Group No. : 9

Research Paper Assigned:

Shugang Li, Xuewei Song, Hanyu Lu, Linyi Zeng, Miaojing Shi, Fang Liu. Friend recommendation for cross marketing in online brand community based on intelligent attention allocation link prediction algorithm.

Project Members:

Atmashree Ray (16-1-5-004)
Anagana Borah (16-1-5-072)
Manjit Borah (16-1-5-009)
Ganesh Shah (16-1-5-059)

Contribution

GitHub Link for the code and readme:

https://github.com/manjitborah2710/DevelopmentOf_IAALPA

Member Name	Scholar ID	Contribution
Atmashree Ray	16-1-5-004	Computation of **AAIs** for the networks, **AUCs** of the AAIs for a network, **best index** for a given network, **complementary indices** for a given best index for construction of **training and testing datasets** for DT and SVM models.
Angana Borah	16-1-5-072	Evaluation of **optimal AAI** using Decision Tree (DT), collection of **datasets** for DT training and testing dataset preparation for DT training and testing, computation of **network features** for DT training, implementation of **C4.5 Decision Tree** to select the best AAI from a test dataset
Manjit Borah	16-1-5-009	Evaluation of **complementary AAIs** using Support Vector Machine (SVM) - preparation of dataset for training SVM by calculating similarity indices between AAI pairs of training networks, preparation of data sample from a given network to feed into SVM for classification, preparation of helper code to help get predicted complementary AAIs
Ganesh Shah	16-1-5-059	Computation of **composite AAIs** from optimal and complementary AAIs

Abstract

It has been seen in online brand communities that circle structures develop around different products of a given brand. The existence of such circle structures around products opens up avenues for a brand to cross-market its products between different circles. One way of successfully cross-marketing a product is to establish friendship among users of different products in an online brand community. Traditionally, a brand network tends to be sparse because of the existence of circles. Intelligent attention allocation link prediction algorithm (IAALP A) adaptively builds attention allocation index (AAI) according to the sparseness of the network and predicts the possible friendships between users in different circles. AAIs of both the direct common friends and indirect ones are used to overcome the network sparsity problem. A decision tree (DT) is used to screen the best AAI for a given network. Support vector machine model identifies the complementary AAIs of the selected AAI which are finally combined to generate a composite AAI for a given network. The composite AAI comprehensively portrays the attention distribution of common friends of users in different circles and predicts their possible friendships for cross-marketing activities.

1. Introduction

The use of the Internet today is growing exponentially, with high speed online services like media streaming services, online video games and online shopping. 80% of the top 500 companies in the world have established online brand communities. Online brand communities aim at bringing the scattered population together by targeting specific customers and always maintain a strong and lasting relationship with these customers[5]. These communities perform various marketing activities which involve important decisions like product evaluation, purchase possibility[6] etc. to be taken by customers. These decisions are hugely affected by the friend group they are a part of. It was also found that the influence of a group is inversely proportional to the personal user requirements[7]. In online brand communities, members who share a common interest in product preference are gathered together in subgroups, called circles[8]. Circle structure provides fertile ground for cross-marketing, where sales are amplified with the influence of friends through forming friendships between users in different circles.

One of the main approaches for link prediction between node pairs is Scoring Link Prediction Algorithm (SLPA). It is used for friend recommendation between the user nodes based on network topology. But, the limitation to SLPA is that it does not take into account friendship establishment among users in different circles. Moreover, not every network circle architecture can work with the same SLPA, i.e. no standard SLPA has been developed which can accommodate all network circles. Therefore, there is a high requirement of building a method that relied solely on the network data to build an appropriate SLPA.

These problems are addressed by the proposed algorithm, Intelligent Attention Allocation Link prediction algorithm (IAALP A). It predicts friendship possibilities among users in different circles by building Attention Allocation Index (AAI) adaptively for certain circles. The AAI refers to the amount of attention assigned to a particular user pair by their common friend in the triadic closure structure. This is determined by the number of friends of common friends. It is specifically useful for the purpose of overcoming the network sparsity problem in detecting possible friendships between users in two different circles. This is implemented by developing AAIs of both the direct common friends and indirect friends based on the principle of attention allocation of common friends in triadic closure structure. It is tough to build an AAI that fits all structural characteristics of network circles, so Decision Tree (DT)[9] is used to first adaptively predict the AAI in different circles. The next step is to implement Support Vector Machine (SVM)[10] to find the complementary AAIs of the AAIs selected by DT. This improves the overall AAI performance. Lastly, the selected AAI and its complementary ones are used to design the composite mutually complementary AAI to comprehensively portray the attention allocated to user node pairs by their common friends and forecast the possible connections between user nodes in different circles. Consequently, the friendships between users in different circles are recommended and successfully cross-marketing is achieved.

2. Theoretical Foundation

In this section, we discuss the several prerequisites that lead to the IAALP algorithm.

2.1 Link Prediction

A social network is a social structure made up of a set of social actors (such as individuals or organizations), sets of relational ties between the actors, and other social interactions between them.[1] Social networks tend to be dynamic in nature, meaning that their structure changes over time. One important change in a network is the formation of new relations between two social actors. As new links are formed, the network gets denser and more strongly connected. Link prediction is the study of a network to predict the future links that might be formed in a given network. Link prediction can also be used to uncover hidden links in a network. The importance of link prediction has been observed in a plethora of fields including bibliographic domain, molecule biology, criminal investigations and recommending systems.[2] The problem statement of link prediction can be formally defined as follows:

"Given a snapshot of a social network at time ***t****, we seek to accurately predict the edges that will be added to the network at time* ***t + dt*** *by defining a similarity or a probability index."[3]*

There are different kinds of link prediction algorithms based upon the various properties of a network. The different kinds of link prediction algorithms are shown in Fig 2.1 with a special focus on the types of algorithms used in this study. This study has used neighbourhood-based algorithms which fall under the algorithms that study the topological similarities of a network. The similarity measure using topological features focuses on the network structure to compute the similarity scores. These algorithms take a node's immediate neighbours into account. Topological measures are more common since they do not need definition of rich features to describe content and are more general. The neighbourhood-based methods use the number of common neighbours to predict a link between two nodes. Larger the number of common neighbours, larger is the likelihood of link formation. Although the IAALP algorithm is also a neighbourhood-based algorithm, it is unique in the sense that this index takes into account not only the direct neighbours of a node, but also the indirect neighbours of a node, i.e, the neighbours of a neighbour.

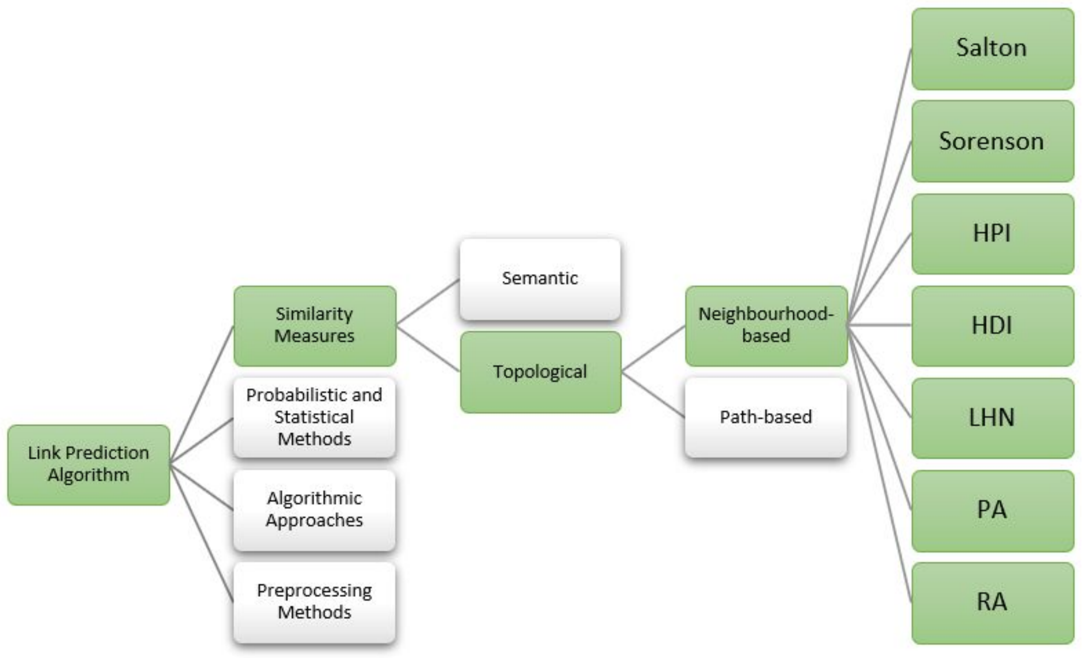


Fig 2.1: Types of Link Prediction Algorithm with the algorithms used in current study highlighted in green

It has been found that the main approach to predict the existence of links between two nodes in a network and recommend friendship between two nodes based on network topology is Scoring Link Prediction Algorithm (SLPA). However, since SLPA is not sufficient to fully characterize the network, the combined link prediction algorithms are developed. These algorithms focus on versatility while IAALP algorithm is more focused on circle structures in a network. IAALP is a methodology which strives to recommend friends in different social networks. IAALP algorithm recommends the user group in one product circle to the target user in another product circle, and the influence of the friend group is used to affect the purchase decision of the target user, in order to realize cross-selling.

2.2 Friend Group Recommendation for Cross-Marketing

An online brand community can be represented using a graph data structure as D(V,E) where D is a graph having a set of nodes V representing the users and a set of edges E representing the friendship ties between the different users. In such online brand communities, people tend to form groups around the product that they use. These groups are called circles. This is shown in Fig 2.2. Different circles emerge around the different products a brand has to offer. Let us suppose that users in the product circle of B purchase the product A. Then, all the members of product circle A can be recommended as friends to the users of product circle B. This is shown in Fig 2.3a and Fig 2.3b. If node 2 which is in product circle A buys product B, then all the nodes in product circle B can be recommended to node 2 as potential friends. As nodes in the same circle are more likely to be connected than nodes in different circles, the network is often sparse. Hence, SLPA cannot guarantee to correctly predict the links but IAALP algorithm can, since it is designed to address the network sparsity issue. This is done by creating mutually complementary AAI for the specific circle structure.

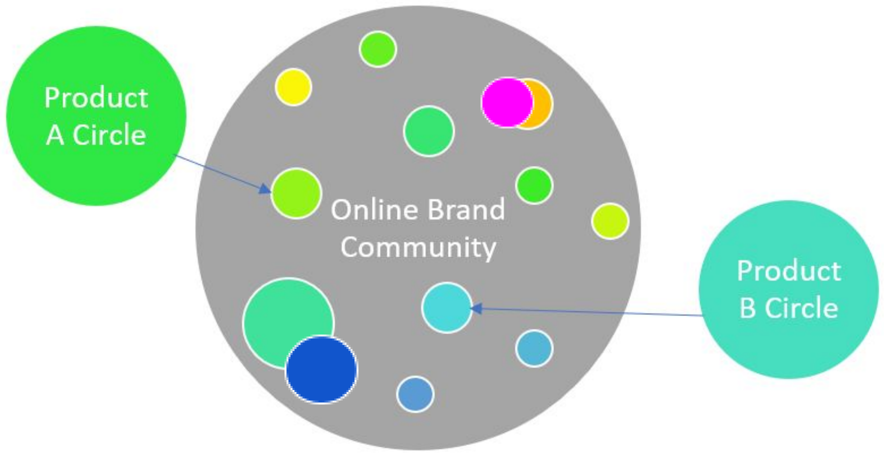


Fig 2.2: Online Brand Community with different Product Circles

The different attention allocation indices (AAIs) that have been used can be categorized based on the point of view of construction. We have AAIs based on microstructure which are computed using node pairs and their neighbours and AAIs based on macrostructure that are computed using node pairs, their common neighbours and the friends of the common neighbours. The list of AAIs with their formulae are given in Table 2.1.

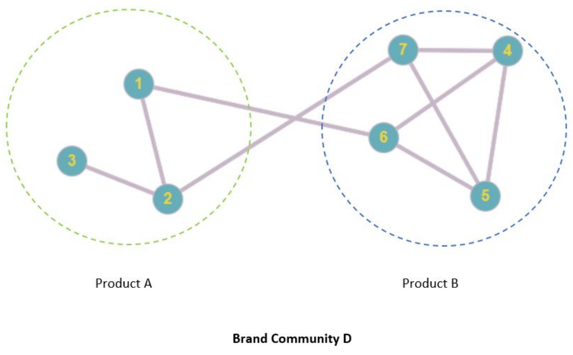


Fig 2.3a: Social Network

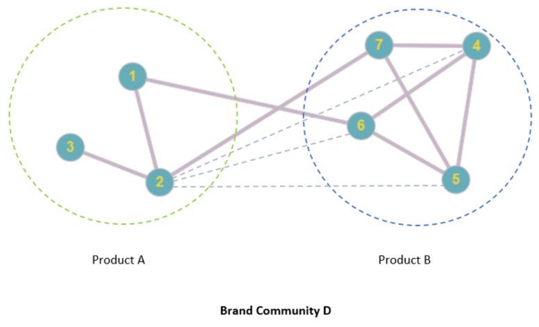


Fig 2.3b: Social Network with Predicted Links

AAI	Formula
Salton	$S_{xy}^{Salton} = \frac{\lvert\Gamma(x) \cap \Gamma(y)\rvert}{\sqrt{k(x) * k(y)}}$
Sorenson	$S_{xy}^{Sorenson} = \frac{2*\lvert\Gamma(x) \cap \Gamma(y)\rvert}{k(x)+ k(y)}$
HPI	$S_{xy}^{HPI} = \frac{\lvert\Gamma(x) \cap \Gamma(y)\rvert}{\min(k(x),k(y))}$
HDI	$S_{xy}^{HDI} = \frac{\lvert\Gamma(x) \cap \Gamma(y)\rvert}{\max(k(x),k(y))}$
LHN	$S_{xy}^{LHN} = \frac{\lvert\Gamma(x) \cap \Gamma(y)\rvert}{k(x) * k(y)}$
PA	$S_{xy}^{PA} = k(x) * k(y)$
RA	$S_{xy}^{RA} = \Sigma_{z\in \Gamma(x) \cap \Gamma(y)} \frac{1}{k(z)}$
RAA	$S_{xy}^{RAA} = \Sigma_{z\in \Gamma(x) \cap \Gamma(y)} \frac{\max(k(x),k(y))}{k(z)}$
WA1	$S_{xy}^{WA1} = \Sigma_{\Gamma(z), z\in \Gamma(x) \cap \Gamma(y)} \frac{1}{k(\Gamma(z))}$
WA2	$S_{xy}^{WA2} = \Sigma_{z\in \Gamma(x) \cap \Gamma(y)} \frac{1}{k(z)*(\rho*c(z) + \phi*(1-c(z)))}$
WA3	$S_{xy}^{WA3} = \Sigma_{z\in \Gamma(x) \cap \Gamma(y)} \left(\frac{1}{k(z)*(1-c(z))} + \frac{1}{k(x)*c(x)} + \frac{1}{k(y)*c(y)} \right)$
RWA	$S_{xy}^{RWA} = \alpha*S_{xy}^{RA} + \beta*S_{xy}^{WA2} + \gamma*S_{xy}^{WA3}$

Table 2.1: AAIs and their Formulae. The first 8 are based on microstructure while the last 4 are based on macrostructure. Γ(·) represents the neighbor set of a node, k(·) denotes the degree of a node, c(·) represents the clustering coefficient of a node while α, β, γ, ρ, and φ are constants.

2.3 Intelligent Attention Allocation Link Prediction Algorithm

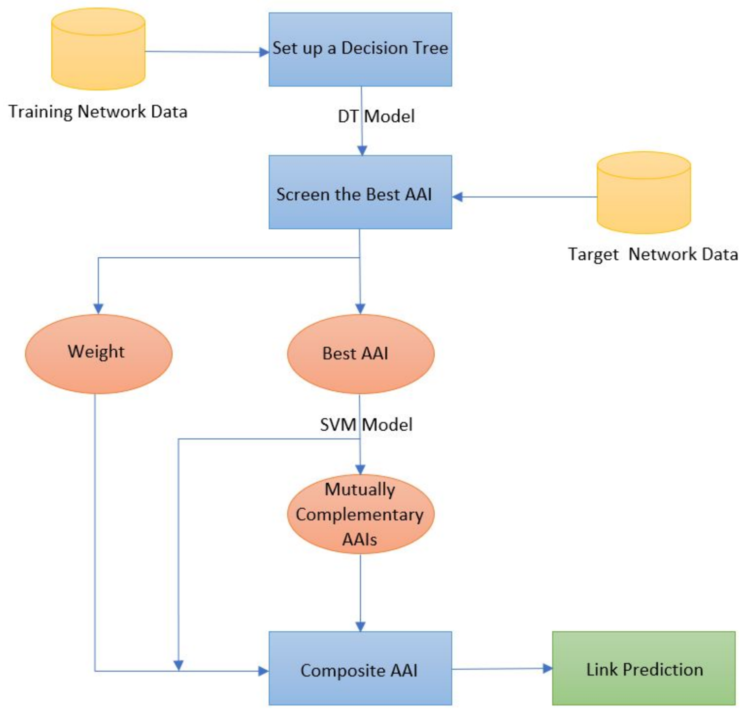


Fig 2.3: The Structure of IAALP Algorithm

The Intelligent Attention Allocation Link Prediction algorithm (IAALP A) is a novel algorithm proposed by Li, et.al., (2019). The IAALP algorithm takes into account the combined effect of various features of the network as well as the prediction performance of the algorithm. To find the most suitable AAI for a given network, the IAALP A uses a decision tree to find the index with the best performance. The best index is based upon the network characteristics pertaining to the common neighbours and their characteristics. It has been seen that a single AAI may overestimate or underestimate the predicted links and random combination of AAIs might not provide the optimal solution. Hence, a support vector machine is then used to find all the complementary indices for the network. The best index along with the complementary indices are then ultimately combined together to perform link prediction.

The performance of various AAIs is measured using the area under the curve (AUC). AUC is one of the most common standard metrics of measuring the accuracy of a link prediction algorithm. AUC is often calculated by random sampling of node pairs. This is done so as to reduce the computation complexity, thereby increasing the efficiency of the process. An attention allocation index (AAI) is considered better if the AUC value is more for that index. Typically, AUC is calculated using the formula:

$$AUC = \frac{m1 + 0.5*(m - m1)}{m}$$

where m is the size of the sample, m1 is the number of instances for which AAI of a connected node pair is greater than that of the unconnected node pair in a sample point.

Despite being a popular metric, AUC has its own disadvantages when it comes to evaluating AAIs for certain networks. AUC is a general method that does not take into account the specifics of a network. More precisely, AUC often overlooks the characteristics of the specific circle structure of the network. In order to find the best AAI using AUC, we must compute the AUC value for all the AAIs. This task is time consuming and therefore, not much feasible. A better solution is to generate a decision tree. A decision tree adaptively calculates the best attention allocation index for a given network with a specific circle structure without having to explicitly calculate and compare all the AAIs. This is done by focusing on the density of common neighbours and the dispersion level of common neighbours' attention.

Support vector machine models can map the input vector to a high dimensional eigenvector space, and construct the optimal classification surface in the eigenvector space. This property is used to identify all the complementary AAIs of the best AAI that is computed by the decision tree. An index is said to be complementary to the best index if their combined AUC is greater than the AUC of the best index. A SVM model classifies an index to be complementary to the best index by considering the score similarity of the two indices based on distance, similarity in direction and the difference in performance ranking.

After the selection of the best index and the identification of the complementary indices, these indices must be combined together to comprehensively portray the attention distribution of common friends of users in different circles. The composite index is computed as:

$$S = w * (B, E_1, E_2, \cdots, E_h)^T$$

where w is the average AUC value of each AAI obtained in training the DT model, B is the best index selected by DT and E_1, E_2, ···, E_h are the complementary AAIs of B identified by the SVM.

3. Specification and Design

The modelling of the IAALP algorithm is completed in four steps. The first step involves the computation of the different attention allocation indices and their AUC values. The second step involves training a decision tree to select the best AAI for a network. In the third step, SVM is used to classify AAIs that are complementary to the best AAI. The fourth step ultimately computes the composite AAI by combining the best AAI and its complementary AAIs based on their weights. In this section, we discuss in detail the design of the IAALP algorithm.

3.1 Algorithm Evaluation and Complementary AAI

For measuring the accuracy of SLPAs, the most common standard metric used is area under the curve (AUC). AUC selects a connected node pair and an unconnected node pair at random and makes a comparison of their AAI score values.
It is given by the formula[3]:

$$AUC = \frac{m1 + 0.5*(m - m1)}{m}$$

where *m* refers to independent comparisons and *m1* refers to the number of the times the score of a connected pair is higher than that of an unconnected pair.

The AUC value obtained by this random sampling method reduces the computation complexity and improves the efficiency when the network size is large. Thus, the accuracy of the algorithm is directly proportional to the value of the AUC.

The complementary AAI for a given best index in a network is selected using the following algorithm:

```
a) Let auc_B = AUC of best index b.
b) Let idx = set of all indices
c) Let cmp_idx = set of complementary indices
d) For i in idx
    i) If AUC(i + b) > auc_b
        1) Append i to cmp_idx
e) Return cpm_idx, the set of complementary indices
```

3.2 DT for screening AAI

Although AUC is most commonly used for evaluating AAI, it is not compatible with various circle structures. Therefore, all AAIs are required to be tried before finding the best AAI for a particular network. DTs are good for multi-class problems. Thus, they are used to adaptively select the best AAI for a specific circle, while lowering the effort required for data preparation[11].

The first step in preparing the data for DT training is calculation of common friend density features and dispersion level of common friends' attention features. Density of common neighbors is required since common neighbors indicate more attention allocated. Two features relating to common neighbors' density are calculated, namely average degree and average clustering coefficient. Next, the dispersion level of common friends' attention is considered. Since higher the number of short paths connecting common neighbors mean higher the attention dispersion, short path related features, namely, average shortest path, average node betweenness and average link betweenness are calculated. These features are independent variables in the DT.

The dependent variable to be used in the DT is the AAI with the maximum AUC value. The AUC of AAI for each network in the training set is calculated.

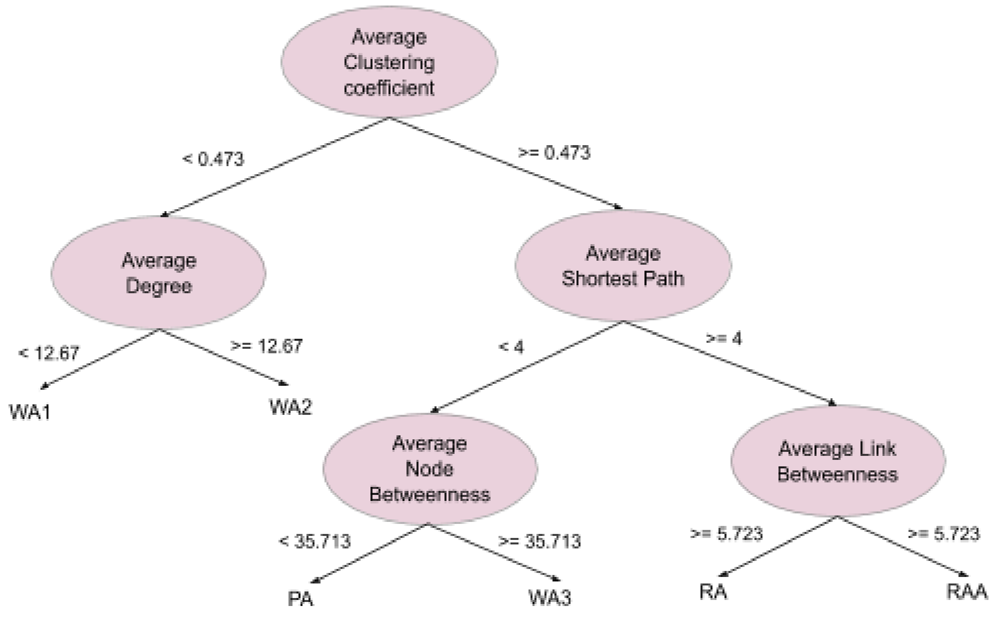


Fig 3.1: An example of DT

For designing the DT, the **C4.5 algorithm**[11] is used. It is a simple and a practical learning algorithm. It is one of the most popular algorithms used for decision tree creation. Assuming that the network training set is *T*, in each sample, the network is labeled by the AAI with the largest AUC. Given that there are *k* types of AAIs, a division of *T* is obtained as $\{S_1, S_2, ..., S_k\}$.

- The prior probability of division is $P_i = |S_i|/|T|$.
- The information entropy used for classifying *T* is $Info(T) = -\sum_{i=1}^{k} P_i log_2 P_i$.

The networks in T are divided by feature A, and the sequence $\{A_1, A_2, .., A_J\}$ is obtained by arranging the values of feature A in ascending order. Defining the ith $(1 \leq i \leq J - 1)$ partition point as $a_i = (A_i + A_{(i+1)})/2$, T is divided into 2 subsets $\{T_1, T_2\}$, where the value of feature A of the networks in T_1 is $V(A,T_1) \in [A_1, a_i]$, and similarly in T_2 is $V(A,T_2) \in (a_i, A_J]$.

- Corresponding to this kind of division, the information gain of feature A is $Gain(A) = Info(T) - Info_A(T)$, where $Info_A(T) = \sum_{i=1}^{2} (|T_i|/|T|)\, Info(T_i)$.
- Corresponding to partition point a i , the information gain rate of A is:

  $$Gain\_Ratio(A, a_i) = Gain(A)/Split(A)$$

  where, $Split(A) = -\sum_{i=1}^{2} (|T_i|/|T|) log_2 (|T_i|/|T|)$

- The information gain rate for each partition point in sequence {A1, A2,.., AJ} is calculated and the partition point with the maximum gain rate is selected as the best branch threshold of the feature A , namely, $Threshold(A) = max(1 \leq i \leq J-1)\{Gain\_Ratio(A,a_i)\}$.

The C4.5 DT has many advantages. It inherently applies Single Pass Pruning process to mitigate overfitting of data. C4.5 DTs work very well with continuous as well as discrete data and can also handle the exception of incomplete data.

3.3 Selecting complementary AAIs based on SVM

Although the best AAI can be selected using the DT method, the combination of the selected AAI and its complementary AAI can fully capture the attention allocation characteristics of nodes in different circles. The complementary AAI of an index X is an index whose combination with X generates a better AUC value than that X has. Therefore, for computing the complementary AAIs, SVM is used. SVM is used because it avoids overfitting which may be caused by a sufficiently diverse network circle. SVM is also known for solving small-sample, non-linear and high-dimensional pattern recognition problems efficiently. Thus, SVM is adopted to find the AAIs that are complementary to the best AAI selected by the DT model[12].

- For the SVM model, a training set $[I_i, M_i]$ $(i = 1, 2, \ldots, l)$ is prepared.
  The input vector I_i is a combination of 7 similarity indices for the scores of two sample AAI in sample *i*. The similarity score is described from three viewpoints to fully capture the complementary relationship:
  - Distance between scores of the two sample indices : For this, distances used are: Euclidean distance, Standard Euclidean distance, Manhattan distance and Chebyshev distance
  - Similarity with respect to direction: For this, distances used are: Cosine distance and Pearson correlation coefficient
  - Difference in performance ranking: For this, Spearman distance.is used

  The value of M_i is 1 if two AAIs are mutually complementary, else -1.
- Next, input to a high dimensional eigenvector space mapping is implemented. For kernel function, Radial Basis Function (RBF) is used. It is given by $K(I_i, V) = exp(-||V-I_i||^2 / \delta^2)$, where V is the input vector, and δ is constant. Use of RBF improves the efficiency of the algorithm.
- An optimal classification function is obtained by:

$$f(v) = sgn\{(W \,.\, \phi(V)) + b\} = sgn(\sum_{i=1}^{l} (a_i M_i K(I_i, V) + b)$$

  where *a* and *b* are constants, *W* is a normal vector which determines the hyperplane direction for classification and *b* is the displacement term indicating the distance between the hyperplane and the origin. $\phi(V)$ is the eigenvector after mapping input *V*.

3.4 Composite AAI

This is the final step of the algorithm. IAALP A build a composite mutually complementary AAI for displaying attention distribution among among common friends in different user circles.

This composite index model consists of the following:

- The best index B selected by the DT model
- The complementary AAIs of B identified by SVM, i.e. $E_1,E_2,...,E_i$.

The average AUC value *w* of each AAI obtained by DT model is computed, which is referred to as the weight. This is performed to increase the weightage of an AAI with a good performance account in the composite AAI.

Additionally, *h* excellent complementary AAIs with larger weights are selected. This is done to improve the efficiency of the algorithm, given by the formula:

$S = w*(B,E_1,E_2,...,E_h)^T$

4. Implementation

The egonet datasets in Twitter were used to make datasets during the implementation. [4]

4.1 Finding the AAIs for the networks and their AUCs

The eight different attention allocation indices based on the microstructure of the network are **Salton**, **Sorenson**, **HPI**, **Hub Depressed Index (HDI)**, **Leicht–Holme–Newman (LHN)**, **Preferential Attachment (PA)**, **RA**, and **Resource Allocation Average (RAA)**. The four indices based on macrostructure of the network that have been developed innovatively are **WA1**, **WA2**, **WA3**, and **RWA**. The value for each of these indices can be computed using the formulae of Table 2.1. With these formulae in mind, a python script has been written to compute the values of these indices. A small part of the script is shown in Fig 4.1.1. Each function takes in a node pair and a networkx graph as mandatory inputs. The function *wa2* additionally takes as input the constant values *rho* and *phi*. The function *rwa* additionally takes as input the constant values *a,b,c,rho,phi.* All the inputs are entered in the order of their mention.

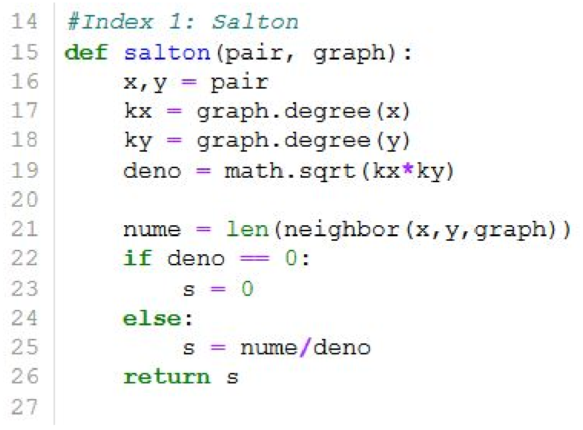

```
#Index 1: Salton
def salton(pair, graph):
    x,y = pair
    kx = graph.degree(x)
    ky = graph.degree(y)
    deno = math.sqrt(kx*ky)

    nume = len(neighbor(x,y,graph))
    if deno == 0:
        s = 0
    else:
        s = nume/deno
    return s
```

Fig 4.1.1: Code snippet showing implementation of computation of an AAI

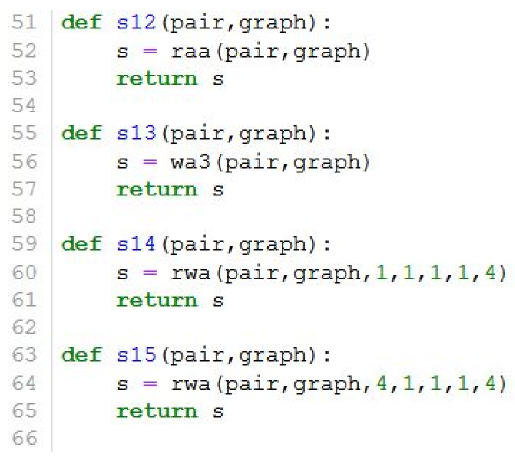
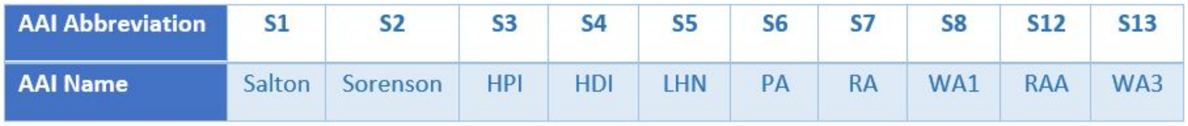

```
def s12(pair,graph):
    s = raa(pair,graph)
    return s

def s13(pair,graph):
    s = wa3(pair,graph)
    return s

def s14(pair,graph):
    s = rwa(pair,graph,1,1,1,1,4)
    return s

def s15(pair,graph):
    s = rwa(pair,graph,4,1,1,1,4)
    return s
```

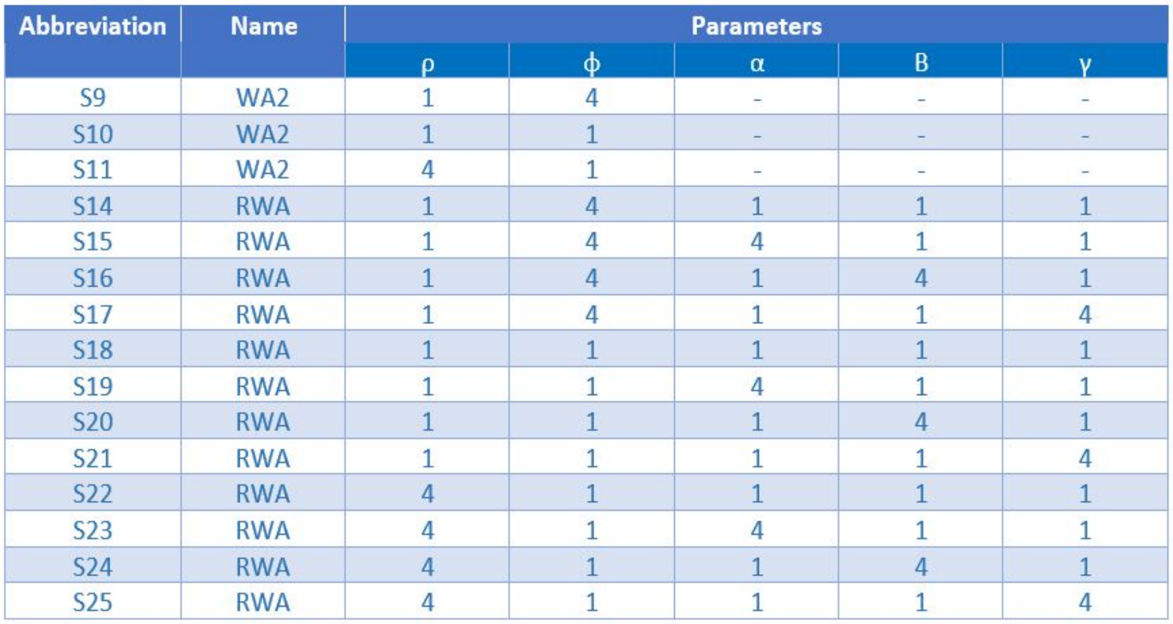
Fig 4.1.2: Code snippet showing the computation of the abbreviated indices

As is defined in the theory, there are 25 different indices which have been used in the IAALP algorithm. These 25 indices are the derivatives of the above mentioned 12 indices. Particularly, two of the four innovatively designed indices based on the macrostructure of the network as associated with constant values (also called parameters). Those two indices are *wa2* which has two parameters ρ and φ, and *rwa* which is basically a weighted combination of wa2, wa3 and ra, and hence, has five parameters α, β, γ, which are the weights and ρ and φ for *wa2.* In order to compute these 25 indices directly, another script is written to directly compute their values. Table 4.1.1a and Table 4.1.1b show the naming convention for every index while Fig 4.1.2 shows a small part of the implementation. The computation of all of these indices is necessary to prepare the training and testing dataset for both the Decision Tree as well as the Support Vector Machine models.



AAI Abbreviation	S1	S2	S3	S4	S5	S6	S7	S8	S12	S13
AAI Name	Salton	Sorenson	HPI	HDI	LHN	PA	RA	WA1	RAA	WA3

Table 4.1.1a: AAIs without Parameters



Abbreviation	Name	Parameters				
		ρ	ϕ	α	B	γ
S9	WA2	1	4	-	-	-
S10	WA2	1	1	-	-	-
S11	WA2	4	1	-	-	-
S14	RWA	1	4	1	1	1
S15	RWA	1	4	4	1	1
S16	RWA	1	4	1	4	1
S17	RWA	1	4	1	1	4
S18	RWA	1	1	1	1	1
S19	RWA	1	1	4	1	1
S20	RWA	1	1	1	4	1
S21	RWA	1	1	1	1	4
S22	RWA	4	1	1	1	1
S23	RWA	4	1	4	1	1
S24	RWA	4	1	1	4	1
S25	RWA	4	1	1	1	4

Table 4.1.1b: AAIs with Parameters

The indices that have been mentioned are computed for node pairs in a network. The performance of an index is commonly measured by area under the ROC curve, abbreviated as AUC. The computation of AUC is possible in many ways. However, in the theory, the preferred method of computation is random sampling of connected and unconnected nodes. This has been discussed at length in the preceding sections. For implementation of this module, another python script was written. This script has several utility functions but three of them deserve special attention:

a) **getNetworkAUC(csvfile)**

The AUC values are computed for each of the 25 indices for a given network. This function is used by the Decision Tree to get the value of each indices and forward these values as the weight vector for the computation of the composite index. The implementation is shown in Fig 4.1.3.

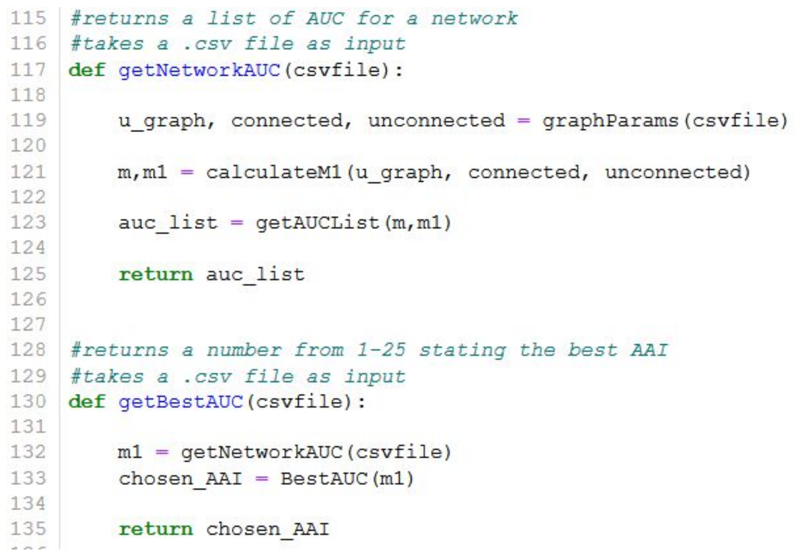
b) **getBestAUC(csvfile)**

Among the 25 indices, there exists one index that has the highest value for the given network. This function attempts to identify and return the best index i.e, the index with the highest AUC value. It should be noted that in case AUC values of two or more indices are exactly the same, the best index is picked at random from those indices. This happens since AUC is computed via random sampling. Although repeated calculation of AUC is possible until the best AUC is found, that would be time consuming and hence, the former is preferred. This best index is used to prepare training data for the Decision tree. This implementation is shown in Fig 4.1.3.

c) **getComplementaryAAI(csvfile, best_index)**

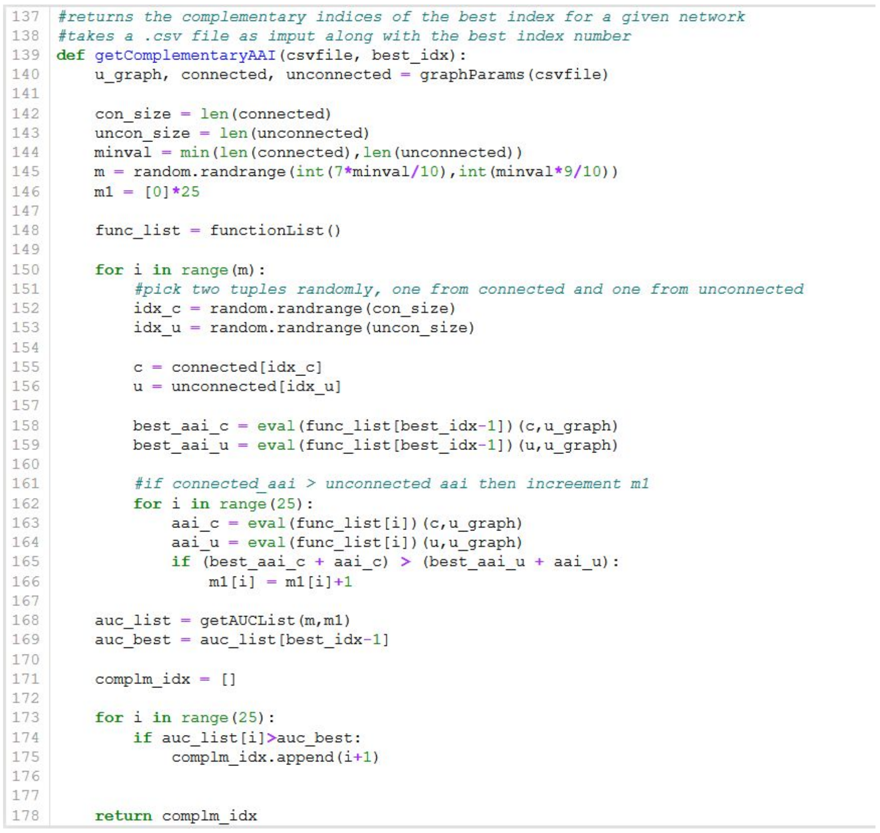
Once a best index is picked, the complementary indices for the best index can be calculated by comparing the AUC values of the best index and the combination of the best index with another index. The calculation process has been discussed in the previous sections. This is used while preparing training data for SVM. The implementation is shown in Fig 4.1.4.

Apart from the above mentioned functions, there are several other useful functions in that script. The function *functionList()* generated a list of the abbreviations of the indices, *getEdgelist(graph)* and *getUnconnectedPairs(graph,edgelist)* generate the connected and unconnected node pairs of a network. The scripts written are a series of helper codes for construction of training and testing datasets for both the Decision Tree as well as the Support Vector Machine Models. The scripts are not intended to function as standalone units, rather they are meant to be used in other driver functions.



```python
#returns a list of AUC for a network
#takes a .csv file as input
def getNetworkAUC(csvfile):

    u_graph, connected, unconnected = graphParams(csvfile)

    m,m1 = calculateM1(u_graph, connected, unconnected)

    auc_list = getAUCList(m,m1)

    return auc_list


#returns a number from 1-25 stating the best AAI
#takes a .csv file as input
def getBestAUC(csvfile):

    m1 = getNetworkAUC(csvfile)
    chosen_AAI = BestAUC(m1)

    return chosen_AAI
```

Fig 4.1.3: Code snippet to get the AUC values of the AAIs for a network and the Best Index

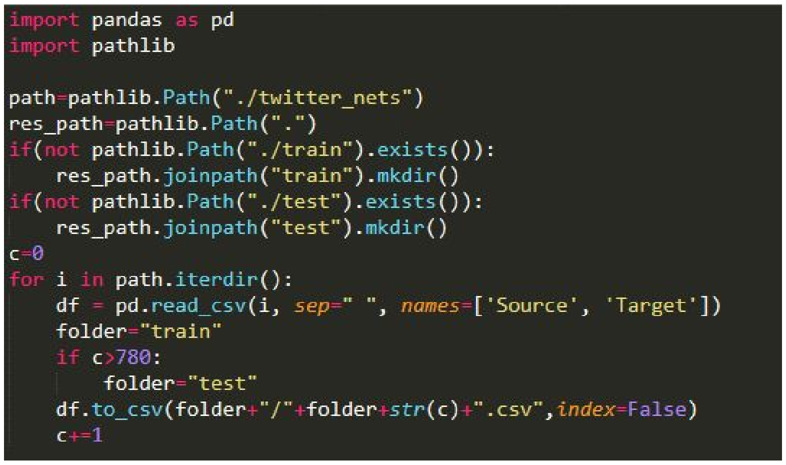


```python
#returns the complementary indices of the best index for a given network
#takes a .csv file as imput along with the best index number
def getComplementaryAAI(csvfile, best_idx):
    u_graph, connected, unconnected = graphParams(csvfile)

    con_size = len(connected)
    uncon_size = len(unconnected)
    minval = min(len(connected),len(unconnected))
    m = random.randrange(int(7*minval/10),int(minval*9/10))
    m1 = [0]*25

    func_list = functionList()

    for i in range(m):
        #pick two tuples randomly, one from connected and one from unconnected
        idx_c = random.randrange(con_size)
        idx_u = random.randrange(uncon_size)

        c = connected[idx_c]
        u = unconnected[idx_u]

        best_aai_c = eval(func_list[best_idx-1])(c,u_graph)
        best_aai_u = eval(func_list[best_idx-1])(u,u_graph)

        #if connected_aai > unconnected aai then increement m1
        for i in range(25):
            aai_c = eval(func_list[i])(c,u_graph)
            aai_u = eval(func_list[i])(u,u_graph)
            if (best_aai_c + aai_c) > (best_aai_u + aai_u):
                m1[i] = m1[i]+1

    auc_list = getAUCList(m,m1)
    auc_best = auc_list[best_idx-1]

    complm_idx = []

    for i in range(25):
        if auc_list[i]>auc_best:
            complm_idx.append(i+1)


    return complm_idx
```

Fig 4.1.4: Code snippet to get all the complementary AAIs for a given best index

4.2 Finding the optimal AAIs using Decision Tree

Decision Trees (DT) are used for multi-class classification purposes. In IAALP A, C4.5 DT is used for selecting the best AAI for a network circle. DT uses some network features as independent variables and the AAI with maximum AUC value as the dependent variable.
The twitter ego-net dataset's files ending with .edges extension is used. They are first converted into .csv files and split into train and test dataset using the following code:



```python
import pandas as pd
import pathlib

path=pathlib.Path("./twitter_nets")
res_path=pathlib.Path(".")
if(not pathlib.Path("./train").exists()):
    res_path.joinpath("train").mkdir()
if(not pathlib.Path("./test").exists()):
    res_path.joinpath("test").mkdir()
c=0
for i in path.iterdir():
    df = pd.read_csv(i, sep=" ", names=['Source', 'Target'])
    folder="train"
    if c>780:
        folder="test"
    df.to_csv(folder+"/"+folder+str(c)+".csv",index=False)
    c+=1
```

Fig. 4.2.1 Code for train-test split and edges-csv conversion

Preparation of the Train Dataset

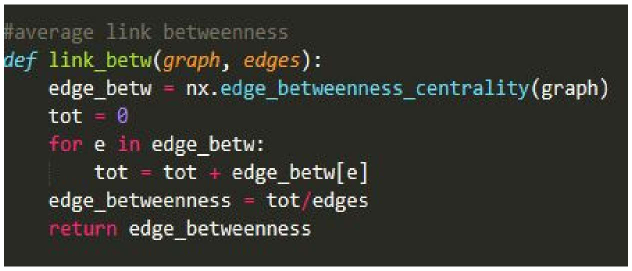
The DT model is trained by preparing a dataset of the form *[Features$_i$, Decision$_i$]* where *i* stands for each network circle in the training set. The *Features$_i$* field contains features which are used as independent variables. They are:

1. Average Degree
2. Average Clustering Coefficient
3. Average Shortest Path
4. Average Node Betweenness
5. Average Link Betweenness

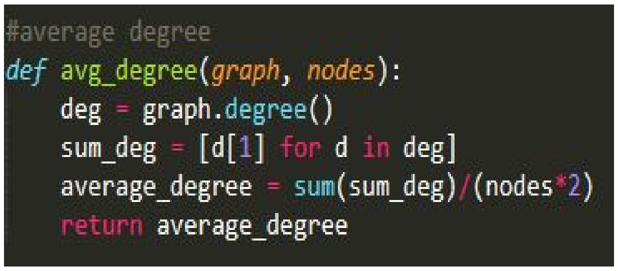
Features	Formulae
Average Degree	$(\sum_{i} degree\ of\ node\ i)/n$
Average Clustering Coefficient	$(\sum_{i}((number\ of\ neighbor\ pairs\ of\ i\ that\ are\ connected)/(number\ of\ neighbor\ pairs\ of\ i)))/n$
Average Shortest Path	$(\sum_{u,\ v\ \in nodes} d(u,\ v))/(n * (n-1))$
Average Node Betweenness	$\sum_{i}((\sum_{s,\ t\ \in nodes\ and\ (s\ and\ t\ are\ different)} ust(node\ i)/ust))/n$
Average Link Betweenness	$\sum_{i}((\sum_{s,\ t\ \in nodes\ and\ (s\ and\ t\ are\ different)} ust(edge\ i)/ust))/e$

Table 4.1.1. Network Features*, where n = number of nodes, d(u, v) = distance between node pair u-v, ust = number of shortest paths betweens s and t, ust(edge i) = number of shortest paths between s and t having edge i, ust(node i) = number of shortest paths between s and t consisting of node i*

Features 1 and 2 are grouped under common friends' density features. Features 3, 4 and 5 are grouped under common friends' attention dispersion level features. The above features are computed for each network circle in the train dataset. Table 4.1.1 lists the above features with their formulae. Fig 4.2.2 shows code snippets for calculation of average link betweenness and average degree. The *networkx* python library is used to compute the features.



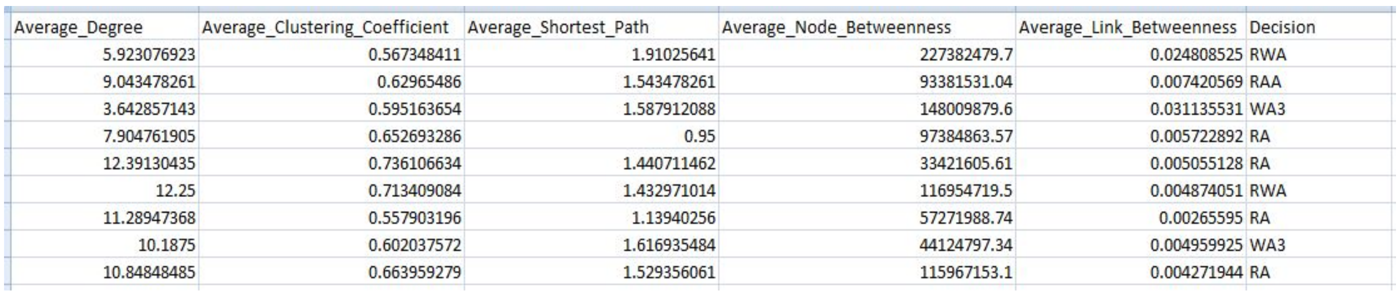
```
#average link betweenness
def link_betw(graph, edges):
    edge_betw = nx.edge_betweenness_centrality(graph)
    tot = 0
    for e in edge_betw:
        tot = tot + edge_betw[e]
    edge_betweenness = tot/edges
    return edge_betweenness
```



```
#average degree
def avg_degree(graph, nodes):
    deg = graph.degree()
    sum_deg = [d[1] for d in deg]
    average_degree = sum(sum_deg)/(nodes*2)
    return average_degree
```

Fig 4.2.2 Code snippets for a) link betweenness and b) average degree

The *Decision$_i$* field contains AAI having the best AUC value. The AAI which contains the best AUC value is computed in the first stage. A snapshot of the train dataset is shown below.



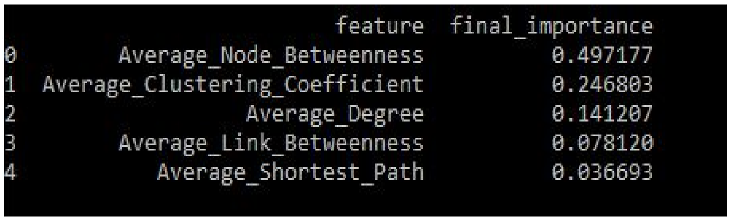
Average_Degree	Average_Clustering_Coefficient	Average_Shortest_Path	Average_Node_Betweenness	Average_Link_Betweenness	Decision
5.923076923	0.567348411	1.91025641	227382479.7	0.024808525	RWA
9.043478261	0.62965486	1.543478261	93381531.04	0.007420569	RAA
3.642857143	0.595163654	1.587912088	148009879.6	0.031135531	WA3
7.904761905	0.652693286	0.95	97384863.57	0.005722892	RA
12.39130435	0.736106634	1.440711462	33421605.61	0.005055128	RA
12.25	0.713409084	1.432971014	116954719.5	0.004874051	RWA
11.28947368	0.557903196	1.13940256	57271988.74	0.00265595	RA
10.1875	0.602037572	1.616935484	44124797.34	0.004959925	WA3
10.84848485	0.663959279	1.529356061	115967153.1	0.004271944	RA

Fig 4.2.3 Train dataset

C4.5 DT building

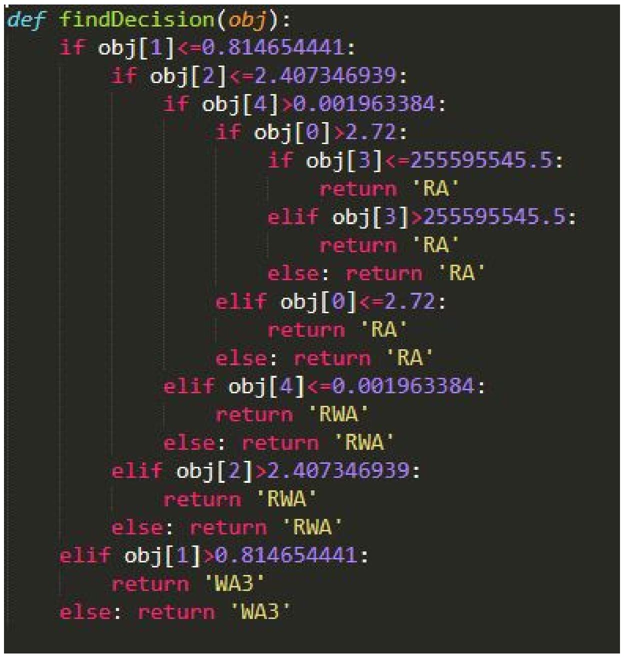
The C4.5 DT is used since it has many advantages as discussed earlier. The C4.5 DT is an improvement over the ID3 DT in ways such as handling numerical and nominal features values and target labels. It uses entropy and gain ratio to find the decision points in a decision tree, as discussed in detail in the "Specification and Design" section.

The C4.5 DT is implemented using the Chefboost framework. Here the dataset is imported as pandas dataframe. *model.fit(df, config)* is used to build the DT model and calculate the accuracy on the training data. The feature importances are also computed using the *model.feature_importance()* function. This ranks the features used according to their order of consideration in the DT. A snapshot of the feature importance table for a sample DT generated is shown in Fig. 4.2.4 The built DT is in the form of python 'if-statements' in the "output/rules" directory. A sample built DT is shown in Fig 4.2.5.



```
                         feature  final_importance
0        Average_Node_Betweenness          0.497177
1  Average_Clustering_Coefficient          0.246803
2                  Average_Degree          0.141207
3        Average_Link_Betweenness          0.078120
4           Average_Shortest_Path          0.036693
```

Fig 4.2.4 Feature Importance Table

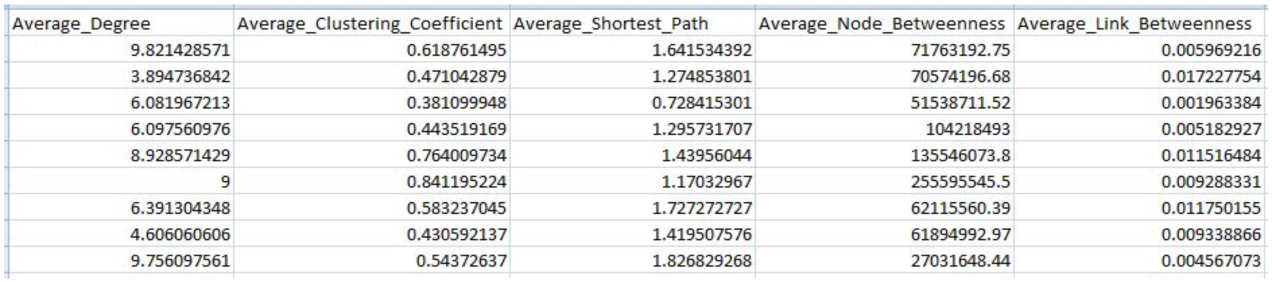


```
def findDecision(obj):
    if obj[1]<=0.814654441:
        if obj[2]<=2.407346939:
            if obj[4]>0.001963384:
                if obj[0]>2.72:
                    if obj[3]<=255595545.5:
                        return 'RA'
                    elif obj[3]>255595545.5:
                        return 'RA'
                    else: return 'RA'
                elif obj[0]<=2.72:
                    return 'RA'
                else: return 'RA'
            elif obj[4]<=0.001963384:
                return 'RWA'
            else: return 'RWA'
        elif obj[2]>2.407346939:
            return 'RWA'
        else: return 'RWA'
    elif obj[1]>0.814654441:
        return 'WA3'
    else: return 'WA3'
```

Fig 4.2.5 Built Decision Tree

Preparation of the Test Dataset

The test dataset is prepared in the same way as the train dataset with the only difference being absence of the Decision column, so as to perform prediction of the best AAI by DT. The best AAI are stored separately to check for the accuracy. A snapshot of the test dataset is shown below.



Average_Degree	Average_Clustering_Coefficient	Average_Shortest_Path	Average_Node_Betweenness	Average_Link_Betweenness
9.821428571	0.618761495	1.641534392	71763192.75	0.005969216
3.894736842	0.471042879	1.274853801	70574196.68	0.017227754
6.081967213	0.381099948	0.728415301	51538711.52	0.001963384
6.097560976	0.443519169	1.295731707	104218493	0.005182927
8.928571429	0.764009734	1.43956044	135546073.8	0.011516484
9	0.841195224	1.17032967	255595545.5	0.009288331
6.391304348	0.583237045	1.727272727	62115560.39	0.011750155
4.606060606	0.430592137	1.419507576	61894992.97	0.009338866
9.756097561	0.54372637	1.826829268	27031648.44	0.004567073

Fig 4.2.6 Test Dataset

Testing and predicting the best AAI

In the C4.5 DT, the best AAI selected is displayed using the *chefboost.predict(model, test_set)* function. The predicted AAI is then compared with the actual AAI to calculate the accuracy. Therefore, the DT model gives us the AAI which is also used in later stages of the algorithm.

The helper code to perform the above DT building is attached with this report along with a README file that contains instructions to execute the code.

4.3 Finding the complementary AAIs using SVM

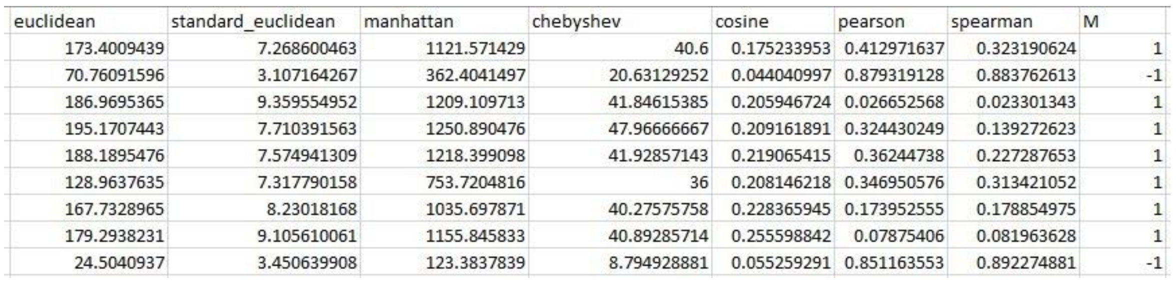
SVM or Support Vector Machine is an algorithm used for classification purposes. The plan of action for using SVM to find the complementary indices is that given an AAI pair, SVM will be used to classify whether these are complementary to each other or not.

Training the SVM

As mentioned in the literature followed for this project, the SVM model is to be trained with samples in the format *[I_i,M_i]* where each I_i is the combination of 7 similarity indices as mentioned below:

1. Euclidean Distance
2. Standardized Euclidean Distance
3. Manhattan Distance
4. Chebyshev Distance
5. Cosine Distance
6. Pearson Correlation Coefficient
7. Spearman Distance

These similarity indices describe the similarity of the AAIs and form the feature set for the samples to be fed to the SVM. The M_i part has a value 1 if the two AAIs in the AAI pair for the sample *i* are complementary and -1 if they are not. However, there is no clear mention in the literature about how to form the dataset that is how to achieve guaranteed complementary AAI pairs for training the SVM. Yet it has been stated that: *If the combination of B and candidate AAI has better AUC than that B has, then the candidate AAI is considered to be the complementary index of B.* Also, as mentioned earlier in section 3.1, the AUC is calculated taking random samples of connected and unconnected node pairs. Thus, to check for a complementary AAI of an AAI B, we calculate the AUC for B as well as the AUC for the combination of B and another AAI by picking up samples at random. These randomly picked samples are also considered for making the SVM training dataset. Since taking all the AAIs one by one is impractical and we need data that is dependent only on the values of the AAIs and not the AAI algorithm, we select the *best* AAI describing a network and its *complementary* indices. Note that the terms *best* and *complementary* used here are based on random sampling. The above-mentioned similarity indices were calculated using the values of the different AAIs from these randomly picked samples and based on the combined AUC score, the value of M was assigned. We took 6 datasets (networks) at random from the training set of networks (that were used for training the decision tree) and were used for the making of the training set for the SVM. The random sampling method of finding out complementary indices produces a very fewer number of complementary pairs for a network. When results from all the 6 networks are combined, this dramatically increases the number of -1s in the *M* part of the samples. If we train the SVM using these raw output samples, the SVM model is induced to bias towards always classifying an AAI pair as non-complementary. Thus, we manually remove samples with -1 as their target variable value and try to balance the number of complementary and non-complementary pairs in the samples. A snapshot of the dataset used for training the Support Vector Machine is shown below in fig 4.3.1. With the dataset prepared, we train the SVM model using RBF or Radial Basis Function as the Kernel Function for the SVM model.



euclidean	standard_euclidean	manhattan	chebyshev	cosine	pearson	spearman	M
173.4009439	7.268600463	1121.571429	40.6	0.175233953	0.412971637	0.323190624	1
70.76091596	3.107164267	362.4041497	20.63129252	0.044040997	0.879319128	0.883762613	-1
186.9695365	9.359554952	1209.109713	41.84615385	0.205946724	0.026652568	0.023301343	1
195.1707443	7.710391563	1250.890476	47.96666667	0.209161891	0.324430249	0.139272623	1
188.1895476	7.574941309	1218.399098	41.92857143	0.219065415	0.36244738	0.227287653	1
128.9637635	7.317790158	753.7204816	36	0.208146218	0.346950576	0.313421052	1
167.7328965	8.23018168	1035.697871	40.27575758	0.228365945	0.173952555	0.178854975	1
179.2938231	9.105610061	1155.845833	40.89285714	0.255598842	0.07875406	0.081963628	1
24.5040937	3.450639908	123.3837839	8.794928881	0.055259291	0.851163553	0.892274881	-1

Fig 4.3.1 Training set for SVM

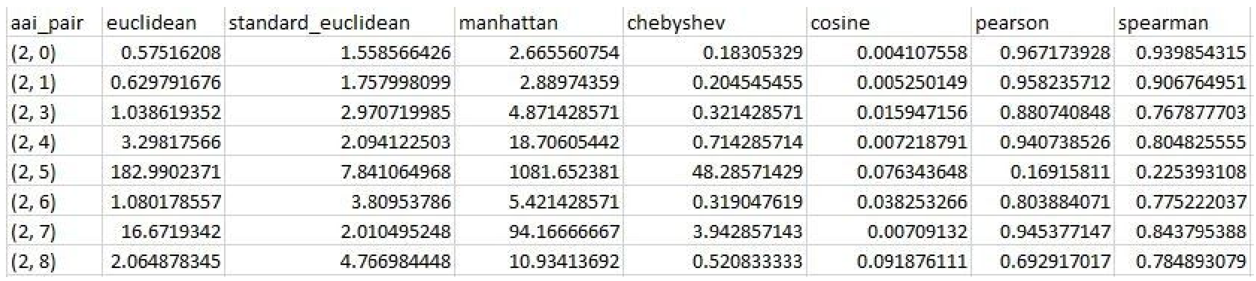
Classifying as complementary or non-complementary

After the trained model is ready, it can be used to find out complementary indices for the optimal AAIs found using the Decision Tree model in earlier steps. The helper code to do so is attached with this report along with a README file that contains instructions to execute the code. For any network whose complementary indices are to be found, the following steps are followed to make the network ready for feeding to the SVM:

a. The values for all the AAIs for the network (25 AAI values corresponding to 25 algorithms) are computed.

b. Each AAI is represented using a number/index as mentioned in the table below. AAI index pairs are generated. In each pair, one index is that of the optimal AAI found in the Decision Tree step. Thus, for a network we will have 24 AAI pairs since the combination of the optimal AAI with itself is discarded. For each AAI pair the 7 similarity indices are calculated in the same way as we did for the training samples. However, this time, the target variable *M* is absent. This will be predicted by the SVM. An example of the sample dataset is shown in fig 4.3.2 . The AAI pair is set as the index while training. This is not a feature and is solely for the purpose of identifying the sample.

c. The 24 samples thus obtained are thus fed into the SVM and for each of these samples SVM predicts the target *M*. The AAI pairs for which the value of *M* turns out to be 1 are noted and are further used for finding out the composite AAI value.



aai_pair	euclidean	standard_euclidean	manhattan	chebyshev	cosine	pearson	spearman
(2, 0)	0.57516208	1.558566426	2.665560754	0.18305329	0.004107558	0.967173928	0.939854315
(2, 1)	0.629791676	1.757998099	2.88974359	0.204545455	0.005250149	0.958235712	0.906764951
(2, 3)	1.038619352	2.970719985	4.871428571	0.321428571	0.015947156	0.880740848	0.767877703
(2, 4)	3.29817566	2.094122503	18.70605442	0.714285714	0.007218791	0.940738526	0.804825555
(2, 5)	182.9902371	7.841064968	1081.652381	48.28571429	0.076343648	0.16915811	0.225393108
(2, 6)	1.080178557	3.80953786	5.421428571	0.319047619	0.038253266	0.803884071	0.775222037
(2, 7)	16.6719342	2.010495248	94.16666667	3.942857143	0.00709132	0.945377147	0.843795388
(2, 8)	2.064878345	4.766984448	10.93413692	0.520833333	0.091876111	0.692917017	0.784893079

Fig 4.3.2 Data sample for which complementary AAIs are to be found using the SVM model

The index for each AAI algorithm is mentioned in table 4.3.1. Please consult table 4.1a and 4.1b for a complete detail of what S1,S2 etc mean.

AAI Algorithm (Abbreviation as used in Paper)	Index used for Code
S1	0
S2	1
...	...
...	...
S25	24

Table 4.3.1 Index (used in the code) representing the AAIs

The 7 similarity indices mentioned above are calculated using the library **scipy** in the **Python** programming language. The **standardized Euclidean distance** is just the normalized Euclidean distance between the AAI vectors and the **Spearman distance** is calculated as the square of the Euclidean distance between the rank vectors of the AAI pattern vectors. The formulae used for calculating the other 5 indices are mentioned in table 4.3.2 below.

Similarity Index	**Formula**
Euclidean Distance	$\sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}$
Manhattan Distance	$\sum_i \lvert u_i - v_i \rvert$
Chebyshev Distance	$max(\lvert u_i - v_i \rvert)$
Cosine Distance	$1 - \frac{u.v}{\lvert u \rvert \lvert v \rvert}$
Pearson Correlation Coefficient	$\frac{CoV(x,y)}{\sigma_x \sigma_y}$

Table 4.3.2 Formulae for different similarity indices

4.4 Finding the composite AAIs for each network

So as to exhaustively depict the attention distribution of common friends of users in various circles, IAALPA designs a composite mutually complementary AAI. The composite index model is essentially made out of the best index B chosen by DT and the complementary AAIs of B recognized by SVM (for example E_1, E_2,...,E_h) as explained in section 4.3. Targeting for making the AAI with great performance represent a huge extent in the composite AAI, average AUC value *w* of each AAI, which is obtained in training DT model, is applied as its weight.In the process of training the DT model, the AUC of AAI for each network in training set is calculated, and the average AUC value *w* of each AAI is used here as its weight in the combined model to find the composite AAI.

Also in addition *h* excellent complementary AAIs with larger weights are selected to build the composite AAI to further improve the accuracy of the algorithm, as shown in formula:

$$S = w * (B, E_1, E_2, \ldots, E_h)^T$$

Essentially, the formula represents the item-wise multiplication of two lists. *w* is the list of the values of the AUC of the indices. Let us define w as

$$w = [wb, w1, w2, \ldots, wh]$$

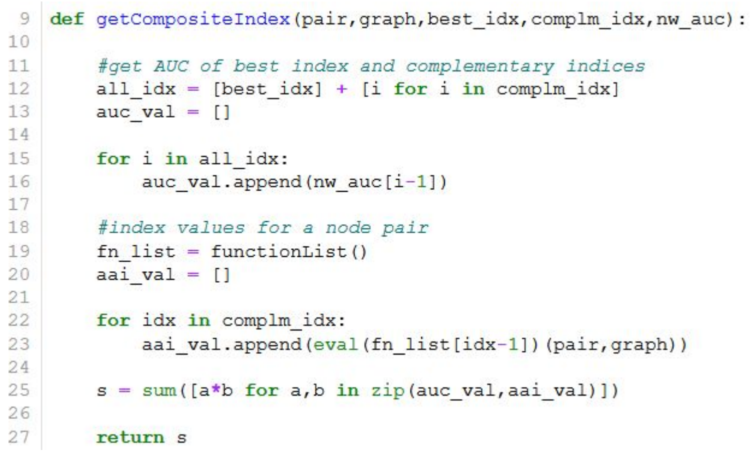
Let *n* = *(source, target)* be a node pair whose possibility of being connected is to be calculated. Also, let,

$$aai\_n = [aai\_b(n), aai\_e1(n), aai\_e2(n), \ldots, aai\_eh(n)]$$

We can calculate the composite index as

$$S(n) = wb * aai\_b(n) + w1 * aai\_e1(n) + w2 * aai\_e2(n) + \ldots + wh * aai\_eh(n)$$

This is implemented as shown in Fig 4.4.1.



```python
def getCompositeIndex(pair,graph,best_idx,complm_idx,nw_auc):

    #get AUC of best index and complementary indices
    all_idx = [best_idx] + [i for i in complm_idx]
    auc_val = []

    for i in all_idx:
        auc_val.append(nw_auc[i-1])

    #index values for a node pair
    fn_list = functionList()
    aai_val = []

    for idx in complm_idx:
        aai_val.append(eval(fn_list[idx-1])(pair,graph))

    s = sum([a*b for a,b in zip(auc_val,aai_val)])

    return s
```

Fig 4.4.1: Code snippet to find the Composite Index which will be the final output of the IAALP algorithm

5. Results and Discussion

The average AUC was computed for each indices using a few datasets. All the datasets could not be used due to resource constraints. The plot of the average AUC for each AAI is shown in Fig 5.1. The plot is found consistent with the results obtained in the concerned study. The average AUC values are shown in Fig 5.2.

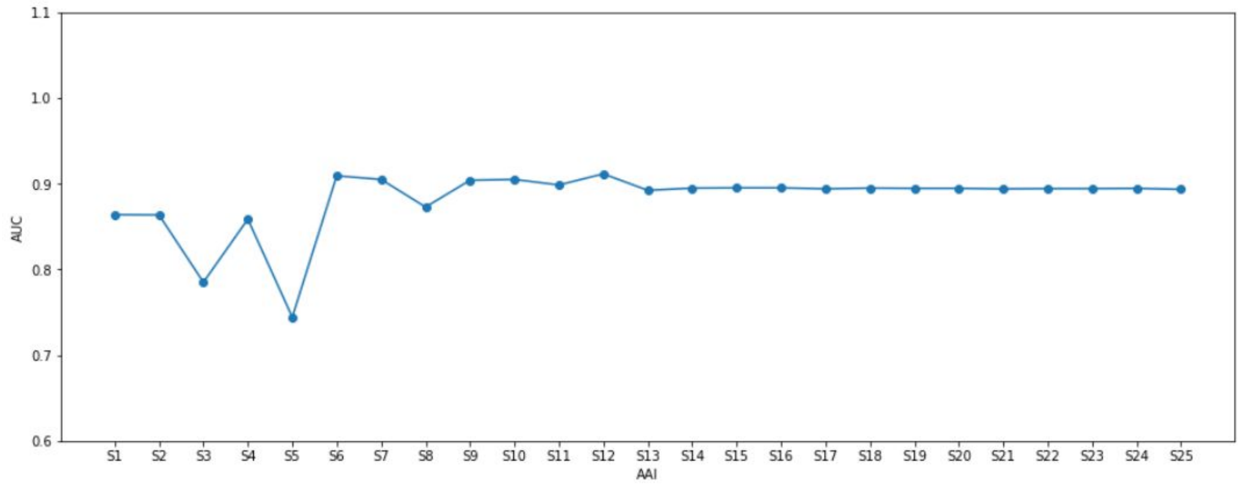
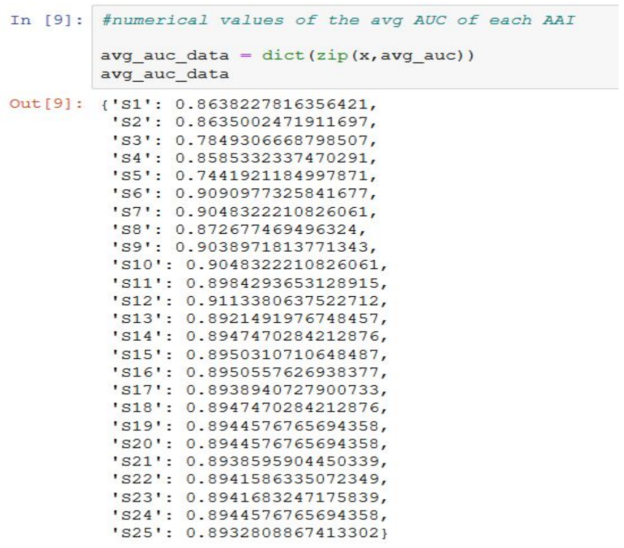


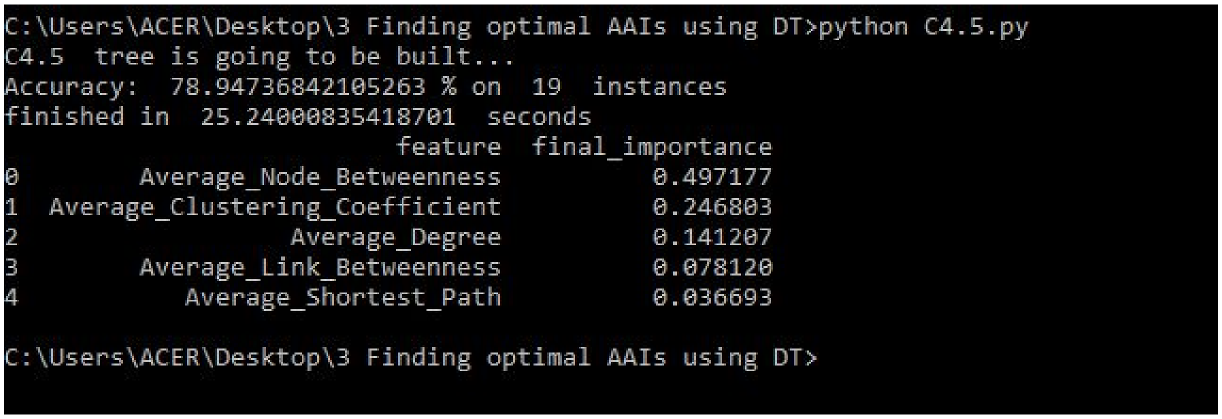
Fig 5.1: Plot of average AUC for each Attention Allocation Index for performance comparison



```
In [9]:  #numerical values of the avg AUC of each AAI

         avg_auc_data = dict(zip(x,avg_auc))
         avg_auc_data

Out[9]:  {'S1': 0.8638227816356421,
          'S2': 0.8635002471911697,
          'S3': 0.7849306668798507,
          'S4': 0.8585332337470291,
          'S5': 0.7441921184997871,
          'S6': 0.9090977325841677,
          'S7': 0.9048322210826061,
          'S8': 0.872677469496324,
          'S9': 0.9038971813771343,
          'S10': 0.9048322210826061,
          'S11': 0.8984293653128915,
          'S12': 0.9113380637522712,
          'S13': 0.8921491976748457,
          'S14': 0.8947470284212876,
          'S15': 0.8950310710648487,
          'S16': 0.8950557626938377,
          'S17': 0.8938940727900733,
          'S18': 0.8947470284212876,
          'S19': 0.8944576765694358,
          'S20': 0.8944576765694358,
          'S21': 0.8938595904450339,
          'S22': 0.8941586335072349,
          'S23': 0.8941683247175839,
          'S24': 0.8944576765694358,
          'S25': 0.8932808867413302}
```

Fig 5.2: Values of average AUC for each index

The Decision Tree model selects the best AAI for a particular network circle. The twitter ego-net dataset is used. Files ending with .edges are used (having size <= 8 KB). These files are edgelists. They are first converted into .csv files and then fed to the DT for train and test. Figure 5.3 provides a snapshot of the DT model tested on the Test dataset consisting of 19 instances. The model has a decent accuracy and speed.



```
C:\Users\ACER\Desktop\3 Finding optimal AAIs using DT>python C4.5.py
C4.5  tree is going to be built...
Accuracy:  78.94736842105263 % on  19  instances
finished in  25.24000835418701  seconds
                       feature  final_importance
0        Average_Node_Betweenness          0.497177
1  Average_Clustering_Coefficient          0.246803
2                  Average_Degree          0.141207
3        Average_Link_Betweenness          0.078120
4           Average_Shortest_Path          0.036693

C:\Users\ACER\Desktop\3 Finding optimal AAIs using DT>
```

Fig 5.3: Accuracy of DT on the test dataset

The model is also tested on a small sample network and it gives the correct predicted value for the network. The network graph is shown in figure 5.4 along with the DT prediction in figure 5.5.

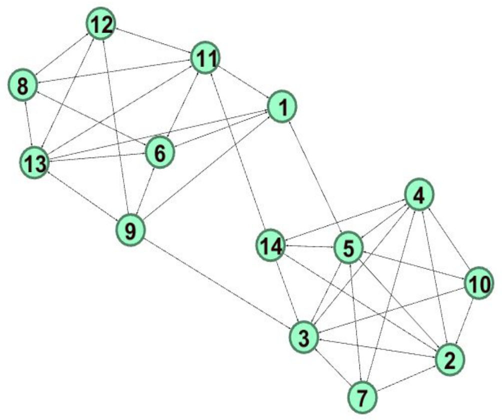
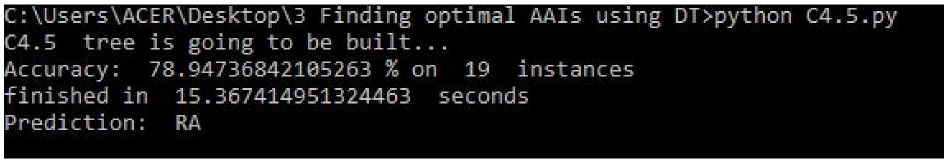


Fig 5.4. Sample Network Graph used for prediction



```
C:\Users\ACER\Desktop\3 Finding optimal AAIs using DT>python C4.5.py
C4.5  tree is going to be built...
Accuracy:  78.94736842105263 % on  19  instances
finished in  15.367414951324463  seconds
Prediction:  RA
```

Fig 5.5. Prediction on the sample network graph (RA)

Discussion

Upon implementing the IAALP algorithm and evaluating the results, the following observations were made:

- Since the AUC value is being computed by random sampling, the output of AUC is different every time the function is run. No clear algorithm has been mentioned in the study and hence, the implementation is based upon our understanding of the computation.

- Several AAIs can have the same value of AUC. In such a case, for ease of computation, we are picking one of the best AAIs at random. An alternative implementation can be to continue random sampling of the best indices until only one remains.

- Despite the discussed shortcomings, our implementation actually gives results very close to the original implementation. This is evident from Fig 5.1 where the plot is almost identical to the original plot and also the closeness of the values to the original computation as shown in Fig 5.2.

- Smaller dataset was used for the Decision Tree due to the limited computational capabilities of our systems.

- The accuracy of the Decision tree model is 78.94%.

- SVM classified AAIs into complementary/non-complementary indices.

- Ultimately, the composite index was computed by a weighted sum of the best index and the complementary indices.

6. Conclusion

The Intelligent Attention Allocation Link Prediction (IAALP) algorithm was implemented in the current study. The algorithm is a novel proposition and combines several old and innovatively designed new attention allocation algorithms. The use of the decision tree ensures that appropriate AAIs are selected adaptively for a specific circle structure. The SVM selects the complementary AAIs that are used to deal with the issues of underfitting and overfitting. IAALP is thus a highly flexible model that can be used for any online brand community to identify potential friendship relationships and ultimately, achieve the target of cross-marketing products.

References

1. Wasserman, Stanley; Faust, Katherine (1994). "Social Network Analysis in the Social and Behavioral Sciences". *Social Network Analysis: Methods and Applications*. Cambridge University Press. pp. 1–27. ISBN 9780521387071.

2. Lü, L., & Zhou, T. (2011). Link prediction in complex networks: A survey. Physica A: Statistical Mechanics and its Applications, 390 (6), 1150–1170. doi: 10.1016/j.physa. 2010.11.027 .

3. Liben-Nowell, D., & Kleinberg, J. (2007). The link-prediction problem for social networks. Journal of the American Society for Information Science and Technology, 58 (7), 1019–1031. doi: 10.1002/asi.20591 .

4. https://snap.stanford.edu/data/ego-Twitter.html

5. John, L. K. , Mochon, D. , Emrich, O. , & Schwartz, J. (2017). What's the value of a like? Social media endorsements don't work the way you might think. Harvard Business Review, 95 (2), 108–115 .

6. Whittler, T. E., & Spira, J. S. (2002). Model's race: A peripheral cue in adver- tising messages? Journal of Consumer Psychology, 12 (4), 291–301. doi: 10.1207/ 15327660260382333 .

7. Solomon, M. R. (2016). Consumer behaviour: Buying, having, and being(12th Edi- tion). Pearson .

8. Wang, X. , & Xue, H. (2010). Brand community social capital, perceived value and brand loyalty. Journal of Management Science, 23 (6), 53–63 .

9. Wang, Q. , Wu, Y. , & Yao, Y. (2017). Classification of heart disease based on ultra- sound data of transesophageal echocardiography. Journal of Computer Applica- tions, 37 (S1), 221 .

10. Shan, Z., Kong, J., Zhang, Y., Li, H., et al. (2018). Remote sensing investigation method of object-oriented crops with special characteristics. Journal of Geo-information Science, 20 (10), 1509–1519. doi: 10.12082/dqxxkx.2018.180116 .

11. Mantas, C. J., Abellán, J., & Castellano, J. G. (2016). Analysis of Credal-C4. 5 for classification in noisy domains. Expert Systems with Applications, 61 , 314–326. doi: 10.1016/j.eswa.2016.05.035 .

12. Wang, X., & Xing, Y. (2019). An online support vector machine for the open-ended environment. Expert Systems with Applications, 120 , 72–86. doi: 10.1016/j.eswa.2018.10.027 .

Appendix A: System Specification

Libraries:

1. Networkx
2. Pandas
3. Numpy
4. Sklearn
5. Chefboost
6. Matplotlib
7. Pickle
8. Scipy

Tools:

Python 3.5 or higher
Jupyter Notebook