CSC 577 Final Project

DeadPool (Dajeong Jeon & Jonggoo Kang)

In [1]:
```
import pandas as pd
import numpy as np
import json
from matplotlib import pyplot as plt
import os
os.chdir("/Users/FEAROFGOD/Desktop/CSC577/CSC577_project")
from surprise import KNNWithMeans, SVD
from surprise import Reader, Dataset
from surprise.model_selection import cross_validate, KFold
%load_ext autoreload
%autoreload
from hybrid import WeightedHybrid
from surprise.model_selection import PredefinedKFold
from surprise.accuracy import rmse
from surprise import Reader, Dataset
from surprise.model_selection import GridSearchCV
from prec_recall import precision_recall_at_k
from prec_recall import pr_eval
from surprise.model_selection import train_test_split
from prec_recall import pr_eval, precision_recall_at_k
```

1. Description of the data

1-1. Load the raw data; json file

The size of the data we have is 229,907

In [2]:
```
json_train = 'yelp_training_set_review.json'
load_train = []

f = open(json_train,"r")
for i in f:
    try:
        extract = json.loads(i)
        load_train.append(extract)
    except:
        continue

len(load_train)
```

Out[2]:
```
229907
```

1-2. A sample of the json file to see how it looks like

The data has 8 instances;'business_id', 'date','review_id','stars','text','type','user_id',and 'votes'

In [3]:
```
load_train[:1]
```

Out[3]:
```
[{'business_id': '9yKzy9PAPeiPPOUJEtnvkg',
  'date': '2011-01-26',
  'review_id': 'fWKvX83p0-ka4JS3dc6E5A',
  'stars': 5,
  'text': 'My wife took me here on my birthday for breakfast and it was excellent.  The weat
her was perfect which made sitting outside overlooking their grounds an absolute pleasure.
Our waitress was excellent and our food arrived quickly on the semi-busy Saturday morning.
It looked like the place fills up pretty quickly so the earlier you get here the better.\n\n
Do yourself a favor and get their Bloody Mary.  It was phenomenal and simply the best I\'ve
ever had.  I\'m pretty sure they only use ingredients from their garden and blend them fresh
when you order it.  It was amazing.\n\nWhile EVERYTHING on the menu looks excellent, I had t
he white truffle scrambled eggs vegetable skillet and it was tasty and delicious.  It came w
ith 2 pieces of their griddled bread with was amazing and it absolutely made the meal comple
te.  It was the best "toast" I\'ve ever had.\n\nAnyway, I can\'t wait to go back!',
  'type': 'review',
  'user_id': 'rLtl8ZkDX5vH5nAx9C3q5Q',
  'votes': {'cool': 2, 'funny': 0, 'useful': 5}}]
```

1-3. Transforming json file to dataframe

We extract'user_id','business_id' and 'stars' into a dataframe and rename them as 'user','item', and 'rate', respectively.

There are 45,981 many unique users and 11,537 many unique restaurants. The stars are in the range of 1 to 5.

In [4]:
```
# Converting json to a datafile and checking the size of dataset
train_df = pd.DataFrame()

train_df['user'] = list(map(lambda train_df: train_df['user_id'], load_train))
train_df['item'] = list(map(lambda train_df: train_df['business_id'], load_train))
train_df['rate'] = list(map(lambda train_df: train_df['stars'], load_train))
train_df.shape
```

Out[4]:
```
(229907, 3)
```

In [5]:
```
# The head function of the dataset
train_df.head()
```

Out[5]:

	user	item	rate
0	rLtl8ZkDX5vH5nAx9C3q5Q	9yKzy9PAPeiPPOUJEtnvkg	5
1	0a2KyEL0d3Yb1V6aivbIuQ	ZRJwVLyzEJq1VAihDhYiow	5
2	0hT2KtfLiobPvh6cDC8JQg	6oRAC4uyJCsJl1X0WZpVSA	4
3	uZetl9T0NcROGOyFfughhg	_1QQZuf4zZOyFCvXc0o6Vg	5
4	vYmM4KTsC8ZfQBg-j5MWkw	6ozycU1RpktNG2-1BroVtw	5

