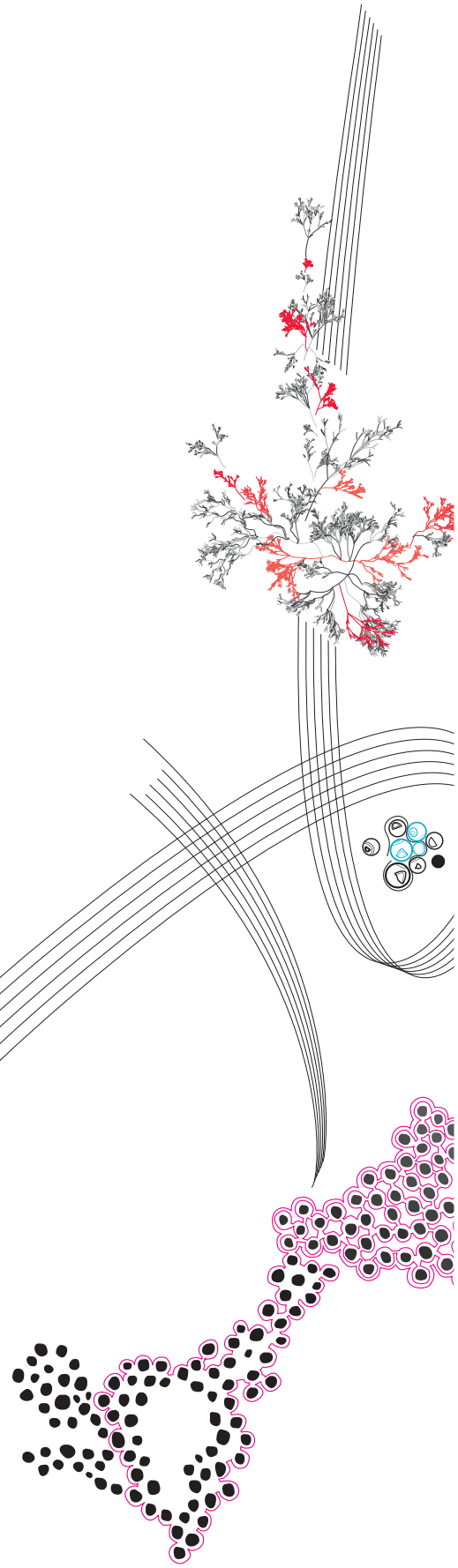

BSc Thesis Applied Mathematics

# Machine-learning methods for discovering growth mechanisms in complex networks

Weiting Cai

Supervisor:
Prof.Dr. Nelly Litvak
Dr. Doina Bucur

June, 2021

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

## Preface

This paper is to fulfill the requirements of the degrees in Applied Mathematics and Technical Computer Science.

First, I would like to thank my superviosrs Doina Bucur and Nelly Litvak, for giving me the opportunity to work on this project and providing technical support and helpful feedback even though you are both very busy. Without your guidance, I will not be able to write this paper, so I am really grateful for your time and patience.

Second, I would like to thank Lilian Spijker and Hajo Broersma for their support throughout my bachelor.

Finally, I would like to thank my family and friends for their help, especially my friend Di Zhuang. Without her help and encouragement during the three years, I will not be able to survive to obtain the degrees.

# Machine-learning methods for discovering growth mechanisms in complex networks

Weiting Cai

June, 2021

**Abstract**

Preferential attachment (PA) and fitness (F) are two hypothetical mechanisms that explain the formation of scale-free networks, which are networks with asymptotically power-law distribution. These mechanisms are interesting because they both lead to scale-free networks, which are often seen in real-life, but they are different with respect to the process of network development. So far, considerable progress has been achieved in random graph theory that describes how scale-free networks develop, while there is little work on uncovering and explaining these mechanisms behind a given network. Therefore, in this article, we aim to train a machine-learning classifier that can differentiate between PA mechanism and F mechanism behind networks. We use a flexible and scalable feature design that organizes features in a matrix. To reduce overfitting by removing the noise in the data, we normalize the feature matrix. We use synthetic networks generated by a PA-based model and an F-based model to evaluate the performance of the classifier and show that the PA and the F mechanisms can be perfectly distinguished by a decision tree classifier. In addition, we clearly see one dominating feature out of all features in the matrix. We show how different parameters of the two models will affect the values of the features and the dominating feature. Importantly, we show that the threshold value of the decision tree model to distinguish the two mechanisms are in accordance with the result of mathematical analysis in a special case.

*Keywords*: scale-free networks, power-law, preferential attachment, fitness, machine-learning classifier, feature design

# 1 Introduction

Network theory is powerful for modelling real-world networks in social, technological and biological scenarios. In order to extract information from these networks, it is important to be able to understand and to predict how the network develops. So far, considerable progress has been achieved in random graph theory on modelling the mechanisms behind the evolution of networks such as preferential attachment (PA) and fitness (F). In particular, in PA models, each node that already has a high degree is more likely to attract edges of later nodes, which is called the 'rich-get-richer' phenomenon [12][3]. In F models, node that has a higher intrinsic fitness value is more likely to attract new links, where higher fitness value indicate higher initial attractiveness of the node. The two mechanisms are interesting because they both lead to asymptotically power-law degree distribution, which is often seen in real-life networks. Networks with asymptotically power-law degree distribution are called scale-free networks.

However, there is no reliable method to recognize the mechanisms behind the growth of a given network. In this thesis, we focus on differentiating the PA mechanism from the F mechanism behind the growth of a given network using data-driven methods. Note that this thesis is a part of a larger project, which includes Di Zhuang's thesis [24] that focuses on analytical studies of the two mechanisms. We will compare our experimental results with the analytical results of that thesis [24] later.

## 1.1 Related work

In the previous work, it has been shown that the PA mechanism is able to produce scale-free network [3]. In addition, it has been shown that the scale-free networks may also result from the F mechanism [7][4] and the combination of PA and F [6]. In this thesis, as for the PA mechanism, we use the PA model described in the textbook of R.V.D Hofstad [12] to generate scale-free networks. As for the F mechanism, we used the model defined by Di Zhuang[24], which can generate scale-free networks given certain parameter values.

In general, there is little research on identifying the mechanisms behind the growth of a network and the existing work is often limited to static networks. In particular, some features, which are computed from static network data, are designed such that they are able to identify the type of a given network such as food web or logistic web [2][13][18]. Specifically, four static features: density, average degree, assortativity, and maximum k-core are used in [18] for predicting the category of a given network. However, since the features are static, they do not help with identifying the mechanism behind the evolution of the given network. Another method is a 'negative approach' [5], which assigns a given network a particular model if the classifier cannot tell the difference between the computed features of the given network and those of the model. In particular, as the input to the classifier, a feature vector is computed for each real network (class 0) and for each network generated by the random graph model (class 1). If a feature vector computed from a real network is misclassified as class 1, then the corresponding model is assigned to the real network. However, according to the authors, this method suffers from overfitting and thus lacks generality: it is likely that all models are rejected.

It is noteworthy that there is an approach taking visualization of adjacency matrices of networks as images, and applying image classification to classify the corresponding networks [11], but this approach does not include the evolution process of the networks.

By contrast, in this project, we will train a classifier that is able to differentiate networks generated by PA models and those generated by F models. In addition, to overcome the above limitations of previous work, we explore an innovative approach, where the features computed from network data are organized into a two-dimensional feature matrix that explicitly includes time dimension. By using this novel feature design, and combine the results with the analytical results of Di Zhuang [24], we can explain what is the most likely mechanism behind the dynamics of networks and how this decision is reached by the classifier.

## 1.2 Research question

Note that this bachelor thesis and Di Zhuang's bachelor thesis [24] form a larger project, which has the following research question: *"What features of a network can enable a machine learning classifier to recognize the (PA) or (F) mechanisms behind the growth of the network in a mathematically interpretable way?"*
Specifically, to answer the research question, the following tasks need to be completed:

1. Analyze a PA-based model and an F-based model and explore which statistical characteristics are different for the two models.
2. Design features for machine learning.
3. Train the machine learning model using synthetic data.
4. Evaluate feature importance based on the training data and the performance of the trained model, and compare the result with that of the mathematical analysis.
5. Interpret the results produced by the classifier based on the results of Task 1.

In this thesis, we mainly focus on the tasks 2, 3 and 4, while the tasks 1 and 5 are dealt in Di Zhuang's thesis [24].

## 1.3 Our Contribution

Overall, our paper makes the following contributions:

1. We add normalization to the feature matrix, which is a feature design that organizes features in a matrix provided by our supervisors N. Litvak and D. Bucur. This slight change turns out to have good performance on classifing networks generated by the PA and F models with different parameters.
2. We show that the feature design is so powerful that even the default decision tree in scikit-learn[19] can perfectly distinguish the networks constructed by the two models.
3. We show that although some features of PA and F mechanisms are similar, they are of different importance and there is one distinctive feature to distinguish PA and F models.
4. We show that the bounds of the distinctive feature are in accordance with the mathematical analysis of Di Zhuang in a special case.
5. We show how the parameters of the PA and F models will affect the threshold value to differentiate networks with PA and F mechanism.

# 2 Models

In this section, before we introduce both models, which are used to generate scale-free networks, we first introduce the concept of scale-free networks. Finally, we give some notations and pseudocodes for the two models.

## 2.1 Scale-free networks

Networks are called scale-free if their degree distribution follows a power law [1]. That is, for an undirected scale-free network, we can write its degree distribution as:

$$P(k) \propto k^{-\gamma},$$

where $\gamma$ is a real constant. An example for a scale-free network is given in Figure 1.

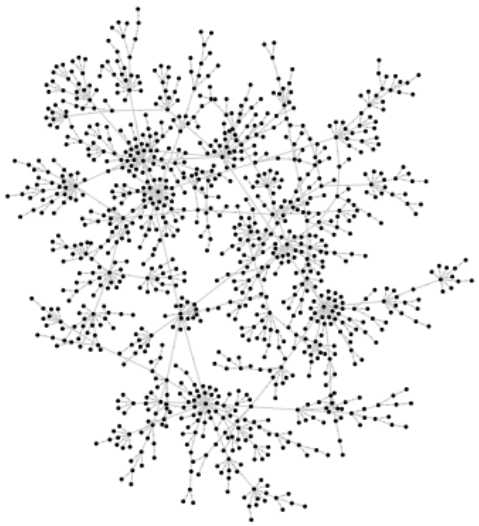

FIGURE 1: A scale-free network generated by the graph randomizer of Cytpscape 3.8.2, Barabasi-Albert model with $N = 1000$ and $m = 1$

## 2.2 Preferential attachment model

For consistency, we use the same model description as in Di Zhuang's thesis [24]. Note that the model is exactly the PA-based model defined by Remco in his book [12] as follows:

> The model produces a graph sequence denoted by $(\mathrm{PA}_t^{(m,\delta)})_{t\geq 1}$. At each time step, the model generates a graph with $t$ nodes and $mt$ edges. As $(\mathrm{PA}_t^{(m,\delta)})_{t\geq 1}$ is defined in terms of $(\mathrm{PA}_{mt}^{(1,\delta/m)})_{t\geq 1}$, we first introduce the special case $(\mathrm{PA}_t^{(1,\delta)})_{t\geq 1}$, and then introduce the general case $(\mathrm{PA}_t^{(m,\delta)})_{t\geq 1}$.
>
> First, we describe the case $m = 1$. In this case, $\mathrm{PA}_1^{(1,t)}$ contains a single node with a self-loop. We denote the nodes of $\mathrm{PA}_1^{(1,\delta)}$ by $v_1^{(1)}, v_2^{(1)}, v_3^{(1)}, .., v_t^{(1)}$. We denote the degree of node $v_i^{(1)}$ in $\mathrm{PA}_t^{(1,\delta)}$ by $D_i(t)$. By convention, a self-loop increases the degree by 2. At each time step $t$, a node $v_t^{(1)}$ arrives with an edge incident to it. The other end point of the edge is connected to $v_t^{(1)}$ with probability $(1+\delta)/(t(2+\delta)+(1+\delta))$, and to $v_i^{(1)}$ $i \in \{1, 2, .., t-1\}$ with probability $(D_i(t)+\delta)/(t(2+\delta)+(1+\delta))$.
>
> Next, we describe the model with $m > 1$ in terms of the model with $m = 1$. Fix $\delta \geq$

$-m$, we start with $\mathrm{PA}_{mt}^{(1,\delta/m)}$ and denote the nodes in $\mathrm{PA}_{mt}^{(1,\delta/m)}$ by $v_1^{(1)}, v_2^{(1)}, v_3^{(1)}, .., v_{mt}^{(1)}$. Then we identify the nodes $v_1^{(1)}, v_2^{(1)}, v_3^{(1)}, .., v_m^{(1)}$ to be the node $v_1^{(m)}$ in $\mathrm{PA}_t^{(m,\delta)}$. In general, we collapse the nodes $v_{(j-1)m+1}^{(1)}, v_{(j-1)m+2}^{(1)}, v_{(j-1)m+3}^{(1)}, .., v_{jm}^{(1)}$ to be the node $v_j^{(m)}$ in $\mathrm{PA}_t^{(m,\delta)}$. The resulting graph would be a multigraph with $t$ nodes and $mt$ edges.

Note that the "rich-get-richer" effect is also called the "old-get-richer" effect in this model for each node that arrives early is more likely to attract new links than the later node. This phenomenon has an intuitive explanation: each node that arrives early has less competitors, thus is more likely to attract new links, thus becoming a node with high degree later, attracting even more new links [24]. To understand this idea, an adjacency matrix of a PA network generated by the model with parameters $t$=100, $m$=10 and $\delta = 0$ is given in Figure 2. The x-axis is the arriving node, while the y-axis is the node linked by the arriving node. We can see the nodes arriving early (at the bottom of Figure 2) have significantly high degree that most of the nodes arriving later will connect to them.

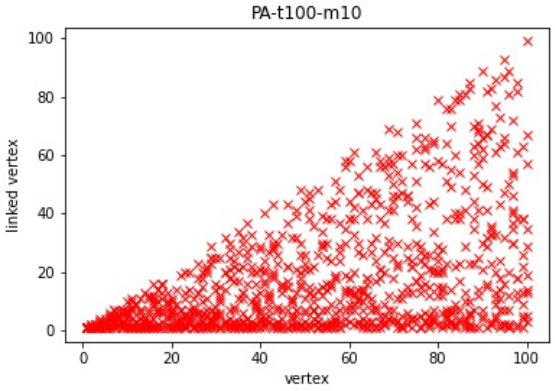


FIGURE 2: An adjacency matrix of a network generated by PA model

## 2.3 Fitness model

In this section, we introduce the F-based model defined in Di Zhuang's thesis [24] as follows:

The model produces a graph sequence denoted by $(\mathrm{F}_t^{(m,\lambda)})_{t\geq 1}$. At each time step, the model generates a graph with $t$ nodes and $mt$ edges. We denote the nodes in $\mathrm{F}_t^{(m,\lambda)}$ by $v_1, v_2, v_3, .., v_t$. For each node $v_i$, we denote the fitness value of the node, which is a random variable of $exp(\lambda)$ distribution, by $\Phi_i$.

Given $m \geq 1$, $\mathrm{F}_1^{(m,\delta)}$ contains $m$ isolated nodes and no edges. At each time step $t \geq 1$, a node $v_t$ with fitness value $\phi_t$ arrives with $m$ edges incident to it. For each edge incident to $v_t$, the other end of the edge is connected to $v_i$ with probability $\phi_i / \sum_j \phi_j$. The resulting graph would be a multigraph with $t$ nodes and $mt$ edges.

In particular, the distribution of asymptotic node degrees is close to power-law distribution for $1 \leq \lambda < 10$ [24].

An adjacency matrix of a F network generated by the model with parameters $t$=100, $m$=10 and $\lambda = 1$ is given in Figure 3. The x-axis is the arriving node, while the y-axis is the node linked by the arriving node. We can see that the nodes with high degree are not the old ones anymore, instead, each node with high fitness value but arrives late is likely to have high degree as well.

## 2.4 Notations

In this thesis, we observe the networks on time horizon [0,$T$]. For both of the PA and F models, we will mainly discuss two of their parameters:

- $m$:= number of new links per arriving node
- $T$:= number of nodes in the final network

Also, we denote a network with size of $x$ generated by PA or F model having $y$ newlinks per new node by: a T$x$-m$y$-PA network or a T$x$-m$y$-F network. For example, we name a network with 1500 nodes generated by the PA model, where each arriving node has 10 new links as: a T1500-m10-PA network.

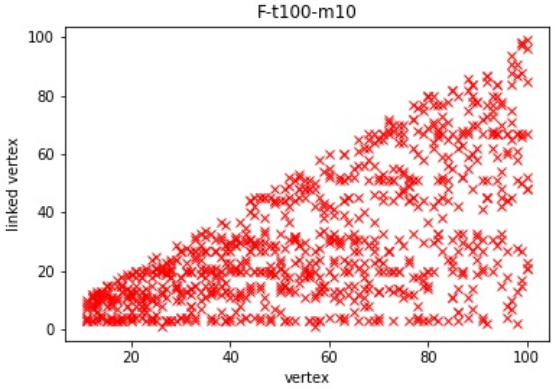


FIGURE 3: An adjacency matrix of a network generated by F model

## 2.5 Pseudocodes of the PA and F models

The algorithms for generating the PA and F networks are presented below, we assume that arrays and lists are indexed from 1, so the loops start at 1. In algorithm 1, at line 21, and in algorithm 2, at line 9, the *probabilities* is a list of probabilities that the arriving node $n_i$ will connect to nodes from the *nodes* list. In algorithm 1, at line 25, the *newlinks* list containing the indices of the nodes to be linked by the arriving node $n_i$ is obtained by weighted random choices, which depends on *nodes* list and the *probabilities* list. Note that the weights here are the probabilities in the *probabilities* list. At the end of the algorithms, since a network is generated with its characteristics are stored in *degrees* and *edges* lists, we will then be able to write the lists to CSV files and document the parameters of the network in a file result.csv, which can be used to compute features of the networks. Finally, the computed features will be used as training datasets and test datasets for the classifier.

**Algorithm 1** Simulation of networks with fitness mechanism

$m$ := number of new links for each arriving node
$\lambda$ :=: parameter for the exponential fitness distribution
$t$ :=: number of nodes in the final network
$t_0$ := number of nodes in the initial network
*edges* := []
*degrees* := []
$fitnesses$ := []
*nodes* := []
**for** i = 1 to $t_0$ **do**
    nodes.append(i)
    degrees[i].append([i, 0])
    fitnesses[i] := a random variable drawn from the exponential distribution with parameter $\lambda$
**end for**
fitnessSum := sum(fitnesses)
// F network evolution
**for** $n_i$ =$t_0$+1 to t **do**
    fitness:= a random variable drawn from the exponential distribution with parameter $\lambda$
    fitnesses.append(fitness)
    fitnessSum:= fitnessSum + fitness
    nodes[$n_i$].append($n_i$)
    probabilities := []
    **for** i = 1 to $n_i$ **do**
        probabilities[i] := $\frac{fitnesses[i]}{fitnessSum}$
    **end for**
    *newlinks* := weighted random choices of m nodes
    **for** i in newlinks **do**
        degrees[i][1]++
        degrees[$n_i$][1]++
        edges.append($n_i$, i)
    **end for**
**end for**
Finally, write edges to a CSV file and degrees to another CSV file as part of the training data and write parameters of the network into a result.csv file.

**Algorithm 2** Simulation of networks with PA mechanism

m:= number of new links for each arriving node
$\delta$ :=: parameter for the preferential attachment model
t:=: number of nodes in the final network
edges:= [(1,1)]
degrees:= [2]
nodes:= [1]
**for** $n_i$ = 2 to mt **do**
    nodes.append($n_i$)
    probabilities = []
    degreeSum = ($n_i$ - 1)*(2 + $\delta$) + (1 + $\delta$)
    **for** i = 1 to $n_i$ **do**
        probabilities[i] = $\frac{(degrees[i]+\delta)}{degreeSum}$
    **end for**
    probabilities.append($\frac{1+\delta}{degreeSum}$)
    j:= weighted random choice of a node from *nodes*
    degrees.append(1)
    degrees[j]++
    edges.append(($n_i$, j))
**end for**
// Collapsing procedure
collapsedNodes := []
collapsedEdges := []
collapsedDegrees := []
**for** i = 1 to t **do**
    collapsedNodes.append(i)
    collapsedDegrees.append([i, 0])
**end for**
**for** edge in edges **do**
    node1 = (edge[0] - 1) // m + 1
    node2 = (edge[1] - 1) // m + 1
    collapsedEdges.append((node1,node2))
    collapsedDegrees[node1][1]++
    collapsedDegrees[node2][1]++
**end for**
Finally, write edges to a CSV file and degrees to another CSV file as part of the training data and write parameters of the network into a result.csv file.

# 3 Methods

In this section, the overall method is introduced and explained.

## 3.1 Feature design

We explore an innovative approach - a flexible and scalable feature design that organizes features in a two-dimensional matrix $M$ as follows: rows of this matrix correspond to time, columns correspond to groups of nodes, and the cells contain the network's incremental statistics.

Formally, learned from the work of our supervisors, the general $M_{a\times b}$ feature matrix is defined as follows:

Let $G(T)$ be the set of nodes generated on the interval [0,$T$] and $d_k(t)$ be the degree of each node $v_k$ at time step $t$. Define

$$F_T(x) = \frac{1}{T}\sum_{v_k\in G(T)} \mathbb{1}\{d_k(T) \leq x\}, x \in [0, 1, .., \max_{v_k\in G(T)} d_k(T)],$$

to be the empirical distribution of the node degrees in the final network. For each $d_k(T)$, let $q_k$ be such that $d_k(T)$ is the $q_k$-quantile of $F_T(x)$. Then we divide $G(T)$ into $b$ groups according to their $F_T$-quantiles as follows:

$$G_j = \{v_k \in G(T) : \frac{j-1}{b} < q_k \leq \frac{j}{b}\}, j \in \{1, 2, .., b\}.$$

In particular, $G_1$ contains the nodes of which degrees end up in the highest $\frac{1}{b}\%$, while $G_b$ contains the nodes with degrees ending up in the lowest $\frac{1}{b}\%$.

Then we divide the time into $a$ consecutive non-overlapping intervals with equal length, which are $T_1 = [0,\frac{T}{a}]$, $T_2 = [\frac{T}{a},\frac{2T}{a}]$,.., $T_a = [\frac{(a-1)T}{a},T]$.

Finally, we compute the incremental statistics in the cells - $M_{ij}$ = the average number of new links received per node in node group $G_j$ during time interval $T_i$) - as follows:

$$M_{ij} = \frac{1}{|G_j|}\sum_{v_k\in G_j}\left(d_k\left(\frac{i}{b}T\right) - d_k\left(\frac{i-1}{b}T\right)\right), i \in \{1, 2, .., a\}, j \in \{1, 2, .., b\}.$$

Note that the matrix size should not be too large, otherwise overfitting will happen since there are too many features. In this thesis, we randomly pick a $3 \times 4$ matrix to see if it works. The time dimension is splitted into 3 consecutive non-overlapping intervals with equal length. The node group dimension divides nodes into 4 groups by their eventual degree. We denote the feature in the entry $M_{ij}$ by ($T_i$, $G_j$) as shown in Figure 4.

However, this method alone is not sufficient to reach our goal, because we found overfitting is likely to occur after we did some experiments. This is why we use the method introduced below in Section 3.2

Note that due to time constraints, we did not try other matrices since this $3 \times 4$ matrix after being normalized is already able to ensure a good performance.

| | $G_1$ | $G_2$ | $G_3$ | $G_4$ |
|---|---|---|---|---|
| $T_1$ | $(T_1,G_1)$ | $(T_1,G_2)$ | $(T_1,G_3)$ | $(T_1,G_4)$ |
| $T_2$ | $(T_2,G_1)$ | $(T_2,G_2)$ | $(T_2,G_3)$ | $(T_2,G_4)$ |
| $T_3$ | $(T_3,G_1)$ | $(T_3,G_2)$ | $(T_3,G_3)$ | $(T_3,G_4)$ |

FIGURE 4: Feature Matrix

### 3.2 Reduce overfitting by normalizing the matrix

Before we introduce the matrix normalization, we learn the concept of overfitting from IBM Cloud Education [15]:

> Overfitting is a concept in data science, which occurs when a statistical model fits exactly against its training data. When machine learning algorithms are constructed, they use a sample dataset to train the model. However, when the model trains for too long on sample data or when the model is too complex, it can start to learn the "noise," or irrelevant information, within the dataset. When the model memorizes the noise and fits too closely to the training set, the model becomes "overfitted", and it is unable to generalize well to new data.

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information [22]. For example, in our case, consider a feature matrix of a T2000-m10-PA network and a feature matrix of T200-m10-PA network before normalization (recall that the notations are defined in Section 2), the value of any feature $(T_i,G_j)$ in the former matrix is likely to be about 10 times larger than that in the latter matrix, which might affect the classification result, but it does not necessarily indicate that the two values convey different information, because the difference is caused by different $mT$ - the sum of all $(T_i, G_j)$'s in the matrix. That is, they are different because they have different scales. In this case, normalization is necessary to make the two networks comparable and thus to obtain more accurate classification results.

Therefore, we use normalized matrix to make networks with different $T$ and $m$ comparable. With common scale for all feature matrices, the noise in the data is removed and so the effect of overfitting is reduced as the effects of different $T$ and $m$ are reduced.

In simple words, to normalize the matrix, we scale the values in the matrix so that if they were all summed, the value would be 1. Formally, let $M'$ denote the normalized matrix, the feature matrix $M$ is scaled as follows:

$$M'_{ij} = \frac{M_{ij}}{\sum_{k\in\{1,2,..,a\},l\in\{1,2,..,b\}} M_{kl}}, i \in \{1,2,..,a\}, j \in \{1,2,..,b\},$$

### 3.3 Classifiers

In this thesis, we did two experiments. The first one is to verify the mathematical results of Di [24], using networks with fixed $\delta$ and $\lambda$. The other one is to explore the effects of varying $\delta$ and $\lambda$.

For the first experiment, we first tried the default random forest classifier in scikit-learn [21] to see how good the feature design is and what features are the most important ones to differentiate the two models. Due to the perfect performance of the random forest classifier, we then use a less complicated classifier - the default decision tree classifier in scikit-learn [19] - to see its performance and to analyse feature importance. Importantly, we check the threshold value that the decision tree classifier use to classify the networks.

For the second experiment, we use the default decision tree classifier in scikit-learn [19] and the feature importance and accuracy rate are analysed as well.

# 4 Experiment setup

First, we describe the experiment setup for PA networks with $\delta$=0 and F networks with $\lambda = 1$ in Section 4.1, which is to verify the results of Di's work [24]. In real life, networks could be generated by a range of parameters. Thus it is important to explore the effects of varying parameters. Therefore, next, we show the experiment setup for PA networks with various $\delta$ and F networks with various $\lambda$ in Section 4.2.

For both of the experiments, the parameters of the classifiers are the same as the default ones described in Figure 7.

## 4.1 PA networks with $\delta$=0 and F networks with $\lambda = 1$

### 4.1.1 Range of $T$ and $m$

Before we generate various networks, we did several experiments to observe what might be the reasonable ranges of the parameters $T$, $m$ of PA and F networks and $\delta$ of PA networks and $\lambda$ of F networks.

Randomly picking the feature ($T_1$,$G_1$) and observe its values, which are computed from various kinds of PA and F networks generated by different $T$ and $m$ and fixed $\delta = 0$ and $\lambda = 1$, we find that:

- Generating PA networks is time-consuming when $T$ is large, while the variance of ($T_1$,$G_1$) become small enough, when $T \geq 1000$.
- Although when $T \geq 1500$, the variance of ($T_1$,$G_1$) is small and generating F networks is fast, extracting features is also time-consuming as $T$ and $m$ increase.

Therefore, for simplicity and due to time constraints, we mainly generate F networks with $T$ = 1500 to 2000 and $m$ = 10 to 15 and PA networks with $T$ = 1000 to 1500 and $m$ = 10 to 15 as dataset. For generality, we also generate some other PA and F networks with $T$ = 2000 to 5000 and $m$ = 15 to 45.

### 4.1.2 Dataset for classification

Since, during the preliminary experiment step, we found that 200 data, which is splitted into 60% training set, 20% validation set and 20% test set, is enough to ensure the random forest classifier a good performance, we decide to generate 10 times more data to see if the performance will change.

Therefore, we randomly choose to generate 480 T1500-m15-PA networks, 480 T1500-m15-PA networks, 480 T1500-m15-F networks, 480 T2000-m10-F networks and around 200 PA and F networks with parameters $T$ range from 2000 to 5000 and $m$ range from 10 to 45. In total, we have around 2100 networks as dataset.

## 4.2 PA networks with various $\delta$ and F networks with various $\lambda$

This experiment is based on the results in Section 5.1 of the above experiment. In addition, due to time constraints and that there is no mathematical analysis on varying $\delta$ and $\lambda$, we do not investigate much in this experiment.

### 4.2.1 Range of $T$ and $m$

Due to time constraints, we are not able to generate sufficient amount of large networks, so we mainly choose $T = 1500$ for F networks, $T = 1000$ for PA networks and $m = 10$ for both models and some other networks shown in Section 4.2.3.

### 4.2.2 Range of $\delta$ and $\lambda$

As for $\lambda$, based on the analysis of $\lambda$ in Di's thesis [24], we choose $\lambda = 1$ to 9 to ensure the generated networks are scale-free networks and to observe what effects it will bring.

As for $\delta$, based on the analysis of $\delta$ in the book of Remco [12] that when $\delta \geq 0$, the generated networks will be scale-free networks, we randomly choose $\delta = 0$ to 100 to observe what effects it will bring.

### 4.2.3 Dataset for classification

At first we use the following data in Figure 5 to observe the results of the decision tree classifier. Note that the file with filename "F/PA-t$x$-m$y$_$z$features" means that it contains $z$ feature matrices extracted from $z$ T$x$-m$y$-F/PA networks with $\lambda = 0$ or $\delta = 1$. While the file with the similar filename having an extra word "-lamda$a$" or "-delta$a$" in the middle means that it contains $z$ feature matrices extracted from $z$ T$x$-m$y$-F/PA networks with $\lambda = a$ or $\delta = a$.

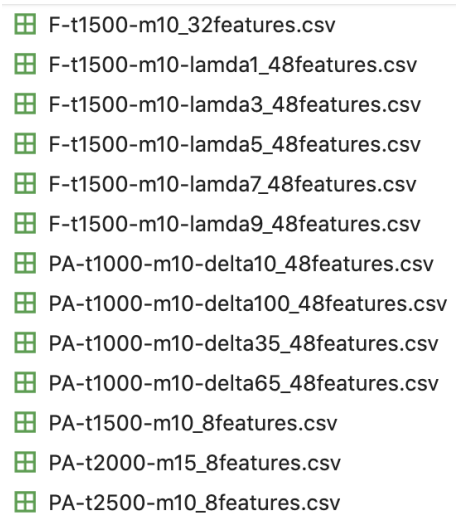


FIGURE 5: Initial dataset in the second experiment

Then we add 3 files shown in Figure 6 to see whether the classification results and feature importances will change in the same classifier:

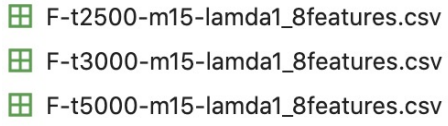


FIGURE 6: Added dataset in the second experiment

### 4.3 Parameters of the classifiers

For the first experiment, we first tried to use the random forest classifier in scikit-learn [21]. Next, we use the default decision tree in scikit-learn [19]. The parameters are listed in Figure 7.

For the second experiment, we use the default decision tree in scikit-learn [19]. The parameters of the classifier are in Figure 7 (right).

Since we found the feature design is so powerful that even the default classifiers can have 100% accuracy in both experiments, we do not investigate further on how to tune the classifiers.

```
Parameters for random forest:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
Parameters for decision tree:

{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

FIGURE 7: Parameters of the classifiers

# 5 Results

## 5.1 PA networks with $\delta = 0$ and F networks with $\lambda = 1$

Before we look into the feature importances of the classifiers, we first introduce the concept of feature importance:

Feature importance describes how important a feature is for the model. It is an approximation of how important a feature is in the data [17]. The higher the feature importance of a feature is, the more important the feature is. Note that the highest value is 1, indicating the feature is dominating, and the lowest is 0, indicating the feature is useless.

### 5.1.1 Feature importances and Performance of the random forest classifier

In this step, we found that the accuracy is 100% (See Figure 8) on both the training set and test set described in 4.1.2, which also proves overfitting did not happen [15]. Then we

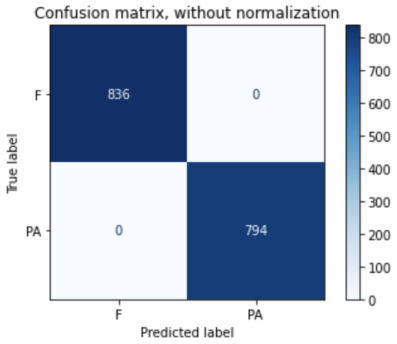


FIGURE 8: Confusion matrix of Random forest

look into the feature importance of each feature in the feature matrix by using the built-in feature importances attribute of the random forest model. At first, we use all features and plot the feature importances in Figure 9 (left). After trying different combinations of the features multiple times by dropping columns of importance 0, we obtain the final feature imporance in Figure 9 (right).

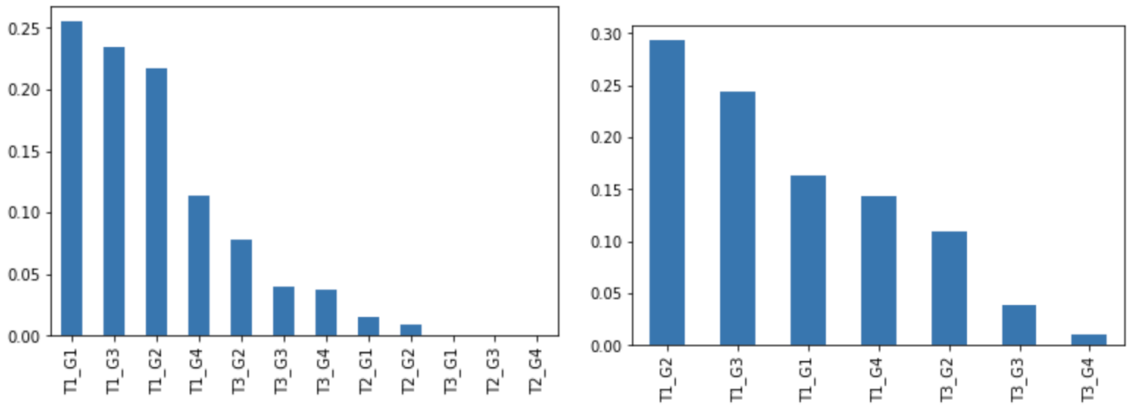


FIGURE 9: Feature importances of Random forest

We next investigated into the values of the features between PA networks and F networks with different parameters $T$ and $m$. We found an interesting phenomenon about the values of the feature $(T_1, G_1)$ as $T$ and $m$ changes: we can distinguish PA and F networks with bare eyes by just observing the values of one feature $(T_1, G_1)$. For example, when $T \geq 1000$ and $m \geq 10$, there is a threshold value such that, if the feature $(T_1, G_1)$ is larger than the threshold value, then it is a PA network, otherwise it is a F network.

To further investigate this phenomenon, we used the decision tree classifier to analyse the accuracy and the feature importances again.

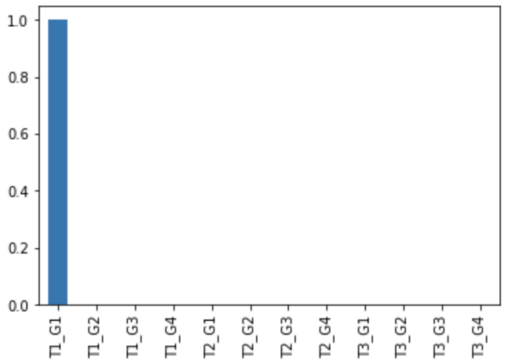


FIGURE 10: Feature importance of the decision tree in the first experiment

### 5.1.2 Feature importances and Performance of the decision tree classifier

We use the same way as described in the above subsection to see the feature importances of the decision tree model and the result is in Figure 10. We found that using only one feature $(T_1, G_1)$, two models can be differentiated with 100% accuracy on both the training set and testset.

It is noteworthy that the features $(T_1, G_2)$ and $(T_1, G_3)$ can also be of importance 1 when selecting all the 12 features ($(T_1, G_1)$ to $(T_3, G_4)$), but so far we have already been satisfied with the results and there is only mathematical analysis for the feature $(T_1, G_1)$, so we stopped here.

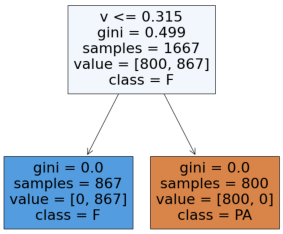


FIGURE 11: Threshold values of the decision tree for networks with $T \geq 1000$ and $m \geq 10$

### 5.1.3 Threshold values of the decision tree with different networks

We use the default method plot_tree in scikit-learn [20] to visualize the decision tree as shown in Figure 11 and 12. Before we reach our conclusion, we first explain the meanings

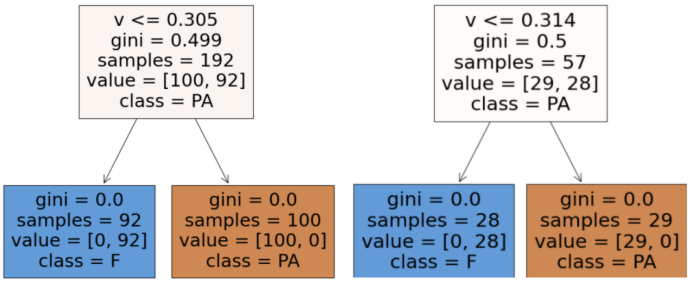


FIGURE 12: Threshold values of the decision tree for networks with $T = 100$ and $m = 10$ (left) and networks with $T = 500$ and $m = 10$ (right)

of the 5 variables:

- $v$: In Figure 11 and 12, the first variable $v$ is the value of the feature $(T_1, G_1)$, which is used to perform multi-class classification on a dataset. The value that the decision tree uses to classify PA and F networks is called "threshold value". For example, in Figure 11, the threshold value is 0.315. That is, if the value of $(T_1, G_1)$ of a network, $v \leq 0.315$, then the network is classified as an F network, otherwise it is classified as a PA network.
- *gini*: This is variable is Gini Impurity. It is the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution in the dataset [23]. Since we are not interested in its value, we skip the details.
- *samples*: This variable indicates the number of networks to be classified.
- *value*: The first entry of this variable is the number of networks classified as PA networks, while the second one is the number of networks classified as F networks.
- *class*: This indicates the majority of the samples are in which class. This indicates which class the majority of the samples are in.

When $T \geq 1000$ and $m \geq 10$, the threshold value that the decision tree uses to differentiate the two models is 0.315 (see Figure 11), which is very close to the analytical value 0.321 calculated in Di Zhuang's thesis [24].

So we now conclude that the two models can be distinguished by the decision tree classifier with 100% accuracy using only one feature $(T_1, G_1)$ and the results are mathematically interpretable. That is, the results are in accordance with the analytical results of Di Zhuang and the difference between PA and F networks is obvious when $T \geq 1000$ and $m \geq 10$.

Although it is rare that a network has $T < 1000$ and $m < 10$, some readers might be curious about how good this feature design is, so we also include two threshold values of the default decision tree trained and tested by networks with $T = 100$ and 500 and $m = 10$ (see Figure 12). We see that it still has 100% accuracy although $T$ is very small. Furthermore, the threshold values changed slightly.

### 5.1.4 Learning curve of decision tree model

To find out how many data the classifier needs to ensure a good performance, we plot its learning curve (see Figure 13) and the data we use here are the same as in the previous

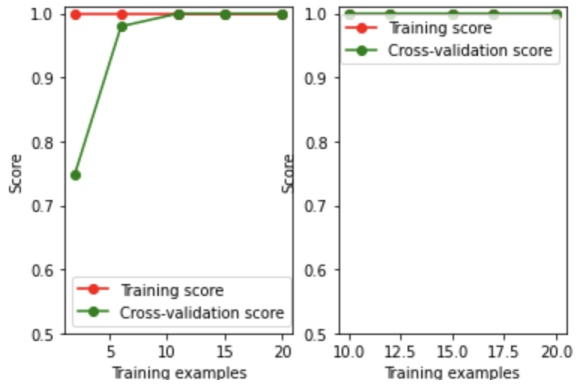


FIGURE 13: learning curve of the default decision tree

section. We see the learning efficiency is also good, since only 10 training examples can train the model to have 100% accuracy.

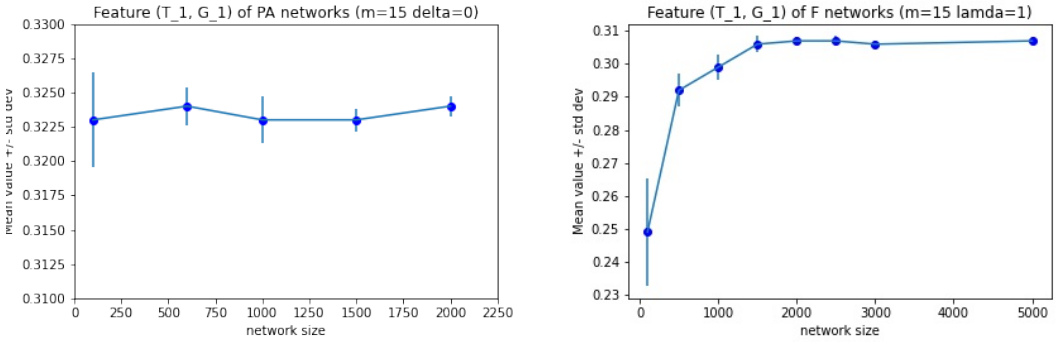


FIGURE 14: Mean values of ($T_1$,$G_1$) with different T.

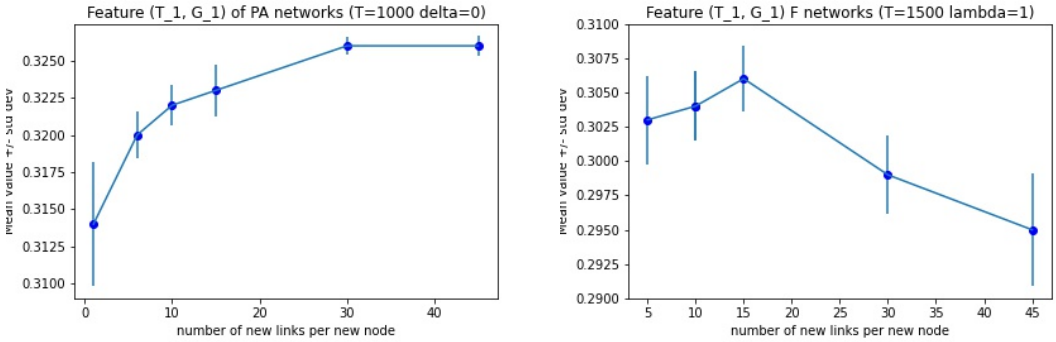


FIGURE 15: Mean values of ($T_1$,$G_1$) with different m.

### 5.1.5 Values of the distinctive feature ($T_1$,$G_1$) of PA and F networks with different parameters

The left part of the two figures, 14 and 15, is for PA networks with $\delta = 0$ and the right part of the two figures is for F networks with $\lambda = 1$. For Figure 14, the x-axis is the network size $T$, while for Figure 15, the x-axis is $m$, the number of newlinks per new arrival node. For both Figures, the y-axis is the mean value $\pm$ standard deviation of the feature ($T_1$,$G_1$) and each data point is computed from the feature ($T_1$,$G_1$) of at least 32 networks.

Note that in Figure 15, $T$ of PA networks is 1000 while $T$ of F networks is 1500. According to Figure 14, we see that the value of $T$ does not affect the values of $(T_1, G_1)$ much when $T \geq 1000$, so we choose $T = 1000$ for PA networks to save time generating PA networks with different $m$. While for F networks, the value of $T$ does not affect the values of $(T_1, G_1)$ when $T \geq 1500$, so we choose $T = 1500$ for F networks.

By observing Figure 14 and 15, for PA networks with $\delta = 0$ and F networks with $\lambda = 1$, we conclude that:

- When $m$ is larger, the difference between PA and F is more significant by observing that as $m$ increases, the values of ($T_1$,$G_1$) of F networks decrease (right Figure 15), but those increase for the PA network (left Figure 15).
- The variances of PA and F networks are large, when $T$ is small (see Figure 14). This means that the PA/F mechanism behind the network has not taken obvious effect yet, if $T$ is small. We can give an intuitive explanation for this: if the networks are small and so the degrees of most nodes are small, then every new link will affect the degree distribution significantly. For example, the degree of a node increasing from 10 to 11 is a significant change, while the number increasing from 100 to 101 is not. This is supported by observing that the variances become negligible as $T$ increases to 2000.

## 5.2 PA networks with various $\delta$ and F networks with various $\lambda$

### 5.2.1 Feature importances and Performance

We use the same method as above to see the feature importances of the decision tree model. When we used the initial dataset as shown in Section 4.2.3 in Figure 5, splitting it into 60% training set, 20% validation set and 20% test set, we found that there is only one important feature $(T_3, G_3)$ shown in Figure 16 (left) and the accuracy is 100% on both training set and test set.

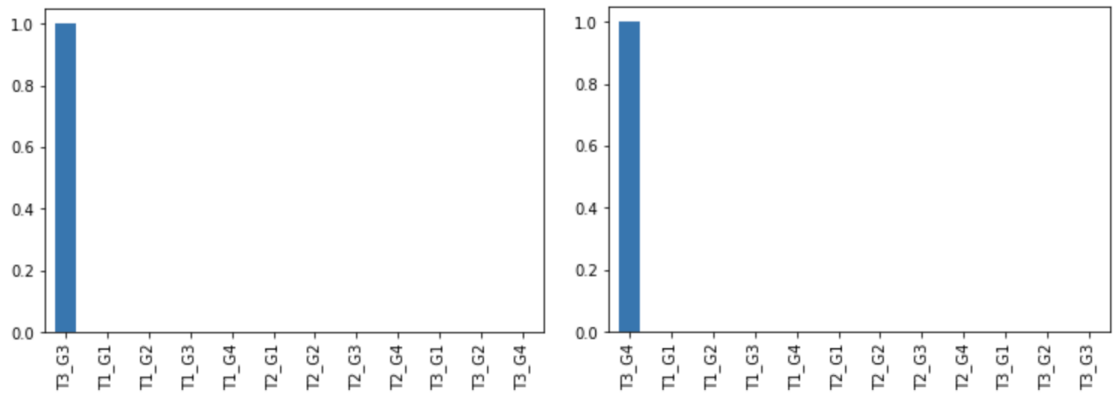


FIGURE 16: Feature importances of the decision tree in the second experiment

However, after we added 3 files as mentioned in Section 4.2.3 in Figure 6 into the initial dataset, splitting it in the same way above, as the new input to the decision tree, although the accuracy is still 100% on both training set and test set, the dominating feature becomes $(T_3, G_4)$ shown in Figure 16 (right).

Now, based on all the observations above, we conclude that:

For PA networks with $0 \leq \delta \leq 100$ and F networks with $1 \leq \lambda < 10$, they can still be distinguished by the decision tree classifier with 100% accuracy, but the dominating feature depends on what datasets are used as input to the decision tree classifier.

# 6 Discussion

In general, the results are in line with the assumption: by including the time dimension, the feature matrix is able to provide information about the temporal changes of the networks during their development, thus revealing which mechanism is driving the growth of the networks.

However, in this work, only a special case of this assumption is investigated in depth, namely the case with $\delta = 0$ and $\lambda = 1$. Although a more general study includes the case with several other values of $\delta$ and $\lambda$, the range of $\delta$ and that of $\lambda$ are still limited, which means that the results might not be generalized. As an improvement, it might be interesting to draw values of $\delta$ and $\lambda$ from a particular probability distribution based on mathematical analyses or observations on real-life networks.

Furthermore, note that the datasets used for training and testing in this work contains only synthetic networks generated by random graph models. Although the classifier performs well on these datasets, this does not necessarily indicate a good performance on real-life networks - it is possible that the models we used do not fit real-life PA or F networks.

# 7 Conclusion

In this project, we explored an innovative method - a two-dimensional feature matrix that explicitly includes the time dimension. This allows us to train a decision tree classifier to recognize the PA and F mechanisms behind the growth of networks. As an improvement, we normalized the feature matrix to make feature matrices of networks of different sizes comparable. As the training and testing sets for the decision tree classifier, we first used a large dataset with networks generated by the PA model with $\delta = 0$ and the F model with $\lambda = 1$ to investigate a special case analysed in Di Zhuang's thesis [24]. Then we used a smaller dataset with networks generated by the two models with varying $\delta$ and $\lambda$ to investigate a more general case.

As a result, with only one dominating feature $(T_1, G_1)$ out of all features in the feature matrix, the decision tree classifier is able to achieve 100% accuracy for networks generated by the PA model with $\delta = 0$ and the F model with $\lambda = 1$. Furthermore, for $\delta = 0$ and $\lambda = 1$, the threshold value 0.315 that the decision tree classifier uses to differentiate PA networks from F networks is in accordance with the analytical value 0.321 obtained in Di Zhuang's work [24] for the same set of $T$'s and $m$'s, which answers the research question for a special case.

Finally, for the more general case with varying $\delta$ and $\lambda$, no fixed dominating feature exists. Instead, the dominating feature changes as the composition of the input dataset changes for small dataset.

# 8 Recommendations

In this thesis, only two mechanisms behind the networks are discussed. Several other mechanisms, namely aging[10], copying[14], deletion[9][8] and similarity[16][25], have been suggested for the growth and decline of networks as well. For example, in a network with aging mechanism, the probability of a node attracting links of later nodes will reduce with increasing age. If these mechanisms behind the networks can be identified in the future, it would be possible to improve the precision and efficiency of prediction tasks related networks such as predicting emerging trends in market or predicting the next superstar in social media.

Additionally, we only experiment on synthetic networks. In the future, real-life networks such as citation networks may be used to provide more practical results. Moreover, to provide insights for fitting models to real-life data, at the same time when the machine-learning classifier identifies the mechanisms behind the growth of networks, it would be interesting to apply regression analysis to find appropriate values for the parameters of the models so that the models fit the networks the best.

## References


[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[2] Niousha Attar and Sadegh Aliakbary. Classification of complex networks based on similarity of topological network features. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27:091102, 09 2017.

[3] Albert-Laszlo Barabasi and Reka Albert. Albert, r.: Emergence of scaling in random networks. science 286, 509-512. *Science (New York, N.Y.)*, 286:509–12, 11 1999.

[4] Ginestra Bianconi and Albert-Laszlo Barabasi. Competition and multiscaling in evolving networks. *EPL (Europhysics Letters)*, 54:436, 05 2001.

[5] Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Anton Krohmer, and Jonathan Striebel. *Towards a Systematic Evaluation of Generative Network Models*, pages 99–114. 05 2018.

[6] Christian Borgs, Jennifer Chayes, Constantinos Daskalakis, and Sebastien Roch. First to market is not everything: an analysis of preferential attachment with fitness. *Proceedings of the Annual ACM Symposium on Theory of Computing*, 10 2007.

[7] Shilpa Chakravartula, Timothy Killingback, Bala Sundaram, and Duc Tran. A statistical construction of power-law networks. *IJPEDS*, 25:223–235, 06 2010.

[8] Colin Cooper, Alan Frieze, and Juan Vera. Random deletion in a scale-free random graph process. *Internet Mathematics*, 1, 01 2003.

[9] Maria Deijfen and Mathias Lindholm. Growing networks with preferential deletion and addition of edges. *Physica A: Statistical Mechanics and its Applications*, 388(19):4297–4303, Oct 2009.

[10] Kamalika Basu Hajra and Parongama Sen. Aging in citation networks. *Physica A: Statistical Mechanics and its Applications*, 346(1):44–48, 2005. Statphys - Kolkata V: Proceedings of the International Conference on Statistical Physics: "Complex Networks: Structure, Function and Processes".

[11] K. Hegde, M. Magdon-Ismail, R. Ramanathan, and B. Thapa. Network signatures from image representation of adjacency matrices: Deep/transfer learning for subgraph classification. *ArXiv*, abs/1804.06275, 2018.

[12] Remco van der Hofstad. *Random Graphs and Complex Networks*, volume 1 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2016.

[13] Kansuke Ikehara and Aaron Clauset. Characterizing the structural diversity of complex networks across domains, 2017.

[14] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 57–65, 2000.

[15] Detect Overfitting. https://corporatefinanceinstitute.com/resources/knowledge/other/overfitting/.

[16] Fragkiskos Papadopoulos, Maksim Kitsak, M. Ángeles Serrano, Marián Boguñá, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489(7417):537–540, Sep 2012.

[17] Piotr Płoński. Random forest feature importance computed in 3 ways with python https://mljar.com/blog/feature-importance-in-random-forest/.

[18] Ryan Rossi and Nesreen Ahmed. Complex networks are structurally distinguishable by domain. *Social Network Analysis and Mining*, 9, 09 2019.

[19] Scikit-learn. Decision tree classifier https://scikit-learn.org/stable/modules/tree.html.

[20] Scikit-learn. Plot_tree https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html.

[21] Scikit-learn. Random forest classifier https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.randomforestclassifier.html.

[22] Azure Machine Learning Studio. Normalize data https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data.

[23] Victor Zhou. Gini impurity https://victorzhou.com/blog/gini-impurity/.

[24] Di Zhuang. Machine-learning methods fordiscovering growth mechanismsin complex networks. 07 2021.

[25] Konstantin Zuev, Marian Boguna, Ginestra Bianconi, and Dmitri Krioukov. Emergence of soft communities from geometric preferential attachment. *Scientific reports*, 5, 01 2015.

# A PA model implementation

```
import numpy as np
from scipy.stats import rv_discrete
import os
from os import path
from multiprocessing import Pool

"""This algorithm generates a network with preferential attachment mechanism.
It uses the PA model introduced in section 8.2 of the book
"RANDOM GRAPHS AND COMPLEX NETWORKS" written by Remco Van Der Hofstad.
For every time t, the model yields a graph of t nodes and mt edges
for some m = 1,2,... We start by defining the model for m = 1 when the graph c
collection of trees. In this case, when t = 1, there is a single node n_1 with
self-loop and, by convention, the degree of one self-loop counts 2.
The evolution of the graph is the same as that in the book.
"""
def pa_sample(id):
    opts = dict(
        # new links
        m=8,
        delta=0,
        # network size
        t=1000,
        # there is no t_0 and lamda for PA model
        t_0=None,
        lamda=None
    )

    # m ranges from 10 to 34 (id from 0 to 24)
    m = opts["m"]
    delta = opts["delta"]
    t = opts["t"]

    # when t = 1, there is a single node n_1 with a single self-loop
    # and the degree of one self-loop counts 2
    nodes = [1]
    edges = [(1, 1)]
    degrees = [2]
    # generate vertices n_2, ..., n_mt and every node comes with one new link.
    for n_index in range(2, m * t + 1):
        total_degree = (n_index - 1) * (2 + delta) + (1 + delta)
        # update nodes list
        nodes.append(n_index)
        # probabilities that the n_index will connect to vertex i
        probabilities = [(degrees[i] + delta) / total_degree
                         for i in range(len(degrees))]
        # probability that n_index will connect to itself
        probabilities.append((1 + delta) / total_degree)
```

```
            # discrete distribution table for each node
            distrib = rv_discrete(values=(nodes, probabilities))
            # new link to node i
            i = distrib.rvs(size=1)[0]
            # update degrees according to the new link
            degrees.append(1)
            degrees[i - 1] += 1
            edges.append((n_index, i))

        # initialize lists to be updated
        collapsed_degrees = [[i + 1, 0] for i in range(t)]
        collapsed_edges = []
        # collapse every m nodes into 1 node.
        # In particular, n1,...,n_m is collapsed into n_1
        newNodes = [i for i in range(1, t + 1)]
        for e in edges:
            # indices of the collapsed nodes
            n1 = int((e[0] - 1) / m) + 1
            n2 = int((e[1] - 1) / m) + 1
            collapsed_edges.append((n1, n2))
            collapsed_degrees[n1 - 1][1] += 1
            collapsed_degrees[n2 - 1][1] += 1
        # save the network as a csv file
        filename = "adjacency/" + "PA-t" + str(t) + "-m" + str(m)
        save_edges(collapsed_edges, path.join(filename, f"{id}_edges.csv"))
        save_degrees(collapsed_degrees, path.join(filename, f"{id}_degrees.csv"))
        append_opts(opts, id, path.join(filename, "0_results.csv"))


# Assign each process different random seed for weighted random choice
def func(seednum):
    rand = np.random.randint(10000)
    np.random.seed(seednum + rand)

import time
if __name__ == "__main__":
    N_samples = 16
    N_processes = 16
    with Pool(processes=N_processes) as pool:
        pool.map(func, range(N_processes))
        pool.map(pa_sample, range(N_samples))
```

# B F model implementation

```
import numpy as np
from scipy.stats import rv_discrete
import os
```

```
from os import path
from multiprocessing import Pool

"""This algorithm generates a network with fitness mechanism. It uses the mode
The fitness of node n_s, denoted by fitness_s, is exponentially distributed
at each time step t, a node n_t arrives with m edges to be connected to the gr
each edge of node n_t is connected to node n_s, s in {1, 2, ..., t}
with probability p_k= fitness_k / sum(fitness of all nodes)
"""


def f_sample(id):
    # lamda: parameter for the exponential fitness distribution
    # t_0: the size of the initial network which are isolated nodes
    # m <= t_0: number of new links per new node have
    # t: number of nodes in the final network
    opts = dict(
        # new links
        m=45,
        # initial size
        t_0=45,
        # network size
        t=1500,
        lamda=1,
        # there is no delta for F model
        delta=None
    )
    m = opts["m"]
    t_0 = opts["t_0"]
    t = opts["t"]
    lamda = opts["lamda"]

    # initialize a fitness list storing the fitness of the initial t_0 nodes
    fitness_list = np.random.exponential(lamda, t_0).tolist()
    nodes = [i + 1 for i in range(t_0)]
    edges = []

    degrees = [[i + 1, 0] for i in range(t_0)]
    # this is used to compute a probability list
    total_fitness = sum(fitness_list)
    # generate nodes t_0 + 1, ... , t and every new node will link to m nodes
    for n_index in range(t_0 + 1, t + 1):
        # Draw a fitness from the exponential distribution
        fitness = np.random.exponential(2, 1)[0]
        # Assign the fitness to the new node
        fitness_list.append(fitness)
        # update the total fitness
        total_fitness += fitness
        # append the new node to the nodes list
```

```
            nodes.append(n_index)
            # probabilities that the n_index will connect to node i
            probabilities = [fitness_list[i] / total_fitness
                             for i in range(len(fitness_list))]
            degrees.append([n_index, 0])

            # discrete distribution table for each node
            distrib = rv_discrete(values=(nodes, probabilities))
            # weighted random choices of m nodes that the new node linked to
            link_list = distrib.rvs(size=m)

            for i in link_list:
                # update degrees according to the new link
                degrees[i - 1][1] += 1
                degrees[n_index - 1][1] += 1
                # add the new edge (n_index, i) to edges list
                edges.append((n_index, i))

        # save the network as a csv file
        filename = "private/" + "F-t" + str(t) + "-m" + str(m) + "-lamda1"
        save_edges(edges, path.join(filename, f"{id}_edges.csv"))
        save_degrees(degrees, path.join(filename, f"{id}_degrees.csv"))
        append_opts(opts, id, path.join(filename, "_results.csv"))


# Assign each process different random seed for weighted random choice
def func(seednum):
    rand = np.random.randint(100)
    np.random.seed(seednum + rand)


if __name__ == "__main__":
    N_samples = 16
    N_processes = 16
    with Pool(processes=N_processes) as pool:
        pool.map(func, range(N_processes))
        pool.map(f_sample, range(N_samples))
```