Assignment 2

Last updated: June 23, 2022

Name: Adam Bakopolus

Uniqname: abakop

Instructions

Please turn in:

1. A Jupyter Notebook file. This file should show all of the required work, including code, results, visualizations (if any), and necessary comments to your code. Irrelevant code and results should be deleted prior to submission. This file is submitted automatically when you submit your notebook to be autograded. This is done in Assignment 2 -- Create.
2. An HTML file of the Notebook. Submit this file in Assignment 2 - Submit.
3. A PDF file of the Notebook. Submit this file in Assignment 2 - Submit.

Before submitting, please select Kernel -> Restart & Run All.

Please do not remove any code outside of the Not Implemented Error sections. The autograder may need it.

Assignment 2

In [1]:
```
import networkx as nx
import pandas as pd
import numpy as np

import urllib
import random
import json
import operator
import random

import matplotlib.pyplot as plt
#import seaborn as sns
```

Dataset description

In this assignment, you will explore several network generative models to understand how varying model parameters affects network structure. Our exploration will be based on a real-world dataset:

Astrophysics collaborations (http://www-personal.umich.edu/~mejn/netdata/astro-ph.zip): a network of coauthorships between scientists posting preprints on the Astrophysics E-Print Archive between Jan 1, 1995 and December 31, 1999. An edge exists between two researchers if they have co-authored at least one paper.

Newman, M. E. (2001). The structure of scientific collaboration networks. Proceedings of the national academy of sciences, 98(2), 404-409.

Let's load the graph from `assets/astro_phy.gml` .

In [2]:
```
G = nx.read_gml('assets/astro_phy.gml')  # load the graph to G
```

`plot_deg_distribution(G, z, log)` is a helper function that can plot the degree distribution of graph `G` using the NetworkX function `degree_histogram` . `z` is a zoom-in parameter. If $z \geq 0$ is specified, a zoom-in plot will be presented and show only the first z values in the node distribution array. If no z is specified, the plot of the complete node distribution will be shown. `log` is a flag indicating whether to plot the node distribution in log-log scale. It is defaulted to `False` .

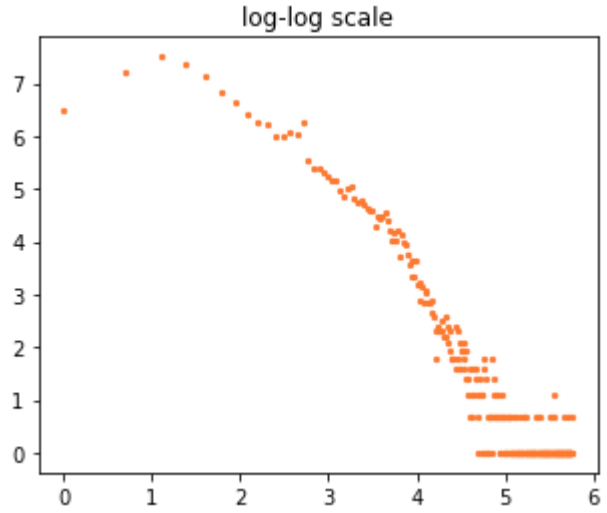
In [3]:
```
class fig_wrapper:
    def __init__(self):
        self.img = None
        self.log = False
        self.x = []
        self.y = []

def plot_deg_distribution(G, z=-1, log=False):
    wrapper = fig_wrapper()
    freq = nx.degree_histogram(G);
    x = [i for i in range(len(freq))]
    if log:
        wrapper.log = True
        freq = [np.log(f+1) for f in freq]
        x = [np.log(i+1) for i in x]
        fig, ax = plt.subplots(figsize = (5, 4))
        ax.scatter(x=x, y=freq, color='#fc7930',s=5);
        ax.set_title("log-log scale")
        wrapper.img = ax
        wrapper.x = x
        wrapper.y = freq
        return wrapper
    if z < 0:
        fig, ax = plt.subplots(figsize = (5, 4))
        ax.scatter(x=x, y=freq, color='#fc7930',s=5);
        ax.set_title("linear scale")
    else:
        fig, ax = plt.subplots(figsize = (5, 4))
        ax.scatter(x=x[:20], y=freq[:20], color='#fc7930',s=5);  # zoom in
        ax.set_title("zoom-in")
    wrapper.img = ax
    wrapper.x = x
    wrapper.y = freq
    print(wrapper)
    return wrapper
```

Part 1. Modeling the real astrophysics collaboration network

Q1. Degree distribution of the astrophysics collaboration network

1.1 (1 point, Autograded) Plot the node degree distribution of the astrophysics collaboration network in log-log scale using the function `plot_deg_distribution(G, z, log)` provided above.

In [4]:
```
ans = plot_deg_distribution(G, -1, True)  # assign the return value of plot_deg_distribution(G, z, log) here
```



In [5]:
```
#hidden tests for Question 1.1 are within this cell
```

1.2 (1 point, Autograded) Does the distribution look more like a power law distribution or a normal distribution?

Indicate the distribution by assigning either of the two strings to `distribution_more_like` :

- `"power law"`
- `"normal"`

In [6]:
```
distribution_more_like = "power law"  # a string with the value "power law" or "normal"
```

In [7]:
```
#hidden tests for Question 1.2 are within this cell
```

Q2. (2 points, Autograded) What is the transitivity and average clustering coefficient of this network?

In [8]:
```
transitivity = nx.transitivity(G)        # transitivity value. This should be a float.
average_clustering = nx.average_clustering(G)  # average clustering coefficient. This should be a float.
```

In [9]:
```
#hidden tests for Question 2 are within this cell
```

Q3. (3 points, Autograded) What is the maximum, minimum, mean, and standard deviation of the PageRank values of the nodes in G?

In [10]:
```
page = nx.pagerank(G)
page_sorted = sorted(page.items(), key=lambda item: item[1])

max_pgv = page_sorted[-1][1]  # maximum PageRank value. This should be a float.
min_pgv = page_sorted[0][1]  # minimum PageRank value. This should be a float.

total = 0
for tup in page_sorted:
    total += tup[1]

mean_pgv = total / len(page_sorted) # mean of PageRank value. This should be a float.

std_list = []
for tup in page_sorted:
    std_list.append(tup[1])

std_pgv = np.std(std_list) # standard deviation of PageRank value.
```

In [11]:
```
#hidden tests for Question 3 are within this cell
```

Now that we have obtained some basic properties of the structure of network G, you will compare them with the properties of random networks with around the same number of nodes and edges using the various models we covered in lecture: the Erdős-Rényi model, the preferential attachment model, and a small world model.

Erdős-Rényi Graph

The NetworkX function `erdos_renyi_graph` with parameters n and p generates an Erdős-Rényi Graph with `n` nodes and where each pair of nodes is connected by an edge with probability `p`. If the parameter directed is set to `True`, the function generates a directed graph. Since in this case we want an undirected graph, you should set the parameter directed to `False`.

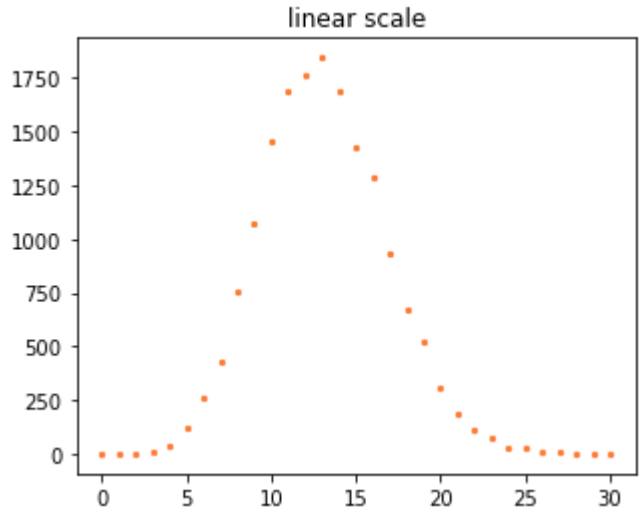
An alternative function, `fast_gnp_random_graph` produces the same graph and has the same parameters as `erdos_renyi_graph`. However, `fast_gnp_random_graph` is faster when the expected number of edges is small (i.e. sparse graph).

Q4. (4 points, Autograded) Use the function `fast_gnp_random_graph` to create a random graph G_1, such that G_1 has the same number of nodes as G and the expected number of edges of G_1 is the same as the number of edges in G. Plot the node degree distribution of G_1.

Hint: to pass the autograder, please make sure your plot is in linear scale.

In [12]:
```
random.seed(0)   # sync the autograder
n = G.number_of_nodes()  # number of nodes. This should be an int.
m = G.number_of_edges()  # number of edges. This should be an int.

p1 = G.number_of_edges()
p2 = (G.number_of_nodes() * (G.number_of_nodes() - 1)) / 2
p = p1/p2  # probability of edges between nodes. This should be a float in [0,1].

G1 = nx.fast_gnp_random_graph(n, p, seed = 0) # assign your networkx graph here.
ans = plot_deg_distribution(G1, -1)  # assign your return value of plot_deg_distribution(G1, z, log) here.
```

```
<__main__.fig_wrapper object at 0x7f4ec193f340>
```



In [13]:
```
#hidden tests for Question 4 are within this cell
```

Q5. (2 points, Autograded) What is the transitivity and average clustering coefficient of G_1?

In [14]:
```
transitivity = nx.transitivity(G1)        # transitivity value. This should be a float.
average_clustering = nx.average_clustering(G1)  # average clustering coefficient. This should be a float.
```

In [15]:
```
#hidden tests for Question 5 are within this cell
```

Q6. (3 points, Autograded) What is the maximum, minimum, mean, and standard deviation of the

PageRank values of the nodes in G_1?

In [16]:
```
page = nx.pagerank(G1)
page_sorted = sorted(page.items(), key=lambda item: item[1])

max_pgv = page_sorted[-1][1]  # maximum PageRank value. This should be a float.
min_pgv = page_sorted[0][1]  # minimum PageRank value. This should be a float.

total = 0
for tup in page_sorted:
    total += tup[1]

mean_pgv = total / len(page_sorted) # mean of PageRank value. This should be a float.

std_list = []
for tup in page_sorted:
    std_list.append(tup[1])

std_pgv = np.std(std_list) # standard deviation of PageRank value.
```

In [17]:
```
#hidden tests for Question 6 are within this cell
```

Preferential attachment model

The function `barabasi_albert_graph(n, avg_m)` generates a random graph using the preferential attachment model with `n` nodes and where each arriving node attaches to `m` existing nodes.

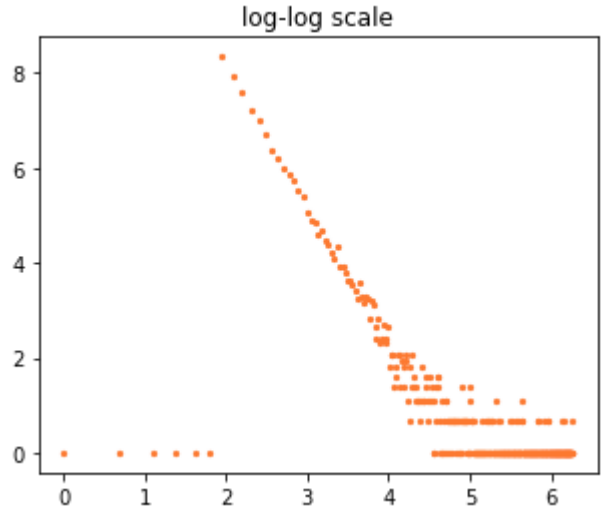
Q7. (4 points, Autograded) Use the preferential attachment model to create a random graph G_2, such that G_2 has around the same number of nodes and edges as G. Plot the node degree distribution of G_2 in log-log scale.

Hint:

1. if you cannot find an integer value for the number of edges added by each node, just round down to an integer value.
2. In the function `barabasi_albert_graph(n, avg_m)` ,
    - `n` is the number of nodes

- `avg_m` is the number of edges to attach from *a new node* to *existing nodes*. Therefore, the total number of edges created by this function is around `n * avg_m`. We already know that G_1 has `m` edges. Therefore, using the variables we can calculate `avg_m`.

In [18]:
```
random.seed(0)  # sync for the autograder

n = G1.number_of_nodes()
avg_m = int(np.floor(G1.number_of_edges()/G1.number_of_nodes())) # average number of edges increased by each ada
G2 = nx.barabasi_albert_graph(n, avg_m) # assign your networkx graph here
ans = plot_deg_distribution(G2, -1, True) # assign your return value of plot_deg_distribution(G, z, log) here
```



In [19]:
```
#hidden tests for Question 7 are within this cell
```

Q8. (2 points, Autograded) What is the transitivity and average clustering coefficient of G_2?

In [20]:
```
transitivity = nx.transitivity(G2)        # transitivity value. This should be a float.
average_clustering = nx.average_clustering(G2)  # average clustering coefficient. This should be a float.
```

In [21]:
```
#hidden tests for Question 8 are within this cell
```

Q9. (3 points, Autograded) What is the maximum, minimum, mean, and standard deviation of the

PageRank values of the nodes in G_2?

In [22]:
```
page = nx.pagerank(G2)
page_sorted = sorted(page.items(), key=lambda item: item[1])

max_pgv = page_sorted[-1][1]  # maximum PageRank value. This should be a float.
min_pgv = page_sorted[0][1]  # minimum PageRank value. This should be a float.

total = 0
for tup in page_sorted:
    total += tup[1]

mean_pgv = total / len(page_sorted) # mean of PageRank value. This should be a float.

std_list = []
for tup in page_sorted:
    std_list.append(tup[1])

std_pgv = np.std(std_list) # standard deviation of PageRank value.
```

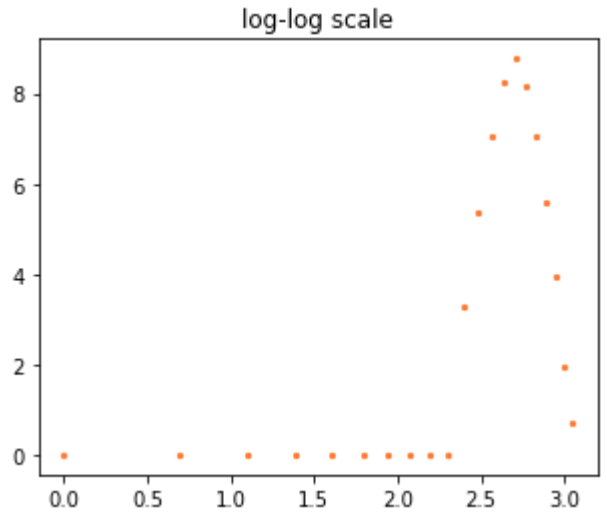
In [23]:
```
#hidden tests for Question 9 are within this cell
```

Small world networks

The function `nx.watts_strogatz_graph(n,k,p)` generates a small world network with `n` nodes. Initially, nodes are connected to their `k` nearest neighbors in a ring topology. Then each edge is rewired with probability `p`.

Q10. (4 points, Autograded) Create a small world network G_3 with the same number of nodes as network G and with a number of edges that is as close as possible to the number of edges in network G. Plot the node degree distribution of G_3 in log-log scale. Set $p = 0.1$.

In [24]:
```
random.seed(0)  # sync for autograder
G3 = nx.watts_strogatz_graph(G.number_of_nodes(), 15, .1) # your networkx graph
ans = plot_deg_distribution(G3, -1, True) # assign your return value of plot_deg_distribution(G, z, log) here
```



In [25]:
```
#hidden tests for Question 10 are within this cell
```

Q11. (2 points, Autograded) What is the transitivity and average clustering coefficient of G_3?

In [26]:
```
transitivity = nx.transitivity(G3)        # transitivity value. This should be a float.
average_clustering = nx.average_clustering(G3)  # average clustering coefficient. This should be a float.
```

In [27]:
```
#hidden tests for Question 11 are within this cell
```

Q12. (3 points, Autograded) What is the maximum, minimum, mean, and standard deviation of the PageRank values of the nodes in G_3?

In [28]:
```
page = nx.pagerank(G3)
page_sorted = sorted(page.items(), key=lambda item: item[1])

max_pgv = page_sorted[-1][1]  # maximum PageRank value. This should be a float.
min_pgv = page_sorted[0][1]  # minimum PageRank value. This should be a float.

total = 0
for tup in page_sorted:
    total += tup[1]

mean_pgv = total / len(page_sorted) # mean of PageRank value. This should be a float.

std_list = []
for tup in page_sorted:
    std_list.append(tup[1])

std_pgv = np.std(std_list) # standard deviation of PageRank value.
```

In [29]:
```
#hidden tests for Question 12 are within this cell
```

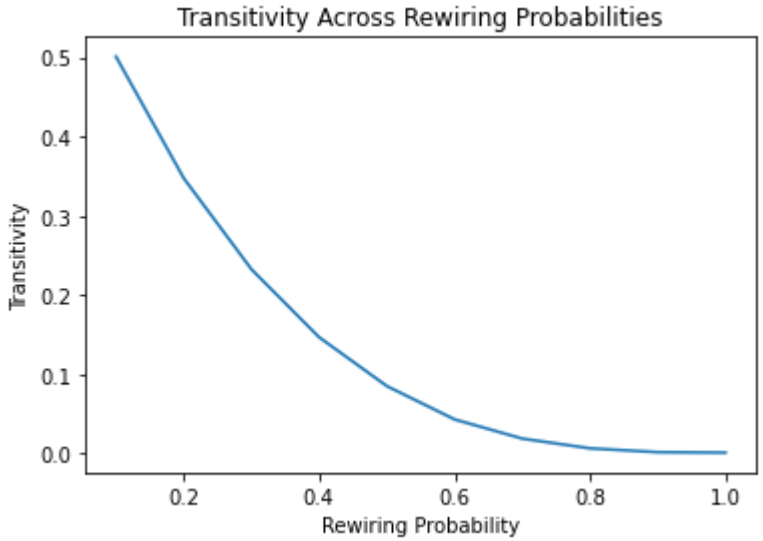
Q13. (3 points, Autograded) Using the same values of `n` and `k` as you did for network G_3, generate a small world network using every p in `np.linspace(0.1, 1, 10)` and compute the graph transitivity and average clustering coefficient. Plot a line graph for both attributes versus p.

In [30]:
```
tran = np.zeros(10)   # assign your graph transitivity values as a numpy array to this variable.
clus = np.zeros(10)   # assign your average clustering coefficient values as a numpy array to this variable.

p_vals = np.linspace(0.1, 1, 10)
for i in range(len(p_vals)):
    graph = nx.watts_strogatz_graph(G.number_of_nodes(), 15, p_vals[i])
    tran[i] = nx.transitivity(graph)
    clus[i] = nx.average_clustering(graph)
```

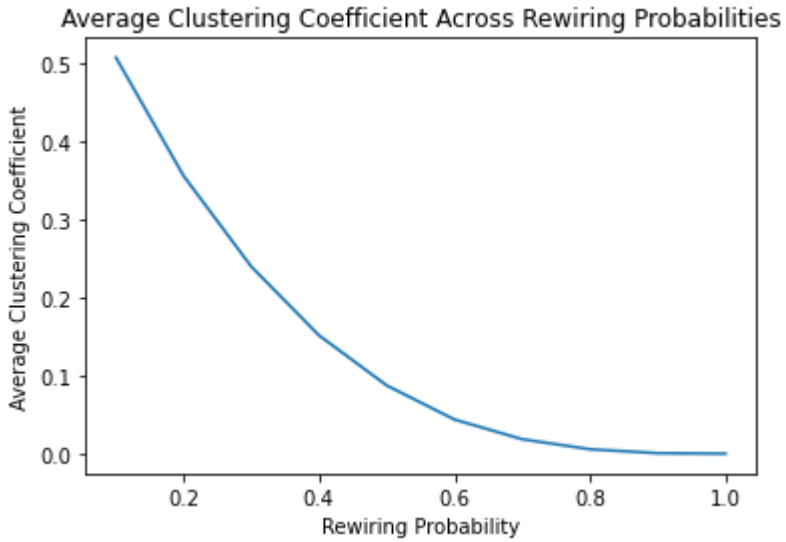
In [31]:
```
#hidden tests for Question 13 are within this cell
```

(2 points, Manually graded) Include the two plots here:

In [32]:
```
plt.plot(p_vals, tran)

plt.title('Transitivity Across Rewiring Probabilities')
plt.xlabel('Rewiring Probability')
plt.ylabel('Transitivity')
plt.show()
```



In [33]:
```
plt.plot(p_vals, clus)

plt.title('Average Clustering Coefficient Across Rewiring Probabilities')
plt.xlabel('Rewiring Probability')
plt.ylabel('Average Clustering Coefficient')
plt.show()
```



Q14. (2 points, Autograded) Compare the features of the simulated networks with the real network G and think of the following questions:

1. In terms of node degree distribution, which model should be most similar to the collaboration network?
2. In terms of transitivity and average clustering coefficient, which model should best approximate the collaboration network?

Indicate the solution of each question with one of the following strings:

- "Erdős-Rényi"
- "preferential attachment model"
- "small world network"

The questions are represented by `q1` and `q2`. For example, if you believe the answer to the first sub-question is Erdős-Rényi you should answer:

```
q1 = "Erdős-Rényi"
```

In [34]:
```
q1 = 'preferential attachment model' # This should be a string
q2 = 'small world network' # This should be a string
```

In [35]:
```
#hidden tests for Question 14 are within this cell
```

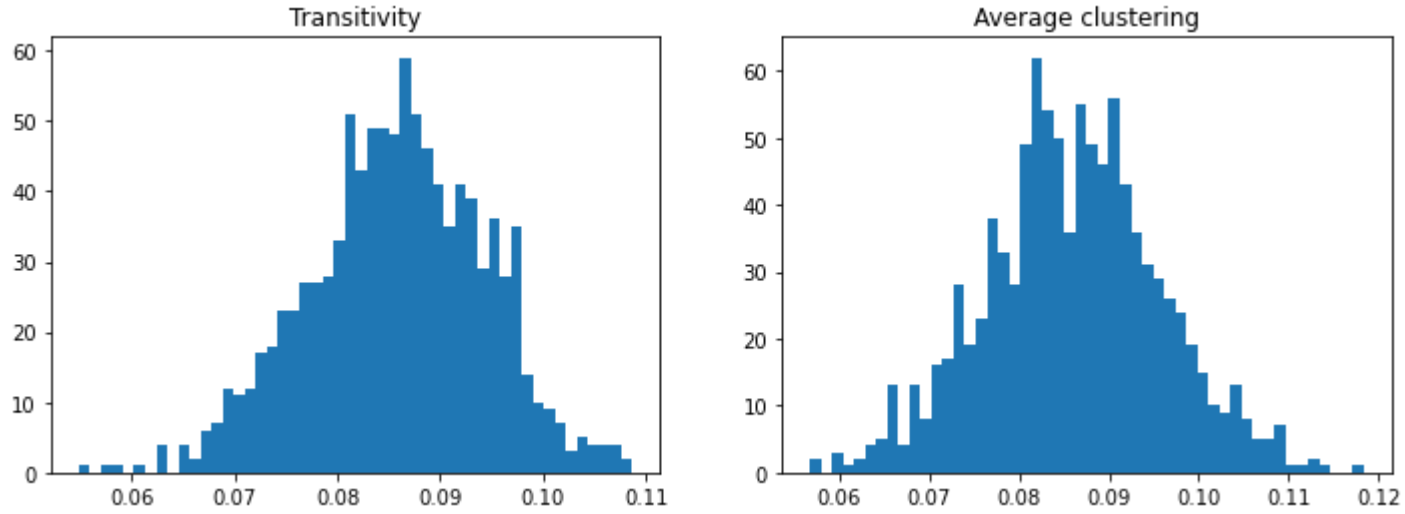
Part 2. Clustering of the generative models

In this part, you will explore the clustering coefficient and transitivity of the aforementioned generative models. Rather than making comparisons based on a single instance of each type of network, you will generate multiple random graphs and construct a distribution of the values. You will construct networks with relatively few nodes and edges to make the computation more efficient. The networks will have 100 nodes and around 425 edges.

Q15. (6 points, Autograded) Create 1000 random Erdős-Rényi graphs with `fast_gnp_random_graph` such that the networks have 100 nodes and are expected to have 425 edges. For each graph, compute its transitivity and average clustering coefficient. Plot a histogram for both attributes.

Note: for the plots, use the function `hist` and set the parameter `bins=50` .

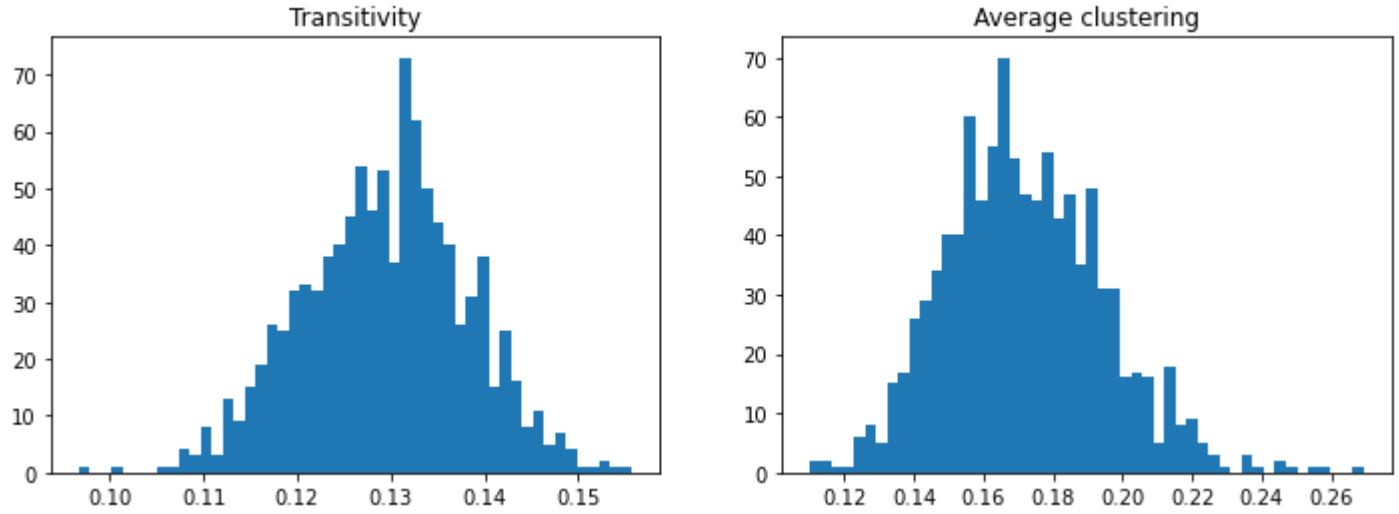
In [36]:
```
random.seed(0)
tran = np.zeros(1000)   # assign your graph transitivity values as a numpy array to this variable.
clus = np.zeros(1000)   # assign your average clustering coefficient values as a numpy array to this variable.

for i in range(1000):

    n = 100  # number of nodes. This should be an int.
    m = 425  # number of edges. This should be an int.

    p1 = m
    p2 = (100 * 99) / 2
    p = p1/p2  # probability of edges between nodes. This should be a float in [0,1].

    graph = nx.fast_gnp_random_graph(n, p) # assign your networkx graph here.
    tran[i] = nx.transitivity(graph)
    clus[i] = nx.average_clustering(graph)

fig, axes = plt.subplots(1,2,figsize = (12, 4))
axes[0].title.set_text("Transitivity");         # plot transitivity in subplot 0
axes[1].title.set_text("Average clustering");   # plot average clustering in subplot 1

axes[0].hist(tran, bins = 50)
axes[1].hist(clus, bins = 50)

plt.show()
```



In [37]:
```
#hidden tests for Question 15 are within this cell
```

Q16. (6 points, Autograded) Create 1000 preferential attachment graphs with 100 nodes and each arriving node connecting to 4 existing nodes. For each graph, compute its transitivity and average clustering coefficient. Plot a histogram for both attributes.

In [38]:
```
random.seed(0)
tran = np.zeros(1000)   # assign your graph transitivity values as a numpy array to this variable.
clus = np.zeros(1000)   # assign your average clustering coefficient values as a numpy array to this variable.

for i in range(1000):

    n = 100  # number of nodes. This should be an int.
    avg_m = 4  # average number of edges increased by each additional node

    graph = nx.barabasi_albert_graph(n, avg_m) # assign your networkx graph here
    tran[i] = nx.transitivity(graph)
    clus[i] = nx.average_clustering(graph)

fig, axes = plt.subplots(1,2,figsize = (12, 4))
axes[0].title.set_text("Transitivity");         # plot transitivity in subplot 0
axes[1].title.set_text("Average clustering");   # plot average clustering in subplot 1

axes[0].hist(tran, bins = 50)
axes[1].hist(clus, bins = 50)

plt.show()
```

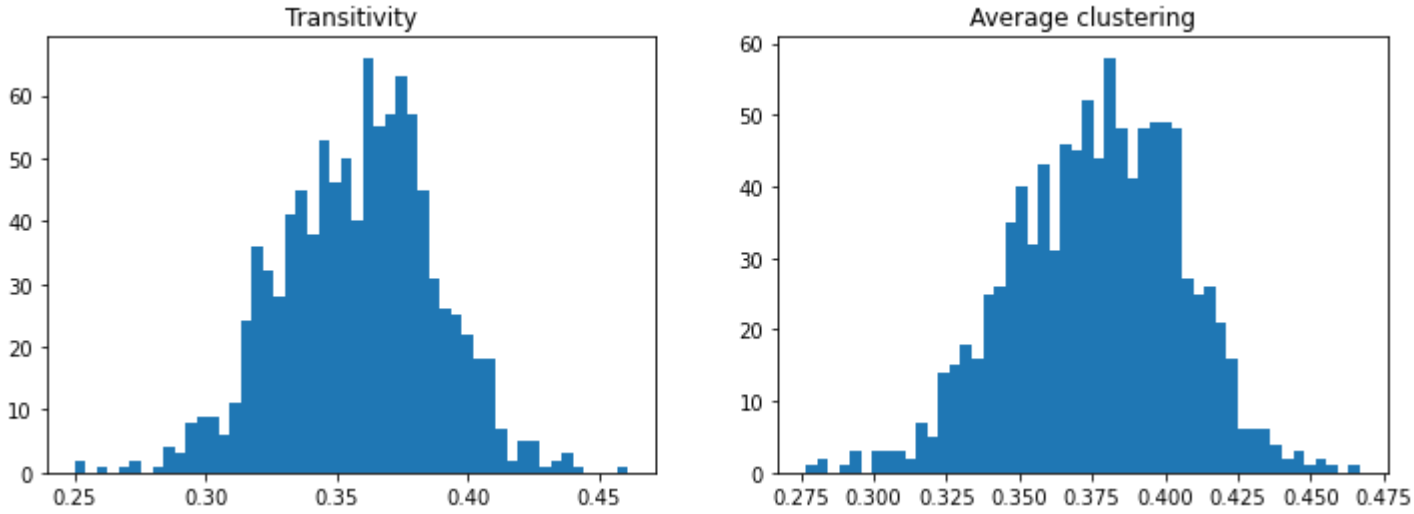


In [39]:
```
#hidden tests for Question 16 are within this cell
```

Q17. (6 points, Autograded) Create 1000 small world graphs with 100 nodes, $k = 4$, and rewiring

probability $p = 0.1$. For each graph, compute its transitivity and average clustering coefficient. Plot a histogram for both attributes.

In [40]:
```
random.seed(0)
tran = np.zeros(1000)   # assign your graph transitivity values as a numpy array to this variable.
clus = np.zeros(1000)   # assign your average clustering coefficient values as a numpy array to this variable.

for i in range(1000):

    n = 100  # number of nodes. This should be an int.
    m = 425  # number of edges. This should be an int.

    graph = nx.watts_strogatz_graph(n, 4, .1) # your networkx graph
    tran[i] = nx.transitivity(graph)
    clus[i] = nx.average_clustering(graph)

fig, axes = plt.subplots(1,2,figsize = (12, 4))
axes[0].title.set_text("Transitivity");          # plot transitivity in subplot 0
axes[1].title.set_text("Average clustering");    # plot average clustering in subplot 1

axes[0].hist(tran, bins = 50)
axes[1].hist(clus, bins = 50)

plt.show()
```



In [41]:
```
#hidden tests for Question 17 are within this cell
```

Q18. (10 points, Manually graded) Compare the distribution plots of clustering coefficient values across the different network generative models.

Which model gives the highest clustering coefficient values?

Which model gives the lowest clustering coefficient values?

Explain your results using the assumptions and mechanisms of the models.

Your explanation should be of the form:

Model X produces networks with the highest clustering coefficient because...

Model Y produces networks with the lowest clustering coefficient because...

Small World random graphs give the highest clustering coefficient values. Erdős-Rényi random graphs give the lowest clustering coefficient values.

The Erdős-Rényi random graphs will have the lowest clustering coefficient values. With these random graphs, edges are created between nodes A and B with probability p. There is nothing specified when creating these random graphs that would put a preference on triangle creation and closure. As a result, triads will be created but the probability alone of an edge forming between two nodes is what would dictate triangle creation. As a result, as n gets larger, the clustering coefficient will eventually go to zero.

On the other hand, Small World randoms graphs will have the highest clustering coefficient values. The clustering coefficient will be dependent on k (connected to the 4 nearest neighbors in this example) and p (probability of rewiring being .1). Most nodes will remain having a degree 4 due to most not being re-wired. As a result of most nodes being connected to their 4 nearest neighbors, the likelihood of closed triads would be high in these types of random graphs. Even when an edge is rewired, it likely will lead to other triad closures, leading to high clustering coefficient values.

Part 3. Link prediction

In this part, we are going to compare various link prediction features to predict if an edge exists in the astrophysics collaboration network. We will use the following features and their corresponding `NetworkX` functions (See [NetworkX link prediction documentation (https://networkx.github.io/documentation/networkx-1.9/reference/algorithms.link_prediction.html)](https://networkx.github.io/documentation/networkx-1.9/reference/algorithms.link_prediction.html)).

- Resource Allocation Index -- `resource_allocation_index(G[, ebunch])` .
- Jaccard Coefficient -- `jaccard_coefficient(G[, ebunch])` .
- Adamic-Adar Index -- `adamic_adar_index(G[, ebunch])` .
- Preferential Attachment Score -- `preferential_attachment(G[, ebunch])` .
- Number of Common Neighbors -- This feature will be provided in our data set.

The `assets/part3.csv` contains the set of pairs of nodes you will use for training and testing in the prediction task. Some of the pairs in `assets/part3.csv` are connected by an edge in the network and others are not. Our goal will be to predict which ones are connected using our link prediction features.

The file contains 4 columns (`v1` , `v2` , `num` , `label`). Columns `v1` and `v2` represent pairs of nodes (v1,v2), `num` is the number of common neighbors between v1 and v2, and `label` is 1 if the edge (v1,v2) exists in the network and 0 otherwise.

The astrophysics collaboration network `G` is fixed. That is, we will not make changes to the network. We will use the structure of the network to "predict" which pairs of nodes are connected (pretending we cannot simply check if the edge `(v1,v2)` exists in `G`). The intuition here is that we will pretend that these edges are not yet present in the network, and use the link prediction features to predict whether they will emerge.

Caveats:

Note that we are cheating slightly since we are using these edges to compute the link prediction features. Typically, if we are trying to predict edges that have not yet emerged, we cannot use them at all.

As you may see, there is approximately the same number of positive and negative labels in the test dataset. This is artificially selected to facilitate the prediction task. In reality, this is rarely the case since in link prediction positive samples are far less common than negative ones.

However, we will use this setup to illustrate the procedure.

Let's begin by importing the libraries we will need.

In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import f1_score
import warnings
warnings.filterwarnings('ignore')
```

Now let's load data from `part3.csv`, which contains the number of common neighbors and labels.

Q19. (5 points, Manually graded) Expand the dataframe to include the other features and print part of the resulting dataframe. This may take some time. Print the top 10 rows of the dataframe.

Recall that you should not change the network `G` in any way.

In [43]:

```
%%time
G = nx.read_gml('assets/astro_phy.gml')
link_pred = pd.read_csv('assets/part3.csv')
edges = list(link_pred[['v1', 'v2']].to_records(index = False))

value_list = []
predictions = nx.resource_allocation_index(G, edges)
for u, v, p in predictions:
    value_list.append(p)

link_pred['rai'] = value_list

value_list = []
predictions = nx.jaccard_coefficient(G, edges)
for u, v, p in predictions:
    value_list.append(p)

link_pred['jac'] = value_list

value_list = []
predictions = nx.adamic_adar_index(G, edges)
for u, v, p in predictions:
    value_list.append(p)

link_pred['ada'] = value_list

value_list = []
predictions = nx.preferential_attachment(G, edges)
for u, v, p in predictions:
    value_list.append(p)

link_pred['pa'] = value_list

print(link_pred.head(10))
```

```
             v1                v2  num  label       rai       jac       ada  \
0    BASKETT, L  SUTHERLAND, TMCW    9      0  0.096704  0.409091  1.983937
1   BROWNE, IWA        JACKSON, N   10      1  0.559679  0.222222  3.442149
2      MACRI, L         AJHAR, EA    8      0  0.144501  0.068966  1.965276
3    ALVAREZ, D         MILLER, K   13      1  0.596234  0.866667  4.158348
4       ABAD, C      HONEYCUTT, K   40      1  0.719897  0.727273  9.872832
5  TILANUS, RPJ       MCMAHON, RG    5      0  0.109267  0.023364  1.301187
6    WILKES, BJ         PETERS, J   20      1  0.414744  0.210526  5.089013
```

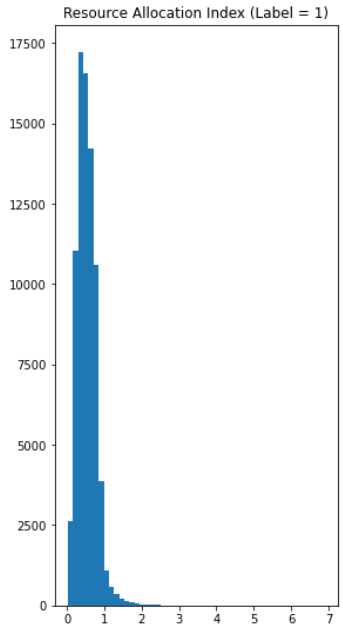
```
7    BONALDI, M         SALEMI, F   35      1  0.847962  0.714286  9.399725
8   TUTHILL, PG      NISHIMOTO, D    8      1  0.656226  0.500000  3.148903
9    MORALES, J         FARACH, H   15      1  0.531780  0.326087  4.464370

     pa
0   240
1   516
2  3700
3   196
4  2256
5  8568
6  2844
7  1764
8   135
9   840
CPU times: user 51.8 s, sys: 109 ms, total: 51.9 s
Wall time: 52 s
```

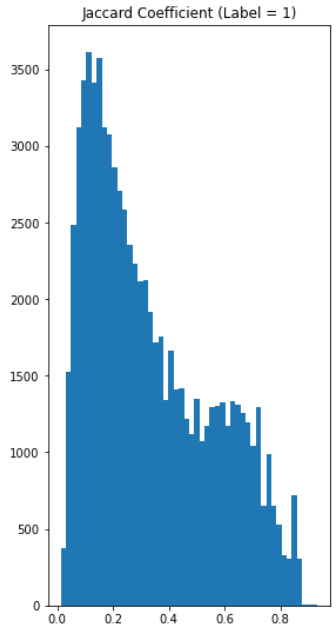
Q20. (4 points, Manually graded) Plot histograms of the values of each of the features (num, rai, jac, ada, pa); one set of histograms will be for pairs of nodes with label = 1 and another set for pairs of nodes with label = 0.

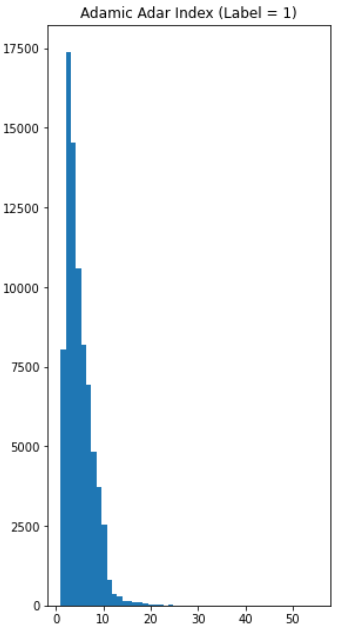
There will be a total of 10 histograms -- 2 per feature.

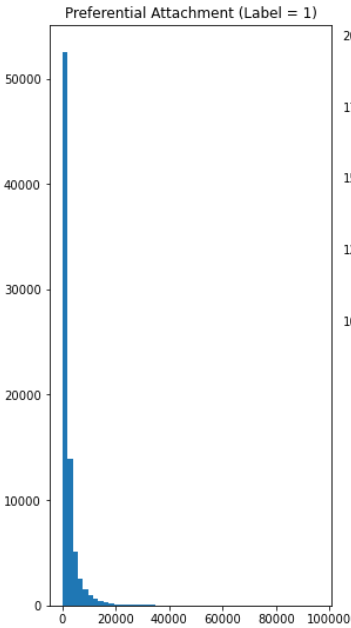
In [44]:

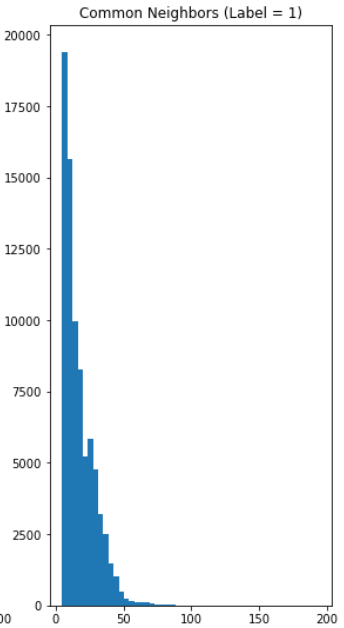
```
label0 = link_pred[link_pred['label'] == 0]
label1 = link_pred[link_pred['label'] == 1]

fig, axes = plt.subplots(2,5,figsize = (25, 20))
axes[0][0].title.set_text("Resource Allocation Index (Label = 1)")
axes[0][1].title.set_text("Jaccard Coefficient (Label = 1)")
axes[0][2].title.set_text("Adamic Adar Index (Label = 1)")
axes[0][3].title.set_text("Preferential Attachment (Label = 1)")
axes[0][4].title.set_text("Common Neighbors (Label = 1)")

axes[1][0].title.set_text("Resource Allocation Index (Label = 0)")
axes[1][1].title.set_text("Jaccard Coefficient (Label = 0)")
axes[1][2].title.set_text("Adamic Adar Index (Label = 0)")
axes[1][3].title.set_text("Preferential Attachment (Label = 0)")
axes[1][4].title.set_text("Common Neighbors (Label = 0)")

axes[0][0].hist(label1['rai'], bins = 50)
axes[0][1].hist(label1['jac'], bins = 50)
axes[0][2].hist(label1['ada'], bins = 50)
axes[0][3].hist(label1['pa'], bins = 50)
axes[0][4].hist(label1['num'], bins = 50)

axes[1][0].hist(label0['rai'], bins = 50)
axes[1][1].hist(label0['jac'], bins = 50)
axes[1][2].hist(label0['ada'], bins = 50)
axes[1][3].hist(label0['pa'], bins = 50)
axes[1][4].hist(label0['num'], bins = 50)

plt.show()
```

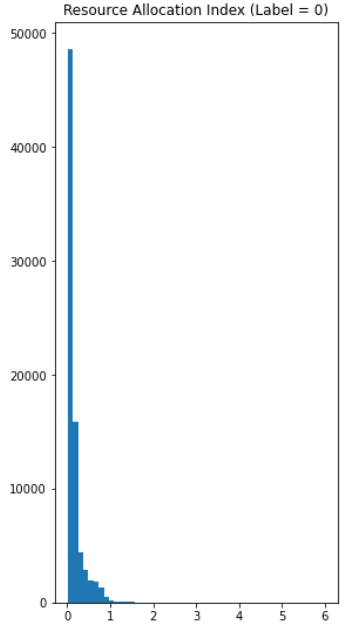

Resource Allocation Index (Label = 1)

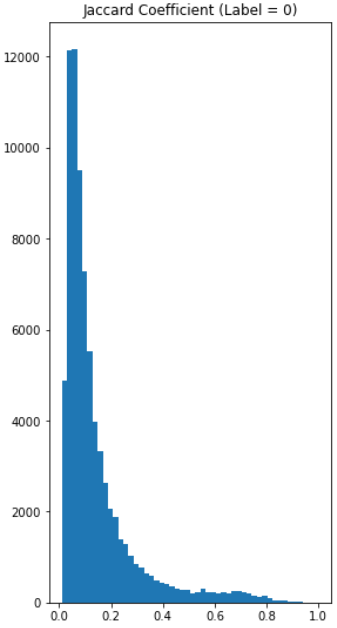

Jaccard Coefficient (Label = 1)

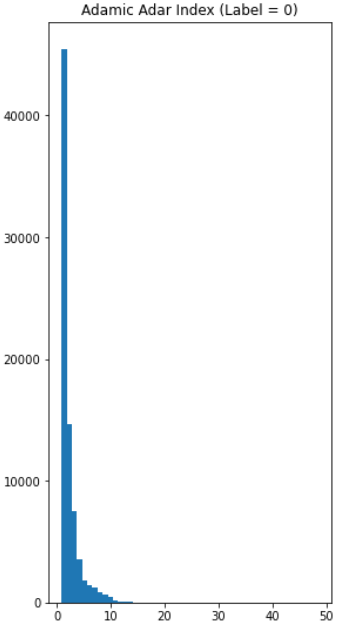

Adamic Adar Index (Label = 1)

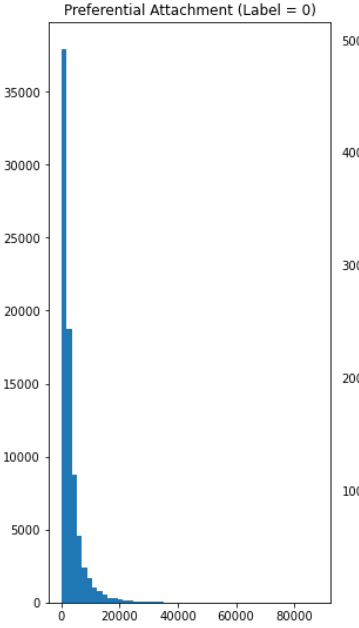

Preferential Attachment (Label = 1)

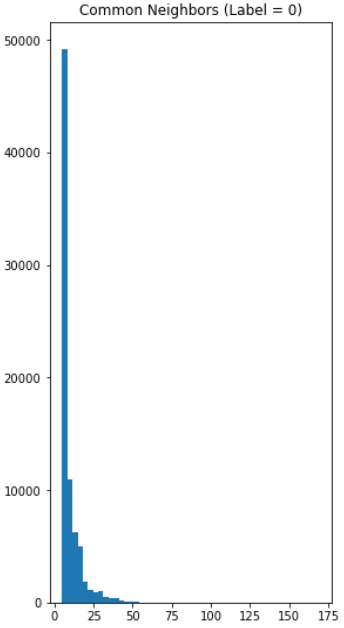

Common Neighbors (Label = 1)


Resource Allocation Index (Label = 0)


Jaccard Coefficient (Label = 0)


Adamic Adar Index (Label = 0)


Preferential Attachment (Label = 0)


Common Neighbors (Label = 0)

Q21. (1 points, Autograded) Measure the correlation between each of the five attributes and the label.

Hint: You may apply `pandas.DataFrame.corr` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html) for this question. The desired correlation dataframe should be in a square size, and the index names and the column names are the same.

In [45]:
```
corr = link_pred.corr()  # Assign your correlation dataframe to this variable. This should be a float.
```

In [46]:
```
#hidden tests for Question 21 are within this cell
```

Q22. (1 point, Autograded) Based on the correlations, which single feature do you expect to perform best at predicting edges?

Set the column name of your selection into the variable `selected_attribute` as a string. For example, if you select the attribute "number of common neighbors", indicate this as

```
selected_attribute = "num"
```

In [47]:
```
selected_attribute = 'rai' #choose your selected attribute. This should be a string.
```

In [48]:
```
#hidden tests for Question 22 are within this cell
```

Q23. (20 points, Manually graded) Link prediction

We are now ready to train and test classifiers for the link prediction task.

First, we will split the data into training and testing.

In [49]:
```
# split data
X_train, X_test, y_train, y_test = train_test_split(
    link_pred[["num", "rai","jac","ada","pa"]], link_pred.label, test_size=0.3, random_state=42)
```

Now use each feature as the only attribute for the link prediction task. Train and test the performance of four different models:

- `LinearSVC`
- `GaussianNB`
- `RandomForestClassifier`
- `AdaBoostClassifier`

You can choose any hyperparameters.

Create a dataframe `F1_score_df`, which stores the F1-score of your test set for each of the four features for each model.

Finally, train and test the performance of each model using all five features, and add a new column to the dataframe with the test F1 score.

You dataframe will have the following format:

model	num	rai	jac	ada	pa	all features
LinearSVC						
GaussianNB						
RandomForestClassifier						
AdaBoostClassifier						

Be sure to print the final dataframe.

In [50]:
```
linear_svc = []
gaussian_nb = []
random_forest = []
ada_boost = []

features = ['num', 'rai', 'jac', 'ada', 'pa', 'all features']
for feature in features:
    if feature == 'all features':
        lin = LinearSVC(random_state = 42).fit(X_train, y_train)
        y_pred = lin.predict(X_test)
        linear_svc.append(f1_score(y_test, y_pred))

        gauss = GaussianNB().fit(X_train, y_train)
        y_pred = gauss.predict(X_test)
        gaussian_nb.append(f1_score(y_test, y_pred))

        rf = RandomForestClassifier(random_state = 42).fit(X_train, y_train)
        y_pred = rf.predict(X_test)
        random_forest.append(f1_score(y_test, y_pred))

        ada = AdaBoostClassifier(random_state = 42).fit(X_train, y_train)
        y_pred = ada.predict(X_test)
        ada_boost.append(f1_score(y_test, y_pred))

    else:
        lin = LinearSVC(random_state = 42).fit(np.array(X_train[feature]).reshape(-1, 1), y_train)
        y_pred = lin.predict(np.array(X_test[feature]).reshape(-1, 1))
        linear_svc.append(f1_score(y_test, y_pred))

        gauss = GaussianNB().fit(np.array(X_train[feature]).reshape(-1, 1), y_train)
        y_pred = gauss.predict(np.array(X_test[feature]).reshape(-1, 1))
        gaussian_nb.append(f1_score(y_test, y_pred))

        rf = RandomForestClassifier(random_state = 42).fit(np.array(X_train[feature]).reshape(-1, 1), y_train)
        y_pred = rf.predict(np.array(X_test[feature]).reshape(-1, 1))
        random_forest.append(f1_score(y_test, y_pred))

        ada = AdaBoostClassifier(random_state = 42).fit(np.array(X_train[feature]).reshape(-1, 1), y_train)
        y_pred = ada.predict(np.array(X_test[feature]).reshape(-1, 1))
        ada_boost.append(f1_score(y_test, y_pred))
```

In [51]:
```
models = ['LinearSVC', 'GaussianNB', 'RandomForestClassifier', 'AdaBoostClassifier']
F1_score_df = pd.DataFrame(models, columns = ['model'])

for i in range(6):
    results = []
    feature = features[i]
    results.append(linear_svc[i])
    results.append(gaussian_nb[i])
    results.append(random_forest[i])
    results.append(ada_boost[i])

    F1_score_df[feature] = results

F1_score_df
```

Out[51]:

	model	num	rai	jac	ada	pa	all features
0	LinearSVC	0.632034	0.813808	0.667639	0.690100	0.670125	0.532569
1	GaussianNB	0.555127	0.789968	0.620464	0.632489	0.651247	0.708911
2	RandomForestClassifier	0.697149	0.803499	0.736217	0.740910	0.599767	0.876475
3	AdaBoostClassifier	0.697286	0.868025	0.742657	0.797866	0.518097	0.869220

End