Final Report

December 19, 2019

1 Final Project for Personalization

1.1 0. Importing libs

we are using several common libs includes: - pandas - numpy - sklearn - scipy
to deal with general data structure and also fastFM for fastorization machine.

In [280]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import pickle
import sys
import json
import collections
import os
import time
import bisect


from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LinearRegression

import scipy
from scipy.sparse import csr_matrix, hstack, vstack

from util import load_json, get_sum_count
from tqdm import tqdm_notebook
from fastFM import als

from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
from pyspark.sql.types import IntegerType
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
```

1.2 1. Data Preprocessing

In this section we first decompress all the json file with self implemented function imported from util.py and then we filter out unactive user and split the dataset into train and test data. Also we generate several auxilliry dataframe for group statistics for further usage.

1.2.1 1.1 Json Data Decompress

```
In [3]: for file_name in ['review','business','checkin','tip','user','photo']:
            load_json(file_name)
```

```
In [4]: review_df = pd.read_csv('./yelp_dataset/review.csv')
        review_df.head(3)
```

```
Out[4]:               business_id  cool                 date  funny  \
        0  ujmEBvifdJM6h6RLv4wQIg     0  2013-05-07 04:34:36    1.0
        1  NZnhc2sEQy3RmzKTZnqtwQ     0  2017-01-14 21:30:33    0.0
        2  WTqjgwHlXbSFevF32_DJVw     0  2016-11-09 20:09:03    0.0

                        review_id  stars  \
        0  Q1sbwvVQXV2734tPgoKj4Q    1.0
        1  GJXCdrto3ASJOqKeVWPi6Q    5.0
        2  2TzJjDVDEuAW6MR5Vuc1ug    5.0

                                                        text  useful  \
        0  Total bill for this horrible service? Over $8G...     6.0
        1  I *adore* Travis at the Hard Rock's new Kelly ...     0.0
        2  I have to say that this office really has it t...     3.0

                          user_id
        0  hG7b0MtEbXx5QzbzE6C_VA
        1  yXQM5uF2jS6es16SJzNHfg
        2  n6-Gk65cPZL6Uz8qRm3NYw
```

```
In [5]: business_df = pd.read_csv('./yelp_dataset/business.csv')
        business_df.head(3)
```

```
Out[5]:                       address  \
        0  2818 E Camino Acequia Drive
        1         30 Eglinton Avenue W
        2    10110 Johnston Rd, Ste 15

                                                   attributes             business_id  \
        0                               {'GoodForKids': 'False'}  1SWheh84yJXfytovILXOAQ
        1  {'RestaurantsReservations': 'True', 'GoodForMe...  QXAEGFB4oINsVuTFxEYKFQ
        2  {'GoodForKids': 'True', 'NoiseLevel': "u'avera...  gnKjwL_1w79qoiV3IC_xQQ

                                                  categories         city  \
        0                                      Golf, Active Life      Phoenix
```

```
1  Specialty Food, Restaurants, Dim Sum, Imported...  Mississauga
2                 Sushi Bars, Restaurants, Japanese    Charlotte

                                               hours  is_open   latitude  \
0                                                NaN        0  33.522143
1  {'Monday': '9:0-0:0', 'Tuesday': '9:0-0:0', 'W...        1  43.605499
2  {'Monday': '17:30-21:30', 'Wednesday': '17:30-...        1  35.092564

    longitude                          name postal_code  review_count  stars  \
0 -112.018481   Arizona Biltmore Golf Club       85016             5    3.0
1  -79.652289   Emerald Chinese Restaurant     L5R 3E7           128    2.5
2  -80.859132  Musashi Japanese Restaurant       28210           170    4.0

  state
0    AZ
1    ON
2    NC
```

In [6]:
```python
user_df = pd.read_csv('./yelp_dataset/user.csv')
user_df.head(3)
```

```
/usr/local/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785: DtypeWarning: Col
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[6]:
```
   average_stars  compliment_cool  compliment_cute  compliment_funny  \
0           4.03                1                0                 1
1           3.63                1                0                 1
2           3.71                0                0                 0

   compliment_hot  compliment_list  compliment_more  compliment_note  \
0               2                0                0                1
1               1                0                0                0
2               0                0                0                1

   compliment_photos  compliment_plain          ...              cool  \
0                  0                 1          ...                25
1                  0                 0          ...                16
2                  0                 0          ...                10

            elite  fans                                            friends  \
0  2015,2016,2017     5  c78V-rj8NQcQjOI8KP3UEA, alRMgPcngYSCJ5naFRBz5g...
1             NaN     4  kEBTgDvFX754S68FllfCaA, aB2DynOxNOJK9st2ZeGTPg...
2             NaN     0  4N-HU_T32hLENLntsNKNBg, pSY2vwWLgWfGVAAiKQzMng...

   funny    name  review_count useful                 user_id  \
0     17  Rashmi            95     84  l6BmjZMeQD3rDxWUbiAiow
1     22   Jenna            33     48  4XChL029mKr5hydo79Ljxg
```

```
2      8   David           16     28  bc8C_eETBWL0olvFSJJd0w

         yelping_since
0  2013-10-08 23:11:33
1  2013-02-21 22:29:06
2  2013-10-04 00:16:10

[3 rows x 22 columns]
```

1.2.2 1.2 Unactive User Filter Out & Train/Test Split

1.2.1 Unactive Filter

```
In [21]: user_count = review_df['user_id'].value_counts()
         user_count = pd.DataFrame(user_count)
         user_count = user_count[user_count['user_id'] >= 5]
         user_count = user_count.reset_index()
         user_count.columns = ['user_id','count']
```

```
In [23]: active_review_df = review_df.merge(user_count,on = 'user_id', how = 'inner')
```

```
In [24]: print(str(len(user_count)*1.0/len(review_df['user_id'].value_counts())*100) + '% of us
         print(str(len(active_review_df)*1.0/len(review_df)*100) + '% of reviews are rated by a
```

```
17.47745150378282% of user are active user
67.87820102657801% of reviews are rated by active user
```

As for here, we can see that only 17% users are active user and 17% user generated 67% reviews, which indicates a long-tail effect within the dataset.

