0.1 INDiC: Improved Non-Intrusive load monitoring using load Division and Calibration

Nipun Batra, Haimonti Dutta, Amarjeet Singh

nipunb,haimonti,amarjeet@iiitd.ac.in

About the dataset The datset contains high frequency, low frequency and raw data of 6 households in the USA. We choose House 2 for our analysis as it has highest reported accuracy in REDD paper. Further we choose to do the analysis of **low frequency** data. This house contains the following appliances/circuits:

- Kitchen
- Kitchen 2
- Stove
- Refrigerator
- Dishwasher
- Disposal
- Washer Dryer
- Microwave
- Lighting

These circuits are samppled once every 3-4 seconds. Also the house contains 2 mains which are sampled at 1 Hz.

Imports In this section we setup the basic imports which shall be required for performing the analysis and mention their version numbers.

```
import platform
print "OS:",platform.platform()
import sys
print "System version:",sys.version
import numpy as np
print "Numpy version:",np.version.version
import matplotlib
print "Matplotlib version:",matplotlib.__version__
import pandas as pd
print "Pandas version:",pd.__version__
import sklearn as sklearn
print "Scikit-learn version:",sklearn.__version__
from sklearn.cluster import MiniBatchKMeans, KMeans
import matplotlib.pyplot as plt
import math
```

```
import itertools
from pandas import Series,DataFrame
import json
import time
import pprint
import datetime
from copy import deepcopy
import warnings
warnings.filterwarnings('ignore')
```

```
OS: Linux-3.5.0-36-generic-x86_64-with-Ubuntu-12.10-quantal
System version: 2.7.3 (default, Apr 10 2013, 05:13:16)
[GCC 4.7.2]
Numpy version: 1.6.2
Matplotlib version:

1.1.1
Pandas version: 0.11.1.dev-58642a6
Scikit-learn version: 0.12.1
```

Defining base paths for data and configurations

```
DATA_PATH="/home/nipun/study/datasets/MIT/low_freq/house_2/"
```

Prettifying CSS and Matplotlib plots

```
s = json.load(open("bmh_matplotlibrc.json") )
matplotlib.rcParams.update(s)

matplotlib.rcParams.update({'font.size': 24})

from IPython.core.display import HTML
def css_styling():
    styles = open("custom.css", "r").read()
    return HTML(styles)
css_styling()
```

```
<IPython.core.display.HTML at 0x9b159d0>
```

Loading and Plotting of Mains and Appliance Data This is how a channel data file looks like

```
!head $DATA_PATH/channel_1.dat
```

```
1303082307 16.57
1303082308 16.55
1303082309 16.64
1303082310 16.61
1303082311 16.72
1303082312 16.70
1303082313 16.79
```

```
1303082314 16.79
1303082315 16.75
1303082334 16.54
```

```
start_time=time.time()
mains_1_data=np.loadtxt('%s/channel_1.dat' %DATA_PATH)
mains_2_data=np.loadtxt('%s/channel_2.dat' %DATA_PATH)
end_time=time.time()
print "Time taken to load Mains data= "+str(end_time-start_time)+" seconds"
```

```
Time taken to load Mains data= 22.3748559952 seconds
```

Assigning Mains 1 and Mains 2 to channel 1 and channel 2 data. Also converting epoch time to Numpy datetime64 type.

```
mains_1_power=mains_1_data[:,1]
mains_2_power=mains_2_data[:,1]
timestamp=mains_1_data[:,0]
timestamp_mains_date=timestamp.astype('datetime64[s]')
```

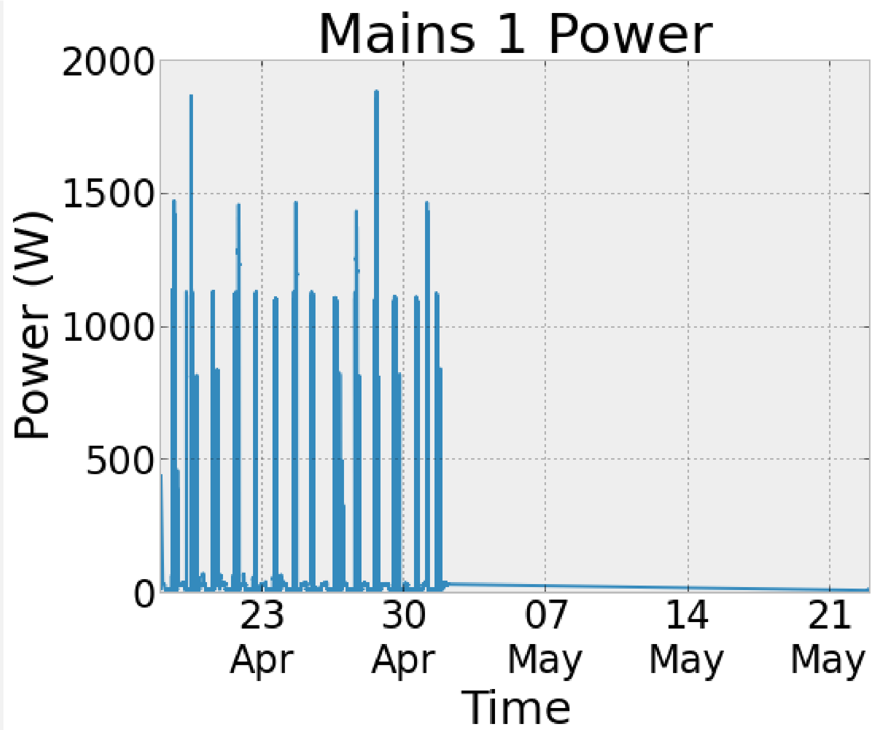
Creating a Pandas DataFrame. Overall statistics about the dataset.

```
df_mains=DataFrame({'Mains_1_Power':mains_1_power,'Mains_2_Power':mains_2_power},index=
    timestamp_mains_date)
df_mains.describe()
```

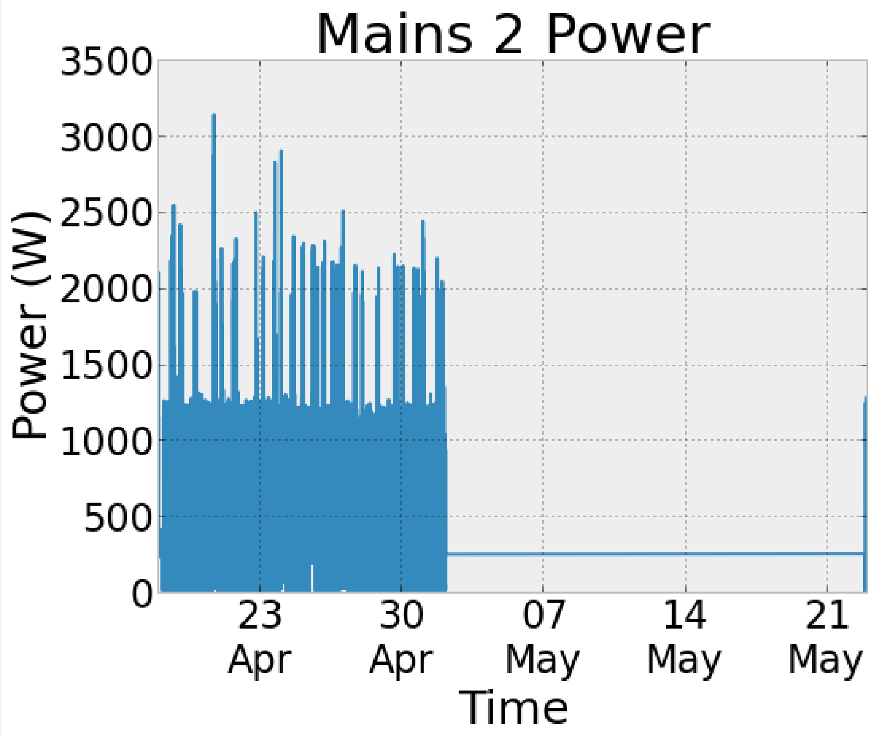
```
        Mains_1_Power   Mains_2_Power
count  1198534.000000  1198534.000000
mean        42.850747      187.432460
std        146.054901      215.213296
min          9.690000       21.190000
25%         14.810000       22.470000
50%         15.110000      218.470000
75%         33.550000      259.130000
max       1887.420000     3149.020000
```

Plotting overall mains consumption.

```
plt.title('Mains 1 Power')
plt.xlabel('Time')
plt.ylabel('Power (W)')
plt.figsize(8,6)
matplotlib.rcParams.update({'font.size': 24})
plt.plot(df_mains.index.to_pydatetime(),df_mains.Mains_1_Power)
plt.gca().xaxis.set_major_formatter( DateFormatter('%d\n%b') )
```




```
plt.title('Mains 2 Power')
plt.xlabel('Time')
plt.ylabel('Power (W)')
plt.figsize(8,6)
matplotlib.rcParams.update({'font.size': 24})
plt.plot(df_mains.index.to_pydatetime(),df_mains.Mains_2_Power)
plt.gca().xaxis.set_major_formatter( DateFormatter('%d\n%b') )
```



Loading appliance data

```
start_time=time.time()
kitchen_data=np.loadtxt('%s/channel_3.dat' %DATA_PATH)
light_data=np.loadtxt('%s/channel_4.dat' %DATA_PATH)
stove_data=np.loadtxt('%s/channel_5.dat' %DATA_PATH)
microwave_data=np.loadtxt('%s/channel_6.dat' %DATA_PATH)
washer_dry_data=np.loadtxt('%s/channel_7.dat' %DATA_PATH)
kitchen_2_data=np.loadtxt('%s/channel_8.dat' %DATA_PATH)
refrigerator_data=np.loadtxt('%s/channel_9.dat' %DATA_PATH)
dishwasher_data=np.loadtxt('%s/channel_10.dat' %DATA_PATH)
disposal_data=np.loadtxt('%s/channel_11.dat' %DATA_PATH)
kitchen_power=kitchen_data[:,1]
light_power=light_data[:,1]
stove_power=stove_data[:,1]
microwave_power=microwave_data[:,1]
washer_dryer_power=washer_dry_data[:,1]
kitchen_2_power=kitchen_2_data[:,1]
refrigerator_power=refrigerator_data[:,1]
dishwasher_power=dishwasher_data[:,1]
disposal_power=disposal_data[:,1]
timestamp=kitchen_data[:,0]
timestamp_appliance_date=timestamp.astype('datetime64[s]')
end_time=time.time()
print "Time taken to load appliance data= "+str(end_time-start_time)+" seconds"
```

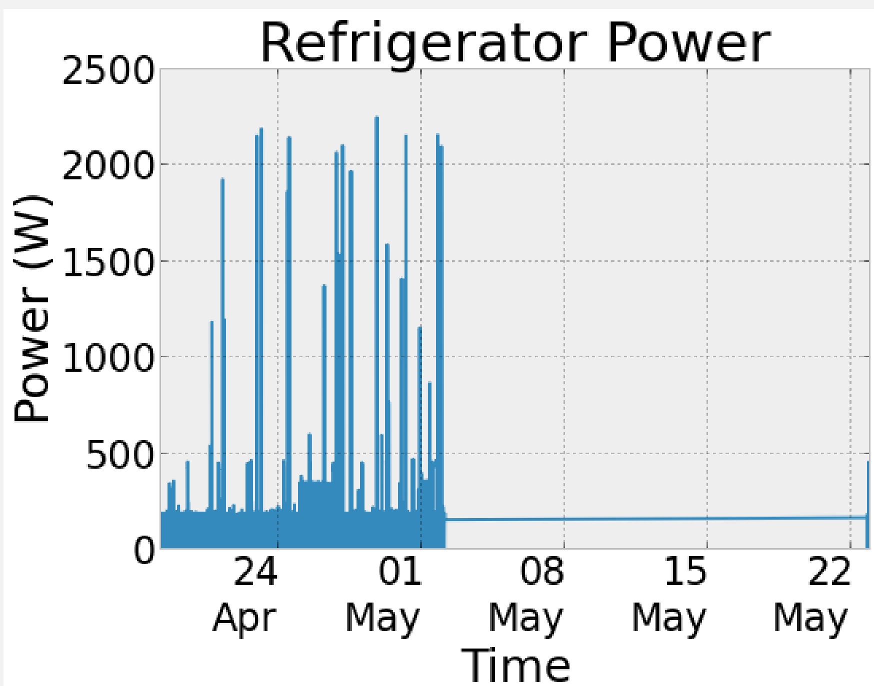
```
Time taken to load appliance data= 25.6279199123 seconds
```

Creating a data frame of all appliances data

```
df_appliances=DataFrame({'kitchen':kitchen_power,'light':light_power,'stove':stove_power
    ,'microwave':microwave_power,\
'washer_dryer':washer_dryer_power,'kitchen_2':kitchen_2_power,'refrigerator':
    refrigerator_power,'dishwasher':dishwasher_power,\
'disposal':disposal_power},index=timestamp_appliance_date)
pd.set_option('display.precision', 2)
print df_appliances.describe().to_string()
```

```
dishwasher  disposal   kitchen  kitchen_2     light  microwave  refrigerator     stove  washer_dryer
count     318759.0  318759.0  318759.0   318759.0  318759.0   318759.0      318759.0  318759.0        31875
mean           8.9       0.1       6.2       10.3      26.5       15.3          79.7       1.4
std           95.7       3.3      37.7       97.4      45.8      110.6          88.0      18.9
min            0.0       0.0       0.0        0.0       0.0        0.0           0.0       0.0
25%            0.0       0.0       0.0        1.0       8.0        4.0           6.0       0.0
50%            0.0       0.0       0.0        1.0       8.0        5.0           7.0       0.0
75%            0.0       0.0      13.0        1.0       9.0        5.0         161.0       1.0
max         1457.0     609.0     805.0     1119.0     289.0     1986.0        2246.0     457.0            5
```

```
df_appliances.refrigerator.plot(rot=0,title='Refrigerator Power')
plt.gca().xaxis.set_major_formatter( DateFormatter('%d\n%b') )
plt.gca().set_ylabel("Power (W)")
plt.gca().set_xlabel("Time");
```



Starting time and ending times of channels

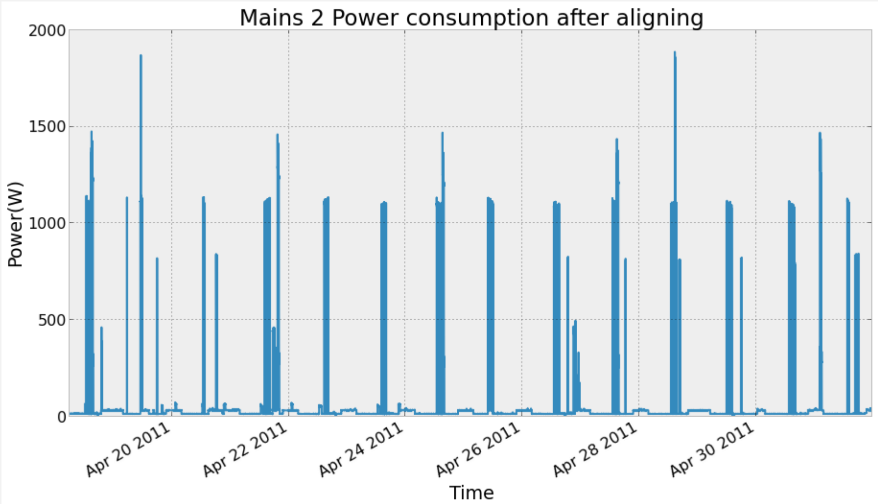
```
print "Mains starting time",df_mains.index[0]
print "Mains ending time",df_mains.index[-1]
print "Mains starting time",df_appliances.index[0]
print "Mains ending time",df_appliances.index[-1]
```

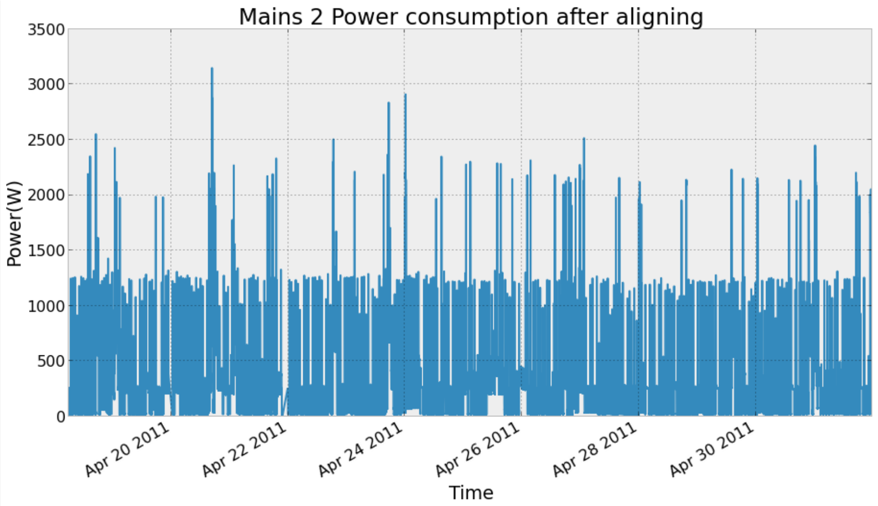
```
Mains starting time 2011-04-18 06:00:00
Mains ending time 2011-05-01 23:59:59
Mains starting time 2011-04-18 05:31:40
Mains ending time 2011-05-22 23:59:08
```

```
preprocessed_start_time='04-18-2011 06:00'
preprocessed_end_time='05-01-2011'
df_appliances=df_appliances[preprocessed_start_time:preprocessed_end_time]
df_mains=df_mains[preprocessed_start_time:preprocessed_end_time]
```

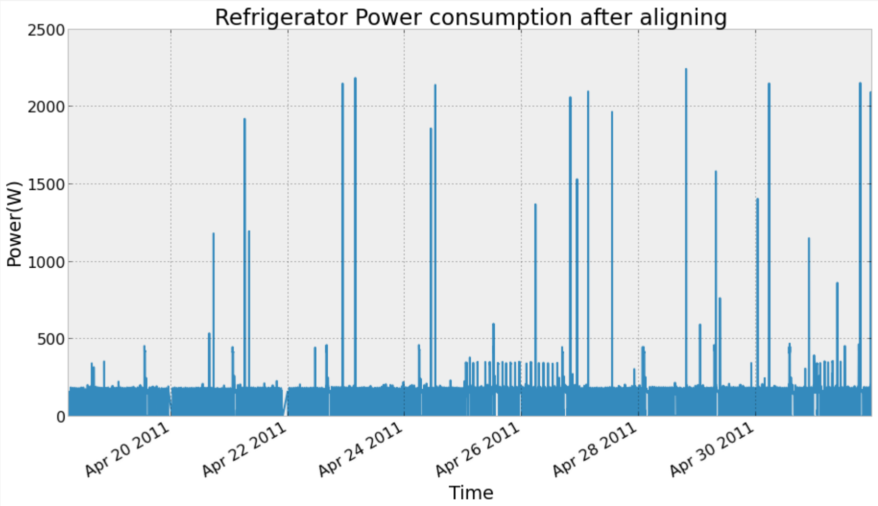
Preprocessing Now plotting the data after aligning the appliances and mains to start at same time

```
df_mains.Mains_1_Power.plot(subplots=False);
plt.gca().set_xlabel('Time');
plt.gca().set_ylabel('Power(W)');
plt.title('Mains 2 Power consumption after aligning');
figure()
df_mains.Mains_2_Power.plot(subplots=False);
plt.gca().set_xlabel('Time');
plt.gca().set_ylabel('Power(W)');
plt.title('Mains 2 Power consumption after aligning');
```






```
df_appliances.refrigerator.plot();
plt.gca().set_xlabel('Time');
plt.gca().set_ylabel('Power(W)');
plt.title('Refrigerator Power consumption after aligning');
```




```
df_appliances_minute=df_appliances.resample('1Min',how='mean')
pd.set_option('display.precision', 2)
print df_appliances_minute.describe().to_string()
```

```
dishwasher  disposal  kitchen  kitchen_2   light  microwave  refrigerator   stove  washer_dryer
count      19580.0   19580.0  19580.0    19580.0  19580.0    19580.0        19580.0  19580.0       19580.0
mean           9.2       0.1      6.2       10.5     26.8       15.5           79.5      1.5           2.2
std           96.0       1.6     34.9       74.1     46.0       97.2           85.4     18.0           0.6
min            0.0       0.0      0.0        0.0      2.0        1.6            1.6      0.0           0.0
25%            0.0       0.0      0.0        1.0      8.0        4.1            6.1      0.1           2.0
50%            0.1       0.0      0.4        1.0      8.6        4.6            7.0      0.5           2.0
```

```
75%          0.1        0.0      13.0         1.0       9.0         5.0            160.7       0.9              2.2
max       1255.6      115.8     794.6      1071.8     185.4      1926.0            598.2     411.0              8.8
```

```
df_mains_minute=df_mains.resample('1Min',how='mean')
df_mains_minute.describe()
```

```
       Mains_1_Power  Mains_2_Power
count        19579.0        19579.0
mean            43.4          188.2
std            129.8          206.7
min             13.4           21.4
25%             14.9           22.5
50%             15.1          214.1
75%             33.7          259.1
max           1312.0         2612.5
```

Downsampling Data As a sanity check, we confirm that 19400 minutes correspond to about 14 days, thus our resampling was correct. We next align mains and appliance time series.

```
print "Number of missing values in Mains",sum(pd.isnull(df_mains_minute.Mains_1_Power))
print "Number of missing values in Appliances",sum(pd.isnull(df_appliances_minute.
    refrigerator))
```

```
Number of missing values in Mains 221
Number of missing values in Appliances 220
```

```
df_appliances_minute.fillna(method='pad',inplace=True)
df_mains_minute.fillna(method='pad',inplace=True)
assert 0==sum(pd.isnull(df_mains_minute.Mains_1_Power))
assert 0==sum(pd.isnull(df_appliances_minute.refrigerator))
```

Filling Missing Data with forward filling

```
total_power=np.sum(df_mains_minute.Mains_1_Power.values+df_mains_minute.Mains_2_Power.
    values)
for appliance in df_appliances_minute:
    print "Contribution of %s to overall load is %f" %(appliance,100*np.sum(
        df_appliances_minute[appliance].values)/total_power)
```

```
Contribution of dishwasher to overall load is 3.918053
Contribution of disposal to overall load is 0.035672
Contribution of kitchen to overall load is 2.706929
Contribution of kitchen_2 to overall load is 4.484684
Contribution of light to overall load is 11.694800
Contribution of microwave to overall load is 6.645737
Contribution of refrigerator to overall load is 34.357508
Contribution of stove to overall load is 0.630176
Contribution of washer_dryer to overall load is 0.933156
```

Finding contributions of different appliances and discarding loads with insignificant contribution Since washer dryer has a peak usage of 8 W, we assume that to be noise in the circuit. Also, disposal has usage of less than .1 % and is thus not further included in the analysis. EVen though stove has less than 0.1 % contribution, it has few instances of being used in high power mode.

Sorting appliances in decreasing order of peak power consumption Creating a dictionary of the form {appliance: peak power} and then sorting it by value

```
appliance_peak_power_dict={}
for appliance in df_appliances_minute:
    if appliance not in ['washer_dryer','disposal']:
        appliance_peak_power_dict[appliance]=df_appliances_minute[appliance].max()
#This variable is named 's' in the algorithm described in the paper
sorted_appliance_list=sorted(appliance_peak_power_dict, key=appliance_peak_power_dict.
    get, reverse=True)
```

```
mapping={}
df_mains_minute_copy=df_mains_minute.copy()
converged=False
```

```
df_mains_minute_copy=df_mains_minute.copy()
converged=False
mapping={}
def thresholding_based_load_assignment(mapping,sorted_appliance_list,converged,TOLERANCE
    ):
    iteration=-1
    while converged==False:
        before_iteration_len_mapping=len(mapping.keys())
        iteration+=1
        for load in sorted_appliance_list:
            if load not in mapping:
                #Since mains 1 has less average load we check it first
                #Note we may need some tolerance as 1-2 values may be off, but more than
                    20 odd values satisying
                #would mean that we can safely assign the other mains

                c1=len(np.where((df_appliances_minute[load].values-df_mains_minute_copy.
                    Mains_1_Power.values)>0)[0])
                c2=len(np.where((df_appliances_minute[load].values-df_mains_minute_copy.
                    Mains_2_Power.values)>0)[0])
                #Using a copy of the Mains Minute DF rather than the original

                if c1>TOLERANCE and c2<TOLERANCE:
                    mapping[load]=2
                elif c2>TOLERANCE and c1<TOLERANCE:
                    mapping[load]=1

                #Subtracting Load from the assigned Mains and filling in zeros for
```

```
                #few spots where appliance value might be more due to error
            if load in mapping:
                #Load has been assined
                if mapping[load]==1:
                    df_mains_minute_copy.Mains_1_Power=df_mains_minute_copy.Mains_1_Power-
                        df_appliances_minute[load]
                    df_mains_minute_copy.Mains_1_Power[df_mains_minute_copy.Mains_1_Power
                        <0]=0
                else:
                    df_mains_minute_copy.Mains_2_Power=df_mains_minute_copy.Mains_2_Power-
                        df_appliances_minute[load]
                    df_mains_minute_copy.Mains_2_Power[df_mains_minute_copy.Mains_2_Power
                        <0]=0

        print "After iteration %d" %iteration
        print mapping
        after_iteration_len_mapping=len(mapping.keys())
        if after_iteration_len_mapping==before_iteration_len_mapping:
            converged=True
```

```
thresholding_based_load_assignment(mapping,sorted_appliance_list,False,300)
```

```
After iteration 0
{'refrigerator': 2, 'light': 2, 'kitchen_2': 1, 'microwave': 2}
After iteration 1
{'refrigerator': 2, 'light': 2, 'dishwasher': 1, 'kitchen_2': 1, 'microwave': 2}
After iteration 2
{'refrigerator': 2, 'light': 2, 'dishwasher': 1, 'kitchen_2': 1, 'microwave': 2}
```

Load Assignment to Mains Now we need to find according to step change the load assignment

Only 'kitchen' and 'stove' have not yet been separated into mains. Finding step changes of more than 15 W threshold in these loads and seeing the times at which these step changes take place.

```
def step_changes(series,threshold):
    diff_series=np.diff(series.values)
    print diff_series
    return {"magnitude":diff_series,"times":series[np.abs(diff_series)>threshold].index}
    #Finding times at which these step changes take place
```

```
step_threshold=30
for load in sorted_appliance_list:
    if load not in mapping:

        step_load=set(step_changes(df_appliances_minute[load],step_threshold)["times"]);
        step_mains_1=set(step_changes(df_mains_minute.Mains_1_Power,step_threshold)["
            times"]);
        step_mains_2=set(step_changes(df_mains_minute.Mains_2_Power,step_threshold)["
            times"]);
```

```
        #Find intersection with Mains 1 and 2
        l1=len(step_load.intersection(step_mains_1));
        l2=len(step_load.intersection(step_mains_2));

        if 1.0*l1/len(step_load)>0.9:
            #More than 90% events detected
            mapping[load]=1
        elif 1.0*l2/len(step_load)>0.9:
            #More than 90% events detected
            mapping[load]=2
        else:
            print "Load could not be assigned"
```

```
[ 0.       0.       0.      ..., -0.0625  0.0625 -0.125 ]
[ 0.07533333 -0.10833333  0.07716667 ...,  0.01706349 -0.19354497
  0.02144108]
[-0.37516667 -0.38566667 -0.10566667 ..., -0.51166667 -0.44259259
  0.21613805]
[ 0.1875 -0.1875  0.25   ...,  0.0625 -0.125   0.1875]
[ 0.07533333 -0.10833333  0.07716667 ...,  0.01706349 -0.19354497
  0.02144108]
[-0.37516667 -0.38566667 -0.10566667 ..., -0.51166667 -0.44259259
  0.21613805]
```

```
print "Appliance to Mains Mapping\n"
pprint.pprint(mapping)
```

```
Appliance to Mains Mapping

{'dishwasher': 1,
 'kitchen': 1,
 'kitchen_2': 1,
 'light': 2,
 'microwave': 2,
 'refrigerator': 2,
 'stove': 1}
```

```
df_mains_train=df_mains_minute[:'2011-4-24']
df_mains_test=df_mains_minute['2011-4-25':]
df_appliances_train=df_appliances_minute[:'2011-4-24']
df_appliances_test=df_appliances_minute['2011-4-25':]
```

Dividing Data into TEST and TRAIN

```
plt.figsize(15,8)
matplotlib.rcParams.update({'font.size': 24})
```

Perform KMeans++ and Mini Batch KMeans on Train set Reshaping data so that it can be used by clustering algos

```
raw_data={}
for key in df_appliances_train:
    raw_data[key]=df_appliances_train[key].values
    length=len(raw_data[key])
    raw_data[key]=raw_data[key].reshape(length,1)
centroids={}
```

```
batch_size=1000
def apply_kmeans(n_clusters, n_init,X,init=None):
    matplotlib.rcParams.update({'font.size': 16})
    if init is None:
        k_means = KMeans(n_clusters=n_clusters, n_init=n_init)
    else:
        k_means=KMeans(init='k-means++',n_clusters=n_clusters, n_init=n_init)
    t0 = time.time()
    k_means.fit(X)
    t_batch = time.time() - t0
    k_means_labels = k_means.labels_
    k_means_cluster_centers = k_means.cluster_centers_
    k_means_labels_unique = np.unique(k_means_labels)
    k_means_inertia=k_means.inertia_
    mbk = MiniBatchKMeans(init='k-means++', n_clusters=n_clusters, batch_size=batch_size,
                    n_init=n_init, max_no_improvement=10, verbose=0)
    t0 = time.time()
    mbk.fit(X)
    t_mini_batch = time.time() - t0
    mbk_means_labels = mbk.labels_
    mbk_means_cluster_centers = mbk.cluster_centers_
    mbk_means_labels_unique = np.unique(mbk_means_labels)
    mbk_inertia=mbk.inertia_
    return [t_batch, t_mini_batch, k_means_labels, mbk_means_labels,
        k_means_cluster_centers, mbk_means_cluster_centers,\
    k_means_labels_unique,mbk_means_labels_unique, k_means_inertia, mbk_inertia]
```

```
def plot_cluster_assignments(X,k_means_labels, mbk_means_labels,k_means_cluster_centers,
    mbk_means_cluster_centers,n_clusters,appliance_name):
    colors = ['#4EACC5', '#FF9C34', '#4E9A06']
    markers=['o','*','.']
    x_temp=np.arange(len(X))
    plt.subplot(2,2,1)
    #plt.rcParams["font.size"]=16
    print "\nKMeans Analysis for %s\n" %appliance_name
    print "-"*80
    centroids[appliance_name]=[]
    for k, col in zip(range(n_clusters), colors):
        my_members = k_means_labels == k
        cluster_center = k_means_cluster_centers[k]
```

```
        centroids[appliance_name].append(k_means_cluster_centers[k][0])
        plt.ylabel('Power (W)');
        plt.plot(x_temp[my_members],X[my_members, 0],markers[k],markersize=20,
            markerfacecolor=col)
        plt.axhline(k_means_cluster_centers[k],linewidth=6,color=col)
        print "State %d Centroid= %0.4f, Fraction of datapoints= %0.4f" %(k,
            cluster_center,sum(my_members)*1.0/np.size(X))
        plt.title('KMeans Cluster Assignment for '+appliance_name+' for K='+str(
            n_clusters))
        plt.xlabel("Samples")
    plt.subplot(2,2,2)
    plt.tight_layout()
    print "\nMini Batch KMeans Analysis\n"
    print "-"*80
    for k, col in zip(range(n_clusters), colors):
        my_members = mbk_means_labels == k
        cluster_center = mbk_means_cluster_centers[k]
        plt.ylabel('Power (W)');
        plt.plot(x_temp[my_members],X[my_members, 0],markers[k],markersize=15,
            markerfacecolor=col)
        plt.axhline(mbk_means_cluster_centers[k],linewidth=3,color=col)
        print "State %d Centroid= %0.4f, Fraction of datapoints= %0.4f" %(k,
            cluster_center,sum(my_members)*1.0/np.size(X))
        #plt.title('Mini Batch KMeans Cluster Assignment for '+appliance_name+' for K='+
            str(n_clusters))
    print "-"*80
```

Peforming clustering for different appliance.
Caveat: Number of states must be known beforehand

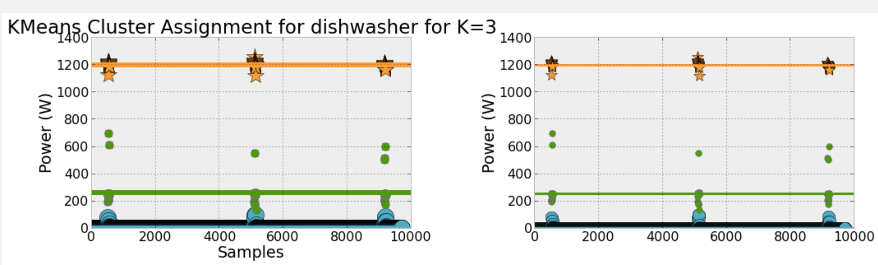
```
appliances_mains=[[],[]]
appliances_states=[[3,2,3,2],[3,3,3]]
appliances_mains[0]=[key for key in mapping if mapping[key]==1]
appliances_mains[1]=[key for key in mapping if mapping[key]==2]
labels_appliance={}
```

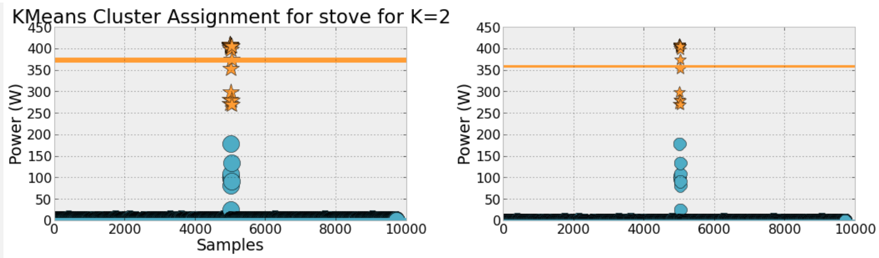
```
for i in range(len(appliances_mains[0])):
    figure()
    [t_batch, t_mini_batch, k_means_labels, mbk_means_labels, k_means_cluster_centers,
        mbk_means_cluster_centers,\
    k_means_labels_unique,mbk_means_labels_unique, k_means_inertia, mbk_inertia]=
        apply_kmeans(appliances_states[0][i],10,raw_data[appliances_mains[0][i]],"kmeans
        ++")
    plot_cluster_assignments(raw_data[appliances_mains[0][i]],k_means_labels,
        mbk_means_labels, k_means_cluster_centers, mbk_means_cluster_centers\
    ,appliances_states[0][i],appliances_mains[0][i])
    print "Time taken for Kmeans clustering : \n",t_batch
    print "Inertia of KMeans cluster assignment: \n",k_means_inertia
    print "Time taken for Mini Batch Kmeans clustering : \n",t_mini_batch
    print "Inertia of Mini Batch KMeans cluster assignment: \n",mbk_inertia
    flattened=k_means_cluster_centers.flatten()
```

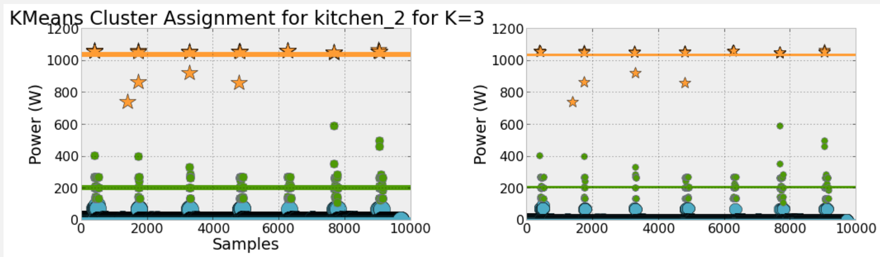
```
    sorted_list=sort(flattened)
    labels=[]
    for label in k_means_labels:
        labels.append(sorted_list.tolist().index(flattened[label]))
    labels_appliance[appliances_mains[0][i]]=np.array(labels)
```

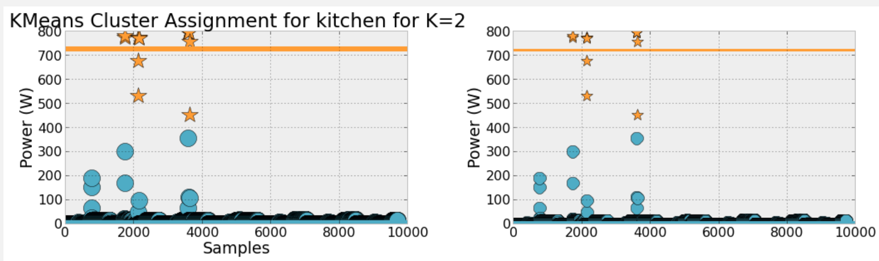
```
KMeans Analysis for dishwasher

--------------------------------------------------------------------------------
State 0 Centroid= 0.1701, Fraction of datapoints= 0.9849
State 1 Centroid= 1195.3714, Fraction of datapoints= 0.0074
State 2 Centroid= 260.5342, Fraction of datapoints= 0.0077

Mini Batch KMeans Analysis
--------------------------------------------------------------------------------
State 0 Centroid= 0.1734, Fraction of datapoints= 0.9849
State 1 Centroid= 1197.7946, Fraction of datapoints= 0.0074
State 2 Centroid= 253.5010, Fraction of datapoints= 0.0077
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.0889909267426
Inertia of KMeans cluster assignment:
817407.717216
Time taken for Mini Batch Kmeans clustering    :
0.0398271083832
Inertia of Mini Batch KMeans cluster assignment:
821540.486286

KMeans Analysis for stove
--------------------------------------------------------------------------------
State 0 Centroid= 0.6031, Fraction of datapoints= 0.9977
State 1 Centroid= 373.8506, Fraction of datapoints= 0.0023

Mini Batch KMeans Analysis
--------------------------------------------------------------------------------
State 0 Centroid= 0.5808, Fraction of datapoints= 0.9977
State 1 Centroid= 359.1779, Fraction of datapoints= 0.0023
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.0339879989624
Inertia of KMeans cluster assignment:
159053.555617
Time taken for Mini Batch Kmeans clustering    :
0.050402879715
Inertia of Mini Batch KMeans cluster assignment:
163794.675347

KMeans Analysis for kitchen_2
--------------------------------------------------------------------------------
State 0 Centroid= 1.4256, Fraction of datapoints= 0.9737
```

```
State 1 Centroid= 1036.0541, Fraction of datapoints= 0.0044
State 2 Centroid= 204.7067, Fraction of datapoints= 0.0219

Mini Batch KMeans Analysis


--------------------------------------------------------------------------------
State 0 Centroid= 1.4235, Fraction of datapoints= 0.9737
State 1 Centroid= 1034.3149, Fraction of datapoints= 0.0044
State 2 Centroid= 204.0398, Fraction of datapoints= 0.0219
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.0543367862701
Inertia of KMeans cluster assignment:
1298012.8041
Time taken for Mini Batch Kmeans clustering    :
0.0346150398254
Inertia of Mini Batch KMeans cluster assignment:
1298237.62336

KMeans Analysis for kitchen


--------------------------------------------------------------------------------
State 0 Centroid= 5.0766, Fraction of datapoints= 0.9987
State 1 Centroid= 727.4287, Fraction of datapoints= 0.0013

Mini Batch KMeans Analysis


--------------------------------------------------------------------------------
State 0 Centroid= 5.0800, Fraction of datapoints= 0.9987
State 1 Centroid= 720.9898, Fraction of datapoints= 0.0013
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.0305271148682
Inertia of KMeans cluster assignment:
875870.13267
Time taken for Mini Batch Kmeans clustering    :
0.0231289863586
Inertia of Mini Batch KMeans cluster assignment:
876409.211242
```







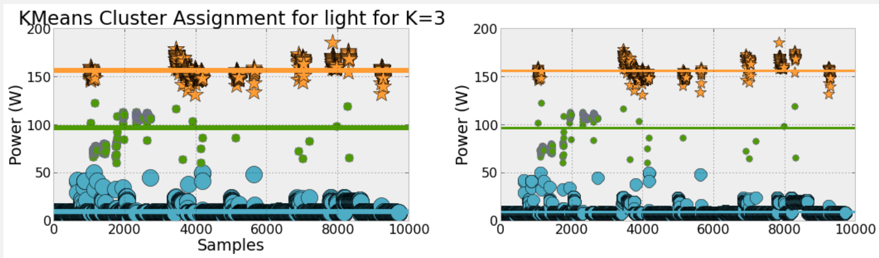


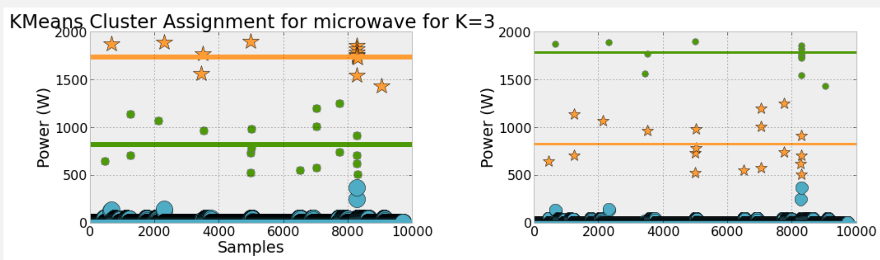

```python
for i in range(len(appliances_mains[1])):
    figure()
    [t_batch, t_mini_batch, k_means_labels, mbk_means_labels, k_means_cluster_centers,
        mbk_means_cluster_centers,\
    k_means_labels_unique,mbk_means_labels_unique, k_means_inertia, mbk_inertia]=
        apply_kmeans(appliances_states[1][i],10,raw_data[appliances_mains[1][i]],"kmeans
        ++")
    plot_cluster_assignments(raw_data[appliances_mains[1][i]],k_means_labels,
        mbk_means_labels, k_means_cluster_centers, mbk_means_cluster_centers\
    ,appliances_states[1][i],appliances_mains[1][i])
    print "Time taken for Kmeans clustering : \n",t_batch
    print "Inertia of KMeans cluster assignment: \n",k_means_inertia
    print "Time taken for Mini Batch Kmeans clustering : \n",t_mini_batch
    print "Inertia of Mini Batch KMeans cluster assignment: \n",mbk_inertia
    flattened=k_means_cluster_centers.flatten()
    sorted_list=sort(flattened)
    labels=[]
    for label in k_means_labels:
        labels.append(sorted_list.tolist().index(flattened[label]))
    labels_appliance[appliances_mains[1][i]]=np.array(labels)
```

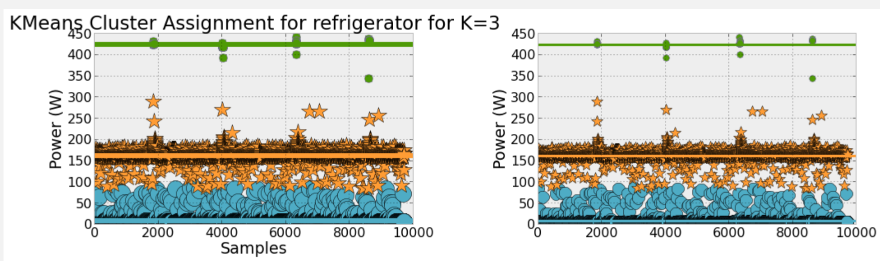
```
KMeans Analysis for light

--------------------------------------------------------------------------------
State 0 Centroid= 9.4713, Fraction of datapoints= 0.8435
```

```
State 1 Centroid= 156.3087, Fraction of datapoints= 0.0987
State 2 Centroid= 96.8437, Fraction of datapoints= 0.0578

Mini Batch KMeans Analysis
--------------------------------------------------------------------------------
State 0 Centroid= 9.4679, Fraction of datapoints= 0.8435
State 1 Centroid= 156.2838, Fraction of datapoints= 0.0987
State 2 Centroid= 96.7811, Fraction of datapoints= 0.0578
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.0586891174316
Inertia of KMeans cluster assignment:
316625.182718
Time taken for Mini Batch Kmeans clustering    :
0.0635697841644
Inertia of Mini Batch KMeans cluster assignment:
316628.075094

KMeans Analysis for microwave
--------------------------------------------------------------------------------
State 0 Centroid= 9.7742, Fraction of datapoints= 0.9966
State 1 Centroid= 1740.2292, Fraction of datapoints= 0.0014
State 2 Centroid= 822.5147, Fraction of datapoints= 0.0020

Mini Batch KMeans Analysis
--------------------------------------------------------------------------------
State 0 Centroid= 9.7108, Fraction of datapoints= 0.9966
State 1 Centroid= 826.5927, Fraction of datapoints= 0.0020
State 2 Centroid= 1785.4789, Fraction of datapoints= 0.0014
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.057156085968
Inertia of KMeans cluster assignment:
3282726.04781
Time taken for Mini Batch Kmeans clustering    :
0.0310339927673
Inertia of Mini Batch KMeans cluster assignment:
3311746.56209

KMeans Analysis for refrigerator
--------------------------------------------------------------------------------
State 0 Centroid= 7.4547, Fraction of datapoints= 0.6079
State 1 Centroid= 161.8361, Fraction of datapoints= 0.3856
State 2 Centroid= 423.5390, Fraction of datapoints= 0.0065

Mini Batch KMeans Analysis
--------------------------------------------------------------------------------
State 0 Centroid= 7.4109, Fraction of datapoints= 0.6078
State 1 Centroid= 161.7781, Fraction of datapoints= 0.3857
```

```
State 2 Centroid= 423.8538, Fraction of datapoints= 0.0065
--------------------------------------------------------------------------------
Time taken for Kmeans clustering    :
0.0458850860596
Inertia of KMeans cluster assignment:
864283.719899
Time taken for Mini Batch Kmeans clustering    :
0.0460529327393
Inertia of Mini Batch KMeans cluster assignment:
864304.529122
```








```python
centroids
```

```
{'dishwasher': [0.1701359028503493, 1195.3713848039199, 260.53415359477123],
 'kitchen': [5.0765980080727378, 727.42865761689313],
 'kitchen_2': [1.4255969429011941, 1036.0540697674414, 204.70668553806448],
 'light': [9.4712692617067837, 156.30873674287119, 96.843738224826836],
 'microwave': [9.7741967140592507, 1740.2291666666672, 822.51469298245615],
 'refrigerator': [7.4547324466263163, 161.83612766450091, 423.53895502645537],
 'stove': [0.60312354161529014, 373.85056818181823]}
```

Sorting centroids and also making the datatype integer

```python
for appliance in centroids:
    centroids[appliance]=list(np.array(centroids[appliance]).astype(int))
```

```
    centroids[appliance].sort()
centroids
```

```
{'dishwasher': [0, 260, 1195],
 'kitchen': [5, 727],
 'kitchen_2': [1, 204, 1036],
 'light': [9, 96, 156],
 'microwave': [9, 822, 1740],
 'refrigerator': [7, 161, 423],
 'stove': [0, 373]}
```

Calibration No need for calibrating state 1 of each appliance since their power is close to 0.

Finding events of transition from state 1 to 2 in Mains and in Appliance and comparing them with cluster centroids assigned. Only if there are greater than 10 number of transitions do we even consider the calibration.

```

calib_centroids=deepcopy(centroids)
```

```
for appliance in labels_appliance:
    l=labels_appliance[appliance]
    #print "%s has %d states" %(appliance,max(l)+1)

    #Finding factors for 2,..K th state
    #Note here indexing starts from 0 and not 1
    for k in range(1,max(l)+1):
        #Finding idx of where appliance is in state k-1 and next state is k
        idx=[]
        for i in range(len(l)-1):
            if l[i]==k-1 and l[i+1]==k:
                idx.append(i)

                diff_appliance=np.diff(df_appliances_train[appliance].values)
                diff_mains_1=np.diff(df_mains_train.Mains_1_Power.values)
                diff_mains_2=np.diff(df_mains_train.Mains_2_Power.values)


                if appliance in appliances_mains[0]:
                    x=diff_mains_1[idx]/diff_appliance[idx]
                    x=x[x<2]
                    x=x[x>0]


                else:
                    x=diff_mains_2[idx]/diff_appliance[idx]
                    x=x[x<2]
                    x=x[x>0]
        #print x,np.std(x)
        if (np.average(x)<0.9 or np.average(x)>1.1) and len(x)>10:
            calib_centroids[appliance][k]=centroids[appliance][k]*np.average(x)
```

```
            print "Average for %s in %dth state is %f and number of transitions was %d" %(
                appliance,k+1,np.average(x),len(idx))
```

```
Average for light in 2th state is 1.178171 and number of transitions was 16
Average for refrigerator in 2th state is 1.330662 and number of transitions was 178
```

After calibration the power in different states is given below.

```
for appliance in calib_centroids:
    calib_centroids[appliance]=list(np.array(calib_centroids[appliance]).astype(int))

pprint.pprint(calib_centroids)
```

```
{'dishwasher': [0, 260, 1195],
 'kitchen': [5, 727],
 'kitchen_2': [1, 204, 1036],
 'light': [9, 113, 156],
 'microwave': [9, 822, 1740],
 'refrigerator': [7, 214, 423],
 'stove': [0, 373]}
```

```
def find_nearest_index(array,value):
    idx = (np.abs(array-value)).argmin()
    return idx
def find_labels(appliance_power_consumption_list, observed_power):
    labels=np.zeros(len(observed_power))
    appliance_power_consumption_array=np.array(appliance_power_consumption_list)
    for i in range(len(observed_power)):
        labels[i]=find_nearest_index(appliance_power_consumption_array,observed_power[i])
    return labels
```

CO based NIALM Finding true labels of appliances on the test set

```
true_labels={}
for appliance in centroids:
    true_labels[appliance]=find_labels(centroids[appliance], df_appliances_test[appliance
        ])
```

```
def print_confusion_matrix(appliance,num_states,true_label,observed_label):
    correct_predicted=0
    conf_arr=[]
    for i in range(num_states):
        counts=[]
        for j in range(num_states):
            idx=np.where(true_label==i)[0]
            counts.append(len(np.where(observed_label[idx]==j)[0]))
        correct_predicted+=counts[i]
```

```
        conf_arr.append(counts)

    norm_conf = []
    for i in conf_arr:
        a = 0
        tmp_arr = []
        a = sum(i, 0)
        for j in i:
            tmp_arr.append(float(j)/float(a))
        norm_conf.append(tmp_arr)

    fig = plt.figure()
    plt.clf()
    ax = fig.add_subplot(111)
    ax.set_aspect(1)
    res = ax.imshow(np.array(norm_conf), cmap=plt.cm.jet,
                    interpolation='nearest')

    width = len(conf_arr)
    height = len(conf_arr[0])

    for x in xrange(width):
        for y in xrange(height):
            ax.annotate(str(conf_arr[x][y]), xy=(y, x),
                        horizontalalignment='center',
                        verticalalignment='center')

    cb = fig.colorbar(res)
    alphabet = ['State 1','State 2','State 3']
    plt.title('Confusion Matrix for '+appliance)
    plt.xticks(range(width), alphabet[:width])
    plt.yticks(range(height), alphabet[:height])
    plt.show()
    return correct_predicted*1.0/len(true_label)
```

0.1.1 Without Calibration

```
def find_nearest(array,value):
    idx = (np.abs(array-value)).argmin()
    diff=array[idx]-value
    return [idx,-diff]
```

```
states_combination=list(itertools.product(centroids['kitchen'],centroids['stove'],\
centroids['kitchen_2'],centroids['dishwasher'],centroids['refrigerator'],centroids['
    light'],\
centroids['microwave']))
sum_combination=np.array(np.zeros(len(states_combination)))
for i in range(0,len(states_combination)):
    sum_combination[i]=sum(states_combination[i])
```

```python
length_sequence=len(df_mains_test.Mains_1_Power.values)
states=np.zeros(length_sequence)
residual_power=np.zeros(length_sequence)
t0=time.time()
for i in range(length_sequence):
    [states[i],residual_power[i]]=find_nearest(sum_combination,df_mains_test.
        Mains_2_Power.values[i]+df_mains_test.Mains_1_Power.values[i])
t1=time.time()
print "Time taken for CO Mains 2 :",t1-t0
```

```
Time taken for CO Mains 2 : 0.26970911026
```

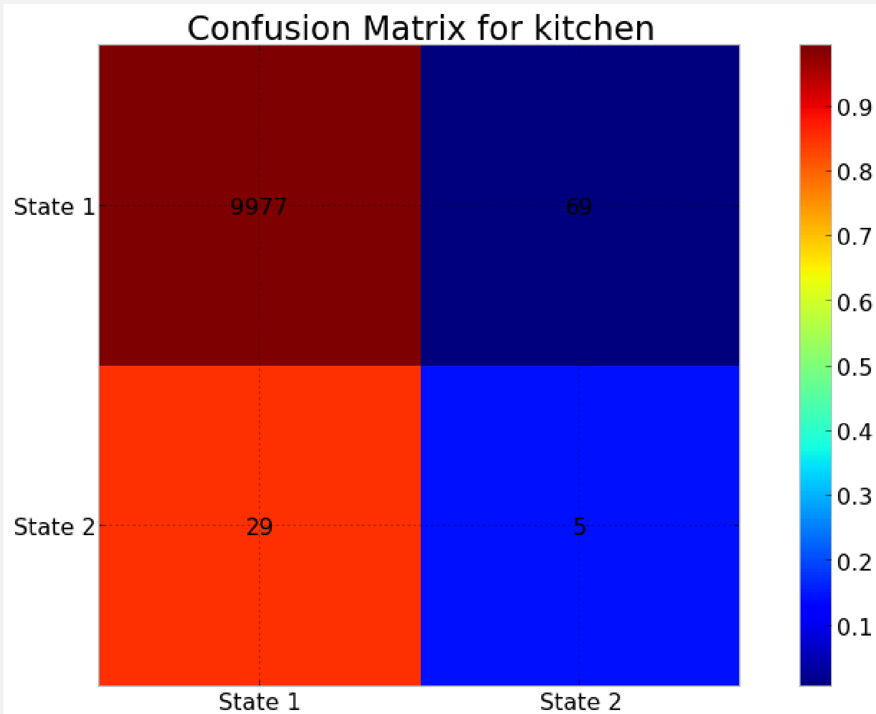
```python
def decode_CO(length_sequence,centroids,appliance_list,states,residual_power):

    co_states={}
    co_power={}
    total_num_combinations=1
    for appliance in appliance_list:
        total_num_combinations*=len(centroids[appliance])


    for appliance in appliance_list:
        co_states[appliance]=np.zeros(length_sequence,dtype=np.int)
        co_power[appliance]=np.zeros(length_sequence)

    for i in range(length_sequence):
        factor=total_num_combinations
        for appliance in appliance_list:
            #assuming integer division (will cause errors in Python 3x)
            factor=factor/len(centroids[appliance])

            temp=int(states[i])/factor
            co_states[appliance][i]=temp%len(centroids[appliance])
            co_power[appliance][i]=centroids[appliance][co_states[appliance][i]]

    return [co_states,co_power]
```

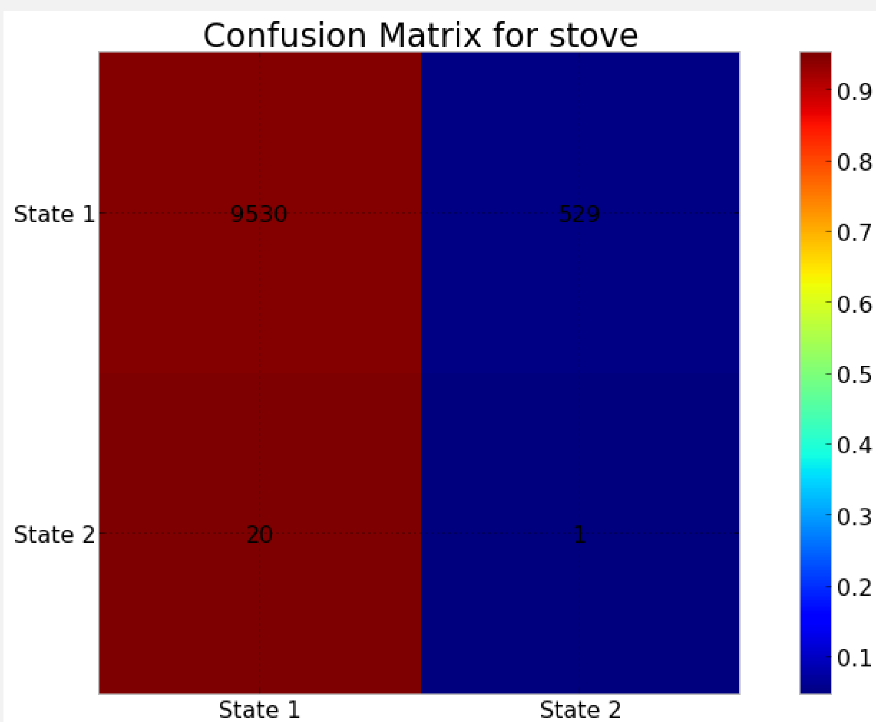
```python
appliance_list=['kitchen','stove','kitchen_2','dishwasher','refrigerator','light','
    microwave']
```

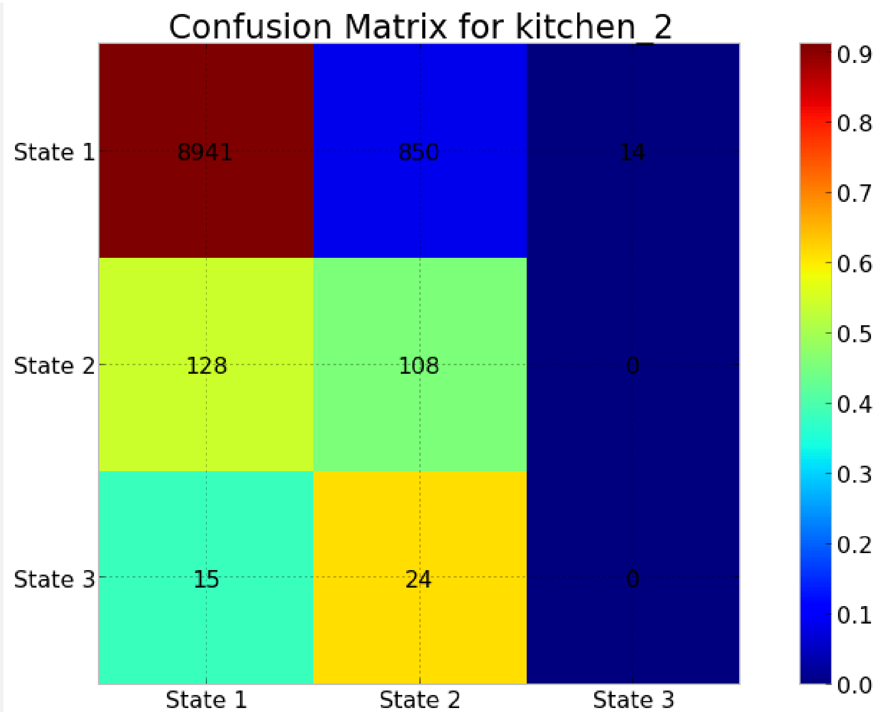
```python
[case_1_states,case_1_power]=decode_CO(length_sequence,centroids,appliance_list,states,
    residual_power)
```

Without load division Printing confusion matrices

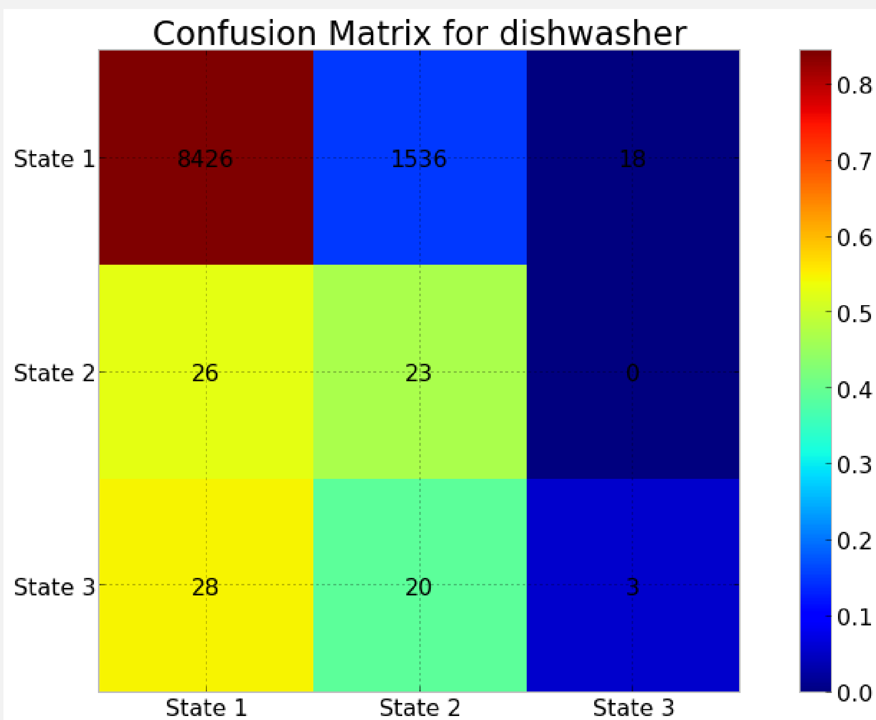
```python
for appliance in appliance_list:
    print_confusion_matrix(appliance,len(centroids[appliance]),true_labels[appliance],
        case_1_states[appliance])
```


Confusion Matrix for kitchen
State 1
State 2
9977
69
29
5
State 1
State 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

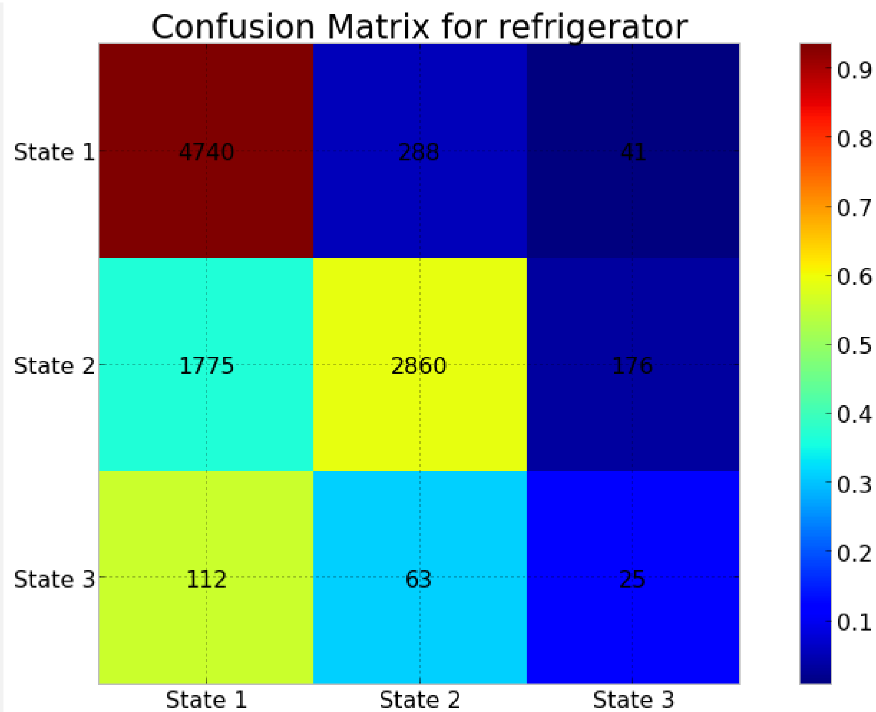

Confusion Matrix for stove
State 1
State 2
9530
529
20
1
State 1
State 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

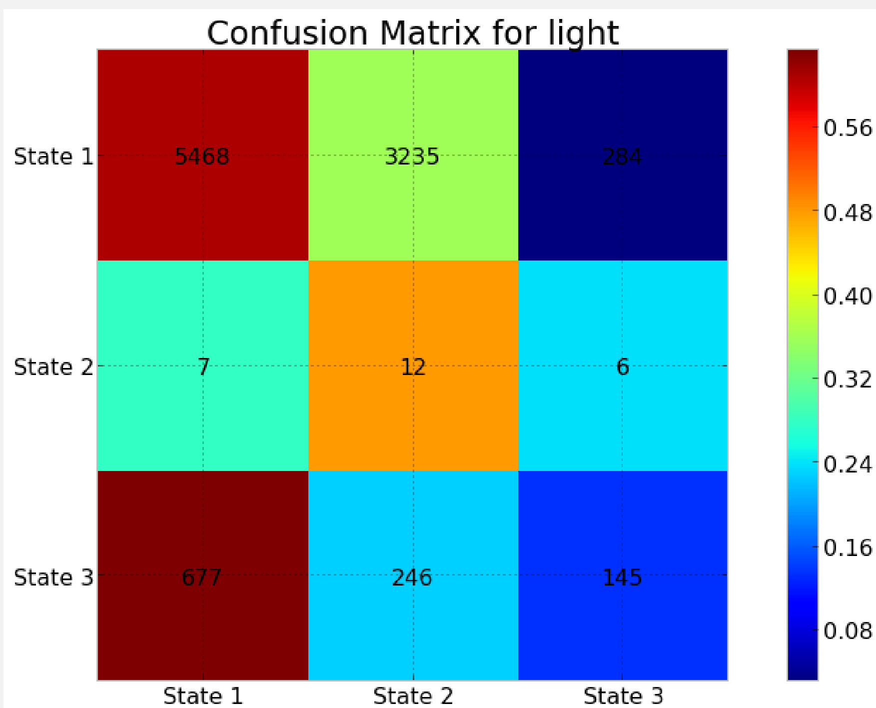


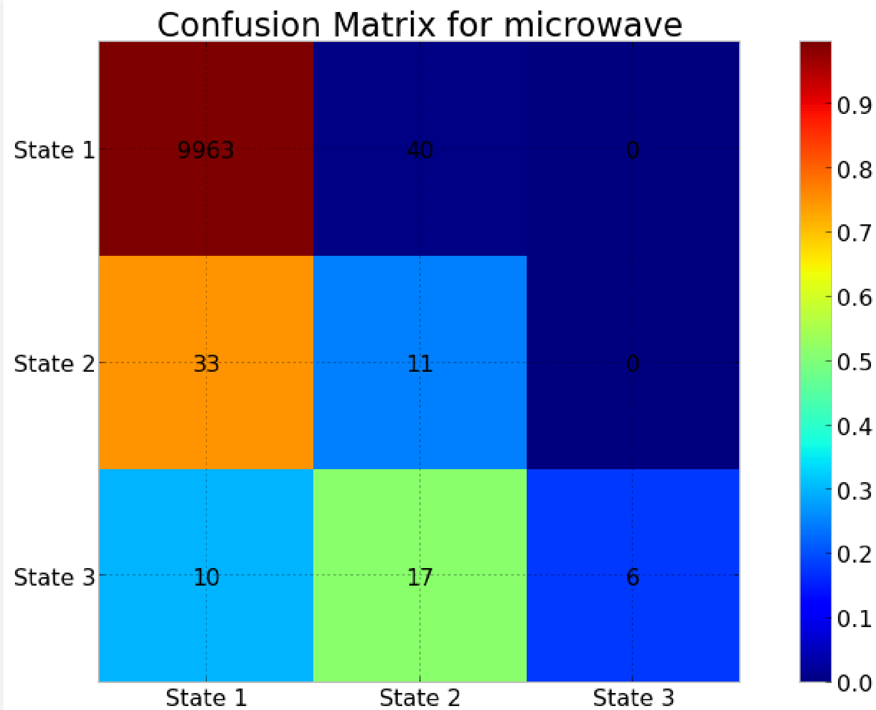
Confusion Matrix for kitchen_2
State 1
State 2
State 3
8941
850
14
128
108
0
15
24
0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0



Confusion Matrix for dishwasher
State 1
State 2
State 3
8426
1536
18
26
23
0
28
20
3
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Confusion Matrix for refrigerator
State 1
State 2
State 3
4740
288
41
1775
2860
176
112
63
25
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix for light
State 1
State 2
State 3
5468
3235
284
7
12
6
677
246
145
0.56
0.48
0.40
0.32
0.24
0.16
0.08




```
MNE=[]
RE=[]
```

```
numerator={}
denominator={}
mne={}
re={}
for appliance in case_1_states:
    numerator[appliance]=np.sum(np.abs(case_1_power[appliance]-df_appliances_test[
        appliance].values))
    denominator[appliance]=np.sum(df_appliances_test[appliance].values)
    mne[appliance]=numerator[appliance]*1.0/denominator[appliance]
    re[appliance]=np.std(case_1_power[appliance]-df_appliances_test[appliance].values)

MNE.append(deepcopy(mne))
RE.append(deepcopy(re))
```

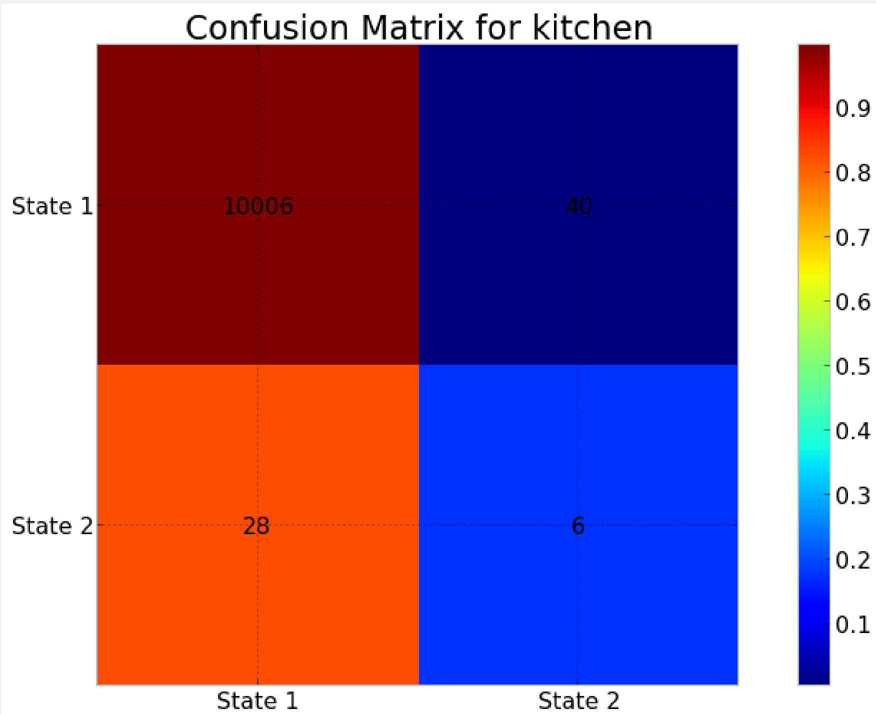
With load division Mains 1

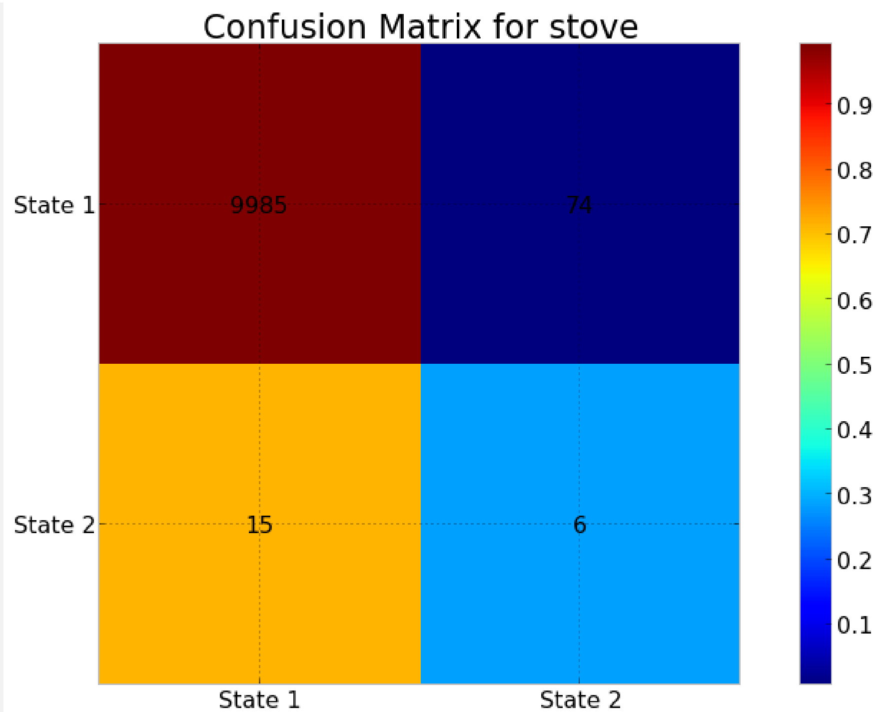
```
states_combination=list(itertools.product(centroids['kitchen'],centroids['stove'],\
centroids['kitchen_2'],centroids['dishwasher']))
sum_combination=np.array(np.zeros(len(states_combination)))
for i in range(0,len(states_combination)):
    sum_combination[i]=sum(states_combination[i])

length_sequence=len(df_mains_test.Mains_1_Power.values)
```

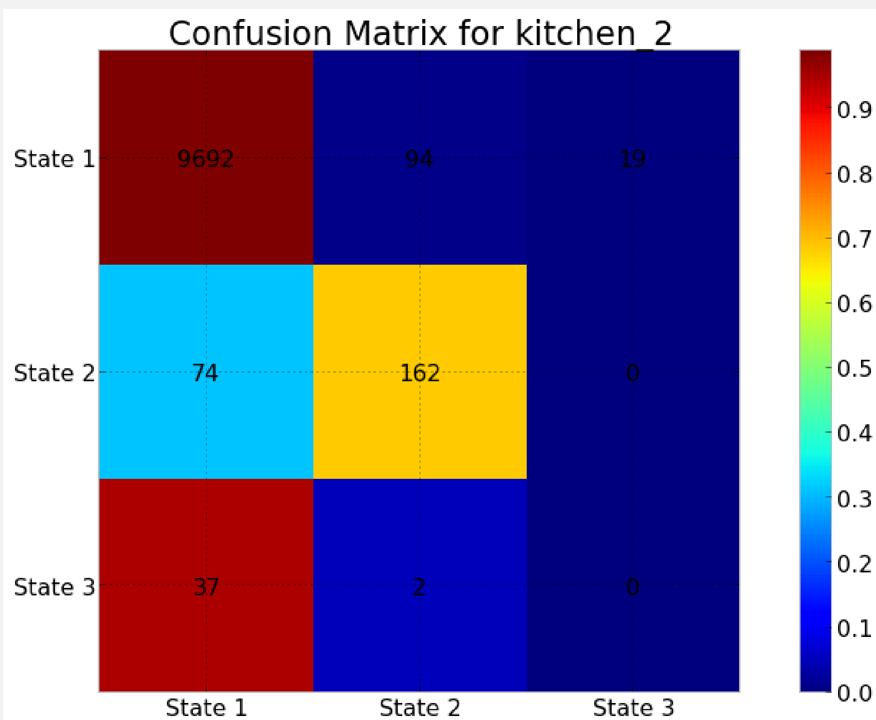
```
states=np.zeros(length_sequence)
residual_power=np.zeros(length_sequence)
t0=time.time()
for i in range(length_sequence):
    [states[i],residual_power[i]]=find_nearest(sum_combination,df_mains_test.
        Mains_1_Power.values[i])
t1=time.time()
print "Time taken for CO Mains :",t1-t0

appliance_list=['kitchen','stove','kitchen_2','dishwasher']


[case_2_mains_1_states,case_2_mains_1_power]=decode_CO(length_sequence,centroids,
    appliance_list,states,residual_power)
```

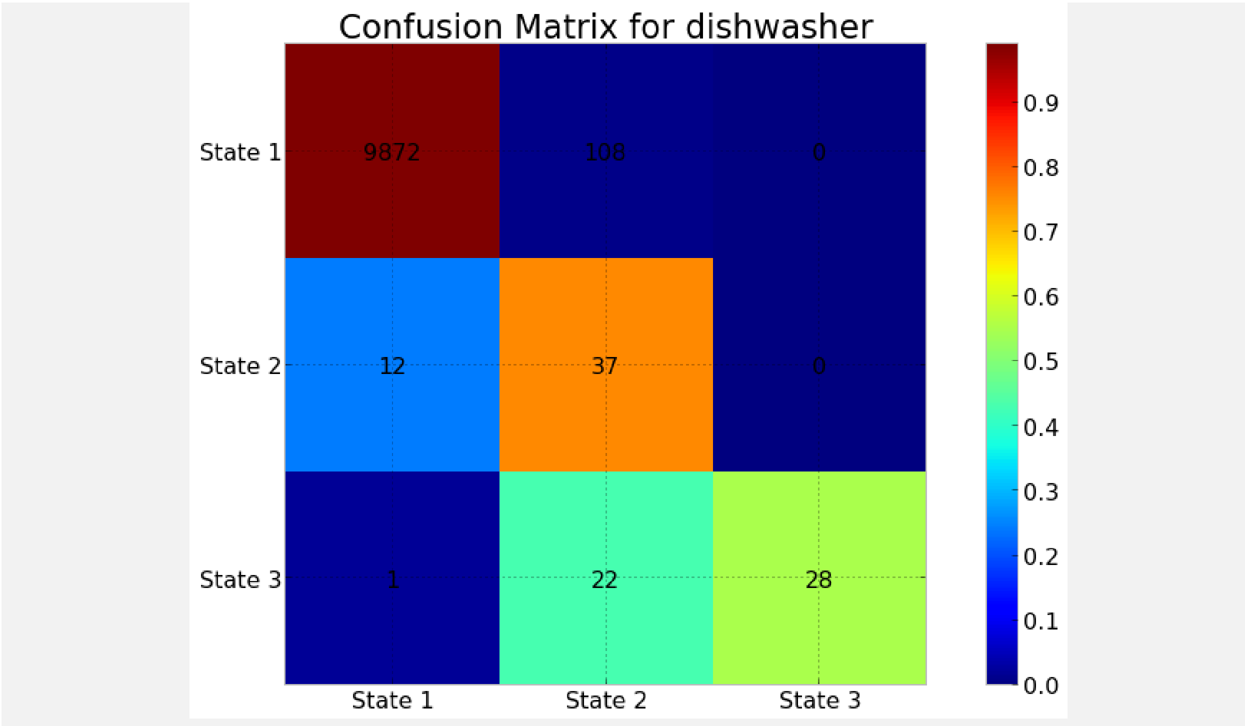
```
Time taken for CO Mains : 0.169198036194
```

```
for appliance in appliance_list:
    print_confusion_matrix(appliance,len(centroids[appliance]),true_labels[appliance],
        case_2_mains_1_states[appliance])
```




Confusion Matrix for stove
State 1
State 2
9985
74
15
6
State 1
State 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix for kitchen_2
State 1
State 2
State 3
9692
94
19
74
162
0
37
2
0
State 1
State 2
State 3
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0




```
numerator={}
denominator={}
mne={}
re={}
for appliance in appliance_list:
    numerator[appliance]=np.sum(np.abs(case_2_mains_1_power[appliance]-df_appliances_test
        [appliance].values))
    denominator[appliance]=np.sum(df_appliances_test[appliance].values)
    mne[appliance]=numerator[appliance]*1.0/denominator[appliance]
    re[appliance]=np.std(case_2_mains_1_power[appliance]-df_appliances_test[appliance].
        values)
```

Mains 2

```
states_combination=list(itertools.product(centroids['refrigerator'],centroids['light'],\
centroids['microwave']))
sum_combination=np.array(np.zeros(len(states_combination)))
for i in range(0,len(states_combination)):
    sum_combination[i]=sum(states_combination[i])
```

```
length_sequence=len(df_mains_test.Mains_2_Power.values)
states=np.zeros(length_sequence)
residual_power=np.zeros(length_sequence)
t0=time.time()
for i in range(length_sequence):
    [states[i],residual_power[i]]=find_nearest(sum_combination,df_mains_test.
```

```
        Mains_2_Power.values[i])
t1=time.time()
print "Time taken for CO Mains :",t1-t0
print residual_power
```

```
Time taken for CO Mains : 0.169050931931
[-14.93716667 -15.91083333 -16.59033333 ...,  25.695       25.25240741
  25.46854545]
```

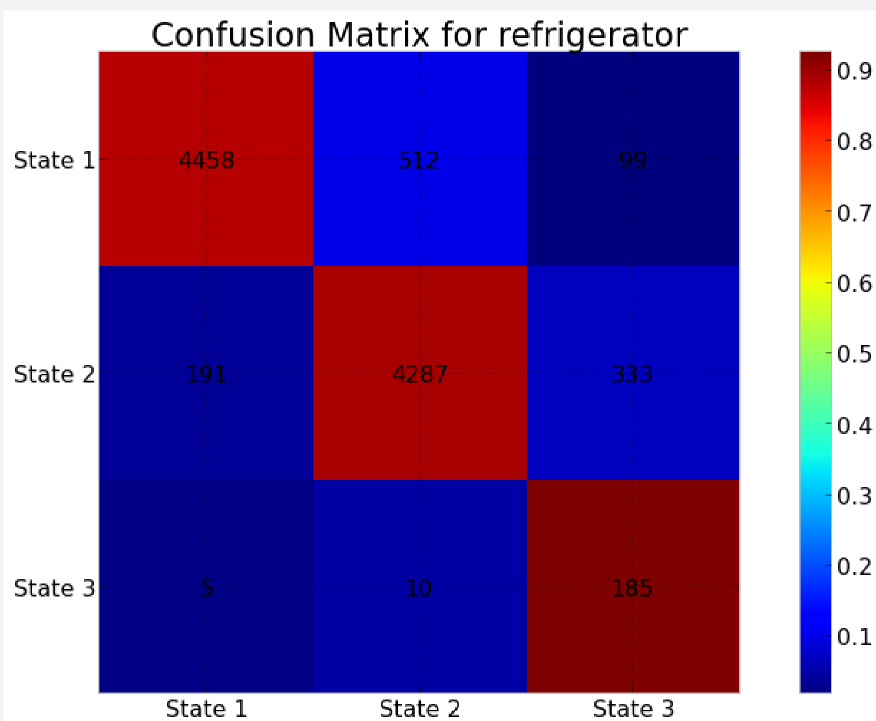
```
appliance_list=['refrigerator','light','microwave']
```

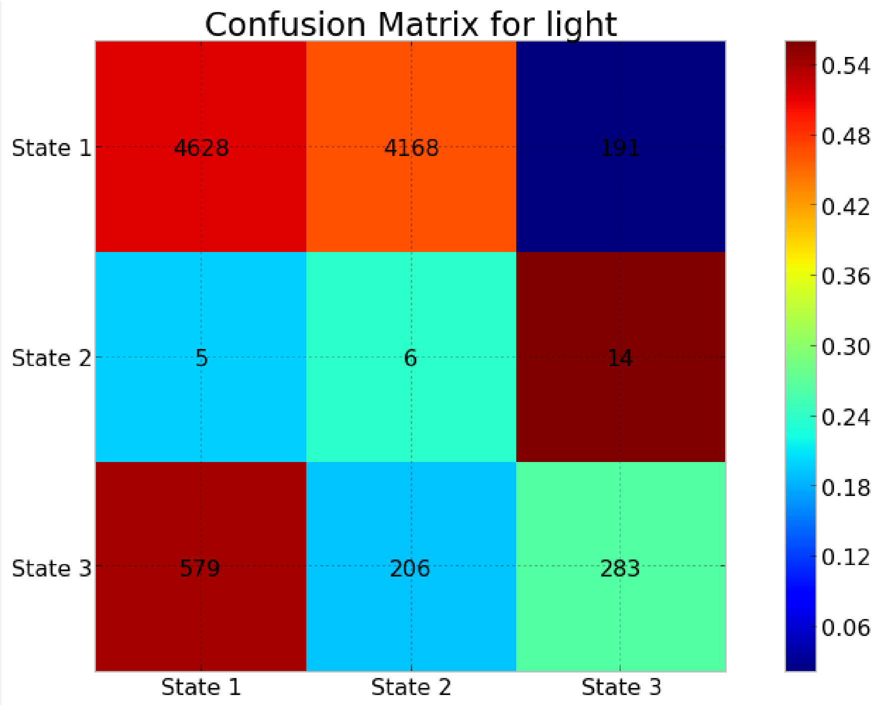
```
[case_2_mains_2_states,case_2_mains_2_power]=decode_CO(length_sequence,centroids,
    appliance_list,states,residual_power)
```

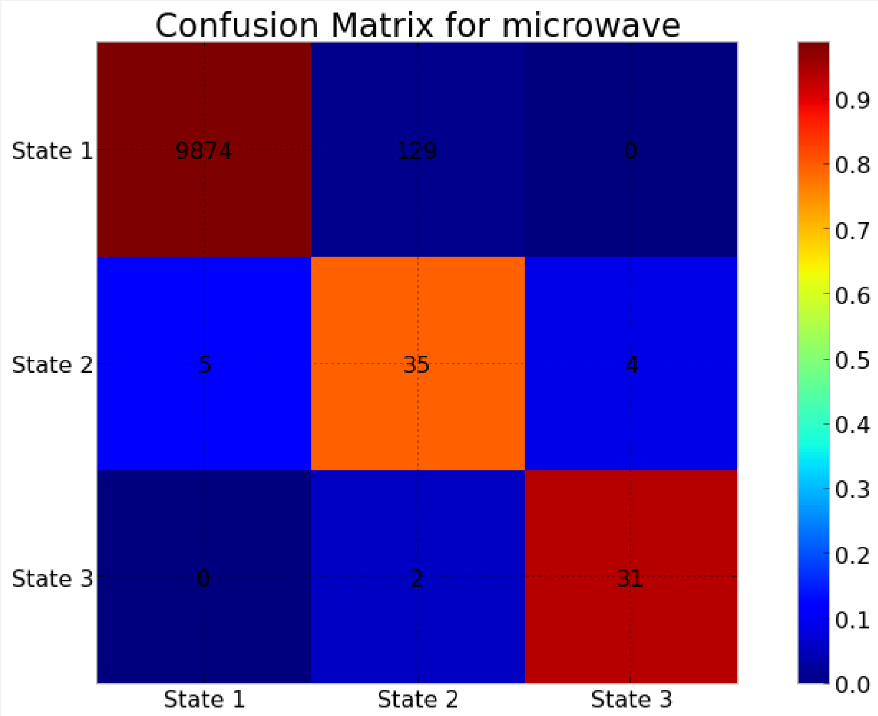
```
case_2_mains_2_states
```

```
{'light': array([1, 1, 1, ..., 1, 1, 1]),
 'microwave': array([0, 0, 0, ..., 0, 0, 0]),
 'refrigerator': array([1, 1, 1, ..., 0, 0, 0])}
```

```
for appliance in appliance_list:
    print_confusion_matrix(appliance,len(centroids[appliance]),true_labels[appliance],
        case_2_mains_2_states[appliance])
```








```
numerator={}
denominator={}
for appliance in appliance_list:
    numerator[appliance]=np.sum(np.abs(case_2_mains_2_power[appliance]-df_appliances_test
        [appliance].values))
```

```
    denominator[appliance]=np.sum(df_appliances_test[appliance].values)
    mne[appliance]=numerator[appliance]*1.0/denominator[appliance]
    re[appliance]=np.std(case_2_mains_2_power[appliance]-df_appliances_test[appliance].
        values)


MNE.append(deepcopy(mne))
RE.append(deepcopy(re))
```

0.1.2 With Calibration

```
states_combination=list(itertools.product(calib_centroids['kitchen'],calib_centroids['
    stove'],calib_centroids['kitchen_2'],calib_centroids['dishwasher'],calib_centroids['
    refrigerator'],calib_centroids['light'],calib_centroids['microwave']))
sum_combination=np.array(np.zeros(len(states_combination)))
for i in range(0,len(states_combination)):
    sum_combination[i]=sum(states_combination[i])
```

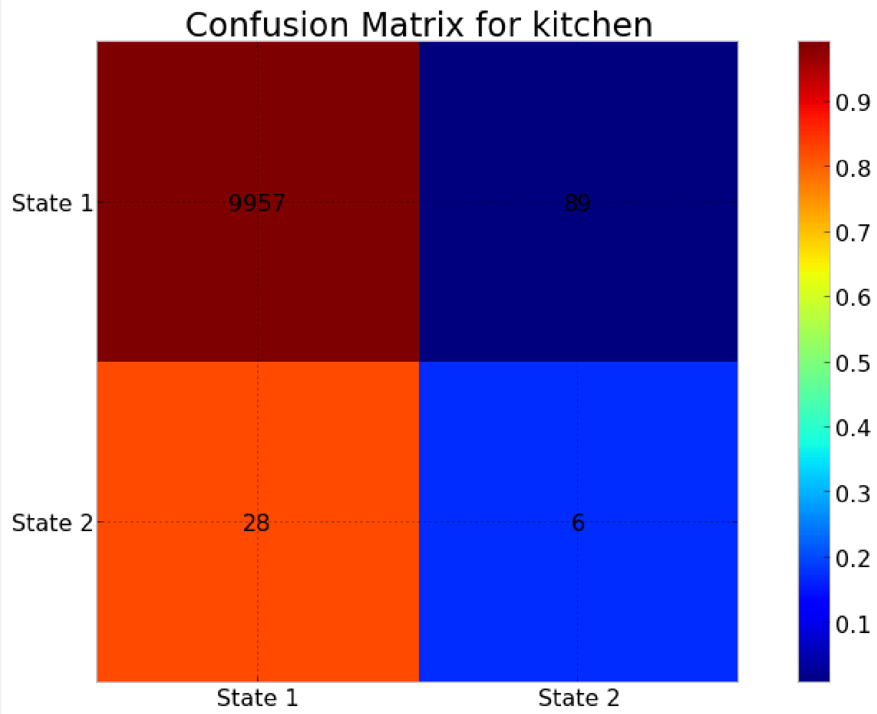
```
length_sequence=len(df_mains_test.Mains_1_Power.values)
states=np.zeros(length_sequence)
residual_power=np.zeros(length_sequence)
t0=time.time()
for i in range(length_sequence):
    [states[i],residual_power[i]]=find_nearest(sum_combination,df_mains_test.
        Mains_2_Power.values[i]+df_mains_test.Mains_1_Power.values[i])
t1=time.time()
print "Time taken for CO Mains 2 :",t1-t0
```

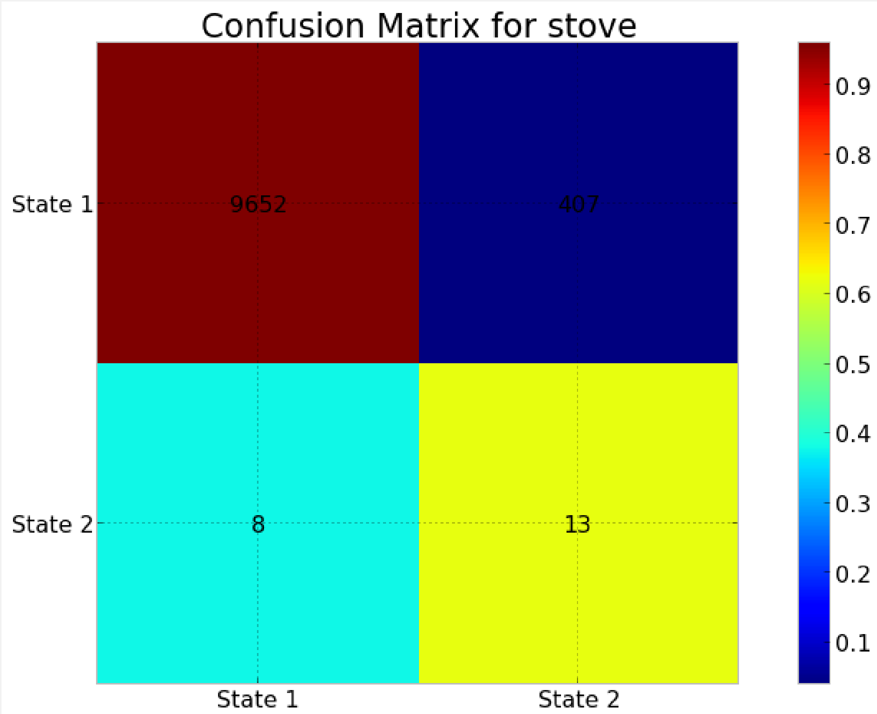
```
Time taken for CO Mains 2 : 0.274182081223
```

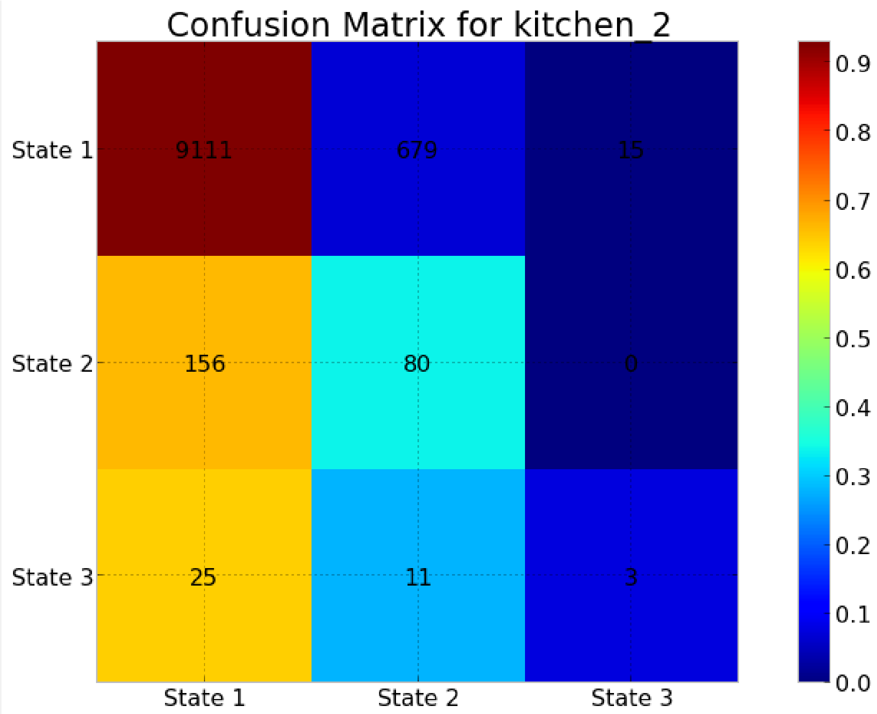
```
appliance_list=['kitchen','stove','kitchen_2','dishwasher','refrigerator','light','
    microwave']
```

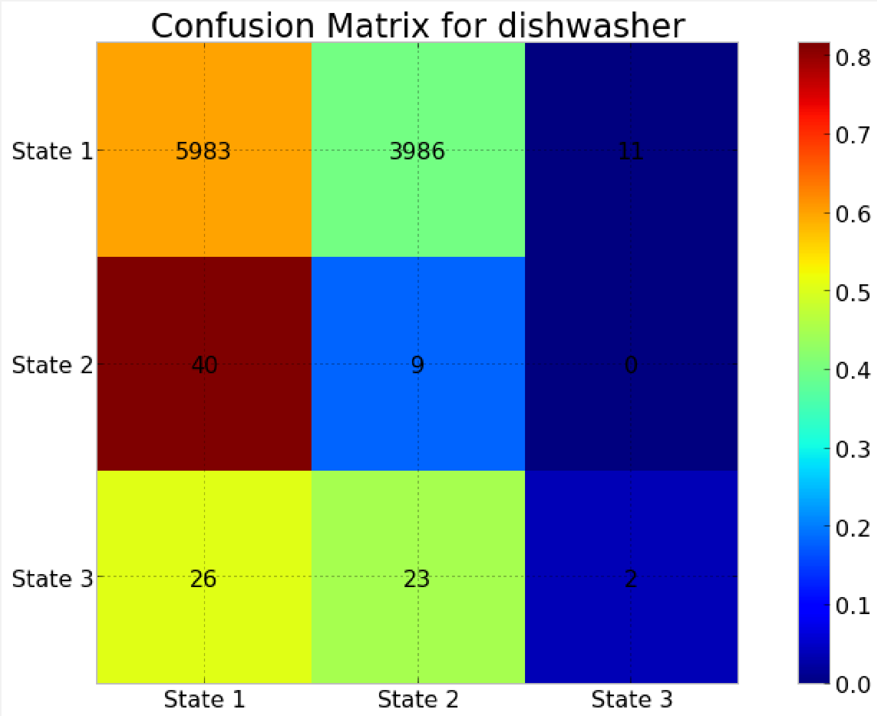
```
[case_3_states,case_3_power]=decode_CO(length_sequence,centroids,appliance_list,states,
    residual_power)
```

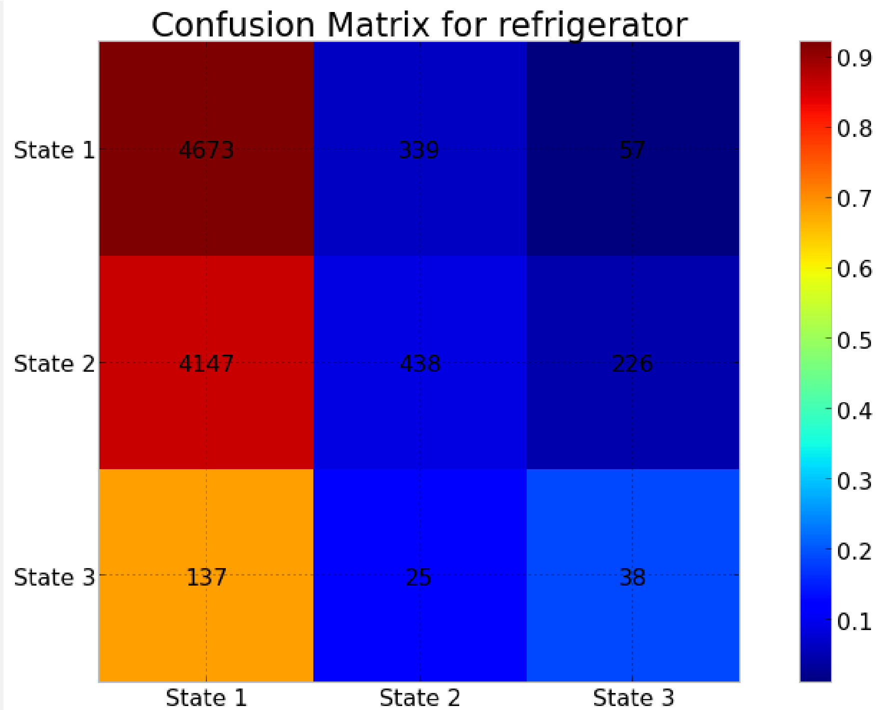
```
for appliance in appliance_list:
    print_confusion_matrix(appliance,len(calib_centroids[appliance]),true_labels[
        appliance],case_3_states[appliance])
```

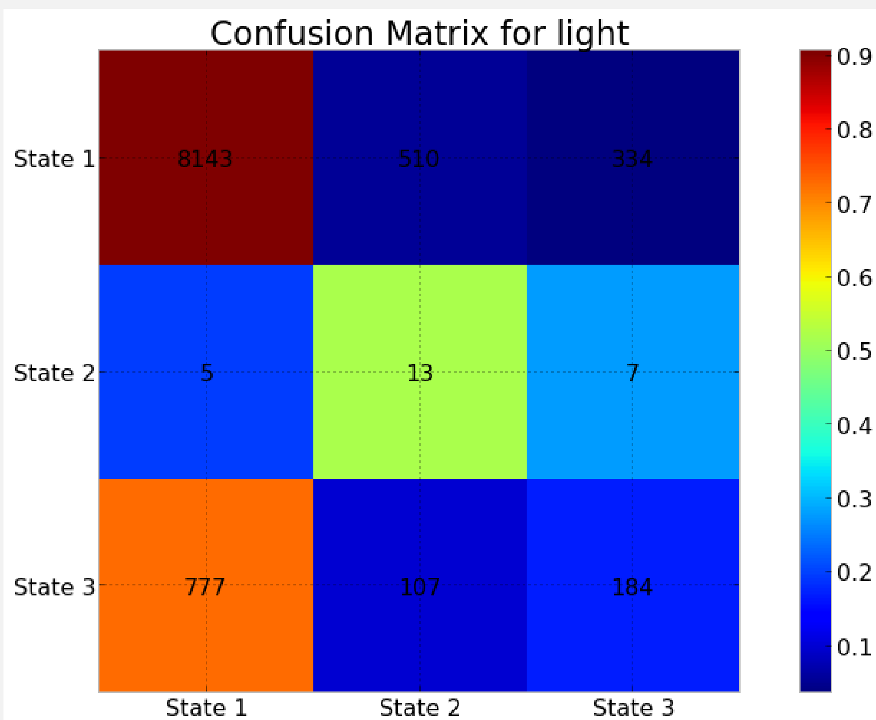

Confusion Matrix for kitchen
State 1
State 2
9957
89
28
6
State 1
State 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

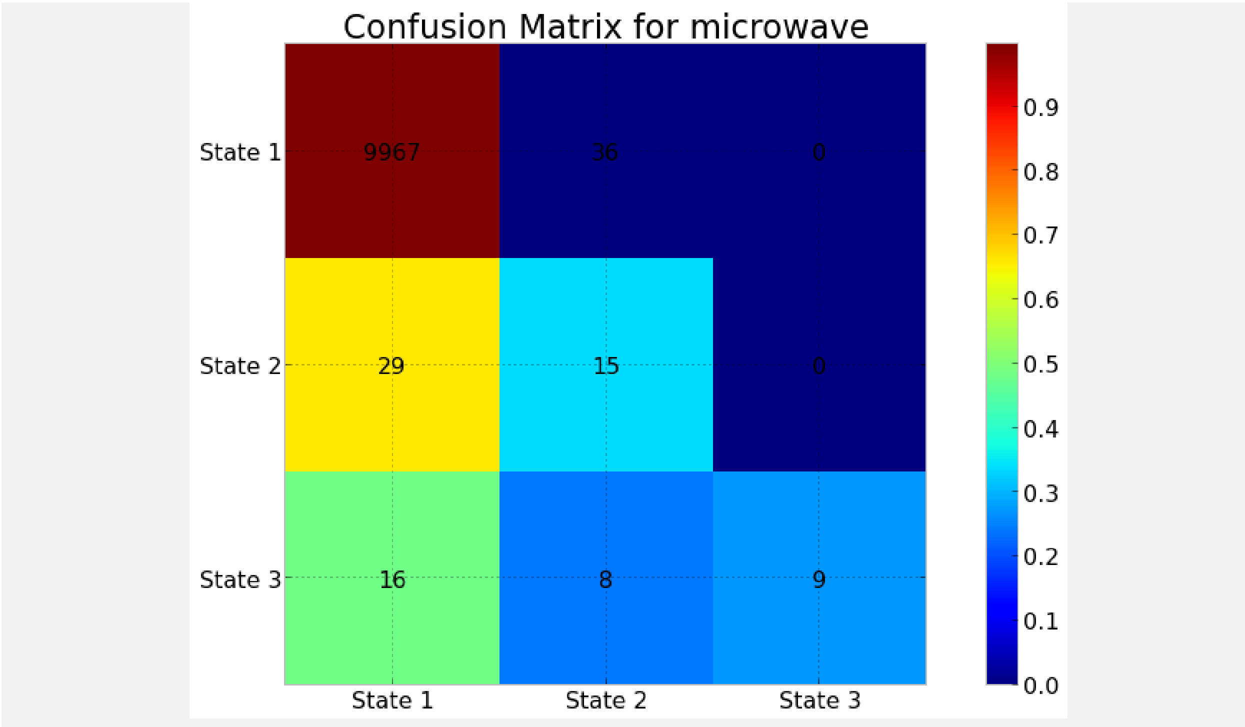

Confusion Matrix for stove
State 1
State 2
9652
407
8
13
State 1
State 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix for kitchen_2
State 1
State 2
State 3
9111
679
15
156
80
0
25
11
3
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Confusion Matrix for dishwasher
State 1
State 2
State 3
5983
3986
11
40
9
0
26
23
2
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Confusion Matrix for refrigerator
State 1
State 2
State 3
4673
339
57
4147
438
226
137
25
38
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix for light
State 1
State 2
State 3
8143
510
334
5
13
7
777
107
184
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1




```
numerator={}
denominator={}
case=2
mne={}
re={}
for appliance in case_3_states:
    numerator[appliance]=np.sum(np.abs(case_3_power[appliance]-df_appliances_test[
        appliance].values))
    denominator[appliance]=np.sum(df_appliances_test[appliance].values)
    mne[appliance]=numerator[appliance]*1.0/denominator[appliance]
    re[appliance]=np.std(case_3_power[appliance]-df_appliances_test[appliance].values)

MNE.append(deepcopy(mne))
RE.append(deepcopy(re))
```

Without Load Division

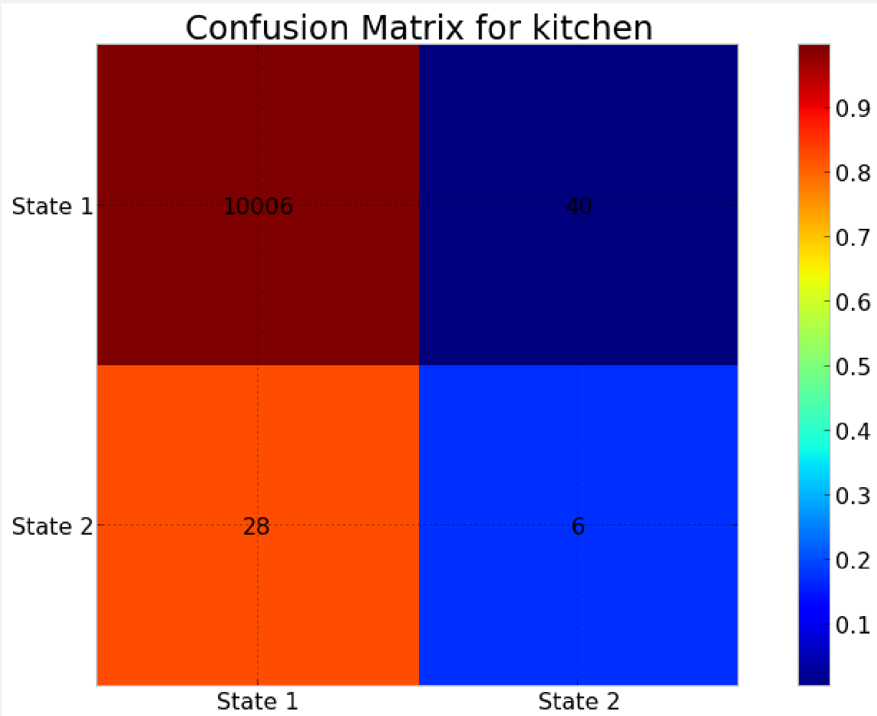
INDiC

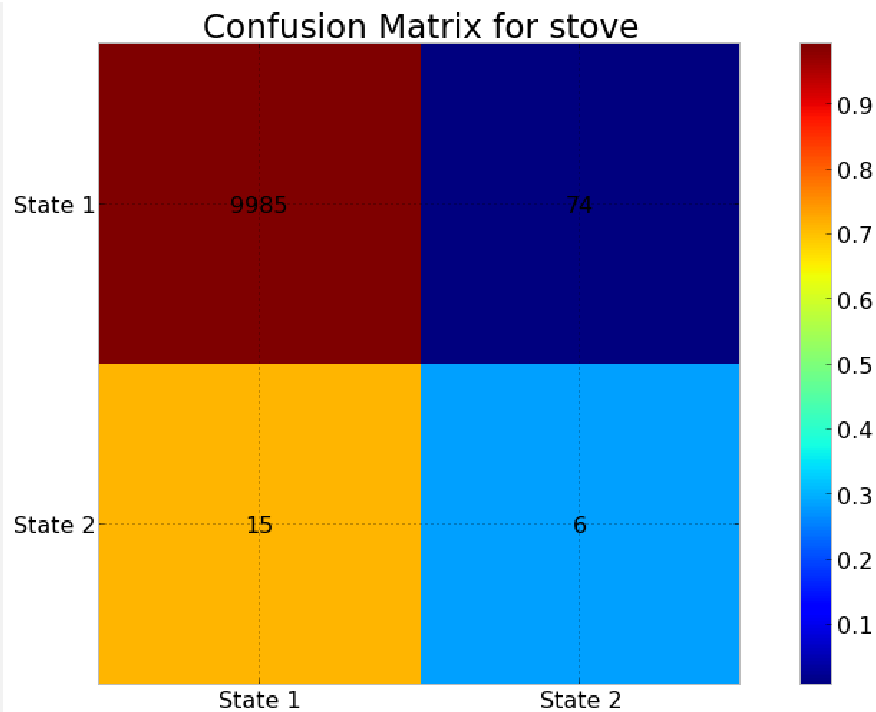
```
states_combination=list(itertools.product(calib_centroids['kitchen'],calib_centroids['
    stove'],\
calib_centroids['kitchen_2'],calib_centroids['dishwasher']))
sum_combination=np.array(np.zeros(len(states_combination)))
for i in range(0,len(states_combination)):
    sum_combination[i]=sum(states_combination[i])
```

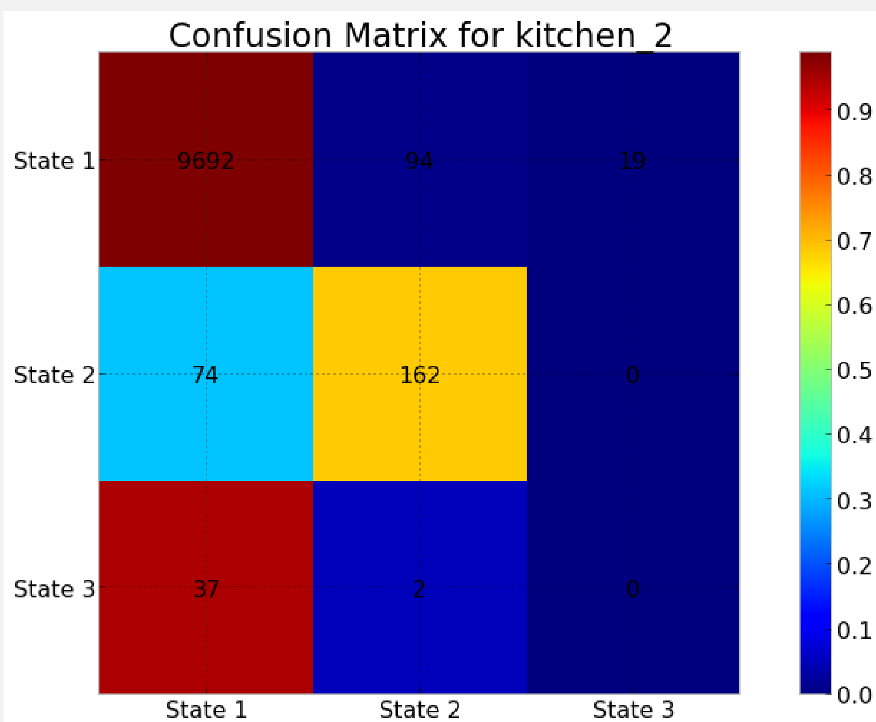
```
length_sequence=len(df_mains_test.Mains_1_Power.values)
states=np.zeros(length_sequence)
residual_power=np.zeros(length_sequence)
t0=time.time()
for i in range(length_sequence):
    [states[i],residual_power[i]]=find_nearest(sum_combination,df_mains_test.
        Mains_1_Power.values[i])
t1=time.time()
print "Time taken for CO Mains :",t1-t0

appliance_list=['kitchen','stove','kitchen_2','dishwasher']


[case_4_mains_1_states,case_4_mains_1_power]=decode_CO(length_sequence,centroids,
    appliance_list,states,residual_power)
```

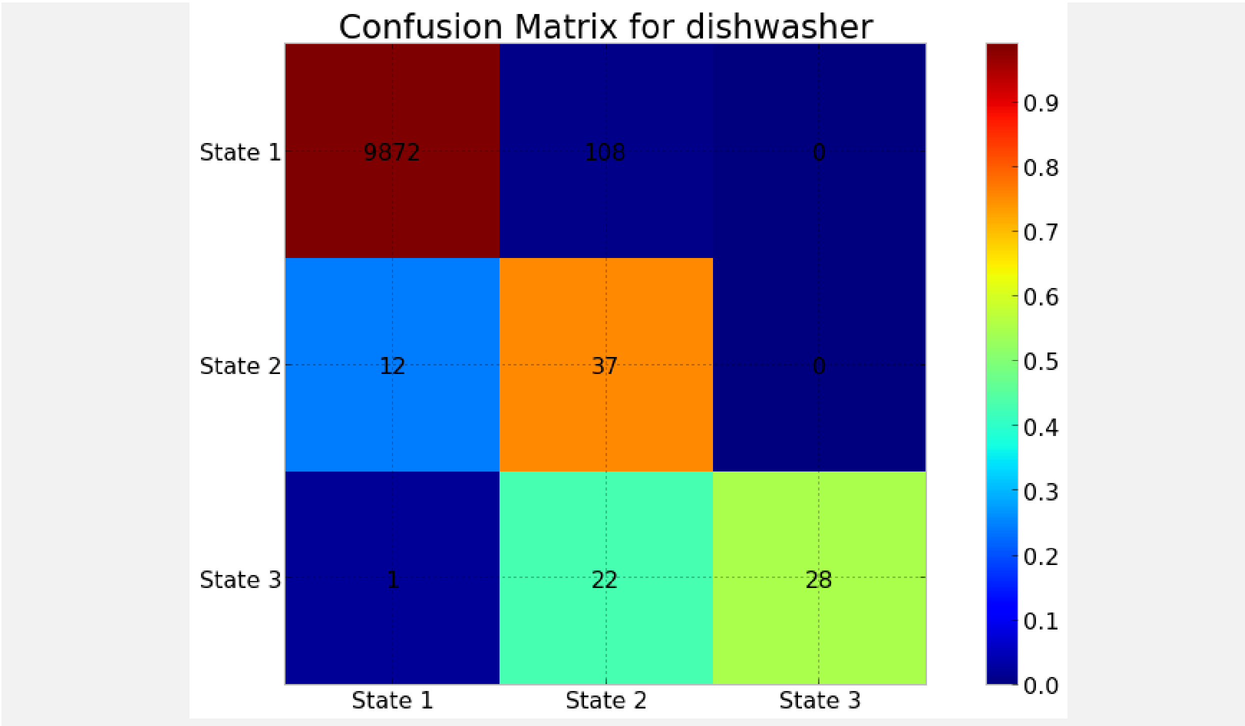
```
Time taken for CO Mains : 0.169928789139
```

```
for appliance in appliance_list:
    print_confusion_matrix(appliance,len(calib_centroids[appliance]),true_labels[
        appliance],case_4_mains_1_states[appliance])
```




Confusion Matrix for stove
State 1
State 2
9985
74
15
6
State 1
State 2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix for kitchen_2
State 1
State 2
State 3
9692
94
19
74
162
0
37
2
0
State 1
State 2
State 3
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0



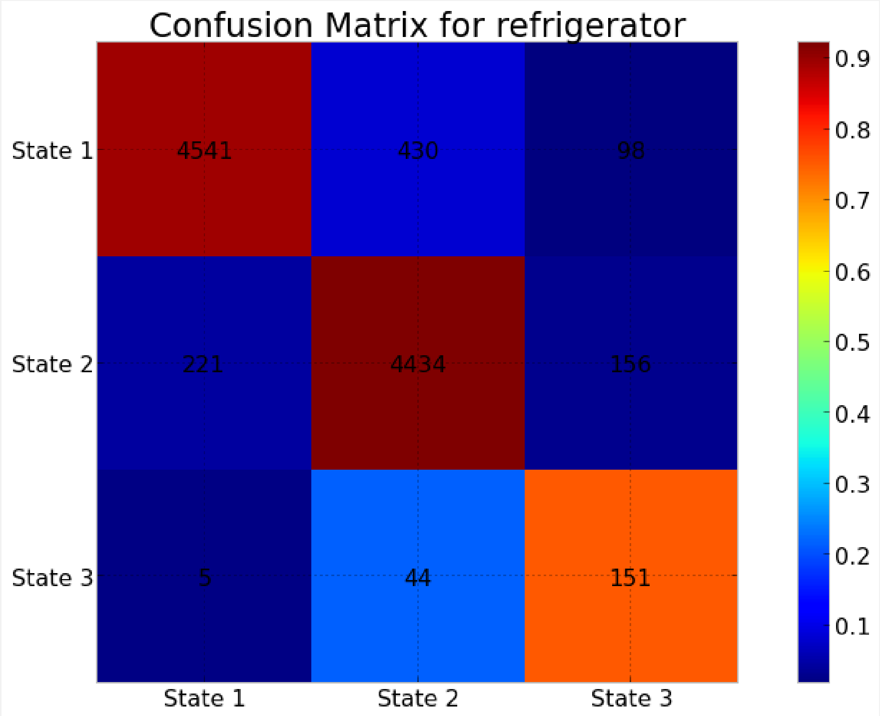

```
numerator={}
denominator={}
case=3
for appliance in appliance_list:
    numerator[appliance]=np.sum(np.abs(case_4_mains_1_power[appliance]-df_appliances_test
        [appliance].values))
    denominator[appliance]=np.sum(df_appliances_test[appliance].values)
    mne[appliance]=numerator[appliance]*1.0/denominator[appliance]
    re[appliance]=np.std(case_4_mains_1_power[appliance]-df_appliances_test[appliance].
        values)
```

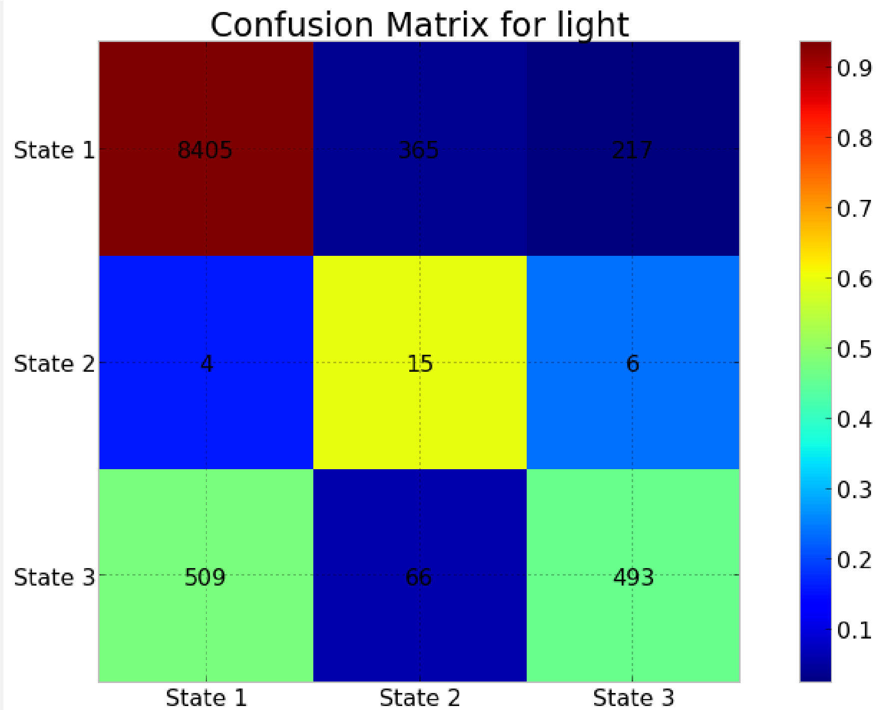
```
states_combination=list(itertools.product(calib_centroids['refrigerator'],
    calib_centroids['light'],\
calib_centroids['microwave']))
sum_combination=np.array(np.zeros(len(states_combination)))
for i in range(0,len(states_combination)):
    sum_combination[i]=sum(states_combination[i])

length_sequence=len(df_mains_test.Mains_2_Power.values)
states=np.zeros(length_sequence)
residual_power=np.zeros(length_sequence)
t0=time.time()
for i in range(length_sequence):
    [states[i],residual_power[i]]=find_nearest(sum_combination,df_mains_test.
        Mains_2_Power.values[i])
t1=time.time()
print "Time taken for CO Mains :",t1-t0
```

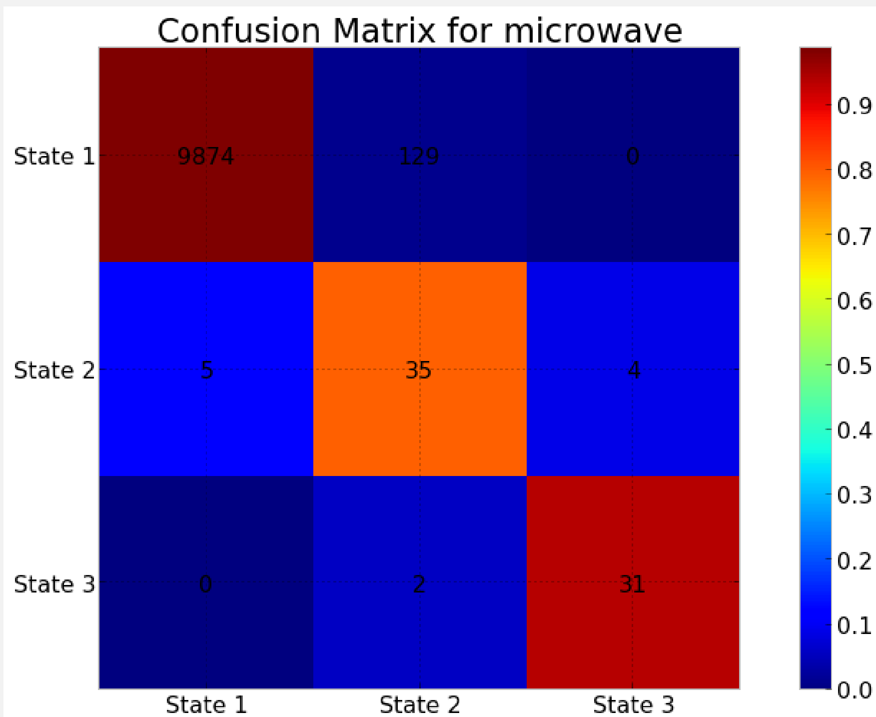
```
appliance_list=['refrigerator','light','microwave']
[case_4_mains_2_states,case_4_mains_2_power]=decode_CO(length_sequence,centroids,
    appliance_list,states,residual_power)
```

```
Time taken for CO Mains : 0.169582128525
```

```
for appliance in appliance_list:
    print_confusion_matrix(appliance,len(calib_centroids[appliance]),true_labels[
        appliance],case_4_mains_2_states[appliance])
```








```
numerator={}
denominator={}
for appliance in case_4_mains_2_states:
```

```
    numerator[appliance]=np.sum(np.abs(case_4_mains_2_power[appliance]-df_appliances_test
        [appliance].values))
    denominator[appliance]=np.sum(df_appliances_test[appliance].values)
    mne[appliance]=numerator[appliance]*1.0/denominator[appliance]
    re[appliance]=np.std(case_4_mains_2_power[appliance]-df_appliances_test[appliance].
        values)

MNE.append(deepcopy(mne))
RE.append(deepcopy(re))
```

With Division

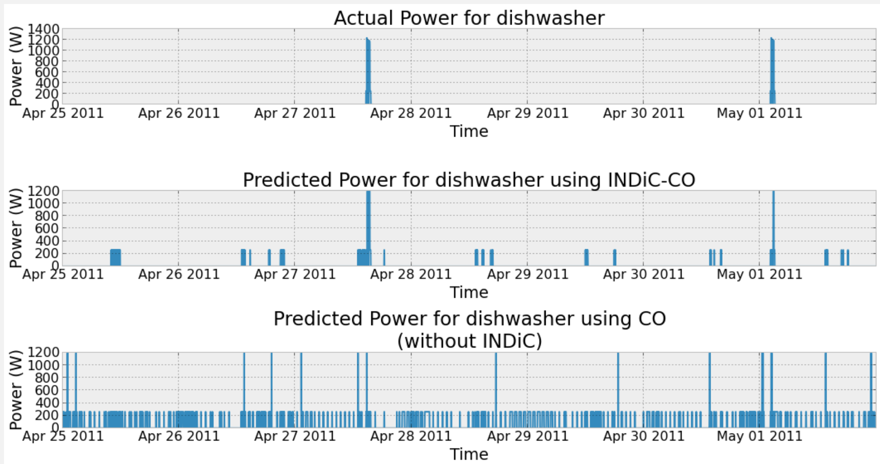
```
from ipy_table import *
results = [

    ['Appliance','Case1','Case1','Case2','Case2','Case3','Case3','Case4','Case4'],
    ['','RE', 'MNE', 'RE', 'MNE','RE', 'MNE','RE', 'MNE']]
for appliance in centroids:
    row=[]
    row.append(appliance)
    for i in range(4):
        row.append(RE[i][appliance])
        row.append(MNE[i][appliance]*100)
    results.append(row)
make_table(results)
set_global_style(float_format='%0.0f')
apply_theme('basic')
```

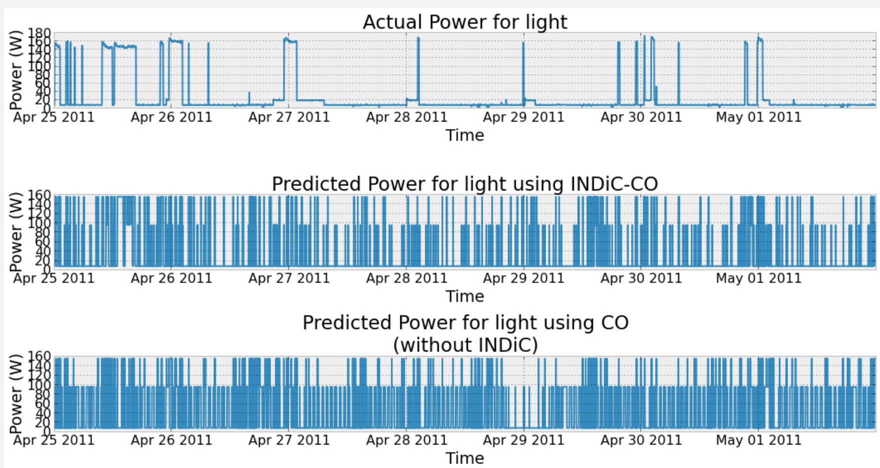
```
<ipy_table.IpyTable at 0x15587390>
```

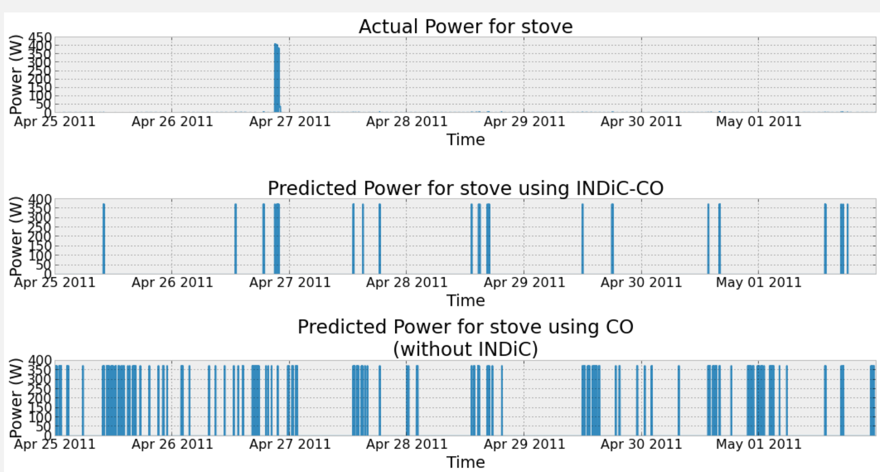
Results We also illustrate the predicted power and ground truth for all appliances and show how the predictions after applying INDiC are much more closer to the ground truth.

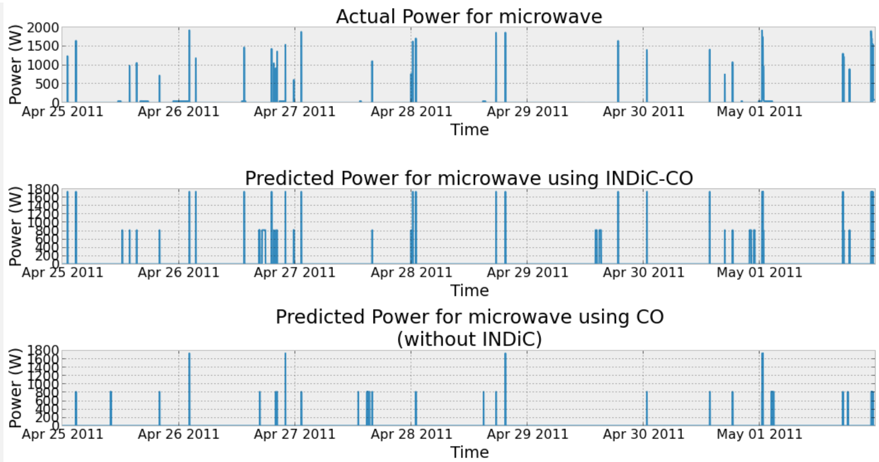
```
for appliance in centroids:
    x=df_appliances_test[appliance].index.to_pydatetime()
    figure()
    plt.subplot(3,1,1)
    plt.title("Actual Power for %s" %appliance)
    plt.xlabel('Time')
    plt.ylabel('Power (W)')
    plt.plot(x,df_appliances_test[appliance].values)
    plt.subplot(3,1,2)
    plt.title("Predicted Power for %s using INDiC-CO" %appliance)
    plt.xlabel('Time')
    plt.ylabel('Power (W)')
    if mapping[appliance]==1:
        plt.plot(x,case_4_mains_1_power[appliance])
    else:
        plt.plot(x,case_4_mains_2_power[appliance])
    plt.subplot(3,1,3)
```

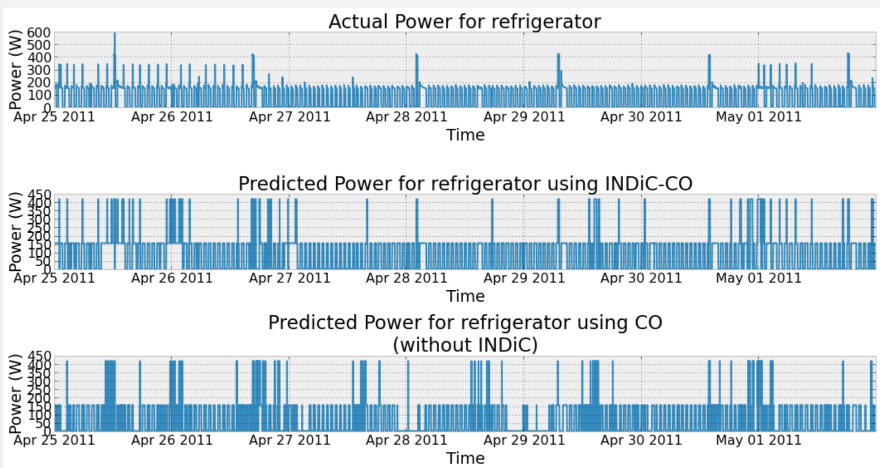
```
    plt.title("Predicted Power for %s using CO \n(without INDiC)" %appliance)
    plt.xlabel('Time')
    plt.ylabel('Power (W)')
    plt.plot(x,case_1_power[appliance])
    plt.tight_layout()
```

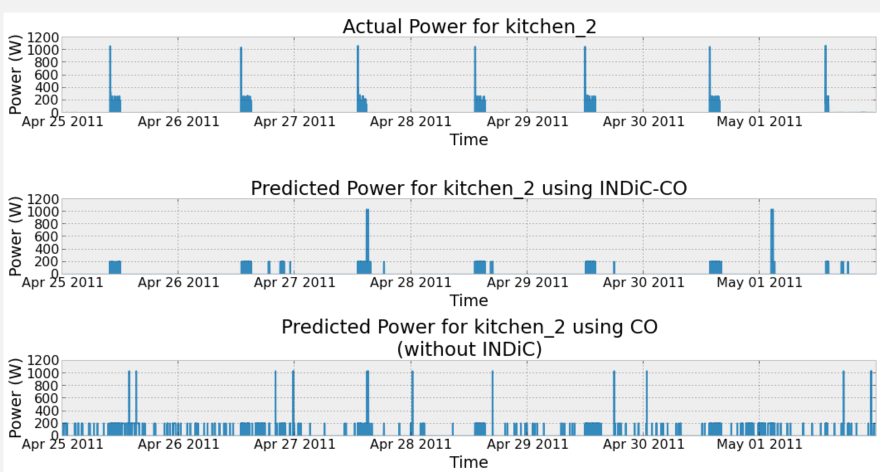


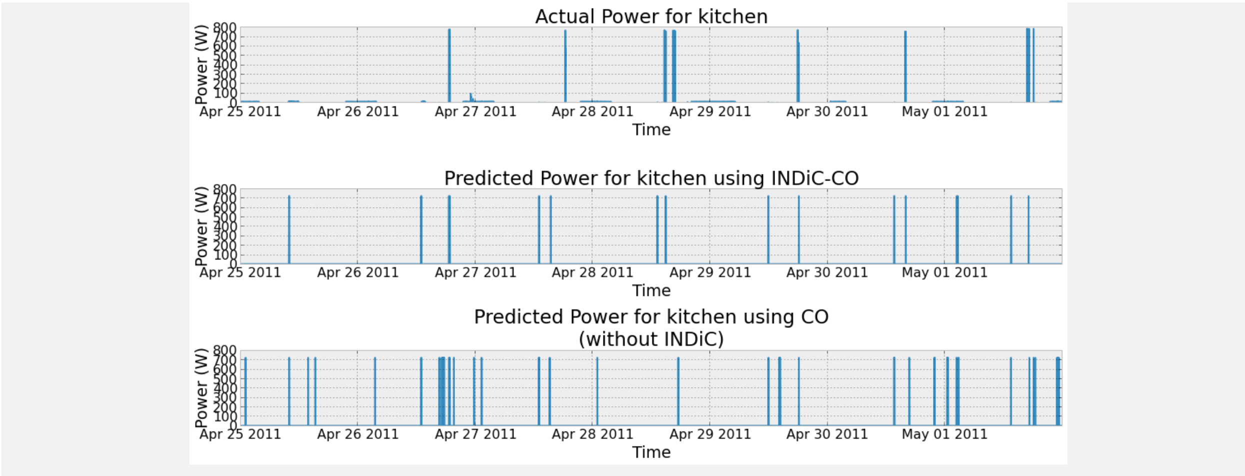





Actual Power for microwave
Power (W)
0
500
1000
1500
2000
Apr 25 2011
Apr 26 2011
Apr 27 2011
Apr 28 2011
Apr 29 2011
Apr 30 2011
May 01 2011
Time
Predicted Power for microwave using INDiC-CO
Power (W)
Time
Predicted Power for microwave using CO
(without INDiC)
Power (W)
Time


Actual Power for refrigerator
Power (W)
0
100
200
300
400
500
600
Apr 25 2011
Apr 26 2011
Apr 27 2011
Apr 28 2011
Apr 29 2011
Apr 30 2011
May 01 2011
Time
Predicted Power for refrigerator using INDiC-CO
Power (W)
Time
Predicted Power for refrigerator using CO
(without INDiC)
Power (W)
Time


Actual Power for kitchen_2
Power (W)
0
200
400
600
800
1000
1200
Apr 25 2011
Apr 26 2011
Apr 27 2011
Apr 28 2011
Apr 29 2011
Apr 30 2011
May 01 2011
Time
Predicted Power for kitchen_2 using INDiC-CO
Power (W)
Time
Predicted Power for kitchen_2 using CO
(without INDiC)
Power (W)
Time



Actual Power for kitchen
Power (W)
0
100
200
300
400
500
600
700
800
Apr 25 2011
Apr 26 2011
Apr 27 2011
Apr 28 2011
Apr 29 2011
Apr 30 2011
May 01 2011
Time
Predicted Power for kitchen using INDiC-CO
Power (W)
Time
Predicted Power for kitchen using CO
(without INDiC)
Power (W)
Time