Assignment 4

May 25, 2021

---

*You are currently looking at **version 1.2** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Assignment 4

In [110]:
```
import networkx as nx
import pandas as pd
import numpy as np
import pickle
```

1.1 Part 1 - Random Graph Identification

For the first part of this assignment you will analyze randomly generated graphs and determine which algorithm created them.

In [111]:
```
P1_Graphs = pickle.load(open('A4_graphs','rb'))
P1_Graphs
```

Out[111]:
```
[<networkx.classes.graph.Graph at 0x7fb8f425e128>,
 <networkx.classes.graph.Graph at 0x7fb8f4230400>,
 <networkx.classes.graph.Graph at 0x7fb8f4230a58>,
 <networkx.classes.graph.Graph at 0x7fb8f4230f28>,
 <networkx.classes.graph.Graph at 0x7fb942b79cf8>]
```

`P1_Graphs` is a list containing 5 networkx graphs. Each of these graphs were generated by one of three possible algorithms: * Preferential Attachment (`'PA'`) * Small World with low probability of rewiring (`'SW_L'`) * Small World with high probability of rewiring (`'SW_H'`)

Anaylze each of the 5 graphs and determine which of the three algorithms generated the graph.

The `graph_identification` function should return a list of length 5 where each element in the list is either `'PA'`, `'SW_L'`, or `'SW_H'`.

In [112]:
```
for i in P1_Graphs:
    print(nx.average_clustering(i))
#def graph_identification():
    #list_1 = []
    # Your Code Here
    #for i in P1_Graphs:
        #print(nx.average_clustering(i))
        #if nx.average_clustering(i) > 0.1:
            #list_1.append('SW_L')
        #elif nx.average_clustering(i) < 0.1 and nx.average_clustering(i) > 0.01:
            #list_1.append('SW_H')
        #else:
            #list_1.append('PA')

    #return list_1# Your Answer Here

def degree_distribution(G):
    degrees = G.degree()
    degree_values = sorted(set(degrees.values()))
    histogram = [list(degrees.values()).count(i)/float(nx.number_of_nodes( G)) for i i
    return histogram

def graph_identification():
    list_1 = []
    for i in P1_Graphs:
        clustering = nx.average_clustering(i)
        shortest_path = nx.average_shortest_path_length(i)
        degree_hist = degree_distribution(i)
        if len(degree_hist)>10:
            list_1.append('PA')
        elif clustering < 0.1:
            list_1.append('SW_H')
        else:
            list_1.append('SW_L')
    return list_1
#graph_identification()
```

```
0.03167539146454044
0.5642419635919628
0.4018222222222227
0.03780379975223251
0.0033037037037037037
```

1.2 Part 2 - Company Emails

For the second part of this assignment you will be workking with a company's email network where each node corresponds to a person at the company, and each edge indicates that at least one email has been sent between two people.

The network also contains the node attributes `Department` and `ManagementSalary`.

`Department` indicates the department in the company which the person belongs to, and `ManagementSalary` indicates whether that person is receiving a management position salary.

```
In [113]: G = nx.read_gpickle('email_prediction.txt')

          print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 1005
Number of edges: 16706
Average degree:  33.2458
```

1.2.1 Part 2A - Salary Prediction

Using network `G`, identify the people in the network with missing values for the node attribute `ManagementSalary` and predict whether or not these individuals are receiving a management position salary.

To accomplish this, you will need to create a matrix of node features using networkx, train a sklearn classifier on nodes that have `ManagementSalary` data, and predict a probability of the node receiving a management salary for nodes where `ManagementSalary` is missing.

Your predictions will need to be given as the probability that the corresponding employee is receiving a management position salary.

The evaluation metric for this assignment is the Area Under the ROC Curve (AUC).

Your grade will be based on the AUC score computed for your classifier. A model which with an AUC of 0.88 or higher will receive full points, and with an AUC of 0.82 or higher will pass (get 80% of the full points).

Using your trained classifier, return a series of length 252 with the data being the probability of receiving management salary, and the index being the node id.

```
Example:

    1       1.0
    2       0.0
    5       0.8
    8       1.0
        ...
    996     0.7
    1000    0.5
    1001    0.0
    Length: 252, dtype: float64
```

```
In [114]: #list_of_emails = G.edges(data=True)
          #print(list_of_emails)
          df = pd.DataFrame(index=G.nodes())
          df['ManagementSalary'] = pd.Series(nx.get_node_attributes(G, 'ManagementSalary'))
          df['Department'] = pd.Series(nx.get_node_attributes(G, 'Department'))
          df['clustering'] = pd.Series(nx.clustering(G))
          df['degree'] = pd.Series(G.degree())
          #df_edges = pd.DataFrame(index=G.edges())
          #print(df)
          test_set = df[df['ManagementSalary'].isnull()]
          #print(len(test_set))
          train_set = df.dropna()
          #print(train_set)
          from sklearn.model_selection import train_test_split
          train_features = train_set.columns.drop('ManagementSalary')
          test_set = test_set[train_features]

          X_train, X_test, y_train, y_test = train_test_split(train_set[train_features],
                                                              train_set.ManagementSalary,
                                                              random_state=0,
                                                              test_size=0.5)
          def salary_predictions():

              # Your Code Here
              from sklearn.ensemble import RandomForestClassifier
              from sklearn.ensemble import GradientBoostingClassifier
              from sklearn.metrics import roc_auc_score
              clf_RF = RandomForestClassifier(max_features = 3, random_state = 0, max_depth=3, m
              clf_RF.fit(X_train, y_train)
              clf_GDBT= GradientBoostingClassifier(learning_rate = 0.01, max_depth = 8, random_s
              clf_GDBT.fit(X_train, y_train)
              roc_score_forest = roc_auc_score(y_test, clf_RF.predict_proba(X_test)[:,1])
              roc_score = roc_auc_score(y_test, clf_GDBT.predict_proba(X_test)[:,1])
              print(roc_score_forest)
              print(roc_score)
              preds = pd.Series(data=clf_RF.predict_proba(test_set)[:,1], index=test_set.index)

              return preds# Your Answer Here
          #salary_predictions()
```

1.2.2 Part 2B - New Connections Prediction

For the last part of this assignment, you will predict future connections between employees of the network. The future connections information has been loaded into the variable `future_connections`. The index is a tuple indicating a pair of nodes that currently do not have a connection, and the `Future Connection` column indicates if an edge between those two nodes will exist in the future, where a value of 1.0 indicates a future connection.

```
In [115]: future_connections = pd.read_csv('Future_Connections.csv', index_col=0, converters={0:
          future_connections.head(10)
```

```
Out[115]:             Future Connection
          (6, 840)                  0.0
          (4, 197)                  0.0
          (620, 979)                0.0
          (519, 872)                0.0
          (382, 423)                0.0
          (97, 226)                 1.0
          (349, 905)                0.0
          (429, 860)                0.0
          (309, 989)                0.0
          (468, 880)                0.0
```

Using network `G` and `future_connections`, identify the edges in `future_connections` with missing values and predict whether or not these edges will have a future connection.

To accomplish this, you will need to create a matrix of features for the edges found in `future_connections` using networkx, train a sklearn classifier on those edges in `future_connections` that have `Future Connection` data, and predict a probability of the edge being a future connection for those edges in `future_connections` where `Future Connection` is missing.

Your predictions will need to be given as the probability of the corresponding edge being a future connection.

The evaluation metric for this assignment is the Area Under the ROC Curve (AUC).

Your grade will be based on the AUC score computed for your classifier. A model which with an AUC of 0.88 or higher will receive full points, and with an AUC of 0.82 or higher will pass (get 80% of the full points).