Train/Test Split To split the last comment and rating of each active user, we need to sort all user by the review time within each user group. Here we user dataframe groupby and apply function to extremely shorten the time consumption

```
In [25]: sorted_group = active_review_df.sort_values('date').groupby('user_id')


         start = time.time()
         train_df = sorted_group.apply(lambda x:x[:-1])
         print('finished in '+ str(time.time() - start) + ' sec.')


         start = time.time()
         test_df = sorted_group.apply(lambda x:x[-1:])
         print('finished in '+ str(time.time() - start) + ' sec.')
```

```
finished in 377.8252320289612 sec.
finished in 343.2567129135132 sec.
```

And we save train and test data for further usage

```
In [26]: train_df = train_df.reset_index(drop = True)
         train_df.to_csv('train.csv')
         test_df = test_df.reset_index(drop = True)
         test_df.to_csv('test.csv')
```

```
In [7]: train_df = pd.read_csv('./train.csv',index_col = 0)
        test_df = pd.read_csv('./test.csv',index_col = 0)
```

```
/usr/local/lib/python3.6/site-packages/numpy/lib/arraysetops.py:472: FutureWarning: elementwis
  mask |= (ar1 == a)
```

1.2.3 1.3 Calculate Sum Dict

sum_dict is a dictionary that contain the rating information for each user for each user_id as a key, the value is another dictionary contain the count of how many ratings scores are under (included) 2,3,4,5 and 6 reletively. This is used for calculating the rank accuracy for each test data point. For instance, when we have a test sample with score of 4 and we predict it as 3, and suppose the user is user A Then the rank of the original score is the sum of the scores rated by A under 4, and the rank of the predicted score is the sum of the scores rated by A under 3. Then we calculated the rank correlation to get the overall rank accuracy. See example below for the sum_dict

```
In [8]: sum_dict = get_sum_count(train_df)
```

```
HBox(children=(IntProgress(value=0, max=286130), HTML(value='')))
```

```
In [9]: sum_dict["---1lKK3aKOuomHnwAkAow"]
```

```
Out[9]: [17, 22, 32, 55, 127]
```

As we can see above, for user "—1lKK3aKOuomHnwAkAow", he or she has 17 score 1, 5 score 2, 10 score 3, 23 score 4 and 72 score 5. So that within all his or her ratings, there are 17 reviews have a score under 2 and 22 reviews have a score under 3 and so on. Thus, when we have a new predicted score come in, we can quickly give the rank of that prediction by reading the corresponding summation of the count of the reviews with specific score. And then accelerate the calculating of rank accuracy

1.3 2. Data Visualization

In this section, we explore the dataset and generate several figure about the data trying to show sme insight of the dataset.

1.3.1 2.1 Active Users

In [44]:
```
active_user_df = user_df.merge(user_count,on = 'user_id', how = 'inner')
```

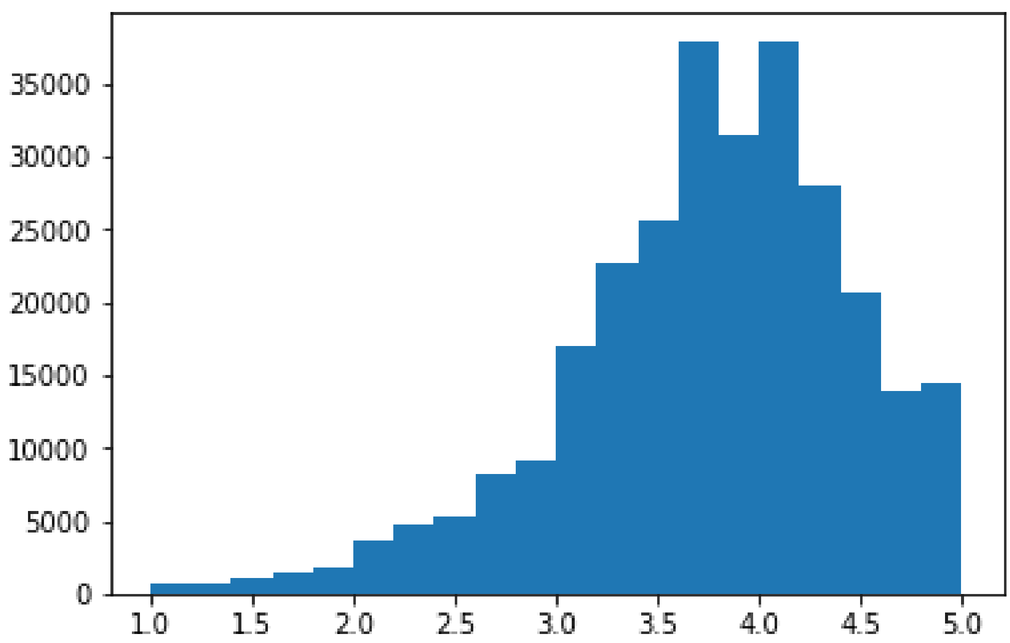
In [45]:
```
active_user_df.describe()[['average_stars','review_count','useful']]
```

Out[45]:

	average_stars	review_count	useful
count	286130.000000	286130.000000	286130.000000
mean	3.748822	60.538759	137.842348
std	0.706024	156.690863	1058.085528
min	1.000000	1.000000	0.000000
25%	3.380000	9.000000	5.000000
50%	3.830000	16.000000	14.000000
75%	4.220000	43.000000	45.000000
max	5.000000	13278.000000	154202.000000

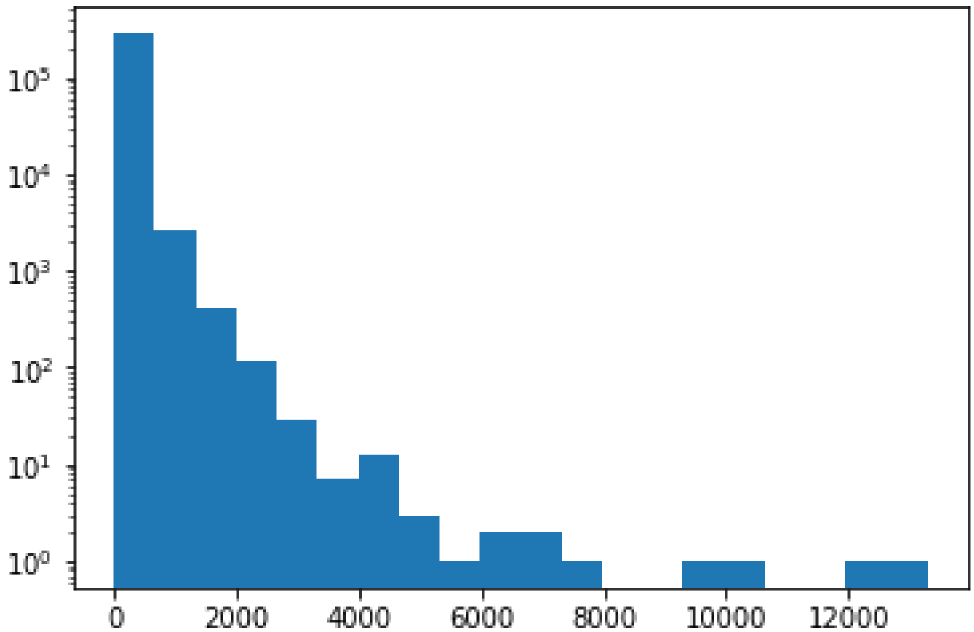
As we can see here, the statistics of average stars, review count and useful are shown as above. In fact people tends to rating 3.74 which is higher than 3.0 as the average overall ratings. And in general for each user they may contribute 22 reviews and there are even a user contribute 13278 reviews(that's incredible actual, saying the user had been an active user for 10 years and contribute review everyday within last 10 years, he or she still need to contribute aroun 3 to 4 reviews per day). And useful is another important statistic, for us to evaluate the importance of an user of a review. Most people have a 0 useful and a few have a high useful, it's like a social media.

