Problem 1

In [4]:

```
import autograd.numpy as ag_np

# Use helper packages
from AbstractBaseCollabFilterSGD import AbstractBaseCollabFilterSGD
from utils import load_dataset
import numpy as np
# Some packages you might need (uncomment as necessary)
import pandas as pd
import matplotlib.pyplot as plt
import os
```

In [4]:

```
from CollabFilterMeanOnly import CollabFilterMeanOnly
train_tuple, valid_tuple, test_tuple, n_users, n_items = load_dataset()
n_epochs=50
model = CollabFilterMeanOnly(n_epochs=n_epochs)
model.init_parameter_dict(n_users, n_items, train_tuple)
model.fit(train_tuple, valid_tuple)
```

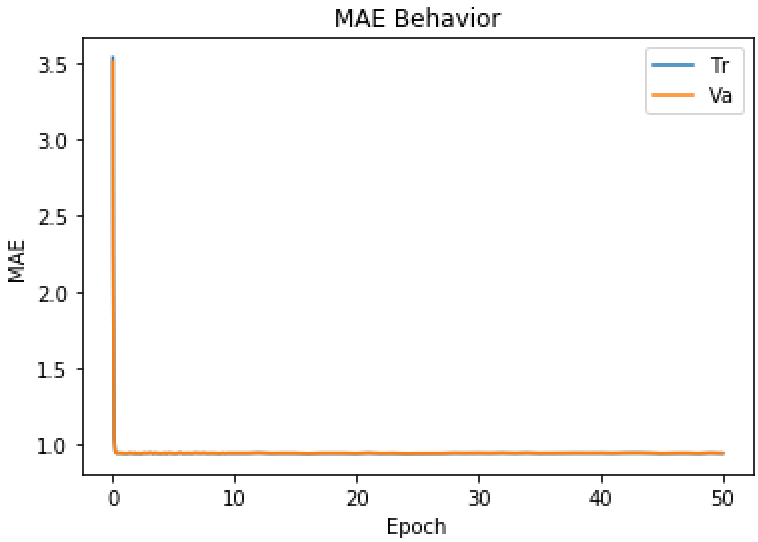
```
 poch       0.000 | loss_total    13.82200 | train_MAE     3.53239 | valid
_MAE     3.50620 | grad_wrt_mu     7.10400
 poch       0.013 | loss_total     9.11851 | train_MAE     2.82199 | valid
_MAE     2.79580 | grad_wrt_mu     5.63320
 poch       0.025 | loss_total     6.60551 | train_MAE     2.29207 | valid
_MAE     2.26637 | grad_wrt_mu     4.63456
 poch       0.100 | loss_total     1.68633 | train_MAE     1.05029 | valid
_MAE     1.03490 | grad_wrt_mu     1.24038
 poch       0.200 | loss_total     1.26837 | train_MAE     0.95532 | valid
_MAE     0.95363 | grad_wrt_mu     0.10245
 poch       0.313 | loss_total     1.28873 | train_MAE     0.94556 | valid
_MAE     0.94652 | grad_wrt_mu     0.04033
 poch       0.400 | loss_total     1.32975 | train_MAE     0.94431 | valid
_MAE     0.94560 | grad_wrt_mu     0.02424
 poch       0.500 | loss_total     1.21539 | train_MAE     0.94557 | valid
_MAE     0.94652 | grad_wrt_mu     0.00853
 poch       0.613 | loss_total     1.27747 | train_MAE     0.94446 | valid
_MAE     0.94571 | grad_wrt_mu     0.01354
 poch       0.713 | loss_total     1.24705 | train_MAE     0.94475 | valid
```

In [17]:

```
plt.figure
plt.plot(model.trace_epoch, model.trace_mae_train, label='Tr')
plt.plot(model.trace_epoch, model.trace_mae_valid, label='Va')
plt.title('MAE Behavior')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
```

Out[17]:

```
<matplotlib.legend.Legend at 0x1a88bf8cf98>
```

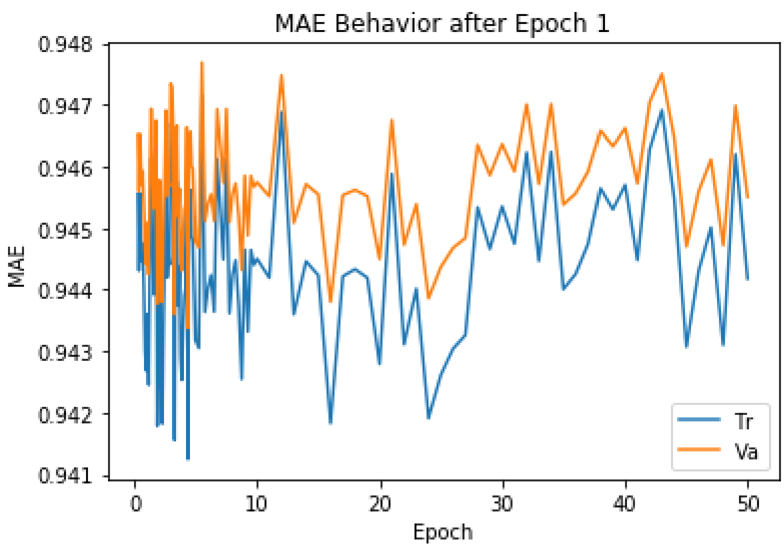


In [18]:

```
plt.figure
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.title('MAE Behavior after Epoch 1')
plt.plot(model.trace_epoch[5:], model.trace_mae_train[5:], label='Tr')
plt.plot(model.trace_epoch[5:], model.trace_mae_valid[5:], label='Va')
plt.legend()
```

Out[18]:

```
<matplotlib.legend.Legend at 0x1a88bff7d68>
```



Questions:

1b No, we are using mean as prediction. Forcing μ to be small by adding penalty, doesn't help the prediction at all. In fact, it's driving μ away from the optimal result.
What we need to consider is whether a parameter implicates complexity of a model so we want to penalize it. Statistical parameters like mean and var are not a measure of complexity and we don't want to penalize it.
1c The acual average is 3.53239. The SGD prediction is 3.53671, which agrees with the actual ave.

In [24]:

```
print(np.mean(train_tuple[2]), model.param_dict["mu"][0])
```

```
3.5323907390739073 3.536711325075617
```

In [ ]:

```
```

Problem2

In [10]:

```
from CollabFilterOneScalarPerItem import CollabFilterOneScalarPerItem
```

In [11]:

```
train_tuple, valid_tuple, test_tuple, n_users, n_items = load_dataset()
model2 = CollabFilterOneScalarPerItem(
        n_epochs=250, step_size=0.5)
model2.init_parameter_dict(n_users, n_items, train_tuple)
model2.fit(train_tuple, valid_tuple)
```