In [6]:
```
# User column
user = train_df.groupby('user').count()
user.shape
```

Out[6]:
```
(45981, 2)
```

In [7]:
```
# Item column
item = train_df.groupby('item').count()
item.shape
```

Out[7]:
```
(11537, 2)
```

In [8]:
```
# Rate column
rate = train_df.groupby('rate').count()
rate
```

Out[8]:

rate	user	item
1	17516	17516
2	20957	20957
3	35363	35363
4	79878	79878
5	76193	76193

1-4. Filtering the data.

Some users only have a small number of reviews according to the dataset. We believe the number of review less than 10 is not enough for determining the preference. Therefore, we filter out the user with number of reviews less than 10, and we left with 134,031 data

In [9]:
```
user['rate'].value_counts()
```

Out[9]:
```
1      22829
2       7432
3       3813
4       2443
5       1712
6       1183
7        936
8        668
9        572
10       427
11       368
12       339
13       280
14       217
16       185
15       178
18       160
17       136
20       116
19       115
22       111
21       105
24        82
23        78
27        60
26        59
28        57
25        54
30        47
32        45
       ...
176        1
191        1
506        1
205        1
263        1
179        1
149        1
183        1
120        1
152        1
248        1
294        1
230        1
340        1
170        1
134        1
376        1
244        1
138        1
473        1
172        1
236        1
165        1
300        1
588        1
371        1
245        1
250        1
196        1
297        1
Name: rate, Length: 208, dtype: int64
```

In [10]:
```
filtered = train_df[train_df.groupby('user').rate.transform(len)>10]
filtered.shape
```

Out[10]:
```
(134031, 3)
```

1-5. Save it as a csv file

In [11]:
```
#filtered.to_csv('yelp.csv',index = False)
```

In [12]:
```
df = pd.read_csv("yelp.csv")
print(df.shape)
```

```
(134031, 3)
```

In [13]:
```
df.isnull().sum()
```

Out[13]:
```
user    0
item    0
rate    0
dtype: int64
```

2. Algorithms / Models

We choose three algorithms to compare: KnnWithMeans(CF item-based), SVD, and Hybrid of the two. We will show how each algoritm performs by comparing each other.

The reason we choose **KNN** is that people tend to think in similar fashion for the analogous preferences. In the yelp dataset, we will focus on item-based for producing predictions to users. So that we could approach the set of reastaurants the target user has rated. In addition, we could solved the problem of sparsness.

We choose **SVD** because this model keeps only the most significant singular vectors to project the data to a lower dimensinal space. Since we have the huge dataset, using the SVD could help us understanding dataset more clearly.

Lastly, we want to evaluate the their **Hybrid** model if the combined model compensates their defects or augments their advantages.

Finally, We choose to evaluate the models using RMSE with 5-fold cross validation to measure the accuracy.

2-1. Finding hyper parameters for the algorithms

In [14]:
```
reader = Reader(rating_scale = (1,5))
data = Dataset.load_from_df(df, reader = reader)
cv = KFold(n_splits = 5, random_state = 1,shuffle=False)
```

In [15]:
```
from surprise import dataset
```

In [16]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.25)
print("train size", train.shape)
print("test size", test.shape)
```

```
train size (100523, 3)
test size (33508, 3)
```

In [17]:
```
#train.to_csv("yelp_train.csv", index = False)
#test.to_csv("yelp_test.csv", index = False)
```

In [18]:
```
trainset = Dataset.load_from_df(train, reader = reader)
testset = Dataset.load_from_df(test, reader = reader)
```

2-1-1. Grid Search for KNNWithMeans

In [19]:
```
# param_grid = {'k': list(range(100,220,20)),
#               'sim_options': {'name': ['cosine','pearson'],
#                               'user_based': [False]}}
# gs_knn = GridSearchCV(KNNWithMeans, param_grid, measures = ['RMSE'], cv = cv)
# gs_knn.fit(data)
# print("The best parameters",gs_knn.best_params)
```

The best parameters {'rmse': {'k': 200, 'sim_options': {'name': 'cosine', 'user_based': False}}}

2-1-2. Grid Search for SVD

In [20]:
```
# param_grid = {'n_factors':list(range(100,500,100)),
#               'n_epochs': list(range(10,50,10)),
#               'lr_all': [0.003,0.004,0.005,0.006,0.007],
#               'reg_all':[0.03,0.04,0.05,0.06,0.07]}
# gs_svd = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=cv)
# gs_svd.fit(data)
# print("The best parameters",gs_svd.best_params)
```

The best parameters {'rmse': {'n_factors': 100, 'n_epochs': 40, 'lr_all': 0.003, 'reg_all': 0.07}}

2-2. Evaluating the models

In [21]:
```
algo1 = KNNWithMeans(k=200, sim_options={'name': 'cosine', 'user_based': False})
algo2 = SVD(n_factors=100, n_epochs=40, lr_all=0.003, reg_all=0.07)
hybrid = WeightedHybrid([algo1, algo2])
```

In [22]:
```
output1 = cross_validate(algo1, data, measures=['RMSE'], cv=cv, verbose=True)
```

```
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0722  1.0665  1.0700  1.0699  1.0827  1.0723  0.0055
Fit time          115.42  117.40  116.01  115.42  22.08   97.27   37.60
Test time         8.53    8.45    8.17    8.00    3.06    7.24    2.10
```

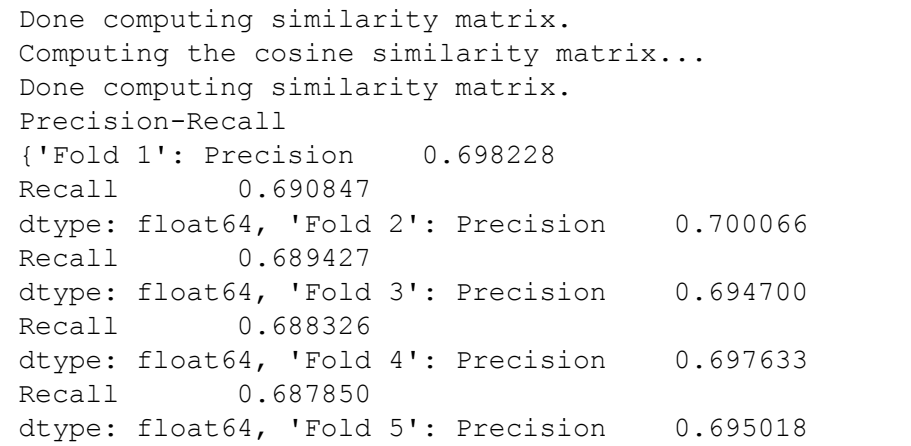
In [23]:
```
output2 = cross_validate(algo2, data, measures=['RMSE'], cv=cv, verbose=True)
```

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0235  1.0144  1.0187  1.0206  1.0272  1.0209  0.0043
Fit time          42.91   43.06   43.14   43.14   20.21   38.49   9.14
Test time         0.76    0.71    0.58    0.42    0.26    0.55    0.18
```

In [24]:
```
output3 = cross_validate(hybrid, data, measures=['RMSE'], cv=cv, verbose=True)
```

```
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
The weight before normalizing [-0.67651503  2.32752663]
Learned weights [0 1]
The weight before normalizing [-0.68513421  2.33893102]
Learned weights [0 1]
The weight before normalizing [-0.67696211  2.32592645]
Learned weights [0 1]
The weight before normalizing [-0.6755239   2.32403569]
Learned weights [0 1]
Computing the cosine similarity matrix...
Done computing similarity matrix.
The weight before normalizing [-0.67657592  2.32946815]
Learned weights [0 1]
Evaluating RMSE of algorithm WeightedHybrid on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.1228  1.1161  1.1167  1.1163  1.1205  1.1185  0.0027
Fit time          198.40  198.03  197.62  195.84  69.19   171.82  51.32
Test time         10.45   10.37   10.12   10.19   3.16    8.86    2.85
```

In [25]:
```
rmse = [output1['test_rmse'], output2['test_rmse'],output3['test_rmse']]
plt_range = [1.00, 1.05, 1.10, 1.15]
plt.boxplot(rmse, labels = ['KNNWithMeans','SVD','Hybrid'], vert = False)
plt.xticks(plt_range)
plt.xlabel("RMSE")
plt.title("The Performances for Each Model")
plt.show()
```



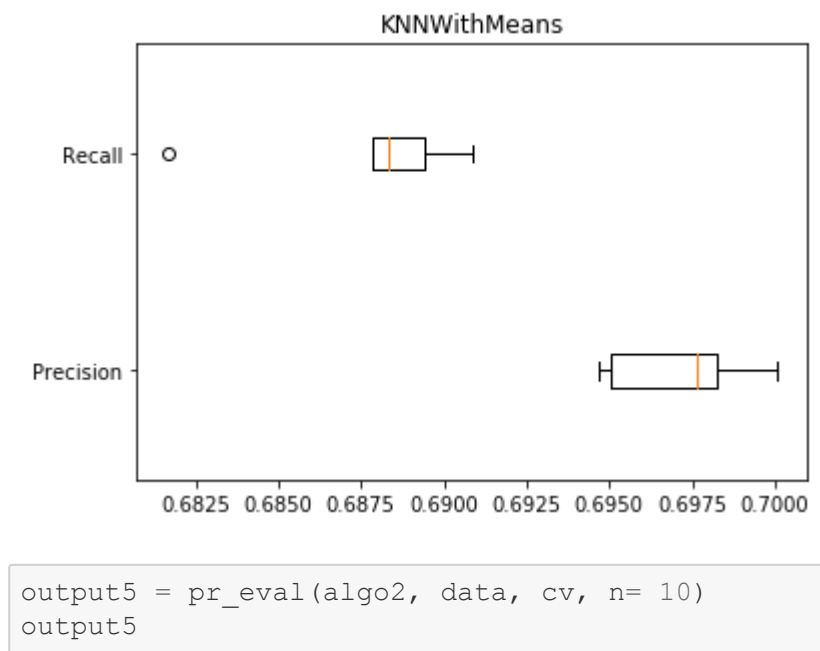
In [26]:
```
output4 = pr_eval(algo1, data, cv, n= 10)
output4
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Precision-Recall
{'Fold 1': Precision    0.698228
Recall       0.690847
dtype: float64, 'Fold 2': Precision    0.700066
Recall       0.689427
dtype: float64, 'Fold 3': Precision    0.694700
Recall       0.688326
dtype: float64, 'Fold 4': Precision    0.697633
Recall       0.687850
dtype: float64, 'Fold 5': Precision    0.695018
Recall       0.681646
dtype: float64}
```

Out[26]:

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Precision	0.698228	0.700066	0.694700	0.697633	0.695018
Recall	0.690847	0.689427	0.688326	0.687850	0.681646

In [27]:
```
pre_rec = [output4.T['Precision'], output4.T['Recall']]
plt.boxplot(pre_rec, labels=["Precision","Recall"], vert=False, )
plt.title('KNNWithMeans')
plt.show()
```



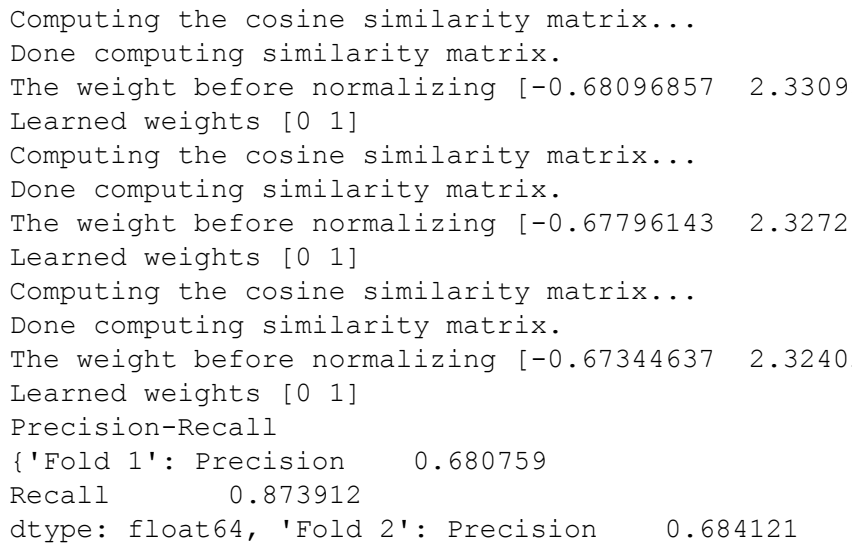
In [28]:
```
output5 = pr_eval(algo2, data, cv, n= 10)
output5
```

```
Precision-Recall
{'Fold 1': Precision    0.705993
Recall       0.728457
dtype: float64, 'Fold 2': Precision    0.700479
Recall       0.719705
dtype: float64, 'Fold 3': Precision    0.694904
Recall       0.718322
dtype: float64, 'Fold 4': Precision    0.698781
Recall       0.719234
dtype: float64, 'Fold 5': Precision    0.691345
Recall       0.714017
dtype: float64}
```

Out[28]:

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Precision	0.705993	0.700479	0.694904	0.698781	0.691345
Recall	0.728457	0.719705	0.718322	0.719234	0.714017

In [29]:
```
pre_rec = [output5.T['Precision'], output5.T['Recall']]
plt.boxplot(pre_rec, labels=["Precision","Recall"], vert=False, )
plt.title('SVD')
plt.show()
```



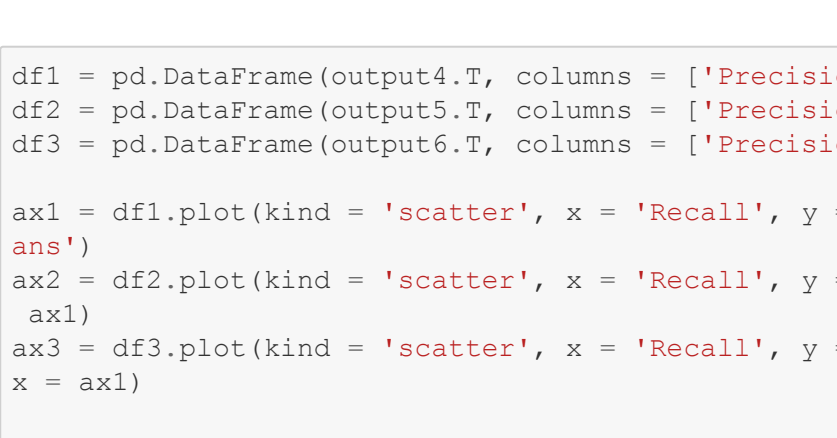
In [30]:
```
output6 = pr_eval(hybrid, data, cv, n= 10)
output6
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
The weight before normalizing [-0.67406346  2.32464422]
Learned weights [0 1]
Computing the cosine similarity matrix...
Done computing similarity matrix.
The weight before normalizing [-0.69241371  2.34791482]
Learned weights [0 1]
Computing the cosine similarity matrix...
Done computing similarity matrix.
The weight before normalizing [-0.68096857  2.33095723]
Learned weights [0 1]
Computing the cosine similarity matrix...
Done computing similarity matrix.
The weight before normalizing [-0.67796143  2.3272786 ]
Learned weights [0 1]
Computing the cosine similarity matrix...
Done computing similarity matrix.
The weight before normalizing [-0.67344637  2.32402487]
Learned weights [0 1]
Precision-Recall
{'Fold 1': Precision    0.680759
Recall       0.873912
dtype: float64, 'Fold 2': Precision    0.684121
Recall       0.876331
dtype: float64, 'Fold 3': Precision    0.678841
Recall       0.876059
dtype: float64, 'Fold 4': Precision    0.681519
Recall       0.875071
dtype: float64, 'Fold 5': Precision    0.677219
Recall       0.874913
dtype: float64}
```

Out[30]:

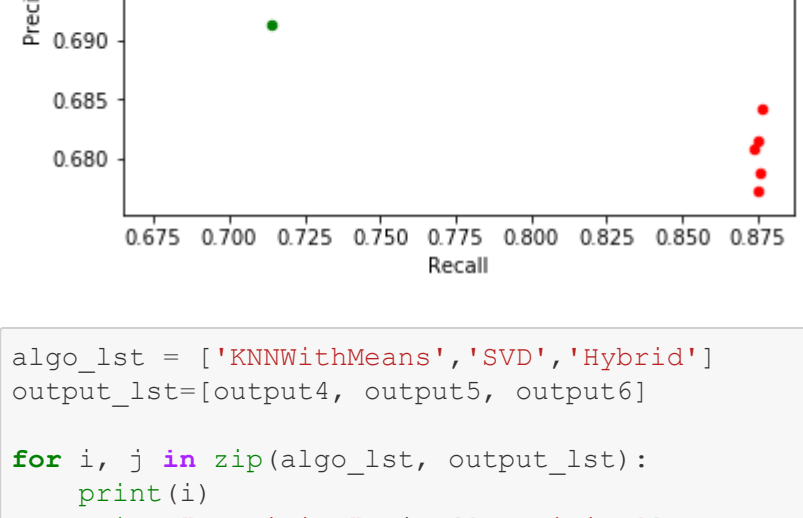
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Precision	0.680759	0.684121	0.678841	0.681519	0.677219
Recall	0.873912	0.876331	0.876059	0.875071	0.874913

In [31]:
```
pre_rec = [output6.T['Precision'], output6.T['Recall']]
plt.boxplot(pre_rec, labels=["Precision","Recall"], vert=False)
plt.title('Hybrid')
plt.show()
```



In [32]:
```
df1 = pd.DataFrame(output4.T, columns = ['Precision', 'Recall'])
df2 = pd.DataFrame(output5.T, columns = ['Precision', 'Recall'])
df3 = pd.DataFrame(output6.T, columns = ['Precision', 'Recall'])

ax1 = df1.plot(kind = 'scatter', x = 'Recall', y = 'Precision', color = 'b', label = 'KNNWithMe
ans')
ax2 = df2.plot(kind = 'scatter', x = 'Recall', y = 'Precision', color = 'g', label = 'SVD',ax =
 ax1)
ax3 = df3.plot(kind = 'scatter', x = 'Recall', y = 'Precision', color = 'r', label = 'Hybrid',a
x = ax1)

ax1.legend()
ax2.legend()
ax3.legend()
```

Out[32]:
```
<matplotlib.legend.Legend at 0x1a3fb33208>
```



In [33]:
```
algo_lst = ['KNNWithMeans','SVD','Hybrid']
output_lst=[output4, output5, output6]

for i, j in zip(algo_lst, output_lst):
    print(i)
    print("Precision", j.T['Precision'].mean())
    print("Recall", j.T['Recall'].mean())
    print()
```

```
KNNWithMeans
Precision 0.6971291118211325
Recall 0.6876192026540442

SVD
Precision 0.6983005030284212
Recall 0.7199470024170995

Hybrid
Precision 0.680491903452159
Recall 0.8752570030712752
```