In [46]:
```
plt.hist(active_user_df['average_stars'],bins = 20)
plt.show()
```



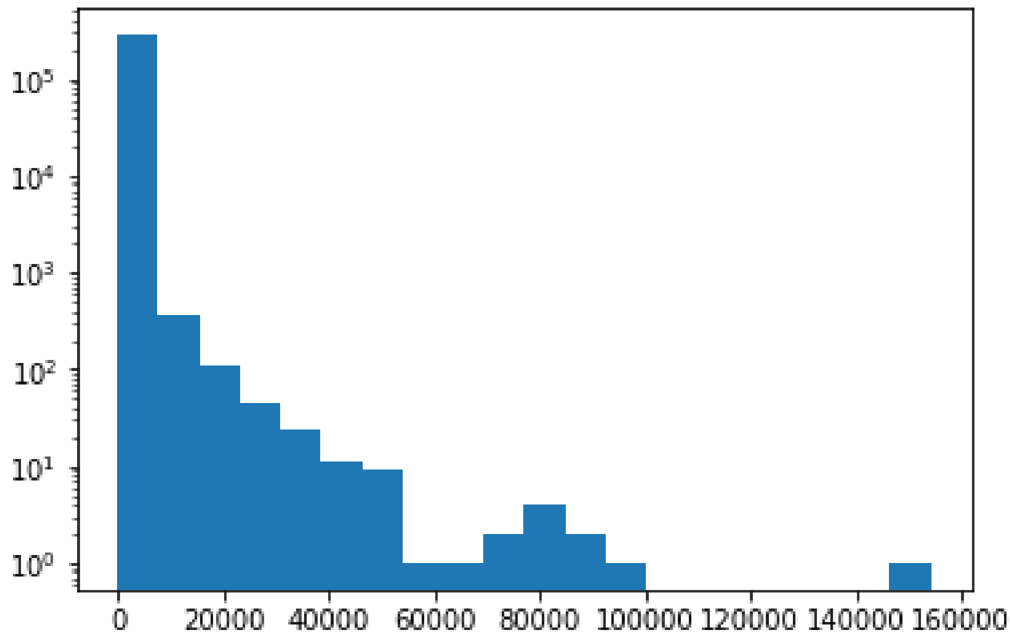
Average stars seems like a right skewed normal distribution.

```
In [48]: plt.hist(active_user_df['review_count'],bins = 20)
         plt.yscale('log')
         plt.show()b
```



To show this clearly, we change the scale for y axis as log-scale. Obviously, there is a long tail effect in this data

```
In [49]: plt.hist(active_user_df['useful'],bins = 20)
         plt.yscale('log')
         plt.show()
```



similar to the review count, useful is long-tailed

1.4 2.2 Business

```
In [50]: business_df.describe()
```

```
Out[50]:               is_open       latitude      longitude   review_count  \
         count  192609.000000  192609.000000  192609.000000  192609.000000
         mean        0.823040      38.541803     -97.594785      33.538962
         std         0.381635       4.941964      16.697725     110.135224
         min         0.000000      33.204642    -115.493471       3.000000
         25%         1.000000      33.637408    -112.274677       4.000000
         50%         1.000000      36.144815    -111.759324       9.000000
         75%         1.000000      43.602989     -79.983614      25.000000
         max         1.000000      51.299943     -72.911982    8348.000000

                        stars
         count  192609.000000
         mean        3.585627
         std         1.018458
         min         1.000000
         25%         3.000000
         50%         3.500000
         75%         4.500000
         max         5.000000
```

There are 82% business are still open, however, including the closed business may help us build better recommend system.

1.5 2.3 Review

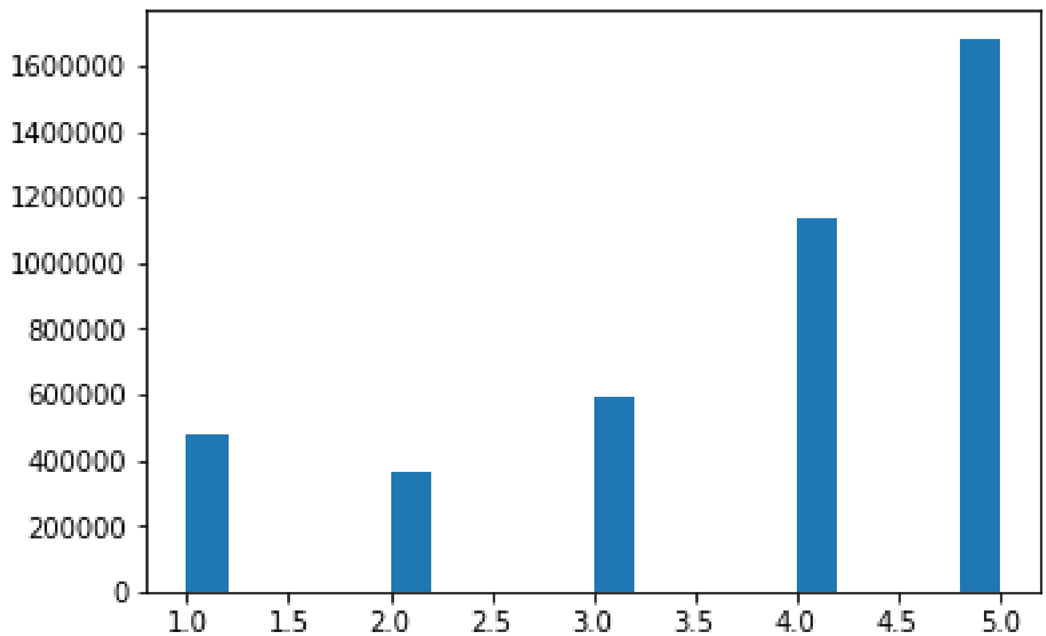
In [53]:
```
train_df.describe()
```

Out[53]:

	cool	funny	stars	useful	count
count	4.252140e+06	4.252140e+06	4.252140e+06	4.252140e+06	4.252140e+06
mean	7.766306e-01	6.230369e-01	3.748399e+00	1.626710e+00	8.740699e+01
std	2.803070e+00	2.597718e+00	1.351254e+00	3.960639e+00	2.116509e+02
min	-1.000000e+00	0.000000e+00	1.000000e+00	-1.000000e+00	5.000000e+00
25%	0.000000e+00	0.000000e+00	3.000000e+00	0.000000e+00	1.100000e+01
50%	0.000000e+00	0.000000e+00	4.000000e+00	1.000000e+00	2.600000e+01
75%	1.000000e+00	0.000000e+00	5.000000e+00	2.000000e+00	7.900000e+01
max	2.900000e+02	9.700000e+02	5.000000e+00	1.241000e+03	4.129000e+03

