Affective-Analysis-med2017

November 27, 2017

0.1 Valence-Arousal Prediction Audio and Visual Features

The Mediaeval 2017 Emotional Impact of Movies Task includes the data in the emotional domain (valence - arousal regression) and fear (binary classification). We have displayed the valence and arousal of all the movies in the dataset. Also the time of the movie where fear is present is specified with the value of the second. According to the Russell's circumplex model we were expectinf the "FEAR" to be appeared in the negative vallence, positive arousal part of the circumflex. However in some movies, we can see that frightment exists in positive valence with negative arousal also.

```
In [2]: import pandas as pd
        from pandas import DataFrame, Series

        import matplotlib.pyplot as plt
        import matplotlib.colors as colors


        import matplotlib
        matplotlib.style.use('ggplot')

        %matplotlib inline

        import numpy as np
        import pylab as pl
        import re, fileinput
        import os.path
        import glob
        import pickle
        import sys
```

```
In [3]: import numpy as np
        print(np.__version__)
        print(np.__path__)
```

```
1.11.3
['/home/yt/anaconda2/lib/python2.7/site-packages/numpy']
```

```
In [4]: from sklearn.metrics import accuracy_score
        from sklearn import preprocessing
        from sklearn import metrics
        from sklearn.svm import LinearSVC
        from sklearn.svm import SVC
        from sklearn import svm
        from sklearn.svm import SVR
        from sklearn.metrics import mean_squared_error
```

```
from sklearn.model_selection import train_test_split, cross_val_score
import scipy
from scipy.stats import pearsonr
```

In [5]:
```
#Dev data
#movieNames = ['After_The_Rain','Attitude_Matters','Barely_legal_stories','Between_Viewings','B

pathcontinuous = "/home/yt/Desktop/cvpr2014/repro/mediaeval/data/dataset/ContinuousLIRIS-ACCEDE/
continuousAnnotationsFolder = pathcontinuous +'continuous-annotations/'
devdatacontinous =  pathcontinuous + "continuous-movies/"
pathcontfeatures = "/home/yt/Desktop/cvpr2014/repro/mediaeval/data/dataset/Continuous/features-c

datahome = '/home/yt/Desktop/mediaeval2017'

med2017visualFeaturesfolder='/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-DevSet-Visual_features/M
med2017audiofolder='/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-DevSet-Audio_features/MEDIAEVAL17
med2017annotationsFolder = '/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-DevSet-Valence_Arousal-an
med2017fearFolder = '/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-DevSet-Fear-annotations/MEDIAEVA
med2017dataFolder = devdatacontinous

### Test Data

med2017visualFeaturesfolderTest='/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-TestSet-Visual_featu
med2017audiofolderTest = '/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-TestSet-Audio_features/MEDI
med2017datafolderTest = '/home/yt/Desktop/mediaeval2017/MEDIAEVAL17-TestSet-Data/MEDIAEVAL17-Tes

med2017testfeatures = "/home/yt/Desktop/cvpr2014/repro/mediaeval/data/dataset/Continuous/feature
```

In [ ]:

In [ ]:

In [9]:
```
files = glob.glob(med2017datafolderTest+'*')
testmovieNames =[ f.split('/')[-1].replace('.mp4','') for f in sorted(files) ]
```

In [10]:
```
files = glob.glob(med2017dataFolder+'*')
movieNames =[ f.split('/')[-1].replace('.mp4','') for f in sorted(files) ]
```

In [11]:
```
movieNames,testmovieNames
```

Out[11]:
```
(['After_The_Rain',
  'Attitude_Matters',
  'Barely_legal_stories',
  'Between_Viewings',
  'Big_Buck_Bunny',
  'Chatter',
  'Cloudland',
  'Damaged_Kung_Fu',
  'Decay',
  'Elephant_s_Dream',
  'First_Bite',
  'Full_Service',
  'Islands',
  'Lesson_Learned',
  'Norm',
  'Nuclear_Family',
```

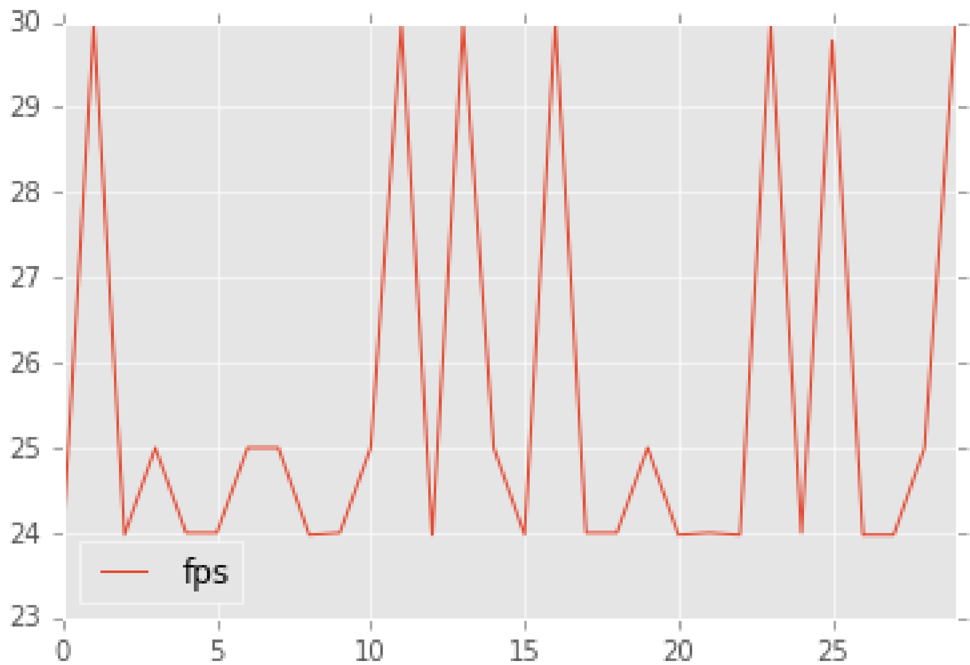
```
          'On_time',
          'Origami',
          'Parafundit',
          'Payload',
          'Riding_The_Rails',
          'Sintel',
          'Spaceman',
          'Superhero',
          'Tears_of_Steel',
          'The_room_of_franz_kafka',
          'The_secret_number',
          'To_Claire_From_Sonny',
          'Wanted',
          'You_Again'],
         ['MEDIAEVAL17_00',
          'MEDIAEVAL17_01',
          'MEDIAEVAL17_02',
          'MEDIAEVAL17_03',
          'MEDIAEVAL17_04',
          'MEDIAEVAL17_05',
          'MEDIAEVAL17_06',
          'MEDIAEVAL17_07',
          'MEDIAEVAL17_08',
          'MEDIAEVAL17_09',
          'MEDIAEVAL17_10',
          'MEDIAEVAL17_11',
          'MEDIAEVAL17_12',
          'MEDIAEVAL17_13'])
```

```
In [12]: fpsMovie = [['After_The_Rain',23.976],
                     ['Attitude_Matters',29.97],
                     ['Barely_legal_stories',23.976],
                     ['Between_Viewings',25],
                     ['Big_Buck_Bunny',24],
                     ['Chatter',24],
                         ['Cloudland',25],
                         ['Damaged_Kung_Fu',25],
                         ['Decay',23.976],
                         ['Elephant_s_Dream',24],
                         ['First_Bite',25],
                         ['Full_Service',29.97],
                         ['Islands',23.976],
                         ['Lesson_Learned',29.97],
                         ['Norm',25],
                         ['Nuclear_Family',23.976],
                         ['On_time',30],
                         ['Origami',24],
                         ['Parafundit',24],
                         ['Payload',25],
                         ['Riding_The_Rails',23.976],
                         ['Sintel',24],
                         ['Spaceman',23.976],
                         ['Superhero',29.97],
                         ['Tears_of_Steel',24],
```

```
                    ['The_room_of_franz_kafka',29.786],
                    ['The_secret_number',23.976],
                    ['To_Claire_From_Sonny',23.976],
                    ['Wanted',25],
                    ['You_Again',29.97]]

contmoviesfps = pd.DataFrame(fpsMovie,columns=['name','fps'])
#contmoviesfps.set_index('name', inplace=True)
#contmoviesfps.index.name = None
#contmoviesfps['After_The_Rain']
```

In [13]:
```
contmoviesfps.plot.line()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f75a0cf6f10>



In [41]:
```
contmoviesfps['f'] = np.round(contmoviesfps['fps'])
contmoviesfps
```

Out[41]:

	name	fps	f
0	After_The_Rain	23.976	24.0
1	Attitude_Matters	29.970	30.0
2	Barely_legal_stories	23.976	24.0
3	Between_Viewings	25.000	25.0
4	Big_Buck_Bunny	24.000	24.0
5	Chatter	24.000	24.0
6	Cloudland	25.000	25.0
7	Damaged_Kung_Fu	25.000	25.0
8	Decay	23.976	24.0
9	Elephant_s_Dream	24.000	24.0

10	First_Bite	25.000	25.0
11	Full_Service	29.970	30.0
12	Islands	23.976	24.0
13	Lesson_Learned	29.970	30.0
14	Norm	25.000	25.0
15	Nuclear_Family	23.976	24.0
16	On_time	30.000	30.0
17	Origami	24.000	24.0
18	Parafundit	24.000	24.0
19	Payload	25.000	25.0
20	Riding_The_Rails	23.976	24.0
21	Sintel	24.000	24.0
22	Spaceman	23.976	24.0
23	Superhero	29.970	30.0
24	Tears_of_Steel	24.000	24.0
25	The_room_of_franz_kafka	29.786	30.0
26	The_secret_number	23.976	24.0
27	To_Claire_From_Sonny	23.976	24.0
28	Wanted	25.000	25.0
29	You_Again	29.970	30.0

In [43]:
```
def getfps(movname):
    return contmoviesfps[ contmoviesfps.name == movname ]['f']
```

In [44]:
```
print contmoviesfps[ contmoviesfps.name == 'You_Again' ]['f']
print getfps('You_Again')
```

```
29    30.0
Name: f, dtype: float64
29    30.0
Name: f, dtype: float64
```

In [18]:
```
movgroups_wodecay = {
    0:['You_Again','Damaged_Kung_Fu','The_secret_number','Spaceman'],
    1:['Cloudland','Origami','Riding_The_Rails','Tears_of_Steel','Sintel'],
    2:['On_time','Elephant_s_Dream','Norm','Big_Buck_Bunny','Chatter','Full_Service'],
    3:['Islands','To_Claire_From_Sonny','Nuclear_Family','After_The_Rain','Parafundit'],
    4:['The_room_of_franz_kafka','Attitude_Matters','Lesson_Learned','Superhero'],
    5:['First_Bite','Wanted','Between_Viewings','Barely_legal_stories','Payload']
}

movgroups = {
    0:['You_Again','Damaged_Kung_Fu','The_secret_number','Spaceman'],
    1:['Cloudland','Origami','Riding_The_Rails','Tears_of_Steel','Sintel'],
    2:['On_time','Elephant_s_Dream','Norm','Big_Buck_Bunny','Chatter','Full_Service'],
    3:['Islands','To_Claire_From_Sonny','Nuclear_Family','After_The_Rain','Parafundit'],
    4:['The_room_of_franz_kafka','Attitude_Matters','Lesson_Learned','Superhero'],
    5:['First_Bite','Wanted','Between_Viewings','Barely_legal_stories','Payload'],
    6:['Decay']
}

mov2groups = {
    0:['Decay'],
    1:['You_Again','Damaged_Kung_Fu','The_secret_number','Spaceman'],
    2:['Cloudland','Origami','Riding_The_Rails','Tears_of_Steel','Sintel'],
```

```
        3:['On_time','Elephant_s_Dream','Norm','Big_Buck_Bunny','Chatter','Full_Service'],
        4:['Islands','To_Claire_From_Sonny','Nuclear_Family','After_The_Rain','Parafundit'],
        5:['The_room_of_franz_kafka','Attitude_Matters','Lesson_Learned','Superhero'],
        6:['First_Bite','Wanted','Between_Viewings','Barely_legal_stories','Payload'],
    }


    def gettraintestmovielist(mlist,groups=movgroups):
        testlist = groups[mlist]
        trainlist =[]
        for idx, group in enumerate(groups):
            if idx != mlist:
                for g in groups[idx]:
                    trainlist.append(g)
        return trainlist, testlist

    def gettraintest2movielist(foldno,groups=mov2groups):
        if foldno==1:
            mlist=[1,2]
        elif foldno==2:
            mlist=[3,4]
        elif foldno==3:
            mlist=[5,6]
        elif foldno==4:
            mlist=[2,3]
        elif foldno==5:
            mlist=[4,5]
        else:
            mlist=[]

        testlist = []
        for i in mlist:
            for f in groups[i]:
                testlist.append(f)

        trainlist =[]
        for idx, group in enumerate(groups):
            for f in groups[idx]:
                if f not in testlist:
                    trainlist.append(f)

        return trainlist, testlist
```

```
In [19]: gettraintest2movielist(4)
```

```
Out[19]: (['Decay',
           'You_Again',
           'Damaged_Kung_Fu',
           'The_secret_number',
           'Spaceman',
           'Islands',
           'To_Claire_From_Sonny',
           'Nuclear_Family',
           'After_The_Rain',
           'Parafundit',
```

```
           'The_room_of_franz_kafka',
           'Attitude_Matters',
           'Lesson_Learned',
           'Superhero',
           'First_Bite',
           'Wanted',
           'Between_Viewings',
           'Barely_legal_stories',
           'Payload'],
          ['Cloudland',
           'Origami',
           'Riding_The_Rails',
           'Tears_of_Steel',
           'Sintel',
           'On_time',
           'Elephant_s_Dream',
           'Norm',
           'Big_Buck_Bunny',
           'Chatter',
           'Full_Service'])
```

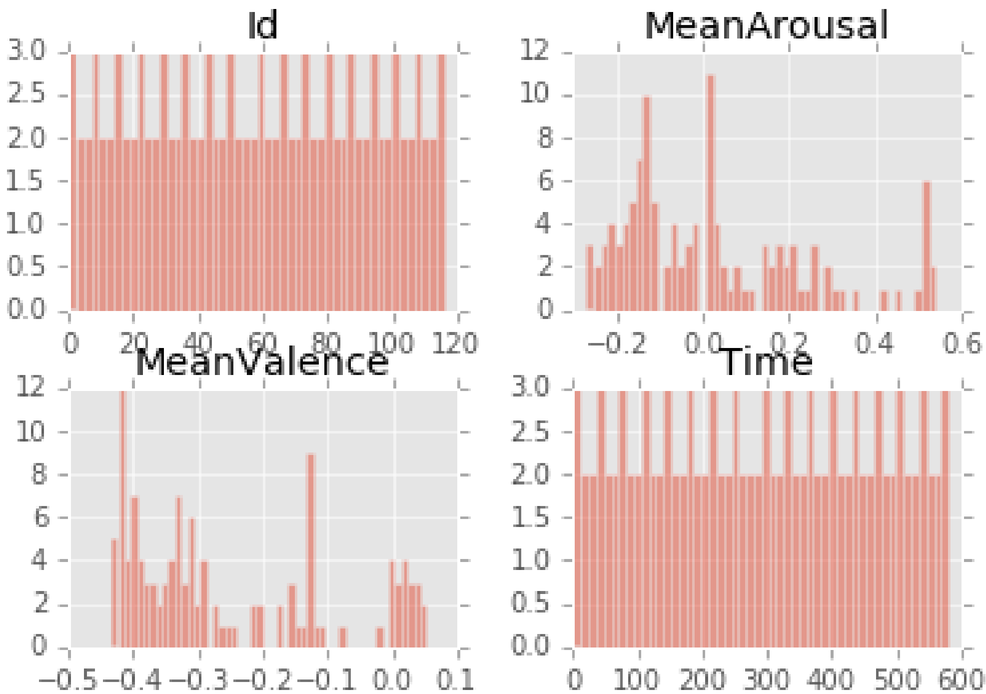
0.2 Valence - Arosal Annotations

Thank you for downloading LIRIS-ACCEDE dataset. This file contains valence/arousal annotations for the LIRIS-ACCEDE continuous part that is used for the first subtask of the MEDIAEVAL 2017 Emotional Impact of Movies task. For each of the 30 movies, consecutive ten seconds-segments sliding over the whole movie with a shift of 5 seconds are considered and provided with valence and arousal annotations. Each txt file contains 4 columns separated by tabulations. The first column is the segment id, starting from 0, the second column is the starting time of the segment in the movie and the third and fourth columns are respectively the valence and arousal values for this segment.

```
In [20]: def getAnnotationDf(movname,folder=med2017annotationsFolder):
             filename = os.path.join(folder, movname + '-MEDIAEVAL2017-valence_arousal.txt')
             annotation = np.genfromtxt(filename, names=True, delimiter='\t', dtype=None)
             df = pd.DataFrame(annotation)
             return df
```

```
In [22]: df = getAnnotationDf(movieNames[0])
         df.hist(alpha=0.5,bins=50)
```

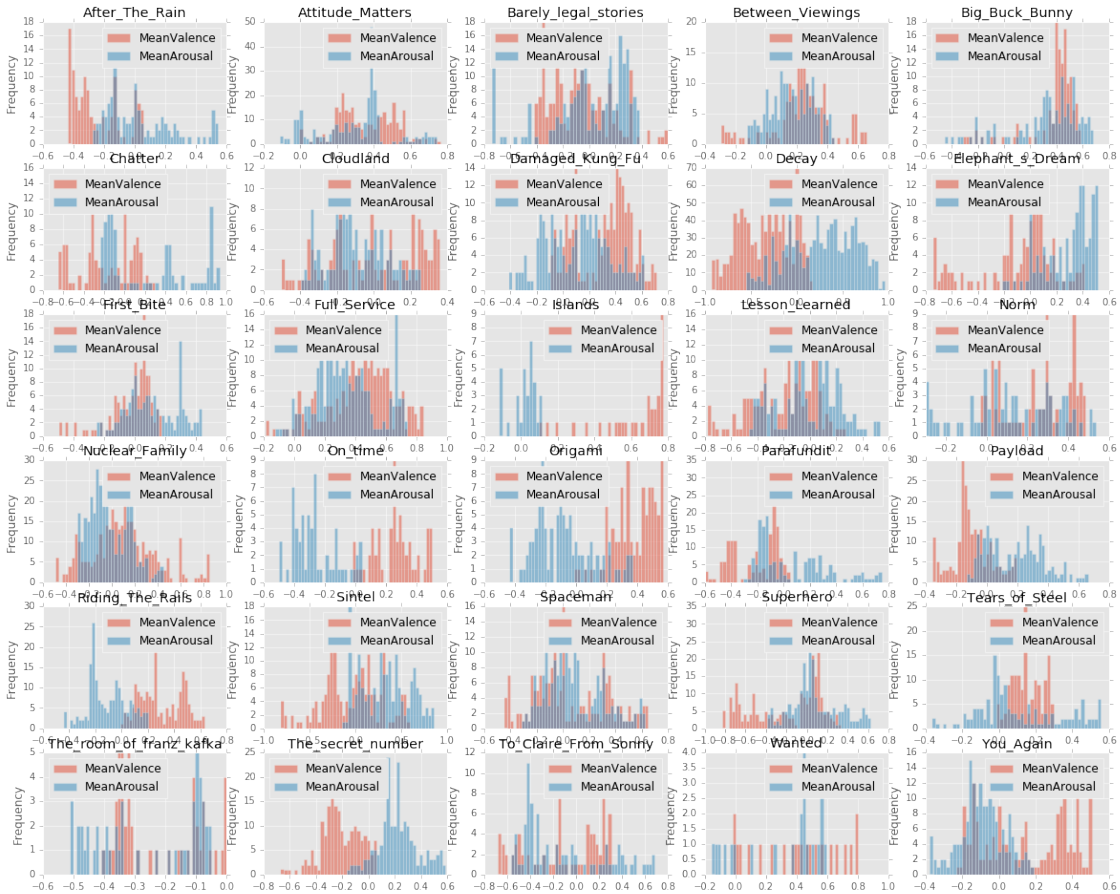
```
Out[22]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f759d3de9d0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f759ced3510>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x7f759ce4ea10>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x7f759ce2e610>]], dtype=object)
```




```
In [23]: #df.head()
         #df.describe()
```

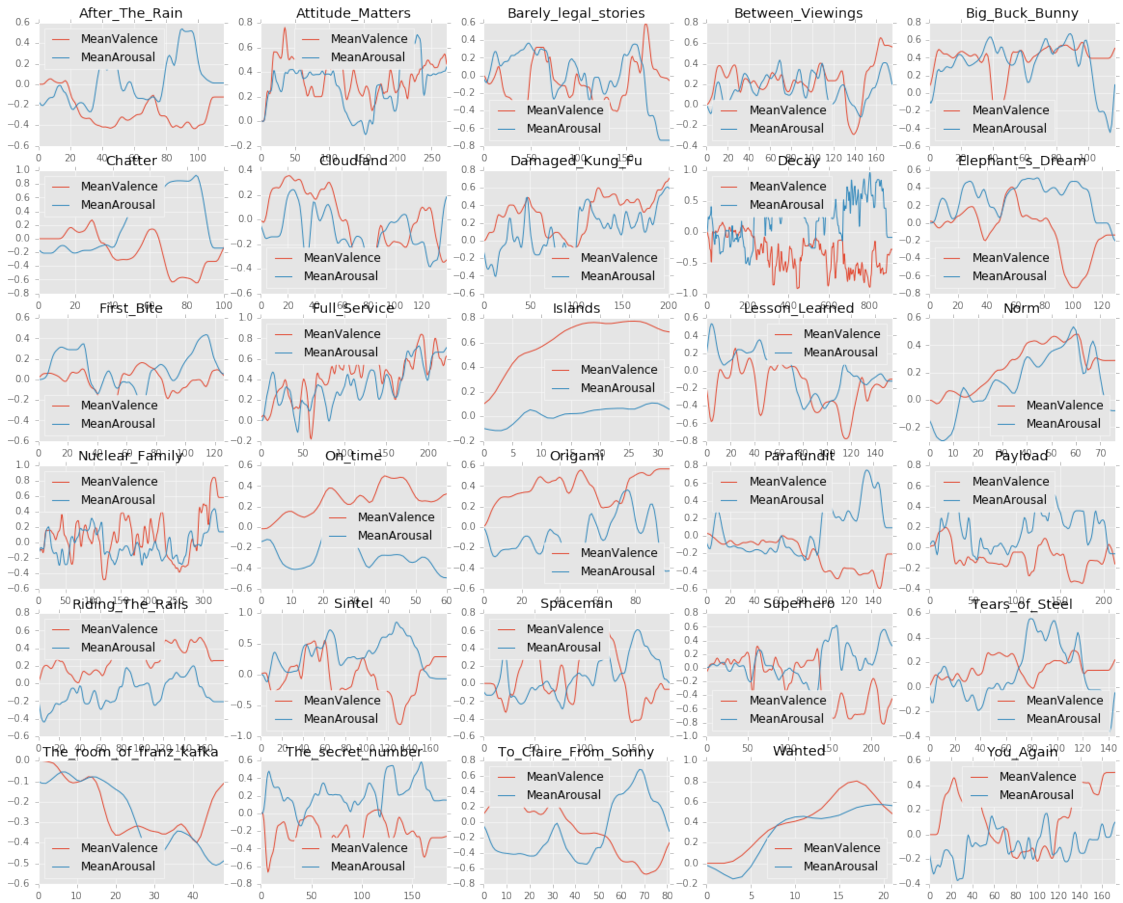
0.3 Valence, Arousal histogram plots for Dev-Set

```
In [24]: fix, axes = plt.subplots(figsize=(20,16))
         for ii, mov in enumerate(movieNames):
             if (ii+1 > 30):
                 plt.subplot(6,5,ii)
             else :
                 plt.subplot(6,5,ii+1)
             df = getAnnotationDf(mov)
             df[['MeanValence','MeanArousal']].plot.hist(ax=plt.gca(),title=mov,alpha=0.5,bins=50)
```



0.4 Valence , Arousal plots for Dev-Set

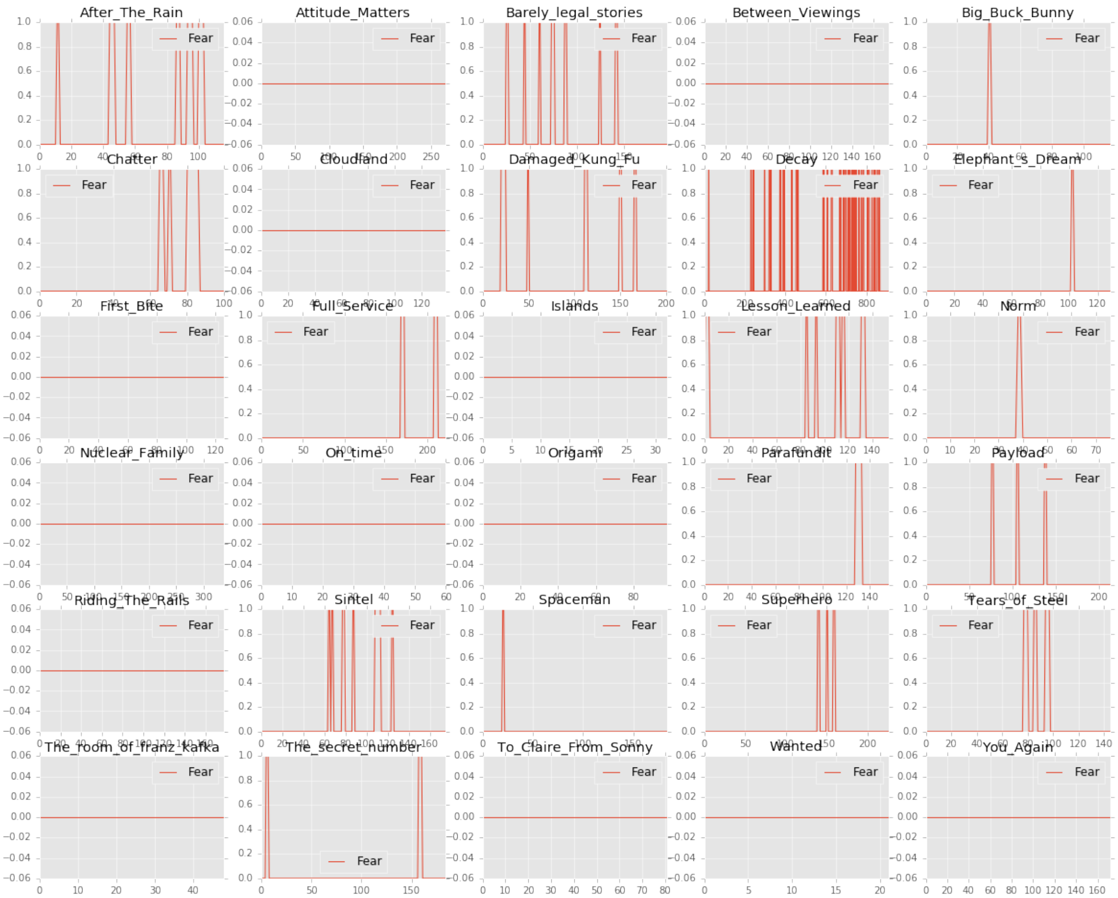
In [25]:
```
fix, axes = plt.subplots(figsize=(20,16))
for ii, mov in enumerate(movieNames):
    plt.subplot(6,5,ii+1)
    df = getAnnotationDf(mov)
    df[['MeanValence','MeanArousal']].plot(ax=plt.gca(),title=mov)
    #.hist(alpha=0.5,bins=50)
```



0.5 Fear Annotations

```
In [26]: def getFearDf(movname):
             filename = os.path.join(med2017fearFolder, movname + '-MEDIAEVAL2017-fear.txt')
             annotation = np.genfromtxt(filename, names=True, delimiter='\t', dtype=None)
             df = pd.DataFrame(annotation)
             return df
```

```
In [27]: fix, axes = plt.subplots(figsize=(20,16))
         for ii, mov in enumerate(movieNames):
             plt.subplot(6,5,ii+1)
             df = getFearDf(mov)
             df[['Fear']].plot(ax=plt.gca(),title=mov)
```



0.6 Audio Features

```
In [28]: def getAudioDf(moviename,folder=med2017audiofolder):
             if 'TestSet' in folder:
                 files = glob.glob(folder+moviename+'/audio_features/*.csv')
             else:
                 files = glob.glob(folder+moviename+'/*.csv')
             files = sorted(files)
             files
             alist = []
             for fname in files:
                 f=open(fname,'r')
                 h = []
                 for l in f :
                     if '@attribute' in l:
                         h.append(l.split()[1])
                     elif l == '\n':
                         l
                     elif l[0] =='@':
                         l
                     else:
                         alist.append(map(float,l.split(',')[1:])) #first attribute is string ,skipped
```

```
            f.close()

        return pd.DataFrame(alist,columns=h[1:])
```

0.7 Visual Features

```
In [29]: visual_feat = ['acc', 'cedd', 'cl', 'eh', 'fc6', 'fcth',
                        'gabor', 'jcd', 'lbp', 'sc', 'tamura'   ]
         visual_feat_wofc16 = ['acc', 'cedd', 'cl', 'eh', 'fcth',
                        'gabor', 'jcd', 'lbp', 'sc', 'tamura'   ]

In [30]: def getVisFeatureDf(moviename,typename,folder=med2017visualFeaturesfolder):
             files = glob.glob(folder+moviename+'/'+typename+'/*.txt')
             files = sorted(files)
             alist = []
             for fname in files:
                 f=open(fname,'r')
                 for l in f:
                     alist.append(map(float,l.split(',')))
                 f.close()
             return pd.DataFrame(alist)

         def getAvgVisFeatureDf(moviename,typename,folder=med2017visualFeaturesfolder):
             df = getVisFeatureDf(moviename,typename,folder)
             dfwindow = df.rolling(10).mean()[9::5] ############### start with 9
             dfwindow.reset_index(inplace=True)
             dfwindow.drop('index',axis=1,inplace=True)
             return dfwindow

         def getAvgVisFeatListDf(moviename,featlist,folder=med2017visualFeaturesfolder):
             df = getVisFeatureDf(moviename,featlist[0],folder)
             for feat in featlist[1:]:
                 tdf = getVisFeatureDf(moviename,feat,folder)
                 df = pd.concat([df,tdf],axis=1)

             dfwindow = df.rolling(10).mean()[9::5] ############### start with 9
             dfwindow.reset_index(inplace=True)
             dfwindow.drop('index',axis=1,inplace=True)
             dfwindow.columns=list(range(len(dfwindow.columns)))
             return dfwindow

In [31]: sum([len(getAnnotationDf(m)) for m in movieNames ])

Out[31]: 5274

In [30]: sum([len(getAudioDf(m)) for m in movieNames ])

Out[30]: 5264

In [31]: sum([len(getVisFeatureDf(m,'cl')) for m in movieNames ])

Out[31]: 26589

In [32]: sum([len(getAvgVisFeatureDf(m,'cl')) for m in movieNames ])

Out[32]: 5271
```

```
In [61]: df = getVisFeatureDf(movieNames[0],'cl')
         #df = getAvgVisFeatureDf(movieNames[0],'cl')
         #df = getAvgVisFeatListDf(movieNames[0],['cl','eh'])
         #df.hist()
         df.head(10)
```

```
Out[61]:      0     1     2     3     4     5     6     7     8     9  ...    23  \
         0   2.0  16.0  16.0  16.0  16.0  16.0  16.0  16.0  16.0  16.0  ...  16.0
         1   2.0  15.0  15.0  16.0  16.0  16.0  15.0  15.0  15.0  16.0  ...  16.0
         2  15.0  16.0  14.0  14.0  16.0  16.0  16.0  15.0  16.0  15.0  ...  16.0
         3   6.0  17.0  15.0   8.0  16.0  12.0  18.0  16.0  16.0  16.0  ...  15.0
         4   3.0  16.0  15.0  11.0  16.0  12.0  16.0  16.0  15.0  16.0  ...  16.0
         5   4.0  14.0  16.0  13.0  15.0  13.0  16.0  15.0  16.0  15.0  ...  15.0
         6   3.0  14.0  16.0  13.0  15.0  12.0  16.0  15.0  16.0  15.0  ...  16.0
         7   4.0  14.0  16.0  13.0  15.0  12.0  16.0  15.0  16.0  15.0  ...  16.0
         8   4.0  14.0  16.0  13.0  15.0  12.0  16.0  15.0  16.0  15.0  ...  16.0
         9   3.0  15.0  16.0  13.0  15.0  12.0  16.0  15.0  16.0  15.0  ...  16.0

              24    25    26    27    28    29    30    31    32
         0  16.0  16.0  16.0  32.0  16.0  16.0  16.0  16.0  16.0
         1  16.0  16.0  16.0  31.0  16.0  16.0  16.0  16.0  16.0
         2  15.0  16.0  16.0  21.0  15.0  16.0  16.0  16.0  16.0
         3  15.0  16.0  15.0  27.0  15.0  16.0  17.0  15.0  17.0
         4  15.0  16.0  15.0  30.0  16.0  16.0  17.0  16.0  16.0
         5  16.0  15.0  15.0  29.0  16.0  16.0  16.0  16.0  16.0
         6  16.0  16.0  16.0  31.0  16.0  16.0  16.0  16.0  16.0
         7  16.0  16.0  16.0  31.0  16.0  16.0  16.0  16.0  16.0
         8  16.0  16.0  16.0  31.0  16.0  16.0  16.0  16.0  16.0
         9  16.0  16.0  16.0  31.0  16.0  16.0  16.0  16.0  16.0

         [10 rows x 33 columns]
```

```
In [18]: #df = getAvgVisFeatListDf(movieNames[0],['fc6'])
         #df = getVisFeatureDf(movieNames[0],'fc6')
         #df.describe()
```

0.8 Low Level Cinematographic Features

fps değerlerine göre, feature çıkarma key frame seçme ve averaging tekrar yapılacak.

```
In [81]: def getLowFeatureDf(movname):
             fname = movname +'.mp4continous_features.txt'
             df = pd.DataFrame(np.genfromtxt( os.path.join(pathcontfeatures,fname)))
             df.columns = ['time','framemean','huemean','satmean','valmean', 'redmean','greenmean','blu
             return df

         def getLowFeature10SecDf(movname):
             pdf = getLowFeatureDf(movname)
             fps = getfps(movname)
             dfwindow = pdf.rolling(10).mean()[9::5]
             dfwindow.reset_index(inplace=True)
             dfwindow.drop('index',axis=1,inplace=True)
             dfwindow.drop('time',axis=1,inplace=True)
             return dfwindow
```

```
def getMovieListLowFeatFearDf(movieNames):
    X = getLowFeature10SecDf(movieNames[0])
    y = getFearDf(movieNames[0]).Fear[:len(X)]

    for mov in movieNames[1:]:
        tX=getLowFeatureDf(mov)
        ty=getFearDf(mov).Fear[:len(tX)]
        X = X.append(tX)
        y = y.append(ty)
        if (X.shape != y.shape):
            print mov, X.shape, y.shape
    return X,y
```

In [70]:
```
#getLowFeatureDf(movieNames[1]).head(10)[2::2]
```

In [71]:
```
#print getLowFeatureDf(movieNames[1]).head(10)
#print getLowFeatureDf(movieNames[1]).head(10).mean()
print getLowFeature10SecDf(movieNames[1]).head(10)
```

```
Attitude_Matters.mp4continous_features.txt
   framemean   huemean     satmean     valmean    redmean  greenmean  \
0       11.0  16.266390  108.533860   24.615390   5.417613   5.983095
1        9.5  10.111689   65.248124   15.309475   7.196943   7.676144
2       14.1   7.779963   10.971608   15.279160  14.210921  14.116898
3       54.6  31.491614   36.296284   59.702565  52.819005  54.486605
4       79.8  47.335400   74.969860   89.290460  75.284470  79.210160
5       70.7  46.443380   82.062840   80.176950  66.029620  69.923720
6       73.1  49.213990   78.506800   82.417040  69.214910  72.537860
7       75.4  45.503130   70.678890   83.767210  70.378780  75.304340
8       79.0  31.727480   83.240130   92.335810  65.965150  76.843300
9       84.5  24.291420  101.637440  103.255080  66.305130  79.824300

     bluemean  lummean     motion
0   24.605360      0.7  26.035474
1   15.304023      0.1   4.688359
2   14.986898      6.0  10.187818
3   57.096935     50.0  15.618866
4   84.692030     71.9  14.010382
5   75.444710     56.9   8.726338
6   77.423400     59.0   3.330288
7   79.491460     51.0  26.182445
8   90.024230     51.4  41.366657
9  102.228710     65.9  24.806420
```

0.9 Train and Test set creation

In [85]:
```
def getFeatureswFearDf(movieNames,featlist=visual_feat_wofc16):
    Xv = getAvgVisFeatListDf(movieNames[0],featlist)
    Xa = getAudioDf(movieNames[0])
    Xd = getAvgVisFeatListDf(movieNames[0],['fc6'])
    Xl = getLowFeature10SecDf(movieNames[0])
    y = getFearDf(movieNames[0])[['Fear']]

    mlen = min(len(Xv),len(Xa), len(Xd), len(Xl),len(y))
```

```
        Xv = Xv[:mlen]
        Xa = Xa[:mlen]
        Xd = Xd[:mlen]
        Xl = Xl[:mlen]
        y  = y[:mlen]

        for mov in movieNames[1:]:
            tXv = getAvgVisFeatListDf(mov,featlist)
            tXa = getAudioDf(mov)
            tXd = getAvgVisFeatListDf(mov,['fc6'])
            tXl = getLowFeature10SecDf(mov)
            ty = getFearDf(mov)[['Fear']]

            mlen = min(len(tXv),len(tXa),len(tXd),len(ty))
            tXv = tXv[:mlen]
            tXa = tXa[:mlen]
            tXd = tXd[:mlen]
            tXl = tXl[:mlen]
            ty = ty[:mlen]

            Xv  = Xv.append(tXv)
            Xa  = Xa.append(tXa)
            Xd = Xd.append(tXd)
            Xl = Xl.append(tXl)
            y  = y.append(ty)

        return Xv,Xa,Xd,Xl,y
```

```
In [77]: def getFeatureswAnnotationsDf(movieNames,featlist=visual_feat_wofc16):
        Xv = getAvgVisFeatListDf(movieNames[0],featlist)
        Xa = getAudioDf(movieNames[0])
        Xd = getAvgVisFeatListDf(movieNames[0],['fc6'])
        Xl = getLowFeature10SecDf(movieNames[0])
        y = getAnnotationDf(movieNames[0])[['MeanValence','MeanArousal']]

        mlen = min(len(Xv),len(Xa), len(Xd), len(Xl),len(y))

        Xv = Xv[:mlen]
        Xa = Xa[:mlen]
        Xd = Xd[:mlen]
        Xl = Xl[:mlen]
        y = y[:mlen]

        for mov in movieNames[1:]:
            tXv = getAvgVisFeatListDf(mov,featlist)
            tXa = getAudioDf(mov)
            tXd = getAvgVisFeatListDf(mov,['fc6'])
            tXl = getLowFeature10SecDf(mov)
            ty = getAnnotationDf(mov)[['MeanValence','MeanArousal']]

            mlen = min(len(tXv),len(tXa),len(tXd),len(ty))
            tXv = tXv[:mlen]
            tXa = tXa[:mlen]
            tXd = tXd[:mlen]
```

```
        tXl = tXl[:mlen]
        ty = ty[:mlen]

        Xv  = Xv.append(tXv)
        Xa  = Xa.append(tXa)
        Xd = Xd.append(tXd)
        Xl = Xl.append(tXl)
        y  = y.append(ty)

    return Xv,Xa,Xd,Xl,y
```

In [73]:
```
def getMovListAudioVisFeatListwAnnotationsDf(movieNames,featlist):
    Xv = getAvgVisFeatureDf(movieNames[0],featlist[0])
    Xa = getAudioDf(movieNames[0])
    y = getAnnotationDf(movieNames[0])[['MeanValence','MeanArousal']]

    mlen = min(len(Xv),len(Xa),len(y))
    print(mlen)

    Xv = Xv[:mlen]
    Xa = Xa[:mlen]
    y = y[:mlen]

    for feattype in featlist[1:]:
        fXv = getAvgVisFeatureDf(movieNames[0],feattype)[:mlen]
        Xv = pd.concat( [Xv,fXv], axis=1 )

    for mov in movieNames[1:]:
        tXv = getAvgVisFeatureDf(mov,featlist[0])
        tXa = getAudioDf(mov)
        ty = getAnnotationDf(mov)[['MeanValence','MeanArousal']]

        mlen = min(len(tXv),len(tXa),len(ty))
        print(mlen)

        tXv = tXv[:mlen]
        tXa = tXa[:mlen]
        ty = ty[:mlen]

        for feattype in featlist[1:]:
            fXv = getAvgVisFeatureDf(mov,feattype)[:mlen]
            tXv = pd.concat( [tXv,fXv], axis=1 )

        Xv  = Xv.append(tXv)
        Xa  = Xa.append(tXa)
        y  = y.append(ty)

    return Xv,Xa,y
```

In [74]:
```
def df2mat(df):
    return df.as_matrix().reshape((len(df),))
```

1 Classification work

```
In [75]: from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.preprocessing import StandardScaler
         from sklearn.decomposition import PCA
         from sklearn.svm import SVC

         from sklearn.preprocessing import MinMaxScaler
         from sklearn.pipeline import Pipeline
         from sklearn.pipeline import make_pipeline
         from sklearn.grid_search import GridSearchCV
         from sklearn.pipeline import make_pipeline
         from sklearn.preprocessing import StandardScaler

         #from sklearn import cross_validation
         from sklearn import metrics   #Additional scklearn functions
         from sklearn.grid_search import GridSearchCV   #Perforing grid search

         import matplotlib.pylab as plt
         %matplotlib inline
         from matplotlib.pylab import rcParams

         def getGridCV(pipe,paramgirid,Xtrain,ytrain): # scoring ?
             grid = GridSearchCV(pipe, param_grid, cv=5,n_jobs=125)
             grid.fit(Xtrain,ytrain)

             return grid
```

```
/home/yt/anaconda2/lib/python2.7/site-packages/sklearn/cross_validation.py:44: DeprecationWarning: This
  "This module will be removed in 0.20.", DeprecationWarning)
/home/yt/anaconda2/lib/python2.7/site-packages/sklearn/grid_search.py:43: DeprecationWarning: This modul
  DeprecationWarning)
```

```
In [76]: def modelfit(alg, X, y, performCV=True, printFeatureImportance=True, cv_folds=5):
             #Fit the algorithm on the data
             alg.fit(X, y)

             #Predict training set:
             dtrain_predictions = alg.predict(X)
             dtrain_predprob = alg.predict_proba(X)[:,1]

             #Perform cross-validation:
             if performCV:
                 cv_score = cross_validation.cross_val_score(alg, X, y, cv=cv_folds, scoring='roc_auc')

             #Print model report:
             print("\nModel Report")
             print("Accuracy : %.4g" % metrics.accuracy_score(y.values, dtrain_predictions))
             print("AUC Score (Train): %f" % metrics.roc_auc_score(y, dtrain_predprob))

             if performCV:
                 print("CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mean(cv_sc

             #Print Feature Importance:
```

```
        if printFeatureImportance:
            feat_imp = pd.Series(alg.feature_importances_, X.columns).sort_values(ascending=False)
            feat_imp.plot(kind='bar', title='Feature Importances')
            plt.ylabel('Feature Importance Score')
```

In [ ]:
```
trainlist, testlist=gettraintest2movielist(1,mov2groups)  # index 1 olanları test , diğerlerini
tXv,tXa,tXd,ty = getFeatureswFearDf(trainlist)
print(tXv.shape,tXa.shape,tXd.shape,ty.shape)
testXv, testXa, testXd, testy = getFeatureswFearDf(testlist)
print(testXv.shape, testXa.shape,testXd.shape, testy.shape)
```

In [ ]:

In [ ]:
```
param_test3 = {'min_samples_split':range(1000,2100,200), 'min_samples_leaf':range(30,71,10)}

pipegrad = GradientBoostingClassifier(learning_rate=0.05,
                                n_estimators=60,max_depth=9,
                                max_features='sqrt', subsample=0.8,
                                random_state=10)

gsearch3 = GridSearchCV(pipegrad , param_grid = param_test3, scoring='roc_auc',n_jobs=4,iid=Fals
#gsearch3.fit(Xtraina,ytrain)
#gsearch3.grid_scores_, gsearch3.best_params_, gsearch3.best_score_
```

In [ ]:
```
pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVC())])

param_grid = [
    {'classifier': [SVC()], 'preprocessing': [StandardScaler()],
     'classifier__gamma': [0.001, 0.01, 1, 10],
     'classifier__C': [0.01, 1, 10,100]},
    {'classifier': [RandomForestClassifier],
     'preprocessing': [None],
     'classifier__n_estimators': [50,100,300]
     'classifier__max_features': [3,5,10]}]

grid = GridSearchCV(pipe, param_grid, cv=5)
#grid.fit(Xtraina,ytrain)

print("Best params:\n{}\n".format(grid.best_params_))
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
#grid.grid_scores_, grid.best_params_, grid.best_score_
```

In [ ]:
```
from joblib import Parallel, delayed

import multiprocessing
# what are your inputs, and what operation do you want to
# perform on each input. For example...

def trainPipe(ii,pipe,valorar, modality):

    #rows = []
    trainlist, testlist=gettraintest2movielist(ii,mov2groups)  # index 1 olanları test , diğerl
    tXv,tXa,tXd,ty = getFeatureswAnnotationsDf(trainlist)
    print(tXv.shape,tXa.shape,tXd.shape,ty.shape)
    testXv, testXa, testXd, testy = getFeatureswAnnotationsDf(testlist)
```

```
        print(testXv.shape, testXa.shape,testXd.shape, testy.shape)

        if modality == 'visual':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                tXv,ty[[valorar]],
                                                testXv, testy[[valorar]])
        else:
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                tXa,ty[[valorar]],
                                                testXa, testy[[valorar]])

        return ii,mse,prs,p1

    def processGroup(ii,pipe,valorar, modality):

        #rows = []
        trainlist, testlist=gettraintest2movielist(ii,mov2groups)  # index 1 olanları test , diğerl
        tXv,tXa,tXd,ty = getFeatureswAnnotationsDf(trainlist)
        print(tXv.shape,tXa.shape,tXd.shape,ty.shape)
        testXv, testXa, testXd, testy = getFeatureswAnnotationsDf(testlist)
        print(testXv.shape, testXa.shape,testXd.shape, testy.shape)

        if modality == 'visual':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                tXv,ty[[valorar]],
                                                testXv, testy[[valorar]])
        else:
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                tXa,ty[[valorar]],
                                                testXa, testy[[valorar]])

        return [ii,mse,prs]

    def crossgroups(pipe,valorar,modality):
        #inputs=range(len(movgroups))
        inputs=[1, 2, 3, 4, 5]
        num_cores = multiprocessing.cpu_count()
        results = Parallel(n_jobs=num_cores)(delayed(processGroup)(i,pipe,valorar,modality) for i in
        pipescores = pd.DataFrame(results,columns=['test-group','MSE','PCC'])

        return pipescores
```

In [ ]:

2 Regression work

In [ ]:
```
#trainlist, testlist=gettraintestmovielist(2,movgroups_wodecay)  # index 1 olanları test , diğe
#trainlist, testlist
#for ii in range(len(movgroups)):
#    trnlist, tstlist=gettraintestmovielist(ii)
```

In [92]:
```
%%time
allXv,allXa,allXd,ally = getFeatureswAnnotationsDf(movieNames)
print(allXv.shape,allXa.shape,allXd.shape,ally.shape)
```

```
((5264, 1271), (5264, 1583), (5264, 4096), (5264, 2))
CPU times: user 1min 41s, sys: 7.74 s, total: 1min 48s
Wall time: 2min 4s
```

In [78]:
```
trainlist, testlist=gettraintest2movielist(2)  # index 1 olanları test , diğerlerini train yap

tXv,tXa,tXd,tXl,ty = getFeatureswAnnotationsDf(trainlist)
print(tXv.shape,tXa.shape,tXd.shape,tXl.shape,ty.shape)
testXv, testXa, testXd, testXl, testy = getFeatureswAnnotationsDf(testlist)
print(testXv.shape, testXa.shape,testXd.shape,testXd.shape, testy.shape)

X_train, X_test, y_train, y_test = train_test_split(tXv, ty,test_size=0.2, random_state=0)
Xa_train, Xa_test, ya_train, ya_test = train_test_split(tXa, ty,test_size=0.2, random_state=0)
Xd_train, Xd_test, yd_train, yd_test = train_test_split(tXd, ty,test_size=0.2, random_state=0)
```

```
Decay.mp4continous_features.txt
You_Again.mp4continous_features.txt
Damaged_Kung_Fu.mp4continous_features.txt
The_secret_number.mp4continous_features.txt
Spaceman.mp4continous_features.txt
Cloudland.mp4continous_features.txt
Origami.mp4continous_features.txt
Riding_The_Rails.mp4continous_features.txt
Tears_of_Steel.mp4continous_features.txt
Sintel.mp4continous_features.txt
The_room_of_franz_kafka.mp4continous_features.txt
Attitude_Matters.mp4continous_features.txt
Lesson_Learned.mp4continous_features.txt
Superhero.mp4continous_features.txt
First_Bite.mp4continous_features.txt
Wanted.mp4continous_features.txt
Between_Viewings.mp4continous_features.txt
Barely_legal_stories.mp4continous_features.txt
Payload.mp4continous_features.txt
((3830, 1271), (3830, 1583), (3830, 4096), (3830, 9), (3830, 2))
On_time.mp4continous_features.txt
Elephant_s_Dream.mp4continous_features.txt
Norm.mp4continous_features.txt
Big_Buck_Bunny.mp4continous_features.txt
Chatter.mp4continous_features.txt
Full_Service.mp4continous_features.txt
Islands.mp4continous_features.txt
To_Claire_From_Sonny.mp4continous_features.txt
Nuclear_Family.mp4continous_features.txt
After_The_Rain.mp4continous_features.txt
Parafundit.mp4continous_features.txt
((1434, 1271), (1434, 1583), (1434, 4096), (1434, 4096), (1434, 2))
```

In [88]:
```
Xl_train, Xl_test, yl_train, yl_test = train_test_split(tXl, ty,test_size=0.2, random_state=0)
Xl_train.shape, Xl_test.shape, yl_train.shape, yl_test.shape
```

Out[88]:
```
((3064, 9), (766, 9), (3064, 2), (766, 2))
```

2.1 Linear Regression - Valence

In [89]:
```
%%time
from sklearn import  linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Create linear regression object
visual_regr = linear_model.LinearRegression()
audio_regr = linear_model.LinearRegression()
nn_regr = linear_model.LinearRegression()
llf_regr = linear_model.LinearRegression() #low level features

# Train the model using the training sets
visual_regr.fit(X_train, y_train[['MeanValence']].as_matrix().reshape((len(y_train))))
audio_regr.fit(Xa_train, ya_train[['MeanValence']].as_matrix().reshape((len(ya_train))))
nn_regr.fit(Xd_train, yd_train[['MeanValence']].as_matrix().reshape((len(yd_train))))
llf_regr.fit(Xl_train, yl_train[['MeanValence']].as_matrix().reshape((len(yl_train))))

# Make predictions using the testing set
visual_y_pred = visual_regr.predict(X_test)
audio_y_pred = audio_regr.predict(Xa_test)
nn_y_pred = nn_regr.predict(Xd_test)
llf_y_pred = llf_regr.predict(Xl_test)
```

```
CPU times: user 46.3 s, sys: 1.55 s, total: 47.9 s
Wall time: 24.7 s
```

In [90]:
```
# The coefficients
print('Visual Coefficients: \n', visual_regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(df2mat(y_test[['MeanValence']]), visual_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(df2mat(y_test[['MeanValence']]),visual_y_pred))

print('pearson score  ',pearsonr(df2mat(y_test[['MeanValence']]),visual_y_pred))
print
print('Audio Coefficients: \n', audio_regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(df2mat(ya_test[['MeanValence']]), audio_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(df2mat(ya_test[['MeanValence']]),audio_y_pred))
print('pearson score  ',pearsonr(df2mat(ya_test[['MeanValence']]),audio_y_pred))
print
print('FC16 Coefficients: \n', nn_regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(df2mat(yd_test[['MeanValence']]), nn_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(df2mat(yd_test[['MeanValence']]),nn_y_pred))

print('pearson score  ',pearsonr(df2mat(yd_test[['MeanValence']]),nn_y_pred))

print('Low Level Cinematographic: \n', llf_regr.coef_)
```

```
        # The mean squared error
        print("Mean squared error: %.2f"
              % mean_squared_error(df2mat(yl_test[['MeanValence']]), llf_y_pred))
        # Explained variance score: 1 is perfect prediction
        print('Variance score: %.2f' % r2_score(df2mat(yl_test[['MeanValence']]),llf_y_pred))

        print('pearson score  ',pearsonr(df2mat(yl_test[['MeanValence']]),llf_y_pred))
```

```
('Visual Coefficients: \n', array([  7.92078163e-03,  -3.41034481e-03,   3.22364570e-02, ...,
        -3.31345262e-06,  -1.76522489e-04,  -2.76182347e-04]))
Mean squared error: 0.16
Variance score: -0.34
('pearson score  ', (0.60696954128786806, 2.7659845977055354e-78))

('Audio Coefficients: \n', array([  1.19360087e-02,  -2.19720263e-02,   4.39910331e+04, ...,
        -8.10105063e-04,  -1.02620160e+04,   0.00000000e+00]))
Mean squared error: 440.49
Variance score: -3768.63
('pearson score  ', (0.015253347313571187, 0.67339184361473414))

('FC16 Coefficients: \n', array([ 0.02109452, -0.00158934,  0.01713063, ..., -0.03857747,
       -0.00368451, -0.01585526]))
Mean squared error: 0.10
Variance score: 0.15
('pearson score  ', (0.70600393665133665, 1.360521357770081e-116))
('Low Level Cinematographic: \n', array([-0.07195637,  0.00135865, -0.00345067,  0.01652764, -0.0034255
        0.03905086,  0.02084622, -0.00250101,  0.00123205]))
Mean squared error: 0.10
Variance score: 0.17
('pearson score  ', (0.4174090024014192, 1.1944386041728667e-33))
```

2.2 Grid Search on Visual Features- Valence

```
In [91]: from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.preprocessing import StandardScaler
         from sklearn.decomposition import PCA
         from sklearn.svm import SVR
         from sklearn.preprocessing import MinMaxScaler
         from sklearn.pipeline import Pipeline
         from sklearn.pipeline import make_pipeline
         from sklearn.grid_search import GridSearchCV
         from sklearn.pipeline import make_pipeline
         from sklearn.preprocessing import StandardScaler


         def getGridCV(pipe,paramgirid,Xtrain,ytrain,jobs=4): # scoring ? # jobs --> number of cores
             grid = GridSearchCV(pipe, param_grid, cv=5,n_jobs=jobs)
             grid.fit(Xtrain,ytrain)

             return grid
```

```
In [98]: %%time
```

```
#X_train, X_test, y_train, y_test
#pipe = Pipeline([('preprocessing', StandardScaler()),('reduce_dim', PCA()) ,('classifier', SVI
pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVR())])

param_grid = [
    {'classifier': [SVR()],
     'preprocessing': [StandardScaler()],
#     'reduce_dim': [PCA()],
#     'reduce_dim__n_components' : [ 800],
     'classifier__gamma': [0.0001, 0.001,0.01, 0.1, 1, 10, 100],
     'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100,200]},
    {'classifier': [RandomForestRegressor(n_estimators=100)],
     'preprocessing': [None], 'classifier__max_features': [3,5,10]}]

grid_vis_valence = getGridCV(pipe,param_grid,X_train,df2mat(y_train[['MeanValence']]))

print("Best params:\n{}\n".format(grid_vis_valence.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_vis_valence.best_score_))
print("All grid scores")

grid_vis_valence.grid_scores_, grid_vis_valence.best_params_, grid_vis_valence.best_score_
```

```
Best params:
{'classifier__gamma': 0.001, 'preprocessing': StandardScaler(copy=True, with_mean=True, with_std=True),
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False), 'classifier__C': 10}

Best cross-validation score: 0.82
All grid scores
CPU times: user 29.9 s, sys: 10.9 s, total: 40.7 s
Wall time: 13min 52s
```

In [99]:
```
tXv.shape,tXa.shape,ty.shape,
testXv.shape, testXa.shape, testy.shape
```

Out[99]:
```
((1434, 1271), (1434, 1583), (1434, 2))
```

In [100]:
```
#scores[0]
#scores.to_csv('grid_vis_valence.txt')
```

In [101]:
```
def gridscores(grid):
    scores = grid.grid_scores_
    rows = []
    params = sorted(scores[0].parameters)
    for row in scores:
        mean = row.mean_validation_score
        std = row.cv_validation_scores.std()
        rows.append([mean, std] + [row.parameters['classifier']])
    scores = pd.DataFrame(rows, columns=['mean_', 'std_'] + ['classifier'])
    #scores.to_csv(filename)
    return scores
```

In [105]:
```
gridscores(grid_vis_valence).tail()
```

Out[105]:
```
        mean_      std_                                        classifier
47 -0.008115  0.008917  SVR(C=200, cache_size=200, coef0=0.0, degree=3...
```

```
48 -0.008115  0.008917  SVR(C=200, cache_size=200, coef0=0.0, degree=3...
49  0.702275  0.022262  (DecisionTreeRegressor(criterion='mse', max_de...
50  0.714504  0.018777  (DecisionTreeRegressor(criterion='mse', max_de...
51  0.734224  0.018017  (DecisionTreeRegressor(criterion='mse', max_de...
```

2.3 Grid Search on Low Level Cinematographic Features- Valence

```
In [93]: %%time

         #Xl_train, Xl_test, yl_train, yl_test
         #pipe = Pipeline([('preprocessing', StandardScaler()),('reduce_dim', PCA()) ,('classifier', SVR
         pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVR())])

         param_grid = [
             {'classifier': [SVR()],
              'preprocessing': [StandardScaler()],
         #     'reduce_dim': [PCA()],
         #     'reduce_dim__n_components' : [ 800],
              'classifier__gamma': [0.0001, 0.001,0.01, 0.1, 1, 10, 100],
              'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100,200]},
             {'classifier': [RandomForestRegressor(n_estimators=20)],
              'preprocessing': [None], 'classifier__max_features': [3,5,9]}]

         grid_llf_valence = getGridCV(pipe,param_grid,Xl_train,df2mat(yl_train[['MeanValence']]))

         print("Best params:\n{}\n".format(grid_llf_valence.best_params_))
         print("Best cross-validation score: {:.2f}".format(grid_llf_valence.best_score_))
         print("All grid scores")

         grid_llf_valence.grid_scores_, grid_llf_valence.best_params_, grid_llf_valence.best_score_
```

```
Best params:
{'classifier__max_features': 9, 'preprocessing': None, 'classifier': RandomForestRegressor(bootstrap=Tru
           max_features=9, max_leaf_nodes=None, min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)}

Best cross-validation score: 0.44
All grid scores
CPU times: user 2.35 s, sys: 384 ms, total: 2.73 s
Wall time: 1min 28s
```

```
In [104]: gridscores(grid_llf_valence).tail()
```

```
Out[104]:        mean_      std_                                         classifier
          47  0.309450  0.032391  SVR(C=200, cache_size=200, coef0=0.0, degree=3...
          48  0.041772  0.020577  SVR(C=200, cache_size=200, coef0=0.0, degree=3...
          49  0.420782  0.029979  (DecisionTreeRegressor(criterion='mse', max_de...
          50  0.433318  0.025610  (DecisionTreeRegressor(criterion='mse', max_de...
          51  0.435960  0.038782  (DecisionTreeRegressor(criterion='mse', max_de...
```

2.4 Metrics and Paralell crossvalidation

```
In [111]: from sklearn import metrics
          from scipy.stats import pearsonr
```

```
def getMetrics(y,y_pred):
    # calculate MAE using scikit-learn
    #mae = metrics.mean_absolute_error(ytestarray, y_pred)
    #print("MAE score: {:.5f}".format(mae))

    mse = metrics.mean_squared_error(y, y_pred)
    # calculate MSE using scikit-learn
    print("MSE score: {:.5f}".format(mse))

    # calculate RMSE using scikit-learn
    #print("RMSE: {:.5f}".format(np.sqrt(metrics.mean_squared_error(ytestarray, y_pred))))

    print("Pearson score:")
    prs = pearsonr(y,y_pred)
    print(prs)

    return mse,prs
```

In [112]:
```
def evaluate_pipe(pipe,trainX,trainy,testX,testy):

    ytrainarray = trainy.as_matrix().reshape((len(trainy),))
    ytestarray = testy.as_matrix().reshape((len(testy),))

    pipe.fit(trainX, ytrainarray)

    print("Train score: {:.2f}".format(pipe.score(trainX, ytrainarray)))
    print("Test score: {:.2f}".format(pipe.score(testX, ytestarray)))

    y_pred = pipe.predict(testX)

    mse, prs = getMetrics(ytestarray,y_pred)

    return y_pred,mse,prs[0],pipe
```

In [113]:
```
from joblib import Parallel, delayed

import multiprocessing
# what are your inputs, and what operation do you want to
# perform on each input. For example...

def trainPipe(ii,pipe,valorar, modality):

    #rows = []
    trainlist, testlist=gettraintest2movielist(ii,mov2groups)  # index 1 olanları test , diğe
    tXv,tXa,tXd,tXl,ty = getFeatureswAnnotationsDf(trainlist)
    #print(tXv.shape,tXa.shape,tXd.shape, tXl.shape,ty.shape)

    testXv, testXa, testXd, testXl, testy = getFeatureswAnnotationsDf(testlist)
    #print(testXv.shape, testXa.shape,testXd.shape, testXl.shape, testy.shape)

    if modality == 'visual':
        y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                               tXv,ty[[valorar]],
                                               testXv, testy[[valorar]])
```

```
        elif modality == 'audio':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXa,ty[[valorar]],
                                                    testXa, testy[[valorar]])
        elif modality == 'deep':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXd,ty[[valorar]],
                                                    testXd, testy[[valorar]])
        else: ## lllf low level features
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXl,ty[[valorar]],
                                                    testXl, testy[[valorar]])


        return ii,mse,prs,p1

    def processGroup(ii,pipe,valorar, modality):

        #rows = []
        trainlist, testlist=gettraintest2movielist(ii,mov2groups)  # index 1 olanları test , diğe
        tXv,tXa,tXd,tXl,ty = getFeatureswAnnotationsDf(trainlist)
        #print(tXv.shape,tXa.shape,tXd.shape, tXl.shape,ty.shape)

        testXv, testXa, testXd, testXl, testy = getFeatureswAnnotationsDf(testlist)
        #print(testXv.shape, testXa.shape,testXd.shape, testXl.shape, testy.shape)

        if modality == 'visual':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXv,ty[[valorar]],
                                                    testXv, testy[[valorar]])
        elif modality == 'audio':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXa,ty[[valorar]],
                                                    testXa, testy[[valorar]])
        elif modality == 'deep':
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXd,ty[[valorar]],
                                                    testXd, testy[[valorar]])
        else: ## lllf low level features
            y_pred_test,mse,prs,p1 = evaluate_pipe(pipe,
                                                    tXl,ty[[valorar]],
                                                    testXl, testy[[valorar]])
        return [ii,mse,prs]

    def crossgroups(pipe,valorar,modality):
        #inputs=range(len(movgroups))
        inputs=[1, 2, 3, 4, 5]
        num_cores = multiprocessing.cpu_count()
        results = Parallel(n_jobs=num_cores)(delayed(processGroup)(i,pipe,valorar,modality) for i
        pipescores = pd.DataFrame(results,columns=['test-group','MSE','PCC'])

        return pipescores
```

```
In [114]: %%time
```

```
pipe_visual_valence = make_pipeline(
    StandardScaler(),
    SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
        gamma=0.001, kernel='rbf', max_iter=-1, shrinking=True,
        tol=0.001, verbose=False))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 649 µs
```

```
In [115]: %%time
          pipe_visual_arousal = make_pipeline(
              StandardScaler(),
              SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
                  gamma=0.001, kernel='rbf', max_iter=-1, shrinking=True,
                  tol=0.001, verbose=False))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 271 µs
```

```
In [116]: %%time
          pipe_audio_valence = make_pipeline(
              StandardScaler(copy=True, with_mean=True, with_std=True),
              SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.001,
                  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
          )
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 271 µs
```

```
In [117]: %%time
          pipe_audio_arousal = make_pipeline(
              StandardScaler(copy=True, with_mean=True, with_std=True),
              #PCA(n_components=800),
              SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.001,
                  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
          )
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 270 µs
```

```
In [118]: ii1,mse1,prs1,pva = trainPipe(4,pipe_visual_valence,'MeanValence','visual')
          ii2,mse2,prs2,pvv = trainPipe(3,pipe_audio_valence,'MeanValence','audio')
          ii3,mse3,prs3,pav = trainPipe(1,pipe_visual_arousal,'MeanArousal','visual')
          ii4,mse4,prs4,paa = trainPipe(1,pipe_audio_arousal,'MeanArousal','audio')
```

```
Train score: 0.95
Test score: -0.73
MSE score: 0.15579
Pearson score:
(0.39414440724472483, 6.1026278496011477e-55)
Train score: 0.94
Test score: -0.03
MSE score: 0.09450
Pearson score:
(0.25340858644380365, 1.7133760887450425e-22)
Train score: 0.93
```

```
Test score: -0.35
MSE score: 0.08113
Pearson score:
(0.14142836196030178, 4.6136082789378667e-08)
Train score: 0.91
Test score: 0.04
MSE score: 0.05775
Pearson score:
(0.37670638887075814, 3.8682936818101966e-51)
```

In [55]:
```
# cross validation takes too much time
# do not run if it not necessary
paa_scores = crossgroups(pipe_audio_arousal,'MeanArousal','audio')
pva_scores = crossgroups(pipe_audio_valence,'MeanValence','audio')
pav_scores = crossgroups(pipe_visual_arousal,'MeanArousal','visual')
pvv_scores = crossgroups(pipe_visual_valence,'MeanValence','visual')

paa_scores.sort_values('PCC', ascending=False)
pav_scores.sort_values('PCC', ascending=False)
pva_scores.sort_values('PCC', ascending=False)
pvv_scores.sort_values('PCC', ascending=False)
```

Out[55]:

	test-group	MSE	PCC
0	1	0.057745	0.376706
3	4	0.089518	0.193804
2	3	0.066033	0.156579
4	5	0.074860	0.148383
1	2	0.090814	0.114442

2.5 Smoothing

In [119]:
```
def getAVprediction(f):
    audiodf = getAudioDf(f)
    visualdf = getAvgVisFeatListDf(f,visual_feat_list)
    annotdf = getAnnotationDf(f)
    ya = df2mat(annotdf[['MeanArousal']])
    yv = df2mat(annotdf[['MeanValence']])

    print(audiodf.shape,visualdf.shape,annotdf.shape)

    mlen = min(len(audiodf),len(visualdf))

    audiodf = audiodf[:mlen]
    visualdf = visualdf[:mlen]
    ya = ya[:mlen]
    yv = yv[:mlen]

    aa = paa.predict(audiodf)
    av = pav.predict(visualdf)

    va = pvv.predict(audiodf)
    vv = pva.predict(visualdf)

    df =pd.DataFrame(np.transpose([vv, va , aa, av ]), columns=['MeanValenceAudio','MeanValen
```

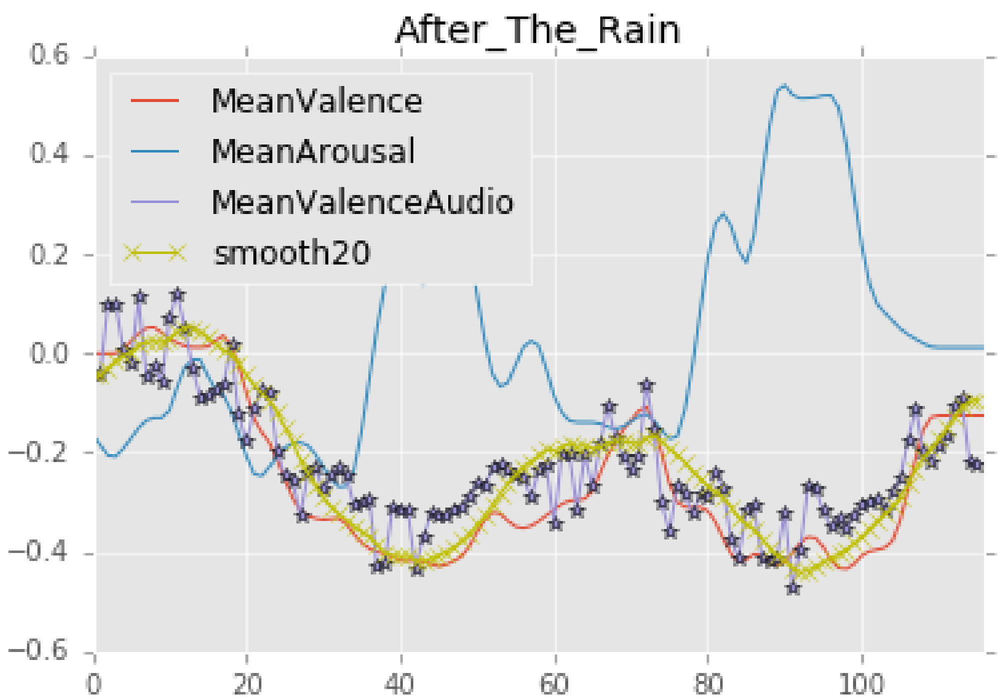
```
        return df
```

In [120]:
```python
import numpy as np

def holt_winters_second_order_ewma( x, span, beta ):
    N = x.size
    alpha = 2.0 / ( 1 + span )
    s = np.zeros(( N, ))
    b = np.zeros(( N, ))
    s[0] = x[0]
    for i in range( 1, N ):
        s[i] = alpha * x[i] + ( 1 - alpha )*( s[i-1] + b[i-1] )
        b[i] = beta * ( s[i] - s[i-1] ) + ( 1 - beta ) * b[i-1]
    return s
```

In [123]:
```python
mov =movieNames[0]
aa = getAVprediction(mov)
dfa = getAnnotationDf(mov)
smooth10 = holt_winters_second_order_ewma( df2mat(aa[['MeanValenceAudio']]), 10, 0.3 )
smooth5 = holt_winters_second_order_ewma( df2mat(aa[['MeanValenceAudio']]), 5, 0.3 )
smooth2 = aa[['MeanValenceAudio']].rolling(window=10).mean()
smooth20 = holt_winters_second_order_ewma( df2mat(aa[['MeanValenceAudio']]), 20, 0.3 )

dfa[['MeanValence','MeanArousal']].plot(ax=plt.gca(),title=mov)
aa[['MeanValenceAudio']].plot(ax=plt.gca(), style=['*-'], title=mov)
#pd.DataFrame(smooth10,columns=['smooth10']).plot(ax=plt.gca(),style=['.-'],title=mov)
#pd.DataFrame(smooth5,columns=['smooth5']).plot(ax=plt.gca(),style=['g+-'],title=mov)
pd.DataFrame(smooth20,columns=['smooth20']).plot(ax=plt.gca(),style=['yx-'],title=mov)
#ax=plt.gca()
#plt.plot(smooth1,label='smoothing')
#plt.plot(smooth2)
```

```
((116, 1583), (117, 1271), (117, 4))
```

Out[123]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f758fea4cd0>
```




```
In [125]: smooth10.shape,aa.shape
```

```
Out[125]: ((116,), (116, 4))
```

2.6 Generating N-fold csv

```
In [126]: visual_feat_list= ['acc', 'cedd', 'cl', 'eh', 'fcth',
                             'gabor', 'jcd', 'lbp', 'sc', 'tamura'   ]
```

```
In [128]: %%time
          # Visual
          pipe_visual_valence = make_pipeline(
              StandardScaler(),
              SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
                  gamma=0.001, kernel='rbf', max_iter=-1, shrinking=True,
                  tol=0.001, verbose=False))

          pipe_visual_arousal = make_pipeline(
              StandardScaler(),
              SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
                  gamma=0.001, kernel='rbf', max_iter=-1, shrinking=True,
                  tol=0.001, verbose=False))

          # Audio
          pipe_audio_valence = make_pipeline(
              StandardScaler(copy=True, with_mean=True, with_std=True),
              SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.001,
                  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False))
```

```
pipe_audio_arousal = make_pipeline(
    StandardScaler(copy=True, with_mean=True, with_std=True),
    #PCA(n_components=800),
    SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.001,
        kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False))

# FC16 -->deep fetures
pipe_deep_valence = make_pipeline(
    StandardScaler(),
    SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
        gamma=0.001, kernel='rbf', max_iter=-1, shrinking=True,
        tol=0.001, verbose=False))

pipe_deep_arousal = make_pipeline(
    StandardScaler(),
    SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
        gamma=0.001, kernel='rbf', max_iter=-1, shrinking=True,
        tol=0.001, verbose=False))

# Low Level Features
pipe_llf_valence = make_pipeline(
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=9, max_leaf_nodes=None, min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False))

pipe_llf_arousal = make_pipeline(
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=9, max_leaf_nodes=None, min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False))
```

```
CPU times: user 4 ms, sys: 0 ns, total: 4 ms
Wall time: 1.79 ms
```

In [132]:
```
%%time
import os
idev_set = {}
smoothing = True

allfold_metric=[]
folddf_dict={}
mean_metric = []

for foldi in [1,2,3,4,5]:
    trainlist, testlist=gettraintest2movielist(foldi,mov2groups)
    os.system("mkdir NfoldCV/fold"+str(foldi))
    testfolder="NfoldCV/fold"+str(foldi)+"/test/"
    trainfolder="NfoldCV/fold"+str(foldi)+"/train/"
    os.system("mkdir "+testfolder)
    os.system("mkdir "+trainfolder)
```

```
ii1,mse1,prs1,pvvis = trainPipe(foldi,pipe_visual_valence,'MeanValence','visual')
ii2,mse2,prs2,pvaud = trainPipe(foldi,pipe_audio_valence,'MeanValence','audio')

ii3,mse3,prs3,pavis = trainPipe(foldi,pipe_visual_arousal,'MeanArousal','visual')
ii4,mse4,prs4,paaud = trainPipe(foldi,pipe_audio_arousal,'MeanArousal','audio')

ii5,mse5,prs5,pvdeep = trainPipe(foldi,pipe_deep_valence,'MeanValence','deep')
ii6,mse6,prs6,padeep = trainPipe(foldi,pipe_deep_arousal,'MeanArousal','deep')

ii7,mse7,prs7,pvlow = trainPipe(foldi,pipe_llf_valence,'MeanValence','llf')
ii8,mse8,prs8,palow = trainPipe(foldi,pipe_llf_arousal,'MeanArousal','llf')


fold_metric=[]
for f in testlist:
    audiodf = getAudioDf(f)
    visualdf = getAvgVisFeatListDf(f,visual_feat_list)
    deepdf = getAvgVisFeatListDf(f,['fc6'])
    lowdf = getLowFeature10SecDf(f)

    annotdf = getAnnotationDf(f)
    ya = df2mat(annotdf[['MeanArousal']])
    yv = df2mat(annotdf[['MeanValence']])

    print(audiodf.shape,visualdf.shape,deepdf.shape,lowdf.shape)

    mlen = min([len(audiodf),len(visualdf),len(deepdf),len(lowdf)])

    audiodf = audiodf[:mlen]
    visualdf = visualdf[:mlen]
    deepdf = deepdf[:mlen]
    lowdf = lowdf[:mlen]

    ya = ya[:mlen]
    yv = yv[:mlen]

    aa = paaud.predict(audiodf)
    va = pvaud.predict(audiodf)

    av = pavis.predict(visualdf)
    vv = pvvis.predict(visualdf)

    ad = padeep.predict(deepdf)
    vd = pvdeep.predict(deepdf)

    al = palow.predict(lowdf)
    vl = pvlow.predict(lowdf)


    if smoothing:
        aa = holt_winters_second_order_ewma( aa, 10, 0.3 )
        av = holt_winters_second_order_ewma( av, 10, 0.3 )
        va = holt_winters_second_order_ewma( va, 10, 0.3 )
        vv = holt_winters_second_order_ewma( vv, 10, 0.3 )
```

```
            ad = holt_winters_second_order_ewma( ad, 10, 0.3 )
            al = holt_winters_second_order_ewma( al, 10, 0.3 )
            vd = holt_winters_second_order_ewma( vd, 10, 0.3 )
            vl = holt_winters_second_order_ewma( vl, 10, 0.3 )

        mseaa, prsaa = getMetrics(ya,aa)
        mseav, prsav = getMetrics(ya,av)
        mseva, prsva = getMetrics(yv,va)
        msevv, prsvv = getMetrics(yv,vv)

        msead, prsad = getMetrics(ya,ad)
        mseal, prsal = getMetrics(ya,al)
        msevd, prsvd = getMetrics(yv,vd)
        msevl, prsvl = getMetrics(yv,vl)

        t = [msevv, prsvv[0], mseva, prsva[0] , mseaa, prsaa[0], mseav, prsav[0],
             msevl, prsvl[0], msevd, prsvd[0] , mseal, prsal[0], msead, prsad[0]]

        fold_metric.append(t)
        allfold_metric.append(t)

        arousal_scores = np.transpose([ aa,av,ad,al ])
        arousal_scores = np.mean(arousal_scores,axis=1)
        valence_scores = np.transpose([va,vv,vd,vl ])
        valence_scores = np.mean(valence_scores,axis=1)

        meandf = pd.DataFrame(np.transpose([valence_scores, arousal_scores]), columns=['MeanVa

        #mseA, prsA= getMetrics(ya,arousal_scores)
        #mseV, prsV = getMetrics(yv,valence_scores)

        #mean_metric.append([mseV, prsV, mseA, prsA])


        df =pd.DataFrame(np.transpose([va, vv, vd, vl, aa, av, ad, al ]),
                         columns=['MeanValenceAudio','MeanValenceVisual',
                                  'MeanValenceDeep','MeanValenceLow',
                                  'MeanArousalAudio','MeanArousalVisual',
                                  'MeanArousalDeep','MeanArousalLow'])
        idev_set[f] = df
        filename=testfolder+str(foldi)+"_"+f+".csv"
        df.to_csv(filename, index=False)


    folddf = pd.DataFrame(fold_metric, columns=['MeanValenceVisualMSE','MeanValenceVisualPCC'
                                                'MeanValenceAudioMSE','MeanValenceAudioPCC',
                                                'MeanArousalAudioMSE','MeanArousalAudioPCC',
                                                'MeanArousalVisualMSE','MeanArousalVisualPCC'
                                                'MeanValenceLowLevelMSE','MeanValenceLowLevelI
                                                'MeanValenceDeepMSE','MeanValenceDeepPCC',
                                                'MeanArousalLowLevelMSE','MeanArousalLowLevelI
                                                'MeanArousalDeepMSE','MeanArousalDeepPCC'])

    folddf.to_csv(testfolder+str(foldi)+"_metrics.csv")
```

```
            folddf.describe().to_csv(testfolder+str(foldi)+"_metrics_stats.csv")
            folddf_dict[foldi] = folddf

            #########################################ATTENTION ################################
            '''
            for f in trainlist:
                audiodf = getAudioDf(f)
                visualdf = getAvgVisFeatListDf(f,visual_feat_list)
                #print(audiodf.shape,visualdf.shape)

                mlen = min(len(audiodf),len(visualdf))

                audiodf = audiodf[:mlen]
                visualdf = visualdf[:mlen]

                aa = paa.predict(audiodf)
                av = pav.predict(visualdf)

                va = pva.predict(audiodf)
                vv = pvv.predict(visualdf)

                if smoothing:
                    aa = holt_winters_second_order_ewma( aa, 10, 0.3 )
                    av = holt_winters_second_order_ewma( av, 10, 0.3 )
                    va = holt_winters_second_order_ewma( va, 10, 0.3 )
                    vv = holt_winters_second_order_ewma( vv, 10, 0.3 )

                df =pd.DataFrame(np.transpose([ va, vv ,aa,av ]), columns=['MeanValenceAudio','MeanVal
                idev_set[f] = df
                filename=trainfolder+str(foldi)+"_"+f+".csv"
                df.to_csv(filename, index=False)

            '''

        allfolddf = pd.DataFrame(allfold_metric, columns=['MeanValenceVisualMSE','MeanValenceVisualPCC
                                                    'MeanValenceAudioMSE','MeanValenceAudioPCC',
                                                    'MeanArousalAudioMSE','MeanArousalAudioPCC',
                                                    'MeanArousalVisualMSE','MeanArousalVisualPCC'
                                                    'MeanValenceLowLevelMSE','MeanValenceLowLevel
                                                    'MeanValenceDeepMSE','MeanValenceDeepPCC',
                                                    'MeanArousalLowLevelMSE','MeanArousalLowLevel
                                                    'MeanArousalDeepMSE','MeanArousalDeepPCC'])

        allfolddf.to_csv("all_metrics.csv")
        allfolddf.describe().to_csv("all_metrics_stats.csv")
        #mean_metricdf=pd.DataFrame(mean_metric,columns=['mseV, prsV, mseA, prsA'])
```

```
Train score: 0.95
Test score: -0.61
MSE score: 0.12757
Pearson score:
(-0.012512400010095973, 0.63041949402968256)
Train score: 0.94
Test score: -0.22
```

```
MSE score: 0.09649
Pearson score:
(0.19071660961172446, 1.3482767640132043e-13)
Train score: 0.93
Test score: -0.35
MSE score: 0.08113
Pearson score:
(0.14142836196030178, 4.6136082789378667e-08)
Train score: 0.91
Test score: 0.04
MSE score: 0.05775
Pearson score:
(0.37670638887075814, 3.8682936818101966e-51)
Train score: 0.94
Test score: -0.44
MSE score: 0.11420
Pearson score:
(-0.042473260038367558, 0.10228235951712471)
Train score: 0.91
Test score: -0.15
MSE score: 0.06865
Pearson score:
(0.090184750907558536, 0.00051123295475250809)
Train score: 0.91
Test score: -0.60
MSE score: 0.12740
Pearson score:
(0.06478722574275321, 0.012639397730183766)
Train score: 0.90
Test score: -0.48
MSE score: 0.08871
Pearson score:
(-0.033876046103210412, 0.19259151861293355)
((173, 1583), (173, 1271), (173, 4096), (179, 9))
MSE score: 0.03803
Pearson score:
(-0.14270504991356223, 0.061070857499530978)
MSE score: 0.05842
Pearson score:
(0.085586539584825025, 0.26288003622058453)
MSE score: 0.06588
Pearson score:
(0.14318743671127038, 0.06019272065557324)
MSE score: 0.05513
Pearson score:
(0.18376150988868972, 0.015515005581048213)
MSE score: 0.06809
Pearson score:
(0.2400740325155189, 0.0014654073286630505)
MSE score: 0.07035
Pearson score:
(-0.076108043158904071, 0.31962334513097079)
MSE score: 0.09297
Pearson score:
```

```
(-0.17085313701665072, 0.024609233002888194)
MSE score: 0.04608
Pearson score:
(0.46032683668365765, 1.8643195364170866e-10)
((201, 1583), (202, 1271), (202, 4096), (202, 9))
MSE score: 0.04436
Pearson score:
(0.55824107270904377, 7.3044049575283206e-18)
MSE score: 0.06578
Pearson score:
(0.022701621131758748, 0.74905554794190299)
MSE score: 0.10260
Pearson score:
(0.26379706369393707, 0.00015439665160572066)
MSE score: 0.22688
Pearson score:
(-0.21872419172803215, 0.0018117675240275593)
MSE score: 0.05647
Pearson score:
(0.34938769693347366, 3.7023271422037065e-07)
MSE score: 0.07003
Pearson score:
(-0.23888866420086124, 0.0006371373103614765)
MSE score: 0.19946
Pearson score:
(-0.01957349847537666, 0.78270326416041613)
MSE score: 0.20120
Pearson score:
(0.21089851909367119, 0.0026538038045598954)
((185, 1583), (185, 1271), (185, 4096), (193, 9))
MSE score: 0.05025
Pearson score:
(0.27144569118584022, 0.00018589179763388005)
MSE score: 0.04612
Pearson score:
(-0.23802891429567338, 0.0011038665668839365)
MSE score: 0.05395
Pearson score:
(0.0038593741111404433, 0.95841908090093109)
MSE score: 0.10323
Pearson score:
(0.10484281350316169, 0.15552957135128831)
MSE score: 0.03756
Pearson score:
(0.058969592851221712, 0.42525881545574817)
MSE score: 0.03711
Pearson score:
(0.050303499200019024, 0.49650774446085233)
MSE score: 0.09958
Pearson score:
(0.029221952051982168, 0.69295216978537189)
MSE score: 0.05514
Pearson score:
(-0.050246652833609136, 0.49699505522464482)
```

```
((184, 1583), (185, 1271), (185, 4096), (192, 9))
MSE score: 0.05778
Pearson score:
(0.18724057580947751, 0.010924768544462072)
MSE score: 0.04949
Pearson score:
(0.44822051213232778, 1.7680946414609396e-10)
MSE score: 0.06731
Pearson score:
(0.42445151802890174, 1.9174301489716938e-09)
MSE score: 0.12614
Pearson score:
(-0.020851464639017978, 0.77875177452306776)
MSE score: 0.05417
Pearson score:
(0.08668545309628567, 0.24198112411285552)
MSE score: 0.05426
Pearson score:
(0.37991889964362324, 1.0425141253693429e-07)
MSE score: 0.08813
Pearson score:
(0.44457420682516086, 2.5795982558780706e-10)
MSE score: 0.08448
Pearson score:
(0.31236080530243404, 1.5837846447224111e-05)
((139, 1583), (139, 1271), (139, 4096), (139, 9))
MSE score: 0.07145
Pearson score:
(0.41599937004470455, 3.523613045505004e-07)
MSE score: 0.21438
Pearson score:
(0.080835173232638724, 0.34416097568381487)
MSE score: 0.07360
Pearson score:
(-0.053054201433692466, 0.53506787592565352)
MSE score: 0.09646
Pearson score:
(-0.011574783082831452, 0.892424574999095)
MSE score: 0.08924
Pearson score:
(0.076174466950318681, 0.37278451964720505)
MSE score: 0.11740
Pearson score:
(0.15933778277756802, 0.06098705647006171)
MSE score: 0.06022
Pearson score:
(0.22767879030758148, 0.0070272294948482902)
MSE score: 0.07563
Pearson score:
(-0.060334727198849317, 0.48046457072975457)
((99, 1583), (99, 1271), (99, 4096), (99, 9))
MSE score: 0.09079
Pearson score:
(0.048612380416510795, 0.63277347944207851)
```

```
MSE score: 0.06502
Pearson score:
(0.080596106056117334, 0.42776161389259404)
MSE score: 0.15675
Pearson score:
(0.45296118178086453, 2.505891748940269e-06)
MSE score: 0.19138
Pearson score:
(0.035103529193333163, 0.73013483958457903)
MSE score: 0.08397
Pearson score:
(0.10990304387839665, 0.27884489551261821)
MSE score: 0.10943
Pearson score:
(-0.31083980065368816, 0.0017391819065588723)
MSE score: 0.14777
Pearson score:
(0.58472036760924262, 2.0875381015978554e-10)
MSE score: 0.21228
Pearson score:
(-0.13795238422599895, 0.17329360151636505)
((179, 1583), (179, 1271), (179, 4096), (187, 9))
MSE score: 0.05865
Pearson score:
(-0.083039881747842104, 0.26910571364408487)
MSE score: 0.07901
Pearson score:
(0.39209161218880301, 5.6985210607552518e-08)
MSE score: 0.14581
Pearson score:
(0.00026708394475527118, 0.99716886768893598)
MSE score: 0.15954
Pearson score:
(-0.25721700842673723, 0.00050916180833854362)
MSE score: 0.07519
Pearson score:
(0.32367400273370744, 9.8843398700950349e-06)
MSE score: 0.13942
Pearson score:
(-0.070102280478676074, 0.35108767341725156)
MSE score: 0.15199
Pearson score:
(0.38130492057705384, 1.3935485361718062e-07)
MSE score: 0.23475
Pearson score:
(-0.080419019808144199, 0.28456094830049516)
((145, 1583), (146, 1271), (146, 4096), (146, 9))
MSE score: 0.03464
Pearson score:
(0.59811188330446297, 1.9630915729488648e-15)
MSE score: 0.05668
Pearson score:
(0.50064858797970146, 1.4303356988237063e-10)
MSE score: 0.07499
```

```
Pearson score:
(-0.23452391445255319, 0.0045211263012450336)
MSE score: 0.04143
Pearson score:
(-0.073772666806773857, 0.37785559481168329)
MSE score: 0.04098
Pearson score:
(0.48820114201488679, 4.695924860449316e-10)
MSE score: 0.07294
Pearson score:
(-0.17916145964777513, 0.031068436668360542)
MSE score: 0.02941
Pearson score:
(0.21961666094253648, 0.0079513105122632679)
MSE score: 0.04719
Pearson score:
(0.24820690905113779, 0.0026108458267766243)
((176, 1583), (176, 1271), (176, 4096), (176, 9))
MSE score: 0.04627
Pearson score:
(0.58488582962320301, 1.5479196972074053e-17)
MSE score: 0.08217
Pearson score:
(0.29524761376898462, 6.9485556221592835e-05)
MSE score: 0.09071
Pearson score:
(0.31939023656776427, 1.5552675088410462e-05)
MSE score: 0.12393
Pearson score:
(0.015984761689839526, 0.83322733856931808)
MSE score: 0.09727
Pearson score:
(0.38368293914790158, 1.4686939272632629e-07)
MSE score: 0.08952
Pearson score:
(0.27182815600413768, 0.00026277854069280268)
MSE score: 0.10103
Pearson score:
(0.14012925651963076, 0.063604640997899198)
MSE score: 0.09016
Pearson score:
(0.33212274858162544, 6.6936797448393209e-06)
Train score: 0.95
Test score: -0.28
MSE score: 0.15103
Pearson score:
(0.098090803348593658, 0.00019902024525627986)
Train score: 0.93
Test score: -0.04
MSE score: 0.12313
Pearson score:
(0.21517994508296434, 1.7503029564855477e-16)
Train score: 0.93
Test score: -0.15
```

```
MSE score: 0.09816
Pearson score:
(0.053944118034337858, 0.041104723520646889)
Train score: 0.90
Test score: -0.06
MSE score: 0.09081
Pearson score:
(0.11444228933541915, 1.3976939487451369e-05)
Train score: 0.93
Test score: -0.11
MSE score: 0.13143
Pearson score:
(0.14813439056871164, 1.7423773945271492e-08)
Train score: 0.91
Test score: -0.15
MSE score: 0.09806
Pearson score:
(-0.114368199799006, 1.4158280576111885e-05)
Train score: 0.91
Test score: -0.34
MSE score: 0.15866
Pearson score:
(0.044030893039751176, 0.095569168134136878)
Train score: 0.91
Test score: -0.31
MSE score: 0.11233
Pearson score:
(-0.1218622393068446, 3.6932767303690234e-06)
((61, 1583), (61, 1271), (61, 4096), (61, 9))
MSE score: 0.34893
Pearson score:
(0.38428641430702593, 0.002231025753671127)
MSE score: 0.16925
Pearson score:
(-0.61546808754075177, 1.302493499804376e-07)
MSE score: 0.06332
Pearson score:
(-0.3837823519471239, 0.0022636137967450608)
MSE score: 0.17941
Pearson score:
(-0.023412536978172242, 0.8578597110452103)
MSE score: 0.29598
Pearson score:
(0.49970779851644959, 4.1300437300475468e-05)
MSE score: 0.25420
Pearson score:
(0.2313042568402906, 0.072880284718976798)
MSE score: 0.14664
Pearson score:
(0.2794493783361558, 0.02917814181272807)
MSE score: 0.11369
Pearson score:
(-0.16947675956227751, 0.19164070203485442)
((129, 1583), (129, 1271), (129, 4096), (129, 9))
```

```
MSE score: 0.03173
Pearson score:
(0.45462981193106139, 6.2247409278380994e-08)
MSE score: 0.09025
Pearson score:
(-0.31990095554435, 0.00021935756640229398)
MSE score: 0.10369
Pearson score:
(-0.02430661779018569, 0.78452775946514053)
MSE score: 0.19786
Pearson score:
(-0.080925359145382805, 0.36193429485020367)
MSE score: 0.04646
Pearson score:
(0.65467983487871917, 3.9283056316570781e-17)
MSE score: 0.06551
Pearson score:
(0.10873271916912633, 0.21997816117990682)
MSE score: 0.08917
Pearson score:
(-0.07340880558442793, 0.40837045992085708)
MSE score: 0.11491
Pearson score:
(-0.31707170302484367, 0.00025111211940690326)
((77, 1583), (77, 1271), (77, 4096), (77, 9))
MSE score: 0.06097
Pearson score:
(-0.14353404156131008, 0.21299751777012751)
MSE score: 0.04272
Pearson score:
(0.57028965226932327, 6.1543326753636661e-08)
MSE score: 0.05179
Pearson score:
(0.70878997222861617, 5.5028175139323514e-13)
MSE score: 0.05174
Pearson score:
(0.027326828216977711, 0.81349951709871138)
MSE score: 0.04950
Pearson score:
(0.24544712045907355, 0.031429534991153453)
MSE score: 0.06846
Pearson score:
(-0.48549244342335657, 7.6278067400859691e-06)
MSE score: 0.09824
Pearson score:
(0.1353827741757723, 0.2404102466479284)
MSE score: 0.04914
Pearson score:
(0.19474865101814529, 0.089648790235771089)
((118, 1583), (118, 1271), (118, 4096), (118, 9))
MSE score: 0.07427
Pearson score:
(0.076914774034189198, 0.40776030782544614)
MSE score: 0.10391
```

```
Pearson score:
(0.35821672678895489, 6.8105014713321408e-05)
MSE score: 0.19622
Pearson score:
(0.006006289128506378, 0.94853147522809467)
MSE score: 0.15548
Pearson score:
(-0.18248426585892769, 0.047948456470259276)
MSE score: 0.10592
Pearson score:
(0.58266180358279973, 4.4595358767940322e-12)
MSE score: 0.09185
Pearson score:
(0.16464136937326293, 0.074813936930929872)
MSE score: 0.18591
Pearson score:
(0.077200169569388216, 0.40601749090701789)
MSE score: 0.31417
Pearson score:
(-0.14038973342282812, 0.12944131259346384)
((100, 1583), (101, 1271), (101, 4096), (101, 9))
MSE score: 0.17646
Pearson score:
(0.25110484672171157, 0.01173573470510441)
MSE score: 0.18819
Pearson score:
(0.05984398386625546, 0.55422134338758555)
MSE score: 0.10355
Pearson score:
(-0.36368427268784354, 0.00019975577902427133)
MSE score: 0.07890
Pearson score:
(0.18196795437387958, 0.06999216424938598)
MSE score: 0.15408
Pearson score:
(0.27491280877768276, 0.0056381835278664907)
MSE score: 0.20193
Pearson score:
(-0.16745793340404988, 0.095851487952360162)
MSE score: 0.07837
Pearson score:
(0.21360839319486147, 0.032849096040908202)
MSE score: 0.11627
Pearson score:
(-0.36377847814878911, 0.00019893872667663701)
((223, 1583), (223, 1271), (223, 4096), (230, 9))
MSE score: 0.08836
Pearson score:
(0.20628286733024673, 0.0019580631080086701)
MSE score: 0.09727
Pearson score:
(0.15282685054730988, 0.022442615246202103)
MSE score: 0.14695
Pearson score:
```

```
(0.032466913796261994, 0.62963769389205015)
MSE score: 0.26745
Pearson score:
(-0.077744742355276314, 0.24759819293321944)
MSE score: 0.09133
Pearson score:
(0.2196088333193435, 0.00096195258076755673)
MSE score: 0.10621
Pearson score:
(0.14564190814052624, 0.029684016578779417)
MSE score: 0.22617
Pearson score:
(0.14036232896875711, 0.036201424690402245)
MSE score: 0.18399
Pearson score:
(0.19327847480509386, 0.0037625387595810792)
((33, 1583), (33, 1271), (33, 4096), (35, 9))
MSE score: 0.02891
Pearson score:
(0.27996887312264895, 0.11455887099953904)
MSE score: 0.01189
Pearson score:
(-0.84525731184528297, 6.0742055672870134e-10)
MSE score: 0.41757
Pearson score:
(0.05292396555359187, 0.76989875675392794)
MSE score: 0.15702
Pearson score:
(0.14196509959368664, 0.43064651317198965)
MSE score: 0.02861
Pearson score:
(-0.034029829484374864, 0.85087422890611764)
MSE score: 0.02505
Pearson score:
(-0.14720859786678145, 0.41363700296078809)
MSE score: 0.33087
Pearson score:
(0.71799625284259905, 2.5514975382395568e-06)
MSE score: 0.25429
Pearson score:
(0.57196425356174996, 0.00050613173515813l8)
((81, 1583), (82, 1271), (82, 4096), (85, 9))
MSE score: 0.18482
Pearson score:
(-0.55183481879086471, 9.3034664231801443e-08)
MSE score: 0.20626
Pearson score:
(-0.016793635965873261, 0.88170828921579203)
MSE score: 0.14134
Pearson score:
(-0.50147246454279659, 1.8532272128251089e-06)
MSE score: 0.16674
Pearson score:
(-0.064321234108344325, 0.56834895464149193)
```

```
MSE score: 0.21321
Pearson score:
(-0.12509694661328899, 0.2658153344322608)
MSE score: 0.23827
Pearson score:
(-0.25644926739833268, 0.020834963014937244)
MSE score: 0.12632
Pearson score:
(-0.36876880708819121, 0.0007051661767412588)
MSE score: 0.15588
Pearson score:
(-0.36951970204793771, 0.00068630493552234611)
((339, 1583), (339, 1271), (339, 4096), (353, 9))
MSE score: 0.03709
Pearson score:
(0.25162041862676082, 2.713332510352873e-06)
MSE score: 0.06818
Pearson score:
(-0.37574938326809637, 8.2660261252623241e-13)
MSE score: 0.10888
Pearson score:
(0.046392347297252889, 0.39450427670589494)
MSE score: 0.09368
Pearson score:
(0.11340781225230985, 0.036880494894198521)
MSE score: 0.06872
Pearson score:
(-0.2836738868731184, 1.0769372344739105e-07)
MSE score: 0.07177
Pearson score:
(0.16124522841773675, 0.0029072296229213277)
MSE score: 0.10195
Pearson score:
(0.097507498274830087, 0.072981403955718283)
MSE score: 0.11340
Pearson score:
(-0.02893475346164363, 0.59549588071051307)
((116, 1583), (117, 1271), (117, 4096), (122, 9))
MSE score: 0.06176
Pearson score:
(0.11361168685697264, 0.22462559821374836)
MSE score: 0.05880
Pearson score:
(-0.015876490969506319, 0.86567561002971205)
MSE score: 0.05700
Pearson score:
(-0.1667343083743891, 0.073634150459950462)
MSE score: 0.20454
Pearson score:
(0.17513695289528455, 0.060051399538115877)
MSE score: 0.06016
Pearson score:
(-0.12969785956416952, 0.16524991175492623)
MSE score: 0.08567
```

```
Pearson score:
(-0.15229261082810308, 0.10267678108529364)
MSE score: 0.08483
Pearson score:
(0.086166742806224342, 0.3577265039449965)
MSE score: 0.18738
Pearson score:
(0.011187790539409031, 0.90512088370064403)
((157, 1583), (157, 1271), (157, 4096), (157, 9))
MSE score: 0.09548
Pearson score:
(-0.19504040000904987, 0.014370101625557253)
MSE score: 0.07815
Pearson score:
(0.28360946012255167, 0.00031893613099997142)
MSE score: 0.10601
Pearson score:
(-0.27790661492685875, 0.00042506474619989664)
MSE score: 0.09367
Pearson score:
(0.12717297129293093, 0.11246950311575327)
MSE score: 0.07380
Pearson score:
(0.29508598846507672, 0.0001755946298769031)
MSE score: 0.08020
Pearson score:
(0.087912462733925612, 0.27357849766881415)
MSE score: 0.06103
Pearson score:
(0.09325151565160586, 0.24538410760507973)
MSE score: 0.09357
Pearson score:
(-0.1248169444404843, 0.1193395936664985)
Train score: 0.95
Test score: -0.42
MSE score: 0.13052
Pearson score:
(-0.030358901087373207, 0.2501017640061674)
Train score: 0.94
Test score: -0.03
MSE score: 0.09450
Pearson score:
(0.25340858644380365, 1.7133760887450425e-22)
Train score: 0.93
Test score: -0.22
MSE score: 0.07402
Pearson score:
(0.04480378814946976, 0.089547996725428611)
Train score: 0.92
Test score: -0.09
MSE score: 0.06603
Pearson score:
(0.15657884026331143, 2.4131042754082872e-09)
Train score: 0.94
```

```
Test score: -0.08
MSE score: 0.09928
Pearson score:
(0.11420442559015752, 1.4269395686392009e-05)
Train score: 0.92
Test score: -0.07
MSE score: 0.06460
Pearson score:
(0.043951943929232261, 0.095817889383945809)
Train score: 0.91
Test score: -0.42
MSE score: 0.13073
Pearson score:
(0.048007082017291876, 0.068864547967358358)
Train score: 0.89
Test score: -0.18
MSE score: 0.07123
Pearson score:
(0.12392924974298156, 2.4533370079318679e-06)
((48, 1583), (49, 1271), (49, 4096), (50, 9))
MSE score: 0.15712
Pearson score:
(0.32158437206419738, 0.025824194190595336)
MSE score: 0.23146
Pearson score:
(0.17038628426764499, 0.2469275898340948)
MSE score: 0.04537
Pearson score:
(0.41968460296794968, 0.0029832811312958392)
MSE score: 0.55452
Pearson score:
(-0.17343031241902915, 0.23845931354699265)
MSE score: 0.15956
Pearson score:
(0.56685500847296288, 2.6639998517920434e-05)
MSE score: 0.10775
Pearson score:
(-0.10216786706255769, 0.48957008823137049)
MSE score: 0.07797
Pearson score:
(-0.18557809770181016, 0.2066486246889413)
MSE score: 0.13711
Pearson score:
(0.075588174022537424, 0.6096189778957446)
((273, 1583), (273, 1271), (273, 4096), (282, 9))
MSE score: 0.07563
Pearson score:
(0.02829031542243372, 0.64165856236096508)
MSE score: 0.06884
Pearson score:
(0.14240374677685239, 0.018567133643855472)
MSE score: 0.06156
Pearson score:
(0.076760888900166482, 0.20610630746887135)
```

```
MSE score: 0.12037
Pearson score:
(0.046400382536222591, 0.4451332272574442)
MSE score: 0.07661
Pearson score:
(-0.069890132426873725, 0.24978006455278851)
MSE score: 0.07392
Pearson score:
(0.28235658382488554, 2.1297649506873571e-06)
MSE score: 0.10926
Pearson score:
(0.26158556442520009, 1.1936018067333818e-05)
MSE score: 0.08284
Pearson score:
(0.19992558583025188, 0.00089453394608321067)
((154, 1583), (154, 1271), (154, 4096), (160, 9))
MSE score: 0.05944
Pearson score:
(0.40732426575141434, 1.5821190241266162e-07)
MSE score: 0.07970
Pearson score:
(-0.035825232393497865, 0.6591413701740898)
MSE score: 0.17537
Pearson score:
(0.025266467434962855, 0.75576740835089573)
MSE score: 0.15996
Pearson score:
(0.30104094128094189, 0.00014837496096601176)
MSE score: 0.08013
Pearson score:
(0.080731455491248258, 0.31958343243751375)
MSE score: 0.09423
Pearson score:
(0.10785995487255988, 0.18302802532816798)
MSE score: 0.10048
Pearson score:
(0.44011785895730571, 1.1201723040151303e-08)
MSE score: 0.16177
Pearson score:
(0.40689943728123984, 1.6343564906884298e-07)
((227, 1583), (227, 1271), (227, 4096), (235, 9))
MSE score: 0.04724
Pearson score:
(0.35377168881417381, 4.2800202992029276e-08)
MSE score: 0.07065
Pearson score:
(-0.26983819735527931, 3.7927215034571073e-05)
MSE score: 0.17374
Pearson score:
(0.041932133877991364, 0.52963710589027968)
MSE score: 0.19223
Pearson score:
(0.042500085962325247, 0.52407189365252227)
MSE score: 0.05489
```

```
Pearson score:
(0.034046182037472922, 0.60985980521500505)
MSE score: 0.05646
Pearson score:
(0.17713376534486658, 0.0074674890609173709)
MSE score: 0.18206
Pearson score:
(0.0016348018907735807, 0.98045792089677453)
MSE score: 0.26585
Pearson score:
(-0.22283610983870511, 0.00072105017174101159)
((127, 1583), (127, 1271), (127, 4096), (127, 9))
MSE score: 0.05000
Pearson score:
(-0.30028497516548674, 0.00060336959832762863)
MSE score: 0.02989
Pearson score:
(0.24386346077682647, 0.0057295670831301717)
MSE score: 0.04345
Pearson score:
(0.14787362629263315, 0.097093911976455755)
MSE score: 0.04050
Pearson score:
(0.24545662721679809, 0.0054109932260004963)
MSE score: 0.03263
Pearson score:
(0.16305501385957494, 0.067004091642123884)
MSE score: 0.02641
Pearson score:
(0.41036326080102264, 1.6544483021510906e-06)
MSE score: 0.03886
Pearson score:
(0.061042581532329311, 0.49539740310921798)
MSE score: 0.05551
Pearson score:
(-0.12261974694671249, 0.16963422946971396)
((22, 1583), (22, 1271), (22, 4096), (22, 9))
MSE score: 0.05998
Pearson score:
(0.87453483038493507, 1.0291974845428646e-07)
MSE score: 0.13872
Pearson score:
(0.73811276825773986, 8.8010073109813524e-05)
MSE score: 0.22162
Pearson score:
(0.7048899517085303, 0.00024914251936631053)
MSE score: 0.16986
Pearson score:
(-0.73601272225304648, 9.4412178541949525e-05)
MSE score: 0.08561
Pearson score:
(0.82125396113295634, 2.808631901789273e-06)
MSE score: 0.17973
Pearson score:
```

```
(0.60785583854109637, 0.0026914615344386101)
MSE score: 0.22122
Pearson score:
(-0.86014219572968043, 2.8645075638639252e-07)
MSE score: 0.18692
Pearson score:
(-0.49216882581131399, 0.01997822148104755)
((176, 1583), (176, 1271), (176, 4096), (176, 9))
MSE score: 0.03349
Pearson score:
(0.089676561938086222, 0.23657021850280163)
MSE score: 0.06777
Pearson score:
(-0.29275261395719271, 8.0514410060265768e-05)
MSE score: 0.05037
Pearson score:
(0.34715126831631465, 2.3549615462541535e-06)
MSE score: 0.14527
Pearson score:
(-0.16631276347574211, 0.027380261281863119)
MSE score: 0.03376
Pearson score:
(-0.31967545696531452, 1.5268012482538962e-05)
MSE score: 0.04135
Pearson score:
(-0.22263642525984428, 0.0029787068721266567)
MSE score: 0.10832
Pearson score:
(0.12967152340705909, 0.086294142962109507)
MSE score: 0.14556
Pearson score:
(-0.16815067369424655, 0.025695473923039461)
((196, 1583), (196, 1271), (196, 4096), (205, 9))
MSE score: 0.09833
Pearson score:
(-0.055415222383969877, 0.44044394167533951)
MSE score: 0.08251
Pearson score:
(0.27737056860584364, 8.2900646684541985e-05)
MSE score: 0.07491
Pearson score:
(0.23308217656186983, 0.0010101685136181665)
MSE score: 0.06244
Pearson score:
(0.23754461594940385, 0.00080094213335871254)
MSE score: 0.08946
Pearson score:
(0.49320801069083725, 2.069964460242544e-13)
MSE score: 0.09793
Pearson score:
(0.12442893117614635, 0.082277478253131114)
MSE score: 0.05590
Pearson score:
(0.40917633497449685, 2.6074809812683541e-09)
```

```
MSE score: 0.09845
Pearson score:
(0.023045392532959601, 0.74850262633705367)
((214, 1583), (214, 1271), (214, 4096), (214, 9))
MSE score: 0.03616
Pearson score:
(0.10696466487483963, 0.11874271549422261)
MSE score: 0.03470
Pearson score:
(0.40922380995245566, 4.7802045644312187e-10)
MSE score: 0.03130
Pearson score:
(0.091114370358508195, 0.18422917788509796)
MSE score: 0.04453
Pearson score:
(0.055159227449901073, 0.42209302088285128)
MSE score: 0.03490
Pearson score:
(0.058283943669042319, 0.3962440001826828)
MSE score: 0.04419
Pearson score:
(0.0035139214975311362, 0.95924318361578831)
MSE score: 0.02686
Pearson score:
(-0.0001903977591666206, 0.99779068756941336)
MSE score: 0.02498
Pearson score:
(0.23364719165953682, 0.00056945102547195539)
Train score: 0.95
Test score: -0.73
MSE score: 0.15579
Pearson score:
(0.39414440724472483, 6.1026278496011477e-55)
Train score: 0.93
Test score: -0.43
MSE score: 0.12825
Pearson score:
(0.21511331755321744, 1.3408213910914969e-16)
Train score: 0.92
Test score: -0.12
MSE score: 0.09909
Pearson score:
(0.10948145105830284, 3.0177650003715585e-05)
Train score: 0.90
Test score: -0.01
MSE score: 0.08952
Pearson score:
(0.19380449143063322, 1.0545379712796257e-13)
Train score: 0.93
Test score: -0.62
MSE score: 0.14586
Pearson score:
(0.095346105486618565, 0.00028254512484677364)
Train score: 0.91
```

```
Test score: -0.07
MSE score: 0.09496
Pearson score:
(-0.040239395370524941, 0.12615248728074427)
Train score: 0.90
Test score: -0.68
MSE score: 0.15131
Pearson score:
(0.18342582760579484, 2.0848276070709078e-12)
Train score: 0.90
Test score: -0.24
MSE score: 0.11005
Pearson score:
(-0.053048318253613468, 0.043705185306607865)
((139, 1583), (139, 1271), (139, 4096), (139, 9))
MSE score: 0.06289
Pearson score:
(0.36283179693702955, 1.1354250982441104e-05)
MSE score: 0.15267
Pearson score:
(-0.022242876138236431, 0.79493804611486207)
MSE score: 0.06140
Pearson score:
(0.17234061966195235, 0.042486469407986373)
MSE score: 0.12394
Pearson score:
(0.055992041163541527, 0.51266792940842598)
MSE score: 0.07210
Pearson score:
(0.066702609035677277, 0.43528487536320848)
MSE score: 0.11282
Pearson score:
(0.030283573286611343, 0.72341748577116394)
MSE score: 0.06557
Pearson score:
(0.12498616725270066, 0.14264457350156826)
MSE score: 0.09863
Pearson score:
(-0.12893018151680136, 0.13036639023924632)
((99, 1583), (99, 1271), (99, 4096), (99, 9))
MSE score: 0.12847
Pearson score:
(-0.088701989997457195, 0.38261540815022654)
MSE score: 0.05242
Pearson score:
(0.08277428482448751, 0.4153396726259172)
MSE score: 0.19348
Pearson score:
(0.28561564811192447, 0.0041609236696977508)
MSE score: 0.21720
Pearson score:
(0.2578497801149881, 0.0099743424486303644)
MSE score: 0.07541
Pearson score:
```

```
(-0.30225167830766259, 0.0023608797439386704)
MSE score: 0.06928
Pearson score:
(-0.16272217005515383, 0.1075621775103697)
MSE score: 0.19488
Pearson score:
(0.66204300889721801, 8.5511305178269216e-14)
MSE score: 0.23201
Pearson score:
(-0.094330301287806437, 0.3530302842563755)
((179, 1583), (179, 1271), (179, 4096), (187, 9))
MSE score: 0.05157
Pearson score:
(-0.18123691430984118, 0.015184916249232898)
MSE score: 0.06642
Pearson score:
(0.23844777828522024, 0.0013073685642818634)
MSE score: 0.15107
Pearson score:
(-0.10140710186179246, 0.17678702050551703)
MSE score: 0.08034
Pearson score:
(-0.23599759487181365, 0.0014709346493365146)
MSE score: 0.05767
Pearson score:
(0.40021522774977059, 2.8442722591197898e-08)
MSE score: 0.09156
Pearson score:
(-0.30399160362207572, 3.5192159821191056e-05)
MSE score: 0.17037
Pearson score:
(-0.10370629267886801, 0.16712432384094145)
MSE score: 0.21739
Pearson score:
(0.10564808776884538, 0.15927479901415986)
((145, 1583), (146, 1271), (146, 4096), (146, 9))
MSE score: 0.03909
Pearson score:
(0.50481231750891764, 9.5071935898563727e-11)
MSE score: 0.04869
Pearson score:
(0.31974981830974175, 8.8401329585924114e-05)
MSE score: 0.08396
Pearson score:
(-0.23442329170487922, 0.0045389174780949204)
MSE score: 0.08182
Pearson score:
(0.070809958088818903, 0.39735865166438866)
MSE score: 0.04977
Pearson score:
(0.18329560725337826, 0.027328064364171272)
MSE score: 0.06514
Pearson score:
(-0.049875924431213928, 0.55133656501821604)
```

```
MSE score: 0.04400
Pearson score:
(0.36621705083942685, 5.8967338891439374e-06)
MSE score: 0.05390
Pearson score:
(0.19445371080655638, 0.019094342129135296)
((176, 1583), (176, 1271), (176, 4096), (176, 9))
MSE score: 0.05496
Pearson score:
(0.56474306006691355, 3.2470621602058288e-16)
MSE score: 0.09542
Pearson score:
(0.30750453945025363, 3.3033075990804747e-05)
MSE score: 0.08829
Pearson score:
(0.44485477893765091, 6.1860741165243507e-10)
MSE score: 0.11876
Pearson score:
(0.20926153457582539, 0.0053146881931174939)
MSE score: 0.10687
Pearson score:
(0.29633784758982606, 6.5125700814423011e-05)
MSE score: 0.10736
Pearson score:
(0.35229803320654929, 1.6259975486415135e-06)
MSE score: 0.09571
Pearson score:
(0.23329330278922897, 0.0018329489211602247)
MSE score: 0.08515
Pearson score:
(0.39542944948313247, 5.5849075087136093e-08)
((61, 1583), (61, 1271), (61, 4096), (61, 9))
MSE score: 0.36022
Pearson score:
(0.28527055724885353, 0.025849223112067918)
MSE score: 0.15018
Pearson score:
(-0.44252953016187729, 0.00035555733012280095)
MSE score: 0.06454
Pearson score:
(-0.38791509345138764, 0.0020085250193224466)
MSE score: 0.17614
Pearson score:
(-0.063992560243149962, 0.62416161951879001)
MSE score: 0.24390
Pearson score:
(0.25326891052280215, 0.048906044874430475)
MSE score: 0.14880
Pearson score:
(0.010996834043365109, 0.93296604052005738)
MSE score: 0.15834
Pearson score:
(0.18811854741726738, 0.14654196240901152)
MSE score: 0.10348
```

```
Pearson score:
(-0.13895345234059822, 0.28552009465852146)
((129, 1583), (129, 1271), (129, 4096), (129, 9))
MSE score: 0.03344
Pearson score:
(0.53042825853446263, 1.0116690837582732e-10)
MSE score: 0.08780
Pearson score:
(-0.31461141532306491, 0.0002821350622801025)
MSE score: 0.10724
Pearson score:
(-0.16898167311440412, 0.055572565495045287)
MSE score: 0.20116
Pearson score:
(-0.06743777474813524, 0.44763983311491573)
MSE score: 0.05937
Pearson score:
(0.27877723757357215, 0.0013783289281870652)
MSE score: 0.05438
Pearson score:
(0.12114458420633573, 0.17144107799508579)
MSE score: 0.08556
Pearson score:
(-0.25934630458854707, 0.0029984997513707525)
MSE score: 0.13558
Pearson score:
(-0.57851443224746046, 6.9646300376854355e-13)
((77, 1583), (77, 1271), (77, 4096), (77, 9))
MSE score: 0.05858
Pearson score:
(-0.17844322116946726, 0.12049189892260256)
MSE score: 0.03214
Pearson score:
(0.73040187573572357, 4.7532541007888531e-14)
MSE score: 0.06190
Pearson score:
(0.64041998314387039, 3.5634558743709561e-10)
MSE score: 0.08350
Pearson score:
(0.11351500916994317, 0.3256174818392541)
MSE score: 0.04937
Pearson score:
(0.25913438659335963, 0.02286243132385363)
MSE score: 0.06919
Pearson score:
(-0.34166517542444802, 0.0023568278889846664)
MSE score: 0.11607
Pearson score:
(0.14402018479648418, 0.21143556444704967)
MSE score: 0.05039
Pearson score:
(0.29966684159520812, 0.0081041278711878554)
((118, 1583), (118, 1271), (118, 4096), (118, 9))
MSE score: 0.09011
```

```
Pearson score:
(-0.045715508183137278, 0.62302457326851179)
MSE score: 0.10848
Pearson score:
(0.2758736657709393, 0.0024965463263193362)
MSE score: 0.23815
Pearson score:
(-0.10779570948545088, 0.24528213738002927)
MSE score: 0.18558
Pearson score:
(-0.46750047270940143, 9.4351016455057917e-08)
MSE score: 0.12246
Pearson score:
(-0.55478117703381424, 7.0927253278502269e-11)
MSE score: 0.11398
Pearson score:
(-0.011243037467491879, 0.90382223999871281)
MSE score: 0.22220
Pearson score:
(0.075619745884499254, 0.41572372843638206)
MSE score: 0.26696
Pearson score:
(-0.12216846064874688, 0.18753560919748435)
((100, 1583), (101, 1271), (101, 4096), (101, 9))
MSE score: 0.18301
Pearson score:
(-0.25934498582528842, 0.0091717193062066677)
MSE score: 0.17725
Pearson score:
(0.091802500348666105, 0.36366517153498779)
MSE score: 0.09678
Pearson score:
(-0.49945135167246846, 1.2247837269889911e-07)
MSE score: 0.09801
Pearson score:
(-0.011890260785346364, 0.90653369925194027)
MSE score: 0.15962
Pearson score:
(0.30148432444867318, 0.0023022012841423484)
MSE score: 0.18868
Pearson score:
(0.14501972845535649, 0.14998432354476599)
MSE score: 0.07728
Pearson score:
(0.30407612441807225, 0.002100031272011148)
MSE score: 0.10581
Pearson score:
(-0.32014000745932908, 0.0011660630555992495)
((223, 1583), (223, 1271), (223, 4096), (230, 9))
MSE score: 0.08924
Pearson score:
(0.23226591385080897, 0.00047065907728965418)
MSE score: 0.11497
Pearson score:
```

```
(0.11624906391693454, 0.083257539404044706)
MSE score: 0.16702
Pearson score:
(-0.10384799422381116, 0.12204429000476036)
MSE score: 0.29688
Pearson score:
(-0.016914609550078737, 0.8016691986387896)
MSE score: 0.10148
Pearson score:
(0.15859729775842893, 0.017784577155126921)
MSE score: 0.12430
Pearson score:
(0.032354164536643096, 0.63082832779207043)
MSE score: 0.26428
Pearson score:
(0.077561960122240509, 0.24871492936134382)
MSE score: 0.17059
Pearson score:
(0.35920406103231772, 3.40500043748225e-08)
Train score: 0.95
Test score: -0.37
MSE score: 0.16040
Pearson score:
(0.0042043202446156837, 0.8738751747262512)
Train score: 0.93
Test score: -0.02
MSE score: 0.11934
Pearson score:
(0.23771484929508557, 8.5516100377675683e-20)
Train score: 0.93
Test score: -0.20
MSE score: 0.08122
Pearson score:
(0.077218695744500798, 0.0035024076098140937)
Train score: 0.91
Test score: -0.11
MSE score: 0.07486
Pearson score:
(0.14838294935332527, 1.7655914702007081e-08)
Train score: 0.94
Test score: -0.05
MSE score: 0.12279
Pearson score:
(0.085076467086707033, 0.0012910344457517514)
Train score: 0.91
Test score: -0.22
MSE score: 0.08207
Pearson score:
(-0.036324800174532212, 0.17008694710401862)
Train score: 0.91
Test score: -0.30
MSE score: 0.15229
Pearson score:
(0.069741356455358547, 0.0083803034525968163)
```

```
Train score: 0.90
Test score: -0.30
MSE score: 0.08774
Pearson score:
(0.078327498771331314, 0.0030577849344160303)
((33, 1583), (33, 1271), (33, 4096), (35, 9))
MSE score: 0.01418
Pearson score:
(-0.20200942469769903, 0.25958117042273687)
MSE score: 0.00873
Pearson score:
(-0.43563233680425806, 0.011276305399354012)
MSE score: 0.34833
Pearson score:
(0.27333899693895258, 0.12376835903718611)
MSE score: 0.18632
Pearson score:
(0.89572367239943573, 1.9265810751456717e-12)
MSE score: 0.00944
Pearson score:
(-0.29406987511702665, 0.096688163365367216)
MSE score: 0.03180
Pearson score:
(-0.15920977331426431, 0.37616239388645034)
MSE score: 0.27773
Pearson score:
(0.85932401063345187, 1.5362563961364059e-10)
MSE score: 0.21833
Pearson score:
(0.36466822460325449, 0.036929104545975693)
((81, 1583), (82, 1271), (82, 4096), (85, 9))
MSE score: 0.19911
Pearson score:
(-0.36867291098376387, 0.00070760864507676217)
MSE score: 0.20454
Pearson score:
(-0.055362771872635892, 0.62349910522344443)
MSE score: 0.13352
Pearson score:
(0.00065436931681591933, 0.99537406171090348)
MSE score: 0.21225
Pearson score:
(-0.17916286285052804, 0.10951386600597236)
MSE score: 0.23064
Pearson score:
(-0.2301713694637334, 0.038716395357690614)
MSE score: 0.20337
Pearson score:
(0.20304106652227466, 0.069071198615763857)
MSE score: 0.14051
Pearson score:
(-0.36932335860950638, 0.00069119183630031477)
MSE score: 0.15708
Pearson score:
```

```
(-0.38604698608339971, 0.00037161797296458182)
((339, 1583), (339, 1271), (339, 4096), (353, 9))
MSE score: 0.04277
Pearson score:
(0.32715006514883138, 6.7483521640292066e-10)
MSE score: 0.06556
Pearson score:
(-0.10519002638909947, 0.05299379252050386)
MSE score: 0.10344
Pearson score:
(0.051892473837656092, 0.34081574299060746)
MSE score: 0.09518
Pearson score:
(0.11774579610700836, 0.030199222685427875)
MSE score: 0.06977
Pearson score:
(-0.19883909216170206, 0.00022928802983559962)
MSE score: 0.05114
Pearson score:
(0.23624820503858415, 1.102082881100707e-05)
MSE score: 0.09222
Pearson score:
(0.082625277866016783, 0.12894615096055609)
MSE score: 0.09143
Pearson score:
(0.12679726598880767, 0.019522750950105069)
((116, 1583), (117, 1271), (117, 4096), (122, 9))
MSE score: 0.06685
Pearson score:
(0.15182825602995079, 0.10374175712401552)
MSE score: 0.06251
Pearson score:
(-0.14380225249744769, 0.12355044501109695)
MSE score: 0.07369
Pearson score:
(-0.19956098312196449, 0.031736531382892889)
MSE score: 0.33684
Pearson score:
(-0.27446156729345317, 0.0028683541083676344)
MSE score: 0.06116
Pearson score:
(0.11646344445438765, 0.21312915780277245)
MSE score: 0.07566
Pearson score:
(-0.077405276236125475, 0.40886538142413864)
MSE score: 0.11864
Pearson score:
(-0.029197993725653567, 0.75569971399968028)
MSE score: 0.18217
Pearson score:
(-0.19891782292124463, 0.032302305498196918)
((157, 1583), (157, 1271), (157, 4096), (157, 9))
MSE score: 0.08745
Pearson score:
```

```
(-0.049387410781277376, 0.5390485411809387)
MSE score: 0.07941
Pearson score:
(0.3583709378841452, 4.0558260211357248e-06)
MSE score: 0.12641
Pearson score:
(-0.30187254146385051, 0.00012190905774430259)
MSE score: 0.13742
Pearson score:
(0.33802836844774486, 1.4925541071008132e-05)
MSE score: 0.08090
Pearson score:
(0.17489529435316697, 0.02846378470629626)
MSE score: 0.08742
Pearson score:
(-0.040227890306332878, 0.61691409473778669)
MSE score: 0.08177
Pearson score:
(-0.013529800489234304, 0.86644097061758818)
MSE score: 0.10532
Pearson score:
(-0.0052689065998956586, 0.9477821986943068)
((48, 1583), (49, 1271), (49, 4096), (50, 9))
MSE score: 0.17952
Pearson score:
(-0.21500600326219518, 0.14222242428408402)
MSE score: 0.14103
Pearson score:
(0.40804319899222824, 0.0039899740852807021)
MSE score: 0.05722
Pearson score:
(0.43146586998712416, 0.0021995250712319348)
MSE score: 0.19605
Pearson score:
(-0.027684949227296033, 0.85182788082142147)
MSE score: 0.14294
Pearson score:
(0.54711917849413993, 5.7234363379061014e-05)
MSE score: 0.19289
Pearson score:
(0.068843717037774352, 0.6419723171697076)
MSE score: 0.10774
Pearson score:
(-0.029277480173052674, 0.8434082848432336)
MSE score: 0.14546
Pearson score:
(0.19652987299501537, 0.18063363519484976)
((273, 1583), (273, 1271), (273, 4096), (282, 9))
MSE score: 0.06545
Pearson score:
(0.13029933291882084, 0.031381343453184225)
MSE score: 0.05944
Pearson score:
(0.30095968921839417, 4.0236659329317112e-07)
```

```
MSE score: 0.05195
Pearson score:
(0.22083565669811267, 0.00023546849005781439)
MSE score: 0.07139
Pearson score:
(-0.034498901936185183, 0.57032771805949678)
MSE score: 0.06381
Pearson score:
(0.044504985779080845, 0.46396599766897539)
MSE score: 0.06983
Pearson score:
(0.36712489309177015, 3.9009100252012744e-10)
MSE score: 0.10426
Pearson score:
(0.24960342791221107, 3.0271265707097501e-05)
MSE score: 0.06734
Pearson score:
(0.18830276891295405, 0.0017780587729148503)
((154, 1583), (154, 1271), (154, 4096), (160, 9))
MSE score: 0.06454
Pearson score:
(0.35301727294949381, 7.1030535524043183e-06)
MSE score: 0.09511
Pearson score:
(-0.12024791933682211, 0.1374179863040895)
MSE score: 0.16642
Pearson score:
(0.15998444093479011, 0.047483230414478057)
MSE score: 0.18846
Pearson score:
(0.37901555529350489, 1.251962614817273e-06)
MSE score: 0.08796
Pearson score:
(0.062991301482951706, 0.43768766665968728)
MSE score: 0.11121
Pearson score:
(-0.17710632801596765, 0.027997192664397659)
MSE score: 0.09390
Pearson score:
(0.533713961822044, 1.024082244551467e-12)
MSE score: 0.20154
Pearson score:
(0.32939375294909207, 3.0275638602599794e-05)
((227, 1583), (227, 1271), (227, 4096), (235, 9))
MSE score: 0.05039
Pearson score:
(0.24416092206439396, 0.00020342466126927088)
MSE score: 0.05968
Pearson score:
(0.044959179587930137, 0.50032421738345678)
MSE score: 0.16606
Pearson score:
(0.094109203627932375, 0.15759435009880604)
MSE score: 0.20270
```

```
Pearson score:
(-0.17314516547800268, 0.0089471561100980448)
MSE score: 0.05966
Pearson score:
(-0.057374724495433585, 0.38958282620358231)
MSE score: 0.04669
Pearson score:
(0.52344677908996484, 2.275969364460074e-17)
MSE score: 0.19318
Pearson score:
(-0.060541130766789908, 0.36390904559134585)
MSE score: 0.26780
Pearson score:
(-0.17096838981053336, 0.0098598224502375719)
CPU times: user 1h 27min 30s, sys: 1min 39s, total: 1h 29min 10s
Wall time: 1h 29min 1s
```

2.7 Evaluation results

```
In [80]: evaldf = pd.read_csv('all_metrics_stats.csv')

In [81]: evaldf.columns = [c.replace('Mean','') for c in evaldf.columns ]

In [82]: evaldf.set_index('Unnamed: 0',inplace=True)
         evaldf.index.name = None

In [83]: armse = [f for f in evaldf.columns if ('Arousal' in f)  and ('MSE' in f) ]
         arpcc = [f for f in evaldf.columns if ('Arousal' in f)  and ('PCC' in f) ]

In [84]: vlmse = [f for f in evaldf.columns if 'Valence' in f   and ('MSE' in f) ]
         vlpcc = [f for f in evaldf.columns if 'Valence' in f   and ('PCC' in f) ]

In [85]: vlmse,armse

Out[85]: (['ValenceVisualMSE',
           'ValenceAudioMSE',
           'ValenceLowLevelMSE',
           'ValenceDeepMSE'],
          ['ArousalAudioMSE',
           'ArousalVisualMSE',
           'ArousalLowLevelMSE',
           'ArousalDeepMSE'])

In [86]: evaldf[vlmse].transpose()[['mean','std']]

Out[86]:                         mean       std
         ValenceVisualMSE    0.154086  0.088704
         ValenceAudioMSE     0.117275  0.075583
         ValenceLowLevelMSE  0.139978  0.071942
         ValenceDeepMSE      0.125003  0.066658

In [87]: evaldf[vlpcc].transpose()[['mean','std']]

Out[87]:                         mean       std
         ValenceVisualPCC    0.017005  0.239715
         ValenceAudioPCC     0.065450  0.294743
         ValenceLowLevelPCC  0.021252  0.267876
         ValenceDeepPCC      0.133248  0.294210
```

In [88]:
```
evaldf[armse].transpose()[['mean','std']]
```

Out[88]:

	mean	std
ArousalAudioMSE	0.086127	0.072037
ArousalVisualMSE	0.091809	0.054271
ArousalLowLevelMSE	0.097695	0.054281
ArousalDeepMSE	0.088623	0.058237

In [89]:
```
evaldf[arpcc].transpose()[['mean','std']]
```

Out[89]:

	mean	std
ArousalAudioPCC	0.152889	0.297723
ArousalVisualPCC	0.080043	0.329962
ArousalLowLevelPCC	0.037407	0.233049
ArousalDeepPCC	0.156133	0.282591

In [ ]:

In [ ]:

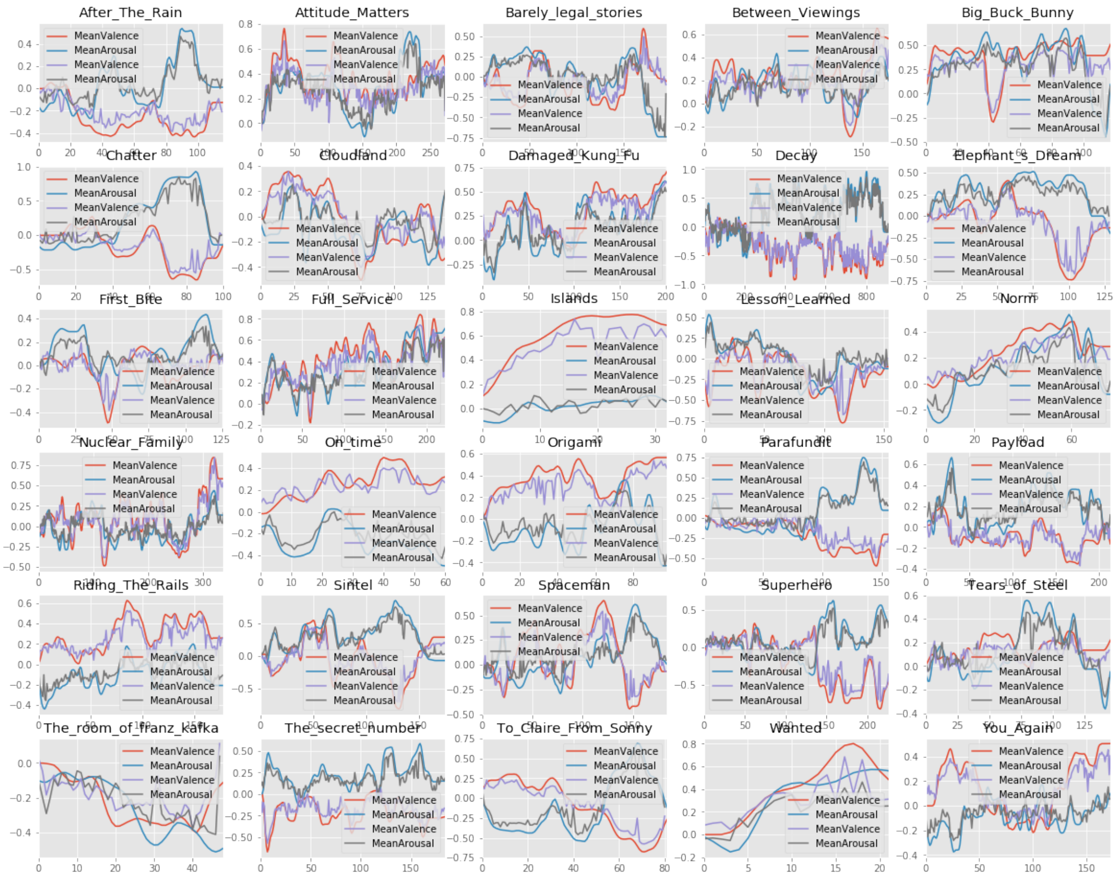
In [66]:
```
#test
f = "Wanted"
audiodf = getAudioDf(f)
visualdf = getAvgVisFeatListDf(f,visual_feat_list)
print(audiodf.shape,visualdf.shape)

mlen = min(len(audiodf),len(visualdf))

audiodf = audiodf[:mlen]
visualdf = visualdf[:mlen]

aa = paa.predict(audiodf)
av = pav.predict(visualdf)
arousal_scores = np.transpose([ aa,av ])


va = pvv.predict(audiodf)  ## look up this is twisted
vv = pva.predict(visualdf)
valence_scores = np.transpose([va,vv ])

df =pd.DataFrame(np.transpose([ aa,av , va,vv ])) #, columns=['MeanValence','MeanArousal'])
```

```
(22, 1583) (22, 1271)
```

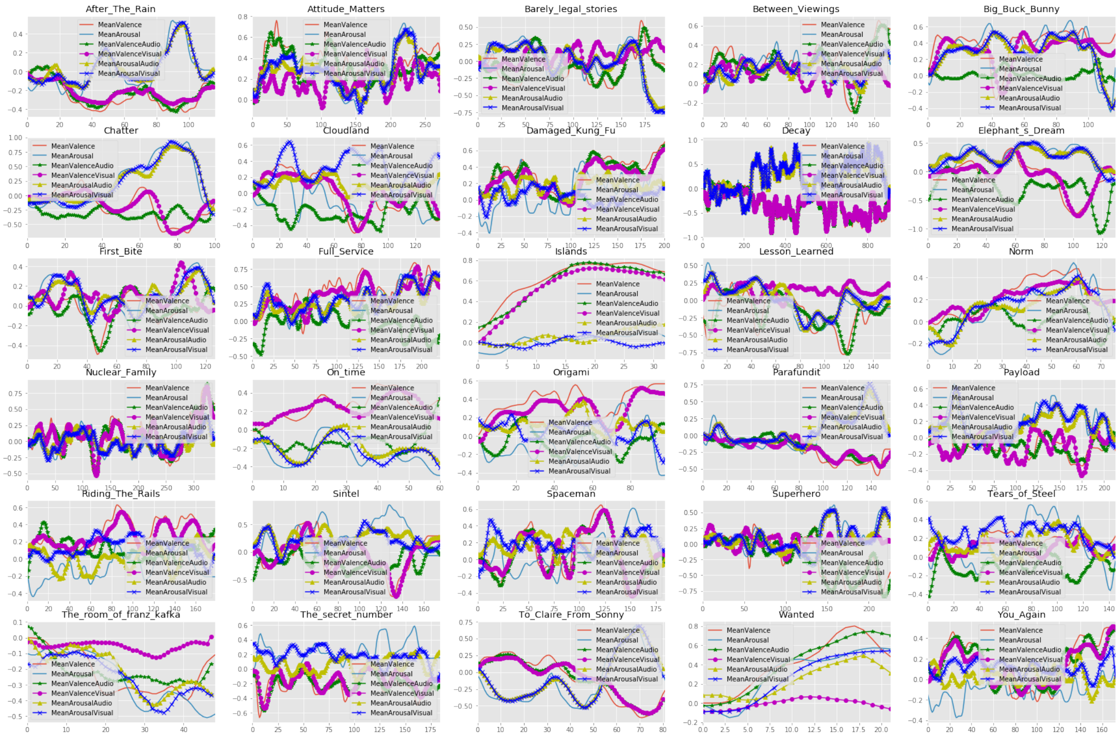
2.8 Visualization

It looks like the pipe are successfully predict the movie "Decay", since it was in all the traiing sets. however

In [185]:
```
fix, axes = plt.subplots(figsize=(20,16))
for ii, mov in enumerate(movieNames):
    plt.subplot(6,5,ii+1)
    dfa = getAnnotationDf(mov)
    dfa[['MeanValence','MeanArousal']].plot(ax=plt.gca(),title=mov)
    dev_set[mov][['MeanValence','MeanArousal']].plot(ax=plt.gca(),title=mov)
```




```
In [74]: fix, axes = plt.subplots(figsize=(30,20))
         for ii, mov in enumerate(movieNames):
             plt.subplot(6,5,ii+1)
             dfa = getAnnotationDf(mov)
             dfa[['MeanValence','MeanArousal']].plot(ax=plt.gca(),title=mov)
             idev_set[mov].plot(ax=plt.gca(),title=mov,style=['g*-','mo-','y^-','bx-'])
```


After_The_Rain
Attitude_Matters
Barely_legal_stories
Between_Viewings
Big_Buck_Bunny
Chatter
Cloudland
Damaged_Kung_Fu
Decay
Elephant_s_Dream
First_Bite
Full_Service
Islands
Lesson_Learned
Norm
Nuclear_Family
On_time
Origami
Parafundit
Payload
Riding_The_Rails
Sintel
Spaceman
Superhero
Tears_of_Steel
The_room_of_franz_kafka
The_secret_number
To_Claire_From_Sonny
Wanted
You_Again
MeanValence
MeanArousal
MeanValenceAudio
MeanValenceVisual
MeanArousalAudio
MeanArousalVisual