CF Recommendation System

COMP9417

Aim: The aim of this project is to build a CF recommendation engine using the **Book-Crossing** dataset.

In [1]:

```
#import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from surprise import Reader, Dataset
from surprise import model_selection, accuracy
from surprise import NMF
from surprise import SVD
from surprise import SVDpp
from surprise import CoClustering
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNWithZScore
plt.style.use('seaborn-white') # Use seaborn-style plots
%matplotlib inline

#Read files.
import os
global directory; directory = '../BookRating'
def files(): return os.listdir(directory)

Users = pd.read_csv('BX-Users.csv', sep = ";", encoding='cp1252')
Ratings = pd.read_csv('BX-Book-Ratings.csv', sep = ";", encoding='cp1252')
Books = pd.read_csv('BX-Books.csv', sep=";", usecols=range(5), encoding ='iso-8859-1')


#Clean data with replacement of missing and invalid data
#Users
Users.columns = Users.columns.str.strip().str.lower().str.replace('-', '_')
Users.user_id = Users.user_id.astype(int)
#User ages
#invalid ages of users such as, 224 years old and 0 years old observed
std = Users.age.std(); mode = int(Users.age.mode().values.tolist()[0]); len_fillna = len(Us
#learn stasticis of data
df_copy = Users.copy()
df_normalize = df_copy.age.value_counts().rename_axis('age').reset_index(name = 'user_count
df_normalize = df_normalize.sort_values(by=['age'])
df_normalize['id'] = df_copy['user_id']
df_normalize = df_normalize.assign(precent_counts=lambda x: x.user_counts.cumsum() / x.user
#define where the 0.5% to 99.5% user count of age interval is
age_high = max(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) &\
                                (df_normalize['precent_counts'] <= 0.995)].age.unique().tol
age_low = min(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) & \
                               (df_normalize['precent_counts'] <= 0.995)].age.unique().toli
#fill invalid and missing ages with random normal distributed numbers
#shifting mean to mode(mean = mode in normaldistribution)
fillna_list = (np.random.normal(mode, std, 1000)).tolist()
Users.loc[(Users.age<5) | (Users.age>90), 'age'] = np.nan
Users.age = Users.age.fillna(pd.Series(fillna_list))
#User locations
#originally format: location : 'farnborough, hants, united kingdom'
#break into three columns to better check validity
tmp_list = Users.location.str.split(',', 2, expand=True)
tmp_list.describe(include=[object])
#fill nan with 'other'
tmp_list.fillna('other', inplace=True)
Users['city'] = tmp_list[0];Users['state'] = tmp_list[1];Users['country'] = tmp_list[2]
Users.drop(columns=['location'], inplace=True)
```

```
#Books
Books.columns = Books.columns.str.strip().str.lower().str.replace('-', '_')
Books.year_of_publication = pd.to_numeric(Books.year_of_publication, errors='coerce')
#replace outscaled publication years
std = Books.year_of_publication.std()
mode = int(Books.year_of_publication.mode().values.tolist()[0])
len_fillna = len(Books.index)
#learn statistical information of book counts on year of publication
df_normalize={}
df_copy = Books.copy()
df_copy.year_of_publication.replace(0, np.nan, inplace=True)
df_normalize = df_copy.year_of_publication.value_counts().rename_axis('year').reset_index(n
df_normalize = df_normalize.sort_values(by=['year'])
df_normalize['id'] = df_copy['isbn']
df_normalize = df_normalize.assign(precent_counts=lambda x: x.book_counts.cumsum() / x.book
year_high = max(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) &\
                                 (df_normalize['precent_counts'] <= 0.995)].year.unique().t
year_low = min(df_normalize.loc[(df_normalize['precent_counts'] >= 0.005) &\
                                (df_normalize['precent_counts'] <= 0.995)].year.unique().to
#fill outscaled year and missing year(originally nan, became 0 after to_numeric)
fillna_list = (np.random.normal(mode, std, 1000)).tolist()
Books.loc[(Books.year_of_publication<1960) | (Books.year_of_publication>2020), 'year_of_pub
Books.year_of_publication = Books.year_of_publication.fillna(pd.Series(fillna_list))
#we cannot recommend books without a title...
Books.dropna(subset=['book_title'], inplace=True)

#Ratings
Ratings.columns = Ratings.columns.str.strip().str.lower().str.replace('-', '_')
#extract explict ratings: 1-10
Ratings_ex = Ratings[Ratings.book_rating != 0]
#rescale with corrected user and book IDs from Users and Books
Book_Ratings_ex = Ratings_ex[Ratings_ex.isbn.isin(Books.isbn)]
User_Book_Ratings_ex = Book_Ratings_ex[Book_Ratings_ex.user_id.isin(Users.user_id)]
#Copy unprocessed df for plotting
User_Book_Ratings_ = User_Book_Ratings_ex.copy()#this only for plotting
#locate lazy users with ratings <10
df_copy = User_Book_Ratings_ex.copy()
df_normalize = df_copy.user_id.value_counts().rename_axis('user_id').reset_index(name = 'ra
df_normalize = df_normalize.sort_values(by=['ratings_counts'])
lazy_users = df_normalize.loc[(df_normalize.ratings_counts < 10),'user_id'].unique().tolist
#remove rows with lazy users
User_Book_Ratings_ex.loc[(User_Book_Ratings_ex['user_id'].isin(lazy_users))] = np.nan
User_Book_Ratings_ex.dropna(inplace=True)
#books with ratings
Book_with_r = User_Book_Ratings_ex.join(Books.set_index('isbn'), on='isbn')
#book,users with ratings
User_Book_r = Book_with_r.join(Users.set_index('user_id'), on='user_id')

#Surprise Read
reader = Reader(rating_scale=(1, 10))
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
```

```
C:\Users\Khushi Nitin Chavan\Anaconda3\lib\site-packages\IPython\core\intera
ctiveshell.py:3437: DtypeWarning: Columns (3) have mixed types.Specify dtype
option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [2]:

```
data
```

Out[2]:

```
<surprise.dataset.DatasetAutoFolds at 0x10eee769970>
```

In [8]:

```
data_train = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']],
data_test = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']],
# Build full trainset
data_train = data_train.build_full_trainset()
data_test = data_test.build_full_trainset()

# Create the trainset and testset
data_trainset = data_train.build_testset()
data_testset = data_test.build_testset()
```

In [3]:

```
from collections import defaultdict
def precision_recall_at_k(predictions, k = 10, threshold = 5):

    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():

        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Number of recommended items in top k
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

        # Number of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold)) for (est, true

        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1

        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1

    return precisions, recalls;
```

In [4]:

```
def get_precision_vs_recall(algo, k_max = 10, verbose = False):
    precision_list = []
    recall_list = []
    f1_score_list = []

    if algo:
        for k_curr in range(1, k_max + 1):
            algo.fit(data_train)
            predictions = algo.test(data_testset)

            # Get precision and recall at k metrics for each user
            precisions, recalls = precision_recall_at_k(predictions, k = k_curr, threshold

            # Precision and recall can then be averaged over all users
            precision = sum(prec for prec in precisions.values()) / len(precisions)
            recall = sum(rec for rec in recalls.values()) / len(recalls)
            f1_score = 2 * (precision * recall) / (precision + recall)

            # Save measures
            precision_list.append(precision)
            recall_list.append(recall)
            f1_score_list.append(f1_score)

            if verbose:
                print('K =', k_curr, '- Precision:', precision, ', Recall:', recall, ', F1

    return {'precision': precision_list, 'recall': recall_list, 'f1_score': f1_score_list};
```

SVD model

Using cross-validation (5 folds)

In [5]:

```
# Load SVD algorithm
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
model = SVD()
# Train on books dataset
%time model_selection.cross_validate(model, data, measures=['RMSE','MAE'],cv=5, verbose=Tru
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    1.5628  1.5772  1.5850  1.5649  1.5711  1.5722  0.0082  
MAE (testset)     1.1976  1.2056  1.2135  1.1991  1.2036  1.2039  0.0056  
Fit time          20.72   16.27   17.31   16.35   16.94   17.52   1.65    
Test time         0.63    0.75    0.36    0.40    0.46    0.52    0.14    
Wall time: 1min 34s
```

Out[5]:

```
{'test_rmse': array([1.56279425, 1.57718016, 1.58504639, 1.56491626, 1.57110
34 ]),
 'test_mae': array([1.19760498, 1.20558172, 1.21349832, 1.19913929, 1.203562
19]),
 'fit_time': (20.722102880477905,
  16.27432894706726,
  17.308595657348633,
  16.354618072509766,
  16.941282987594604),
 'test_time': (0.6283349990844727,
  0.7470462322235107,
  0.3645334243774414,
  0.4029219150543213,
  0.4607696533203125)}
```

In [9]:

```
k_max = 10
metrics = get_precision_vs_recall(model, k_max, True)
np.mean(metrics['f1_score'])
```

```
K = 1 - Precision: 1.0 , Recall: 0.0558141944311832 , F1 score: 0.1057273045
3061001
K = 2 - Precision: 0.9998482319016543 , Recall: 0.11158984458342144 , F1 sco
re: 0.20077215486041972
K = 3 - Precision: 0.9998482319016544 , Recall: 0.16730279458190025 , F1 sco
re: 0.28664225890097456
K = 4 - Precision: 0.9998102898770679 , Recall: 0.22285051912279724 , F1 sco
re: 0.36446451948627645
K = 5 - Precision: 0.9995548135781861 , Recall: 0.27804400802378787 , F1 sco
re: 0.43506650430103927
K = 6 - Precision: 0.9993372793038899 , Recall: 0.33339292292589284 , F1 sco
re: 0.4999841317897729
K = 7 - Precision: 0.998785855213234 , Recall: 0.38799602187922577 , F1 scor
e: 0.5588837652601341
K = 8 - Precision: 0.9977695509832405 , Recall: 0.442125878911235 , F1 scor
e: 0.6127385788170228
K = 9 - Precision: 0.9957668143394321 , Recall: 0.49466670790086115 , F1 sco
re: 0.660979083650536
K = 10 - Precision: 0.9917245433105175 , Recall: 0.5440654538996029 , F1 sco
re: 0.7026521396542112
```

Out[9]:

```
0.4427910441250996
```

In [10]:

```
# Load NMF algorithm
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
model1 = NMF()
# Train on books dataset
%time model_selection.cross_validate(model1, data, measures=['RMSE','MAE'], cv=5, verbose=T
```

```
Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    2.4679  2.4529  2.4561  2.4811  2.4716  2.4659  0.0103  
MAE (testset)     2.0606  2.0522  2.0532  2.0744  2.0699  2.0621  0.0089  
Fit time          28.28   27.93   27.37   30.56   27.08   28.24   1.23    
Test time         0.40    0.45    0.88    0.56    0.44    0.55    0.17    
Wall time: 2min 29s
```

Out[10]:

```
{'test_rmse': array([2.46792147, 2.45294874, 2.45607113, 2.48109992, 2.47157
001]),
 'test_mae': array([2.06055934, 2.05218425, 2.05324615, 2.0744266 , 2.069890
66]),
 'fit_time': (28.27625870704651,
  27.93046808242798,
  27.368402004241943,
  30.56468963623047,
  27.078638792037964),
 'test_time': (0.40392017364501953,
  0.4463157653808594,
  0.8756861686706543,
  0.5649993419647217,
  0.43733716011047363)}
```

In [11]:

```
k_max = 10
metrics = get_precision_vs_recall(model1, k_max, True)
np.mean(metrics['f1_score'])
```

```
K = 1 - Precision: 1.0 , Recall: 0.05521648994577378 , F1 score: 0.104654334
86281336
K = 2 - Precision: 1.0 , Recall: 0.10978004544832297 , F1 score: 0.197841087
33722028
K = 3 - Precision: 1.0 , Recall: 0.16356320019880272 , F1 score: 0.281141927
0923261
K = 4 - Precision: 1.0 , Recall: 0.21585646243125461 , F1 score: 0.355068988
98184586
K = 5 - Precision: 1.0 , Recall: 0.26674262202630006 , F1 score: 0.421147307
0979717
K = 6 - Precision: 0.9999747053169423 , Recall: 0.3159269418144046 , F1 scor
e: 0.4801558706628909
K = 7 - Precision: 1.0 , Recall: 0.3618322603474522 , F1 score: 0.5313903494
328087
K = 8 - Precision: 0.9999810289877068 , Recall: 0.4039449434170231 , F1 scor
e: 0.5754395717612479
K = 9 - Precision: 1.0 , Recall: 0.44146904190000014 , F1 score: 0.612526567
0889468
K = 10 - Precision: 0.9999696463803307 , Recall: 0.4740243165580756 , F1 sco
re: 0.64316400219078
```

Out[11]:

```
0.42025300065088517
```

In [12]:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
CoClusteringmodel = CoClustering()

%time model_selection.cross_validate(CoClusteringmodel, data, measures=['RMSE','MAE'], cv=5
```

```
Evaluating RMSE, MAE of algorithm CoClustering on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.8317  1.8484  1.8422  1.8414  1.8355  1.8398  0.0058
MAE (testset)     1.4251  1.4331  1.4271  1.4261  1.4256  1.4274  0.0029
Fit time          18.18   16.90   17.78   16.90   17.87   17.52   0.53
Test time         0.52    0.52    0.29    0.28    0.32    0.39    0.11
Wall time: 1min 33s
```

Out[12]:

```
{'test_rmse': array([1.83170976, 1.84843945, 1.84220046, 1.84135031, 1.83549
203]),
 'test_mae': array([1.42509478, 1.43306447, 1.42713557, 1.42610254, 1.425615
68]),
 'fit_time': (18.176786184310913,
  16.8986554145813,
  17.778788328170776,
  16.896860122680664,
  17.870314359664917),
 'test_time': (0.5244486331939697,
  0.5230734348297119,
  0.28968381881713867,
  0.2822399139404297,
  0.31627321243286133)}
```

In [13]:

```
k_max = 10
metrics = get_precision_vs_recall(CoClusteringmodel, k_max, True)
np.mean(metrics['f1_score'])
```

```
<ipython-input-4-01a5b02f8352>:8: DeprecationWarning: `np.int` is a deprecat
ed alias for the builtin `int`. To silence this warning, use `int` by itsel
f. Doing this will not modify any behavior and is safe. When replacing `np.i
nt`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precis
ion. If you wish to review your current use, check the release note link for
additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/d
evdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations)
  algo.fit(data_train)

K = 1 - Precision: 0.9965093337380483 , Recall: 0.054887186501702157 , F1 sc
ore: 0.10404370301529046
K = 2 - Precision: 0.9962816815905297 , Recall: 0.10971424180054523 , F1 sco
re: 0.19766128790119783
K = 3 - Precision: 0.9933474983558469 , Recall: 0.16252666574447433 , F1 sco
re: 0.27934780765526
K = 4 - Precision: 0.9929807254515101 , Recall: 0.2156468050453936 , F1 scor
e: 0.354340961978979
K = 5 - Precision: 0.9934335002782444 , Recall: 0.27003894490648944 , F1 sco
re: 0.42464833368118976
K = 6 - Precision: 0.9906991450397101 , Recall: 0.3201165840121933 , F1 scor
e: 0.4838807149854622
K = 7 - Precision: 0.9907612976895132 , Recall: 0.37290278091696105 , F1 sco
re: 0.5418601969934774
K = 8 - Precision: 0.9898051586699338 , Recall: 0.4224594401355472 , F1 scor
e: 0.5921730722821456
K = 9 - Precision: 0.98920278386054 , Recall: 0.4710374723665644 , F1 score:
0.6381848151091793
K = 10 - Precision: 0.9862390299368523 , Recall: 0.5095498497434057 , F1 sco
re: 0.6719370044024241
```

Out[13]:

```
0.4288077898004605
```

In [14]:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
KNNBasicmodel = KNNBasic()

%time model_selection.cross_validate(KNNBasicmodel, data, measures=['RMSE','MAE'], cv=5, ve
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.9486  1.9532  1.9491  1.9589  1.9376  1.9495  0.0070
MAE (testset)     1.5231  1.5305  1.5304  1.5338  1.5214  1.5278  0.0048
Fit time          2.73    2.30    2.16    2.20    1.63    2.20    0.35
Test time         3.13    1.97    2.42    2.06    1.35    2.19    0.59
Wall time: 26.7 s
```

Out[14]:

```
{'test_rmse': array([1.9486053 , 1.9531744 , 1.94905795, 1.95886549, 1.93760
435]),
 'test_mae': array([1.52307379, 1.53054371, 1.53035045, 1.53378455, 1.521423
33]),
 'fit_time': (2.7271480560302734,
  2.301398277282715,
  2.1637730598449707,
  2.199181079864502,
  1.6296672821044922),
 'test_time': (3.1341958045959473,
  1.9652774333953857,
  2.424579620361328,
  2.057032823562622,
  1.3464152812957764)}
```

In [15]:

```
k_max = 10
metrics = get_precision_vs_recall(KNNBasicmodel, k_max, True)
np.mean(metrics['f1_score'])
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 1 - Precision: 1.0 , Recall: 0.0558141944311832 , F1 score: 0.1057273045
3061001
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 2 - Precision: 1.0 , Recall: 0.1116283888623664 , F1 score: 0.2008376000
123678
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 3 - Precision: 1.0 , Recall: 0.16744258329355116 , F1 score: 0.286853650
3459855
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 4 - Precision: 0.9998861739262407 , Recall: 0.22311344118740634 , F1 sco
re: 0.364821120634061
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 5 - Precision: 0.9996281681590531 , Recall: 0.27855444078094754 , F1 sco
re: 0.43569809731857934
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 6 - Precision: 0.9994005160115341 , Recall: 0.33377167521692325 , F1 sco
re: 0.5004178554526498
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 7 - Precision: 0.9990716851317855 , Recall: 0.3886483440786239 , F1 scor
e: 0.5596050325269689
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 8 - Precision: 0.9986353518490413 , Recall: 0.44287241559246354 , F1 sco
re: 0.6136186853219457
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 9 - Precision: 0.9980725451510077 , Recall: 0.49545593149324185 , F1 sco
re: 0.6621915420945946
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 10 - Precision: 0.9974907674406827 , Recall: 0.5445345687871568 , F1 sco
re: 0.7044867450053716
```

Out[15]:

```
0.4434257633243134
```

In [16]:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
KNNWithMeansmodel = KNNWithMeans()

%time model_selection.cross_validate(KNNWithMeansmodel, data, measures=['RMSE','MAE'], cv=5
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    1.7892  1.7989  1.7925  1.7986  1.7980  1.7955  0.0039  
MAE (testset)     1.3906  1.3935  1.3866  1.3921  1.3905  1.3906  0.0023  
Fit time          1.52    2.96    2.04    2.01    2.03    2.11    0.47    
Test time         1.97    2.62    1.46    1.29    1.37    1.74    0.50    
Wall time: 25.9 s
```

Out[16]:

```
{'test_rmse': array([1.7892331 , 1.79889853, 1.79252724, 1.79864463, 1.79796
805]),
 'test_mae': array([1.39055248, 1.39349098, 1.38658826, 1.39214935, 1.390454
18]),
 'fit_time': (1.5228564739227295,
  2.9607598781585693,
  2.0371079444885254,
  2.0131595134735107,
  2.0295889377593994),
 'test_time': (1.97261643409729,
  2.616595983505249,
  1.455655574798584,
  1.2900912761688232,
  1.3743383884429932)}
```

In [18]:

```
k_max = 10
metrics = get_precision_vs_recall(KNNWithMeansmodel, k_max, True)
np.mean(metrics['f1_score'])
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 1 - Precision: 1.0 , Recall: 0.0557610755967622 , F1 score: 0.1056319974
0100972
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 2 - Precision: 1.0 , Recall: 0.11139784589392698 , F1 score: 0.200464390
50694165
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 3 - Precision: 1.0 , Recall: 0.16678232636527177 , F1 score: 0.285884217
81263607
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 4 - Precision: 1.0 , Recall: 0.22189241975087412 , F1 score: 0.363194690
73409055
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 5 - Precision: 0.9999696463803309 , Recall: 0.27676027864209213 , F1 sco
re: 0.4335323744542044
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 6 - Precision: 0.999893762331158 , Recall: 0.3311963249486476 , F1 scor
e: 0.49757885298343946
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 7 - Precision: 0.9998142647558341 , Recall: 0.38508257213650193 , F1 sco
re: 0.5560140488079885
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 8 - Precision: 0.9997329604174345 , Recall: 0.43792602727380936 , F1 sco
re: 0.6090583197248677
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 9 - Precision: 0.9994547189038002 , Recall: 0.48853384179334786 , F1 sco
re: 0.6562785043129665
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 10 - Precision: 0.9989488252426476 , Recall: 0.5354894162341955 , F1 sco
re: 0.6972278307690934
```

Out[18]:

```
0.4404865227507238
```

In [19]:

```
data = Dataset.load_from_df(User_Book_Ratings_ex[['user_id', 'isbn', 'book_rating']], reade
KNNWithZScoremodel = KNNWithZScore()

%time model_selection.cross_validate(KNNWithZScoremodel, data, measures=['RMSE','MAE'], cv=
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithZScore on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    1.7919  1.8036  1.7951  1.7914  1.7995  1.7963  0.0047  
MAE (testset)     1.3755  1.3933  1.3801  1.3786  1.3839  1.3823  0.0062  
Fit time          2.01    3.67    3.96    2.46    2.77    2.97    0.73    
Test time         1.78    4.29    1.87    1.60    1.67    2.24    1.03    
Wall time: 30.9 s
```

Out[19]:

```
{'test_rmse': array([1.791883  , 1.80364886, 1.79506702, 1.79137289, 1.79949
415]),
 'test_mae': array([1.37549781, 1.39334241, 1.38013343, 1.37856644, 1.383889
83]),
 'fit_time': (2.007124185562134,
  3.670954942703247,
  3.956085205078125,
  2.458000659942627,
  2.7651689052581787),
 'test_time': (1.7838726043701172,
  4.291193246841431,
  1.869053840637207,
  1.603726863861084,
  1.6671013832092285)}
```

In [20]:

```
k_max = 10
metrics = get_precision_vs_recall(KNNWithZScoremodel, k_max, True)
np.mean(metrics['f1_score'])
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 1 - Precision: 1.0 , Recall: 0.0557610755967622 , F1 score: 0.1056319974
0100972
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 2 - Precision: 1.0 , Recall: 0.11137399662132981 , F1 score: 0.200425773
7897703
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 3 - Precision: 1.0 , Recall: 0.16677173216360153 , F1 score: 0.285868653
7671745
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 4 - Precision: 1.0 , Recall: 0.2218711656470582 , F1 score: 0.3631662189
680421
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 5 - Precision: 1.0 , Recall: 0.27674753717199757 , F1 score: 0.433519594
30443826
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 6 - Precision: 0.9999494106338848 , Recall: 0.3310665599816539 , F1 scor
e: 0.49743927772879526
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 7 - Precision: 0.9998446183755032 , Recall: 0.3847135260378307 , F1 scor
e: 0.5556339402245543
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 8 - Precision: 0.9997795749047836 , Recall: 0.43720501067215817 , F1 sco
re: 0.6083692812063879
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 9 - Precision: 0.99965520937975 , Recall: 0.4876773339649928 , F1 score:
0.6555483366190763
Computing the msd similarity matrix...
Done computing similarity matrix.
K = 10 - Precision: 0.9991999050847126 , Recall: 0.5345184458864329 , F1 sco
re: 0.6964652669866893
```

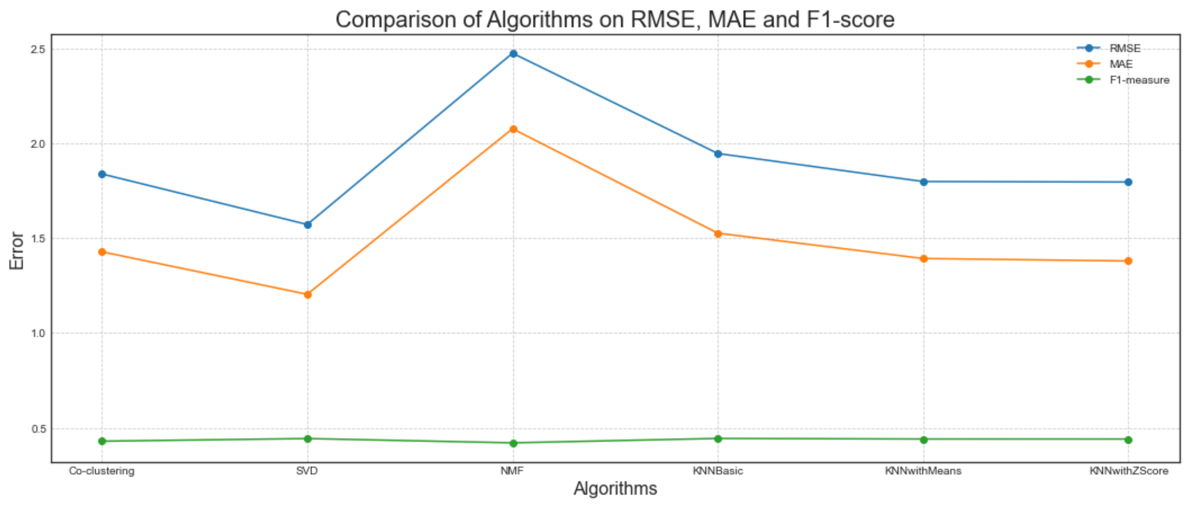
Out[20]:

```
0.4402068340995938
```

Comparison of different Algorithms

In [21]:

```
import matplotlib.pyplot as plt
Algorithms = ['Co-clustering' ,'SVD','NMF','KNNBasic','KNNwithMeans','KNNwithZScore']
#mannullt recode...since re-running takes too much time
mae_results =[1.4286,1.2041,2.0782,1.5268,1.3931,1.3801]
rmse_results =[1.8393,1.5726,2.4767,1.9473,1.7994,1.7967]
f1_results =[0.4289,0.4428,0.4202,0.4434,0.4404,0.4402]
plt.figure(figsize=(18,7))
plt.title('Comparison of Algorithms on RMSE, MAE and F1-score', loc='center', fontsize=20)
plt.plot(Algorithms, rmse_results, label='RMSE', marker='o')
plt.plot(Algorithms, mae_results, label='MAE', marker='o')
plt.plot(Algorithms, f1_results, label='F1-measure', marker='o')
plt.xlabel('Algorithms', fontsize=16)
plt.ylabel('Error', fontsize=16)
plt.grid(ls='dashed')
plt.legend()
plt.show()
```



In [ ]: