Assignment 4

October 14, 2020

---

*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 [1]: 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 [2]: P1_Graphs = pickle.load(open('A4_graphs','rb'))
        P1_Graphs
```

```
Out[2]: [<networkx.classes.graph.Graph at 0x7f73c00c1518>,
         <networkx.classes.graph.Graph at 0x7f738f2e19b0>,
         <networkx.classes.graph.Graph at 0x7f738f2e19e8>,
         <networkx.classes.graph.Graph at 0x7f738f2e1a20>,
         <networkx.classes.graph.Graph at 0x7f738f2e1a58>]
```

`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 [279]:
```
def graph_identification():

    from operator import itemgetter

    characteristics = [(nx.average_shortest_path_length(x), nx.average_clustering(x))

    networks = pd.DataFrame(characteristics, columns = ['average_sp_length', 'average_

    degrees = [x.degree() for x in P1_Graphs]

    degree_values = [sorted(set(x.values())) for x in degrees]

    N1_degree_distribution = [list(degrees[0].values()).count(i)/len(degrees[0]) for i
    N2_degree_distribution = [list(degrees[1].values()).count(i)/len(degrees[1]) for i
    N3_degree_distribution = [list(degrees[2].values()).count(i)/len(degrees[2]) for i
    N4_degree_distribution = [list(degrees[3].values()).count(i)/len(degrees[3]) for i
    N5_degree_distribution = [list(degrees[4].values()).count(i)/len(degrees[4]) for i

    degree_distributions = pd.DataFrame(columns = ['N1','N2','N3','N4','N5'])
    degree_distributions['N1'] = pd.Series(N1_degree_distribution)
    degree_distributions['N2'] = pd.Series(N2_degree_distribution)
    degree_distributions['N3'] = pd.Series(N3_degree_distribution)
    degree_distributions['N4'] = pd.Series(N4_degree_distribution)
    degree_distributions['N5'] = pd.Series(N5_degree_distribution)

    skew = degree_distributions.aggregate('skew',axis=0).to_frame().rename(columns = {

    n = networks.append(skew)

    labels = []

    for i,c in enumerate(n.columns):
        if n.loc['average_clustering',c] > 0.1:
            labels.append('SW_L')
        else:
            if n.loc['skew',c] > 3:
                labels.append('PA')
            else:
                labels.append('SW_H')

    return labels

graph_identification()
```

Out[279]:
```
['PA', 'SW_L', 'SW_L', 'PA', 'SW_H']
```

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 [2]: 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 [20]: def salary_predictions():

             from sklearn.model_selection import train_test_split
             from sklearn.metrics import roc_auc_score
             from sklearn.ensemble import GradientBoostingClassifier
             from sklearn.model_selection import GridSearchCV

             nodes = G.nodes(data=True)

             degree_centrality = nx.degree_centrality(G)
             clustering = nx.clustering(G)
             betweenness = nx.betweenness_centrality(G, endpoints = False)
             closeness = nx.closeness_centrality(G)

             features = pd.DataFrame(columns = ['degree','clustering','betweenness','closeness']

             for i,x in enumerate([degree_centrality,clustering,betweenness,closeness]):
                 features.iloc[:,i] = pd.Series(pd.Series(list(x.values())))


             G_features = pd.DataFrame([(x[1].get('Department'),x[1].get('ManagementSalary')) fo


             G_features_vectorized = pd.get_dummies(G_features, columns = ['Department']).fillna

             node_features_vectorized = G_features_vectorized.merge(features, how = 'inner', rig

             test = node_features_vectorized.where(node_features_vectorized['ManagementSalary']

             X = node_features_vectorized.where(node_features_vectorized['ManagementSalary'] !=

             y = np.array(node_features_vectorized.where(node_features_vectorized['ManagementSal
                 node_features_vectorized.columns.difference(['ManagementSalary']),axis=1)).resh

             GB = GradientBoostingClassifier()

             param_grid = {'learning_rate':[0.03,0.05,0.07,0.1,0.2], 'n_estimators':[60,80,100,1

             clf = GridSearchCV(GB,param_grid, scoring = 'roc_auc', cv = 5)

             clf.fit(X, y)

             y_predict = clf.predict_proba(test)[:,1]

             test_nodes = [x[0] for x in nodes if x[1].get('ManagementSalary') != 0.0 and x[1].g
```

```
    return pd.Series(y_predict, index = test_nodes)

salary_predictions()
```

Out[20]:
```
1      0.044951
2      0.947928
5      0.947928
8      0.168053
14     0.078300
18     0.073646
27     0.076550
30     0.488017
31     0.171554
34     0.034202
37     0.031048
40     0.044951
45     0.043897
54     0.171554
55     0.443125
60     0.102756
62     0.947928
65     0.947928
77     0.081183
79     0.071303
97     0.032407
101    0.008247
103    0.466788
108    0.069507
113    0.199790
122    0.008247
141    0.660722
142    0.947928
144    0.032407
145    0.488017
         ...
913    0.031048
914    0.031048
915    0.006523
918    0.046298
923    0.036759
926    0.046298
931    0.031048
934    0.006523
939    0.006523
944    0.006523
945    0.036759
947    0.071080
950    0.071970
```

```
951     0.027016
953     0.008247
959     0.006523
962     0.006523
963     0.150329
968     0.071080
969     0.071080
974     0.041237
984     0.006523
987     0.041237
989     0.071080
991     0.071080
992     0.006523
994     0.006523
996     0.006523
1000    0.027016
1001    0.041237
Length: 252, dtype: float64
```

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 [15]: future_connections = pd.read_csv('Future_Connections.csv', index_col=0, converters={0:
         future_connections
```

```
Out[15]:             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
         (228, 715)                0.0
         (397, 488)                0.0
         (253, 570)                0.0
         (435, 791)                0.0
         (711, 737)                0.0
         (263, 884)                0.0
         (342, 473)                1.0
```

```
(523, 941)          0.0
(157, 189)          0.0
(542, 757)          0.0
(731, 870)          0.0
(497, 973)          0.0
(93, 398)           0.0
(102, 604)          0.0
(206, 303)          1.0
(57, 447)           0.0
(417, 758)          0.0
(834, 837)          0.0
(261, 557)          0.0
(514, 740)          0.0
...                 ...
(672, 848)          NaN
(28, 127)           NaN
(202, 661)          NaN
(54, 195)           NaN
(295, 864)          NaN
(814, 936)          NaN
(839, 874)          NaN
(139, 843)          NaN
(461, 544)          NaN
(68, 487)           NaN
(622, 932)          NaN
(504, 936)          NaN
(479, 528)          NaN
(186, 670)          NaN
(90, 395)           NaN
(329, 521)          NaN
(127, 218)          NaN
(463, 993)          NaN
(123, 142)          NaN
(764, 885)          NaN
(144, 824)          NaN
(742, 985)          NaN
(506, 684)          NaN
(505, 916)          NaN
(149, 214)          NaN
(165, 923)          NaN
(673, 755)          NaN
(939, 940)          NaN
(555, 905)          NaN
(75, 101)           NaN

[488446 rows x 1 columns]
```

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 [54]: def new_connections_predictions():

             from sklearn.model_selection import train_test_split
             from sklearn.metrics import roc_auc_score
             from sklearn.preprocessing import MinMaxScaler
             from sklearn.ensemble import GradientBoostingClassifier
             from sklearn.linear_model import LogisticRegression
             from sklearn.model_selection import GridSearchCV

             jaccard_coefficient = nx.jaccard_coefficient(G)
             resource_allocation_index = nx.resource_allocation_index(G)
             preferential_attachment = nx.preferential_attachment(G)
             cn_soundarajan_hopcroft = nx.cn_soundarajan_hopcroft(G, community = 'Department')
             ra_index_soundarajan_hopcroft = nx.ra_index_soundarajan_hopcroft(G,  community = 'D

             X = pd.DataFrame(columns = ['jacc_coeff', 'ra_index', 'pref_attach', 'cn_sound_hop'

             X['jacc_coeff'] = pd.Series(jaccard_coefficient)
             X['ra_index'] = pd.Series(resource_allocation_index)
             X['pref_attach'] = pd.Series(preferential_attachment)
```

```
        X['cn_sound_hop'] = pd.Series(cn_soundarajan_hopcroft)
        X['ra_index_sound_hop'] = pd.Series(ra_index_soundarajan_hopcroft)

        X['nodes'] = X['jacc_coeff'].apply(lambda x: (x[0], x[1]))
        X['jacc_coeff'] = X['jacc_coeff'].apply(lambda x: x[2])
        X['ra_index'] = X['ra_index'].apply(lambda x: x[2])
        X['pref_attach'] = X['pref_attach'].apply(lambda x: x[2])
        X['cn_sound_hop'] = X['cn_sound_hop'].apply(lambda x: x[2])
        X['ra_index_sound_hop'] = X['ra_index_sound_hop'].apply(lambda x: x[2])
        X.set_index('nodes', inplace = True)

        future_connections.fillna(-1,inplace = True)

        test = future_connections.where(future_connections['Future Connection'] == -1).drop


        test_scaled = MinMaxScaler().fit_transform(test)

        X = future_connections.where(future_connections['Future Connection'] != -1).dropna(

        X_scaled = MinMaxScaler().fit_transform(X)

        y = np.array(future_connections.where(future_connections['Future Connection'] != -1

        #X_train, X_test, y_train, y_test = train_test_split(X_scaled,y,random_state=0)

        clf = LogisticRegression().fit(X_scaled,y)

        y_predict = clf.predict_proba(test_scaled)[:,1]

        return pd.Series(y_predict, index = test.index)

    new_connections_predictions()
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A
  y = column_or_1d(y, warn=True)
```

```
Out[54]: (107, 348)    0.034935
         (542, 751)    0.011265
         (20, 426)     0.830287
         (50, 989)     0.011206
         (942, 986)    0.011142
         (324, 857)    0.011214
         (13, 710)     0.350547
         (19, 271)     0.336952
         (319, 878)    0.011185
         (659, 707)    0.011259
```

(49, 843)	0.011167
(208, 893)	0.011245
(377, 469)	0.014182
(405, 999)	0.020378
(129, 740)	0.019561
(292, 618)	0.031603
(239, 689)	0.011181
(359, 373)	0.012851
(53, 523)	0.181762
(276, 984)	0.011170
(202, 997)	0.011157
(604, 619)	0.179275
(270, 911)	0.011182
(261, 481)	0.132012
(200, 450)	0.999979
(213, 634)	0.011284
(644, 735)	0.108502
(346, 553)	0.011436
(521, 738)	0.011740
(422, 953)	0.023941
	...
(672, 848)	0.011182
(28, 127)	0.978953
(202, 661)	0.011462
(54, 195)	0.999996
(295, 864)	0.011223
(814, 936)	0.011275
(839, 874)	0.011142
(139, 843)	0.011218
(461, 544)	0.011961
(68, 487)	0.012001
(622, 932)	0.011237
(504, 936)	0.023184
(479, 528)	0.011210
(186, 670)	0.011199
(90, 395)	0.247998
(329, 521)	0.039135
(127, 218)	0.358809
(463, 993)	0.011139
(123, 142)	0.928722
(764, 885)	0.011182
(144, 824)	0.011151
(742, 985)	0.011141
(506, 684)	0.011265
(505, 916)	0.011149
(149, 214)	0.999993
(165, 923)	0.011889
(673, 755)	0.011139

```
(939, 940)    0.011142
(555, 905)    0.011301
(75, 101)     0.024556
Length: 122112, dtype: float64
```

```
In [ ]:
```