CmpE58Y - Affordance Learning

March 26, 2018

Mine Melodi Caliskan - 2015705009

In this project, we simulate the affordance learning approach for goal emulation and planning in perceptual space.

In the first step of learning,

- We simulate the robot's discovery of commonalities in its action-effect experiences by discovering effect categories.

In the second step,

- Affordance predictors for each behavior are obtained by learning the mapping from the object features to the effect categories.

0.1 The unsupervised discovery of effect categories

For clustering X-means algorithm in WEKA software is used.

Clustering is done in channels and then cross product of clusters assigned as clusters for all channels.

```
In [16]: import pandas as pd
         from scipy.io import arff
         from io import StringIO
         import matplotlib.pyplot as plt
         import numpy as np
         import sklearn
```

Push-right Behavior

Effect_0 (Behaivor 0) clustering results for visibility, first column in the data set.

```
In [17]: vis_e0, meta_e0 = arff.loadarff(open('resvis0.arff'))
```

```
In [18]: df_vis0=pd.DataFrame(vis_e0)
         df_vis0['Cluster']=df_vis0['Cluster'].apply(lambda x: x.decode('UTF-8')).str.extract(
         df_vis0=df_vis0.drop('Instance_number',axis=1)
         df_vis0=df_vis0.rename(columns={'Cluster':'ClusterVis'})
```

```
In [19]: df_vis0.head()
```

```
Out[19]:    f_000  ClusterVis
         0   -1.0           0
         1   -1.0           0
         2    0.0           1
         3    0.0           1
         4    0.0           1
```

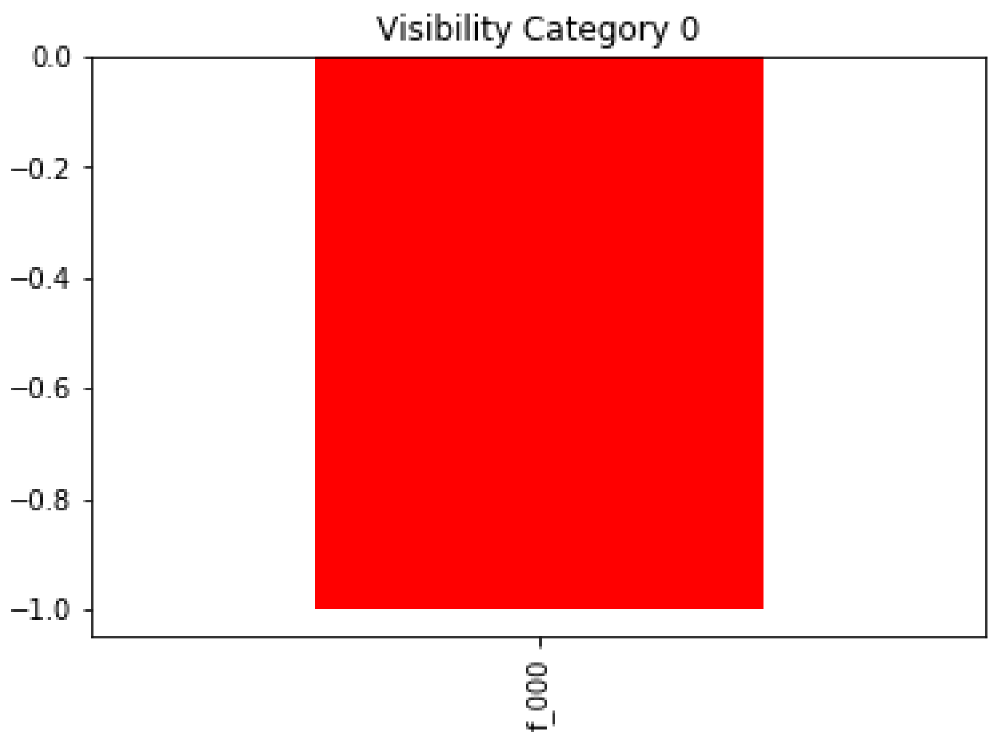
There are 2 categories for visibility in effect_0:

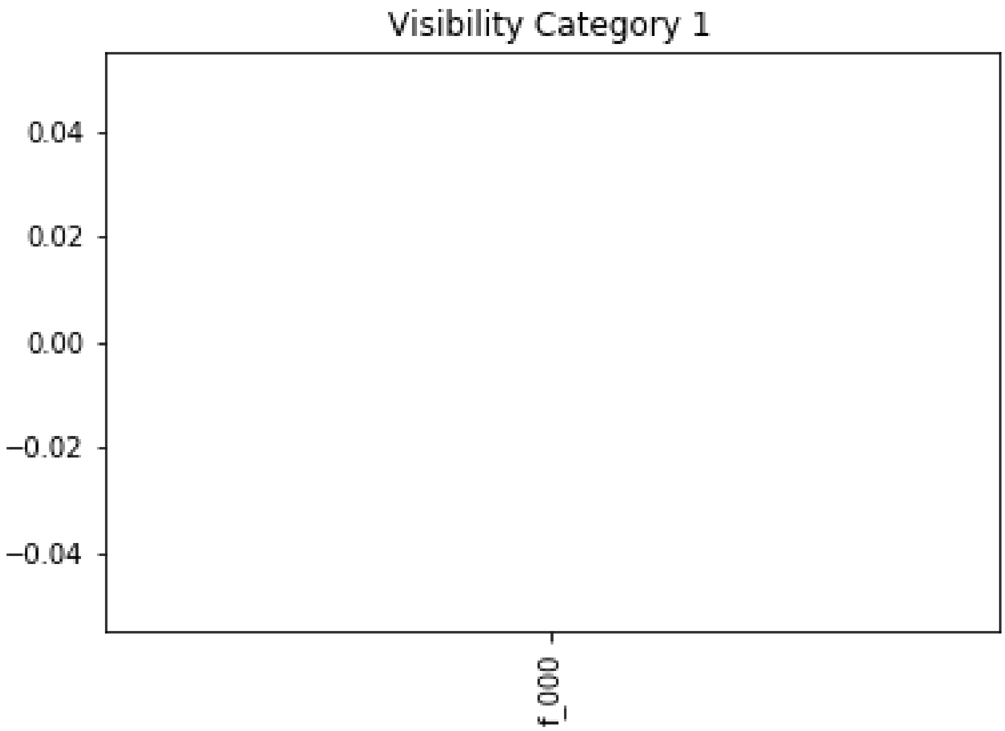
```
In [20]: set(df_vis0['ClusterVis'])
```

```
Out[20]: {0, 1}
```

The prototype visibility effect vectors:

```
In [21]: vis0plt=df_vis0
         for c in [0,1]:
             prototype_visibility_vectors= vis0plt[vis0plt['ClusterVis']==c].drop('ClusterVis'
             prototype_visibility_vectors.plot(kind='bar',color='red')
             plt.title('Visibility Category {}'.format(c))
             plt.show()
```





Effect_0 clustering results for position features, 2-7 columns in the data set.

```
In [22]: pos_e0, meta_e0 = arff.loadarff(open('respos0.arff'))
```

```
In [23]: df_pos0=pd.DataFrame(pos_e0)
         df_pos0['Cluster']=df_pos0['Cluster'].apply(lambda x: x.decode('UTF-8')).str.extract(
         df_pos0=df_pos0.drop('Instance_number',axis=1)
         df_pos0=df_pos0.rename(columns={'Cluster':'ClusterPos'})
```

```
In [24]: df_pos0.head()
```

Out[24]:

	f_001	f_002	f_003	f_004	f_005	f_006	ClusterPos
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2
2	-0.004347	-0.000436	0.000086	0.006332	0.122859	0.125860	1
3	-0.000594	-0.002421	-0.000013	0.002235	0.122554	0.124156	1
4	0.002804	0.001867	0.000109	0.004191	0.123729	0.118895	1

There are 4 clusters for position features in effect_0:

```
In [25]: set(df_pos0['ClusterPos'])
```

```
Out[25]: {0, 1, 2, 3}
```

```
In [26]: for cluster in set(df_pos0['ClusterPos']):
             print('Percentage of ', cluster, ':',
                   (df_pos0[df_pos0['ClusterPos']==cluster].count()/df_pos0.count()).mean())
```

```
Percentage of  0 : 0.3672365497923308
Percentage of  1 : 0.3553559354776393
Percentage of  2 : 0.26620303293731284
Percentage of  3 : 0.011204481792717087
```

We can drop cluster3, since it's percentage is too low.

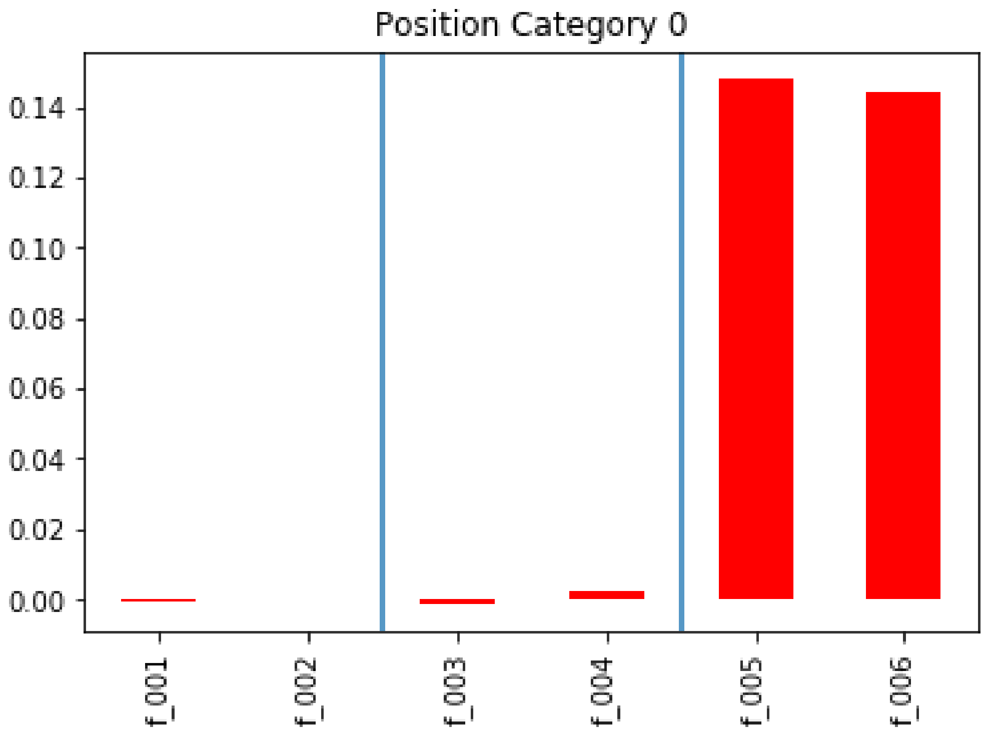
```
In [27]: df_pos0filtered=df_pos0[df_pos0['ClusterPos'].isin([0,1,2])]
```

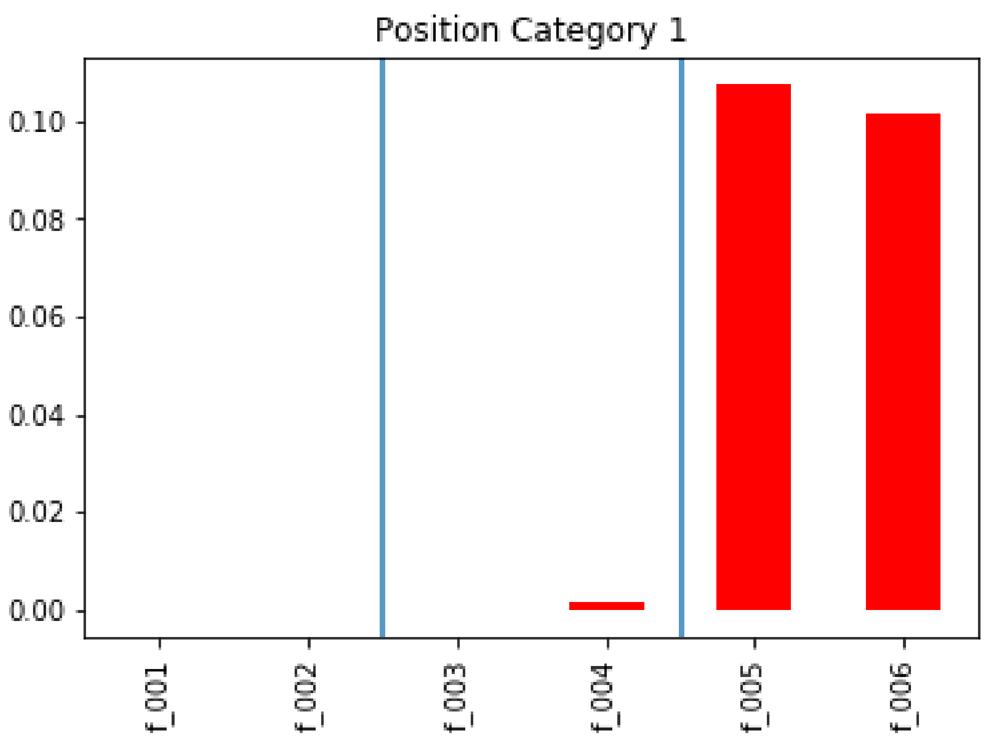
```
In [28]: df_pos0filtered.describe()
```

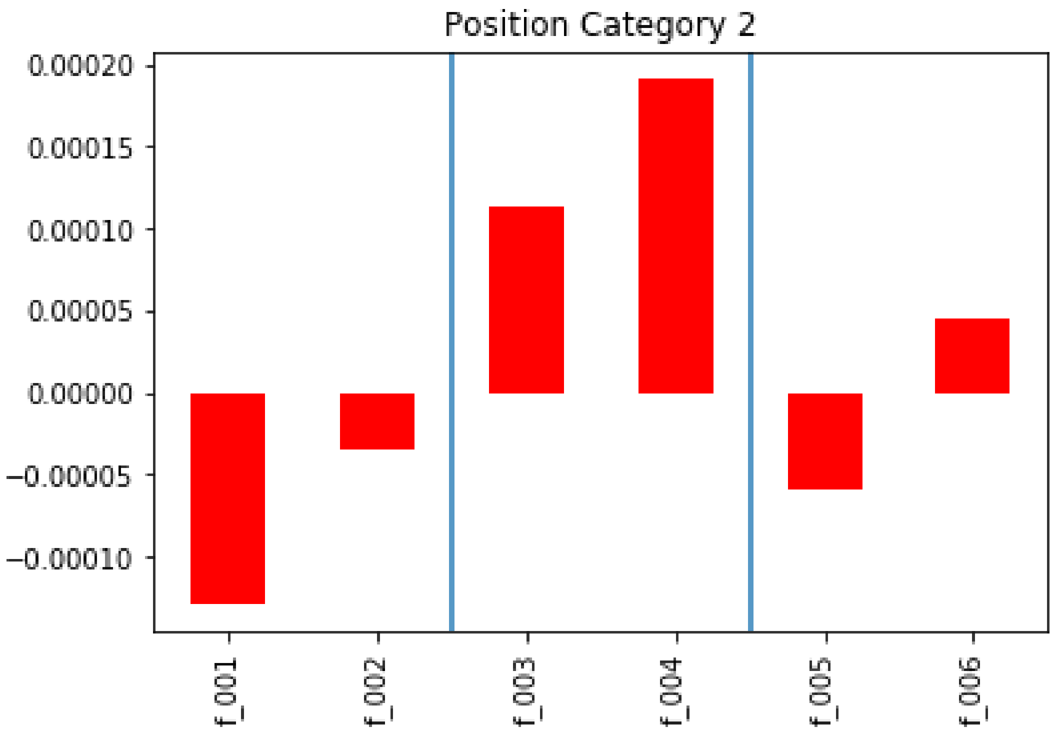
```
Out[28]:                f_001          f_002          f_003          f_004          f_005  \
         count  10237.000000  10237.000000  10237.000000  10237.000000  10237.000000
         mean      -0.000287     -0.000175     -0.000609      0.001492      0.093798
         std        0.003808      0.003821      0.007857      0.008740      0.060750
         min       -0.162622     -0.124154     -0.209760     -0.199112     -0.051552
         25%       -0.001637     -0.001008     -0.001927     -0.001769      0.002526
         50%        0.000000      0.000000      0.000000      0.000000      0.113892
         75%        0.001139      0.001083      0.001258      0.003926      0.140627
         max        0.014549      0.020093      0.125366      0.169477      0.208361

                       f_006    ClusterPos
         count  10237.000000  10237.000000
         mean       0.090207      0.897822
         std        0.059129      0.793876
         min       -0.052509      0.000000
         25%        0.004812      0.000000
         50%        0.108949      1.000000
         75%        0.136657      2.000000
         max        0.213719      2.000000
```

The prototype effect vectors are computed as the average of category members and their plots are shown below.

```
In [29]: pos0plt=df_pos0filtered
         for c in [0,1,2]:
             prototype_position_vectors=pos0plt[pos0plt['ClusterPos']==c].drop('ClusterPos',ax
             prototype_position_vectors = prototype_position_vectors.mean()
             prototype_position_vectors.plot(kind='bar',color='red')
             plt.title('Position Category {}'.format(c))
             plt.axvline(1.5)
             plt.axvline(3.5)
             plt.show()
```


Position Category 0
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
f_001
f_002
f_003
f_004
f_005
f_006


Position Category 1
0.10
0.08
0.06
0.04
0.02
0.00
f_001
f_002
f_003
f_004
f_005
f_006



Effect_0 clustering results for shape features, 8-42 columns in the data set.

```
In [30]: shape_e0, meta_e0 = arff.loadarff(open('resshape0.arff'))

In [31]: df_shape0=pd.DataFrame(shape_e0)
         df_shape0['Cluster']=df_shape0['Cluster'].apply(lambda x: x.decode('UTF-8')).str.extra
         df_shape0=df_shape0.drop('Instance_number',axis=1)
         df_shape0=df_shape0.rename(columns={'Cluster':'ClusterShape'})

In [32]: df_shape0.head()

Out[32]:       f_007     f_008     f_009     f_010     f_011  f_012  f_013  f_014  \
         0  0.000000  0.000000  0.000000  0.000000  0.000000    0.0    0.0    0.0
         1  0.000000  0.000000  0.000000  0.000000  0.000000    0.0    0.0    0.0
         2  0.010084 -0.008318 -0.009685 -0.013715  0.016847    0.0    0.0    0.0
         3  0.011869  0.011687 -0.004677  0.003745 -0.014930    0.0    0.0    0.0
         4  0.023592  0.015878 -0.017123 -0.013989 -0.000454    0.0    0.0    0.0

            f_015  f_016      ...          f_034  f_035  f_036  f_037  f_038  f_039  \
         0    0.0    0.0      ...       0.000000    0.0    0.0    0.0    0.0    0.0
         1    0.0    0.0      ...       0.000000    0.0    0.0    0.0    0.0    0.0
         2    0.0    0.0      ...       0.008019    0.0    0.0    0.0    0.0    0.0
         3    0.0    0.0      ...       0.000903    0.0    0.0    0.0    0.0    0.0
         4    0.0    0.0      ...       0.000478    0.0    0.0    0.0    0.0    0.0
```

```
   f_040  f_041  f_042  ClusterShape
0    0.0    0.0    0.0             1
1    0.0    0.0    0.0             1
2    0.0    0.0    0.0             2
3    0.0    0.0    0.0             2
4    0.0    0.0    0.0             1

[5 rows x 37 columns]
```

There are 3 clusters for shape features of effect_0:

```
In [33]: set(df_shape0['ClusterShape'])
```

```
Out[33]: {0, 1, 2}
```

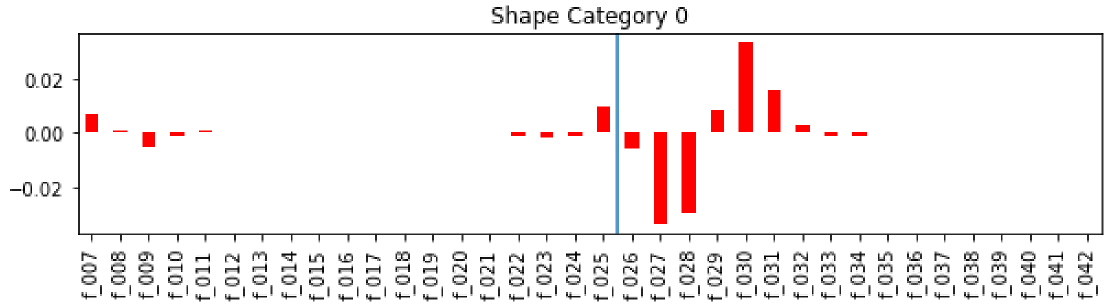
```
In [34]: for cluster in set(df_shape0['ClusterShape']):
             print('Percentage of cluster ', cluster, ':',
                   (df_shape0[df_shape0['ClusterShape']==cluster].count()/df_shape0.count()).me
```

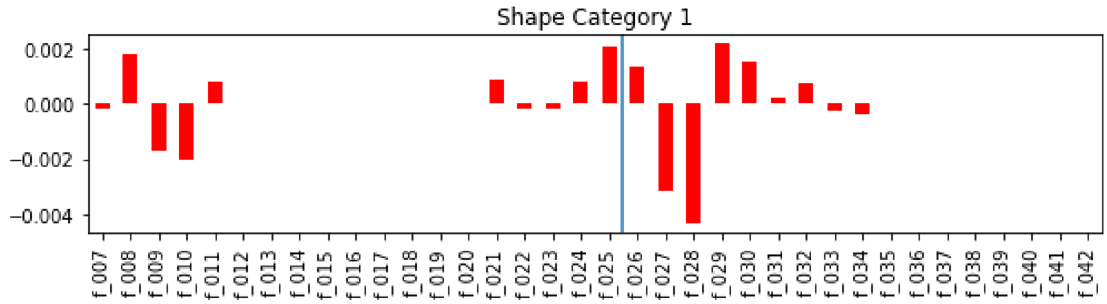
```
Percentage of cluster  0 : 0.10750507099391485
Percentage of cluster  1 : 0.5596445474741619
Percentage of cluster  2 : 0.3328503815319231
```

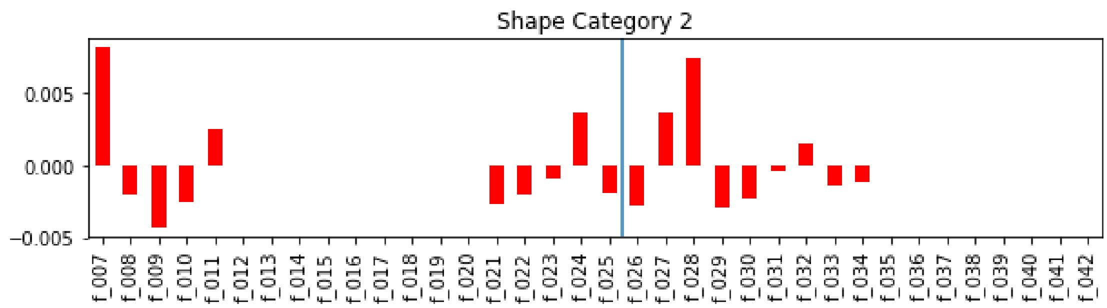
All categories are above the threshold 3%.

The prototype effect vectors are computed as the average of category members and their plots are shown below.

```
In [35]: shape0plt=df_shape0.copy()
         for c in [0,1,2]:
             plt.figure(figsize=(10,2))
             prototype_shape_vectors = shape0plt[shape0plt['ClusterShape']==c].drop('ClusterSha
             prototype_shape_vectors = prototype_shape_vectors.mean()
             prototype_shape_vectors.plot(kind='bar',color='red')
             plt.title('Shape Category {}'.format(c))
             plt.axvline(18.5)
             plt.show()
```







To get the clusters for all channels we cross product obtained clusters for

- Visibility
- Position
- Shape clusters

For the representation of these obtained clusters I used MultiLabelBinarizer package.

```
In [36]: df_e0=df_vis0.merge(df_pos0filtered,right_index=True, left_index=True)
         df_e0=df_e0.merge(df_shape0,right_index=True, left_index=True)
```

```
In [37]: df_e0['ClusterAllChannels']=df_e0[['ClusterVis','ClusterPos','ClusterShape']].values.
```

```
In [38]: df_e0clusters=pd.DataFrame(df_e0.ClusterAllChannels)
         c, r =df_e0clusters.shape
         df_e0clusters= df_e0clusters.values.reshape(c,)
```

```
In [39]: df_e0clusters=[np.array(i) for i in df_e0clusters]
```

```
In [40]: from sklearn.preprocessing import MultiLabelBinarizer
         target0= df_e0clusters
         target0 = MultiLabelBinarizer().fit_transform(target0)
         d0 = np.dtype((np.void, target0.dtype.itemsize * target0.shape[1]))
         _, ulabels0 = np.unique(np.ascontiguousarray(target0).view(d0), return_inverse=True)
```

```
In [41]: df_e0['BinarizedCluster']=ulabels0
```

There are 4 categories for all chanells:

```
In [42]: set(df_e0['BinarizedCluster'])

Out[42]: {0, 1, 2, 3}

In [43]: for cluster in set(df_e0['BinarizedCluster']):
             print('Percentage of cluster ', cluster, ':',(df_e0[df_e0['BinarizedCluster']==clu

Percentage of cluster  0 : 0.15561199570186598
Percentage of cluster  1 : 0.23512747875354087
Percentage of cluster  2 : 0.2762528084399724
Percentage of cluster  3 : 0.33300771710462085
```

All categories are above the threshold 3%

```
In [44]: df_e0svm=df_e0.drop(['ClusterVis','ClusterPos','ClusterShape','ClusterAllChannels'],ax

In [45]: df_e0svm.head()

Out[45]:    f_000     f_001     f_002     f_003     f_004     f_005     f_006  \
         0   -1.0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
         1   -1.0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
         2    0.0 -0.004347 -0.000436  0.000086  0.006332  0.122859  0.125860
         3    0.0 -0.000594 -0.002421 -0.000013  0.002235  0.122554  0.124156
         4    0.0  0.002804  0.001867  0.000109  0.004191  0.123729  0.118895

               f_007     f_008     f_009        ...            f_034  f_035  f_036  \
         0  0.000000  0.000000  0.000000        ...         0.000000    0.0    0.0
         1  0.000000  0.000000  0.000000        ...         0.000000    0.0    0.0
         2  0.010084 -0.008318 -0.009685        ...         0.008019    0.0    0.0
         3  0.011869  0.011687 -0.004677        ...         0.000903    0.0    0.0
         4  0.023592  0.015878 -0.017123        ...         0.000478    0.0    0.0

            f_037  f_038  f_039  f_040  f_041  f_042  BinarizedCluster
         0    0.0    0.0    0.0    0.0    0.0    0.0                 3
         1    0.0    0.0    0.0    0.0    0.0    0.0                 3
         2    0.0    0.0    0.0    0.0    0.0    0.0                 1
         3    0.0    0.0    0.0    0.0    0.0    0.0                 1
         4    0.0    0.0    0.0    0.0    0.0    0.0                 0

         [5 rows x 44 columns]
```

0.2 Learning Effect Category Prediction

```
In [46]: init_0, meta_0 = arff.loadarff(open('initial_0.arff'))
```

```
In [47]: df_init0=pd.DataFrame(init_0)

In [48]: df_init_0model=df_init0.merge(df_e0svm[['BinarizedCluster']],right_index=True,left_inc

In [49]: df_init_0model.head()

Out[49]:    f_000     f_001     f_002     f_003     f_004     f_005     f_006  \
         0    1.0  0.709839  0.541428 -0.182616 -0.261186  0.742176  0.851840
         1    1.0  0.581219  0.363101  0.309100  0.110585  0.597955  0.816736
         2    1.0  0.482915  0.330467  0.379856  0.196052  0.638606  0.825588
         3    1.0  0.478965  0.333463  0.379853  0.196561  0.763304  0.946122
         4    1.0  0.478468  0.330749  0.379688  0.194896  0.886331  1.068601

               f_007     f_008     f_009        ...           f_034  f_035  f_036  \
         0  0.500000  0.148810  0.062500        ...        0.005952    0.0    0.0
         1  0.237242  0.178067  0.162324        ...        0.013029    0.0    0.0
         2  0.142857  0.156735  0.160816        ...        0.015510    0.0    0.0
         3  0.148612  0.138765  0.161146        ...        0.024172    0.0    0.0
         4  0.154315  0.149239  0.146193        ...        0.019289    0.0    0.0

            f_037  f_038  f_039  f_040  f_041  f_042  BinarizedCluster
         0    0.0    0.0    0.0    0.0    0.0    0.0                 3
         1    0.0    0.0    0.0    0.0    0.0    0.0                 3
         2    0.0    0.0    0.0    0.0    0.0    0.0                 1
         3    0.0    0.0    0.0    0.0    0.0    0.0                 1
         4    0.0    0.0    0.0    0.0    0.0    0.0                 0

         [5 rows x 44 columns]

In [50]: from sklearn import svm, datasets, cross_validation
         from sklearn import metrics
         from sklearn.model_selection import train_test_split

C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\cross_val
  "This module will be removed in 0.20.", DeprecationWarning)


In [51]: df_train0,df_test0, df_train0y, df_test0y=train_test_split(
         df_init_0model.drop('BinarizedCluster',axis=1),df_init_0model[['BinarizedCluster']])

In [52]: clf = svm.SVC()
         clf.fit(df_train0,df_train0y)
         preds = clf.predict(df_test0)
         clf.score(df_test0,df_test0y)

C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\val
  y = column_or_1d(y, warn=True)


Out[52]: 0.41523437499999999
```

```
In [53]: df0 = pd.DataFrame(df_test0)
         df0['target'] = df_test0y
         df0['preds'] = preds
```

```
In [54]: print(metrics.confusion_matrix(df0['target'],df0['preds']))
```

```
[[  0  84   0 315]
 [  0 325   0 275]
 [  0 149   0 592]
 [  0  82   0 738]]
```

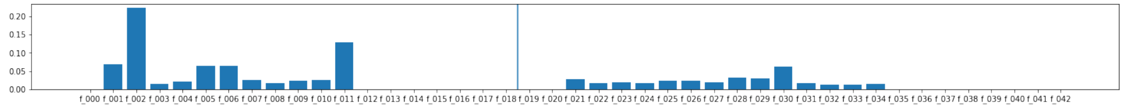
We can use feature importance to help to estimate the relative importance of the parameters.

For this we can use a random forest model to summarize the relative parameter importance scores.

```
In [55]: from sklearn.ensemble import RandomForestRegressor
         X = df_init_0model.drop('BinarizedCluster',axis=1)
         y = df_init_0model[['BinarizedCluster']].values
         # fit random forest model
         model = RandomForestRegressor(n_estimators=500, random_state=1)
         model.fit(X, y)
         # show importance scores
         print(model.feature_importances_)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\ipykernel_launche



[ 0.          0.06843798  0.22174065  0.01418764  0.0206865   0.06313142
  0.0634482   0.02552345  0.01778596  0.02286646  0.02505544  0.12833981
  0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.02862672  0.0160627   0.01809977  0.01712558
  0.02356493  0.02369066  0.01964328  0.03158727  0.03056717  0.06110781
  0.01662626  0.01311098  0.01345153  0.01553181  0.          0.          0.
  0.          0.          0.          0.          0.        ]
```

```
In [56]: # plot importance scores
         names = df_init_0model.columns.values[0:-1]
         ticks = [i for i in range(len(names))]
         plt.figure(figsize=(25,2))
         plt.bar(ticks, model.feature_importances_)
         plt.xticks(ticks, names)
         plt.axvline(18.5)
         plt.show()
```



0.2.1 Channel specific feature importance analysis

Position Features

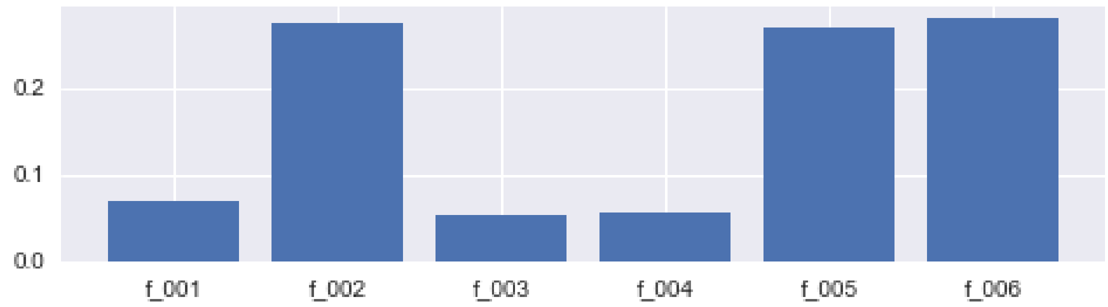
```
In [78]: from sklearn.ensemble import RandomForestRegressor
         Xpos = df_init_Omodel.iloc[:,1:7]
```

```
In [79]: ypos = df_init_Omodel[['BinarizedCluster']].values
         # fit random forest model
         modelpos = RandomForestRegressor(n_estimators=500, random_state=1)
         modelpos.fit(Xpos, ypos)
         # show importance scores
         print(modelpos.feature_importances_)
         # plot importance scores
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\ipykernel_launche
  after removing the cwd from sys.path.
```

```
[ 0.06885524  0.2741726   0.0537541   0.05512375  0.26931841  0.27877591]
```

```
In [262]: namespos = Xpos.columns.values
          tickspos = [i for i in range(len(namespos))]
          plt.figure(figsize=(8,2))
          plt.bar(tickspos, modelpos.feature_importances_)
          plt.xticks(tickspos, namespos)
          plt.show()
```



We can drop feature f_0003,f_004 in position features.

```
In [89]: Pos0filtered= Xpos.ix[:, 0:2].join(Xpos.ix[:, 4:])
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\ipykernel_launche
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate_ix
  """Entry point for launching an IPython kernel.
```

In [98]:
```
Pos0filtered.head()
```

Out[98]:

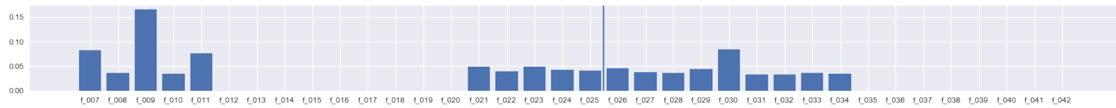
	f_001	f_002	f_005	f_006
0	0.709839	0.541428	0.742176	0.851840
1	0.581219	0.363101	0.597955	0.816736
2	0.482915	0.330467	0.638606	0.825588
3	0.478965	0.333463	0.763304	0.946122
4	0.478468	0.330749	0.886331	1.068601

Shape Features

In [84]:
```
from sklearn.ensemble import RandomForestRegressor
Xshape = df_init_0model.iloc[:,7:-1]
yshape = df_init_0model[['BinarizedCluster']].values
# fit random forest model
modelshape = RandomForestRegressor(n_estimators=500, random_state=1)
modelshape.fit(Xshape, yshape)
# show importance scores
print(modelshape.feature_importances_)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\ipykernel_launche



[ 0.0830202   0.03603354  0.16504617  0.03512287  0.0759817   0.          0.
  0.          0.          0.          0.          0.          0.          0.
  0.04932744  0.03878753  0.04848131  0.04249404  0.04130712  0.04522395
  0.03805515  0.03578108  0.04450586  0.08356079  0.03372793  0.03346815
  0.03597087  0.03410431  0.          0.          0.          0.          0.
  0.          0.          0.        ]
```

In [85]:
```
# plot importance scores
namesshape=Xshape.columns.values
ticksshape = [i for i in range(len(namesshape))]
plt.figure(figsize=(25,2))
plt.bar(ticksshape, modelshape.feature_importances_)
plt.xticks(ticksshape, namesshape)
plt.axvline(18.5)
plt.show()
```



We can drop 12 to 20 and 35 to 42 features in shape features.

```
In [ ]: df_init_0modelfiltered=df_init_0model.drop(['f_012','f_013','f_013','f_013','f_013','f_

In [88]: Shape0filtered=Xshape.ix[:, 0:5].join(Xshape.ix[:, 14:28])

In [91]: filteredmodel0=df_init_0model.ix[:,0:1].merge(Pos0filtered,right_index=True,left_inde

In [92]: filteredmodel0=filteredmodel0.merge(Shape0filtered,right_index=True,left_index=True)

In [93]: filteredmodel0=filteredmodel0.merge(df_init_0model.ix[:,-1:],right_index=True,left_in

In [95]: filteredmodel0.head()

Out[95]:    f_000     f_001     f_002     f_005     f_006     f_007     f_008  \
         0    1.0  0.709839  0.541428  0.742176  0.851840  0.500000  0.148810
         1    1.0  0.581219  0.363101  0.597955  0.816736  0.237242  0.178067
         2    1.0  0.482915  0.330467  0.638606  0.825588  0.142857  0.156735
         3    1.0  0.478965  0.333463  0.763304  0.946122  0.148612  0.138765
         4    1.0  0.478468  0.330749  0.886331  1.068601  0.154315  0.149239

               f_009     f_010     f_011         ...              f_026     f_027  \
         0  0.062500  0.017857  0.008929         ...           0.178571  0.169643
         1  0.162324  0.110749  0.039088         ...           0.153637  0.141694
         2  0.160816  0.182041  0.113469         ...           0.081633  0.102857
         3  0.161146  0.172784  0.134288         ...           0.071620  0.089526
         4  0.146193  0.183756  0.124873         ...           0.064975  0.090355

               f_028     f_029     f_030     f_031     f_032     f_033     f_034  \
         0  0.074405  0.065476  0.372024  0.026786  0.005952  0.011905  0.005952
         1  0.127036  0.122150  0.199240  0.082519  0.056460  0.033116  0.013029
         2  0.142857  0.164898  0.201633  0.126531  0.079184  0.050612  0.015510
         3  0.123545  0.201432  0.186213  0.123545  0.095792  0.068039  0.024172
         4  0.148223  0.152284  0.191878  0.148223  0.100508  0.063959  0.019289

            BinarizedCluster
         0                 3
         1                 3
         2                 1
         3                 1
         4                 0

         [5 rows x 25 columns]
```

In [96]:
```
df_train0f,df_test0f, df_train0yf, df_test0yf =train_test_split(
        filteredmodel0.drop('BinarizedCluster',axis=1),filteredmodel0[['BinarizedCluster']
```

In [97]:
```
clf = svm.SVC()
clf.fit(df_train0f,df_train0yf)
preds = clf.predict(df_test0f)
clf.score(df_test0f,df_test0yf)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

Out[97]:
```
0.42070312500000001
```

After dropping less relevant features score is increased to %42 from %40.
We can further optimize by grid searching parameters of svm algorithm.

In [ ]:
```
from sklearn.grid_search import GridSearchCV
def svc_param_selection(X, y, nfolds):
    Cs = [0.001, 0.01, 0.1, 1, 10]
    gammas = [0.001, 0.01, 0.1, 1]
    param_grid = {'C': Cs, 'gamma' : gammas}
    grid_search = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=nfolds)
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_
```

In [213]:
```
XX0,yy0=df_train0f, df_train0yf
c, r = yy0.shape
yy0 = yy0.values.reshape(c,)
bestparams0=svc_param_selection(XX0, yy0, 10)
```

In [214]:
```
bestparams0
```

Out[214]:
```
{'C': 10, 'gamma': 1}
```

In [215]:
```
clfopt0 = svm.SVC(C=10,gamma=1)
clfopt0.fit(df_train0f,df_train0yf)
predsopt0 = clfopt0.predict(df_test0f)
clfopt0.score(df_test0f,df_test0yf)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

Out[215]:
```
0.63398437500000004
```

0.2.2 Channel specific predictions

Visibility

```
In [115]: df_train0vis,df_test0vis, df_train0yvis, df_test0yvis =train_test_split(
              df_vis0.drop('ClusterVis',axis=1), df_vis0[['ClusterVis']],random_state=123)
```

```
In [118]: clfvis = svm.SVC()
          clfvis.fit(df_train0vis,df_train0yvis)
          predsvis = clfvis.predict(df_test0vis)
          clfvis.score(df_test0vis,df_test0yvis)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[118]: 1.0
```

Position

```
In [112]: df_train0pos,df_test0pos, df_train0ypos, df_test0ypos =train_test_split(
              df_pos0filtered.drop('ClusterPos',axis=1), df_pos0filtered[['ClusterPos']],rando
```

```
In [113]: clf = svm.SVC()
          clf.fit(df_train0pos,df_train0ypos)
          predspos = clf.predict(df_test0pos)
          clf.score(df_test0pos,df_test0ypos)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[113]: 0.99726562500000004
```

Shape

```
In [168]: df_train0shape,df_test0shape, df_train0yshape, df_test0yshape =train_test_split(
              df_shape0.drop('ClusterShape',axis=1), df_shape0[['ClusterShape']],random_state=
```

```
In [169]: clfshape = svm.SVC()
          clfshape.fit(df_train0shape,df_train0yshape)
          predsshape = clfshape.predict(df_test0shape)
          clfshape.score(df_test0shape,df_test0yshape)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[169]: 0.57087678640401696
```

```
In [ ]: Xshape.ix[:, 0:5].join(Xshape.ix[:, 14:28])
```

In [135]: predsvis,predsvis,predsshape

Out[135]: (array([1, 1, 1, ..., 1, 1, 1]),
 array([1, 1, 1, ..., 1, 1, 1]),
 array([1, 1, 1, ..., 1, 1, 1]))

In [263]: df_test0shape.head()

Out[263]:
```
         f_007     f_008     f_009     f_010     f_011     f_021     f_022  \
7583  0.017017  0.006819 -0.010780 -0.001221 -0.004152  0.005655 -0.006723
3980 -0.019104  0.006588 -0.003953  0.012516 -0.003953 -0.002635 -0.011858
8405  0.002971  0.017183 -0.008840 -0.016983 -0.000469 -0.000486  0.005598
2762  0.019023 -0.010531 -0.007006  0.008428 -0.003759  0.002885  0.000938
6818  0.014764 -0.014026  0.009770 -0.017552 -0.006415  0.000281  0.007250

         f_023     f_024     f_025     f_026     f_027     f_028     f_029  \
7583 -0.018995  0.012379  0.005708 -0.022886  0.005880 -0.005684  0.004061
3980  0.002635  0.019763 -0.009881 -0.000659 -0.013834  0.004611  0.028327
8405  0.003603 -0.002577  0.014731 -0.021786  0.000781 -0.005854  0.012884
2762 -0.013458  0.003481  0.007834  0.029110 -0.014063 -0.011188  0.003372
6818 -0.018286  0.024215  0.017871 -0.028081  0.004986  0.013422 -0.011544

         f_030     f_031     f_032     f_033     f_034
7583  0.011293  0.011103 -0.003627  0.001396 -0.007244
3980  0.008564 -0.011858  0.000659 -0.004611 -0.001318
8405  0.007092 -0.011224  0.001267  0.003976 -0.001865
2762 -0.009476 -0.007440  0.003822 -0.004285  0.002314
6818  0.025066 -0.009623 -0.006415 -0.002138 -0.003543
```

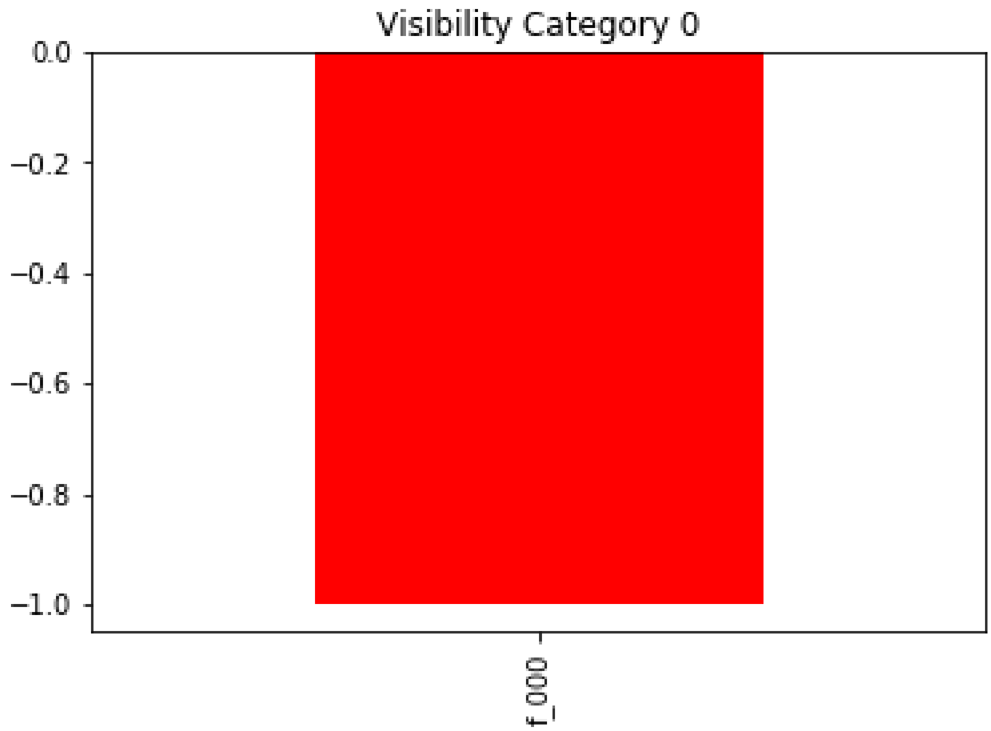
For Behaivor 0, visibility and position can be predicted easily.
Lift Behaivor

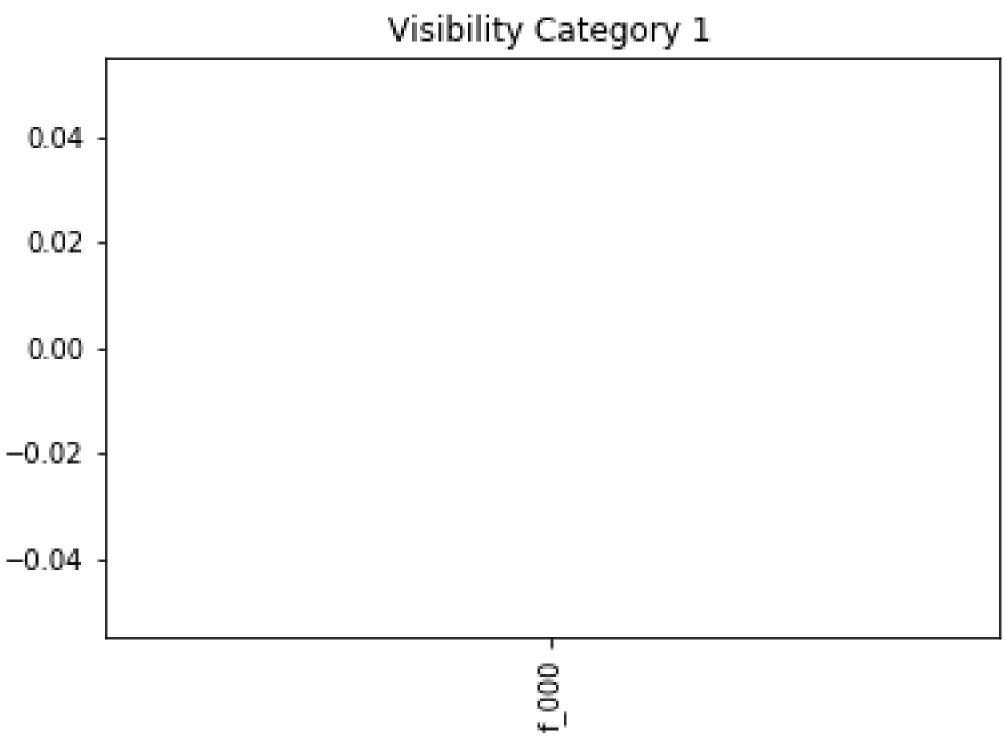
In [57]:
```python
vis_e3, meta_e3 = arff.loadarff(open('resvis3.arff'))
df_vis3=pd.DataFrame(vis_e3)
df_vis3['Cluster']=df_vis3['Cluster'].apply(lambda x: x.decode('UTF-8')).str.extract(
print('Visibility Clusters',(set(df_vis3['Cluster'])))
vis3plt=df_vis3
df_vis3=df_vis3.drop('Instance_number',axis=1)
df_vis3=df_vis3.rename(columns={'Cluster':'ClusterVis'})
pos_e3, meta_e3 = arff.loadarff(open('respos3.arff'))
df_pos3=pd.DataFrame(pos_e3)
df_pos3['Cluster']=df_pos3['Cluster'].apply(lambda x: x.decode('UTF-8')).str.extract(
print('Position Clusters',(set(df_pos3['Cluster'])))
df_pos3=df_pos3.drop('Instance_number',axis=1)
df_pos3=df_pos3.rename(columns={'Cluster':'ClusterPos'})
for cluster in set(df_pos3['ClusterPos']):
    print('Percentage of position', cluster, ':',
          (df_pos3[df_pos3['ClusterPos']==cluster].count()/df_pos3.count()).mean())
shape_e3, meta_e3 = arff.loadarff(open('resshape3.arff'))
```

```
df_shape3=pd.DataFrame(shape_e3)
df_shape3['Cluster']=df_shape3['Cluster'].apply(lambda x: x.decode('UTF-8')).str.extra
print('Shape Clusters',(set(df_shape3['Cluster'])))
df_shape3=df_shape3.drop('Instance_number',axis=1)
df_shape3=df_shape3.rename(columns={'Cluster':'ClusterShape'})
for cluster in set(df_shape3['ClusterShape']):
    print('Percentage of shape cluster ', cluster, ':',
          (df_shape3[df_shape3['ClusterShape']==cluster].count()/df_shape3.count()).me
```

```
Visibility Clusters {0, 1}
Position Clusters {0, 1, 2, 3}
Percentage of position 0 : 0.16607997111652675
Percentage of position 1 : 0.3177182056142251
Percentage of position 2 : 0.5055510425128622
Percentage of position 3 : 0.010650780756385954
Shape Clusters {0, 1, 2, 3}
Percentage of shape cluster  0 : 0.03420886361584981
Percentage of shape cluster  1 : 0.0691398140626411
Percentage of shape cluster  2 : 0.14531997472696104
Percentage of shape cluster  3 : 0.7513313475945477
```

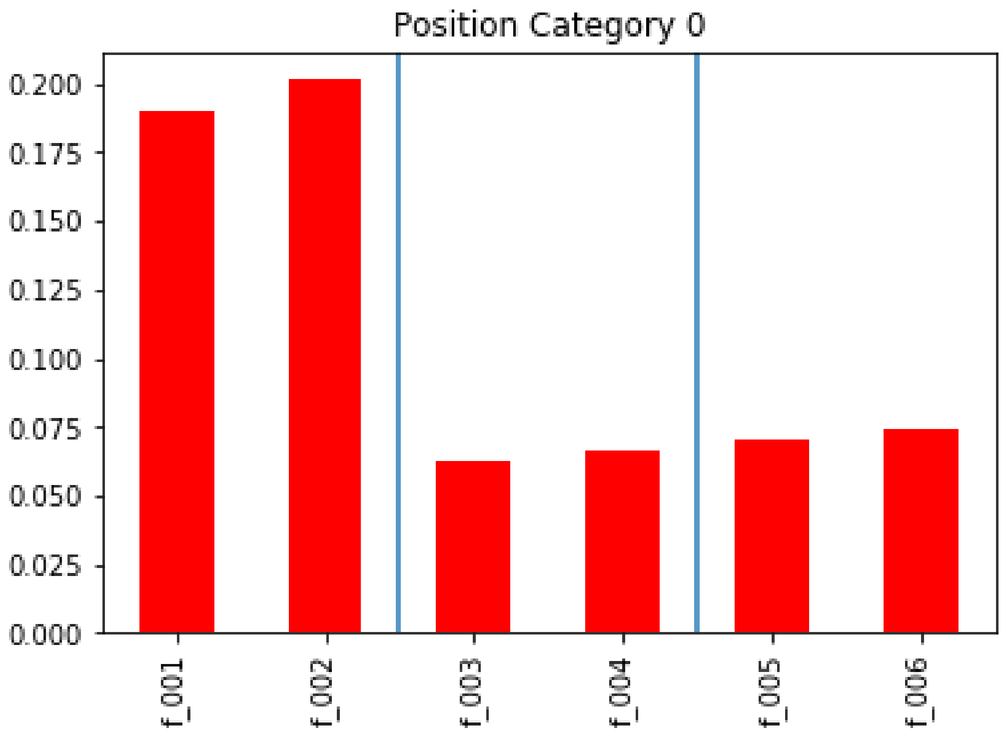
In [58]:
```
vis3plt=df_vis3
for c in [0,1]:
    prototype_visibility_vectors= vis3plt[vis3plt['ClusterVis']==c].drop('ClusterVis'
    prototype_visibility_vectors = prototype_visibility_vectors.mean()
    prototype_visibility_vectors.plot(kind='bar',color='red')
    plt.title('Visibility Category {}'.format(c))
    plt.show()
```

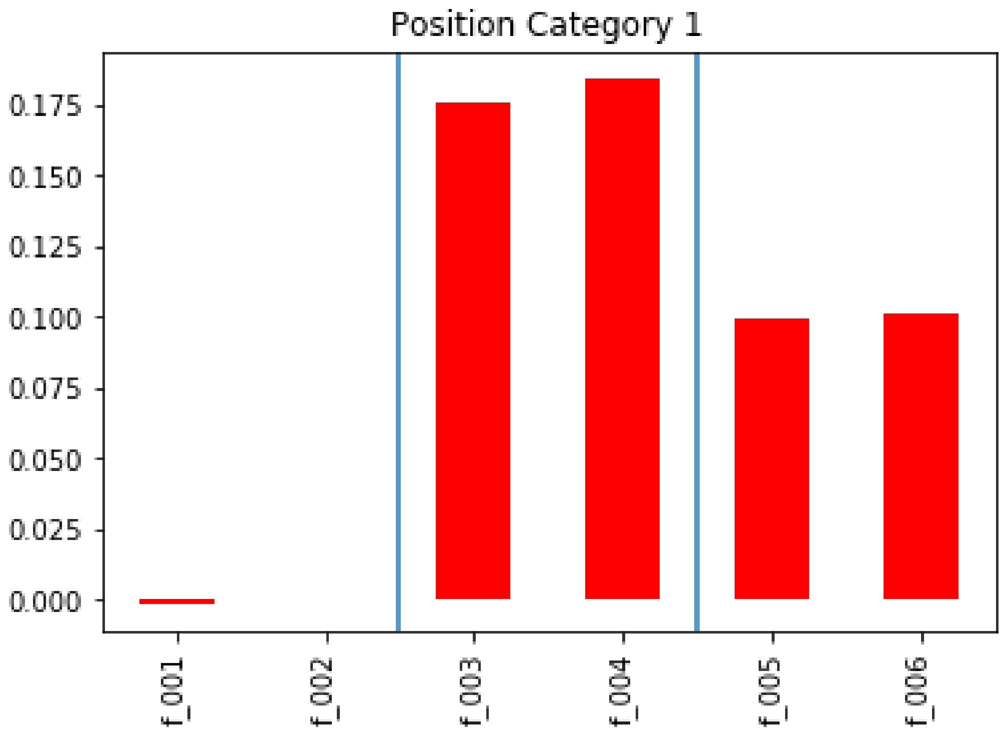

Visibility Category 0
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
f_000

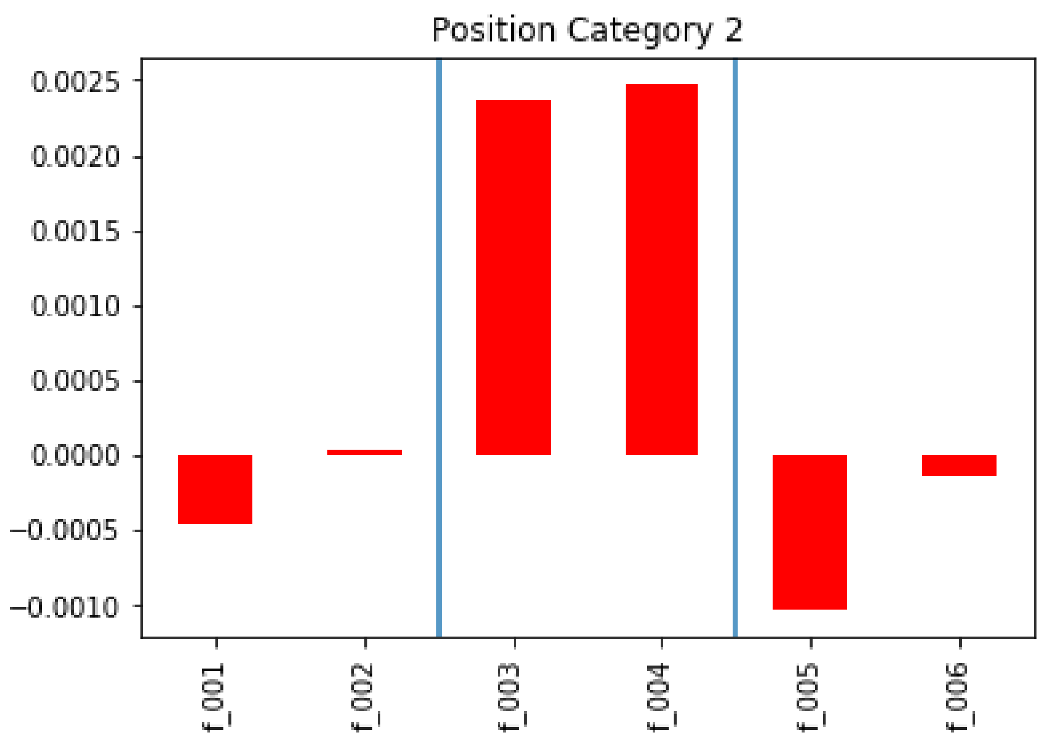

Visibility Category 1
0.04
0.02
0.00
−0.02
−0.04
f_000

```
In [59]: df_pos3filtered=df_pos3[df_pos3['ClusterPos'].isin([0,1,2])]
```

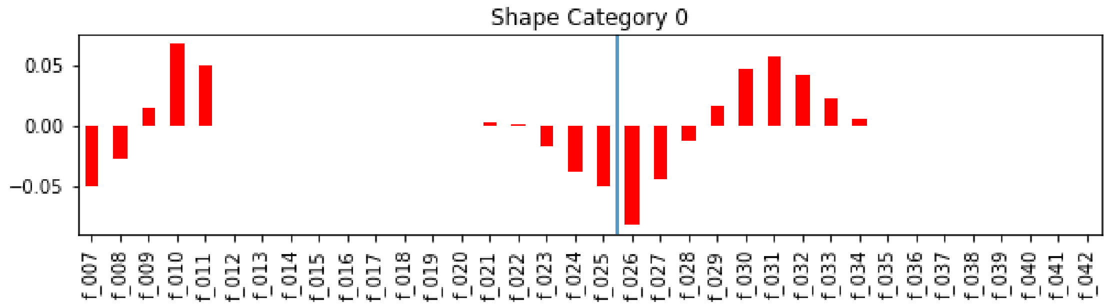
```
In [60]: pos3plt=df_pos3filtered
         for c in [0,1,2]:
             prototype_position_vectors= pos3plt[pos3plt['ClusterPos']==c].drop('ClusterPos',ax
             prototype_position_vectors=prototype_position_vectors.mean()
             prototype_position_vectors.plot(kind='bar',color='red')
             plt.title('Position Category {}'.format(c))
             plt.axvline(1.5)
             plt.axvline(3.5)
             plt.show()
```

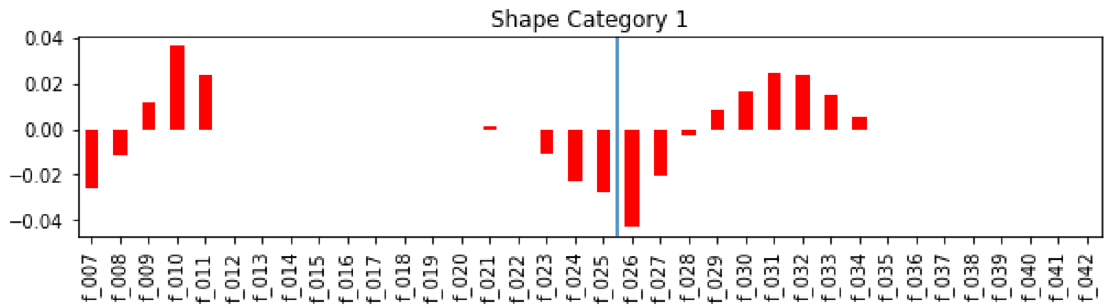


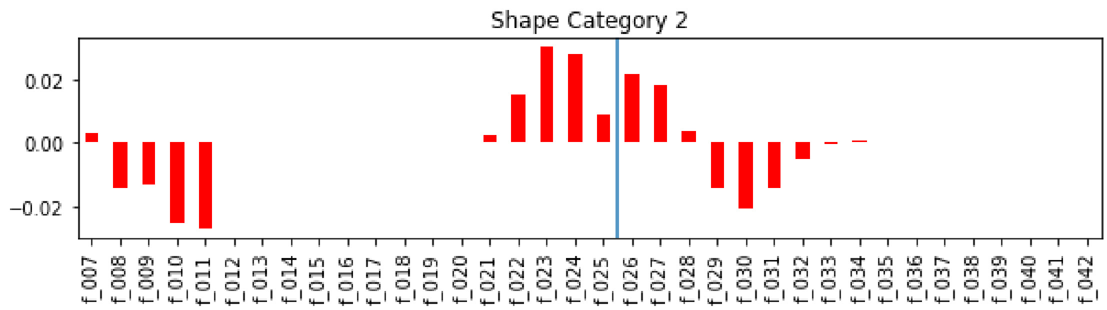

Position Category 1
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
f_001
f_002
f_003
f_004
f_005
f_006

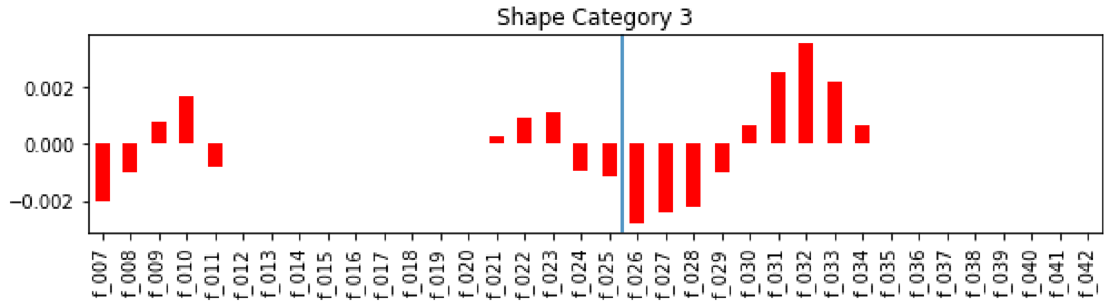

Position Category 2
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−0.0005
−0.0010
f_001
f_002
f_003
f_004
f_005
f_006

```
In [61]: shape3plt=df_shape3.copy()
         for c in [0,1,2,3]:
             plt.figure(figsize=(10,2))
             prototype_shape_vectors = shape3plt[shape3plt['ClusterShape']==c].drop('ClusterSha
             prototype_shape_vectors = prototype_shape_vectors.mean()
             prototype_shape_vectors.plot(kind='bar',color='red')
             plt.title('Shape Category {}'.format(c))
             plt.axvline(18.5)
             plt.show()
```









In [62]:
```
df_e3=df_vis3.merge(df_pos3filtered,right_index=True, left_index=True)
df_e3=df_e3.merge(df_shape3,right_index=True, left_index=True)
df_e3.head()
```

Out[62]:
```
   f_000  ClusterVis     f_001     f_002     f_003     f_004     f_005  \
0    0.0           1  0.198161  0.215867  0.043056  0.070716  0.054568
1    0.0           1  0.003186  0.001764  0.218123  0.218315  0.074561
2    0.0           1  0.002179  0.006259  0.001258 -0.005005 -0.002999
3    0.0           1 -0.001696 -0.000362 -0.004771  0.006622  0.001296
4    0.0           1  0.000697  0.004150  0.204055  0.210232  0.075725

      f_006  ClusterPos     f_007    ...         f_034  f_035  f_036  \
0  0.073154           0 -0.107797    ...      0.000916    0.0    0.0
1  0.084652           1 -0.063562    ...      0.009316    0.0    0.0
2 -0.004648           2  0.021382    ...      0.001872    0.0    0.0
3  0.002427           2  0.010606    ...     -0.004545    0.0    0.0
4  0.080523           1 -0.020880    ...     -0.008600    0.0    0.0

   f_037  f_038  f_039  f_040  f_041  f_042  ClusterShape
0    0.0    0.0    0.0    0.0    0.0    0.0             2
1    0.0    0.0    0.0    0.0    0.0    0.0             3
2    0.0    0.0    0.0    0.0    0.0    0.0             3
3    0.0    0.0    0.0    0.0    0.0    0.0             3
4    0.0    0.0    0.0    0.0    0.0    0.0             3

[5 rows x 46 columns]
```

In [179]:
```
df_e3['ClusterAllChannels']=df_e3[['ClusterVis','ClusterPos','ClusterShape']].values
```

In [180]:
```
df_e3clusters=pd.DataFrame(df_e3.ClusterAllChannels)
c, r =df_e3clusters.shape
df_e3clusters= df_e3clusters.values.reshape(c,)
```

In [181]:
```
df_e3clusters=[np.array(i) for i in df_e3clusters]
```

In [182]:
```
from sklearn.preprocessing import MultiLabelBinarizer
target3= df_e3clusters
```

```
        target3 = MultiLabelBinarizer().fit_transform(target3)
        d3 = np.dtype((np.void, target3.dtype.itemsize * target3.shape[1]))
        _, ulabels3 = np.unique(np.ascontiguousarray(target3).view(d3), return_inverse=True)
```

```
In [183]: df_e3['BinarizedCluster']=ulabels3
```

```
In [184]: set(df_e3['BinarizedCluster'])
```

```
Out[184]: {0, 1, 2, 3, 4, 5, 6, 7}
```

```
In [185]: for cluster in set(df_e3['BinarizedCluster']):
              print('Percentage of cluster ', cluster, ':',
                    (df_e3[df_e3['BinarizedCluster']==cluster].count()/df_e3.count()).mean())
```

```
Percentage of cluster  0 : 0.05711577410820236
Percentage of cluster  1 : 0.2189581242587355
Percentage of cluster  2 : 0.023994161116686428
Percentage of cluster  3 : 0.2714168415290578
Percentage of cluster  4 : 0.2283550770915064
Percentage of cluster  5 : 0.036401788158014765
Percentage of cluster  6 : 0.035033299881397646
Percentage of cluster  7 : 0.12872913055378166
```

Cluster 2 is below threshold so we can drop that cluster.

```
In [186]: df_e3filtered=df_e3[df_e3['BinarizedCluster'].isin([0,1,3,4,5,6,7])]
```

```
In [187]: df_e3svm=df_e3filtered.drop(['ClusterVis','ClusterPos','ClusterShape','ClusterAllCha
```

```
In [188]: df_e3svm.head()
```

```
Out[188]:    f_000     f_001     f_002     f_003     f_004     f_005     f_006  \
          0    0.0  0.198161  0.215867  0.043056  0.070716  0.054568  0.073154
          1    0.0  0.003186  0.001764  0.218123  0.218315  0.074561  0.084652
          2    0.0  0.002179  0.006259  0.001258 -0.005005 -0.002999 -0.004648
          3    0.0 -0.001696 -0.000362 -0.004771  0.006622  0.001296  0.002427
          4    0.0  0.000697  0.004150  0.204055  0.210232  0.075725  0.080523

                f_007     f_008     f_009        ...            f_034  f_035  f_036  \
          0 -0.107797  0.032510 -0.022526        ...         0.000916    0.0    0.0
          1 -0.063562  0.012507  0.008988        ...         0.009316    0.0    0.0
          2  0.021382 -0.008606 -0.017945        ...         0.001872    0.0    0.0
          3  0.010606  0.001515 -0.003030        ...        -0.004545    0.0    0.0
          4 -0.020880  0.005550  0.026560        ...        -0.008600    0.0    0.0

             f_037  f_038  f_039  f_040  f_041  f_042  BinarizedCluster
          0    0.0    0.0    0.0    0.0    0.0    0.0                 7
          1    0.0    0.0    0.0    0.0    0.0    0.0                 1
```

```
2    0.0    0.0    0.0    0.0    0.0    0.0                   3
3    0.0    0.0    0.0    0.0    0.0    0.0                   3
4    0.0    0.0    0.0    0.0    0.0    0.0                   1

[5 rows x 44 columns]
```

0.3 Learning Effect Category Prediction

```
In [189]: init_3, meta_3 = arff.loadarff(open('initial_3.arff'))

In [190]: df_init3=pd.DataFrame(init_3)

In [191]: df_init_3model=df_init3.merge(df_e3svm[['BinarizedCluster']],right_index=True,left_i

In [192]: df_init_3model.head()

Out[192]:    f_000     f_001     f_002     f_003     f_004     f_005     f_006  \
          0    1.0  0.511011  0.330399 -0.224832 -0.341808  0.702123  0.802668
          1    1.0  0.516181  0.330754 -0.064936 -0.301608  0.687565  0.916989
          2    1.0  0.518997  0.330663 -0.044129 -0.270895  0.931139  1.166367
          3    1.0  0.474202  0.332638 -0.060203 -0.286377  0.981291  1.175582
          4    1.0  0.452989  0.330266  0.115750 -0.102665  0.795211  1.008686

                f_007     f_008     f_009        ...            f_034  f_035  f_036  \
          0  0.424870  0.132124  0.113990        ...         0.005181    0.0    0.0
          1  0.299682  0.157393  0.112083        ...         0.008744    0.0    0.0
          2  0.206573  0.173709  0.129577        ...         0.004695    0.0    0.0
          3  0.250000  0.183333  0.115152        ...         0.007576    0.0    0.0
          4  0.174201  0.190739  0.117971        ...         0.015436    0.0    0.0

             f_037  f_038  f_039  f_040  f_041  f_042  BinarizedCluster
          0    0.0    0.0    0.0    0.0    0.0    0.0                 7
          1    0.0    0.0    0.0    0.0    0.0    0.0                 1
          2    0.0    0.0    0.0    0.0    0.0    0.0                 3
          3    0.0    0.0    0.0    0.0    0.0    0.0                 3
          4    0.0    0.0    0.0    0.0    0.0    0.0                 1

          [5 rows x 44 columns]

In [193]: df_train3,df_test3, df_train3y, df_test3y = train_test_split(df_init_3model.drop('Bi
                                                                     df_init_3model[['Binariz

In [194]: clf3 = svm.SVC()
          clf3.fit(df_train3,df_train3y)
          preds3 = clf3.predict(df_test3)
          clf3.score(df_test3,df_test3y)

C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[194]: 0.58915887850467286
```

```
In [198]: from sklearn.grid_search import GridSearchCV
          def svc_param_selection(X, y, nfolds):
              Cs = [0.001, 0.01, 0.1, 1, 10]
              gammas = [0.001, 0.01, 0.1, 1]
              param_grid = {'C': Cs, 'gamma' : gammas}
              grid_search = GridSearchCV(svm.SVC(kernel='rbf'), param_grid, cv=nfolds)
              grid_search.fit(X, y)
              grid_search.best_params_
              return grid_search.best_params_
```

```
In [207]: XX,yy=df_train3, df_train3y
```

```
In [209]: c, r = yy.shape
          yy = yy.values.reshape(c,)
```

```
In [210]: bestparams=svc_param_selection(XX, yy, 10)
```

```
In [211]: bestparams
```

```
Out[211]: {'C': 10, 'gamma': 1}
```

```
In [212]: clf3opt = svm.SVC(C=10,gamma=1)
          clf3opt.fit(df_train3,df_train3y)
          preds3opt = clf3opt.predict(df_test3)
          clf3opt.score(df_test3,df_test3y)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[212]: 0.80112149532710275
```

After grid search optimization prediction accuracy increased to %80 from %58.
Channel specific predictions
Visibility

```
In [258]: df_train3vis,df_test3vis, df_train3yvis, df_test3yvis =train_test_split(
              df_vis3.drop('ClusterVis',axis=1), df_vis3[['ClusterVis']],random_state=123)
```

```
In [259]: clfvis3 = svm.SVC()
          clfvis3.fit(df_train3vis,df_train3yvis)
          predsvis3 = clfvis3.predict(df_test3vis)
          clfvis3.score(df_test3vis,df_test3yvis)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[259]: 1.0
```

Position

```
In [219]: df_train3pos,df_test3pos, df_train3ypos, df_test3ypos =train_test_split(
              df_pos3filtered.drop('ClusterPos',axis=1), df_pos3filtered[['ClusterPos']],randor
```

```
In [221]: clfpos3 = svm.SVC()
          clfpos3.fit(df_train3pos,df_train3ypos)
          predspos3 = clfpos3.predict(df_test3pos)
          clfpos3.score(df_test3pos,df_test3ypos)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[221]: 0.99598686610726017
```

Shape

```
In [222]: df_train3shape,df_test3shape, df_train3yshape, df_test3yshape =train_test_split(
              df_shape3.drop('ClusterShape',axis=1), df_shape3[['ClusterShape']],random_state=
```

```
In [223]: clfshape3 = svm.SVC()
          clfshape3.fit(df_train3shape,df_train3yshape)
          predsshape3 = clfshape3.predict(df_test3shape)
          clfshape3.score(df_test3shape,df_test3yshape)
```

```
C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\envs\py35\lib\site-packages\sklearn\utils\va
  y = column_or_1d(y, warn=True)
```

```
Out[223]: 0.7487364620938628
```

For behaivor 3 visibility and position features still easy to predict.

Additionally, for lift behaivor, shape features are more predictable than for push-right behaivor.

```
In [ ]:
```