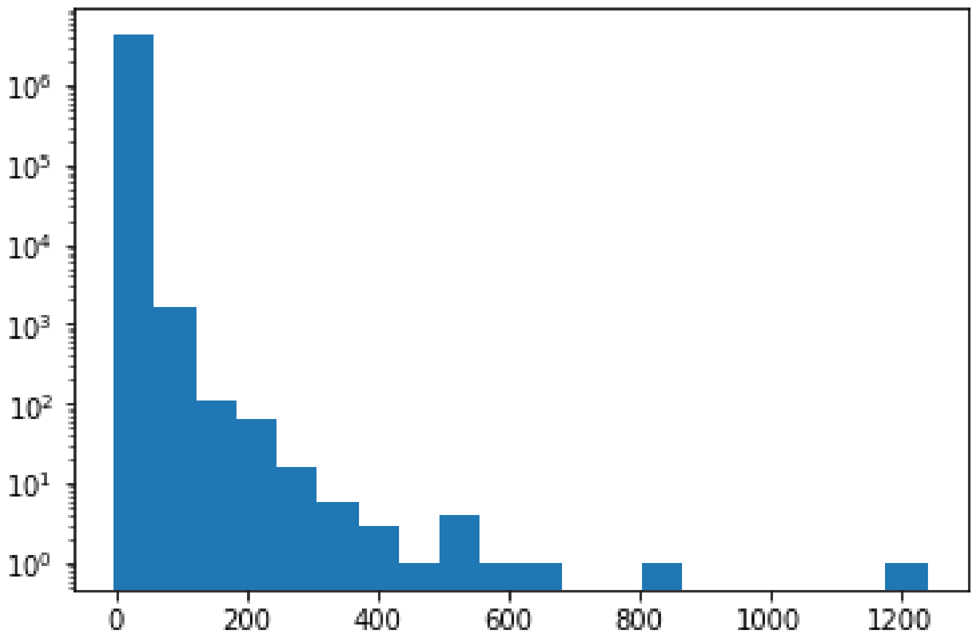
Similar to the useful in active users, reviews have cool and funny attribute, but also seems long tail.

In [57]:
```
plt.hist(train_df['stars'],bins = 20)
plt.show()
```

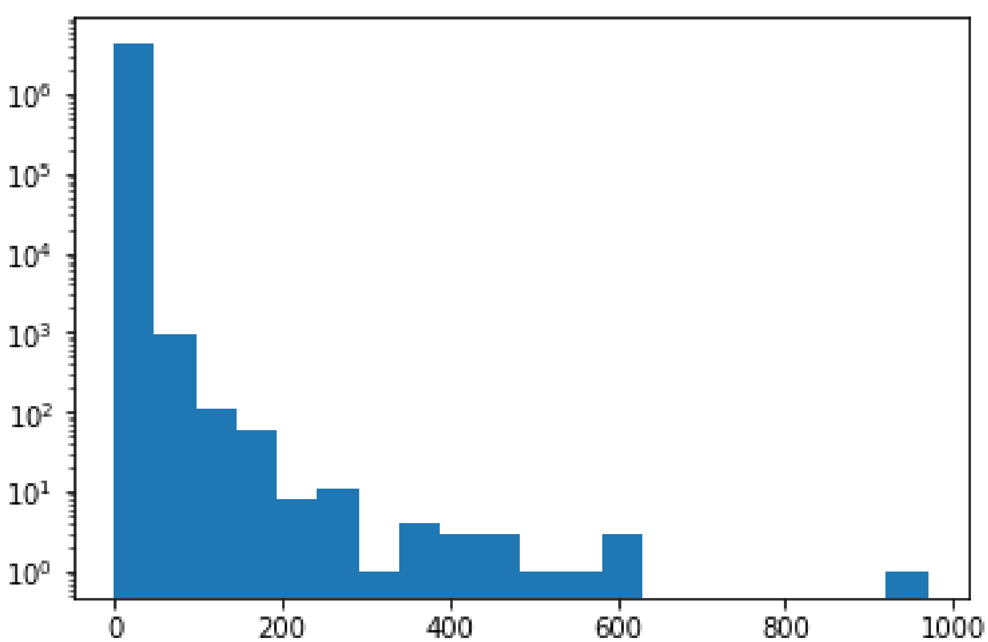


Not same as average stars for each user, the stars distribution of reviews are not normal distribution, bu a more complicated one. People tends to rate more extreme score than average case. We can see rather than rate a 2, people tends to directly give a 1 instead or a 3.

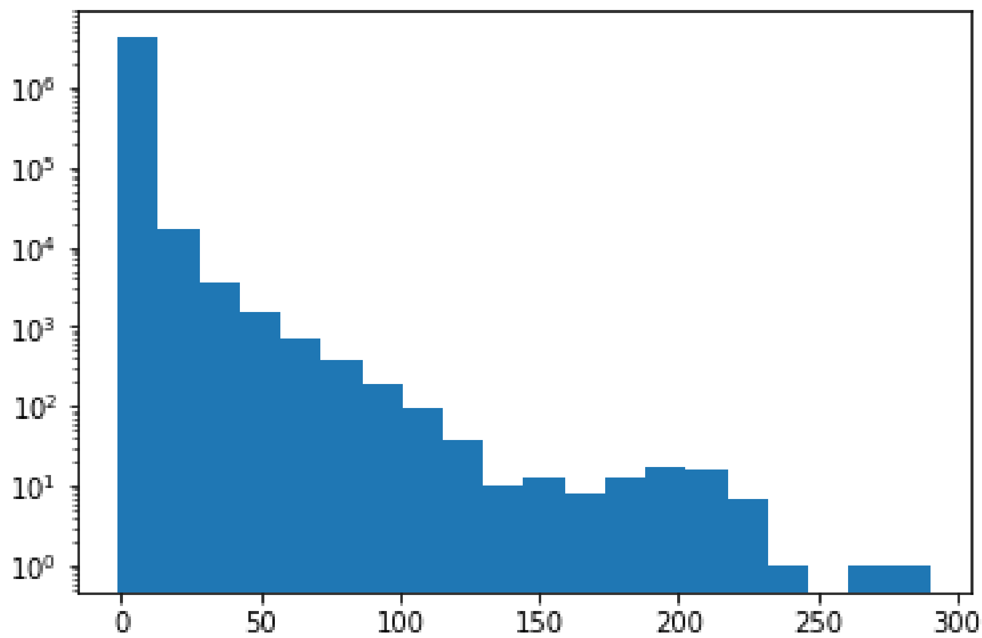
In [56]:
```
plt.hist(train_df['useful'],bins = 20)
plt.yscale('log')
plt.show()
```




```
In [58]: plt.hist(train_df['funny'],bins = 20)
         plt.yscale('log')
         plt.show()
```



```
In [59]: plt.hist(train_df['cool'],bins = 20)
         plt.yscale('log')
         plt.show()
```



Seems clicking cool attribute action is less concentrate rather than useful or funny.

1.6 3. Baseline

In this section, we will first discuss about the metrics we used to evaluate or recommend system and then talk about several baseline from random guess to a simple collaborative filtering and use the above metrics to have a sense of the difficulty of the problem.

1.6.1 3.1 Metrics

For the problem, we set up 3 method to evaluate our result. Mean Absolute Error, Mean Square Error, and Rank Accuracy. MAE and MSE are easy to understand and easy to implement. These two method evaluate the general error, from different perspective, MAE is the most geenral one and MSE punish more on larger difference. Rank Accuracy, as we discussed a bit in section 1.3, we calculate the correlation score of the rank of true score for each test sample and the rank of predicted score for each test sample. As absolute score may not mean the same for different user, however, rank may be more meaningful.

```
In [92]: def mae(true, pred):
             '''
             true: as a vector
             pred: as a vector
             calculating mean absolute error
```

```
    '''
    assert len(true) == len(pred)
    return np.mean(np.abs(true - pred))


def mse(true, pred):
    '''
    true as a vector
    pred as a vector
    calculating mean square error
    '''
    assert len(true) == len(pred)
    return np.mean(np.abs(true - pred) * np.abs(true - pred))


def get_rank(sum_dict, arraylike):
    score = arraylike['stars']
    user = arraylike['user_id']
    return sum_dict[user][min(max(1,int(score + 0.5)),5) - 1]


def get_rank_pred(sum_dict, arraylike):
    score = arraylike['prediction']
    user = arraylike['user_id']
    return sum_dict[user][min(max(1,int(score + 0.5)),5) - 1]


def rank_accuracy(sum_dict, prediction):
    '''
    input prediction dataframe and make user_id as index
    '''
    rank_true = prediction.apply(lambda x: get_rank(sum_dict, x), axis=1)
    rank_pred = prediction.apply(lambda x: get_rank_pred(sum_dict, x), axis=1)
    return np.corrcoef(rank_true, rank_pred)[0][1]
```

In [86]:
```
def get_metrics(sum_dict,pred_df):
    return (mae(pred_df['stars'],pred_df['prediction']),
            mse(pred_df['stars'],pred_df['prediction']),
            rank_accuracy(sum_dict,pred_df))
```

Code is shown above and also could be found in metrics.py

1.6.2 3.2 Random Guess

We first apply a random guess regressor to have a sense of lower bound of the problem.

In [290]:
```
y_pred = np.random.choice([1,2,3,4,5],size = [len(test_df)])
pred_df = test_df.copy()
pred_df['prediction'] = y_pred
```

In [291]:
```
mae_score,mse_score,rank_accuracy_score = get_metrics(sum_dict,pred_df)
result = pd.DataFrame(columns = ['mae_score','mse_score','rank_accuracy_score'])
result.loc['random_guess'] = [mae_score,mse_score,rank_accuracy_score]
result
```

Out[291]:

	mae_score	mse_score	rank_accuracy_score
random_guess	1.780044	4.904673	0.650275

1.6.3 3.3 Baseline - Matrix Factorization

We then tried to trian a matrix factorization model as our baseline model only using user_id, business_id and stars.

Since we will use ALS in spark, which requires user_id and business_id to be numerical, we first transform user_id and business_id to user_index and business_index

In [281]:
```
train = train_df[['user_id','business_id','stars','date']]
test = test_df[['user_id','business_id','stars','date']]
```

In [282]:
```
n = train_df.shape[0]
m = test_df.shape[0]
train_user_id = train_df['user_id'].unique()
train_business_id = train_df['business_id'].unique()
user_id_index = {}
business_id_index = {}
for i in range(len(train_user_id)):
    user_id_index[train_user_id[i]] = i
for i in range(len(train_business_id)):
    business_id_index[train_business_id[i]] = i

nu = len(user_id_index)
nb = len(business_id_index)
test_user_id = test_df['user_id'].unique()
test_business_id = test_df['business_id'].unique()
for i in range(len(test_user_id)):
    if test_user_id[i] not in user_id_index:
        user_id_index[test_user_id[i]] = nu
        nu += 1
for i in range(len(test_business_id)):
    if test_business_id[i] not in business_id_index:
        business_id_index[test_business_id[i]] = nb
        nb += 1
```

In [283]:
```
train['user_index'] = train.user_id.map(user_id_index)
train['business_index'] = train.business_id.map(business_id_index)
test['user_index'] = test.user_id.map(user_id_index)
test['business_index'] = test.business_id.map(business_id_index)

train = train[['user_index','business_index','stars','date','user_id','business_id']]
test = test[['user_index','business_index','stars','date','user_id','business_id']]
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  after removing the cwd from sys.path.
```

In [284]:
```python
train.to_csv('spark_train.csv', index=False, header=False)
test.to_csv('spark_test.csv', index=False, header=False)
```

Encoding here

In [285]:
```python
sc = SparkContext('local')
spark = SparkSession(sc)

spark_train = spark.read.csv("spark_train.csv")
spark_test = spark.read.csv('spark_test.csv')

spark_train = spark_train.withColumnRenamed('_c0', 'user_index')
spark_train = spark_train.withColumnRenamed('_c1', 'business_index')
spark_train = spark_train.withColumnRenamed('_c2', 'rating')
spark_train = spark_train.withColumnRenamed('_c3', 'date')
spark_train = spark_train.withColumnRenamed('_c4', 'user_id')
spark_train = spark_train.withColumnRenamed('_c5', 'business_id')
spark_train = spark_train.withColumn("user_index", spark_train["user_index"].cast(In
spark_train = spark_train.withColumn("business_index", spark_train["business_index"]
spark_train = spark_train.withColumn("rating", spark_train["rating"].cast('float'))
spark_train.show()
```

```
+----------+--------------+------+-------------------+--------------------+--------------------
|user_index|business_index|rating|               date|             user_id|         business_i
```

```
+----------+-------------+------+-------------------+--------------------+--------------------
|         0|            0|   4.0|2008-11-11 04:31:46|---1lKK3aKOuomHnw...|5cbsjFtrntUAeUx51..
|         0|            1|   4.0|2008-11-11 04:40:05|---1lKK3aKOuomHnw...|--9e1ONYQuAa-CB_R..
|         0|            2|   5.0|2009-01-16 21:49:36|---1lKK3aKOuomHnw...|ifEHr-ZnGFSKgJVsy..
|         0|            3|   4.0|2010-10-16 23:27:02|---1lKK3aKOuomHnw...|kosTPb88O4Q0XGbVb..
|         0|            4|   5.0|2010-10-16 23:31:28|---1lKK3aKOuomHnw...|rq5dgoksPHkJwJNQK..
|         0|            5|   1.0|2010-10-17 04:19:01|---1lKK3aKOuomHnw...|2BbFeotL85cIaBjSq..
|         0|            6|   1.0|2010-11-05 20:08:08|---1lKK3aKOuomHnw...|78TC3sZSYBzBsSJ0z..
|         0|            7|   5.0|2010-11-05 21:49:42|---1lKK3aKOuomHnw...|qtOt27b-3Re6zM5OS..
|         0|            8|   5.0|2010-11-05 21:58:29|---1lKK3aKOuomHnw...|CWNMLT-ppaUjLMmrn..
|         0|            0|   1.0|2010-11-05 22:12:55|---1lKK3aKOuomHnw...|5cbsjFtrntUAeUx51..
|         0|            9|   4.0|2010-11-09 20:09:42|---1lKK3aKOuomHnw...|DXlDzOcpdUE_F21to..
|         0|           10|   5.0|2010-11-09 20:11:47|---1lKK3aKOuomHnw...|KskYqH1Bi7Z_61pH6..
|         0|           11|   1.0|2010-11-09 20:18:19|---1lKK3aKOuomHnw...|eduRavkml8awmPach..
|         0|           12|   5.0|2010-11-09 20:21:52|---1lKK3aKOuomHnw...|-ErwgUmZ1-jHW_rSu..
|         0|           13|   5.0|2010-11-09 20:24:31|---1lKK3aKOuomHnw...|ow5ku7hfMqU94mylT..
|         0|           14|   3.0|2010-11-09 20:34:20|---1lKK3aKOuomHnw...|vW65SNLam99SyOuVa..
|         0|           15|   5.0|2010-11-16 03:11:16|---1lKK3aKOuomHnw...|RRw9I8pHt5PzgYGT2..
|         0|           16|   3.0|2010-11-16 03:15:16|---1lKK3aKOuomHnw...|iPM85PQMs7QoAxw-O..
|         0|            2|   2.0|2010-11-16 03:23:07|---1lKK3aKOuomHnw...|ifEHr-ZnGFSKgJVsy..
|         0|           17|   4.0|2010-11-16 03:27:20|---1lKK3aKOuomHnw...|pwLQfEe_yJYwAWWug..
+----------+-------------+------+-------------------+--------------------+--------------------
only showing top 20 rows
```

In [286]:
```
spark_test = spark_test.withColumnRenamed('_c0', 'user_index')
spark_test = spark_test.withColumnRenamed('_c1', 'business_index')
spark_test = spark_test.withColumnRenamed('_c2', 'rating')
spark_test = spark_test.withColumnRenamed('_c3', 'date')
spark_test = spark_test.withColumnRenamed('_c4', 'user_id')
spark_test = spark_test.withColumnRenamed('_c5', 'business_id')
spark_test = spark_test.withColumn("user_index", spark_test["user_index"].cast(Intege
spark_test = spark_test.withColumn("business_index", spark_test["business_index"].cas
spark_test = spark_test.withColumn("rating", spark_test["rating"].cast('float'))
spark_test.show()
```

```
+----------+-------------+------+-------------------+--------------------+--------------------
|user_index|business_index|rating|               date|             user_id|         business_id
+----------+-------------+------+-------------------+--------------------+--------------------
|         0|          121|   5.0|2018-10-11 23:29:57|---1lKK3aKOuomHnw...|Hqs4YNST_ZHbshwyi..
|         1|        73463|   2.0|2014-04-21 16:58:28|--0kuuLmuYBe3Rmu0...|PYe_FDw6QTbTf66Wc..
|         2|        29288|   5.0|2018-10-04 02:02:28|--2HUmLkcNHZp0xw6...|KW9RNyBPmc77f9FsO..
|         3|         3746|   3.0|2018-01-11 04:24:17|--2vR0DIsmQ6WfcSz...|BLIJ-p5wYuAhw6Pp6..
|         4|        15598|   4.0|2018-09-03 19:32:11|--3WaS23LcIXtxyFU...|UKrfUw8quQiQM2N9i..
|         5|        12851|   5.0|2018-05-15 19:54:15|--44NNdtngXMzsxyN...|YNf4Yi9l7wa1U8k5K..
|         6|         2272|   3.0|2012-06-17 16:59:06|--4q8EyqThydQm-eK...|rcaPajgKOJC2vo_l3..
|         7|        22441|   1.0|2018-11-12 20:37:07|--4rAAfZnEIAKJE80...|HTaA1mo9cB1dXMwfJ..
```

```
|         8|        10397|   4.0|2018-05-24 21:19:54|--7gjElmOrthETJ8X...|UxWH8zRYIBgs6Q2oy..
|         9|         4345|   2.0|2016-08-24 14:55:34|--Br-QsbO9ad5GbZx...|x6PA-2j7LpZAYFo2V..
|        10|        21192|   3.0|2018-10-20 17:56:22|--BumyUHiO_7YsHur...|OL3CAgRf8wuH5MjNw..
|        11|        17407|   5.0|2015-12-11 16:31:56|--CH8yRGXhO2MmbF-...|TZpTyyGvQkKPnt59P..
|        12|         5409|   2.0|2017-11-16 03:38:26|--CIuK7sUpaNzalLA...|g-NKvwy8iLePEQHso..
|        13|         2590|   5.0|2018-10-13 02:26:42|--DxiDMQgNO8E5gTM...|C8D_GU9cDDjbOJfCa..
|        14|         9878|   5.0|2014-12-19 23:10:38|--HCoE1ghaAlcaAfs...|0ldxjei8v4q95fApI..
|        15|       176270|   1.0|2018-03-26 07:55:42|--HOeLECewlqBqvFh...|PssNPwuSuOjHAx5WG..
|        16|        74684|   1.0|2018-07-18 00:30:19|--JqEn95hN31sznM1...|12g_z3nFKbN2telbz..
|        17|           59|   4.0|2017-10-19 21:25:53|--LUapetRSkZpFZ2d...|gHoP4eJimaMltfUlp..
|        18|        33893|   5.0|2018-03-05 21:43:05|--NIc98RMssgy0mSZ...|SoUlHQO5jZf2XFHr8..
|        19|       113492|   1.0|2018-10-27 19:02:02|--Nnm_506G_p8MxAO...|vp77iQDlV1OkbjNhm..
+----------+-------------+------+-------------------+--------------------+-------------------
only showing top 20 rows
```

In [287]:
```
als_5 = ALS(maxIter=5, rank = 5, userCol="user_index", itemCol="business_index", rat
          coldStartStrategy="drop")

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")

model_5 = als_5.fit(spark_train)
```

In [288]:
```
predictions_5 = model_5.transform(spark_test)
spark_prediction = predictions_5.toPandas()
spark_prediction.rename(columns={"rating": "stars"}, inplace=True)
```

In [292]:
```
mae_score,mse_score,rank_accuracy_score = get_metrics(sum_dict,spark_prediction)
result.loc['MF(Baseline)'] = [mae_score,mse_score,rank_accuracy_score]
result
```

Out[292]:

	mae_score	mse_score	rank_accuracy_score
random_guess	1.780044	4.904673	0.650275
MF(Baseline)	1.336100	3.154648	0.827526

1.7 4 Feature Engineer

In this section, we deal with part of the original features and convert them into union form. Feature engineering includes modification of following features:

- Average stars, useful, cool and funny of the user
- Count of friends of the user
- Embedded vector of business feature
- Summation of review text vector
- Categorical vector
- Name vector
- Feature vector of user

1.7.1 4.1 user part

```
In [163]: train_df = pd.read_csv('./train.csv',index_col = 0)
          test_df = pd.read_csv('./test.csv',index_col = 0)
```

```
/usr/local/lib/python3.6/site-packages/numpy/lib/arraysetops.py:472: FutureWarning: elementwis
  mask |= (ar1 == a)
```

```
In [167]: y_train = train_df['stars']
          x_train = train_df[['user_id','business_id','text']]
          y_test = test_df['stars']
          x_test = test_df[['user_id','business_id','text']]
```

```
In [168]: x_train.head()
```

```
Out[168]:                    user_id             business_id  \
          0  ---1lKK3aKOuomHnwAkAow  5cbsjFtrntUAeUx51FaFTg
          1  ---1lKK3aKOuomHnwAkAow  --9e1ONYQuAa-CB_Rrw7Tw
          2  ---1lKK3aKOuomHnwAkAow  ifEHr-ZnGFSKgJVsywiAFg
          3  ---1lKK3aKOuomHnwAkAow  kosTPb88O4Q0XGbVbEOGCA
          4  ---1lKK3aKOuomHnwAkAow  rq5dgoksPHkJwJNQKlGQ7w

                                                          text
          0  I like it, and so far I think it is one of the...
          1  So when you go to a restaurant like this pleas...
          2  The Wild Boar was amazing, so good my husband ...
          3  While its not Lotus it was tasty an the women ...
          4  Best coffee in town, they brew each cup. If yo...
```

```
In [12]: user_df = pd.read_csv('./yelp_dataset/user.csv')
         business_df = pd.read_csv('./yelp_dataset/business.csv')
```

```
/usr/local/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785: DtypeWarning: Co
  interactivity=interactivity, compiler=compiler, result=result)
```

```
In [13]: user_df['friends_count'] = user_df['friends'].apply(lambda x:len(x.split(',')) if x e
         for col in ['cool','fans','funny','useful']:
             user_df['average_' + col] = user_df[col] / user_df['review_count']
```

```
In [169]: x_train = x_train.reset_index().merge(user_df[['user_id','average_stars','average_co
                                on = 'user_id',how= 'inner').set_index('index')
          x_test = x_test.reset_index().merge(user_df[['user_id','average_stars','average_cool
                                on = 'user_id',how= 'inner').set_index('index')
```

1.7.2 4.2 business part

4.2.1 Text

```
In [172]: x_train['text'].fillna("",inplace = True)
```

```
In [173]: if 'train_text_tfidf_1' in os.listdir('./pickle/') and 'train_text_tfidf_2' in os.lis
              f = open('./pickle/train_text_tfidf_1','rb')
              train_text_tfidf_1 = pickle.load(f)
              f.close()
              f = open('./pickle/train_text_tfidf_2','rb')
              train_text_tfidf_2 = pickle.load(f)
              f.close()
              f = open('./pickle/test_text_tfidf','rb')
              test_text_tfidf = pickle.load(f)
              f.close()
              train_text_tfidf = vstack((train_text_tfidf_1,train_text_tfidf_2))
          else:
              tfidf_vec = TfidfVectorizer(stop_words='english')
              tfidf_vec.fit(x_train['text'])
              train_text_tfidf = tfidf_vec.transform(x_train['text'])
              test_text_tfidf = tfidf_vec.transform(x_test['text'])
          x_train['text_tfidf'] = [train_text_tfidf[i] for i in tqdm_notebook(range(len(train_c
          x_test['text_tfidf'] = [test_text_tfidf[i] for i in tqdm_notebook(range(len(test_df))
```

```
HBox(children=(IntProgress(value=0, max=4252140), HTML(value='')))
```

```
HBox(children=(IntProgress(value=0, max=286130), HTML(value='')))
```

```
In [174]: del train_text_tfidf_1,train_text_tfidf_2,train_text_tfidf,test_text_tfidf
          import gc
          gc.collect()
```

```
Out[174]: 7
```

```
In [176]: if 'train_text_tfidf_1' not in os.listdir('./pickle/') or 'train_text_tfidf_2' not in
              f = open('./pickle/train_text_tfidf_1','wb')
              pickle.dump(train_text_tfidf[:train_text_tfidf.shape[0]//2],f)
              f.close()
              f = open('./pickle/train_text_tfidf_2','wb')
              pickle.dump(train_text_tfidf[train_text_tfidf.shape[0]//2:],f)
              f.close()
          if 'test_text_tfidf' not in os.listdir('./pickle/'):
              f = open('./pickle/test_text_tfidf','wb')
              pickle.dump(test_text_tfidf,f)
              f.close()
```

```
In [177]: start = time.time()
          if 'business_text_tfidf' not in os.listdir('./pickle/'):
              print('generating')
              business_text_tfidf_df = x_train[['business_id','text_tfidf']].groupby('business_
              business_df['review_text_tfidf'] = business_df['business_id'].map(business_text_
              print(time.time()-start)
              business_text_tfidf = vstack(business_text_tfidf_df['review_text_tfidf'].values)
              f = open('./pickle/business_text_tfidf','wb')
              pickle.dump(train_text_tfidf,f)
              f.close()
          else:
              print('loading')
              f = open('./pickle/business_text_tfidf','rb')
              business_text_tfidf = pickle.load(f)
              f.close()
              business_text_tfidf_df_temp = x_train[['business_id','text_tfidf']].groupby('busi
              business_text_tfidf_df_temp = pd.DataFrame(business_text_tfidf_df_temp,columns =
              business_text_tfidf_df_temp['review_text_tfidf'] = [business_text_tfidf[i] for i
              business_df['review_text_tfidf'] = business_df['business_id'].map(business_text_
              print(time.time()-start)
          x_train = x_train.reset_index().merge(business_df[['business_id','review_text_tfidf']
          x_test = x_test.reset_index().merge(business_df[['business_id','review_text_tfidf']]
          x_train.sort_index(inplace = True)
          x_test.sort_index(inplace = True)
```

```
loading
```

```
HBox(children=(IntProgress(value=0, max=183398), HTML(value='')))
```

```
77.64045786857605
```

```
In [182]: del business_text_tfidf, business_text_tfidf_df_temp
          gc.collect()

Out[182]: 305
```

4.2.2 Categorical and name

```
In [183]: def clean_cate(x):
              if type(x) == str:
                  res = []
                  terms = x.split(',')
                  for t in terms:
                      res.append(t.strip().replace(' ', '_'))
                  # print(res)
```

```
                return ' '.join(res)
            else:
                return ''
```

```
In [184]: business_df['cleaned_categories'] = business_df['categories'].apply(lambda x : clean_
          cate_tfidf_vec = TfidfVectorizer()
          cate_tfidf = cate_tfidf_vec.fit_transform(business_df['cleaned_categories'])
          name_tfidf_vec = TfidfVectorizer()
          name_tfidf = name_tfidf_vec.fit_transform(business_df['name'])
          if 'business_sparse_vec' not in os.listdir('./pickle/'):
              print('generating')
              business_sparse_vec = hstack((cate_tfidf,name_tfidf))
              business_sparse_vec  = business_sparse_vec.tocsr()
              f = open('business_sparse_vec','wb')
              pickle.dump(business_sparse_vec,f)
              f.close()
          else:
              print('loading')
              f = open('./pickle/business_sparse_vec','rb')
              business_sparse_vec = pickle.load(f)
              f.close()
          business_profile = {}
          for i,business in enumerate(tqdm_notebook(business_df['business_id'])):
              business_profile[business] = business_sparse_vec[i]
          business_df['business_profile'] = list(business_profile.values())
```

```
loading
```

```
HBox(children=(IntProgress(value=0, max=192609), HTML(value='')))
```

```
In [185]: x_train = x_train.reset_index().merge(business_df[['business_id','business_profile']]
          x_test = x_test.reset_index().merge(business_df[['business_id','business_profile']],
          x_train.sort_index(inplace = True)
          x_test.sort_index(inplace = True)
```

```
In [186]: del business_sparse_vec
          gc.collect()
```

```
Out[186]: 116
```

1.8 5 Factorization Machine

In this section, we utilize factorization machine as a method to include side information of user and business. We join the features and one-hotted index together to run the FM based on fastFM.

The following cells may take a long time to run the first time to extract TF-iDF vector of text for each review and business. Also may cost amount of space to save the middle result so that accelerating second time running.

Making the sparse matrix for FM

```
In [189]: def build_sparse_matrix_fm(vec,x_df):

              ## build one-hot id
              start = time.time()
              x_sparse_id = vec.transform(x_df[['user_id','business_id']].to_dict(orient = 're
              print(time.time() - start)
              x_sparse_user_stat = scipy.sparse.csr_matrix(x_df[['average_stars','average_cool
              print(time.time() - start)
              x_sparse_business_text = vstack(x_df['text_tfidf'].values)
              print(time.time() - start)
              x_sparse_business_profile = vstack(x_df['business_profile'].values)
              print(time.time() - start)
              assert x_sparse_id.shape[0] == x_sparse_user_stat.shape[0]
              assert x_sparse_business_text.shape[0] == x_sparse_user_stat.shape[0]
              assert x_sparse_user_stat.shape[0] == x_sparse_business_profile.shape[0]

              x_sparse = hstack([x_sparse_id,x_sparse_user_stat,x_sparse_business_text,x_sparse
              print(x_sparse.shape)
              return x_sparse
          vec = DictVectorizer()
          vec.fit(x_train[['user_id','business_id']].to_dict(orient = 'record'))
          x_train_fm = build_sparse_matrix_fm(vec,x_train[:100])
          x_test_fm = build_sparse_matrix_fm(vec,x_test[:100])
```

```
0.002684354782104492
0.004668235778808594
0.012681007385253906
0.015171051025390625
(100, 1093502)
0.002542734146118164
0.003988027572631836
0.012691020965576172
0.014397859573364258
(100, 1093502)
```

```
In [198]: # vec = DictVectorizer()
          # vec.fit(x_train[['user_id','business_id']].to_dict(orient = 'record'))
          if "x_train_fm.npz" not in os.listdir('./pickle/'):
              print('generating')
              x_train_fm = build_sparse_matrix_fm(vec,x_train)
              scipy.sparse.save_npz('./pickle/x_train_fm.npz', x_train_fm)
          else:
```

```
            print('loading')
            x_train_fm = scipy.sparse.load_npz('./pickle/x_train_fm.npz')
        if "x_test_fm.npz" not in os.listdir('./pickle/'):
            print('generating')
            x_test_fm = build_sparse_matrix_fm(vec,x_test)
            scipy.sparse.save_npz('./pickle/x_test_fm.npz', x_test_fm)
        else:
            print('loading')
            x_test_fm = scipy.sparse.load_npz('./pickle/x_test_fm.npz')
```

```
generating
77.54142785072327
79.19007897377014
199.43242406845093
261.8962299823761
(4252140, 1093502)
generating
6.621103286743164
6.77220606803894
16.9272141456604
21.169249057769775
(286130, 1093502)
```

```
In [199]: x_train_fm = scipy.sparse.csr_matrix(x_train_fm)
          x_test_fm = scipy.sparse.csr_matrix(x_test_fm)
```

Training…

```
In [203]: start = time.time()
          fm = als.FMRegression(n_iter=15, init_stdev=0.1,rank = 2, l2_reg_w = 0.1, l2_reg_V =
          fm.fit(x_train_fm,y_train)
          print(str(time.time() - start) + 'sec')
```

```
1172.6893401145935sec
```

inference and evaluate

```
In [293]: y_pred = fm.predict(x_test_fm)
          pred_df = test_df.copy()
          pred_df['prediction'] = np.clip(y_pred,1,5)
          mae_score,mse_score,rank_accuracy_score = get_metrics(sum_dict,pred_df)
          result.loc['Factorization Machine'] = [mae_score,mse_score,rank_accuracy_score]
          result
```

Out[293]:

	mae_score	mse_score	rank_accuracy_score
random_guess	1.780044	4.904673	0.650275
MF(Baseline)	1.336100	3.154648	0.827526
Factorization Machine	0.835237	1.603286	0.839589

As we can see, the FM is much better than baseline and extremely small on mae

1.9 6 Content-Based Method

In this section, we use content based method to rate the business, we extracted TF-iDF vector of business name and categorical in last section and we also extracted TF-iDF vector of each review text and we'll sum them together weighted by stars for each user, so that we can build the user profile.

Finally, we build a regressor to map the user-bias score, business-bias score and the cosine similarity between user profile and business vector representation to the final score.

All metrics mentioned in baseline section are used to evaluate the model performance

```
In [249]: def get_user_profile(x):
              cate = scipy.sparse.csr_matrix(x['stars'].values).dot(vstack(x['business_profile
              return cate
          def get_cosine_similarity(x):
              a = x['business_profile'].toarray().squeeze()
              b = x['user_profile_total'].toarray().squeeze()
              return np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b))
```

```
In [247]: xy_train = pd.concat([x_train,y_train],axis = 1)
          start = time.time()
          user_df['user_profile_total'] = user_df['user_id'].map(xy_train[[
              'user_id',
              'review_text_tfidf',
              'business_profile','stars'
          ]].groupby('user_id').apply(lambda x:get_user_profile(x)))
          print(time.time() - start)

          x_train = x_train.reset_index().merge(user_df[['user_id','user_profile_total']],on =
          x_test = x_test.reset_index().merge(user_df[['user_id','user_profile_total']],on = 'u
```

```
218.85624384880066
```

```
In [258]: business_mean = xy_train[['business_id','stars']].groupby('business_id').apply(np.mea
```

```
In [264]: x_train['cosine_similarity'] = x_train[['business_profile',
                                                  'user_profile_total']].apply(
              lambda x:get_cosine_similarity(x),axis = 1)
          x_train['user_average_stars'] = x_train['average_stars']
          x_train['business_average_stars'] = x_train['business_id'].map(business_mean['stars']
```

```
In [267]: x_test['cosine_similarity'] = x_test[['business_profile',
                                                'user_profile_total']].apply(
              lambda x:get_cosine_similarity(x),axis = 1)
          x_test['user_average_stars'] = x_test['average_stars']
          x_test['business_average_stars'] = x_test['business_id'].map(business_mean['stars'])
```

```
In [273]: x_train_cb = x_train[['user_average_stars','business_average_stars','cosine_similari
          x_test_cb = x_test[['user_average_stars','business_average_stars','cosine_similarity
          x_test_cb.fillna(3.6,inplace = True)
```

```
In [274]: lr = LinearRegression()
          lr.fit(x_train_cb,y_train)
```

```
Out[274]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                   normalize=False)
```

```
In [294]: y_pred = lr.predict(x_test_cb)
          pred_df = test_df.copy()
          pred_df['prediction'] = np.clip(y_pred,1,5)
          mae_score,mse_score,rank_accuracy_score = get_metrics(sum_dict,pred_df)
          result.loc['Content-Based Method'] = [mae_score,mse_score,rank_accuracy_score]
          result
```

Out[294]:

	mae_score	mse_score	rank_accuracy_score
random_guess	1.780044	4.904673	0.650275
MF(Baseline)	1.336100	3.154648	0.827526
Factorization Machine	0.835237	1.603286	0.839589
Content-Based Method	1.103736	1.777739	0.845142

Though the mae and mse score are worse the FM, Content base method achieve better on rank accuracy score