1 Import libraries

In [ ]:
```
! pip install -q surprise
```

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import ast
from scipy import stats
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer, CountVector
izer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from surprise import Reader, Dataset, SVD, evaluate

import warnings; warnings.simplefilter('ignore')
```

In [ ]:
```
! pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

2 Import Product Information

In [ ]:
```
product_file_import = drive.CreateFile({'id': '1qZD9r6Luv2pOh4jjfOPHIc4b
1KeoOnn0'})
product_file_import.GetContentFile('product.csv')
product = pd.read_csv('product.csv')

product.head()
```

Out[ ]:

	Unnamed: 0	asin	avg.rating	avg.helpful.ratio	also_bought	also_viewed	brand
0	1	7806397051	2.50	NaN	['B00KR26VFE', 'B00E7LQHZ0', 'B00BMW24TU', 'B0...	['B008GOR6O0', 'B00EOFEKF8', 'B00IIFVJZ4', 'B0...	COKA
1	2	9759091062	3.09	NaN	['B0054GLD1U', 'B003BRZCUC', 'B0054GBXOW', 'B0...	['B0054GBXOW', 'B0054GLD1U', 'B006VDOPPQ', 'B0...	Xtreme Brite
2	3	9788072216	5.00	NaN	['B006C5OHSI', 'B006P14842', 'B0072CSVB4', 'B0...	['B0072CSVB4', 'B005YWBOHW', 'B00CGOUL2A', 'B0...	Prada
3	4	9790790961	4.60	NaN	['B007P7OPQQ', 'B0017JT658', 'B0084HM1DA', 'B0...	['B005M2AQRI', 'B000VOHKK8', 'B0017JT658', 'B0...	Versace
4	5	9790794231	4.50	NaN	['B0019M21OQ', 'B000E7YM8K', 'B0006V31FY', 'B0...	['B000E7YM8K', 'B0019M21OQ', 'B0006V31FY', 'B0...	

In [ ]:
```
product.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11346 entries, 0 to 11345
Data columns (total 12 columns):
Unnamed: 0           11346 non-null int64
asin                 11346 non-null object
avg.rating           11346 non-null float64
avg.helpful.ratio    792 non-null float64
also_bought          11346 non-null object
also_viewed          11346 non-null object
brand                11330 non-null object
categories           11346 non-null object
description          10664 non-null object
price                10941 non-null float64
salesRank            11346 non-null object
title                11346 non-null object
dtypes: float64(3), int64(1), object(8)
memory usage: 1.0+ MB
```

In [ ]:
```
product.describe(include = 'O')
```

Out[ ]:

	asin	also_bought	also_viewed	brand	categories	description	salesRank	title
count	11346	11346	11346	11330	11346	10664	11346	11346
unique	11346	11128	11231	2012	251	9790	10157	11314
top	B0037MQIT0				[['Beauty', 'Makeup', 'Nails', 'Nail Polish']]			
freq	1	209	103	1804	671	184	187	7

```
In [ ]: product['also_bought'] = product['also_bought'].fillna('')
        product['also_viewed'] = product['also_viewed'].fillna('')
        product['brand'] = product['brand'].fillna('')
        product['description'] = product['description'].fillna('')
        product['title'] = product['title'].fillna('')
```

```
In [ ]: product.shape
```

```
Out[ ]: (11346, 12)
```

3 Content based recommendation system : Using Product Description

- Let us first try to build a recommender using descriptions.

3.1 Generating TF-IDF Matrix

```
In [ ]: tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_w
        ords='english')
        tfidf_matrix = tf.fit_transform(product['description'])
```

```
In [ ]: tfidf_matrix
```

```
Out[ ]: <11346x262450 sparse matrix of type '<class 'numpy.float64'>'
                with 874430 stored elements in Compressed Sparse Row format>
```

3.2 Generating Cosine for Item Recommendation

- Since we have used the TF-IDF Vectorizer, calculating the Dot Product will directly give us the Cosine Similarity Score.
- Therefore, we will use sklearn's linear_kernel instead of cosine_similarities since it is much faster.

```
In [ ]: cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
In [ ]: cosine_sim
```

```
Out[ ]: array([[1.        , 0.0044185 , 0.        , ..., 0.        , 0.0113141
        3,
                 0.01138293],
               [0.0044185 , 1.        , 0.        , ..., 0.        , 0.0075171
        7,
                 0.        ],
               [0.        , 0.        , 1.        , ..., 0.        , 0.
        ,
                 0.00551829],
               ...,
               [0.        , 0.        , 0.        , ..., 0.        , 0.
        ,
                 0.        ],
               [0.01131413, 0.00751717, 0.        , ..., 0.        , 1.
        ,
                 0.00793195],
               [0.01138293, 0.        , 0.00551829, ..., 0.        , 0.0079319
        5,
                 1.        ]])
```

- We now have a pairwise cosine similarity matrix for all the beauty product in our dataset.

3.3 Constructing Function for Recommendation

- Generate 10(default) recommendations for each imput according to the rank of cosine similarity score

```
In [ ]: product = product.reset_index()
        titles = product['asin']
        indices = pd.Series(product.index, index=product['asin'])
```

```
In [ ]: def get_recommendations_asin(title, out_num = 20):
            idx = indices[title]
            sim_scores = list(enumerate(cosine_sim[idx]))
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
            sim_scores = sim_scores[1:31]
            movie_indices = [i[0] for i in sim_scores]
            return titles.iloc[movie_indices].head(out_num)
```

3.3.1Testing the output of the recommendation function

```
In [ ]: list(get_recommendations_asin('7806397051'))

Out[ ]: ['B008GOR6O0',
         'B00D5TB1LK',
         'B0068Y6CA4',
         'B0006ZHK7A',
         'B008XWX4A0',
         'B0073SBK4M',
         'B0019ANSAO',
         'B0009I4MCU',
         'B0047ZVSSM',
         'B004B4JSTA',
         'B004LXKY4E',
         'B000NUMRXK',
         'B0000530ED',
         'B0036QQWAC',
         'B000EVIUZC',
         'B009C7IRZW',
         'B0037BOLVS',
         'B001E3SG2Q',
         'B004DK46XK',
         'B0009I4MG6']
```

```
In [ ]: type(list(get_recommendations_asin('7806397051').head(10)))

Out[ ]: list
```

4 User Information Preparation

4.1 User Data Import & Cleaning

```
In [ ]: BB_file_import = drive.CreateFile({'id': '153U3ooeV1FNCiGQ0bzQREYDC6gkS6
        O5D'})
        BB_file_import.GetContentFile('Beauty_5.json')
```

In [ ]:
```
import pandas as pd
import csv
import json
import re

# data in json file is not in valid json format
# process each line before paser it
def process_line(l):
    # replace "" with "
    l_n = l.replace('""', '"')
    # extract valid json part and the extra field
    m = re.match(r'^"(.*)"\t"(\d{4})"}"$', l_n)
    if not m:
        assert(False)
    json_str = m[1]
    year = m[2]
    # parse json part
    json_v = json.loads(json_str)
    return (json_v, year)

# read json file
with open('Beauty_5.json') as f:
    lines = [line.rstrip('\n') for line in f]

# each line is a json item
json_rows = []
# there is an extra field on each line
year_rows = []
# parse json line by line
for l in lines:
    json_v, year = process_line(l)
    json_rows.append(json_v)
    year_rows.append(year)

# get column name for csv file
col_name = [k for k, v in json_rows[0].items()]

# write csv file
with open("Beauty_5.csv", "w") as fw:
    cvs_writer = csv.writer(fw)
    cvs_writer.writerow([*col_name, 'year'])
    for jr, year in zip(json_rows, year_rows):
        row = [jr[k] if k in jr else '' for k in col_name]
        cvs_writer.writerow([*row, year])

# load to pandas and verify
beauty_data = pd.read_csv('Beauty_5.csv')
beauty_data.head(10)
```

Out[ ]:

	reviewerID	asin	reviewerName	helpful	reviewText	overall	
0	A1YJEY40YUW4SE	7806397051	Andrea	[3, 4]	Very oily and creamy. Not at all what I expect...	1.0	Don't w
1	A60XNB876KYML	7806397051	Jessica H.	[1, 1]	This palette was a decent price and I was look...	3.0	
2	A3G6XNM240RMWA	7806397051	Karen	[0, 1]	The texture of this concealer pallet is fantas...	4.0	
3	A1PQFP6SAJ6D80	7806397051	Norah	[2, 2]	I really can't tell what exactly this thing is...	2.0	Do not
4	A38FVHZTNQ271F	7806397051	Nova Amor	[0, 0]	It was a little smaller than I expected, but t...	3.0	
5	A3BTN14HIZET6Z	7806397051	S. M. Randall "WildHorseWoman"	[1, 2]	I was very happy to get this palette, now I wi...	5.0	'
6	A1Z59RFKN0M5QL	7806397051	tasha "luvely12b"	[1, 3]	PLEASE DONT DO IT! this just rachett the palet...	1.0	
7	AWUO9P6PL1SY8	7806397051	TreMagnifique	[0, 1]	Chalky,Not Pigmented,Wears off easily,Not a Co...	2.0	Chalky Wears
8	A3LMILRM9OC3SA	9759091062	NaN	[0, 0]	Did nothing for me. Stings when I put it on. I...	2.0	I Brighteni
9	A30IP88QK3YUIO	9759091062	Amina Bint Ibraheem	[0, 0]	I bought this product to get rid of the dark s...	3.0	

In [ ]:
```
beauty_data.describe(include='O')
```

Out[ ]:

	reviewerID	asin	reviewerName	helpful	reviewText	summary	reviewTim
count	143560	143560	142418	143560	143549	143559	14356
unique	20607	11891	18283	611	143502	100064	56
top	ALNFHVS3SC4FV	B004OHQR1Q	Amazon Customer	[0, 0]	great product	Love it	04 5, 201
freq	161	376	1697	91986	4	757	63

In [ ]:
```
beauty_data.describe()
```

Out[ ]:

	overall	unixReviewTime	year
count	143560.000000	1.435600e+05	143560.000000
mean	4.202243	1.382359e+09	2013.406165
std	1.156612	1.454541e+07	0.491118
min	1.000000	1.356998e+09	2013.000000
25%	4.000000	1.368749e+09	2013.000000
50%	5.000000	1.383610e+09	2013.000000
75%	5.000000	1.395274e+09	2014.000000
max	5.000000	1.406074e+09	2014.000000

4.2 Data Subset

- We only want data from 2013 to 2014

In [ ]:
```
# We only need data from 2013 to 2014
beauty = beauty_data.loc[(beauty_data['year']==2013) | (beauty_data['yea
r']==2014),['reviewerID', 'asin', 'overall',  'year']]
```

In [ ]:
```
beauty.head()
```

Out[ ]:

	reviewerID	asin	overall	year
0	A1YJEY40YUW4SE	7806397051	1.0	2014
1	A60XNB876KYML	7806397051	3.0	2014
2	A3G6XNM240RMWA	7806397051	4.0	2013
3	A1PQFP6SAJ6D80	7806397051	2.0	2013
4	A38FVHZTNQ271F	7806397051	3.0	2013

In [ ]:
```
beauty['year'].value_counts()
```

Out[ ]:
```
2013    85251
2014    58309
Name: year, dtype: int64
```

In [ ]:
```
beauty['overall'].value_counts()
```

Out[ ]:
```
5.0    83393
4.0    28568
3.0    16162
2.0     8114
1.0     7323
Name: overall, dtype: int64
```

In [ ]:
```
user = combine[['asin','reviewerID','overall','year']]
user.head()
```

Out[ ]:

	asin	reviewerID	overall	year
0	7806397051	A1YJEY40YUW4SE	1.0	2014
1	7806397051	A60XNB876KYML	3.0	2014
2	7806397051	A3G6XNM240RMWA	4.0	2013
3	7806397051	A1PQFP6SAJ6D80	2.0	2013
4	7806397051	A38FVHZTNQ271F	3.0	2013

In [ ]:
```
import pandas
user.to_csv('user.csv')
```

4.3 Train-test split

- Customers in 2013 are set as training set--using 'user' in the following code
- Customers in 2014 are set as testing set--using 'user_testing' in the following code
- All the customers exist both in training and testing set, but there is no such requirement for the products they bought.

5 Model Evaluation

5.1 Loading training dataset as 'user'

In [ ]:
```
product_file_import_training = drive.CreateFile({'id': '1ILcnNDOmMj50_gE
MJv24aqlmUy8bvo6j'})
product_file_import_training.GetContentFile('training.csv')
training = pd.read_csv('training.csv')

training.head()
```

Out[ ]:

	Unnamed: 0	asin	reviewerID	overall	year
0	8	B0020HEBX8	A00473363TJ8YSZ3YAGG9	4	2013
1	9	B0019LVFI0	A00473363TJ8YSZ3YAGG9	3	2013
2	10	B001L2BEWE	A00473363TJ8YSZ3YAGG9	4	2013
3	11	B006R5GXCG	A00473363TJ8YSZ3YAGG9	4	2013
4	31	B005J5TIYK	A01198201H0E3GHV2Z17I	5	2013

In [ ]:
```
user = training
user.shape
```

Out[ ]:
```
(44309, 5)
```

In [ ]:
```
reviewerID_length = len(list(set(user['reviewerID'])))
unique_reviewerID = list(set(user['reviewerID']))
```

In [ ]:
```
reviewerID_length
```

Out[ ]:
```
9413
```

In [ ]:
```
user = user.sort_values(by=['reviewerID'])
```

In [ ]:
```
user.head(30)
```

Out[ ]:

	Unnamed: 0	asin	reviewerID	overall	year
0	8	B0020HEBX8	A00473363TJ8YSZ3YAGG9	4	2013
1	9	B0019LVFI0	A00473363TJ8YSZ3YAGG9	3	2013
2	10	B001L2BEWE	A00473363TJ8YSZ3YAGG9	4	2013
3	11	B006R5GXCG	A00473363TJ8YSZ3YAGG9	4	2013
4	31	B005J5TIYK	A01198201H0E3GHV2Z17I	5	2013
5	33	B008U1Q4DI	A01198201H0E3GHV2Z17I	5	2013
6	34	B002MZ8BK2	A01198201H0E3GHV2Z17I	3	2013
7	35	B0057US3O8	A01198201H0E3GHV2Z17I	3	2013
11	42	B005Z41P28	A02155413BVL8D0G7X6DN	5	2013
10	41	B00117CH5M	A02155413BVL8D0G7X6DN	3	2013
8	36	B003FO70Z6	A02155413BVL8D0G7X6DN	5	2013
9	38	B0055MYJ0U	A02155413BVL8D0G7X6DN	5	2013
30	80	B002QFGKUQ	A03364251DGXSGA9PSR99	3	2013
27	77	B0098TKSLU	A03364251DGXSGA9PSR99	4	2013
28	78	B004Z40048	A03364251DGXSGA9PSR99	1	2013
29	79	B004Y3H7MS	A03364251DGXSGA9PSR99	5	2013
31	81	B001G2GCO4	A03364251DGXSGA9PSR99	4	2013
37	87	B001G2L1O0	A03364251DGXSGA9PSR99	4	2013
33	83	B005OZJYUS	A03364251DGXSGA9PSR99	1	2013
34	84	B008K1YYM6	A03364251DGXSGA9PSR99	4	2013
35	85	B005V1A05S	A03364251DGXSGA9PSR99	4	2013
36	86	B0016L3QDK	A03364251DGXSGA9PSR99	5	2013
26	76	B004L8J15C	A03364251DGXSGA9PSR99	4	2013
32	82	B003AJJTXM	A03364251DGXSGA9PSR99	5	2013
25	75	B004ZI6AQQ	A03364251DGXSGA9PSR99	3	2013
19	69	B004QLOFTG	A03364251DGXSGA9PSR99	3	2013
23	73	B003ZS4WJY	A03364251DGXSGA9PSR99	5	2013
22	72	B007M6EALK	A03364251DGXSGA9PSR99	3	2013
21	71	B0092DUN6M	A03364251DGXSGA9PSR99	3	2013
20	70	B003ZS6OJK	A03364251DGXSGA9PSR99	4	2013

5.2 Generate lists: for unique User_ID , asin, rating, and ratingAvg

In [ ]:
```
## Distribute space for lists storing asin list and rating lists for all
 customers
df_asin = [] # Storing asin list
df_rating = [] # Storing rating list
for i in range(reviewerID_length):
  list_in_list1 = []
  list_in_list2 = []
  df_asin.append(list_in_list1)
  df_rating.append(list_in_list2)
```

In [ ]:
```
## For a specific customer, store the items he/her bought and the corres
ponding ratings in two seperates lists
pos_in_list = 0
for i in range(user.shape[0]):
  if i == 0:
    df_asin[pos_in_list].append(user['asin'].iloc[0])
    df_rating[pos_in_list].append(user['overall'].iloc[0])
  if i > 0:
    if user['reviewerID'].iloc[i] == user['reviewerID'].iloc[i-1]:
      df_asin[pos_in_list].append(user['asin'].iloc[i])
      df_rating[pos_in_list].append(user['overall'].iloc[i])
    if user['reviewerID'].iloc[i] != user['reviewerID'].iloc[i-1]:
      pos_in_list = pos_in_list + 1
      df_asin[pos_in_list].append(user['asin'].iloc[i])
      df_rating[pos_in_list].append(user['overall'].iloc[i])
```

In [ ]:
```
## Store the order of the customers in the list
df_reviewerID = []
for i in range(user.shape[0]):
  if i == 0:
    df_reviewerID.append(user['reviewerID'].iloc[0])
  else:
    if user['reviewerID'].iloc[i] != user['reviewerID'].iloc[i-1]:
      df_reviewerID.append(user['reviewerID'].iloc[i])
```

In [ ]:
```
## Calcualte the average rating for each customer
df_ratingAvg = []
for i in range(len(df_rating)):
  df_ratingAvg.append(np.mean(df_rating[i]))
```

5.3 Generate decommendation set

In [ ]:
```
df_recommend = []
for i in range(len(df_reviewerID)):

  if i%200 ==0:
    print('step: ',i)

  recommend_list = []
  # Only remain ratings greater than average
  list_boolean = (df_rating[i] >= np.mean(df_rating[i]))
  list_boolean
  aboveAvg = [] # storing asin for recommendation
  for k in range(len(list_boolean)):
    if list_boolean[k] == True:
      aboveAvg.append(df_asin[i][k])

  # recommendation for each asin
  for j in range(len(aboveAvg)):
    recommend_list = recommend_list + (list(get_recommendations_asin(abo
veAvg[j], 20)))
  df_recommend.append(recommend_list)
```

- Storing the recommendation list for training set

In [ ]:
```
import pandas as pd
trainng_remommendation_dataset = pd.DataFrame(
    {'df_reviewerID': df_reviewerID,
     'df_asin': df_asin,
     'df_rating': df_rating,
     'df_ratingAvg': df_ratingAvg,
     'df_recommend': df_recommend
    })
```

```
In [ ]: trainng_remommendation_dataset.head()
```

Out[ ]:

	df_asin	df_rating	df_ratingAvg	df_recommend	df_reviewerID
0	[B0020HEBX8, B0019LVFI0, B001L2BEWE, B006R5GXCG]	[4, 3, 4, 4]	3.750000	[B003GDBEBM, B0063KG6ZY, B00BMHBKDG, B005IC4S6...	A00473363TJ8YSZ3YAGG9
1	[B005J5TIYK, B008U1Q4DI, B002MZ8BK2, B0057US3O8]	[5, 5, 3, 3]	4.000000	[B007TXSLHU, B007TY3IAE, B006EOCS1U, B006EPIJT...	A01198201H0E3GHV2Z17I
2	[B005Z41P28, B00117CH5M, B003FO70Z6, B0055MYJ0U]	[5, 3, 5, 5]	4.500000	[B005Z49PQG, B005Z446JC, B003Z4OD24, B003Z4SGJ...	A02155413BVL8D0G7X6DN
3	[B002QFGKUQ, B0098TKSLU, B004Z40048, B004Y3H7M...	[3, 4, 1, 5, 4, 4, 1, 4, 4, 5, 4, 5, 3, 3, 5, ...	3.923077	[B008UQAW54, B00BBCXHP6, B00KWE08Q0, B0092KGYE...	A03364251DGXSGA9PSR99
4	[B00ARBCWYY]	[5]	5.000000	[B007MKVTSS, B00EPZCE78, B008H7RWH2, B00838FLX...	A0388397363MZHRU6ALSX

```
In [ ]: trainng_remommendation_dataset.to_csv('trainng_remommendation_dataset.cs
        v')
```

5.4 Loading testing dataset as 'user_testing'

```
In [ ]: product_file_import_testing = drive.CreateFile({'id': '1CwoRWxFmdt9HJqCq
        wyxJsFLl-DBIZWoA'})
        product_file_import_testing.GetContentFile('testing.csv')
        user_testing = pd.read_csv('testing.csv')

        user_testing.head()


        reviewerID_length_testing = len(list(set(user_testing['reviewerID'])))
        unique_reviewerID_testing = list(set(user_testing['reviewerID']))
```

5.5 Adjusting the testing data structure for unique User_ID, asin

In [ ]:
```
df_asin_testing = []
df_rating_testing = []
for i in range(reviewerID_length_testing):
  list_in_list3 = []
  list_in_list4 = []
  df_asin_testing.append(list_in_list3)
  df_rating_testing.append(list_in_list4)


pos_in_list = 0
for i in range(user_testing.shape[0]):

  if i == 0:
    df_asin_testing[pos_in_list].append(user_testing['asin'].iloc[0])
    df_rating_testing[pos_in_list].append(user_testing['overall'].iloc[0
])
  if i > 0:
    if user_testing['reviewerID'].iloc[i] == user_testing['reviewerID'].
iloc[i-1]:
      df_asin_testing[pos_in_list].append(user_testing['asin'].iloc[i])
      df_rating_testing[pos_in_list].append(user_testing['overall'].iloc
[i])
    if user_testing['reviewerID'].iloc[i] != user_testing['reviewerID'].
iloc[i-1]:
      pos_in_list = pos_in_list + 1
      df_asin_testing[pos_in_list].append(user_testing['asin'].iloc[i])
      df_rating_testing[pos_in_list].append(user_testing['overall'].iloc
[i])


df_reviewerID_testing = []
for i in range(user_testing.shape[0]):
  if i == 0:
    df_reviewerID_testing.append(user_testing['reviewerID'].iloc[0])
  else:
    if user_testing['reviewerID'].iloc[i] != user_testing['reviewerID'].
iloc[i-1]:
      df_reviewerID_testing.append(user_testing['reviewerID'].iloc[i])
```

In [ ]:
```
df_reviewerID_testing == df_reviewerID
```

Out[ ]:
```
True
```

- The order of reviewer ID are consistent with training and testing dataset

5.6 Precision Testing

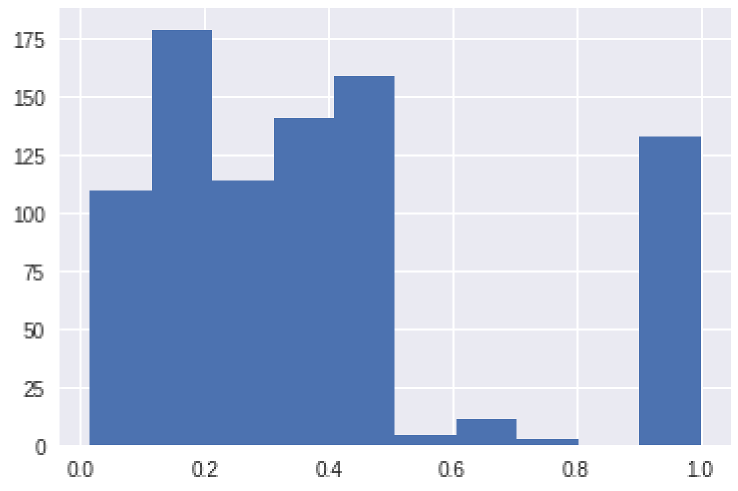
5.6.1 Customer-oriented

In [ ]:
```
## Function used for precision testing
def intersection(lst1, lst2):
    lst3 = [value for value in lst1 if value in lst2]
    return lst3
```

In [ ]:
```
product_match = []
visualize_match = []
for i in range(len(df_recommend)):
  unique = len(set(intersection( df_asin_testing[i],df_recommend[i] )))
  product_match.append(unique*1.0/len(set(df_asin_testing[i])))
  if unique != 0:
    visualize_match.append(unique*1.0/len(set(df_asin_testing[i])))
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.hist(visualize_match)
```

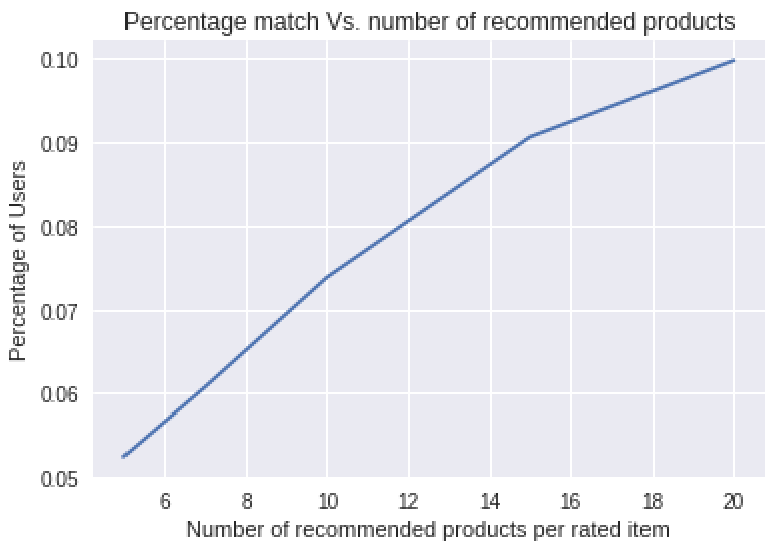
Out[ ]:
```
(array([110., 179., 114., 141., 159.,   4.,  11.,   2.,   0., 133.]),
 array([0.015625 , 0.1140625, 0.2125   , 0.3109375, 0.409375 , 0.507812
5,
        0.60625  , 0.7046875, 0.803125 , 0.9015625, 1.       ]),
 <a list of 10 Patch objects>)
```



In [ ]:
```
len(visualize_match)/9413
```

Out[ ]:
```
0.0906193562094975
```

In [ ]:
```
out = [5, 7, 10, 15, 20]
pre = [0.05248061191968554, 0.060767024328056946, 0.07383405927971953,
0.0906193562094975, 0.09975565707000957]
plt.plot(out, pre)
plt.xlabel('Number of recommended products per rated item')
plt.ylabel('Percentage of Users')
plt.title('Percentage match Vs. number of recommended products')
plt.show()
```



5.6.2 Item-oriented

In [ ]:
```
## items bought by buy users in 2014
rated = list(set(user_testing['asin']))
len(rated)
```

Out[ ]:
```
8630
```

```
In [ ]: ## items recommended & bought by users in 2014
        recommend_rated = []
        for i in range(len(df_recommend)):
          unique = list(set(intersection(df_asin_testing[i],df_recommend[i] )))
          if len(unique) != 0:
            recommend_rated = recommend_rated + unique

        recommend_rated = list(set(recommend_rated))
        len(recommend_rated)

Out[ ]: 980
```

```
In [ ]: ## item recommended to users in 2014
        recommend = []
        for i in range(len(df_recommend)):
          recommend = recommend + list(set(df_recommend[i]))

        recommend = list(set(recommend))
        len(recommend)

Out[ ]: 10295
```

```
In [ ]: precision = len(recommend_rated)/len(recommend)
        recall = len(recommend_rated)/len(rated)
        f1 = 2*recall*precision/(recall+precision)
```

```
In [ ]: precision

Out[ ]: 0.09519184069936863
```

```
In [ ]: recall

Out[ ]: 0.11355735805330243
```

```
In [ ]: f1

Out[ ]: 0.1035667107001321
```