspired to build a near accurate recommender system. As I got introduced to Nearest Neighor Algorithm in Unsupervised Learning in Udacity's Machine Learning Nanodegree Program, I realized that clustering is at the base of Recommender System. I then researched further to find Collaborative Filtering technique implementing Matrix Factorization to be one of the popular methods to build an efficient recommender engine.

In the more general sense, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. In terms of recommendation system, it is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating) and finding similarities between items based on user feedback. In the mathematical discipline of linear algebra, a matrix decomposition or matrix factorization is a dimensionality reduction technique that factorizes a matrix into a product of matrices, usually two.

In building Recconmendation System using item to item Collaborative Filtering technique, we generate two matrices through convergence: an user matrix with latent features of the users and an item matrix with a few latent features of the Items. We then take the item similarity matrix and assign rating to users based on previous ratings of the user on similar items.

I will use Amazon Product Ratings Only Dataset for this project. These datasets include no metadata or reviews, but only (user,item,rating,timestamp) tuples. I will use reviewerID(user), asin(item) and overall(rating) columns for my project.

Following are the details on the dataset.

- reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin - ID of the product, e.g. 0000013714
- overall: rating of the product,
- reviewTime - time of the review (raw)

Sample Ratings Only Data: { "reviewerID": "A2SUAM1J3GNN3B", "asin": "0000013714", "overall": 5.0, "reviewTime": "09 13, 2009" }

Source for the data: http://jmcauley.ucsd.edu/data/amazon/links.html (http://jmcauley.ucsd.edu/data/amazon/links.html)

Reference:

https://www.quora.com/What-is-a-laymans-explanation-of-matrix-factorization-in-collaborative-filtering (https://www.quora.com/What-is-a-laymans-explanation-of-matrix-factorization-in-collaborative-filtering)
http://vcp.med.harvard.edu/papers/matrices-3.pdf (http://vcp.med.harvard.edu/papers/matrices-3.pdf)
http://mahout.apache.org/users/recommender/matrix-factorization.html (http://mahout.apache.org/users/recommender/matrix-factorization.html)

Problem Statement

With the advent of internet commerce, it has become convenient for the users to find the items of their interest without stepping out of their house. However with the consistent increase of number of online retailers it is also very difficult for the user to find the items that really meets their need the best. Users feel lost in the sea of products available to them and often fear of making a wrong purchasing decision. And once they make a purchase that is not ideal, customers shy away from making further online purchase investment or recommending others. This is a no win situation for the user as well as the company selling the product

Recommendation system is considered a semi supervised learning or a combination of supervised(ranking a recommended item) and unsupervised learning(forming clusters of similar groups of customers/items). But overall it is an information retrieval system, which is another large area of machine learning.

To make a buyer feel confident about making frequent purchase decision, there is a need for a system which learns the user preferences, spending pattern and generate recommendations based on his interest and past buying habit, The Recommender System. I believe, one of the best ways to build the recommendation engine is by using Collaborative Filtering technique. Collaborative filtering is the technique of recommending items to users based on past interactions between users and items.

So, my goal is to build a recommendation system with item to item Collaborative Filtering Technique. I will implement the Matrix Factorization algorithms in sklearn Library: Non-Negative Matrix Factorization.

Metrics

I will use Root Mean Square Error (RMSE) as evaluation metrics.

Root mean squared error (RMSE): RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It’s the square root of the average of squared differences between prediction and actual observation.

root-mean-square-error.png

where n is the total number of observations, and Z(fi) and Z(oi) are forcasted and observed values of an observation i.

Reference:

http://www.statisticshowto.com/rmse/ (http://www.statisticshowto.com/rmse/) https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d (https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d)

II. Analysis

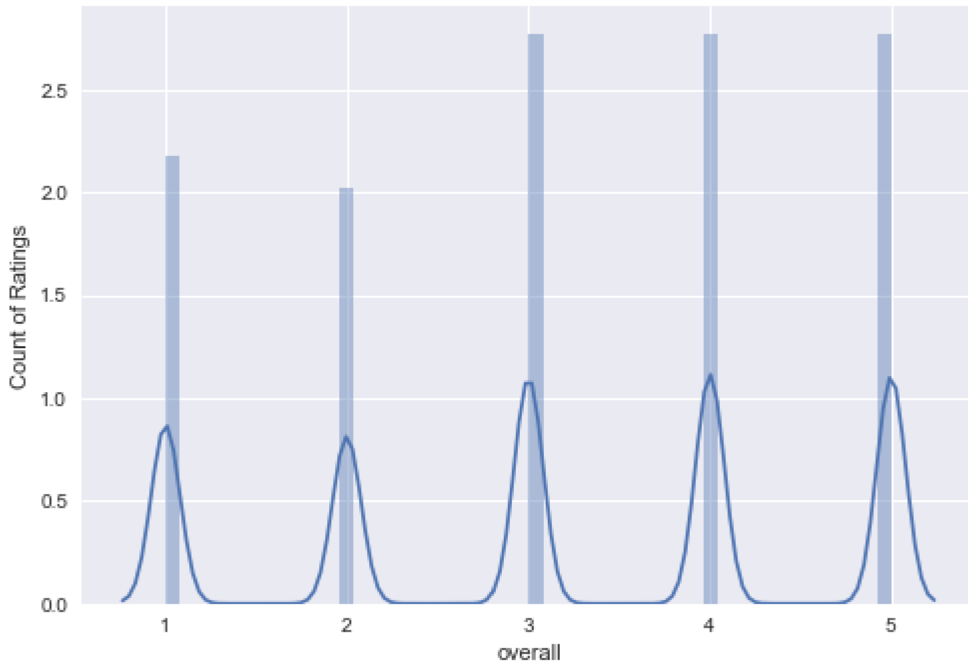
Data Exploration & Visualization

Amazon ratings source dataset has around 82.5 million rows and 4 columns. The first three columns, reviewerID, asin and overall are required for this project. The dataset has 21,176,522 unique reviewers and 9,874,211 items.

Due to limitation of my personal computer performance as well as more accuracy. The final dataset of 10 million rows and 3 columns consist of a set of 576,489 reviewers who have rated atleast 20 items as well 112,980 items that are rated by atleast 100 reviewers.
This filtered dataset does not have any null values.

In [2]:
```
### Load Libraries and Dataset
import pandas as pd
import numpy as np
import seaborn as sns;
import matplotlib.pyplot as plt;
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import NMF
from IPython.display import display
import itertools


#raw_data= pd.read_csv("item_dedup.csv", skiprows=2000000,nrows=1000000, heade
r=None)
raw_data_step1= pd.read_csv("item_dedup.csv", nrows=7000000, header=None)
raw_data_step1.columns = ['reviewerID', 'asin','overall','reviewTime']
raw_data_step1.drop(['reviewTime'], axis = 1, inplace = True)

raw_data=raw_data_step1.assign(rnk=raw_data_step1.groupby(['overall'])['asin']
                    .rank(method='min', ascending=False)) .query('rnk < 50000
0')
raw_data.drop(['rnk'], axis = 1, inplace = True)

display(raw_data.shape)
#raw_data.isnull().any()
raw_data.head()

sns.set(color_codes=True)
plt.figure('overall')
plt.ylabel('Count of Ratings')
plt.xlabel('Ratings 1 to 5')
sns.distplot(raw_data['overall'])
plt.show()
```

```
(2258334, 3)
```



In [3]:
```
# Generate Subset of the data

item_rating_count=raw_data.groupby('asin').aggregate({'reviewerID': np.count_n
onzero})
item_rating_count_gte_200 = item_rating_count[item_rating_count.reviewerID > 2
00]

asin_rating_count_gte_200=item_rating_count_gte_200.index

shortlisted_data = raw_data[raw_data.asin.isin(asin_rating_count_gte_200)]

final_data=shortlisted_data.assign(rnk=shortlisted_data.groupby(['asin'])['rev
iewerID']
                    .rank(method='min', ascending=False)) .query('rnk < 25')
final_data.drop(['rnk'], axis = 1, inplace = True)

display(final_data.isnull().any())
display(final_data.shape)
display(final_data.nunique())
final_data.head()
```

```
reviewerID    False
asin          False
overall       False
dtype: bool
```

```
(25368, 3)
```

```
reviewerID    21093
asin           1057
overall           5
dtype: int64
```

Out[3]:

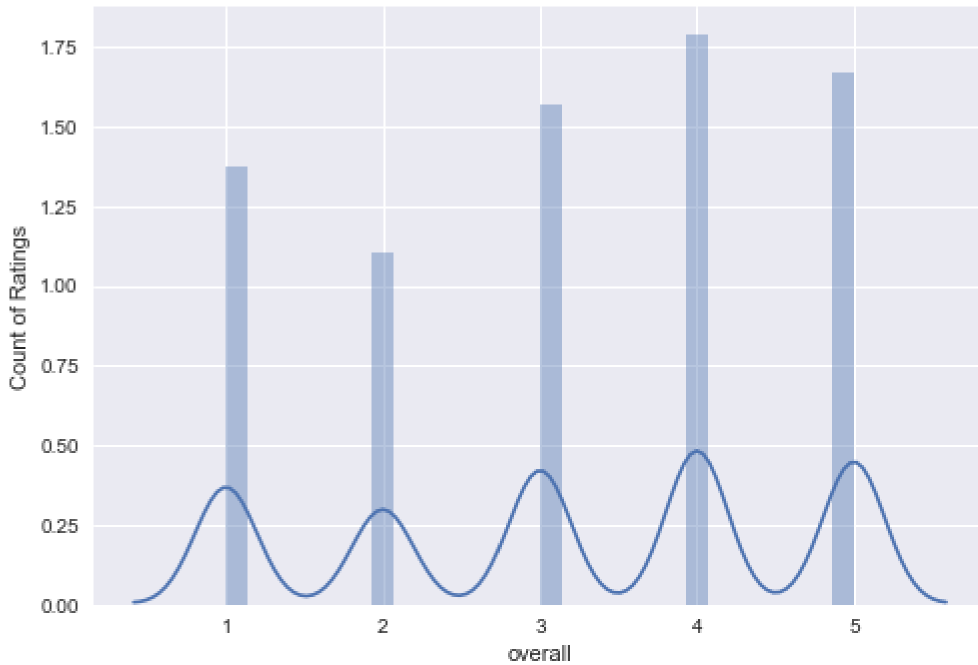
	reviewerID	asin	overall
1810	AUX3BTT3AUN4N	0002007770	1.0
2444	AWC67YBKLTIQ4	0002007770	1.0
2465	ATOEBBOWB6ZQJ	0002007770	2.0
2888	AU0PC4RM01IOD	0002007770	2.0
3167	AUPS99MM70UHD	0002007770	2.0

Exploratory Visualization

The count of items for each ratings in the dataset is obtained and the graph is plotted as shown below in Figure 1. From the graph it is obvious that users have mostly rated items as 4 or 5, which is a good news because this implies there is high potential for product recommendation of similar items. About 50% of all ratings are 5, 30% of all ratings are 4, 5% are 3 and the rest are 1 and 2.

In [4]:

```
# Rating and its count
sns.set(color_codes=True)
plt.figure('overall')
plt.ylabel('Count of Ratings')
plt.xlabel('Ratings 1 to 5')
sns.distplot(final_data['overall'])
plt.show()

display(final_data['overall'].describe([.1,.15,.2,.3,.4,.5,.6,.7,.8,.9]).apply
(lambda x: '%.0f' % x))
```




```
count    25368
mean         3
std          1
min          1
10%          1
15%          1
20%          2
30%          2
40%          3
50%          3
60%          4
70%          4
80%          5
90%          5
max          5
Name: overall, dtype: object
```

Algorithms and Techniques

I will use Non-Negative Matrix Factorization (NMF) model which is implemented in sklearn library in Python.

Non-Negative Matrix Factorization (NMF) is a recent technique for linear dimensionality reduction and data analysis that yields a parts based, sparse non-negative representation for non-negative input data. Essentialy, NMF is an unsupervised learning algorithm coming from linear algebra that not only reduces data dimensionality, but also performs clustering simultaneously.

70% of the items in this dataset has receieved rating from .05% of total reviewers, this can be considered a sparse matrix. NMF tend to perform better than SVD and other matrix factorization alogorithms for sparse matrix.

NMF finds two non-negative matrices (W, H) whose product approximates the original non- negative matrix X.

NMF has quite a few parameters but the most deciding factor is n_components, an integer or none. n_components is the total number of latest features of the factored Matrices. The optimum value for n_components can be generated by implementing NMF model in iterative fashion with different n_components and generate the RMSE value for each value of n_components in NMF. The n_components(latent features) with least RMSE score will be used to find the RMSE score in test dataset.

The parameters and attribute of NMF in sklearn are noted as below: class sklearn.decomposition.NMF(n_components=None, init=None, solver='cd', beta_loss='frobenius', tol=0.0001, max_iter=200, random_state=None, alpha=0.0, l1_ratio=0.0, verbose=0, shuffle=False)

The details of this class and its default parameters can be found in sklearn documentation. [http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html)](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html)

In [5]:
```
display(item_rating_count_gte_200['reviewerID'].describe([.1,.15,.2,.3,.4,.5,.6,.7,.8,.9]).apply(lambda x: '%.0f' % x))
```

```
count    1057
mean      485
std       667
min       201
10%       216
15%       224
20%       233
30%       251
40%       273
50%       309
60%       353
70%       421
80%       536
90%       818
max      8516
Name: reviewerID, dtype: object
```

Benchmark

I will use nearest neighors algorithm in sklearn library to generate the benchmark numbers.

The benchmark score for RMSE is 4 with k neighors and are generated by implementing nearest neighors algorithm in the Methodology/Implementation Section.

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

The final dataset is pivoted to create a sparse dataframe with "reviewerID" as Index, "asin" as Column and "overall" as value.
This final dataframe has n rows(reviewers) and n columns(items).

We then split this dataframe into train and test dataset(20% for testing). 10 fold validation will be performed on training dataset to find the optimum k value for k neighors algorithm and optimum p latent features for Non Negative Matrix Factorization based on their RMSE and MAE scores.

These k and p values will then be used on test dataset for kneighors and NMF algorithm. The RMSE and MAE score from kneighors is will be considered as baseline and will be used as benchmark to compare the performance of NMF algorithm based on the metrics.

In [6]:
```
# Preprocessing Data
#final_data=raw_data
pivot_data=pd.pivot_table(final_data, values = 'overall', index='reviewerID',
columns ='asin')
pivot_data.fillna(0, inplace=True)
display(pivot_data.shape)

train_data, test_data = train_test_split(pivot_data, test_size=0.2)

display(train_data.head())
test_data.head()
```

```
(21093, 1057)
```

asin	0002007770	0002247399	0007124015	0007230206	0007267622	0007
reviewerID						
AN3GHUSTAQ9TY	0.0	0.0	0.0	0.0	0.0	0.0
AU19KV4I00FHX	0.0	0.0	0.0	0.0	0.0	0.0
AXK9WUGRUL86Z	0.0	0.0	0.0	0.0	0.0	0.0
AXPA6Y0KHGF4U	0.0	0.0	0.0	0.0	0.0	0.0
ATJM3SAR187CD	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1057 columns

Out[6]:

asin	0002007770	0002247399	0007124015	0007230206	0007267622	0007
reviewerID						
AWJJV7SHBQSXO	0.0	0.0	0.0	0.0	0.0	0.0
AY2RRT8235Y19	0.0	0.0	0.0	0.0	0.0	0.0
AX6JZKKXU2GAS	0.0	0.0	0.0	0.0	0.0	0.0
AYS30U8QJ3CP1	0.0	0.0	0.0	0.0	0.0	0.0
AZII059R8KQ9S	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1057 columns

In [141]:

```
cols = train_data.columns
bt = train_data.apply(lambda x: x > 1)
bt.apply(lambda x: list(cols[x.values]), axis=1)
```

```
Out[141]: reviewerID
          AL6CEWLV2JB90                                                 [0062311077]
          A1IZV9DB89P8JX                                    [0007447868, 009928264X]
          A1VUP643ST8MLL                                    [0140049975, 0143120530]
          A3GKFFNZKEGX72                        [0007167040, 0060530677, 0140306013]
          AQ0Q4CES379O4     [0061458031, 0061950726, 0062107909, 006225783...
          A32JGNK1JNUSST    [0002007770, 0007444117, 0061992704, 009928264...
          A1S8FJT0AQSF60                                    [0061340634, 0062120395]
          A236N1TURISZAF    [0006514006, 0007147295, 000721278X, 006080924...
          A20CCIZOG87D4V                                    [0060838728, 0140298479]
          A18VVDLRLG9SVO    [0060931418, 0060959037, 0060987103, 007553666...
          A1LQ7HPY4UZ3LR     [0143115626, 0143119494, 0143170090, 0143170104]
          A2GXX77LTFZV7G                        [000721278X, 0099464462, 0099740915]
          AS8ZHVA1D8SG9      [0061231401, 0061558230, 006199104X, 0142422037]
          AAZI6GOIXQCCV     [006056251X, 0061031127, 0099710013, 009991170...
          AX603WWABCK40     [0141014083, 0141039280, 0141348550, 014143976...
          A37WSVY0T27N5H                                                [0152023984]
          A1UPQKBGVN4KYB                                                [0195038630]
          A37ATPLT2QQAPW    [0006476155, 0020519109, 0060098902, 006029323...
          A27XRSRKRMY9KX    [0020519109, 0060652926, 0060794410, 006123400...
          A3839T2HJCJ5C0    [0007444117, 0061974552, 0061974617, 006207203...
          AH62BQTCMR3BR                                     [0060194995, 014028334X]
          A204271Z2ZM93S                        [0060081600, 0142403873, 0142501123]
          A1U78IWKBER52V                                                [0062304828]
          A2CUWCJHUPGL9W    [0007386648, 0020519001, 0061031127, 006112429...
          A1H1L4C71L8NIC    [0060558121, 0061122416, 006207993X, 014303666...
          A78RMO6NRXFZ6      [0007230206, 0062060554, 0140390464, 0141345713]
          AZGXZ2UUK6X       [0060175400, 0060245867, 0061146307, 009946126...
          A2M66S05KKNSWC    [0060293233, 0060530677, 0060930187, 009948168...
          AJBFZ07U43KUQ                         [0007447868, 0140058893, 0151015392]
          A3JIH8AAZLQQ4O    [0007444117, 0062059939, 0062068520, 006208561...
                                                          ...
          A1W9MHW1Z1GVNB                                                [0143113879]
          A2KUBAR7E86MVI                                                [0141345713]
          A3QNLYESISQMJH                                                [0141301066]
          ALNFHVS3SC4FV                                     [0061256684, 0061583251]
          A3N10W4T5GBPR2    [0060554738, 006073132X, 006124189X, 006170780...
          A1FAWFMP6CDKG                                     [0060838582, 0140434941]
          A27KTPDJQ0U7I6    [0020519109, 0060753943, 0061950718, 006196755...
          A1DYMH30TSRONY    [000721278X, 0007230206, 003061368X, 006054562...
          A1C88IUSMT5XMG    [0007410956, 0061009059, 006114097X, 006207104...
          A19V8EJVBEPF7X    [0061767891, 0061809462, 0062105655, 006212426...
          A2988B52PKKNHL                        [0061965499, 0071383832, 0141034599]
          A2G68FQPT40UU9                                    [0141039280, 0143121006]
          AA32DW440RBE6                                     [0061474096, 0061726826]
          AHGY6OGGP3Y2O     [0061969559, 0062024027, 0141326085, 014240663...
          AVDRCJR88Q8YP                                                 [0140390030]
          A2CK1SYLSD82B3                        [0007230206, 0061928127, 0062257838]
          A246GZE5MZKHGL    [0061138061, 0061726826, 0061957917, 006197806...
          A300Y3XGBASEN6                                                [0131983334]
          A2LWFNEYBPL5UJ                                    [0142429112, 0143123629]
          A2DUKTVSRMQCIT                        [0061992704, 0140177191, 0151008116]
          A3EUKRR3N1W7TR                                                [0091779251]
          A2IIB10X0CECYV    [0061732370, 0061995541, 0062026488, 006205993...
          A3B9YVPW42GS5Q     [0061348155, 0061732370, 0062316869, 0141326085]
          A1AIOAUEPSCA35                                                [006218850X]
          AT29RT9XL3RN1                                                 [0060892994]
```

```
A2OO9RZKYX81A0                [0060755334, 0141040343, 0141040378]
A30WXSK6N4J0CZ                            [0061474096, 0141188936]
A3JJ21YCMGSKGH                            [0061373311, 0062213652]
A22AQTN1FHX2TM                            [0061340634, 0099443635]
A12HCU3QCTQ2XT    [0060764864, 0060850523, 0060890096, 007553666...
Length: 11660, dtype: object
```

Implementation

- Programming Language: Python 3.6
- Libraries : Pandas,Numpy, Scikit-learn
- Goal: Implement NMF in sklearn to build a recommender system using item to item collaborative filtering technique
- Workflow :
  - Establish the baselines with K-nearest neighbors for comparison.

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:

- *Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?*
- *Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?*
- *Was there any part of the coding process (e.g., writing complicated functions) that should be documented?*

In [7]:

```
#Functions

def join_groundtruth_predicted_value(v_item_similarity,v_raw_data,v_data):

    item_prediction = v_data.dot(v_item_similarity) /((v_item_similarity).sum(
axis=1))

    item_prediction['reviewerID']=item_prediction.index
    prediction_melt=pd.melt(item_prediction, id_vars=['reviewerID'], var_name=
'asin', value_name='overall_pred')

    final=pd.merge(v_raw_data, prediction_melt, how='inner', on=['reviewerID',
'asin'])
    return final



def get_item_similarity_knn(v_data,v_i):
    v_data_t=v_data.transpose()
    model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute',n_neig
hbors=v_i)
    model_knn.fit(v_data_t)
    itemtoitem_1=model_knn.kneighbors_graph(v_data_t)
    itemtoitem=itemtoitem_1.toarray()
    item_similarity=pd.DataFrame(itemtoitem, columns=v_data.columns,index=v_da
ta.columns)

    return item_similarity
```

In [8]:
```
n_neighbors_approx=np.floor(sqrt(train_data.shape[1])).astype(np.int64)
neighors=range(1,n_neighbors_approx+8)
RMSE_score = []

for i in neighors:

        item_similarity_knn=get_item_similarity_knn(train_data,i)
        final=join_groundtruth_predicted_value(item_similarity_knn,raw_data,tr
ain_data)
        rmse=sqrt(mean_squared_error(final.overall_pred, final.overall))

        RMSE_score.append(rmse)
        print ('Item-based CF RMSE for Training Set: ' + str(rmse) + ' for ' +
 str(i) + ' neighors')

neighors_list=(list(neighors))
plt.plot(neighors_list, RMSE_score)
plt.xlabel('Number of Neighbors K')
plt.ylabel('RMSE')
plt.show()
```

```
Item-based CF RMSE for Training Set: 1.4009469018219187 for 1 neighors
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-8-f305901a8247> in <module>()
      6 
      7         item_similarity_knn=get_item_similarity_knn(train_data,i)
----> 8         final=join_groundtruth_predicted_value(item_similarity_knn,ra
w_data,train_data)
      9         rmse=sqrt(mean_squared_error(final.overall_pred, final.overal
l))
     10 

<ipython-input-7-f089fe0a8007> in join_groundtruth_predicted_value(v_item_sim
ilarity, v_raw_data, v_data)
      8     prediction_melt=pd.melt(item_prediction, id_vars=['reviewerID'],
 var_name='asin', value_name='overall_pred')
      9 
---> 10     final=pd.merge(v_raw_data, prediction_melt, how='inner', on=['rev
iewerID','asin'])
     11     return final
     12 

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\pandas\cor
e\reshape\merge.py in merge(left, right, how, on, left_on, right_on, left_ind
ex, right_index, sort, suffixes, copy, indicator)
     52                          right_index=right_index, sort=sort, suffixes
=suffixes,
     53                          copy=copy, indicator=indicator)
---> 54     return op.get_result()
     55 
     56 

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\pandas\cor
e\reshape\merge.py in get_result(self)
    567                 self.left, self.right)
    568 
--> 569         join_index, left_indexer, right_indexer = self._get_join_info
()
    570 
    571         ldata, rdata = self.left._data, self.right._data

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\pandas\cor
e\reshape\merge.py in _get_join_info(self)
    732         else:
    733             (left_indexer,
--> 734              right_indexer) = self._get_join_indexers()
    735 
    736             if self.right_index:

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\pandas\cor
e\reshape\merge.py in _get_join_indexers(self)
    711                                   self.right_join_keys,
    712                                   sort=self.sort,
--> 713                                   how=self.how)
    714 
    715     def _get_join_info(self):

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\pandas\cor
```

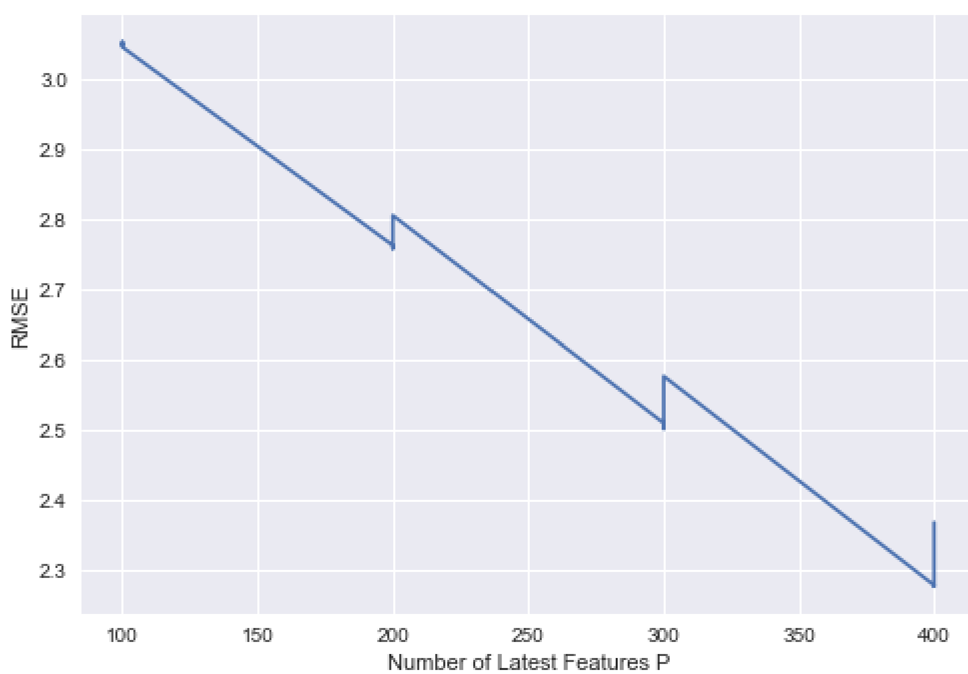
```
e\reshape\merge.py in _get_join_indexers(left_keys, right_keys, sort, how, **
kwargs)
    988     # `count` is the num. of unique keys
    989     # set(lkey) | set(rkey) == range(count)
--> 990     lkey, rkey, count = fkeys(lkey, rkey)
    991
    992     # preserve left frame order if how == 'left' and sort == False

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\pandas\cor
e\reshape\merge.py in _factorize_keys(lk, rk, sort)
   1403
   1404     llab = rizer.factorize(lk)
-> 1405     rlab = rizer.factorize(rk)
   1406
   1407     count = rizer.get_count()

pandas\_libs\hashtable.pyx in pandas._libs.hashtable.Int64Factorizer.factoriz
e (pandas\_libs\hashtable.c:34578)()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTa
ble.get_labels (pandas\_libs\hashtable.c:15606)()

C:\Users\ksaha\AppData\Local\Continuum\Anaconda3\lib\site-packages\numpy\core
\numeric.py in asarray(a, dtype, order)
    461
    462
--> 463 def asarray(a, dtype=None, order=None):
    464     """Convert the input to an array.
    465

KeyboardInterrupt:
```

In [9]:

```
#NMF

def join_groundtruth_predicted_value_NF(nmf,v_raw_data,v_data,v_i):

    W = nmf.transform(v_data);
    H = nmf.components_
    item_prediction_NF = np.dot(W,H)

    item_prediction_NF=pd.DataFrame(item_prediction_NF, columns=v_data.columns
,index=v_data.index)

    item_prediction_NF['reviewerID']=item_prediction_NF.index
    prediction_melt=pd.melt(item_prediction_NF, id_vars=['reviewerID'], var_na
me='asin', value_name='overall_pred')

    final=pd.merge(v_raw_data, prediction_melt, how='inner', on=['reviewerID',
'asin'])
    return final
```

In [10]:
```
latent_features=range(100,500,100)
alpha_values=(x * 0.001 for x in range(0, 10,5))
l1_ratio_values=(x * 0.01 for x in range(0, 10,5))
RMSE_nf_score = []
combo=[]
import itertools
for combination in itertools.product(latent_features, alpha_values, l1_ratio_v
alues):
    i=combination[0]
    a=combination[1]
    l1=combination[2]

    nmf = NMF(n_components =i,alpha=a,l1_ratio=l1)
    W = nmf.fit(train_data);
    final=join_groundtruth_predicted_value_NF(nmf,raw_data,train_data,i)
    rmse_nf=sqrt(mean_squared_error(final.overall_pred, final.overall))
    combo.append(i)
    RMSE_nf_score.append(rmse_nf)
    print ('Item-based CF RMSE: ' + str(rmse_nf) + ' for ' + str(i) + ' latent
 features, '+ str(a) + ' alpha, '+ str(l1) + ' l1_ratio')


latent_features_list=(list(combo))
plt.plot(latent_features_list, RMSE_nf_score)
plt.xlabel('Number of Latest Features P')
plt.ylabel('RMSE')
plt.show()
```

```
Item-based CF RMSE: 3.04953645107407 for 100 latent features, 0.0 alpha, 0.0
l1_ratio
Item-based CF RMSE: 3.0479485419997854 for 100 latent features, 0.0 alpha, 0.
05 l1_ratio
Item-based CF RMSE: 3.0527688437188716 for 100 latent features, 0.005 alpha,
0.0 l1_ratio
Item-based CF RMSE: 3.0452927657734743 for 100 latent features, 0.005 alpha,
0.05 l1_ratio
Item-based CF RMSE: 2.762020298934044 for 200 latent features, 0.0 alpha, 0.0
l1_ratio
Item-based CF RMSE: 2.7615568095427783 for 200 latent features, 0.0 alpha, 0.
05 l1_ratio
Item-based CF RMSE: 2.7575609251071014 for 200 latent features, 0.005 alpha,
0.0 l1_ratio
Item-based CF RMSE: 2.8048302055638628 for 200 latent features, 0.005 alpha,
0.05 l1_ratio
Item-based CF RMSE: 2.5087847816700086 for 300 latent features, 0.0 alpha, 0.
0 l1_ratio
Item-based CF RMSE: 2.5084161162870067 for 300 latent features, 0.0 alpha, 0.
05 l1_ratio
Item-based CF RMSE: 2.5008726117992417 for 300 latent features, 0.005 alpha,
0.0 l1_ratio
Item-based CF RMSE: 2.575814372007266 for 300 latent features, 0.005 alpha,
0.05 l1_ratio
Item-based CF RMSE: 2.2772217387502893 for 400 latent features, 0.0 alpha, 0.
0 l1_ratio
Item-based CF RMSE: 2.275612697464898 for 400 latent features, 0.0 alpha, 0.0
5 l1_ratio
Item-based CF RMSE: 2.2813731124131316 for 400 latent features, 0.005 alpha,
0.0 l1_ratio
Item-based CF RMSE: 2.366976207758109 for 400 latent features, 0.005 alpha,
0.05 l1_ratio
```



Refinement

I initially chose the k neighors based on the sqroot of the total number of samples. I then decided to iterate over 20 values around k to find the minimum RMSE scores. The K respect to the minimum RMSE value is then choses as the final K neighors.

In case of NMF, I started with 2 latent features and then iterated over 40 consecutive p values find the minimum RMSE scores. The K respect to the minimum RMSE value is then chosen as the final p latent factor.

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

The k neighors and p latent factors are obtained from the training dataset.

We then use the optimized parameters and implement the models with test dataset.

The RMSE score for KNN in th etest dataset with k neighors is 3 The RMSE score for the NMF dataset with p latent factors is 2.

In [12]:
```
k=32

item_similarity_knn=get_item_similarity_knn(train_data,k)
final=join_groundtruth_predicted_value(item_similarity_knn,raw_data,test_data)
rmse=sqrt(mean_squared_error(final.overall_pred.astype(int), final.overall.ast
ype(int)))


print ('Item-based CF RMSE for Test Set using knn: ' + str(rmse) + ' for ' + s
tr(k) + ' neighors')
```

```
Item-based CF RMSE for Test Set using knn: 3.4342412473476323 for 32 neighors
```

In [11]:
```
p=300
nmf = NMF(n_components =p,alpha=0,l1_ratio=0)
W = nmf.fit(train_data);
final_nf=join_groundtruth_predicted_value_NF(nmf,raw_data,test_data,p)
rmse_nf = sqrt(mean_squared_error(final_nf.overall_pred, final_nf.overall))
print ('Item-based CF RMSE for Test Set using NF: ' + str(rmse_nf) + ' for ' +
 str(p) + ' latent features')
```

```
Item-based CF RMSE for Test Set using NF: 2.622810632770465 for 300 latent fe
atures
```

In [6]:
```
brand_new_data_step1= pd.read_csv("item_dedup.csv", skiprows=7000000,nrows=100
0000, header=None)
brand_new_data_step1.columns = ['reviewerID', 'asin','overall','reviewTime']
brand_new_data_step1.drop(['reviewTime'], axis = 1, inplace = True)

brand_new_data=brand_new_data_step1.assign(rnk=brand_new_data_step1.groupby([
'overall'])['asin']
                    .rank(method='min', ascending=False)) .query('rnk < 10000
0')
brand_new_data.append(final_data)
brand_new_data.drop(['rnk'], axis = 1, inplace = True)

brand_new_data_pivot=pd.pivot_table(brand_new_data, values = 'overall', index=
'reviewerID', columns ='asin')
brand_new_data_pivot.fillna(0, inplace=True)



p=300
a=0
l1=0
nmf = NMF(n_components =p, random_state=r,alpha=a,l1_ratio=l1)
W = nmf.fit(brand_new_data_pivot);
final_nf=join_groundtruth_predicted_value_NF(nmf,brand_new_data,brand_new_data
_pivot,p)
rmse_nf = sqrt(mean_squared_error(final_nf.overall_pred, final_nf.overall))
print ('Item-based CF RMSE for Test Set using NF: ' + str(rmse_nf) + ' for ' +
 str(p) + ' latent features')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-63544584d992> in <module>()
      7 brand_new_data.drop(['rnk'], axis = 1, inplace = True)
      8 
----> 9 brand_new_data_pivot=pd.pivot_table(brand_new_data, values = 'overal
l', index='reviewerID', columns ='asin')
     10 brand_new_data_pivot.fillna(0, inplace=True)
     11 

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\pivot.py in pivot
_table(data, values, index, columns, aggfunc, fill_value, margins, dropna, ma
rgins_name)
    140         to_unstack = [agged.index.names[i] or i
    141                       for i in range(len(index), len(keys))]
--> 142         table = agged.unstack(to_unstack)
    143 
    144     if not dropna:

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\frame.py in unstack(self,
 level, fill_value)
   3952         """
   3953         from pandas.core.reshape.reshape import unstack
-> 3954         return unstack(self, level, fill_value)
   3955 
   3956     _shared_docs['melt'] = ("""

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\reshape.py in uns
tack(obj, level, fill_value)
    447 def unstack(obj, level, fill_value=None):
    448     if isinstance(level, (tuple, list)):
--> 449         return _unstack_multiple(obj, level)
    450 
    451     if isinstance(obj, DataFrame):

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\reshape.py in _un
stack_multiple(data, clocs)
    337         dummy.index = dummy_index
    338 
--> 339         unstacked = dummy.unstack('__placeholder__')
    340         if isinstance(unstacked, Series):
    341             unstcols = unstacked.index

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\frame.py in unstack(self,
 level, fill_value)
   3952         """
   3953         from pandas.core.reshape.reshape import unstack
-> 3954         return unstack(self, level, fill_value)
   3955 
   3956     _shared_docs['melt'] = ("""

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\reshape.py in uns
tack(obj, level, fill_value)
    451     if isinstance(obj, DataFrame):
    452         if isinstance(obj.index, MultiIndex):
--> 453             return _unstack_frame(obj, level, fill_value=fill_value)
    454         else:
```

```
    455             return obj.T.stack(dropna=False)

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\reshape.py in _un
stack_frame(obj, level, fill_value)
    494         unstacker = _Unstacker(obj.values, obj.index, level=level,
    495                                value_columns=obj.columns,
--> 496                                fill_value=fill_value)
    497         return unstacker.get_result()
    498

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\reshape.py in __i
nit__(self, values, index, level, value_columns, fill_value)
    108
    109         self._make_sorted_values_labels()
--> 110         self._make_selectors()
    111
    112     def _make_sorted_values_labels(self):

~\Anaconda3\envs\py36\lib\site-packages\pandas\core\reshape\reshape.py in _ma
ke_selectors(self)
    142
    143         selector = self.sorted_labels[-1] + stride * comp_index + sel
f.lift
--> 144         mask = np.zeros(np.prod(self.full_shape), dtype=bool)
    145         mask.put(selector, True)
    146

ValueError: negative dimensions are not allowed
```

In [30]:

```
p=300
r_state=range(43,600,30)
rmse_r_state=[]
for r in r_state:
    nmf = NMF(n_components =p, random_state=r,alpha=a,l1_ratio=l1)
    W = nmf.fit(train_data);
    final_nf=join_groundtruth_predicted_value_NF(nmf,raw_data,test_data,p)
    rmse_nf = sqrt(mean_squared_error(final_nf.overall_pred, final_nf.overall
))
    rmse_r_state.append(rmse_nf)
    print ('Item-based CF RMSE for Test Set using NF: ' + str(rmse_nf) + ' for
 ' + str(p) + ' neighors')


r_state_list=(list(r_state))
plt.plot(r_state_list, rmse_r_state)
plt.xlabel('Random State with latent features 300')
plt.ylabel('RMSE')
plt.show()

from statistics import mean,variance
mean=mean(rmse_r_state)
var=variance(rmse_r_state)
mean,'{:f}'.format(var)
```

Out[30]: (1.1937712652986678, '0.000016')

In [ ]:
```
p=300
r_state=range(43,600,30)
rmse_r_state=[]
for r in r_state:
    nmf = NMF(n_components =p, random_state=r)
    W = nmf.fit(train_data);
    final_nf=join_groundtruth_predicted_value_NF(nmf,raw_data,test_data,p)
    rmse_nf = sqrt(mean_squared_error(final_nf.overall_pred, final_nf.overall
))
    rmse_r_state.append(rmse_nf)
    print ('Item-based CF RMSE for Test Set using NF: ' + str(rmse_nf) + ' for
 ' + str(p) + ' neighors')


r_state_list=(list(r_state))
plt.plot(r_state_list, rmse_r_state)
plt.xlabel('Random State with latent features 300')
plt.ylabel('RMSE')
plt.show()

from statistics import mean,variance
mean=mean(rmse_r_state)
var=variance(rmse_r_state)
mean,'{:f}'.format(var)
```

Justification

The results obtained are reasonalbly good buy further improvement can be done.

Comparing our results with those in our Benchmark RMSE score in KNN, we can say that the RMSE score has improved significantly for our NMF Model.

V. Conclusion

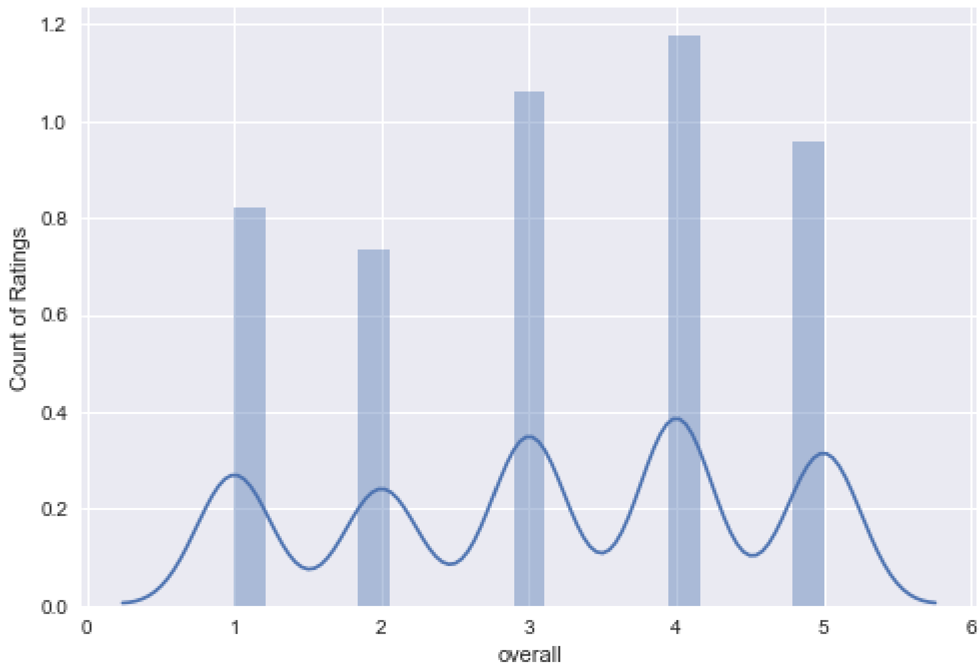
(approx. 1-2 pages)

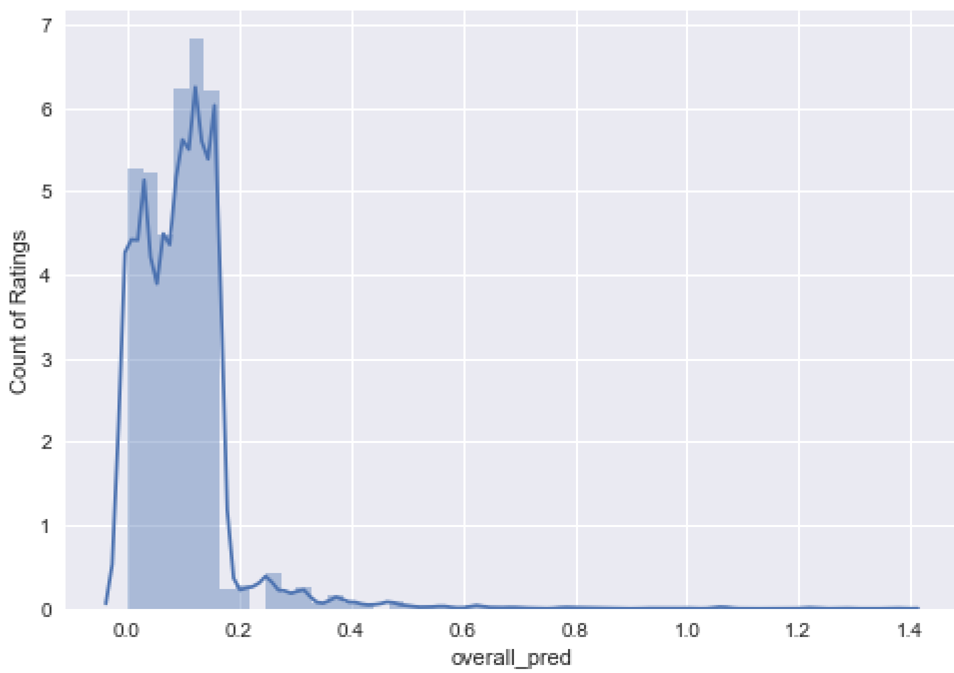
Free-Form Visualization

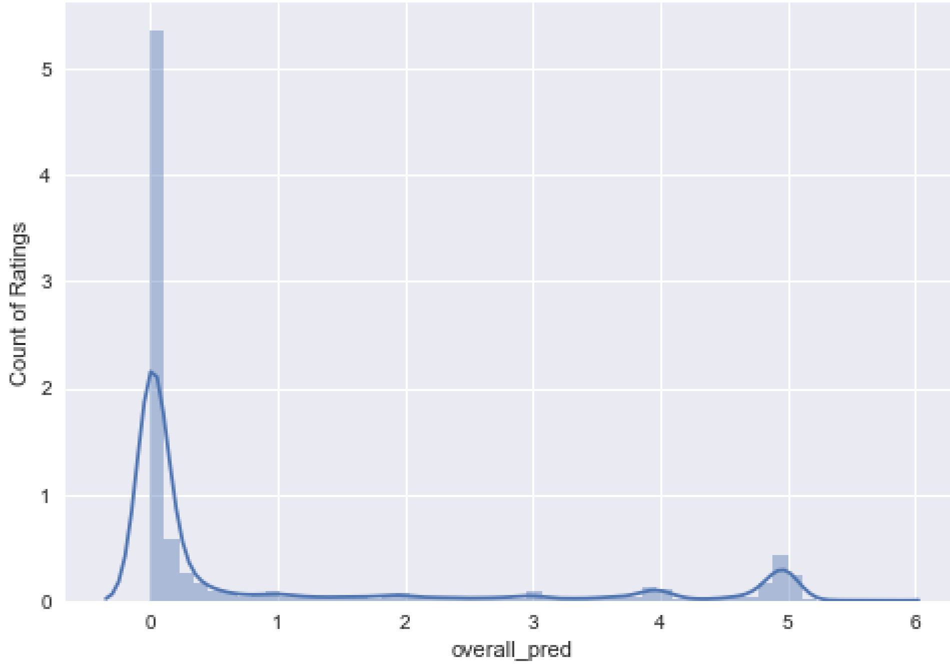
The charts below visualizes the prediction accuracy of the KNN and NMF dataset and compares it visually with the original dataset.

We also add the predicted rating of the unrated items.

In [13]:
```
sns.set(color_codes=True)
plt.figure('overall')
plt.ylabel('Count of Ratings')
plt.xlabel('Ratings 1 to 5')
sns.distplot(final['overall'])
plt.show()


sns.set(color_codes=True)
plt.figure('overall')
plt.ylabel('Count of Ratings')
plt.xlabel('Ratings 1 to 5')
sns.distplot(final['overall_pred'])
plt.show()

sns.set(color_codes=True)
plt.figure('overall')
plt.ylabel('Count of Ratings')
plt.xlabel('Ratings 1 to 5')
sns.distplot(final_nf['overall_pred'])
plt.show()
```


Count of Ratings
overall
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4
5
6


Count of Ratings
overall_pred
0
1
2
3
4
5
6
7
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4



Reflection

The Udacity ML Nanodegree Capstone Project has been a challenging yet enriching experience. Each section has been challenging in its way but implementation section has been the most challenging for me. Besides handling 82 milion dataset and reducing it down to a reasonable number that my PC can handle has been a time consuming process as well.

Improvement

There are several ways the model can be improved, given more time and more powerful Server:

1. Value of k in k neighors and p latent factors in NMF can be optimized by by implementing K fold cross validation. BY splitting the dataset in K folds and then choosing random K-1 folds at a time, K RMSE scores can be generated for a specific k or p values. The k/p value with the lowest average RMSE score can then be marked as optimized and used in the final dataset.
2. A few other We can also try a few other matrix factorization algorithm such as SVD and bench mark the performance as well.

Since I have have the timestamp for each review, I can use recurrent neural network to provide session based recommendation which will give recommendation on immediate products the user might be interested in.