Using your trained classifier, return a series of length 122112 with the data being the probability of the edge being a future connection, and the index being the edge as represented by a tuple of nodes.

```
Example:

    (107, 348)    0.35
    (542, 751)    0.40
    (20, 426)     0.55
    (50, 989)     0.35
              ...
    (939, 940)    0.15
    (555, 905)    0.35
    (75, 101)     0.65
    Length: 122112, dtype: float64
```

```
In [120]: #print(G.nodes(data=True))
          #df = pd.DataFrame(index=G.nodes())
          from sklearn.model_selection import train_test_split
          def new_connections_predictions():
```

```
    # Your Code Here
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.metrics import roc_auc_score
    for node in G.nodes():
        G.node[node]['community'] = G.node[node]['Department']
    preferential_attachment = list(nx.preferential_attachment(G))
    df = pd.DataFrame(index=[(x[0], x[1]) for x in preferential_attachment])
    df['preferential_attachment'] = [x[2] for x in preferential_attachment]
    cn_soundarajan_hopcroft = list(nx.cn_soundarajan_hopcroft(G))
    df_cn_soundarajan_hopcroft = pd.DataFrame(index=[(x[0], x[1]) for x in cn_soundara
    df_cn_soundarajan_hopcroft['cn_soundarajan_hopcroft'] = [x[2] for x in cn_soundara
    df = df.join(df_cn_soundarajan_hopcroft,how='outer')
    df['cn_soundarajan_hopcroft'] = df['cn_soundarajan_hopcroft'].fillna(value=0)
    df['resource_allocation_index'] = [x[2] for x in list(nx.resource_allocation_index
    df['jaccard_coefficient'] = [x[2] for x in list(nx.jaccard_coefficient(G))]
    df = future_connections.join(df,how='outer')
    df_train = df[~pd.isnull(df['Future Connection'])]
    df_test = df[pd.isnull(df['Future Connection'])]
    features = ['cn_soundarajan_hopcroft', 'preferential_attachment', 'resource_alloca
    df_test = df_test[features]
    X_train, X_test, y_train, y_test = train_test_split(df_train[features],
                                                        df_train['Future Connection'],
                                                        random_state=0,
                                                        test_size=0.5)
    clf_RF = RandomForestClassifier(max_features = 3, random_state = 0, max_depth=3, m
    clf_RF.fit(X_train, y_train)
    clf_GDBT= GradientBoostingClassifier(learning_rate = 0.01, max_depth = 8, random_s
    clf_GDBT.fit(X_train, y_train)
    roc_score_forest = roc_auc_score(y_test, clf_RF.predict_proba(X_test)[:,1])
    roc_score = roc_auc_score(y_test, clf_GDBT.predict_proba(X_test)[:,1])
    print(roc_score_forest)
    print(roc_score)
    #test_proba = clf_RF.predict_proba(X_test)[:, 1]
    preds = pd.Series(data=clf_GDBT.predict_proba(df_test)[:,1], index=df_test.index)

    return preds# Your Answer Here
#new_connections_predictions()
```

```
0.909935316911
0.911898080959
```

```
Out[120]: (0, 9)          0.066847
          (0, 19)         0.074851
          (0, 20)         0.123484
          (0, 35)         0.065019
          (0, 38)         0.063241
```

(0, 42)	0.292358
(0, 43)	0.063405
(0, 50)	0.063207
(0, 53)	0.187755
(0, 54)	0.072581
(0, 62)	0.169726
(0, 63)	0.074851
(0, 69)	0.066242
(0, 72)	0.062611
(0, 83)	0.099173
(0, 90)	0.066242
(0, 91)	0.090698
(0, 95)	0.089859
(0, 100)	0.066242
(0, 106)	0.322864
(0, 108)	0.065331
(0, 109)	0.062642
(0, 110)	0.062611
(0, 118)	0.063241
(0, 122)	0.066242
(0, 131)	0.072581
(0, 133)	0.184411
(0, 140)	0.077779
(0, 149)	0.094359
(0, 151)	0.063241
	...
(988, 989)	0.062673
(988, 996)	0.062673
(988, 997)	0.063548
(988, 1000)	0.062835
(988, 1002)	0.062673
(989, 994)	0.062673
(989, 996)	0.062673
(989, 1003)	0.062673
(989, 1004)	0.062673
(990, 994)	0.062642
(990, 998)	0.063548
(991, 994)	0.062673
(991, 999)	0.062673
(991, 1003)	0.062673
(992, 994)	0.062673
(992, 995)	0.062673
(992, 997)	0.062673
(992, 1000)	0.062642
(993, 1000)	0.062673
(994, 996)	0.064112
(995, 998)	0.064112
(995, 1000)	0.062673

```
(996, 997)      0.062673
(997, 998)      0.062673
(997, 1004)     0.062673
(998, 999)      0.064112
(1000, 1002)    0.062673
(1000, 1003)    0.062673
(1000, 1004)    0.062673
(1001, 1002)    0.062673
Length: 122112, dtype: float64
```

In [ ]:

In [ ]: