MOOC_Analysis_0607

June 8, 2020

In [1]:
```
import numpy as np
import pandas as pd
import copy
import csv

from datetime import datetime
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
def CheckTheRunningTime():
    print("Current Time =", datetime.now().strftime("%H:%M:%S"))
```

In [ ]:

In [3]:
```
### Load the MOOC data from JODIE
mooc_df = pd.read_csv('../data/mooc.csv')
feat_list = ['feat_' + str(i) for i in range(4)]
index_list = ['user_id', 'item_id', 'timestamp']

mooc_df.columns = ['state_label'] + feat_list
mooc_df.index.names = index_list
print('Data size: ' , np.shape(mooc_df))

# Check the stats of data
print('# of users: ', len(mooc_df.index.unique(level='user_id')))
print('# of items: ', len(mooc_df.index.unique(level='item_id')))
print('# of timestamps: ', len(mooc_df.index.unique(level='timestamp')))

mooc_df.head()
```

```
Data size:  (411749, 5)
# of users:  7047
# of items:  97
# of timestamps:  345600
```

Out[3]:

		state_label	feat_0	feat_1	feat_2	feat_3
user_id	item_id	timestamp				

```
        0       0       0.0                 0 -0.319991 -0.435701  0.106784 -0.067309
                1       6.0                 0 -0.319991 -0.435701  0.106784 -0.067309
                2       41.0                0 -0.319991 -0.435701  0.106784 -0.067309
                1       49.0                0 -0.319991 -0.435701  0.106784 -0.067309
                2       51.0                0 -0.319991 -0.435701  0.106784 -0.067309
```

In [4]:
```
# A positive (drop-out) example
tmp_posId = 46
tmp_df = mooc_df.loc[pd.IndexSlice[[tmp_posId],:,:]]
tmp_df
```

Out[4]:

			state_label	feat_0	feat_1	feat_2	feat_3
user_id	item_id	timestamp					
46	1	62918.0	0	-0.319991	-0.435701	1.108826	3.545219
	3	62948.0	0	-0.319991	-0.435701	0.106784	-0.067309
		63301.0	0	-0.319991	-0.435701	0.106784	-0.067309
	8	63745.0	0	-0.319991	-0.435701	0.106784	-0.067309
	3	63760.0	0	-0.319991	-0.435701	0.106784	-0.067309
	5	64474.0	0	-0.319991	-0.435701	0.106784	-0.067309
	4	64477.0	0	-0.319991	-0.435701	0.106784	-0.067309
	5	64478.0	0	-0.319991	-0.435701	0.106784	-0.067309
	4	64487.0	0	-0.319991	-0.435701	0.106784	-0.067309
		64936.0	0	-0.319991	-0.435701	0.106784	-0.067309
	15	64936.0	0	-0.319991	2.108722	-0.394237	-0.067309
	5	65572.0	1	-0.319991	-0.435701	0.106784	-0.067309

In [ ]:

0.0.1 Generate the states and actions from data

In [357]:
```
# CheckTheRunningTime()
```

In [6]:
```
### Convert features to discrete states
unique_state_feat = np.array(mooc_df[feat_list].drop_duplicates())
print('# of unique states: ', len(unique_state_feat))

# Dictionary from states to features
# state_feat_dict = {i:list(unique_state_feat[i]) for i in range(len(unique_state_feat
state_feat_dict = {', '.join(map(str, list(unique_state_feat[i]))):i for i in range(le

# Descretize each feature
for feat in feat_list:
    tmp_feat_unique = np.unique(mooc_df[feat])
    tmp_feat_dict = {tmp_feat_unique[i]:i for i in range(len(tmp_feat_unique))}
    mooc_df[feat+'_d'] = mooc_df[feat].apply(lambda x: tmp_feat_dict[x])

# Get the dictionary from discrete features to indexes
feat_d_list = [feat_list[i]+'_d' for i in range(len(feat_list))]
unique_d_feat = np.array(mooc_df[feat_d_list].drop_duplicates())
```

```
        state_feat_dict = {', '.join(map(str, unique_d_feat[i])):i for i in range(len(unique_d

        # Convert the features to state indexes
        state_array = np.zeros([len(mooc_df)])

        for tmp_feat in unique_d_feat: #[tmp_start:tmp_start+1]:
            tmpIdx = [idx for idx, val in enumerate(np.array(mooc_df[feat_d_list])) if (val ==
            tmpState = state_feat_dict[', '.join(map(str, tmp_feat))]
            state_array[tmpIdx] = tmpState

        mooc_df['state_idx'] = state_array
```

```
# of unique states:  858
```

In [358]:
```
# CheckTheRunningTime()
```

In [35]:
```
### Generate the actions
mooc_sa_df = copy.deepcopy(mooc_df)
mooc_sa_df.reset_index(inplace=True)

# Check the adjacent difference of states and items
mooc_sa_df['state_diff'] = mooc_sa_df.groupby('user_id')['state_idx'].diff().fillna(0)
mooc_sa_df['item_diff'] = mooc_sa_df.groupby('user_id')['item_id'].diff().fillna(0).to

# Convert the difference to boolean
state_diff_flag = [diff != 0 for diff in mooc_sa_df['state_diff'].tolist()]
item_diff_flag = [diff != 0 for diff in mooc_sa_df['item_diff'].tolist()]

# Generate the actions
# 0: Both state and item keep the same
# 1: Item keeps the same while state changes
# 2: State keeps the same while item changes
# 3: Both state and item changes
action_list = [item_diff_flag[i]*2 + state_diff_flag[i]*1 for i in range(len(state_di
mooc_sa_df['action'] = action_list

# Check the unique numbers for actions
print('Unique # of actions: ', np.unique(action_list, return_counts=True))

# Get the state-action pairs
mooc_sa_df.state_idx = mooc_sa_df.state_idx.astype(int)
mooc_sa_df.action = mooc_sa_df.action.astype(int)
mooc_sa_df['SA'] = mooc_sa_df.groupby('user_id').apply(
    lambda x: pd.Series([tuple(i) for i in x[['state_idx', 'action']].values])).tolist

# Slice the columns
sel_col = index_list + ['state_label'] + feat_list + ['state_idx', 'action', 'SA']
mooc_sa_df = mooc_sa_df[sel_col]
```

```
Unique # of actions:  (array([0, 1, 2, 3]), array([ 60866,  21381, 132412, 197090], dtype=int64
```

```
In [360]: mooc_sa_df[:100]
```

Out[360]:

	user_id	item_id	timestamp	state_label	feat_0	feat_1	feat_2	\
0	0	0	0.0	0	-0.319991	-0.435701	0.106784	
1	0	1	6.0	0	-0.319991	-0.435701	0.106784	
2	0	2	41.0	0	-0.319991	-0.435701	0.106784	
3	0	1	49.0	0	-0.319991	-0.435701	0.106784	
4	0	2	51.0	0	-0.319991	-0.435701	0.106784	
5	0	3	55.0	0	-0.319991	-0.435701	0.106784	
6	0	4	59.0	0	-0.319991	-0.435701	0.106784	
7	0	5	62.0	0	-0.319991	-0.435701	0.106784	
8	0	6	65.0	0	-0.319991	-0.435701	0.106784	
9	0	7	113.0	0	-0.319991	-0.435701	1.108826	
10	0	8	226.0	0	-0.319991	-0.435701	0.607805	
11	0	9	974.0	0	-0.319991	-0.435701	1.108826	
12	0	0	1000.0	0	-0.319991	-0.435701	0.106784	
13	0	9	1172.0	0	-0.319991	-0.435701	1.108826	
14	0	2	1182.0	0	-0.319991	-0.435701	0.106784	
15	0	1	1185.0	0	-0.319991	-0.435701	0.106784	
16	0	10	1687.0	0	-0.319991	-0.435701	0.106784	
17	1	10	7262.0	0	-0.319991	-0.435701	0.106784	
18	1	1	7266.0	0	-0.319991	-0.435701	0.106784	
19	1	2	7273.0	0	-0.319991	-0.435701	0.607805	
20	1	7	7289.0	0	-0.319991	-0.435701	0.106784	
21	1	0	7299.0	0	-0.319991	-0.435701	0.106784	
22	1	11	7319.0	0	-0.319991	-0.435701	0.106784	
23	1	12	7839.0	0	-0.319991	-0.435701	0.106784	
24	1	11	7846.0	0	-0.319991	-0.435701	0.106784	
25	2	1	37868.0	0	-0.319991	-0.435701	0.607805	
26	3	1	37953.0	0	-0.319991	-0.435701	0.106784	
27	4	1	37969.0	0	-0.319991	-0.435701	1.108826	
28	4	3	38018.0	0	-0.319991	-0.435701	0.607805	
29	3	10	38113.0	0	-0.319991	-0.435701	0.106784	
..	...	...	...	...	...	...	...	
70	12	7	38931.0	0	-0.319991	-0.435701	0.106784	
71	12	8	38941.0	0	-0.319991	-0.435701	0.106784	
72	14	1	39047.0	0	-0.319991	-0.435701	0.607805	
73	14	2	39065.0	0	-0.319991	-0.435701	0.106784	
74	14	1	39073.0	0	-0.319991	-0.435701	1.108826	
75	13	4	39105.0	0	-0.319991	-0.435701	0.106784	
76	13	13	39105.0	0	-0.319991	2.108722	-0.394237	
77	14	3	39113.0	0	-0.319991	-0.435701	0.106784	
78	14	9	39133.0	0	-0.319991	-0.435701	0.106784	
79	14	13	39133.0	0	-0.319991	2.108722	-0.394237	
80	14	9	39157.0	0	-0.319991	-0.435701	0.106784	

81	13	8	39161.0	0	-0.319991	-0.435701	0.106784
82	14	7	39162.0	0	-0.319991	-0.435701	0.106784
83	15	1	39164.0	0	-0.319991	-0.435701	0.106784
84	15	2	39190.0	0	-0.319991	-0.435701	0.106784
85	13	15	39193.0	0	-0.319991	2.108722	-0.394237
86	16	1	39193.0	0	-0.319991	-0.435701	0.106784
87	15	3	39197.0	0	-0.319991	-0.435701	0.106784
88	17	1	39220.0	0	-0.319991	-0.435701	0.106784
89	13	8	39229.0	0	-0.319991	-0.435701	0.106784
90	13	5	39232.0	0	-0.319991	-0.435701	0.106784
91	15	13	39241.0	0	-0.319991	2.108722	-0.394237
92	18	1	39251.0	0	-0.319991	-0.435701	0.106784
93	13	4	39253.0	0	-0.319991	-0.435701	0.106784
94	13	16	39254.0	0	-0.319991	2.108722	-0.394237
95	17	10	39258.0	0	-0.319991	-0.435701	0.106784
96	18	3	39273.0	0	-0.319991	-0.435701	0.106784
97	19	1	39304.0	0	-0.319991	-0.435701	0.106784
98	19	2	39313.0	0	-0.319991	-0.435701	0.106784
99	19	3	39319.0	0	-0.319991	-0.435701	0.106784

	feat_3	state_idx	action	SA
0	-0.067309	0	0	(0, 0)
1	-0.067309	0	2	(0, 2)
2	-0.067309	0	2	(0, 2)
3	-0.067309	0	2	(0, 2)
4	-0.067309	0	2	(0, 2)
5	-0.067309	0	2	(0, 2)
6	-0.067309	0	2	(0, 2)
7	-0.067309	0	2	(0, 2)
8	-0.067309	0	2	(0, 2)
9	12.777235	1	3	(1, 3)
10	149.451211	2	3	(2, 3)
11	3.344523	3	3	(3, 3)
12	-0.067309	0	3	(0, 3)
13	1.136867	4	3	(4, 3)
14	-0.067309	0	3	(0, 3)
15	-0.067309	0	2	(0, 2)
16	-0.067309	0	2	(0, 2)
17	-0.067309	0	0	(0, 0)
18	-0.067309	0	2	(0, 0)
19	0.936171	5	3	(0, 2)
20	-0.067309	0	3	(0, 2)
21	-0.067309	0	2	(0, 2)
22	-0.067309	0	2	(0, 2)
23	-0.067309	0	2	(9, 3)
24	-0.067309	0	2	(0, 3)
25	1.337563	6	0	(9, 3)
26	-0.067309	0	0	(0, 3)

```
27    7.157747          7       0  (0, 2)
28    0.133387          8       3  (0, 2)
29   -0.067309          0       2  (0, 0)
..         ...        ...     ...     ...
70   -0.067309          0       3  (0, 2)
71   -0.067309          0       2  (0, 2)
72    2.742435         17       0  (0, 0)
73   -0.067309          0       3  (0, 0)
74    4.147307         18       3  (0, 2)
75   -0.067309          0       2  (9, 3)
76   -0.067309          9       3  (0, 0)
77   -0.067309          0       3  (0, 2)
78   -0.067309          0       2  (5, 3)
79   -0.067309          9       3  (0, 3)
80   -0.067309          0       3  (0, 2)
81   -0.067309          0       3  (0, 2)
82   -0.067309          0       2  (0, 2)
83   -0.067309          0       0  (0, 2)
84   -0.067309          0       2  (0, 2)
85   -0.067309          9       3  (0, 0)
86   -0.067309          0       0  (0, 0)
87   -0.067309          0       2  (0, 0)
88   -0.067309          0       0  (5, 3)
89   -0.067309          0       3  (0, 3)
90   -0.067309          0       2  (0, 0)
91   -0.067309          9       3  (0, 2)
92   -0.067309          0       0  (0, 2)
93   -0.067309          0       2  (0, 2)
94   -0.067309          9       3  (9, 3)
95   -0.067309          0       2  (0, 3)
96   -0.067309          0       2  (9, 3)
97   -0.067309          0       0  (9, 2)
98   -0.067309          0       2  (0, 3)
99   -0.067309          0       2  (0, 2)

[100 rows x 11 columns]
```

In [ ]:

0.0.2 Split the data for training, validation, and testing

In [37]:
```
def GetTrajbyIdxfromDf(df, idx):
    return np.array(df.loc[df.user_id.isin(idx)].groupby('user_id').apply(lambda x: np
```

In [38]:
```
### Reformulate the data for modeling
# Check the labels
all_labs = mooc_sa_df.groupby('user_id').apply(lambda x: x['state_label'].tolist()[-1
print('Perc of drop-out students: ', all_labs.count(1) / len(all_labs))
```

```
        print('Perc of positive iteractions: ', mooc_sa_df['state_label'].tolist().count(1) /

        # Get the userIds
        all_userId = mooc_df.index.unique(level='user_id')

        # Split the training, validation, and testing data
        tr_prec, val_prec, te_prec = 0.6, 0.2, 0.2

        # Get the indexes
        tr_idx = np.arange(0, int(len(all_userId) * tr_prec))
        val_idx = np.arange(int(len(all_userId) * tr_prec), int(len(all_userId) * (tr_prec +
        te_idx = np.arange(int(len(all_userId) * (tr_prec + val_prec)), len(all_userId))

        # Slice the trajectoreis
        tr_traj = GetTrajbyIdxfromDf(mooc_sa_df, tr_idx)
        val_traj = GetTrajbyIdxfromDf(mooc_sa_df, val_idx)
        te_traj = GetTrajbyIdxfromDf(mooc_sa_df, te_idx)

        # Slice the labels
        tr_lab = [all_labs[idx] for idx in tr_idx]
        val_lab = [all_labs[idx] for idx in val_idx]
        te_lab = [all_labs[idx] for idx in te_idx]

        # Check the number of splitted data
        print('\nThe # of training, validation, and testing data: ', len(tr_idx), len(val_idx
```

```
Perc of drop-out students:  0.5769831133815808
Perc of positive iteractions:  0.009874948087305616

The # of training, validation, and testing data:  4228 1409 1410
```

```
In [ ]:
```

0.0.3 Learn the rewards by MLIRL

```
In [39]: from mdp import MDP
         from MLIRL import MLIRL, DemoWeights
```

```
In [40]: # Normalize the transitions
         def NormTransitions(P, addTurb = False):
             norm_P = copy.deepcopy(P)
             # Add a small value to each element of P with the probability of 0.1
             for i in range(0, len(P)):
                 # ------------------------------------------------------------
                 # Add turbulence to the transitions
                 if addTurb:
                     for j in range(0, len(P[i])):
                         norm_P[i,j] = P[i,j]
```

```
                        random.seed(np.prod(np.shape(P))+i*len(P)+j)
                        if random.random() < 0.1:
                            norm_P[i,j] += random.random()*0.01
                # ------------------------------------------------------------
                # Normalize the transitions
                tmp_sum = np.sum(norm_P[i], axis=1)
                tmp_sum[tmp_sum == 0] = 1e-10
                norm_P[i] = norm_P[i] / tmp_sum[:, None]
            return norm_P
```

In [41]:
```
# Number of states and actions
n_actions = len(np.unique(mooc_sa_df['action']))
n_states = len(np.unique(mooc_sa_df['state_idx']))

# Estimate the transition probabilities
init_trans = np.zeros([n_states, n_actions, n_states])

# Check the start and end states
nextStates = []
for tmpId in all_userId[:1]:
    tmpState = mooc_sa_df.loc[mooc_sa_df.user_id == tmpId].state_idx.tolist()
    tmpCurState, tmpNextState = tmpState[:-1], tmpState[1:]
    tmpAction = mooc_sa_df.loc[mooc_sa_df.user_id == tmpId].action.tolist()[:-1]
    for idx in range(len(tmpCurState)):
        init_trans[tmpCurState[idx], tmpAction[idx], tmpNextState[idx]] += 1

# Normalize the transitions
addTurb_flag = False
transitions = NormTransitions(init_trans, addTurb = addTurb_flag)

# Set the basic parameters
para = {}
para['n_states'], para['n_actions'] = n_states, n_actions
para['transitions'] = transitions
para['baseline_rewards'] = []

# States to feats mapping
para['reward_pattern'] = 'S'
para['feat_map'] = np.eye((para['n_states']))
para['GAMMA'] = 0.85

para['MLIRL_ITERS'] = 300
para['MLIRL_QLITERS'] = 60
para['MLIRL_w'] = 0.5
para['MLIRL_clip'] = 0.01
para['MLIRL_weight'] = 'ones'
para['verbo'] = True
```

In [42]:
```
# Learn the rewards by MLIRL
```

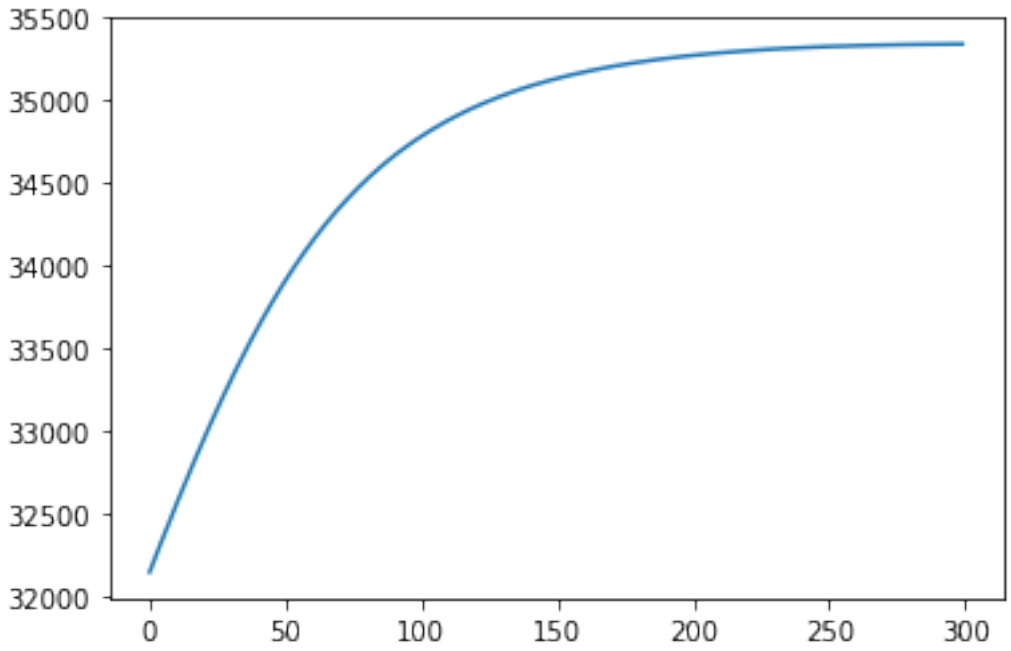
```
        neg_idx = [idx for idx, val in enumerate(tr_lab) if val == 0]
        selDemo = np.array([tr_traj[i] for i in neg_idx])

        w = DemoWeights(selDemo, para)
        temp_reward, llh = MLIRL(selDemo, w, para)
```

```
iteration 10: llh = 32528.81600787856
-------------------------------------
iteration 20: llh = 32921.54525946642
-------------------------------------
iteration 30: llh = 33282.529637350264
-------------------------------------
iteration 40: llh = 33605.86946148189
-------------------------------------
iteration 50: llh = 33888.84693236293
-------------------------------------
iteration 60: llh = 34131.85799059327
-------------------------------------
iteration 70: llh = 34337.62194438375
-------------------------------------
iteration 80: llh = 34510.18417783568
-------------------------------------
iteration 90: llh = 34654.071211814764
-------------------------------------
iteration 100: llh = 34773.718017684194
-------------------------------------
iteration 110: llh = 34873.1490112546
-------------------------------------
iteration 120: llh = 34955.84628679117
-------------------------------------
iteration 130: llh = 35024.734866411156
-------------------------------------
iteration 140: llh = 35082.039532735645
-------------------------------------
iteration 150: llh = 35129.614760644465
-------------------------------------
iteration 160: llh = 35169.174832487115
-------------------------------------
iteration 170: llh = 35202.099394046534
-------------------------------------
iteration 180: llh = 35229.502489028906
-------------------------------------
iteration 190: llh = 35252.28841155181
-------------------------------------
iteration 200: llh = 35271.1953965485
-------------------------------------
iteration 210: llh = 35286.829607340274
-------------------------------------
```

```
iteration 220: llh = 35299.69157351108
-------------------------------------
iteration 230: llh = 35310.19681919202
-------------------------------------
iteration 240: llh = 35318.692036333036
-------------------------------------
iteration 250: llh = 35325.46783871191
-------------------------------------
iteration 260: llh = 35330.76888262271
-------------------------------------
iteration 270: llh = 35334.801949537934
-------------------------------------
iteration 280: llh = 35337.74244235895
-------------------------------------
iteration 290: llh = 35339.73963912541
-------------------------------------
iteration 300: llh = 35340.92096729562
-------------------------------------
```

In [43]:
```
# Check the convergence
plt.plot(llh)
```

Out[43]: [<matplotlib.lines.Line2D at 0x1ca78af4710>]



In [ ]:

0.0.4 Define the normality score

```
In [347]: def RewardForGivenTrainingDemo(demo, reward):
              state_list = [demo[1][i][0] for i in range(len(demo[1]))]
              reward_sum = np.sum([reward[i] for i in state_list])
              reward_avg = reward_sum / len(state_list)
              return reward_sum, reward_avg

In [348]: def CalNormalityScoreForGivenDemo(demo, reward, reward_avg, reward_std):
              state_list = [demo[1][i][0] for i in range(len(demo[1]))]
              score = sum([(reward[i] - reward_avg) / reward_std for i in state_list]) / len(de
              return score

In [349]: def CheckNormalityScoreStats(traj, temp_reward, reward_avg, reward_std, verbo = True
              score_list = [CalNormalityScoreForGivenDemo(traj[idx], temp_reward, reward_avg, r
                            for idx in range(len(traj))]
              if verbo:
                  print('Max Score: ', np.max(score_list))
                  print('Min Score: ', np.min(score_list))
                  print('Mean Score: ', np.mean(score_list))
                  print('Median Score: ', np.median(score_list))
                  print('\n')
              return score_list

In [350]: from sklearn import metrics
          from sklearn.metrics import confusion_matrix
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import recall_score
          from sklearn.metrics import precision_score
          from sklearn.metrics import f1_score
          from sklearn.metrics import roc_curve, auc

          def ModelEvaluate(real_lab, pre_lab, labels_list, avg_patterns='weighted', verbo=Fals
              conf_matrix = confusion_matrix(real_lab, pre_lab, labels=labels_list)
              acc = accuracy_score(real_lab, pre_lab)
              recall = precision_score(real_lab, pre_lab, labels=labels_list, average=avg_patte
              precision = recall_score(real_lab, pre_lab, labels=labels_list, average=avg_patte
              fscore = f1_score(real_lab, pre_lab, labels=labels_list, average=avg_patterns)

              if verbo:
                  print('Performance Measurements:')
                  print('Confusion matrix: \n', conf_matrix)
                  print('Accuracy: ', acc)
                  print('Recall: ', recall)
                  print('Precision: ', precision)
                  print('F-score: ', fscore)
              return conf_matrix, [acc, recall, precision, fscore]
```

Determine the class label by normality score

```
In [351]: def CheckScoreBasedPredResults(traj, real_lab, temp_reward, reward_avg, reward_std, s
              # Check the normality score
              score_list = CheckNormalityScoreStats(traj, temp_reward, reward_avg, reward_std,

              # Get the predicted label
              # Defined by: r - mean(r) / std(r)
              pred_lab = [0 if score_list[i] > score_thres else 1 for i in range(len(score_list

              # Get the predicted scores
              prob_lab = [(i - score_thres) / (max(score_list) - score_thres) for i in score_li
              prob_lab = [1 - (i - min(prob_lab)) / (max(prob_lab) - min(prob_lab)) for i in pr

              _, _ = ModelEvaluate(real_lab, pred_lab, labels_list = np.arange(2), verbo=True)
              te_fpr, te_tpr, thresholds = metrics.roc_curve(real_lab, prob_lab)
              te_roc_auc = metrics.auc(te_fpr, te_tpr)
              print('AUC: ', te_roc_auc)

In [352]: # Get the average rewards for all trajectories
          reward_avg_list = []
          for idx in range(len(selDemo)):
              tmp_demo = selDemo[idx]
              tmp_reward_sum, tmp_reward_avg = RewardForGivenTrainingDemo(tmp_demo, temp_reward
              reward_avg_list.append(tmp_reward_avg)
          reward_avg, reward_std = np.mean(reward_avg_list), np.std(reward_avg_list)

          print('Average reward in the demos: ', reward_avg, '(', reward_std, ')')

Average reward in the demos:  2.237078969177945 ( 0.036239201066882276 )


In [353]: # Search for a threshold to determine the class labels
          score_thre = 10
          print('Validation: ')
          CheckScoreBasedPredResults(val_traj, val_lab, temp_reward, reward_avg, reward_std, sc

          print('\n* Testing: ')
          CheckScoreBasedPredResults(te_traj, te_lab, temp_reward, reward_avg, reward_std, scor

Validation:
Performance Measurements:
Confusion matrix:
 [[ 70 445]
 [ 18 876]]
Accuracy:  0.6713981547196594
Recall:  0.7114981387043651
Precision:  0.6713981547196594
F-score:  0.5867257542873795
AUC:  0.5395365000760193
```

```
* Testing:
Performance Measurements:
Confusion matrix:
 [[ 44 494]
 [ 22 850]]
Accuracy:  0.6340425531914894
Recall:  0.6454998311381289
Precision:  0.6340425531914894
F-score:  0.5300263649307253
AUC:  0.4942116998056001
```

In [ ]:

Determine the class label by log-likelihood

In [354]:
```
def NormalizeAList(score, stats = False):
    norm_score = [1 - (i - min(score)) / (max(score) - min(score)) for i in score]
    if stats:
        return norm_score, max(score), min(score)
    else:
        return norm_score
```

In [355]:
```
from EM_IRL_utils import calLogLLH_MLIRL

def CheckLLHBasedPredResults(traj, real_lab, temp_reward, thres, para):
    # Check the normality score
    logLLH = calLogLLH_MLIRL(traj, temp_reward, para)
    norm_logLLH = NormalizeAList(logLLH)
    pred_lab = [0 if norm_logLLH[i] > thres else 1 for i in range(len(norm_logLLH))]

    # Get the predicted scores
    prob_lab = NormalizeAList([(thres - i) / thres for i in norm_logLLH])

    _, _ = ModelEvaluate(real_lab, pred_lab, labels_list = np.arange(2), verbo=True)
    te_fpr, te_tpr, thresholds = metrics.roc_curve(real_lab, prob_lab)
    te_roc_auc = metrics.auc(te_fpr, te_tpr)
    print('AUC: ', te_roc_auc)
```

In [359]:
```
llh_thres = 0.15
print('* Validation: ')
CheckLLHBasedPredResults(val_traj, val_lab, temp_reward, llh_thres, para)

print('\n* Testing: ')
CheckLLHBasedPredResults(te_traj, te_lab, temp_reward, llh_thres, para)
```

```
* Validation:
Performance Measurements:
Confusion matrix:
```

```
 [[374 141]
 [161 733]]
Accuracy:  0.7856635911994322
Recall:  0.7876452427014616
Precision:  0.7856635911994322
F-score:  0.7864925803263381
AUC:  0.14471232162637648

* Testing:
Performance Measurements:
Confusion matrix:
 [[274 264]
 [129 743]]
Accuracy:  0.7212765957446808
Recall:  0.7157296841962657
Precision:  0.7212765957446808
F-score:  0.7112958349897711
AUC:  0.23168441560656183
```

In [ ]:

In [ ]:

In [ ]: