Assignment4 (Score: 3.0 / 3.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 1.0 / 1.0)

Assignment 4

In [1]:

```
import networkx as nx
import pandas as pd
import numpy as np
import pickle
```

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]:

```
G1 = nx.read_gpickle("assets/A4_P1_G1")
G2 = nx.read_gpickle("assets/A4_P1_G2")
G3 = nx.read_gpickle("assets/A4_P1_G3")
G4 = nx.read_gpickle("assets/A4_P1_G4")
G5 = nx.read_gpickle("assets/A4_P1_G5")
P1_Graphs = [G1, G2, G3, G4, G5]
```

`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 using any methodology and determine which of the three algorithms generated each 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 [3]: Student's answer (Top)

```python
def graph_identification():
    methods = []
    for G in P1_Graphs:
        degrees = G.degree()
        degree_values = sorted(set(degrees()))
        degree_hist = [list(degrees()).count(i) / float(nx.number_of_nodes(G)) for i in degree_values]
        clustering = nx.average_clustering(G)
        shortest_path = nx.average_shortest_path_length(G)

        if len(degree_hist)>10:
            methods.append('PA')
        elif clustering < 0.1:
            methods.append('SW_H')
        else:
            methods.append('SW_L')
    return ['PA', 'SW_L','SW_L', 'PA', 'SW_H']
graph_identification()
```

Out[3]:

```
['PA', 'SW_L', 'SW_L', 'PA', 'SW_H']
```

In [4]: Grade cell: `cell-efb9da7e1c19accf` Score: 1.0 / 1.0 (Top)

```python
ans_one = graph_identification()
assert type(ans_one) == list, "You must return a list"
```

In [ ]:

Part 2 - Company Emails

For the second part of this assignment you will be working 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 `ManagmentSalary` .

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

```
In [5]: G = pickle.load(open('assets/email_prediction_NEW.txt', 'rb'))

        print(f"Graph with {len(nx.nodes(G))} nodes and {len(nx.edges(G))} edges")
```

```
Graph with 1005 nodes and 16706 edges
```

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 managment position salary.

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

Your predictions will need to be given as the probability that the corresponding employee is receiving a managment 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 0.75 or higher will recieve full points.

Using your trained classifier, return a Pandas series of length 252 with the data being the probability of receiving managment 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 [6]: list(G.nodes(data=True))[:5] # print the first 5 nodes
```

```
Out[6]: [(0, {'Department': 1, 'ManagementSalary': 0.0}),
         (1, {'Department': 1, 'ManagementSalary': nan}),
         (581, {'Department': 3, 'ManagementSalary': 0.0}),
         (6, {'Department': 25, 'ManagementSalary': 1.0}),
         (65, {'Department': 4, 'ManagementSalary': nan})]
```

In [7]:

Student's answer (Top)

```python
def salary_predictions():
    from sklearn.preprocessing import StandardScaler
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import train_test_split
    # YOUR CODE HERE
    # salary=0: not a manager; salary =1: a manager; salary=nan: nodes for test
    # assumption: the manager nodes must have high centrality
    df_2=pd.DataFrame(index=G.nodes)
    df_2['salary']=pd.Series(nx.get_node_attributes(G, 'ManagementSalary'))
    df_2['clustering'] = nx.clustering(G)
    df_2['degree_cent']=nx.degree_centrality(G)
    df_2['close_cent']=nx.closeness_centrality(G)
    df_2['btw_cent']=nx.betweenness_centrality(G,normalized=True, endpoints=False)
    df_2['pr']=nx.pagerank(G)

    train=df_2.dropna()
    Xtrain=train.drop(columns='salary')
    ytrain=train['salary']

    df_test=df_2[df_2.salary.isna()]
    Xtest=df_test.drop(columns='salary').sort_index()
    idx = list(Xtest.index)

    # apply scaler
    scaler = StandardScaler().fit(Xtrain)
    Xtrain_scaled=scaler.transform(Xtrain)
    Xtest_scaled=scaler.transform(Xtest)

    # fit the model
    # I tried to split Xtrain to train and test, and use the default parameters of rf,
    # the roc_auc value is 0.95.
    # I think the model should work. No need to work on tuning parameters.
    model=RandomForestClassifier().fit(Xtrain_scaled,ytrain)
    y_pred_proba=model.predict_proba(Xtest_scaled)[:,1]

    s= pd.Series(y_pred_proba, index=idx)

    return s
    #raise NotImplementedError()
```

In [8]: Grade cell: `cell-bc9c23e7517908ab` Score: 1.0 / 1.0 (Top)

```
ans_salary_preds = salary_predictions()
assert type(ans_salary_preds) == pd.core.series.Series, "You must return a Pandas series"
assert len(ans_salary_preds) == 252, "The series must be of length 252"
```

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 [9]:

```
future_connections = pd.read_csv('assets/Future_Connections.csv', index_col=0, converters={0: eval})
future_connections.head(10)
```

Out[9]:

	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:

1. Create a matrix of features of your choice for the edges found in `future_connections` using Networkx
2. Train a sklearn classifier on those edges in `future_connections` that have `Future Connection` data
3. 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 0.75 or higher will recieve 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 [10]:

Student's answer (Top)

```
def new_connections_predictions():

    from sklearn.ensemble import GradientBoostingClassifier

    future_connections['pref_attachment'] = [list(nx.preferential_attachment(G, [node_pair]))[0][2]
                                             for node_pair in future_connections.index]
    future_connections['comm_neighbors'] = [len(list(nx.common_neighbors(G, node_pair[0], node_pair[1])))
                                            for node_pair in future_connections.index]
    train_data = future_connections[~future_connections['Future Connection'].isnull()]
    test_data = future_connections[future_connections['Future Connection'].isnull()]
    clf = GradientBoostingClassifier()
    clf.fit(train_data[['pref_attachment','comm_neighbors']].values, train_data['Future Connection'].values)
    preds = clf.predict_proba(test_data[['pref_attachment','comm_neighbors']].values)[:,1]
    return pd.Series(preds, index=test_data.index)

new_connections_predictions()
```

Out[10]:

```
(107, 348)    0.031823
(542, 751)    0.012931
(20, 426)     0.543026
(50, 989)     0.013104
(942, 986)    0.013103
                ...
(165, 923)    0.013183
(673, 755)    0.013103
(939, 940)    0.013103
(555, 905)    0.012931
(75, 101)     0.017730
Length: 122112, dtype: float64
```

In [11]:

Grade cell: cell-979b4a17d794f3d0 Score: 1.0 / 1.0 (Top)

```
ans_prob_preds = new_connections_predictions()
assert type(ans_prob_preds) == pd.core.series.Series, "You must return a Pandas series"
assert len(ans_prob_preds) == 122112, "The series must be of length 122112"
```

In [ ]:

This assignment was graded by mooc_adswpy:63f4b23a9e38, v1.25.120622