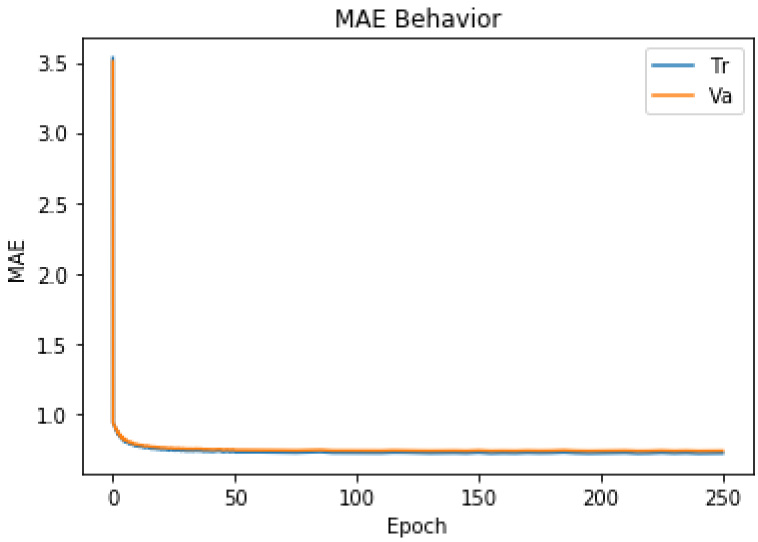
```
 poch       0.000 | loss_total    13.82200 | train_MAE     3.53239 | valid
_MAE     3.50620 | grad_wrt_mu     7.10400 | grad_wrt_b_per_user     0.007
 3 | grad_wrt_c_per_item     0.00423
 poch       0.013 | loss_total     1.18426 | train_MAE     0.93962 | valid
_MAE     0.94187 | grad_wrt_mu     0.07603 | grad_wrt_b_per_user     0.001
 8 | grad_wrt_c_per_item     0.00082
 poch       0.025 | loss_total     1.23511 | train_MAE     0.94317 | valid
_MAE     0.94429 | grad_wrt_mu     0.12661 | grad_wrt_b_per_user     0.001
 2 | grad_wrt_c_per_item     0.00084
 poch       0.100 | loss_total     1.28834 | train_MAE     0.93797 | valid
_MAE     0.93971 | grad_wrt_mu     0.01212 | grad_wrt_b_per_user     0.001
 4 | grad_wrt_c_per_item     0.00088
 poch       0.200 | loss_total     1.24465 | train_MAE     0.93125 | valid
_MAE     0.93359 | grad_wrt_mu     0.14125 | grad_wrt_b_per_user     0.001
 6 | grad_wrt_c_per_item     0.00086
 poch       0.313 | loss_total     1.24821 | train_MAE     0.92916 | valid
_MAE     0.93086 | grad_wrt_mu     0.02762 | grad_wrt_b_per_user     0.001
 6 | grad_wrt_c_per_item     0.00088
 poch       0.400 | loss_total     1.26587 | train_MAE     0.92292 | valid
```

In [12]:

```
plt.figure
plt.plot(model2.trace_epoch, model2.trace_mae_train, label='Tr')
plt.plot(model2.trace_epoch, model2.trace_mae_valid, label='Va')
plt.title('MAE Behavior')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
```

Out[12]:

```
<matplotlib.legend.Legend at 0x1d12e3fa7b8>
```

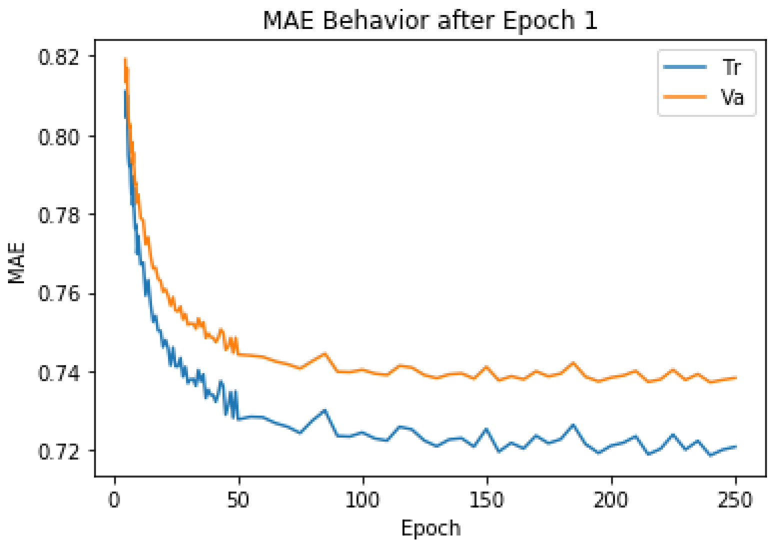


In [13]:

```
plt.figure
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.title('MAE Behavior after Epoch 1')
plt.plot(model2.trace_epoch[50:], model2.trace_mae_train[50:], label='Tr')
plt.plot(model2.trace_epoch[50:], model2.trace_mae_valid[50:], label='Va')
plt.legend()
```

Out[13]:

```
<matplotlib.legend.Legend at 0x1d12dda95c0>
```



Questions:

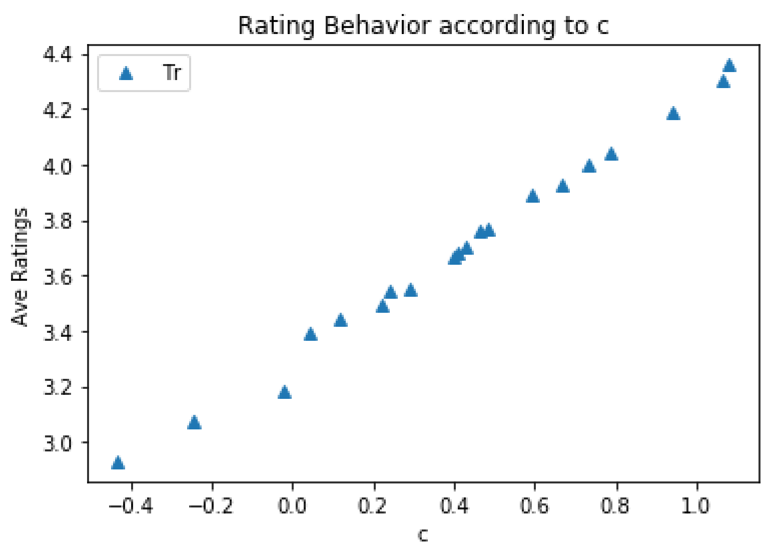
2a The MAE performance on the Validation set improved by over 20% (from 0.945 to 0.741).
2b From the Picture Below, rating seems to have a linear dependence on trained c. When c is larger, the average rating of a movie seems higher. When c is small(very negative), the average rating is lower.

In [14]:

```
#2b
data_path='C:/Users/xush4/Documents/comp135-19s-assignments-master/project3/data_movie_lens
sel_df = pd.read_csv(os.path.join(data_path, "select_movies.csv"))
aveT=[]; c_sel=[]
for i in sel_df["item_id"]:
    idx=train_tuple[1]
    aveTid=np.mean(train_tuple[2][np.where(idx==i)])
    aveT.append(aveTid);
    c_sel.append(model2.param_dict["c_per_item"][i])
    ##print(i, model2.param_dict["c_per_item"][i], aveid)
plt.plot(c_sel, aveT, linestyle='', marker='^', label="Tr")
plt.xlabel('c')
plt.ylabel('Ave Ratings')
plt.title('Rating Behavior according to c')
plt.legend()
```

Out[14]:

```
<matplotlib.legend.Legend at 0x1d12c10c9e8>
```



In [ ]:

Problem3

In [6]:

```
from CollabFilterOneVectorPerItem import CollabFilterOneVectorPerItem as CFV
```

In [4]:

```
K=[0,2,10,50]
model3={};
for i in range(4):
    train_tuple, valid_tuple, test_tuple, n_users, n_items = load_dataset()
    model3[i] = CFV(n_epochs=250, step_size=0.5, n_factors=K[i])
    model3[i].init_parameter_dict(n_users, n_items, train_tuple)
    model3[i].fit(train_tuple, valid_tuple)
```

```
:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3118:
untimeWarning: Mean of empty slice.
 out=out, **kwargs)
:\ProgramData\Anaconda3\lib\site-packages\numpy\core\_methods.py:85: Runt
meWarning: invalid value encountered in double_scalars
 ret = ret.dtype.type(ret / rcount)
```

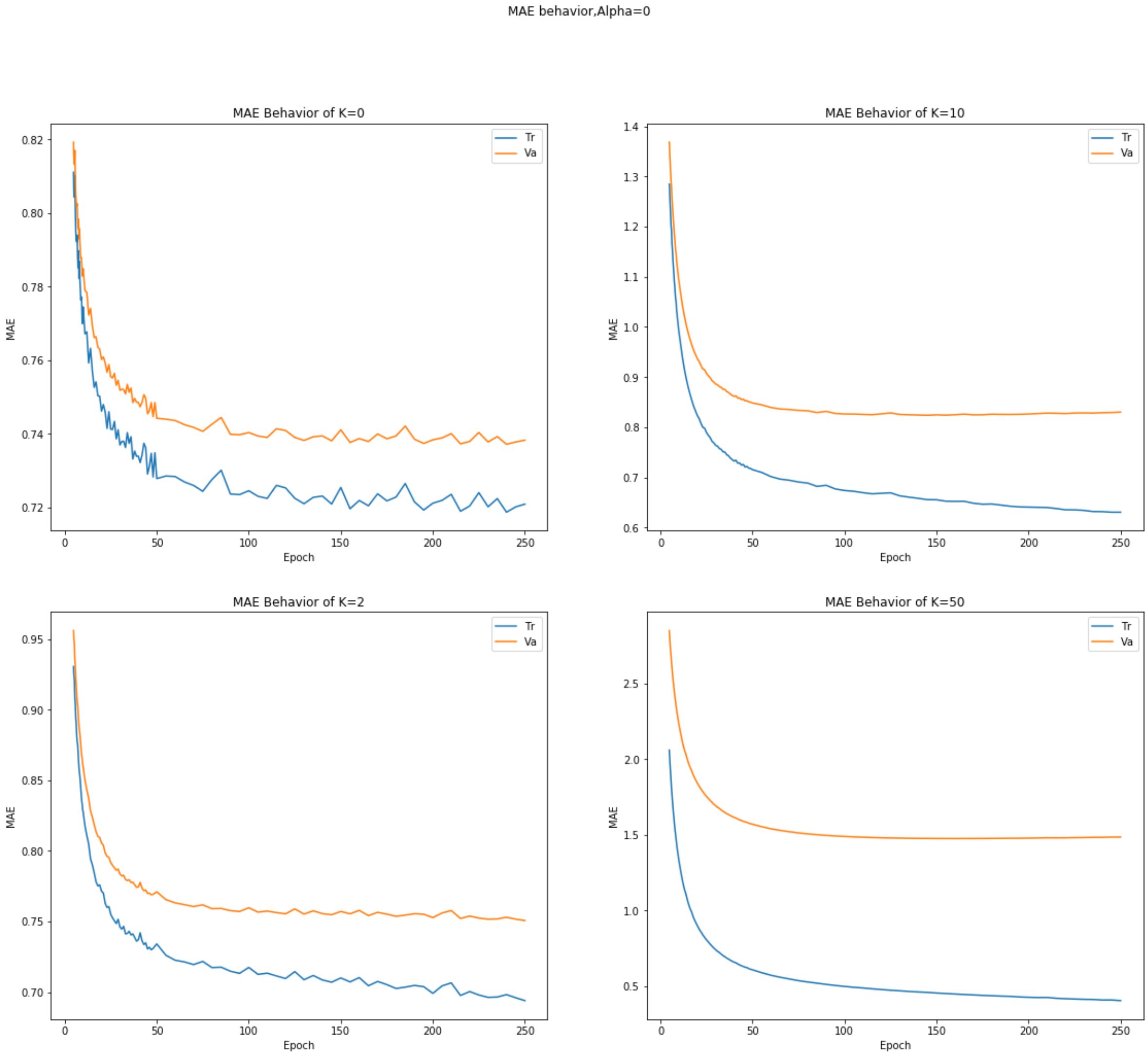
In [16]:

```
fig3a, axes = plt.subplots(nrows=2, ncols=2, figsize=(19,16))
for i in range(2):
    for j in range(2):
        axes[i,j].set_xlabel('Epoch')
        axes[i,j].set_ylabel('MAE')
        axes[i,j].set_title('MAE Behavior of K=' + str(int(K[i+2*j])))
        axes[i,j].plot(model3[i+2*j].trace_epoch[50:], model3[i+2*j].trace_mae_train[50:],
        axes[i,j].plot(model3[i+2*j].trace_epoch[50:], model3[i+2*j].trace_mae_valid[50:],
        axes[i,j].legend()

fig3a.suptitle('MAE behavior,'+ 'Alpha='+str(0))
```

Out[16]:

```
Text(0.5,0.98,'MAE behavior,Alpha=0')
```



In [7]:

```
### 3b
K=[0,2,10,50]
a=np.logspace(-10,2)
tr_error=[];
va_error=[];
for i in range(4):
    te=[]; ve=[];
    for j in range(int(a.size)):
        train_tuple, valid_tuple, test_tuple, n_users, n_items = load_dataset()
        model3b = CFV(n_epochs=250, step_size=0.5, n_factors=K[i], alpha=a[j])
        model3b.init_parameter_dict(n_users, n_items, train_tuple)
        model3b.fit(train_tuple, valid_tuple)
        te.append(model3b.trace_mae_train[-1])
        ve.append(model3b.trace_mae_valid[-1])
    tr_error.append(te)
    va_error.append(ve)
```

```
:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3118:
untimeWarning: Mean of empty slice.
 out=out, **kwargs)
:\ProgramData\Anaconda3\lib\site-packages\numpy\core\_methods.py:85: Runt
meWarning: invalid value encountered in double_scalars
 ret = ret.dtype.type(ret / rcount)
```

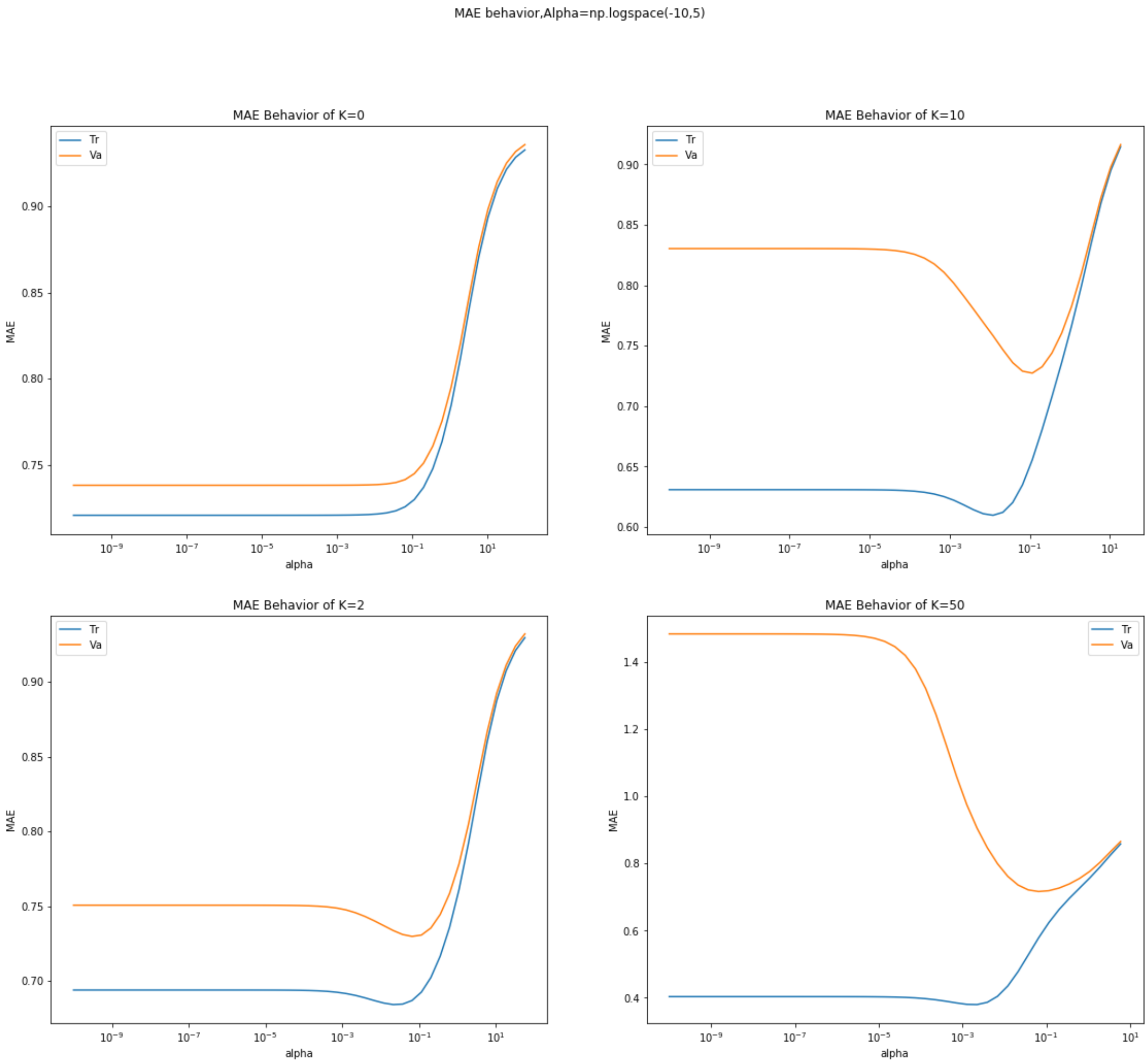
In [8]:

```
fig3a, axes = plt.subplots(nrows=2, ncols=2, figsize=(19,16))
for i in range(2):
    for j in range(2):
        axes[i,j].set_xlabel('alpha')
        axes[i,j].set_ylabel('MAE')
        axes[i,j].set_title('MAE Behavior of K=' + str(int(K[i+2*j])))
        axes[i,j].plot(a, tr_error[i+2*j], label='Tr')
        axes[i,j].plot(a, va_error[i+2*j], label='Va')
        axes[i,j].legend()
        axes[i,j].set_xscale('log')

fig3a.suptitle('MAE behavior,'+ 'Alpha=np.logspace(-10,5)')
```

Out[8]:

```
Text(0.5,0.98,'MAE behavior,Alpha=np.logspace(-10,5)')
```



3c We can use a small K or shrink the size of each batch.

3d K=0 and K=2 are performing a little better than the model in question 1 and 2.
I would recommend use 0 or 2 factors.
There's no need to look into more than 50 factors because it's not increasing the performance(probably overfitting).

In [7]:

```
#2 Factors:
train_tuple, valid_tuple, test_tuple, n_users, n_items = load_dataset()
model3e = CFV(n_epochs=250, step_size=0.5, n_factors=2, alpha=0.1)
model3e.init_parameter_dict(n_users, n_items, train_tuple)
model3e.fit(train_tuple, valid_tuple)
```

```
 poch       0.000 | loss_total     6.21473 | train_MAE     1.50279 | valid
_MAE     1.49613 | grad_wrt_mu     0.92071 | grad_wrt_b_per_user     0.002
 7 | grad_wrt_c_per_item     0.00149 | grad_wrt_U     0.00348 | grad_wrt_V
 .00234
 poch       0.013 | loss_total     6.17911 | train_MAE     1.41942 | valid
_MAE     1.42198 | grad_wrt_mu     0.12156 | grad_wrt_b_per_user     0.002
 9 | grad_wrt_c_per_item     0.00141 | grad_wrt_U     0.00342 | grad_wrt_V
 .00239
 poch       0.025 | loss_total     6.03396 | train_MAE     1.41146 | valid
_MAE     1.41516 | grad_wrt_mu     0.03582 | grad_wrt_b_per_user     0.002
 9 | grad_wrt_c_per_item     0.00138 | grad_wrt_U     0.00333 | grad_wrt_V
 .00230
 poch       0.100 | loss_total     5.76014 | train_MAE     1.39247 | valid
_MAE     1.39530 | grad_wrt_mu     0.19923 | grad_wrt_b_per_user     0.002
 5 | grad_wrt_c_per_item     0.00135 | grad_wrt_U     0.00326 | grad_wrt_V
 .00225
 poch       0.200 | loss_total     5.32516 | train_MAE     1.35524 | valid
_MAE     1.36099 | grad_wrt_mu     0.01691 | grad_wrt_b_per_user     0.002
 0 | grad_wrt_c_per_item     0.00132 | grad_wrt_U     0.00293 | grad_wrt_V
```

In [28]:

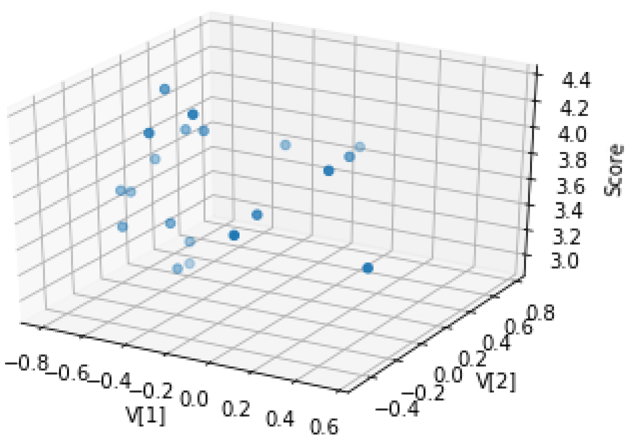
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

data_path='C:/Users/xush4/Documents/comp135-19s-assignments-master/project3/data_movie_lens
sel_df = pd.read_csv(os.path.join(data_path, "select_movies.csv"))
aveT=[]; idex=[]
for i in sel_df["item_id"]:
    idx=train_tuple[1]
    aveTid=np.mean(train_tuple[2][np.where(idx==i)])
    aveT.append(aveTid);
    idex.append(i)

V=model3e.param_dict['V'][idex]
#print(V[:,0].size, V[:,1].size, aveT)
ax.scatter(V[:,0], V[:,1], aveT, zdir='z')

ax.set_xlabel('V[1]')
ax.set_ylabel('V[2]')
ax.set_zlabel('Score')
print(V[:,0], V[:,1], aveT)
```

```
  0.08239941 -0.32913701 -0.0372406   0.22327492 -0.11569277 -0.32949667
 -0.51060424 -0.50746085 -0.79073236 -0.592902   -0.65224791 -0.64892778
 -0.80690433 -0.80711085 -0.6925586   0.54269781  0.06421546 -0.53089196
 -0.50387622 -0.48682363] [ 0.3561754   0.53450276 -0.15376184 -0.04260292
 .77782893 -0.15445973
 -0.19469097  0.1935403   0.25841157  0.19861086  0.13559631  0.2752468
  0.02343947  0.09642642 -0.13832645 -0.27475537 -0.48355823  0.1091072
 -0.07069955 -0.05248082] [3.888888888888889, 3.76271186440678, 3.7007299270
 72993, 4.045, 3.67816091954023, 4.359574468085106, 4.191335740072202, 4.002
 50980392157, 3.6666666666666665, 2.9302325581395348, 4.306990881458966, 3.9
 8030303030303, 3.546153846153846, 3.4939759036144578, 3.3943661971830985,
 .5535714285714284, 3.7710843373493974, 3.180722891566265, 3.441624365482233
 , 3.076923076923077]
```



The trend is not that obvious, it seems to me items with smaller V[1] and bigger V[2] are more controversial(either very high scores or very low scores.).

Problem 4

In [160]:

```
# Reference https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.pr
from surprise.prediction_algorithms.matrix_factorization import SVD
```

In [149]:

```
import pandas as pd
import numpy as np

from surprise import SVD
from surprise import Dataset, Reader, accuracy
from surprise.model_selection import cross_validate, KFold

reader = Reader(
    line_format='user item rating', sep=',',
    rating_scale=(1, 5), skip_lines=1)

## Load the training set into surprise's custom dataset object
train_df = pd.read_csv('data_movie_lens_100k/ratings_train.csv')
train_set = Dataset.load_from_file('data_movie_lens_100k/ratings_train.csv', reader=reader)


## Load the test set into surprise's custom dataset object
## (Need to use intermediate pandas DataFrame because the true ratings are missing)
test_df = pd.read_csv('data_movie_lens_100k/ratings_test_masked.csv')
test_set = Dataset.load_from_df(test_df, reader=reader)
test_set = test_set.build_full_trainset().build_testset()
print(type(test_df['user_id'][0]))
```

```
<class 'numpy.int64'>
```

In [48]:

```
numF=5
kf = KFold(n_splits=numF)
K=[0,2,5,10]
a=np.logspace(-3,1,17)
tr_error=[];
va_error=[];

for i in range(4):
    te=[]; ve=[];
    for j in range(int(a.size)):
        sumTe=0; sumVa=0;
        for trainset, validset in kf.split(train_set):
            model4 = SVD(n_epochs=250, n_factors=K[i], lr_all=a[j])
        # train and test algorithm.
            model4.fit(trainset)
            pre1 = model4.test(trainset.build_testset())
            pre2 = model4.test(validset)
    # Compute and print Root Mean Squared Error
            sumTe=sumTe+accuracy.mae(pre1, verbose=True)
            sumVa=sumVa+accuracy.mae(pre2, verbose=True)
        print(sumTe/numF)
        te.append(sumTe/numF);
        ve.append(sumVa/numF);
    tr_error.append(te)
    va_error.append(ve)
```

```
MAE:  0.5863
MAE:  0.7427
MAE:  0.5889
MAE:  0.7405
MAE:  0.5840
MAE:  0.7499
MAE:  0.5906
MAE:  0.7426
0.5869958185262447
MAE:  0.5368
MAE:  0.7519
MAE:  0.5346
MAE:  0.7572
MAE:  0.5365
MAE:  0.7670
MAE:  0.5365
MAE:  0.7524
MAE:  0.5381
MAE:  0.7549
```

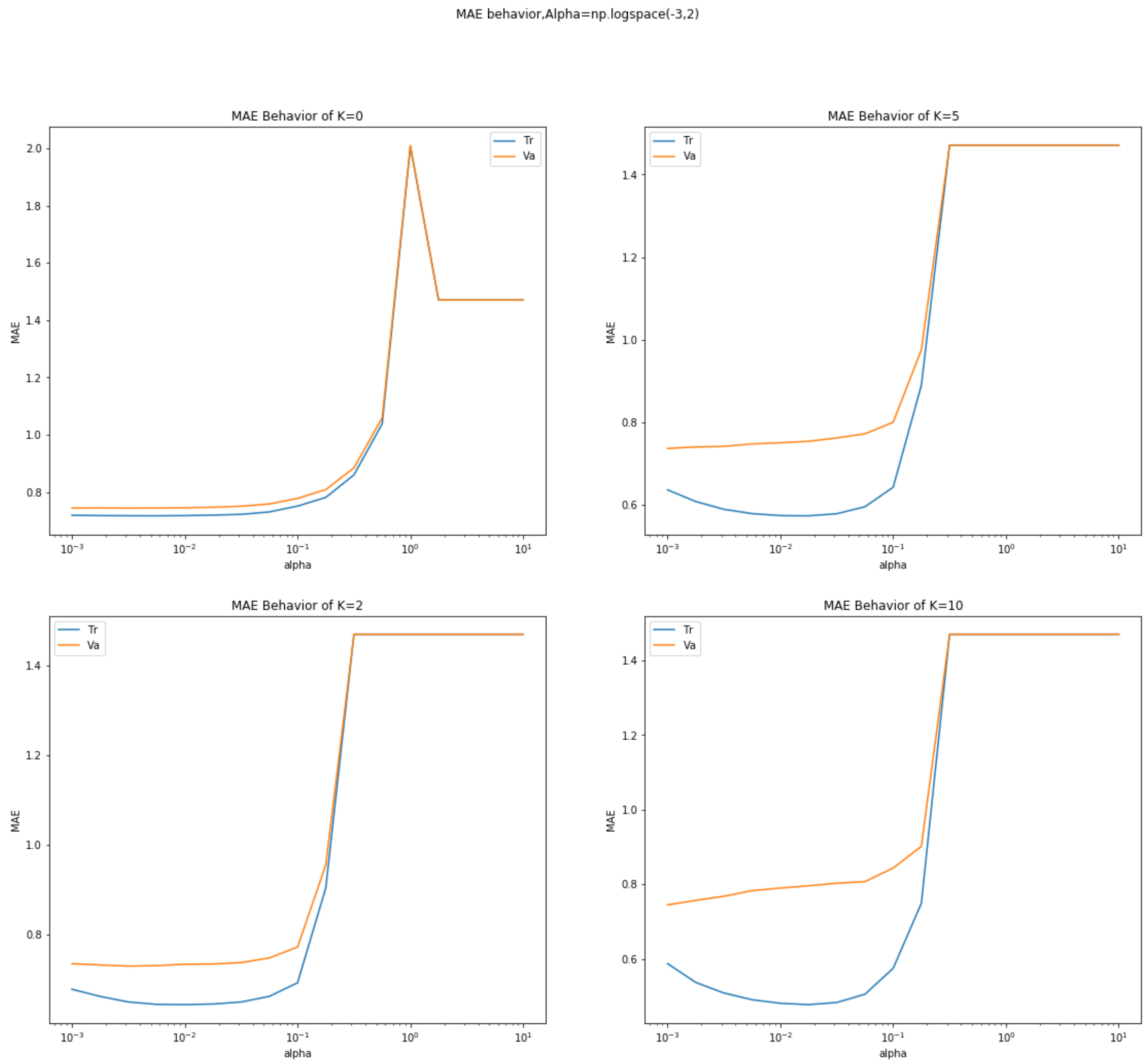
In [49]:

```
#print(tr_error)
fig3a, axes = plt.subplots(nrows=2, ncols=2, figsize=(19,16))
for i in range(2):
    for j in range(2):
        axes[i,j].set_xlabel('alpha')
        axes[i,j].set_ylabel('MAE')
        axes[i,j].set_title('MAE Behavior of K=' + str(int(K[i+2*j])))
        axes[i,j].plot(a, tr_error[i+2*j], label='Tr')
        axes[i,j].plot(a, va_error[i+2*j], label='Va')
        axes[i,j].legend()
        axes[i,j].set_xscale('log')

fig3a.suptitle('MAE behavior,'+ 'Alpha=np.logspace(-3,2)')
```

Out[49]:

```
Text(0.5,0.98,'MAE behavior,Alpha=np.logspace(-3,2)')
```



The trend on $\alpha \leq 0.1$ is pretty identical. The trend over $\alpha > 0.1$ is a diffrent. It's probably because the step size we use is larger so the method can not converge near enough to the optimal solution.

In [202]:

```
reader = Reader(
    line_format='user item rating', sep=',',
    rating_scale=(1, 5), skip_lines=1)

## Load the training set into surprise's custom dataset object
## (Need to use intermediate pandas DataFrame here because that's what needed on test set)
train_df = pd.read_csv('data_movie_lens_100k/ratings_train.csv')
train_set = Dataset.load_from_df(train_df, reader=reader)
train_set = train_set.build_full_trainset()

## Load the test set into surprise's custom dataset object
## (Need to use intermediate pandas DataFrame because the true ratings are missing)
test_df = pd.read_csv('data_movie_lens_100k/ratings_test_masked.csv')
test_set = Dataset.load_from_df(test_df, reader=reader)
test_set = test_set.build_full_trainset().build_testset()

# Use the SVD algorithm
    ## Fit model to training set
model = SVD(n_factors=2)
model.fit(train_set)

## Measure predictions on train set
print("Making predictions on training set (showing first 10):")
tr_pred = model.test(train_set.build_testset())
tr_mae = accuracy.mae(tr_pred)
tr_predicted_ratings_N = np.asarray([p.est for p in tr_pred], dtype=np.float64)
print(tr_predicted_ratings_N[:10])

## Measure predictions on test set
print("Making predictions on test set (showing first 10):")
te_pred = model.test(test_set)
te_mae = accuracy.mae(te_pred) # should be NaN because no real labels on testset
te_predicted_ratings_N = np.asarray([p.est for p in te_pred], dtype=np.float64)
print(te_predicted_ratings_N[:10])

print("n_factors %6d  tr_MAE %7.3f  test_MAE %7.3f" % (n_factors, tr_mae, te_mae))

print("Making test set predictions in the original order")
for row in test_df.values[:10]:
        userid = row[0]
        itemid = row[1]
        rhat = model.predict(userid, itemid)
        print("user %4d  item %4d  predicted rating % 8.3f" % (userid, itemid, rhat.est))
tep0=[]
for row in test_df.values[:]:
        userid = row[0]
        itemid = row[1]
        rhat = model.predict(userid, itemid)
        #print("user %4d  item %4d  predicted rating % 8.3f" % (userid, itemid, rhat.est))
        tep0.append(rhat.est)
np.savetxt('predicted_ratings_test4.txt', np.asarray(tep0))
```

```
Making predictions on training set (showing first 10):
MAE:  0.7184
[2.69729623 3.61916657 2.80442954 3.72018483 2.77842302 3.47178856
 3.28690477 3.43117354 3.70169473 2.63116753]
Making predictions on test set (showing first 10):
MAE:  nan
[4.20307785 3.23106528 2.51561078 3.45627649 3.97710067 3.86847339
```

```
 2.98476368 3.7159181  3.16108109 3.64082798]
n_factors      2  tr_MAE   0.718  test_MAE     nan
Making test set predictions in the original order
user  503  item  204  predicted rating    4.203
user  795  item  185  predicted rating    3.908
user   42  item  403  predicted rating    3.751
user  327  item  740  predicted rating    3.450
user  285  item   98  predicted rating    3.989
user  279  item   11  predicted rating    4.483
user  496  item  588  predicted rating    2.373
user  499  item  266  predicted rating    3.572
user  357  item  126  predicted rating    4.146
user  932  item  182  predicted rating    3.213
```

Problem5

In [50]:

```
#Kmeans:
```

In [196]:

```
numF=5
kf = KFold(n_splits=numF)
train_df = pd.read_csv('data_movie_lens_100k/ratings_train.csv')
train_set = Dataset.load_from_file('data_movie_lens_100k/ratings_train.csv', reader=reader)

total_sample=train_df.shape[0]
K_nbh=((1-1/numF)*total_sample)**np.linspace(0.25, 0.75)
K_nbh=[1,5,10,25,50, int(np.sqrt((1-1/numF)*total_sample))]
```

In [90]:

```
print(K_nbh)
```

```
[1, 5, 10, 25, 50, 268]
```

In [91]:

```
##MSD with mean
from surprise.prediction_algorithms.knns import KNNWithMeans as KNNM
te=[]; ve=[];
for k in K_nbh:
    sumTe=0; sumVa=0;
    for trainset, validset in kf.split(train_set):
            model5=KNNM(k=int(k))
        # train and test algorithm.
            model5.fit(trainset)
            pre1 = model5.test(trainset.build_testset())
            pre2 = model5.test(validset)
            sumTe=sumTe+accuracy.mae(pre1, verbose=True)
            sumVa=sumVa+accuracy.mae(pre2, verbose=True)
    te.append(sumTe/numF);
    ve.append(sumVa/numF);
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0522
MAE:  0.9743
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0491
MAE:  0.9796
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0504
MAE:  0.9749
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0492
MAE:  0.9698
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0512
```

In [197]:

```
##MSD with mean
from surprise.prediction_algorithms.knns import KNNWithMeans as KNNM
tea=[]; vea=[];
for k in K_nbh:
    sumTe=0; sumVa=0;
    for trainset, validset in kf.split(train_set):
            model5=KNNM(k=int(k),sim_options={'name': 'pearson'})
        # train and test algorithm.
            model5.fit(trainset)
            pre1 = model5.test(trainset.build_testset())
            pre2 = model5.test(validset)
            sumTe=sumTe+accuracy.mae(pre1, verbose=True)
            sumVa=sumVa+accuracy.mae(pre2, verbose=True)
    tea.append(sumTe/numF);
    vea.append(sumVa/numF);
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1289
MAE:  0.9988
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1305
MAE:  1.0057
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1313
MAE:  1.0045
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1326
MAE:  1.0004
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1307
```

In [92]:

```
##MSD
from surprise.prediction_algorithms.knns import KNNBasic as KNN
teb=[]; veb=[];
for k in K_nbh:
    sumTe=0; sumVa=0;
    for trainset, validset in kf.split(train_set):
            model5=KNN(k=int(k))
        # train and test algorithm.
            model5.fit(trainset)
            pre1 = model5.test(trainset.build_testset())
            pre2 = model5.test(validset)
            sumTe=sumTe+accuracy.mae(pre1, verbose=True)
            sumVa=sumVa+accuracy.mae(pre2, verbose=True)
    teb.append(sumTe/numF);
    veb.append(sumVa/numF);
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0000
MAE:  0.9784
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0000
MAE:  0.9739
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0000
MAE:  0.9722
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0000
MAE:  0.9740
Computing the msd similarity matrix...
Done computing similarity matrix.
MAE:  0.0000
```

In [93]:

```
## Pearson
from surprise.prediction_algorithms.knns import KNNBasic as KNN
tec=[]; vec=[];

for k in K_nbh:
    sumTe=0; sumVa=0;
    for trainset, validset in kf.split(train_set):
            model5=KNN(k=int(k),sim_options={'name': 'pearson'})
        # train and test algorithm.
            model5.fit(trainset)
            pre1 = model5.test(trainset.build_testset())
            pre2 = model5.test(validset)
            sumTe=sumTe+accuracy.mae(pre1, verbose=True)
            sumVa=sumVa+accuracy.mae(pre2, verbose=True)
    tec.append(sumTe/numF);
    vec.append(sumVa/numF);
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1378
MAE:  1.0490
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1337
MAE:  1.0582
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1337
MAE:  1.0559
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1344
MAE:  1.0568
Computing the pearson similarity matrix...
Done computing similarity matrix.
MAE:  0.1350
```

Summary of 5:

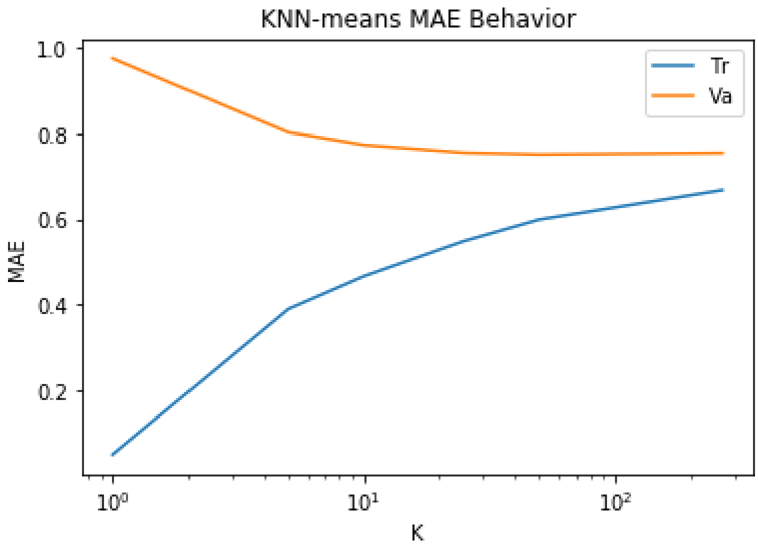
I use surprise KNN for problem 5. I look into the tradiational KNN, KNN-Means(KNNM, subtract means), KNN using pearson correlation(KNNP) and KNN-Means with pearson correlation(KNNMP). I looked into how k affects the error of the methods. It seems choosing K=20 should be a good strategy in all of the cases. We observe the test error behavior as the following:
KNN: 0.7653 KNNM: 0.7427 KNNP: 0.7953 KNNMP:0.7367 This shows when using KNN here, subtract mean and use pearson correlation is better for prediction.

In [94]:

```
plt.figure
plt.xlabel('K')
plt.ylabel('MAE')
plt.title('KNN-means MAE Behavior')
plt.plot(K_nbh, te, label='Tr')
plt.plot(K_nbh, ve, label='Va')
plt.xscale('log')
plt.legend()
```

Out[94]:

```
<matplotlib.legend.Legend at 0x20e89fc7748>
```

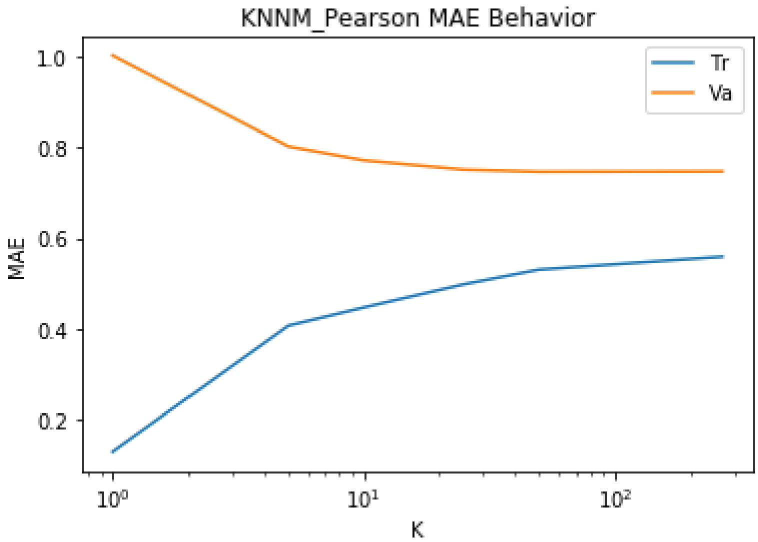


In [198]:

```
plt.figure
plt.xlabel('K')
plt.ylabel('MAE')
plt.title('KNNM_Pearson MAE Behavior')
plt.plot(K_nbh, tea, label='Tr')
plt.plot(K_nbh, vea, label='Va')
plt.xscale('log')
plt.legend()
```

Out[198]:

```
<matplotlib.legend.Legend at 0x20e8a34f6d8>
```

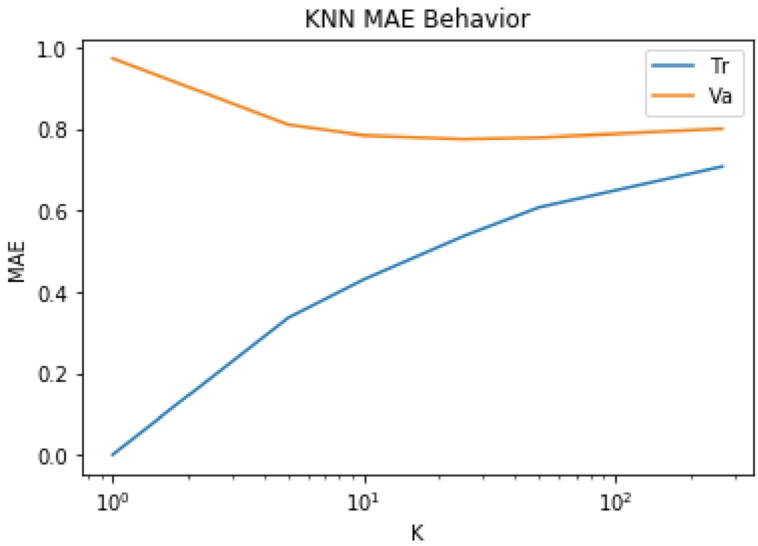


In [95]:

```
plt.figure
plt.xlabel('K')
plt.ylabel('MAE')
plt.title('KNN MAE Behavior')
plt.plot(K_nbh, teb, label='Tr')
plt.plot(K_nbh, veb, label='Va')
plt.xscale('log')
plt.legend()
```

Out[95]:

```
<matplotlib.legend.Legend at 0x20e8a36f780>
```

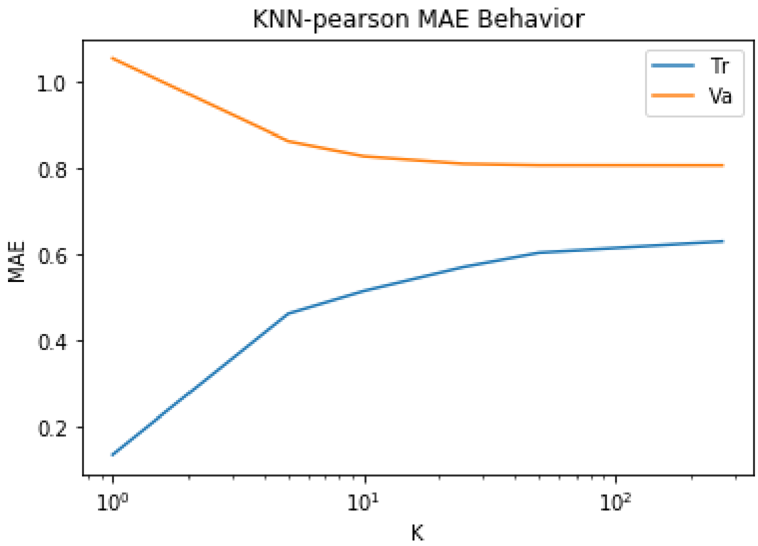


In [96]:

```
plt.figure
plt.xlabel('K')
plt.ylabel('MAE')
plt.title('KNN-pearson MAE Behavior')
plt.plot(K_nbh, tec, label='Tr')
plt.plot(K_nbh, vec, label='Va')
plt.xscale('log')
plt.legend()
```

Out[96]:

```
<matplotlib.legend.Legend at 0x20e880df5c0>
```



In [200]:

```
### Comming back to get result
train_set = train_set.build_full_trainset()
```

In [189]:

```
model=KNNM(K=20)
model.fit(train_set)

## Measure predictions on train set
print("Making predictions on training set (showing first 10):")
tr_pred = model.test(train_set.build_testset())
tr_mae = accuracy.mae(tr_pred)
tr_predicted_ratings_N = np.asarray([p.est for p in tr_pred], dtype=np.float64)
print(tr_predicted_ratings_N[:10])

## Measure predictions on test set
print("Making predictions on test set (showing first 10):")
te_pred = model.test(test_set)
#te_mae = accuracy.mae(te_pred) # should be NaN because no real labels on testset
te_predicted_ratings_N1 = np.asarray([p.est for p in te_pred], dtype=np.float64)
#print(te_predicted_ratings_N[:10])

print("n_factors %6d  tr_MAE %7.3f  test_MAE %7.3f" % (n_factors, tr_mae, te_mae))

#print("Making test set predictions in the original order")
tep1=[]
for row in test_df.values[:]:
        userid = row[0]
        itemid = row[1]
        rhat = model.predict(userid, itemid)
        #print("user %4d  item %4d  predicted rating % 8.3f" % (userid, itemid, rhat.est))
        tep1.append(rhat.est)
np.savetxt('predicted_ratings_test5M.txt', np.asarray(tep1))
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Making predictions on training set (showing first 10):
MAE:  0.5981
[2.62806394 3.51219776 2.90599123 3.98393553 2.89749724 2.91080894
 3.57222821 3.2620783  3.64707388 2.3300006 ]
Making predictions on test set (showing first 10):
MAE:  nan
n_factors      2  tr_MAE   0.598  test_MAE     nan
```

In [190]:

```
#train_set = train_set.build_full_trainset()
model=KNN(K=20)
model.fit(train_set)

## Measure predictions on train set
print("Making predictions on training set (showing first 10):")
tr_pred = model.test(train_set.build_testset())
tr_mae = accuracy.mae(tr_pred)
tr_predicted_ratings_N = np.asarray([p.est for p in tr_pred], dtype=np.float64)
print(tr_predicted_ratings_N[:10])

## Measure predictions on test set
print("Making predictions on test set (showing first 10):")
te_pred = model.test(test_set)
#te_mae = accuracy.mae(te_pred) # should be NaN because no real labels on testset
te_predicted_ratings_N2 = np.asarray([p.est for p in te_pred], dtype=np.float64)
#print(te_predicted_ratings_N2[:10])

print("n_factors %6d  tr_MAE %7.3f  test_MAE %7.3f" % (n_factors, tr_mae, te_mae))


#print("Making test set predictions in the original order")
tep2=[]
for row in test_df.values[:]:
        userid = row[0]
        itemid = row[1]
        rhat = model.predict(userid, itemid)
        #print("user %4d  item %4d  predicted rating % 8.3f" % (userid, itemid, rhat.est))
        tep2.append(rhat.est)
np.savetxt('predicted_ratings_test5.txt', np.asarray(tep2))
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Making predictions on training set (showing first 10):
MAE:  0.6015
[2.47094827 3.91616195 3.00816135 4.30859069 3.07263422 3.1585108
 3.73761608 3.51404379 3.84158116 2.40704005]
Making predictions on test set (showing first 10):
MAE:  nan
n_factors      2  tr_MAE   0.602  test_MAE     nan
```

In [191]:

```
model=KNN(K=20, sim_options={'name': 'pearson'})
model.fit(train_set)

## Measure predictions on train set
print("Making predictions on training set (showing first 10):")
tr_pred = model.test(train_set.build_testset())
tr_mae = accuracy.mae(tr_pred)
tr_predicted_ratings_N = np.asarray([p.est for p in tr_pred], dtype=np.float64)
print(tr_predicted_ratings_N[:10])

## Measure predictions on test set
print("Making predictions on test set (showing first 10):")
te_pred = model.test(test_set)
#te_mae = accuracy.mae(te_pred) # should be NaN because no real labels on testset
te_predicted_ratings_N = np.asarray([p.est for p in te_pred], dtype=np.float64)
print(te_predicted_ratings_N[:10])

print("n_factors %6d  tr_MAE %7.3f  test_MAE %7.3f" % (n_factors, tr_mae, te_mae))

#print("Making test set predictions in the original order")
tep3=[]
for row in test_df.values[:]:
        userid = row[0]
        itemid = row[1]
        rhat = model.predict(userid, itemid)
        #print("user %4d  item %4d  predicted rating % 8.3f" % (userid, itemid, rhat.est))
        tep3.append(rhat.est)
np.savetxt('predicted_ratings_test5p.txt', np.asarray(tep3))
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Making predictions on training set (showing first 10):
MAE:  0.6067
[2.46275582 4.12447317 3.10790543 4.44937415 3.13762163 3.51988639
 3.83512304 3.43602947 3.96805273 2.54442012]
Making predictions on test set (showing first 10):
MAE:  nan
[3.81880269 2.38702228 2.33748343 3.77718842 3.93141071 3.53435695
 3.0212087  3.80168179 3.0288308  3.69002411]
n_factors      2  tr_MAE   0.607  test_MAE     nan
```

In [201]:

```
model=KNNM(K=20, sim_options={'name': 'pearson'})
model.fit(train_set)

## Measure predictions on train set
print("Making predictions on training set (showing first 10):")
tr_pred = model.test(train_set.build_testset())
tr_mae = accuracy.mae(tr_pred)
tr_predicted_ratings_N = np.asarray([p.est for p in tr_pred], dtype=np.float64)
print(tr_predicted_ratings_N[:10])

## Measure predictions on test set
print("Making predictions on test set (showing first 10):")
te_pred = model.test(test_set)
#te_mae = accuracy.mae(te_pred) # should be NaN because no real labels on testset
te_predicted_ratings_N = np.asarray([p.est for p in te_pred], dtype=np.float64)
print(te_predicted_ratings_N[:10])

print("n_factors %6d  tr_MAE %7.3f  test_MAE %7.3f" % (n_factors, tr_mae, te_mae))

#print("Making test set predictions in the original order")
tep4=[]
for row in test_df.values[:]:
        userid = row[0]
        itemid = row[1]
        rhat = model.predict(userid, itemid)
        #print("user %4d  item %4d  predicted rating % 8.3f" % (userid, itemid, rhat.est))
        tep4.append(rhat.est)
np.savetxt('predicted_ratings_test5Mp.txt', np.asarray(tep4))
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Making predictions on training set (showing first 10):
MAE:  0.5375
[2.52974265 3.7955393  3.02586446 4.18648324 2.9166546  3.13912557
 3.62204029 3.24213867 3.80484038 2.38314212]
Making predictions on test set (showing first 10):
[3.5294804 3.5294804 3.5294804 3.5294804 3.5294804 3.5294804 3.5294804
 3.5294804 3.5294804 3.5294804]
n_factors      2  tr_MAE   0.538  test_MAE     nan
```

```
I use surprise KNN for problem 5. I look into the tradiational KNN, KNN-Means(KNNM,
subtract means), KNN using pearson correlation(KNNP) and KNN-Means with pearson
correlation(KNNMP). I looked into how k affects the error of the methods. It seems
choosing K=20 should be a good strategy in all of the cases. We observe the test error
behavior as the following:
KNN:  0.7653
KNNM: 0.7427
KNNP: 0.7953
KNNMP:0.7367
This shows when using KNN here, subtract mean and use pearson correlation is better for
prediction.
```

In [ ]: