matching_plot_constelation

April 27, 2017

In [1]:
```
load_ext autoreload
```

In [2]:
```
autoreload 2
```

In [24]:
```
import numpy as np
import pyphi
import random
import makegridslibrary as me
import matplotlib
import matplotlib.pyplot as plt
import itertools
from pypci import pci
import pickle
from tsne import tsne
import bitarray as bit
from IPython.core.display import display, HTML

def i_to_bitlist(o):
    if not o:
        return [0]
    shifts = list( range( int( np.ceil(np.log2(o)) + 1 ) ) )
    shifts.reverse() # little endian
    return [(o >> shift) & 1 for shift in shifts]
```

In [4]:
```
# Load results from matching_LB.py

temp = 0
# temp = .2

# Old measure

# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrando
results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrandom_r

print('EMD results')

with open(results_filename, 'rb') as f:
    results = pickle.load(f)
```

```
big_results_bar = results[2]
big_results_shuffled = results[3]

# for network in big_results_bar:

network = 'Grid'

print(network)
print('\tT : ', temp)
print('\tfor the bar')
for input_stim in big_results_bar[network][temp]:
    print('\t', input_stim)
    this_result = big_results_bar[network][temp][input_stim]
    bar_emd_big_phi = this_result[0]
    bar_emd_concepts = this_result[1]

print('\tfor the shuffled')
for input_stim in big_results_shuffled[network][temp]:
    print('\t', input_stim)
    this_result = big_results_shuffled[network][temp][input_stim]
    shu_emd_big_phi = this_result[0]
    shu_emd_concepts = this_result[1]
print('done.')

# New measure

# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrando
# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrando
# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrando
results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrandom_E

print('\nNew cut + entropy distance results')

with open(results_filename, 'rb') as f:
    results = pickle.load(f)

big_results_bar_new = results[2]
big_results_shuffled_new = results[3]

# for network in big_results_bar_new:

print(network)
print('\tT : ', temp)
print('\tfor the bar')
for input_stim in big_results_bar_new[network][temp]:
    print('\t', input_stim)
    this_result = big_results_bar_new[network][temp][input_stim]
```

```
            bar_new_big_phi = this_result[0]
            bar_new_concepts = this_result[1]

        print('\tfor the shuffled')
        for input_stim in big_results_shuffled_new[network][temp]:
            print('\t', input_stim)
            this_result = big_results_shuffled_new[network][temp][input_stim]
            shu_new_big_phi = this_result[0]
            shu_new_concepts = this_result[1]
        print('done.')


EMD results
Grid
        T :  0
        for the bar
         (0, 0, 0, 1, 1)
        for the shuffled
         (0, 0, 1, 0, 1)
done.

New cut + entropy distance results
Grid
        T :  0
        for the bar
         (0, 0, 0, 1, 1)
        for the shuffled
         (0, 0, 1, 0, 1)
done.


In [196]: # preprocess: build masks, labels, etc

          def copy_and_mask(X):
              Y = X.copy()
              Y_invalid = Y == -1
              Y[Y_invalid] = 0
              Y_big_invalid = np.any(Y_invalid, axis=0)
              return(Y, Y_invalid, Y_big_invalid)

          # Use EMD
          [bar_concepts, bar_big_phi, shu_concepts, shu_big_phi] = [bar_emd_concept

          # Use new cuts and distances
          # [bar_concepts, bar_big_phi, shu_concepts, shu_big_phi] = [bar_new_conce

          (bar, bar_invalid, bar_big_phi_invalid) = copy_and_mask(bar_concepts)
          (shu, shu_invalid, shu_big_phi_invalid) = copy_and_mask(shu_concepts)
```

```
N = int(np.ceil(np.log2(bar.shape[0])))

# Generate mechanisms labels
mechanisms_orders = np.array([sum(i_to_bitlist(m)) for m in range(2**N)])
# null mechanism is impossible
# mechanisms_orders = mechanisms_orders[1:]
# created a sorted index
mo_sorted_idx = np.argsort(mechanisms_orders)
sorted_mechanisms_orders = mechanisms_orders[mo_sorted_idx]

# Generate states labels
gs = np.array(list(itertools.product((0, 1), repeat=N)))
# print(grid_states)
# print(np.abs(np.diff(grid_states, axis=1)))
def neighbor(i,N,d):
    j = i + d
    # cycle right
    while j > N-1:
        j = j-N
    # cycle left
    while j < 0:
        j = j+N
    return j

def state_label_cont(s):
    if not np.sum(s) or np.sum(s) == len(s):
        label = 'full'
    else:
        second_order = [s[i] + s[neighbor(i,N,1)] for i in range(N)]
        n1 = np.sum(1*[s2 == 1 for s2 in second_order])
        n2 = np.sum(1*[s2 == 2 for s2 in second_order])
#         print(n2, s)
        if n2 == 0:
            label = 'shuffled'
        elif n2 == 1:
    #         if n1 > 0
            label = 'mixed'
        elif n2 == 2:
            label = 'bigbar2'
        elif n2 == 3:
            label = 'bigbar3'
        else:
            raise ValueError('Invalid state')
    return label

def state_label_acti(s):
#     s = [s[i] + s[neighbor(i,N,1)] for i in range(N)]
```

```
        #     s = [s[neighbor(i,N,-1) + s[i] + s[neighbor(i,N,1)] for i in range
        #     s = [1*(si > 1) for si in s]
        #     s = [1*(si != 1) for si in s]

            s = [1*(s[neighbor(i,N,-1)] == s[i]) +  1*(s[neighbor(i,N,1)] == s[i]

            return np.sum(s)

        states_labels = [state_label_cont(s) for s in gs]
        # states_labels = [state_label_acti(s) for s in gs]

        sl_sorted_idx = np.argsort(states_labels)
        sorted_states_labels = [states_labels[sl] for sl in sl_sorted_idx]
```

In [197]:
```
s=gs[6]
ss = [1*(s[neighbor(i,N,-1)] == s[i]) +  1*(s[neighbor(i,N,1)] == s[i]) f
# print(s)
# print(ss)
print(np.sum(ss))

plt.figure(figsize=(10,10))

print(states_labels)
plt.plot(states_labels, list(range(2**N)))
plt.yticks(range(2**N), gs)
plt.show()

plt.figure(figsize=(10,4))
plt.hist(states_labels)
plt.show()
```

```
6
['full', 'shuffled', 'shuffled', 'mixed', 'shuffled', 'shuffled', 'mixed', 'bigbar2
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-197-db10d8ffea54> in <module>()
      9
     10 print(states_labels)
---> 11 plt.plot(states_labels, list(range(2**N)))
     12 plt.yticks(range(2**N), gs)
```

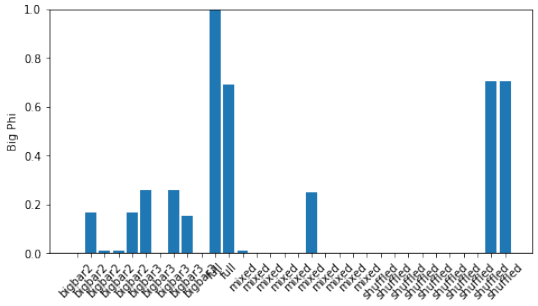
```
     13 plt.show()


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
   3316                         mplDeprecation)
   3317     try:
-> 3318         ret = ax.plot(*args, **kwargs)
   3319     finally:
   3320         ax._hold = washold


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
   1890                     warnings.warn(msg % (label_namer, func.__name__),
   1891                                   RuntimeWarning, stacklevel=2)
-> 1892             return func(ax, *args, **kwargs)
   1893         pre_doc = inner.__doc__
   1894         if pre_doc is None:


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
   1405
   1406         for line in self._get_lines(*args, **kwargs):
-> 1407             self.add_line(line)
   1408             lines.append(line)
   1409


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
   1785             line.set_clip_path(self.patch)
   1786
-> 1787         self._update_line_limits(line)
   1788         if not line.get_label():
   1789             line.set_label('_line%d' % len(self.lines))


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
   1807         Figures out the data limit of the given line, updating self.dat
   1808         """
-> 1809         path = line.get_path()
   1810         if path.vertices.size == 0:
   1811             return


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
    987         """
    988         if self._invalidy or self._invalidx:
--> 989             self.recache()
    990         return self._path
```

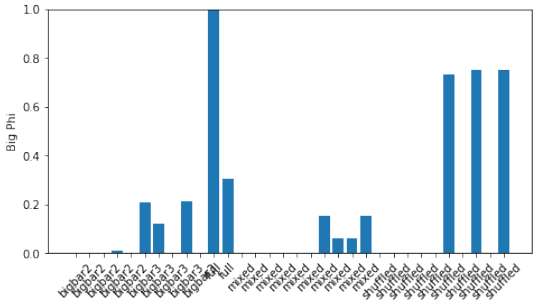
```
    991


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/matplo
    674                 x = ma.asarray(xconv, np.float_).filled(np.nan)
    675             else:
--> 676                 x = np.asarray(xconv, np.float_)
    677             x = x.ravel()
    678         else:


    /home/leonardo/anaconda3/envs/nsdm-pyphi/lib/python3.4/site-packages/numpy/
    529
    530     """
--> 531     return array(a, dtype, copy=False, order=order)
    532
    533


    ValueError: could not convert string to float: 'full'
```

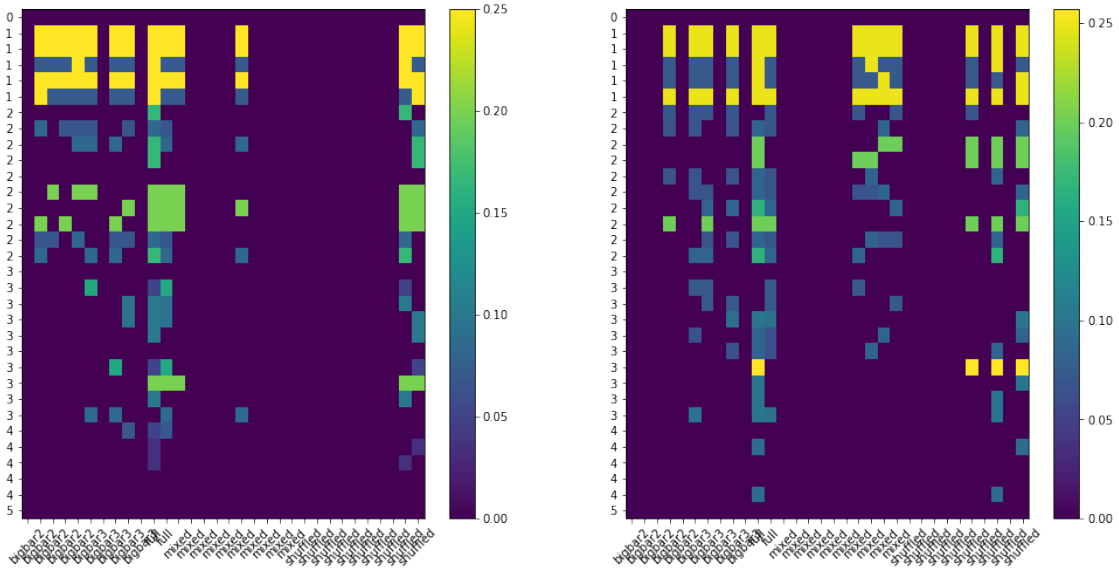
```
In [198]: # [np.sum(sorted_mechanisms_orders==o) for o in np.unique(mechanisms_orde
          # print(sl_sorted_idx)
          mechanisms_orders
          # print(mechanisms_orders)
          # print(sorted_mechanisms_orders)
          #print(sorted_states_labels)

Out[198]: array([0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3,
                 4, 2, 3, 3, 4, 3, 4, 4, 5])

In [200]: # Big Phi

          plt.figure(figsize=(17,4))

          plt.subplot(1,2,1)
          plt.ylim([0, 1])
          Y = bar_big_phi.copy()
          Y[bar_big_phi_invalid] = 0
          plt.bar(range(len(Y)), Y[sl_sorted_idx])
          plt.xticks(range(len(Y)), sorted_states_labels, rotation=45)
          plt.ylabel('Big Phi')

          plt.subplot(1,2,2)
          plt.ylim([0, 1])
          # Y = bar_big_phi_new.copy()
          # Y[bar_big_phi_new_invalid] = 0
```

```
Y = shu_big_phi.copy()
Y[shu_big_phi_invalid] = 0
plt.bar(range(len(Y)), Y[sl_sorted_idx])
plt.xticks(range(len(Y)), sorted_states_labels, rotation=45)

plt.ylabel('Big Phi')

plt.show()



# All concepts

def norm_min_max(Y):
    Z = (Y-min(Y.flatten()))/(max(Y.flatten())-min(Y.flatten()))
    return Z

plt.figure(figsize=(17,8))

plt.subplot(1,2,1)
# Y = X[sl_sorted_idx, :]

# Why numpy, why...
# Y = bar[mo_sorted_idx, sl_sorted_idx].copy()
bar_sorted = bar[mo_sorted_idx, :].copy()[:, sl_sorted_idx]

# bar_sorted[bar_invalid[mo_sorted_idx, :][:, sl_sorted_idx]] = 0
# bar_sorted = np.ma.masked_where(X_invalid[sl_sorted_idx, :], X[sl_sorte

# bar_sorted = norm_min_max(Y)

plt.imshow(bar_sorted, aspect='auto')
plt.yticks(range(len(mo_sorted_idx)), sorted_mechanisms_orders)
plt.xticks(range(len(sl_sorted_idx)), sorted_states_labels, rotation=45)
# plt.xticks(range(len(sl_sorted_idx)), range(len(sl_sorted_idx)), rotati

plt.colorbar()

plt.subplot(1,2,2)
# Y = X_new[sl_sorted_idx, :]

# shu_sorted = shu[mo_sorted_idx, sl_sorted_idx].copy()
shu_sorted = shu[mo_sorted_idx, :].copy()[:, sl_sorted_idx]

# shu_sorted[shu_invalid[mo_sorted_idx, :][:, sl_sorted_idx]] = 0
# shu_sorted = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :], X_new

# shu_sorted = norm_min_max(Y_shu)
```

```
plt.imshow(shu_sorted, aspect='auto')
plt.yticks(range(len(mo_sorted_idx)), sorted_mechanisms_orders)
plt.xticks(range(len(sl_sorted_idx)), sorted_states_labels, rotation=45)
# plt.xticks(range(len(sl_sorted_idx)), range(len(sl_sorted_idx)), rotati
plt.colorbar()



# plt.figure(figsize=(17,8))

# Z = Y-Y_new
# # Z = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :] | ((Y_new[sl_s
# # Z = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :], Z)
# Z = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :] | (Y[sl_sorted_i

# plt.imshow(Z, aspect='auto',cmap='bwr')
# plt.yticks(range(len(labels)), labels[sl_sorted_idx])
# plt.clim([-1, 1])
# plt.colorbar()

plt.show()
```





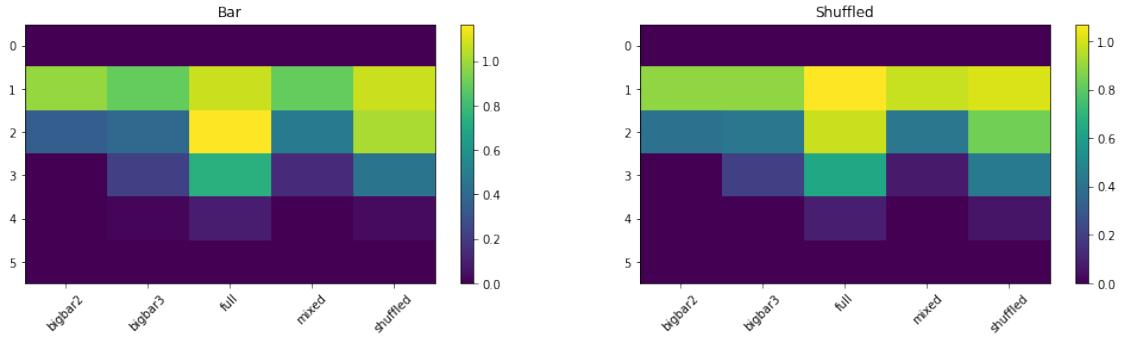


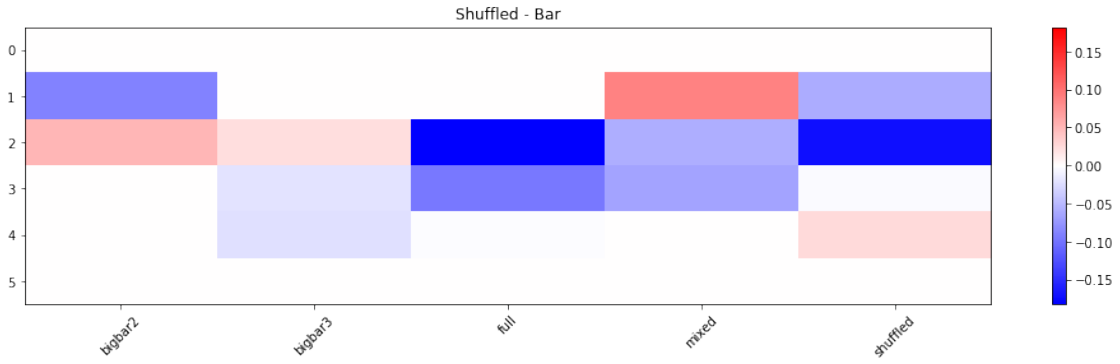
In [201]:
```
# number_only = 1
# number_only = 0

for number_only in range(2):

    display(HTML("<center><h1 style='font-size: %dpx;height: 100;'>%s</h1
            ((40, '# of concepts') if number_only else (50, '&Sigma; &phi

    # unique mechanisms
    unique_mechanism_orders = np.unique(sorted(sorted_mechanisms_orders))

    # unique and states
    unique_states_labels = np.unique(sorted(sorted_states_labels))
    nstates = len(unique_states_labels)

    # BAR

    # find all the concepts for the bar
    if number_only:
        bar_sorted_n = 1. * (bar_sorted > 0)
    else:
        bar_sorted_n = bar_sorted

    # sum all the mechanisms for a given order, per state
    bar_sorted_cpmo = np.array([np.sum(bar_sorted_n[sorted_mechanisms_orc
                                for o in unique_mechanism_orders])

    # remove invalid states
```

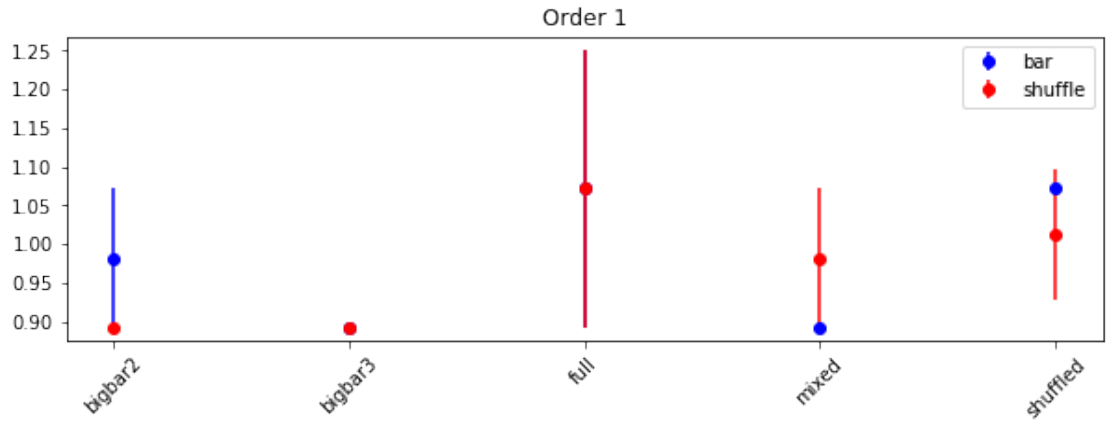
```
invalid_bar_states = np.all(bar_invalid[mo_sorted_idx, :][:, sl_sorte
sorted_bar_states_labels_noinv = np.delete(sorted_states_labels, np.w
bar_sorted_cpmo_noinv = np.delete(bar_sorted_cpmo, np.where(invalid_b

# take the mean and std accross valid states
bar_sorted_cpmosc = np.array([np.mean(bar_sorted_cpmo_noinv[:, [ss ==
                        for s in unique_states_labels]).T
bar_sorted_cpmosc_std = np.array([np.std(bar_sorted_cpmo_noinv[:, [ss
                        for s in unique_states_labels]).T

# SHUFFLED

# find all the concepts for the shuffled
if number_only:
    shu_sorted_n = 1. * (shu_sorted > 0)
else:
    shu_sorted_n = shu_sorted

# sum all the mechanisms for a given order, per state
shu_sorted_cpmo = np.array([np.sum(shu_sorted_n[sorted_mechanisms_orc
                            for o in np.unique(sorted(mechanisms_orde

# remove invalid states
invalid_shu_states = np.all(shu_invalid[mo_sorted_idx, :][:, sl_sorte
sorted_shu_states_labels_noinv = np.delete(sorted_states_labels, np.w
shu_sorted_cpmo_noinv = np.delete(shu_sorted_cpmo, np.where(invalid_s

# take the mean and std accross valid states
shu_sorted_cpmosc = np.array([np.mean(shu_sorted_cpmo_noinv[:, [ss ==
                        for s in unique_states_labels]).T
shu_sorted_cpmosc_std = np.array([np.std(shu_sorted_cpmo_noinv[:, [ss
                        for s in unique_states_labels]).T

plt.figure(figsize=(17,4))

plt.subplot(1,2,1)
plt.imshow(bar_sorted_cpmosc, aspect='auto')
plt.xticks(range(len(unique_states_labels)), unique_states_labels, ro
plt.colorbar()
plt.title('Bar')

plt.subplot(1,2,2)
plt.imshow(shu_sorted_cpmosc, aspect='auto')
plt.xticks(range(len(unique_states_labels)), unique_states_labels, ro
plt.colorbar()
plt.title('Shuffled')

plt.show()
```

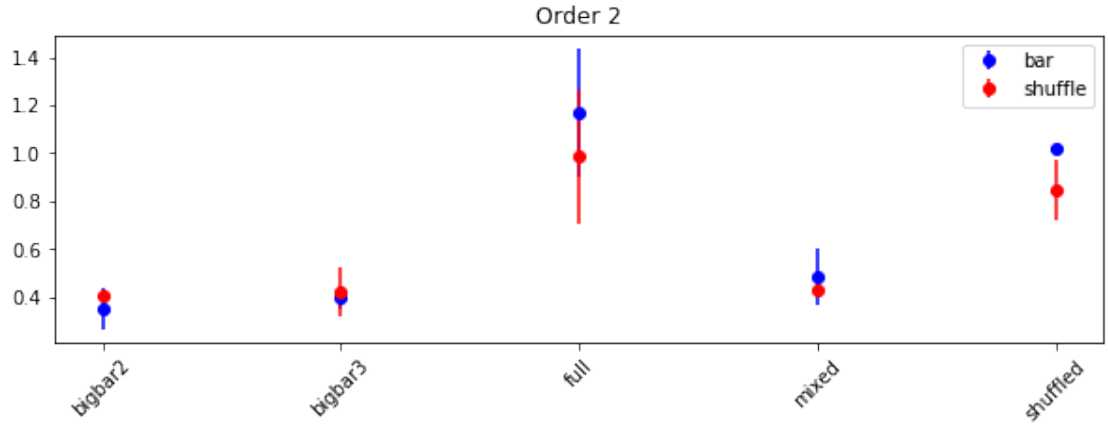
```
plt.figure(figsize=(17,4))
the_diff = shu_sorted_cpmosc-bar_sorted_cpmosc
the_diff_max = max(abs(the_diff.flatten()))
plt.imshow(the_diff, aspect='auto', cmap='bwr', vmin=-the_diff_max, v
plt.colorbar()
plt.xticks(range(len(unique_states_labels)), unique_states_labels, ro
plt.title('Shuffled - Bar')
plt.show()


for o in range(1, len(unique_mechanism_orders)):

    plt.figure(figsize=(10,3))

    hb = plt.errorbar(range(nstates), bar_sorted_cpmosc[o, :], yerr=b
    hs = plt.errorbar(range(nstates), shu_sorted_cpmosc[o, :], yerr=s
    plt.legend([hb, hs], ['bar', 'shuffle'])
    plt.xticks(range(len(unique_states_labels)), unique_states_labels
    plt.title('Order %d' % o)

    plt.show()
```

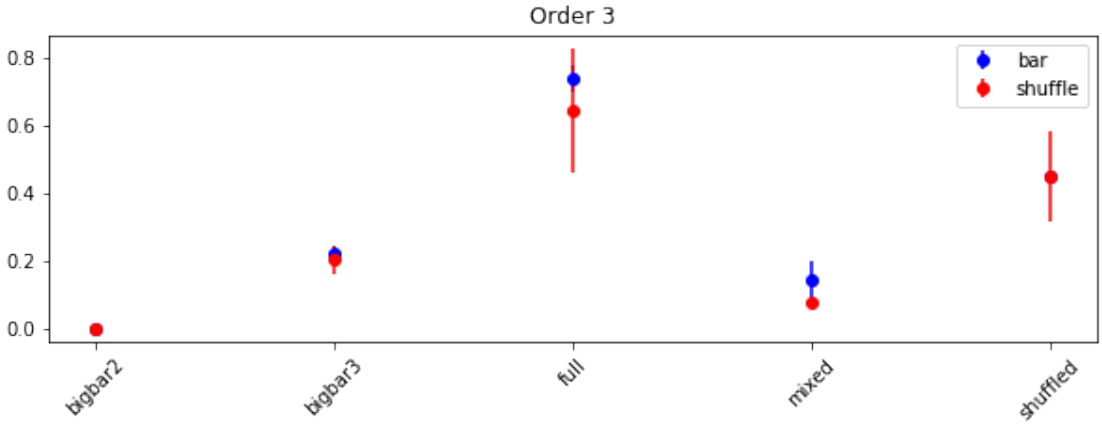
```
<IPython.core.display.HTML object>
```

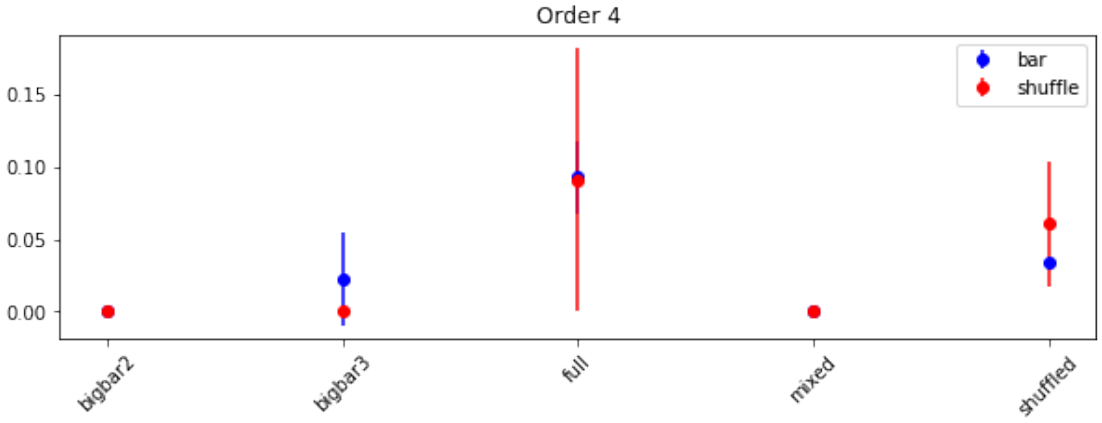


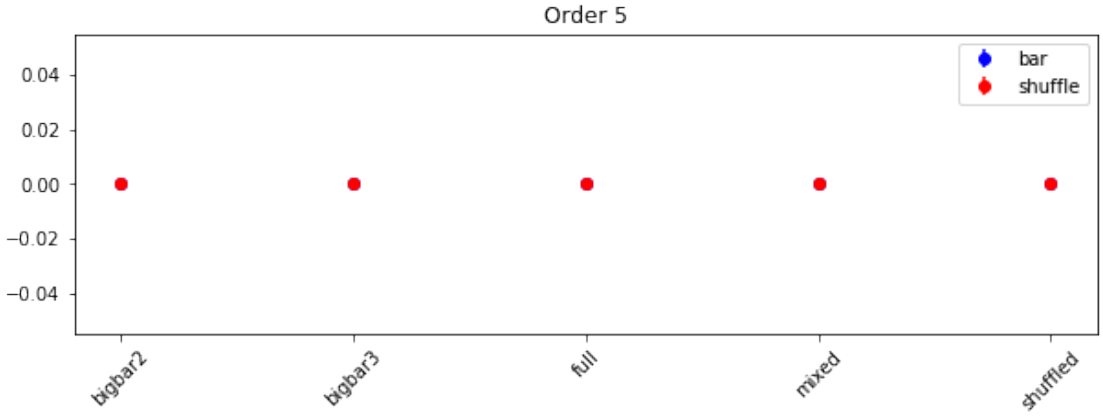

Shuffled - Bar
0
1
2
3
4
5
bigbar2
bigbar3
full
mixed
shuffled
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15


Order 1
1.25
1.20
1.15
1.10
1.05
1.00
0.95
0.90
bar
shuffle
bigbar2
bigbar3
full
mixed
shuffled

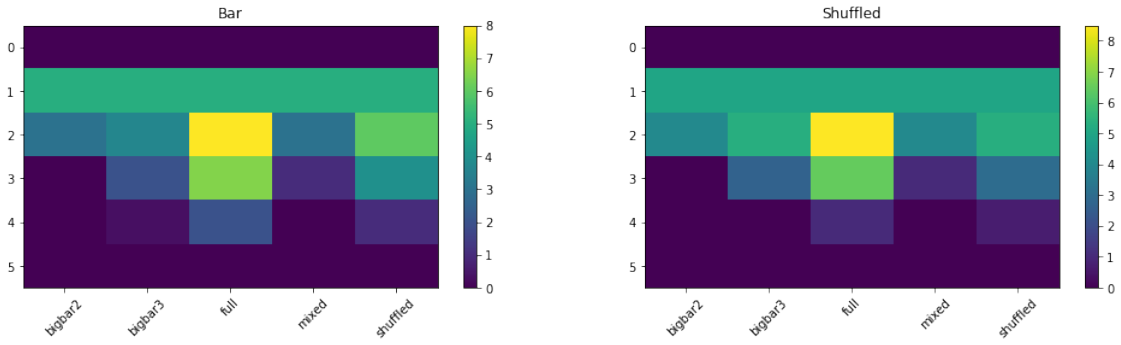

Order 2
1.4
1.2
1.0
0.8
0.6
0.4
bar
shuffle
bigbar2
bigbar3
full
mixed
shuffled

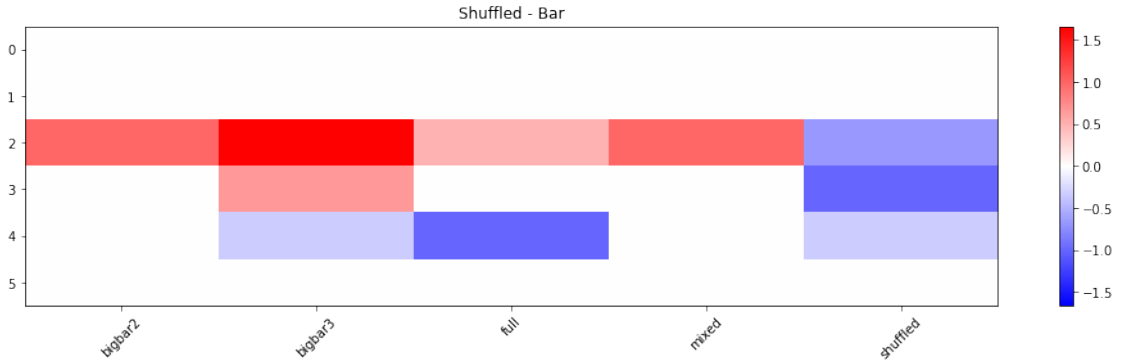

Order 3
bar
shuffle
0.8
0.6
0.4
0.2
0.0
bigbar2
bigbar3
full
mixed
shuffled

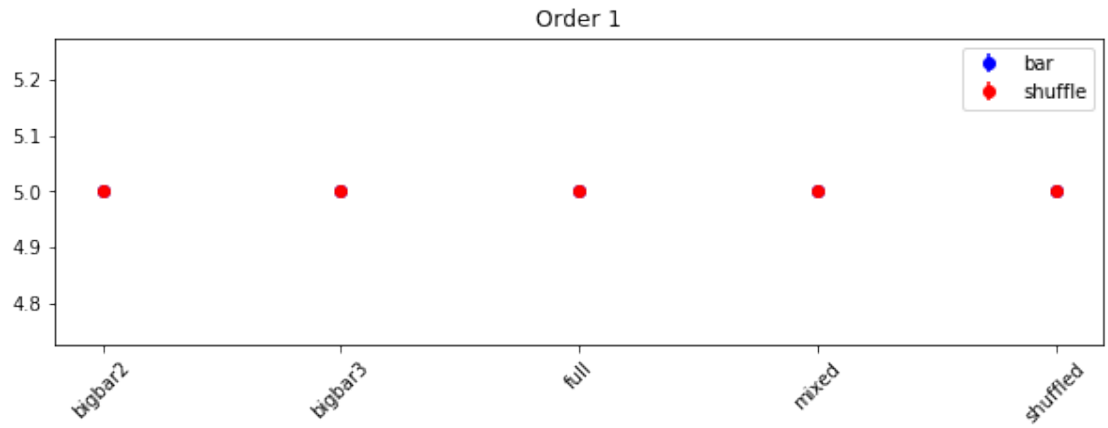

Order 4
bar
shuffle
0.15
0.10
0.05
0.00
bigbar2
bigbar3
full
mixed
shuffled

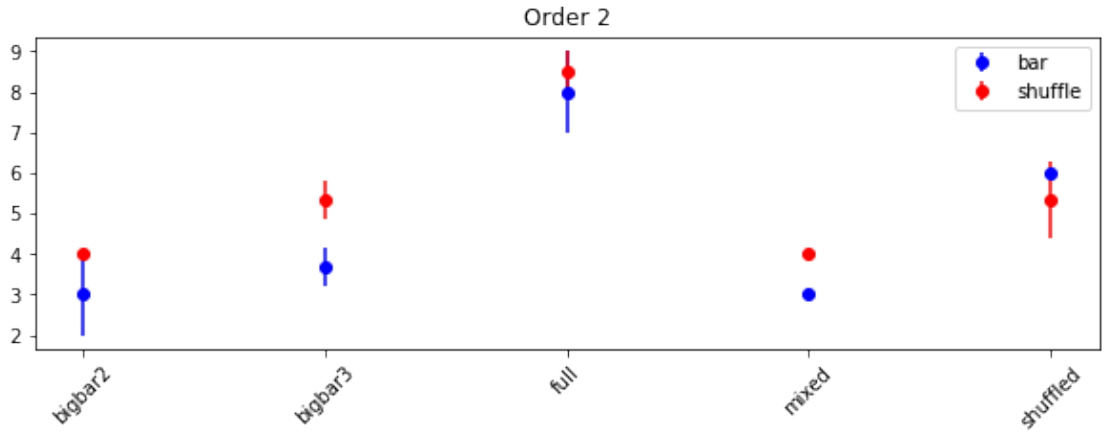

Order 5
bar
shuffle
0.04
0.02
0.00
−0.02
−0.04
bigbar2
bigbar3
full
mixed
shuffled

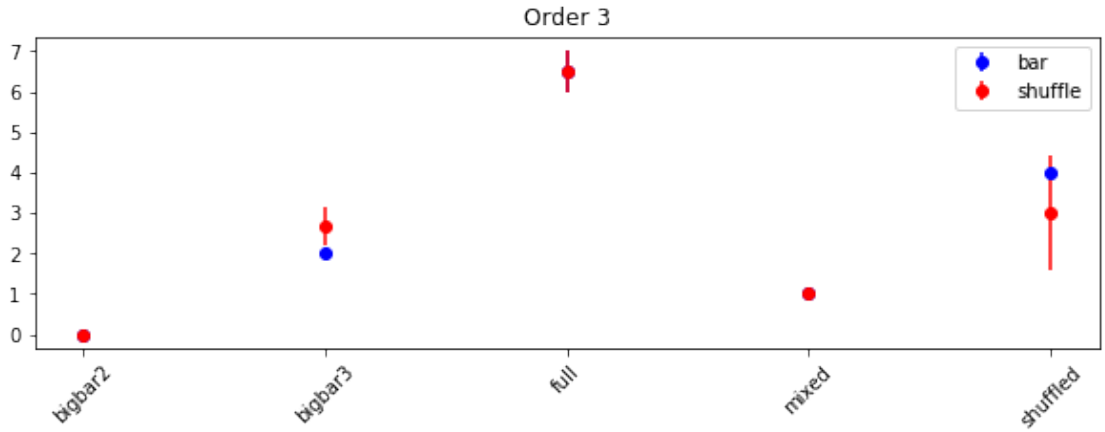
<IPython.core.display.HTML object>

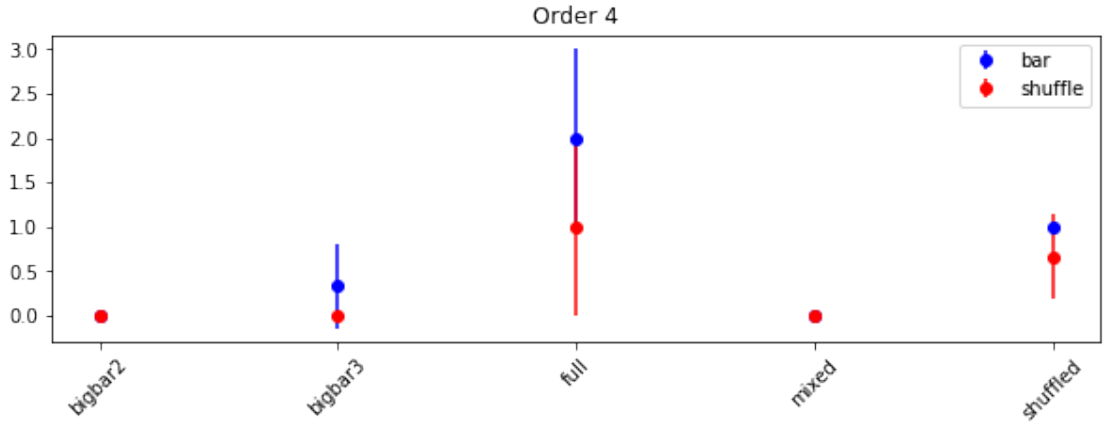


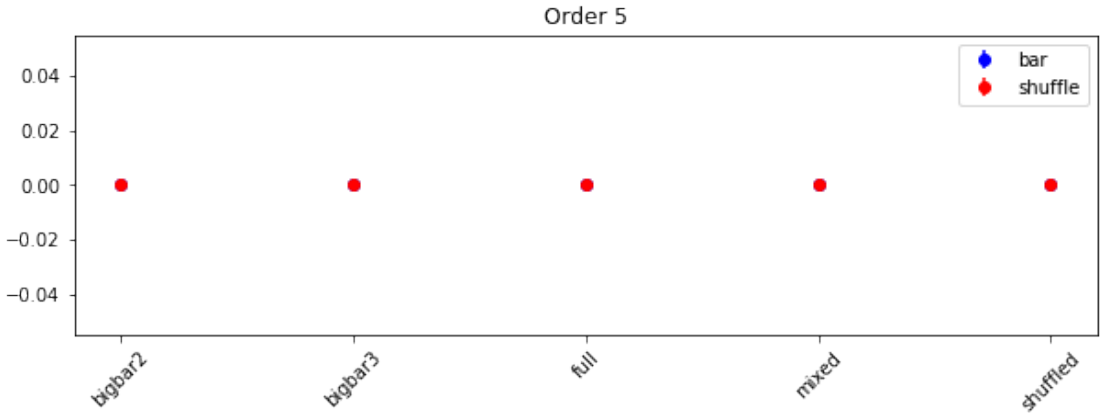





Order 2
bar
shuffle
9
8
7
6
5
4
3
2
bigbar2
bigbar3
full
mixed
shuffled


Order 3
bar
shuffle
7
6
5
4
3
2
1
0
bigbar2
bigbar3
full
mixed
shuffled


Order 4
bar
shuffle
3.0
2.5
2.0
1.5
1.0
0.5
0.0
bigbar2
bigbar3
full
mixed
shuffled



In [208]:
```
# State labels

X = bar.T.copy()

# continuity
# labels = states_labels.copy()
# unique_labels_values = np.unique(labels)
# unique_labels = unique_labels_values

# bar vs shuffled
# labels = states_labels.copy()
# labels_to_int = {'full': 1, 'shuffled': 2, 'mixed': 3, 'bigbar2': 4, 'b
# labels = np.array([labels_to_int[s] for s in labels])
# unique_labels_values = list(labels_to_int.values())
# unique_labels = labels_to_int.keys()

# Mechanism labels

X = bar.copy()
labels = mechanisms_orders.copy()
unique_labels_values = np.unique(labels)
unique_labels = unique_labels_values

# X = np.loadtxt("./tsne/mnist2500_X.txt");
# labels = np.loadtxt("./tsne/mnist2500_labels.txt");

Y = tsne.tsne(X, 2, 50, 20.0)
```

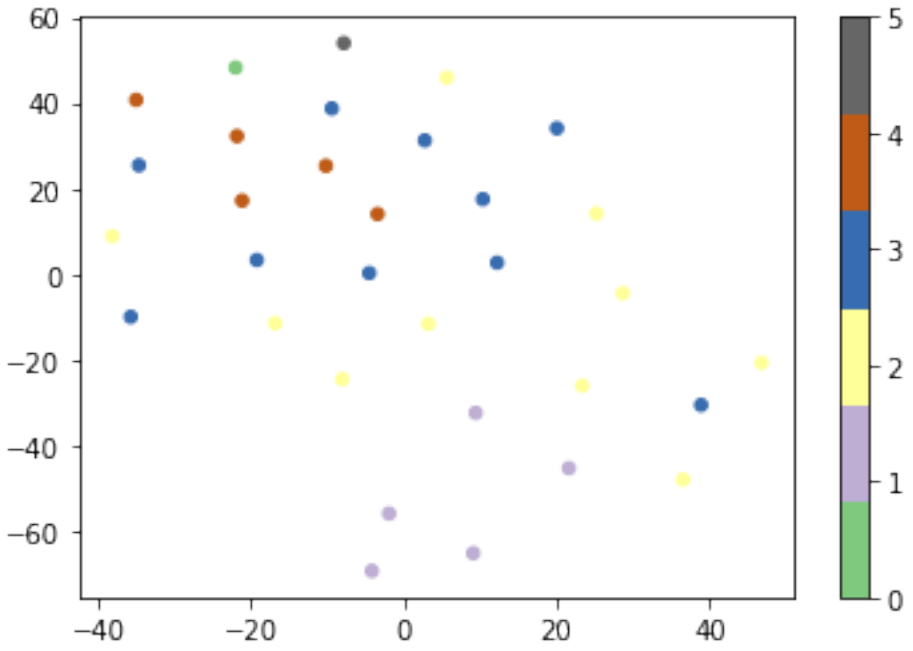
```
Preprocessing the data using PCA...
Computing pairwise distances...
Computing P-values for point  0  of  32 ...
Mean value of sigma:  0.232840008782
```

```
Iteration  10 : error is  10.8421054037
Iteration  20 : error is  10.5130750976
Iteration  30 : error is  10.1467157004
Iteration  40 : error is  10.4974449252
Iteration  50 : error is  9.60596882615
Iteration  60 : error is  9.56683722161
Iteration  70 : error is  10.6079576952
Iteration  80 : error is  10.3190645416
Iteration  90 : error is  10.4364399743
Iteration  100 : error is  9.42144935303
Iteration  110 : error is  0.981397460112
Iteration  120 : error is  0.869544627301
Iteration  130 : error is  0.797414343533
Iteration  140 : error is  0.742332215212
Iteration  150 : error is  0.652262402185
Iteration  160 : error is  0.563451187057
Iteration  170 : error is  0.533495533184
Iteration  180 : error is  0.516064522462
Iteration  190 : error is  0.507405689709
Iteration  200 : error is  0.500660731866
Iteration  210 : error is  0.488217069869
Iteration  220 : error is  0.478906036814
Iteration  230 : error is  0.468670871253
Iteration  240 : error is  0.464441261938
Iteration  250 : error is  0.461782534312
Iteration  260 : error is  0.457934031193
Iteration  270 : error is  0.452314351276
Iteration  280 : error is  0.448374677711
Iteration  290 : error is  0.444617747016
Iteration  300 : error is  0.441538968758
Iteration  310 : error is  0.437780923505
Iteration  320 : error is  0.432649981967
Iteration  330 : error is  0.425376737527
Iteration  340 : error is  0.414465379722
Iteration  350 : error is  0.39847412371
Iteration  360 : error is  0.372593376458
Iteration  370 : error is  0.345702571443
Iteration  380 : error is  0.325076743765
Iteration  390 : error is  0.314441612079
Iteration  400 : error is  0.312203191338
Iteration  410 : error is  0.311068172773
Iteration  420 : error is  0.310298741795
Iteration  430 : error is  0.309307825529
Iteration  440 : error is  0.307057359275
Iteration  450 : error is  0.299212767555
Iteration  460 : error is  0.296661877733
Iteration  470 : error is  0.295686901023
Iteration  480 : error is  0.295246680407
```

```
Iteration  490 : error is  0.294818536377
Iteration  500 : error is  0.294183050505
Iteration  510 : error is  0.29244740447
Iteration  520 : error is  0.290123105781
Iteration  530 : error is  0.289501746518
Iteration  540 : error is  0.289309833641
Iteration  550 : error is  0.289232088086
Iteration  560 : error is  0.289196393759
Iteration  570 : error is  0.289172760524
Iteration  580 : error is  0.289156120085
Iteration  590 : error is  0.289145558047
Iteration  600 : error is  0.289139461979
Iteration  610 : error is  0.289136053566
Iteration  620 : error is  0.289133758356
Iteration  630 : error is  0.289131829592
Iteration  640 : error is  0.289129966819
Iteration  650 : error is  0.289127996891
Iteration  660 : error is  0.289125816518
Iteration  670 : error is  0.289123370645
Iteration  680 : error is  0.289120634697
Iteration  690 : error is  0.289117594623
Iteration  700 : error is  0.289114238678
Iteration  710 : error is  0.289110553792
Iteration  720 : error is  0.28910652488
Iteration  730 : error is  0.289102135268
Iteration  740 : error is  0.289097373856
Iteration  750 : error is  0.289092309817
Iteration  760 : error is  0.28908688164
Iteration  770 : error is  0.289081014301
Iteration  780 : error is  0.28907467243
Iteration  790 : error is  0.289067824626
Iteration  800 : error is  0.289060479403
Iteration  810 : error is  0.289052782585
Iteration  820 : error is  0.289044693006
Iteration  830 : error is  0.28903599914
Iteration  840 : error is  0.289026624569
Iteration  850 : error is  0.289016910744
Iteration  860 : error is  0.289006613117
Iteration  870 : error is  0.288995708
Iteration  880 : error is  0.288984641153
Iteration  890 : error is  0.288972997685
Iteration  900 : error is  0.288960697502
Iteration  910 : error is  0.288947358275
Iteration  920 : error is  0.28893327515
Iteration  930 : error is  0.288918917261
Iteration  940 : error is  0.288904440792
Iteration  950 : error is  0.288889343742
Iteration  960 : error is  0.28887319453
```

```
Iteration  970 : error is  0.288855923795
Iteration  980 : error is  0.288838101773
Iteration  990 : error is  0.288819937401
Iteration  1000 : error is  0.288800558712
```

```
In [209]: plt.figure()

          cmap = plt.get_cmap('Accent', len(unique_labels))

          plt.scatter(Y[:,0], Y[:,1], 20, c=labels, label=labels, cmap=cmap,
                      vmin=min(unique_labels_values), vmax=max(unique_labels_values
          cbar = plt.colorbar(ticks=unique_labels_values)
          cbar.ax.set_yticklabels(unique_labels)  # horizontal colorbar

          plt.show()
```




```
In [116]: unique_labels_values

Out[116]: [1, 2, 3, 5, 4]

In [106]: for key, value in labels_to_int.items():
              print(key, value)

bigbar3 5
shuffled 2
```

```
bigbar2 4
mixed 3
full 1
```