CHAPTER 1
INTRODUCTION

The vast majority of the targets of approved drugs are proteins [1, 2]. Knowledge of which proteins are the targets of approved drugs enables the division of the human proteome into two classes: approved drug targets and non-targets. A protein is an approved drug target if it is the target of an approved drug, and a non-target otherwise. For this classification, PPI networks plays an important role.

Protein-protein interactions (PPIs) handle a wide range of biological processes, including cell-to-cell interactions and metabolic and developmental control [3]. Protein-protein interaction is becoming one of the major objectives of system biology. On the basis of their interaction surface, they may be homo- or heterooligomeric; as judged by their stability, they may be obligate or non-obligate; as measured by their persistence, they may be transient or permanent [4]. A given PPI may be a combination of these three specific pairs. The transient interactions would form signalling pathways while permanent interactions will form a stable protein complex.

The protein-protein interactions (PPI) have a great role in many biological events like Enzyme substrate interactions or hormone-receptor binding or immunological functions (like antigen recognition and presentation etc.) to name a few of them, revealing the fact that an altered PPI can have a potential to disease onset. Thus exploring the molecular details of PPI will not only enrich our knowledge but also could be highly useful in modern medical biochemistry fields like molecular medicine research and other areas.

The main molecular targets for drugs are proteins (mainly enzymes, receptors, and transport proteins) and nucleic acid (DNA and RNA). It is indeed gratifying that development of research in cell biology, molecular biology, and biochemistry produced a remarkable compendium of knowledge on the function and molecular properties of individual proteins [5]. However, a protein usually carries out a typical function by regulating other molecules. In other words, it rarely acts alone. Therefore, in the past few years, with the emergence of high-throughput technologies like yeast 2 hybrid protein interactions, research of protein-protein interactions (PPI) has been occurring with high frequency. PPI provides us with

more information for understanding the relationship between drug targets and other proteins in a systematic point of view.

1.1 PROTEIN

Proteins are large biomolecules, or macromolecules, consisting of one or more long chains of amino acid residues.

Proteins perform a vast array of functions within organisms, including catalysing metabolic reactions, DNA replication, responding to stimuli, providing structure to cells and organisms, and transporting molecules from one location to another. Proteins differ from one another primarily in their sequence of amino acids, which is dictated by the nucleotide sequence of their genes, and which usually results in protein folding into a specific three-dimensional structure that determines its activity.

In a protein chain, amino acids form a highly stable covalent bond viz. peptide bond (-CO-NH-) between the previous residues carboxylic group to next amino group. Side chains of different amino acids are important for PPI, as they actively take part in it.

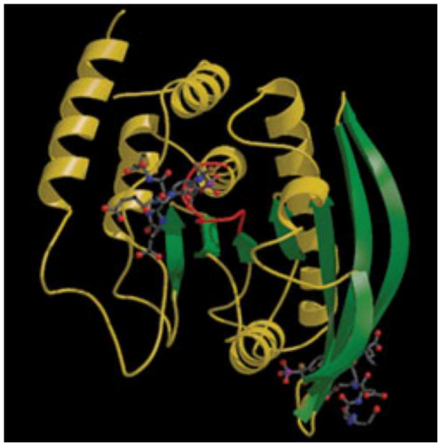


Fig. 1.1 Proteins of different Structure

1.2 PPI

Protein–protein interactions (**PPIs**) are the physical contacts of high specificity established between two or more protein molecules as a result of biochemical events steered by electrostatic forces including the hydrophobic effect. Many are physical contacts with molecular associations between chains that occur in a cell or in a living organism in a specific biomolecular context [6].

Proteins rarely act alone as their functions tend to be regulated. Many molecular processes within a cell are carried out by molecular machines that are built from a large number of protein components organized by their PPIs.

PPIs have been studied from different perspectives: biochemistry, quantum chemistry, molecular dynamics, signal transduction, among others. All this information enables the creation of large protein interaction networks – similar to metabolic or genetic/epigenetic networks – that empower the current knowledge on biochemical cascades and molecular etiology of disease, as well as the discovery of putative protein targets of therapeutic interest.

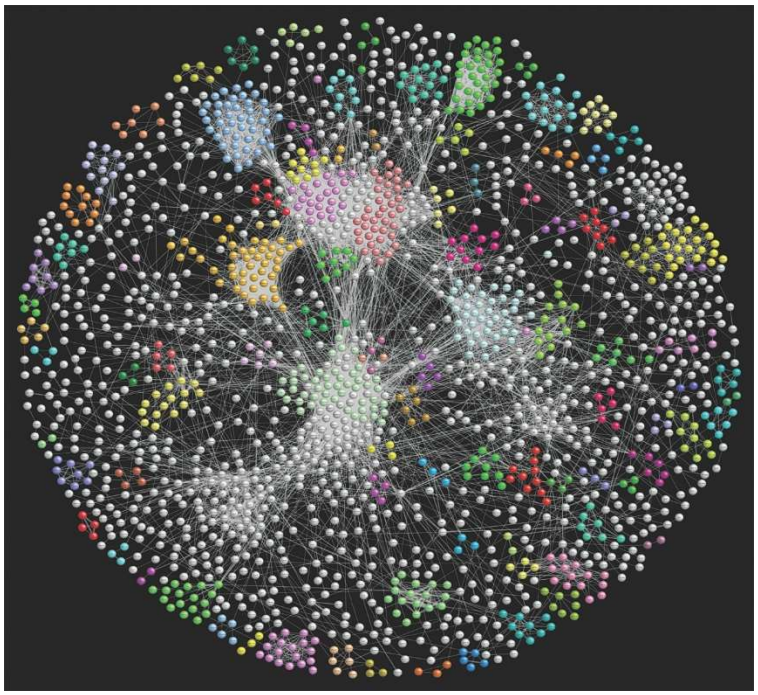


Fig 1.2 Protein–protein interactions

CHAPTER 2
DATA COLLECTION

2.1 Database: In our project the main resource of protein-protein interactions Database is from three most widely used PPI databases i.e. STRING (version 10.0, PPI), DrugBank and Uniprot. The databases contained a combined (2029) nonredundant nodes, out of which 196 are target drugs and remaining 505 are non drug targets. In which, the main information of drug targets was extracted from the DrugBank database.
From the dataset, after converting into the graph we can observe that there are 701 nodes(vertices) and 18966 edges. These 701 nodes interact with all those edges to become the perfect network protein-protein interaction model.

2.1.1 STRING (version 10.0,PPI): STRING is a biological database and web resource of known and predicted protein–protein interactions[32][33]. The STRING database contains information from numerous sources, including experimental data, computational prediction methods and public text collections [34]. It is freely accessible and it is regularly updated [35]. The resource also serves to highlight functional enrichments in user-provided lists of proteins, using a number of functional classification systems [36][37]. The extracted data from STRING contains (--num--) proteins.

2.1.2 DrugBank: The DrugBank database is a comprehensive, freely accessible, online database containing information on drugs and drug targets. As both a bioinformatics and a cheminformatics resource, DrugBank combines detailed drug (i.e. chemical, pharmacological and pharmaceutical) data with comprehensive drug target (i.e. sequence, structure, and pathway) information [38]. In which, the main information of drug targets was extracted from the DrugBank database, in which the approved targets set contains (--num--) proteins.

2.1.3 UniProt: The Universal Protein Resource (UniProt) is a comprehensive resource for protein sequence and annotation data. UniProt is a freely accessible database of protein sequence and functional information, many entries being derived from genome sequencing projects [39]. It contains a large amount of information about the biological function of proteins derived from the research literature. In our project the extracted data from Uniport contains (--num--) proteins.

CHAPTER 3

NETWORK TOPOLOGY

3.1 Network Topology

The drug targets network is represented as an undirected network $G = (V, E)$, where V denotes the protein in D set or PT set and E is the interactions between each proteins pair. For each node $i \in V$, k_i denotes the degree of it. A is the adjacency matrix for the network, where $A_{i,} = 0$ when there are no interactions (no edge) between nodes i and j. Similarly, $A_{i,} = 1$ when there is an interaction between each other.

The drug targets are the special proteins through which the drugs carry out their specific functions, they are thought institutively as.

1. the intermediaries which play an important role on interactions of the drug targets network;
2. the sources which receive the drug stimulus and convert it into another stimulus that can be responded to by normal proteins;
3. the proteins which have special topological and functional significance.

According to these three points, we analyzed the listed topological features of the drug targets network, including degree and betweenness for the intermediary function, eccentricity,
and average distance for the source function, modularity, coreness, cluster coefficient, and eigenvector centrality for special topology. However, from analyses of the PPI topological indices, the drug targets do not have the first two
characteristics. Actually, the results show they are similar with other proteins on intermediary and source functions. In comparison, there are some significant differences on special topology.

3.1.1 Degree: The degree of a node in a network (sometimes referred to incorrectly as the connectivity) is the number of connections or edges the node has to other nodes. In protein interaction networks, the hubs are defined as ones that have a higher degree than others.

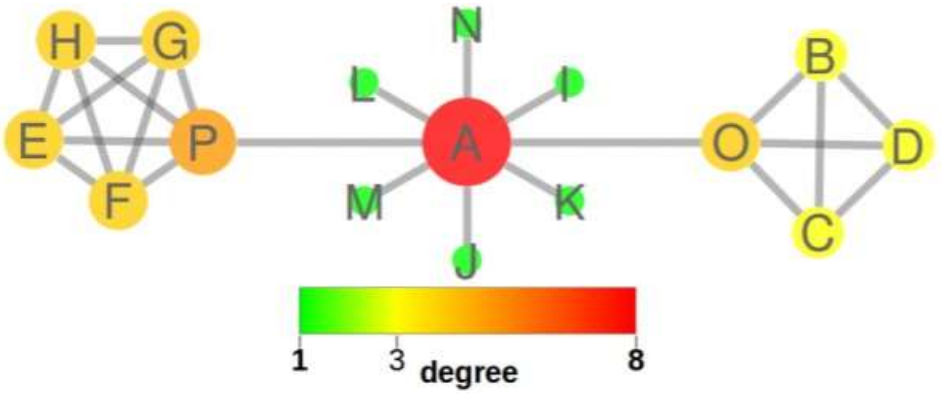


Fig. 3.1 For hub identification, the numerical values used as centralities are simply the number of interactions

3.1.2 Betweenness: Betweenness is the number of times a node is in the shortest paths between two other nodes. It is basically defined as the number of shortest paths in the graph that pass through the node divided by the total number of shortest paths.

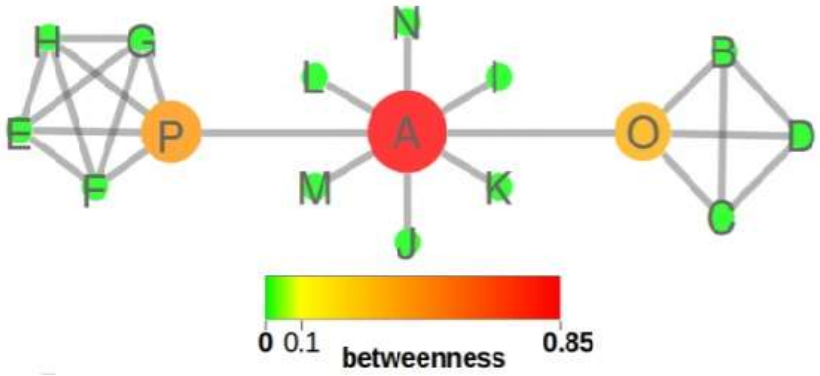


Fig 3.2 A protein is considered to be central if it participates in numerous shortest paths than other proteins

In the PPI network, if the drug targets are the proteins that play an important role on intermediary, they may have higher betweenness than others.

The betweenness centrality of a node V is given by

$$\text{Btwn}(v) = \sum_{s \neq v \neq t} \frac{\sigma st(v)}{\sigma st}, \quad (1)$$

where σst is the total number of shortest paths from node s to node t and $\sigma st(v)$ is the number of those paths that pass through v.

The normalized betweenness (NB) can be calculated without a loss of precision

$$\text{NB}(v) = \frac{Btwn(v) - \text{mi}\ \ (Btwn)}{\max(Btwn) - \min(Btwn)}, \quad (2)$$

It results in max(NB) = 1 and min(NB) = 1.

3.1.3. Average Distance: Average distance, also called average path length, is a concept in network topology that is defined as the average number of steps along the shortest paths for all possible pairs of network nodes. Here, we consider an unweighted graph G; let (Vi, Vj) denote the shortest distance between Vi and Vj. Assume that (Vi, Vj) = 0 if $i = j$ or DG if Vi cannot be reached from Vj, where DG is the diameter of the PPI network.

3.1.4. **Eccentricity*:*** The eccentricity (V) of a vertex V is the greatest geodesic distance between it and any other vertex. It can be thought of as how far a protein is from the proteins farthest from it in the graph. If the eccentricity of the node V is low, the other nodes are in proximity. On the contrary, if it is high, it implies that there is at least one node (and all its neighbors) that is far from node V. Let (Vi, Vj) denote the distance (number of edges connected) between vertices i and j; then the eccentricity can be calculated as

$$(\mathrm{V}) = \max_{\mathrm{V}i,\mathrm{V}j\in V} d(\mathrm{V}i, \mathrm{V}j), \quad (3)$$

The eccentricity shows the easiness of a protein to be functionally reached by all other proteins in the network. Thus, a protein with low eccentricity is subject to a more stringent or complex regulation so that it could easily influence several other proteins.

3.1.5. Modularity: Modularity is the degree to which the components of the networks may be separated and recombined.

Modularity Q of networks, which is a benefit function defined as that measures the quality of a division of a network into groups or communities.

$$Q = \frac{1}{2m}\sum_{i,j}\left(A_{i,j} - \frac{k_i k_j}{2m}\right)\delta\left(c_i, c_j\right), \quad (4)$$

where the degree of node i assigned to community ci is ki, if there is no interaction between proteins i and j, and 1 otherwise, m = (1/2)Σi,jAi,j, and $\delta(ci, cj)$ is 1 if $ci = cj$ and 0 otherwise.

3.1.6 Coreness: A k-core of a graph G is a maximal connected subgraph of G in which all vertices have degree k at least. K-Core measures the small interlinked core areas on a network graph.

It is a measure that can help identify tightly interlinked subgraph within a network graph. The value of k is referred to as the coreness of the group.

For example, a subgraph is the two-core if it contains all nodes that are connected to at least two other nodes within the subgraph. Similarly, a subgraph is the three-core if it contains all entities that are linked to at least three other nodes within the subgraph.

Any node in the three-core must have at least three links to all other members of the three-core. Clearly any such node must necessarily have at least two links to every other member of the three-core. We can say that, the three-core is a subset (is contained within) the two-core. More generally, the K-Core form a nested hierarchy of entity groupings on the chart.

The zero-core (entire chart) contains the one-core, which contains the two-core, which contains the three-core, and so on. As k increases, the core sizes decrease, but the cores become more interlinked. The K-Cores with the biggest coreness values (k-values) represent the most cohesive regions of the graph.

The coreness of a protein is n if this protein is in the n-core of the PPI network but not in the n + 1-core. We examined $rc_{D(k)}$ and $rc_{PT(k)}$ defined as follows for drug targets and other proteins in k-core, respectively.

$$rc_{D(k)}=\frac{p_k^D}{DR}$$
$$rc_{PT(k)}=\frac{p_k^{PT}}{PR} \quad (5)$$

where p_k^D and p_k^{PT} denote the proportion of the drug targets and other proteins in k-core of the PPI network, respectively. DR and PR are the drug target's ratio and pending test ratio

3.1.7 Cluster Coefficient: The clustering coefficient measures the degree of connectivity in the neighborhood of a protein in the network. A protein with a high cluster coefficient is part of a densely connected part of the network. In a dense part of a network, if protein A is connected to protein B, and protein B is connected to protein C, the probability is high that protein A is also connected to protein C (Fig. 2 (C)). The cluster coefficient can also be measured for the entire network and indicates the average cluster coefficient of all proteins in the network.

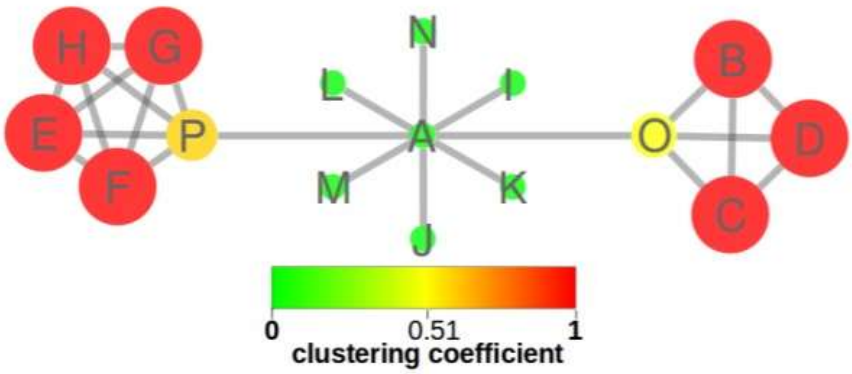


Fig. 3.3 A protein with a high cluster coefficient is part of a densely connected part of the network

For unweighted graphs, the clustering of a node u is the fraction of possible triangles through that node that exist,

$$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)},$$

where $T(u)$ is the number of triangles through node $u$and $deg(u)$ is the degree of u.

3.1.8 Clique: A clique, C, in an undirected graph $G = (V, E)$ is a subset of the vertices, $C \subseteq V$, such that every two distinct vertices are adjacent. This is equivalent to the condition that the induced subgraph of G induced by C is a complete graph.

A maximal clique is a clique that cannot be extended by including one more adjacent vertex, that is, a clique which does not exist exclusively within the vertex set of a larger clique. Some authors define cliques in a way that requires them to be maximal, and use other terminology for complete subgraphs that are not maximal.

A maximum clique of a graph G is a clique, that includes the largest possible number of vertices.

Moreover, the clique number ω(*G*) of a graph *G* is the number of vertices in a maximum clique in *G*.

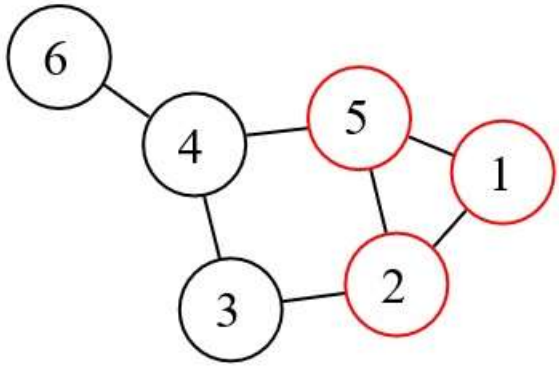


Fig 3.4 The graph shown has one maximum clique ,the triangle {1,2,5},and four more maximal cliques , the pairs {2,3},{3,4},{4,5} and {4,6}.

3.1.9 Closeness Centrality: Closeness centrality [7] of a node u is the reciprocal of the sum of the shortest path distances from u to all $n-1$other nodes. Since the sum of distances depends on the number of nodes in the graph, closeness is normalized by the sum of minimum possible distances $n-1$.

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)},$$

where $d(v,u)$ is the shortest-path distance between v and u, and n is the number of nodes in the graph.

Thus, higher values of closeness indicate higher centrality.

3.1.10 Stress Centrality: Stress centrality measures the amount of communication that passes an element in an all-to-all scenario. More precisely, it is not only an all-to-all scenario but every vertex sends as many goods or information units to every other vertex as there are shortest paths between them and stress centrality measures the according stress [8].
The stress is a node centrality index. Stress is calculated by measuring the number of shortest paths passing through a node. To calculate the stress of a node v, all shortest paths in a graph G are calculated and then the number of shortest paths passing through v is counted. A stressed node is a node traversed by a high number of shortest paths.

3.1.11 MNC-Maximum Neighborhood Component: The neighborhood of a node v, nodes adjacent to v, induce a subnetwork N(v)[27]. The score of node v, MNC(v), is defined to be the size of the maximum connected component of N(v). The neighborhood N(v) is the set of nodes adjacent to v and does not contain node v.[28]

$$MNC(v) = |V(MC(v))|$$

where MC(v) is a maximum connected component of the G[N(v)] and G[N(v)] is the induced subgraph of G by N(v).

3.1.12 DMNC - Density of Maximum Neighborhood Component: For a node v, let N be the node number and E be the edge number of MNC(v), respectively. The score of node v, DMNC(v), is defined to be E/N^{ϵ} for some $1 \leq \epsilon \leq 2$. We may assume that the MNC has a strong community structure, such as a clique percolation in a random network. In our system, ϵ is set to be 1.7, which is close to 1.67, the ϵ-value as we assume the neighborhood sub-network has a four-community [29].

$$\frac{|E(MNC(v))|}{|V(MNC(v))|^{\varepsilon}} \quad \mathbf{1 \leq \epsilon \leq 2}$$

3.1.13 Radiality Centrality: The radiality is a node centrality index and will give high centralities to vertices that are a short distance to every other vertex in its reachable neighborhood compared to its diameter.[30]

Radiality centrality defined as:

$$C_{rad}(v) = \frac{\sum_{w \in v}(\Delta_G + 1 - dist(v,w))}{n-1}$$

where *d* is diameter of graph G with *n* vertices and *d(v,w)* is distance between vertex *v* and *w*.

The radiality of a node *v* is calculated by computing the shortest path between the node *v* and all other nodes in the graph. The value of each path is then subtracted by the value of the diameter +1 (G+1) and the resulting values are summated. Finally, the obtained value is divided for the number of nodes -1 (n-1). The radiality should be always compared to the closeness and to the eccentricity: a node with high eccentricity + high closeness+ high radiality is a consistent indication of a high central position in the graph.

CHAPTER 4
TOOLS USED

4.1 Tools Used: In our project three tools has been used i.e. Cytohubba API and Cytoscape, R and python.
For feature extraction R language has been used as it has plethora of packages for network graph. Python is used for performing machine learning algorithms for training our model for testing and Ctyscope has been used for visualizing and validating the all topological feature.

4.1.1 Cytoscape: Cytoscape is an open source bioinformatics software platform for visualizing molecular interaction networks and integrating with gene expression profiles and other state data [40].
4.1.1.1 Cytohubba API: CytoHubba is a Java plugin for Cytoscape, a facilitated platform for the analysis and visualization of molecular interaction networks based on our previous web application, Hubba [41].
CytoHubba provides a user-friendly interface to analyze topology of protein-protein interaction networks, such as human, yeast, fly etc. We can obtain the identified nodes (also called essential nodes) with scores from calculation and the subnetwork composed of these nodes by utilizing cytoHubba in Cytoscape.

We have used cytoHubba for ranking nodes in the protein-protein interaction network by their network features. CytoHubba provides 11 topological analysis methods including Degree, Edge Percolated Component, Maximum Neighborhood Component, Density of Maximum Neighborhood Component, Maximal Clique Centrality and six centralities (Bottleneck, EcCentricity, Closeness, Radiality, Betweenness, and Stress) based on shortest paths.

4.1.2 R: R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing[31]. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

4.1.3 Python: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding [42].
Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Some of the library that has been used are in our python program are pandas, NumPy, NetworkX, community, matplotlib etc.

4.1.3.1 Pandas: pandas is a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

Some of the feature of pandas are:

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.

4.1.3.2 NumPy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

4.1.3.3 NetworkX: NetworkX is a python library for studying graphs and networks. NetworkX is free software released under the BSD-new license. NetworkX is suitable for operation on large real-world graphs: e.g., graphs in excess of 10 million nodes and 100 million edges.[4] Due to its dependence on a pure-Python "dictionary of dictionary" data structure, NetworkX is a reasonably efficient, very scalable, highly portable framework for network and social network analysis.

CHAPTER 5
Machine Learning

Data Preparation: As our data was very large, we applied Stratified sampling, before giving it to our machine learning algorithms.

5.1 Stratified sampling: It is a method of sampling from a population.
In this method the total population is divided into smaller groups or strata to complete the sampling process. The strata is formed based on some common characteristics in the population data. Then simple random sampling or systematic sampling is applied within each stratum. The objective is to improve the precision of the sample by reducing sampling error. It can produce a weighted mean that has less variability than the arithmetic mean of a simple random sample of the population.

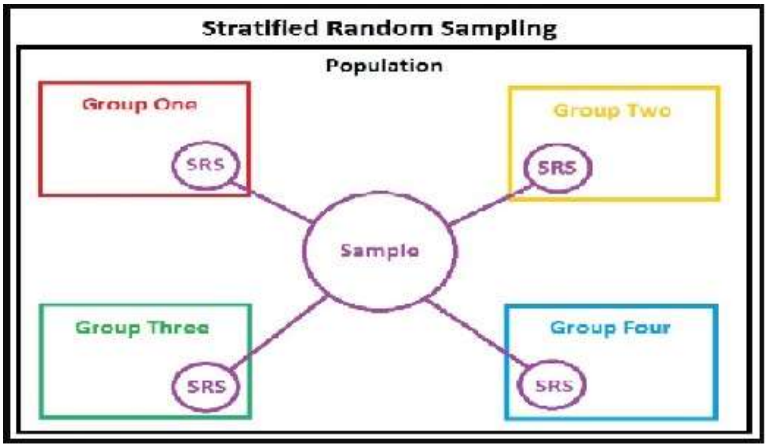


Fig 5.1 Stratified Random Sampling

5.2 MACHINE LEARNING: The machine learning algorithm (classifier) for DrugTarget protein-protein interaction prediction uses a set of various features (or descriptors) of the proteins or protein pairs with known Drugtarget Protein and non-DrugTarget Protein as learning set to learn which proteins is a DrugTarget Protein and which are not, and then the algorithm can classify new protein pairs to DrugTarget or non-DrugTarget classes. We have used some of the machine learning algorithms to address protein-protein interaction prediction. The algorithms which are used are briefly describe below.

5.2.1 RANDOM FOREST: Random Forest (RF) algorithm is a classification method that consists of many decision trees, in training phase each tree is constructed based on random feature vectors sampled from a data set independently and for every node in a tree, a small fraction of variables is randomly selected and then each classification tree is completely grown. To classify a new object, put the input vector down each of the trees in the forest, and finally according to the majority voting one class is assigned to the object. The RF is a practical classifier when there are a large dataset and large number of features and no need to feature selection or feature deletion, also it can rank features according to importance for classification. In addition, RF can be used for recovering missing data, but in some databases containing noisy data RF may be overfit [18]. Decision trees and random forest are widely used in bioinformatics and computational biology for classifying biological data [19] especially for PPI prediction [20, 21,22].

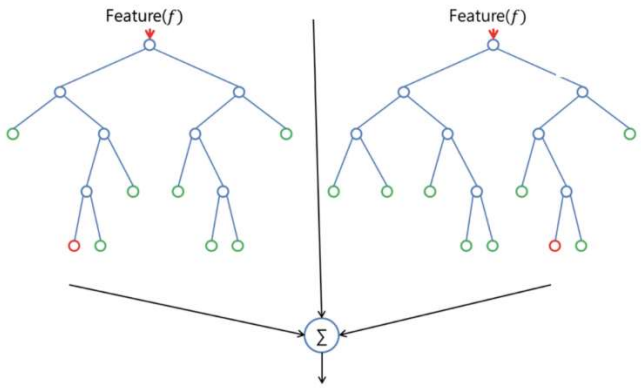


Fig 5.2 Random Forest

Algorithm:

Build Random Forest classi_er and complex network:
1: procedure Build RF Classifier(HINT;DrugTarget;GO □ slim)
2: do initialize
3: protein list annotate with target/nontarget labels
4: protein list annotate with GO-Slim
5: Remove proteins with < 1 GO-Slim annotation
6: NT (unseen data) randomly select 1,000 nontargets
7: T (train data) 50/50 split target/nontarget
8: TE (test data) 50/50 split target/nontarget
9: proteins matrix convert list to binary matrix
10: end initialize
11:

12: PPInetwork igraph builds complex network
13: NetStatistics calculate k-coreness for all nodes
14:
15: while
16: accuracy_accuracy from last iteration do
17: RF parameters (Num Trees, Cuto_)
18: RF accuracy test data (TE)
19: modify RF sampling classi_er accuracy
20: end while
21:
22: Candidate list RFmodel(unseen data (NT))
23: return PPInetwork;NetStatistics;RandomForest;CandList
24: end procedure

5.2.2 GRADIENT BOOSTING: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in form of an ensemble of weak prediction models, typically decision tress. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differential loss function.

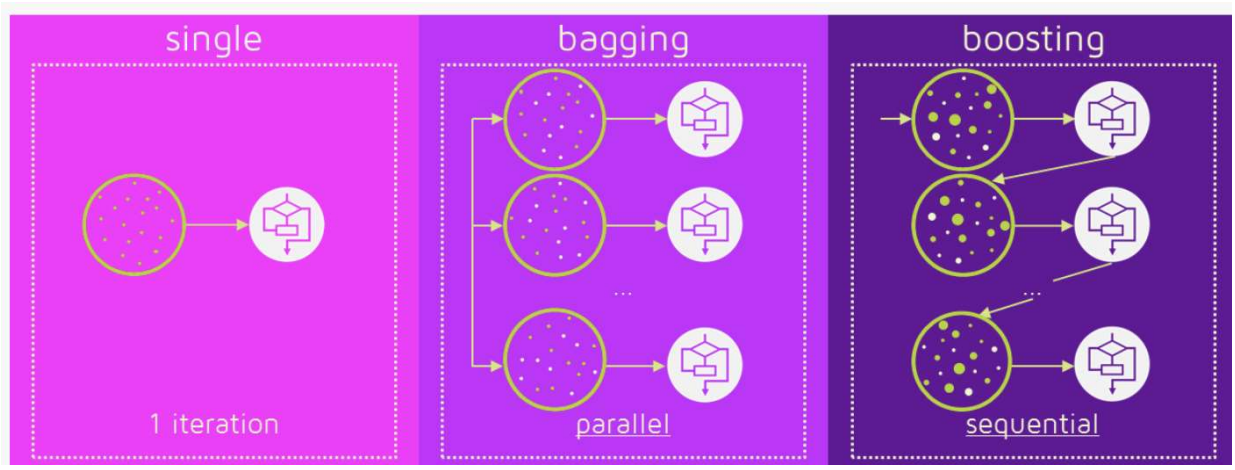


Fig 5.3 Gradient Boosting

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min \sum_{i=0}^{n} L(y, \gamma)$$

2. For $m = 1$ to M:

1. Compute so-called *pseudo-residuals*:

$$\gamma_{im} = -\left[\frac{\partial L(y, F(x))}{\partial F(x)}\right]_{F(x)-Fm-1(x)} \quad \text{for i=1,…..,n.}$$

2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set . $\{(x_i,\gamma_{im})\}^n_{i=1}$
3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:
4. Update the model:

$\gamma_m = \arg\min \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma_m h_m(x))$

3. Output $F_M(x)$

5.2.3 K-Nearest Neighbhours(KNN):

The nearest neighbor algorithm [23] is an instance-based lazy learning method, and one of the simplest understandings of machine learning algorithms. Hu et al. [24] proposed a protein sequence-based model, in which the classifier is implemented by the improved IBK (Instance-based k means) algorithm of the k-nearest neighbors, which overcomes the shortcomings of the recent neighbor algorithm, which is sensitive to some data.

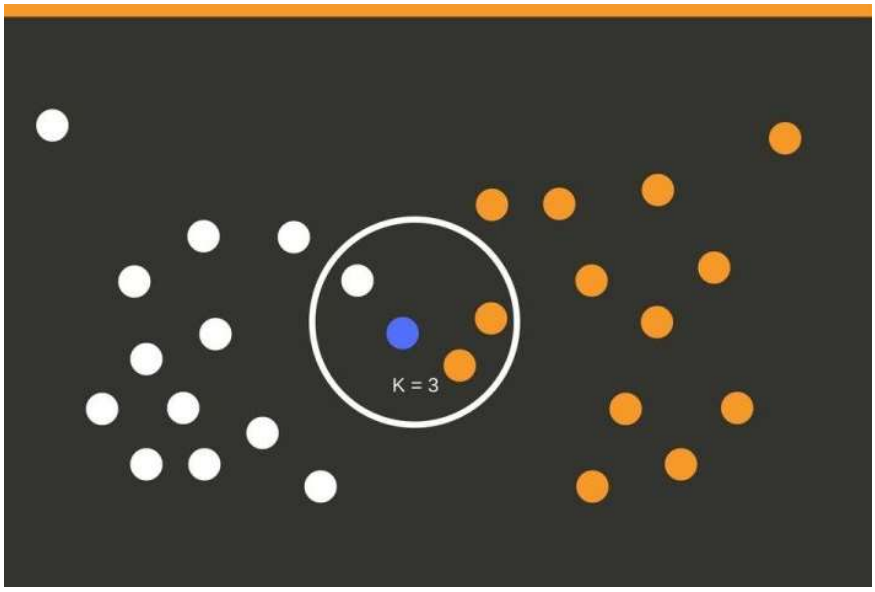


Fig 5.4 KNN

Algorithm:

1. Calculate "d(x, x_i)" i =1, 2, ….., **n**; where **d** denotes the Euclidean distance between the points.
2. Arrange the calculated **n** Euclidean distances in non-decreasing order.
3. Let **k** be a +ve integer, take the first **k** distances from this sorted list.
4. Find those **k**-points corresponding to these **k**-distances.

5. Let $\mathbf{k}_i$ denotes the number of points belonging to the i^{th} class among **k** points i.e. $k \geq 0$
6. If $k_i > k_j \; \forall \; i \neq j$ then put x in class i.

5.2.4 STACKING: Stacking is a way to ensemble multiple classification or regression model. There are many ways to ensemble models. Stacking, also called Super Learning [25] or Stacked Regression [26], is a class of algorithms that involves training a second-level "meta learner" to find the optimal combination of the base learners.

5.2.5 GAUSSIAN NAIVE BAYES: It is a special type of NB algorithm. It used when the features have continuous values. It's also assumed that all the features are following a gaussian distribution i.e, normal distribution. When plotted, it gives a bell-shaped curve which is symmetric about the mean of the feature values as shown below:

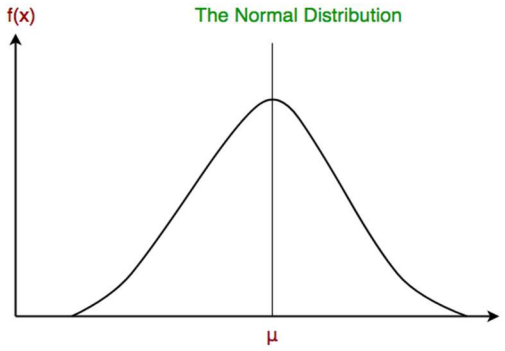


Fig 5.5 Normal Distribution

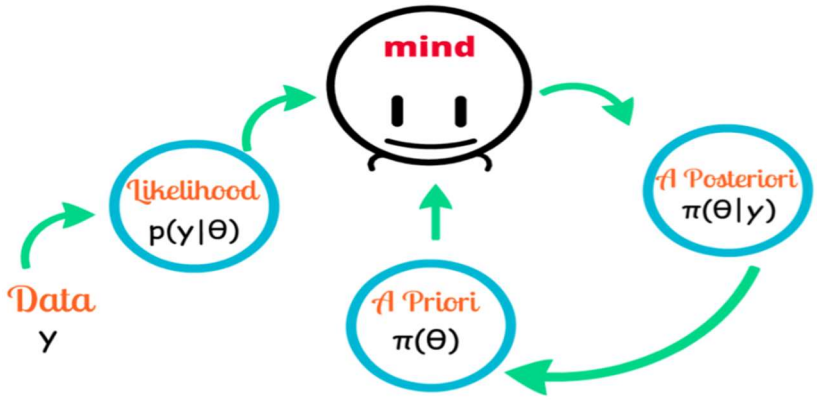


Fig 5.6 Naïve Bayes

5.2.6 XGBoost[44] is an open-source software library which provides a gradient boosting framework for C++, Java, Python[45] R[46] and Julia[47.] It is highly efficient**,** flexible and portable. It implements machine learning algorithms under the Gradient Boosting frame work. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

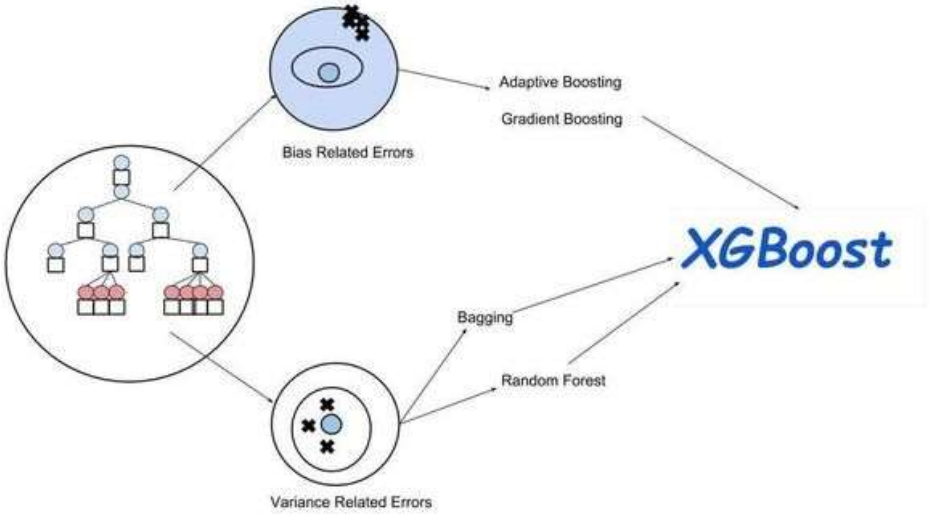


Fig 5.7 XGBoost

Algorithm:

1.Initialize model with a constant value:

$F_0(x) = \arg\min \sum_{i=0}^{n} L(y, \gamma)$

2.F0(x) should be a function which minimizes the loss function or MSE (mean squared error):

$\arg\min \sum_{i=0}^{n} L(y, \gamma) = \arg\min \sum_{i=0}^{n} L(y, \gamma)^2$

3.the function minimizes at the mean i=1ny:

$F_0(x) = \frac{\sum_{i=1}^{n} y}{n}$

5.2.7 MULTILAYER PERCEPTRON (MLP): It is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer are capable of approximating any continuous function.

Multilayer perceptions are often applied to supervised learning problems[3]: they train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (RMSE).

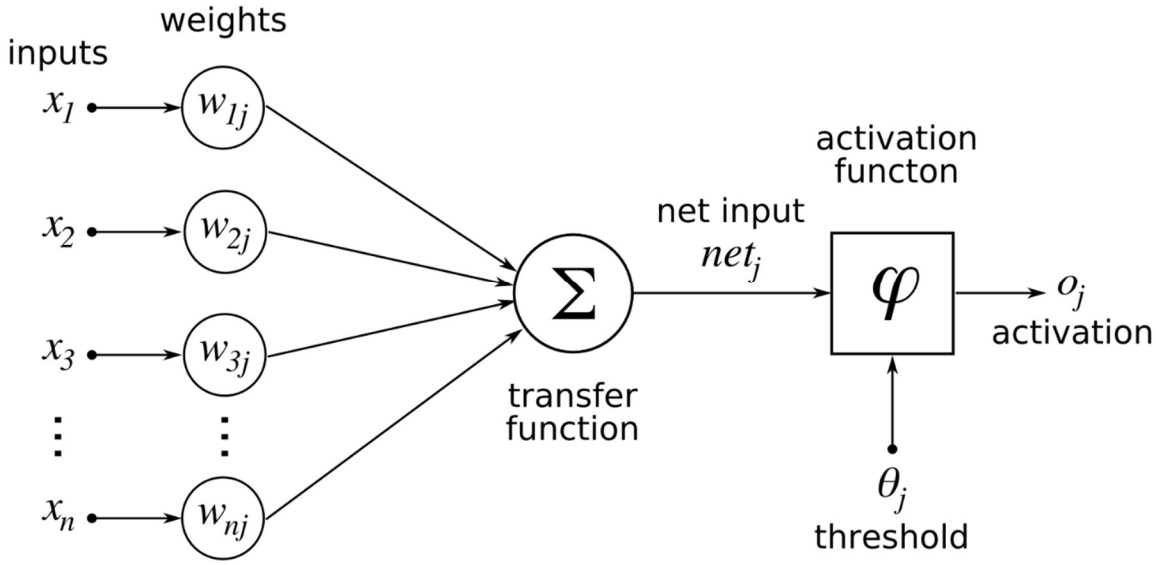


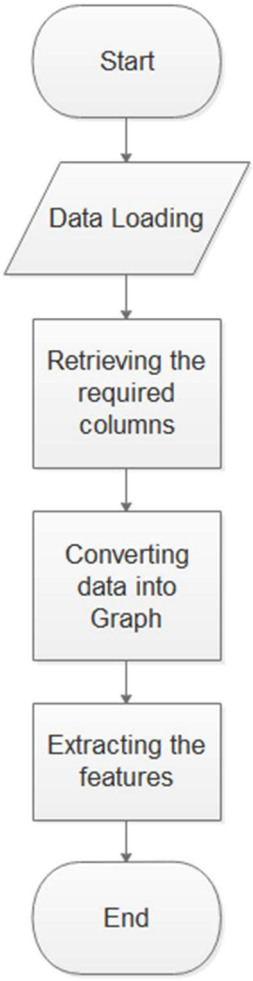
Fig 5.8 Multi Layered Perceptron

5.2.8 GRID SEARCH: It is a method to perform hyper-parameter optimisation, that is, it is a method to find the best combination of hyper-parameters (an example of an hyper-parameter is the learning rate of the optimiser), for a given model (e.g. a CNN) and test dataset. Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary. Grid-searching does NOT only apply to one model type. Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model. It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run. Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination.

CHAPTER 6

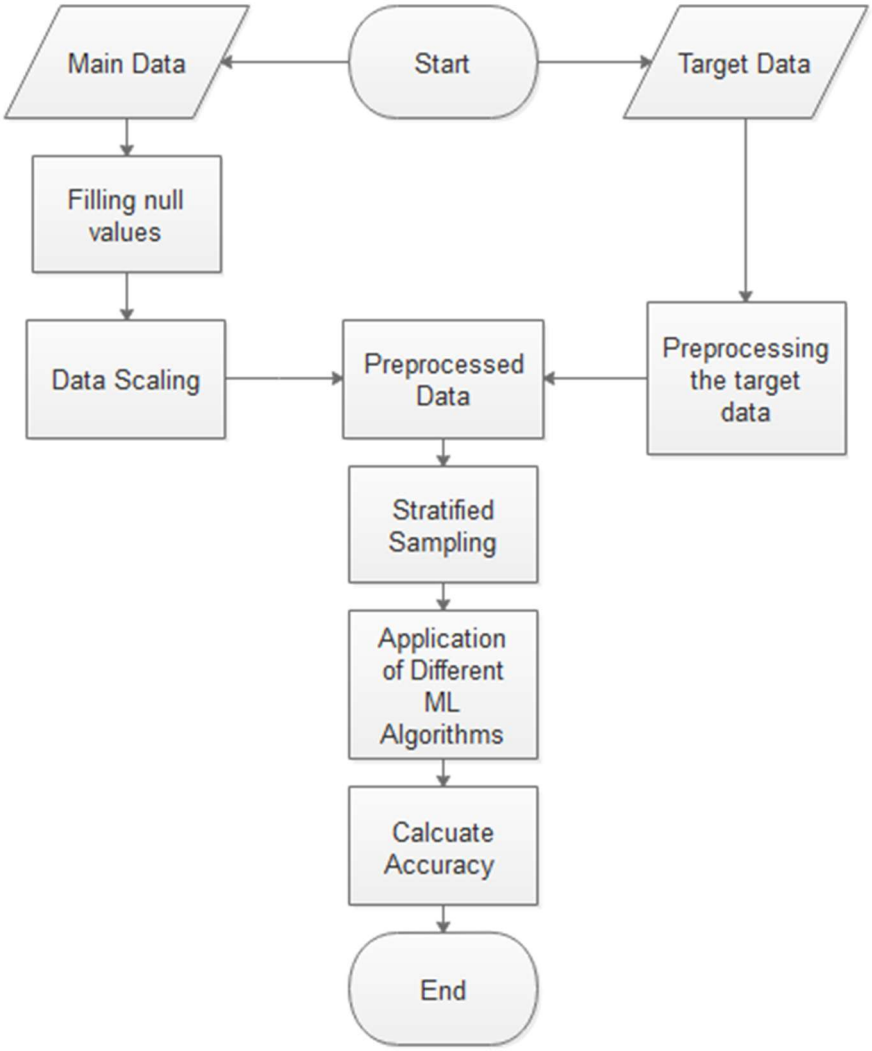
Methodology

6.1 Flowchart for feature extraction in R:



- In this project we have used R –studio to extract the features from the available nodes and edges .
- The features mentioned in the section 3 Network Topology are extracted using in built libraries of R.
- The libraries used for extraction are ‘sna’ , ‘CINNA’ , ‘igraph’ , ‘centiserve’ ,’ parbclus’.
- After the data loading the required columns are extracted . The data is converted into graph to extract the features.

6.2 Methodology Flowchart:



We have used python packages ‘pandas’,’sklearn’ ,’mlxtend’ for the Machine Learning implementation in python.

- Loading the data using pandas package.
- Target data preprocessing for targets as 1 and non-targets as 0.
- The target data is added to main data and further preprocessed
- The main data is checked for the null values.
- The null values are replaced with 0.
- Data scaling is done for the whole dataset.
- Strastified sampiling is applied to split into test and train data.
- The Machine Learning algorithms are applied for the test and train data.
- Acurracy has been calculated.
- we tried to optimise using Grid search.

CHAPTER 7
Code

7.1 R-Code for Feature Extraction:

In [1]:
```
library(sna)
library(CINNA)
library(igraph)
library(centiserve)
library(prabclus)
```

```
Loading required package: statnet.common

Attaching package: 'statnet.common'

The following object is masked from 'package:base':

    order

Loading required package: network
network: Classes for Relational Data
Version 1.13.0.1 created on 2015-08-31.
copyright (c) 2005, Carter T. Butts, University of California-Irvine
                    Mark S. Handcock, University of California -- Los Angeles
                    David R. Hunter, Penn State University
                    Martina Morris, University of Washington
                    Skye Bender-deMoll, University of Washington
 For citation information, type citation("network").
 Type help("network-package") to get started.

sna: Tools for Social Network Analysis
Version 2.4 created on 2016-07-23.
copyright (c) 2005, Carter T. Butts, University of California-Irvine
 For citation information, type citation("sna").
 Type help(package="sna") to get started.


Attaching package: 'igraph'
```

Loading the data

In [2]:
```
data1<-read.csv('C:/Users/vennela/Desktop/SCN5A_string_500-500-interactions.csv',sep=',',header=TRUE,stringsAsFactors = FALSE)
str(data1)
```

```
'data.frame':   18966 obs. of  15 variables:
 $ node1                                : chr  "CDKN1A" "MRPL2" "NUP62" "MRPL40" ...
 $ node2                                : chr  "CCND1" "MRPL24" "NUP214" "MRPL32" ...
 $ node1_string_internal_id             : int  4447168 4445791 4450981 4440454 4446201 4442484 4442794 4451051 4440454 4446389
...
 $ node2_string_internal_id             : int  4433391 4442484 4442243 4433279 4434620 4436789 4438233 4437667 4434635 4436109
...
 $ node1_external_id                    : chr  "9606.ENSP00000384849" "9606.ENSP00000373404" "9606.ENSP00000471191" "9606.ENSP0
0000333401" ...
 $ node2_external_id                    : chr  "9606.ENSP00000227507" "9606.ENSP00000354525" "9606.ENSP00000352400" "9606.ENSP0
0000223324" ...
 $ neighborhood_on_chromosome           : num  0 0.135 0 0 0.185 0.185 0 0 0 0 ...
 $ gene_fusion                          : num  0 0 0 0 0 0 0 0 0 0.007 ...
 $ phylogenetic_cooccurrence            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ homology                             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ coexpression                         : num  0.084 0.47 0.182 0.851 0.467 0.872 0.607 0.63 0.208 0.986 ...
 $ experimentally_determined_interaction: num  0.993 0.981 0.953 0.989 0.99 0.919 0.978 0.991 0.982 0.77 ...
 $ database_annotated                   : num  0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 ...
 $ automated_textmining                 : num  0.857 0.2 0.916 0 0.086 0.228 0 0.761 0.472 0.937 ...
 $ combined_score                       : num  0.999 0.999 0.999 0.999 0.999 0.999 0.999 0.999 0.999 0.999 ...
```

In [3]:
```
e1<-data1[1:2]
e1<-as.matrix(e1)
e1
```

node1	node2

Creating Graph

In [4]:
```
g<-graph_from_edgelist(e1,directed = FALSE)
```

In [5]:
```
nodes<-vertex.attributes(g)
```

In [6]:
```
d1<-graph_extract_components(g, directed=FALSE)
```

In [7]:
```
c1<-proper_centralities(g)
c1
```

```
 [1] "subgraph centrality scores"
 [2] "Topological Coefficient"
 [3] "Average Distance"
 [4] "Barycenter Centrality"
 [5] "BottleNeck Centrality"
 [6] "Centroid value"
 [7] "Closeness Centrality (Freeman)"
 [8] "ClusterRank"
 [9] "Decay Centrality"
[10] "Degree Centrality"
[11] "Diffusion Degree"
[12] "DMNC - Density of Maximum Neighborhood Component"
[13] "Eccentricity Centrality"
[14] "Harary Centrality"
[15] "eigenvector centralities"
[16] "K-core Decomposition"
[17] "Geodesic K-Path Centrality"
[18] "Katz Centrality (Katz Status Index)"
[19] "Kleinberg's authority centrality scores"
[20] "Kleinberg's hub centrality scores"
```

Maximum Neighbourhood Component

In [8]:
```
mnc<-calculate_centralities(g,include = c1[36])
mnc
```

$`MNC - Maximum Neighborhood Component` =

CDKN1A	105
CCND1	141
MRPL2	98
MRPL24	94
NUP62	49
NUP214	59
MRPL40	79
MRPL32	83
MRPL19	82
MRPL44	81
MRPL17	79
MRPL18	78
MRPL13	89
SNRNP70	53
SNRPD1	73
MRPS9	82
CDK1	140
CCNA2	102
SNRPG	81
CDC45	45
ORC2	34
MCM4	53

Degree

```
In [9]: deg<-calculate_centralities(g,include = c1[10])
        deg
```

$`Degree Centrality` =

105 141 98 94 49 60 82 83 82 81 79 78 89 55 73 82 140 102 81 46 34 54 44 80 55 71 101 101 62 56 51 76
192 86 67 80 80 82 82 54 79 80 72 80 52 51 100 80 78 81 61 62 58 45 155 82 51 49 85 71 97 79 93 87
27 16 79 84 67 111 46 40 93 74 82 84 79 58 97 84 83 41 100 80 21 111 59 77 75 78 73 40 166 47 45 99
63 96 103 80 80 92 90 104 50 46 80 58 50 63 36 48 84 85 68 36 84 102 100 75 64 74 81 47 79 78 28 64
15 115 108 53 81 59 108 73 78 161 129 42 115 34 38 51 52 96 112 113 52 82 80 28 21 132 52 9 82 79 87
49 67 48 71 20 33 87 79 50 62 49 55 205 78 45 79 74 63 71 58 49 13 16 72 53 34 21 45 55 66 79 44 56
69 29 28 133 59 66 42 83 79 119 28 214 98 43 83 78 78 160 52 128 51 74 21 43 79 58 27 79 126 46 72 48
79 26 81 45 43 38 68 31 20 118 11 80 81 53 65 52 53 23 87 27 120 77 72 47 37 65 79 74 57 99 41 78 99
49 80 42 55 46 53 83 55 80 86 162 21 78 136 47 93 31 42 54 79 70 61 51 23 25 78 51 116 11 24 80 115
29 28 37 106 22 37 30 15 71 107 54 82 49 22 78 31 85 81 21 98 31 140 53 79 26 24 78 50 134 80 78 26
48 62 177 39 70 15 78 62 72 80 82 51 65 91 79 78 81 27 33 23 39 67 76 63 77 46 10 54 21 14 16 156 25
38 36 24 107 74 83 36 78 29 58 21 18 46 48 62 173 51 80 29 14 23 47 25 53 45 34 61 64 50 78 78 68 49
28 41 45 47 56 95 79 5 20 75 41 28 17 88 103 56 42 39 29 24 70 99 37 72 14 18 18 35 46 78 54 72 61
37 66 69 44 14 43 33 56 36 41 34 35 19 63 45 68 78 52 68 50 61 29 21 36 33 49 39 98 14 18 11 26 60
67 94 72 9 27 63 29 68 44 22 33 56 13 6 20 46 39 104 28 41 66 22 22 30 55 9 14 50 6 42 93 62 18 25
53 21 21 51 80 75 44 7 2 56 17 30 24 55 12 15 10 24 48 48 7 24 36 42 72 25 22 26 63 53 32 63 16 83
24 33 200 74 61 87 38 44 43 13 7 5 53 13 28 47 56 40 34 21 13 29 3 39 72 23 9 12 51 85 55 82 120 34
27 33 61 61 47 27 15 19 15 21 15 78 116 35 5 28 52 48 44 56 51 53 17 20 24 8 26 67 37 70 62 83 54 26
29 10 65 4 27 35 34 38 27 15 43 15 18 26 48 9 75 23 57 10 22 46 23 10 45 16 32 15 17 32 8 5 34 24 9

Eccentricity

```
In [10]: ecc<-calculate_centralities(g,include = c1[13])
         ecc
```

$`Eccentricity Centrality` =

CDKN1A	3
CCND1	3
MRPL2	3
MRPL24	3
NUP62	4
NUP214	4
MRPL40	4
MRPL32	4
MRPL19	4
MRPL44	4
MRPL17	4
MRPL18	4
MRPL13	4
SNRNP70	4
SNRPD1	4
MRPS9	4
CDK1	3
CCNA2	3
SNRPG	4
CDC45	4
ORC2	4

Closeness Centrality

```
In [11]: cl<-calculate_centralities(g,include = c1[28])
         cl
```

$`Closeness centrality (Latora)` =

CDKN1A	386.166666666667
CCND1	405.833333333333
MRPL2	348.5
MRPL24	344.333333333333
NUP62	317.833333333333
NUP214	340.583333333333
MRPL40	320.166666666667
MRPL32	313.916666666667
MRPL19	336.083333333333
MRPL44	311.083333333333
MRPL17	299.833333333333
MRPL18	298.583333333333
MRPL13	350.666666666667
SNRNP70	336.416666666667
SNRPD1	324.916666666667
MRPS9	318.583333333333

Radiality

```
In [12]: rc<-calculate_centralities(g,include = c1[25])
         rc
```

$`Radiality Centrality` =

CDKN1A	3.01714285714286
CCND1	3.08285714285714
MRPL2	2.71428571428571
MRPL24	2.69
NUP62	2.59
NUP214	2.75428571428571
MRPL40	2.51428571428571
MRPL32	2.44714285714286
MRPL19	2.65285714285714
MRPL44	2.43142857142857
MRPL17	2.33428571428571
MRPL18	2.32571428571429
MRPL13	2.75714285714286
SNRNP70	2.73285714285714
SNRPD1	2.58142857142857
MRPS9	2.49285714285714
CDK1	3.09142857142857
CCNA2	2.98428571428571
SNRPG	2.63285714285714
CDC45	2.62428571428571
ORC2	2.56857142857143
MCM4	2.68857142857143
NEB	2.58

Clustering Centrality

```
In [14]: cc<-calculate_centralities(g,include = c1[21])
         cc
```

$`clustering coefficient` =

0.525641025641026 0.381560283687943 0.696191878813381 0.744452070464425 0.791666666666667 0.583050847457627
0.916591388136104 0.898031148986189 0.920204757603131 0.933024691358025 0.987990912041545 1 0.795199182839632
0.738047138047138 0.664764079147641 0.923215898825655 0.335046248715313 0.517957678120753 0.528703703703704
0.714009661835749 0.732620320855615 0.690426275331936 0.711416490486258 0.430379746835443 0.692929292929293
0.630181086519115 0.66019801980198 0.657227722772277 0.712321523003702 0.7597402597402 0.871372549019608 0.604561403508772
0.259871291448517 0.375649794801642 0.548168249660787 0.968670886075949 0.963291139240506 0.92261367058115 0.916892502258356
0.720475192173305 0.977280103862382 0.963291139240506 0.661189358372457 0.563291139240506 0.720211161387632 0.738039215686274
0.672727272727273 0.969303797468354 1 0.950617283950617 0.585792349726776 0.318878900052882 0.312764670296431
0.38989898989899 0.284206116464181 0.916892502258356 0.834509803921569 0.860544217687075 0.5296918767507 0.640241448692153
0.539089347079038 0.987990912041545 0.424731182795699 0.443197006148089 0.501424501424501 0.85 0.987990912041545
0.891853126792886 0.544549977385798 0.457493857493857 0.581642512077295 0.544871794871795 0.536933146330061
0.506849315068493 0.924721469436917 0.874928284566839 0.987990912041545 0.683000604960677 0.533505154639175
0.595238095238095 0.431677931237144 0.751219512195122 0.669090909090909 0.968670886075949 0.257142857142857
0.278787878787879 0.726475745178258 0.507860560492139 0.608648648648649 0.586413586413586 0.67427701674277 0.470512820512821
0.245052939028843 0.839962997224792 0.895959595959596 0.68315811172954 0.621095750128008 0.715789473684211 0.634304207119741
0.968354430379747 0.955696202531646 0.434543717152413 0.44769038701623 0.623412994772218 0.619591836734694 0.721739130434783

Average Distance

```
In [15]: ad<-calculate_centralities(g,include = c1[3])
         ad
```

$`Average Distance` =

CDKN1A	1.98433048433048
CCND1	1.91880341880342
MRPL2	2.28632478632479
MRPL24	2.31054131054131
NUP62	2.41025641025641
NUP214	2.24643874643875
MRPL40	2.48575498575499
MRPL32	2.55270655270655
MRPL19	2.34757834757835
MRPL44	2.56837606837607
MRPL17	2.66524216524217
MRPL18	2.67378917378917
MRPL13	2.24358974358974
SNRNP70	2.26780626780627
SNRPD1	2.41880341880342
MRPS9	2.50712250712251
CDK1	1.91025641025641
CCNA2	2.01709401709402
SNRPG	2.36752136752137
CDC45	2.37606837606838
ORC2	2.43162393162393
MCM4	2.31196581196581
NEB	2.42022792022792
TTN	2.0997150997151

Coreness

```
In [16]: co<-calculate_centralities(g,include = c1[16])
         co
```

```
$`K-core Decomposition` =
CDKN1A  43
CCND1   43
MRPL2   78
MRPL24  78
NUP62   43
NUP214  43
MRPL40  78
MRPL32  78
MRPL19  78
MRPL44  78
MRPL17  78
MRPL18  78
MRPL13  78
SNRNP70 43
SNRPD1  43
MRPS9   78
CDK1    43
CCNA2   43
SNRPG   43
CDC45   39
ORC2    31
MCM4    43
NEB     32
TTN     32
```

Bottleneck

```
In [19]: bottle_data<-bottleneck(g)
         bottle_data
```

```
CDKN1A  73
CCND1   166
MRPL2   260
MRPL24  170
NUP62   8
NUP214  55
MRPL40  9
MRPL32  22
MRPL19  77
MRPL44  15
MRPL17  21
MRPL18  20
MRPL13  205
SNRNP70 22
SNRPD1  13
MRPS9   17
CDK1    357
CCNA2   64
```

Edge Percolated Component

```
In [ ]: ec<-calculate_centralities(g,include = c1[33])
        ec
```

Closeness Centrality

```
In [ ]: clvi<-calculate_centralities(g, include = c1[39])
        clvi
```

```
In [ ]: df<-data.frame(mnc,deg,ecc,cl,rc,bc,cc,ad,co,modularity,denmnc,sc)#creating dataframe
```

```
In [ ]: write.csv(df,'new_Features.csv') #saving file
```

```
In [ ]:
```

7.2 Python code:

In [1]:
```
import numpy as nb
import pandas as nd
from sklearn.model_selection import train_test_split
from pandas import DataFrame
```

Loading Data

In [2]:
```
drug_data = nd.read_csv('C:/Users/Freeware Sys/Desktop/new_Features1.csv',sep=',')
drug_data.head() #first 5 rows of the data
```

Out[2]:

	Proteins	MNC...Maximum.Neighborhood.Component	Degree.Centrality	Eccentricity.Centrality	Closeness.centrality..Latora.	Radiality.Centrality	Shortest.Paths.Be
0	AAAS	44	44	4	302.833333	2.475714	
1	ABCC8	29	30	3	332.833333	2.774286	
2	ABCC9	43	43	3	330.333333	2.715714	
3	ACTA2	72	72	3	372.833333	2.997143	
4	ACTB	120	120	3	405.166667	3.137143	

In [3]:
```
drug_data.columns #checking the column names
```

Out[3]:
```
Index(['Proteins', 'MNC...Maximum.Neighborhood.Component', 'Degree.Centrality',
       'Eccentricity.Centrality', 'Closeness.centrality..Latora.',
       'Radiality.Centrality', 'Shortest.Paths.Betweenness.Centrality',
       'clustering.coefficient', 'Average.Distance', 'K.core.Decomposition',
       'modularity', 'DMNC...Density.of.Maximum.Neighborhood.Component',
       'Stress.Centrality', 'MCC', 'classes'],
      dtype='object')
```

In [4]:
```
drug_data[drug_data.isnull().any(axis=1)] #checking for null values
```

Out[4]:

	Proteins	MNC...Maximum.Neighborhood.Component	Degree.Centrality	Eccentricity.Centrality	Closeness.centrality..Latora.	Radiality.Centrality	Shortest.Paths.E
92	CBY3	1	1	3	267.166667	2.294286	

In [5]:
```
drug_data.shape #to check the dimensions of data
```

Out[5]:
```
(701, 15)
```

Filling the Null Values

In [6]:
```
drug_data.fillna({'clustering.coefficient':0},inplace=True)
```

In [7]:
```
data=drug_data.iloc[:,1:14] #creating object for the features in data
```

In [8]:
```
data.head()
```

Out[8]:

	MNC...Maximum.Neighborhood.Component	Degree.Centrality	Eccentricity.Centrality	Closeness.centrality..Latora.	Radiality.Centrality	Shortest.Paths.Betweennes
0	44	44	4	302.833333	2.475714	
1	29	30	3	332.833333	2.774286	
2	43	43	3	330.333333	2.715714	
3	72	72	3	372.833333	2.997143	1

Target Classification

In [10]:
```
data1 =  nd.read_csv('C:/Users/Freeware Sys/Desktop/project/new datasets/703ids_scn5a_drugtargetinfo.csv',sep=',')
data_target = data1.loc[:,['Gene names','Cross-reference (DrugBank)']]
#changing column names
data_target=data_target.rename(columns = {'Gene names':'Gene names' ,'Cross-reference (DrugBank)':'target'})
data_target.head()
```

Out[10]:

	Gene names	target
0	CDKN1A hCG_15367	NaN
1	CDKN1A CAP20 CDKN1 CIP1 MDA6 PIC1 SDI1 WAF1	NaN
2	CCND1	NaN
3	CCND1 BCL1 PRAD1	DB01169;
4	CCND1	NaN

In [11]:
```
data_target.columns
```

Out[11]:
```
Index(['Gene names', 'target'], dtype='object')
```

In [12]:
```
data_target.target.notnull()
```

Out[12]:
```
0       False
1       False
2       False
3        True
4       False
5       False
6       False
```

In [14]:
```
y= data_target.drop_duplicates('Gene names') #droping duplicates from data
c=data_target[data_target.target.notnull()] #checking for null values.Returns True or False

data2=c.append(y) #appending dataframe
data2.drop_duplicates('Gene names')#droping duplicates from data
```

Out[14]:

	Gene names	target
3	CCND1 BCL1 PRAD1	DB01169;
71	CDK1 CDC2 CDC28A CDKN1 P34CDC2	DB04014;DB05037;DB03496;DB02950;DB02052;DB0211...
72	CCNA2 CCN1 CCNA	DB08463;DB07137;DB06948;DB08248;DB08309;DB0824...
131	SLC25A15 ORC1 ORNT1 SP1855	DB00129;
329	TP53 P53	DB08363;DB00945;DB05404;
432	CKS1B CKS1 PNAS-143 PNAS-16	DB02681;
473	RB1	DB00030;DB00071;
549	CAMK2B CAM2 CAMK2 CAMKB	DB07168;
610	CDK4	DB03496;DB09073;DB02733;DB11730;
661	CDK2 CDKN2	DB07054;DB07504;DB08463;DB07501;DB07137;DB0813...
672	SYK	DB08846;DB02010;
687	SKP1 EMC19 OCP2 SKP1A TCEB1L	DB06980;DB06981;DB06982;DB07950;DB01750;
714	CDK6 CDKN6	DB07379;DB03496;DB09073;DB11730;
739	RYR2	DB09085;
766	SRC SRC1	DB08564;DB06882;DB06883;DB08192;DB04739;DB0796...
809	SIRT1 SIR2L1	DB05073;
819	TNNC2	DB01373;DB01023;DB04682;

In [15]:
```
data2.loc[data2['target'].notnull(), 'target'] = 1 #target
data2.loc[data2['target'].isnull(), 'target'] = 0 #non-target
#data1.to_csv('target_classes.csv')
data2.drop_duplicates('Gene names') #droping duplicates from data
```

Out[15]:

	Gene names	target
3	CCND1 BCL1 PRAD1	1
71	CDK1 CDC2 CDC28A CDKN1 P34CDC2	1
72	CCNA2 CCN1 CCNA	1
131	SLC25A15 ORC1 ORNT1 SP1855	1
329	TP53 P53	1
432	CKS1B CKS1 PNAS-143 PNAS-16	1
473	RB1	1
549	CAMK2B CAM2 CAMK2 CAMKB	1
610	CDK4	1
661	CDK2 CDKN2	1
672	SYK	1
687	SKP1 EMC19 OCP2 SKP1A TCEB1L	1
714	CDK6 CDKN6	1
739	RYR2	1

```
In [17]: data2.to_csv('target_classified.csv',sep=',') #saving to file
```

Scaling the Data

```
In [18]: from sklearn.preprocessing import StandardScaler
feature_scaler=StandardScaler()
data1=feature_scaler.fit_transform(data)
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:625: DataConversionWarning: Data with input dty
pe int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\base.py:462: DataConversionWarning: Data with input dtype int64, floa
t64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

```
In [19]: data1.tolist() #converting to list
```

```
Out[19]: [[-0.2917231725446723,
  -0.30468744084245697,
  1.2521392338136397,
  -0.8656933976058412,
  -0.9847798594004175,
  -0.4419971522999529,
  1.6333329239899173,
  0.9847798597218638,
  0.42588652265662513,
  1.0338357602262038,
  0.16191081512546207,
  -0.01594948554937182,
  -0.40501664206654436],
 [-0.7414987090999843,
```

```
In [21]: data1=DataFrame.from_records(data1) #converting to Dataframe
```

```
In [22]: type(data1) #checking type of data
```

```
Out[22]: pandas.core.frame.DataFrame
```

```
In [23]: proteins=nd.DataFrame(drug_data.iloc[:,0]) #Assigning nodes   to proteins object
```

```
In [24]: classes=nd.DataFrame(drug_data.iloc[:,14]) #Assigning labels  to classes object
```

```
In [25]: f=[proteins,data1,classes]
```

```
In [26]: data_scaled=nd.concat(f,axis=1) #scaling the data
```

```
In [27]: data_scaled.head()
```

Out[27]:

	Proteins	0	1	2	3	4	5	6	7	8	9	10	11	12	classes
0	AAAS	-0.291723	-0.304687	1.252139	-0.865693	-0.984780	-0.441997	1.633333	0.984780	0.425887	1.033836	0.161911	-0.015949	-0.405017	0
1	ABCC8	-0.741499	-0.726556	-0.789139	0.050178	0.398343	-0.227271	-0.819179	-0.398343	-0.445782	-1.254750	-0.828001	-0.211947	-0.334901	1
2	ABCC9	-0.321708	-0.334821	-0.789139	-0.026145	0.127013	-0.176767	-0.166933	-0.127013	-0.173386	-0.491888	0.107574	0.788109	-0.321917	1
3	ACTA2	0.547858	0.539049	-0.789139	1.271339	1.430722	0.866287	-0.906261	-1.430722	-0.173386	-0.491888	0.389614	0.636705	-0.072616	0
4	ACTB	1.987140	1.985455	-0.789139	2.258445	2.079268	1.970747	-1.124841	-2.079268	0.425887	0.270974	0.871291	0.173359	0.831097	0

```
In [28]: data_scaled.columns=[drug_data.columns]
```

```
In [28]: data_scaled.columns=[drug_data.columns]
```

```
In [29]: data_scaled.head()
```

Out[29]:

	Proteins	MNC...Maximum.Neighborhood.Component	Degree.Centrality	Eccentricity.Centrality	Closeness.centrality..Latora.	Radiality.Centrality	Shortest.Paths.Be
0	AAAS	-0.291723	-0.304687	1.252139	-0.865693	-0.984780	
1	ABCC8	-0.741499	-0.726556	-0.789139	0.050178	0.398343	
2	ABCC9	-0.321708	-0.334821	-0.789139	-0.026145	0.127013	
3	ACTA2	0.547858	0.539049	-0.789139	1.271339	1.430722	
4	ACTB	1.987140	1.985455	-0.789139	2.258445	2.079268	

Stratified Sampling

```
In [35]: from sklearn.model_selection import StratifiedShuffleSplit
```

```
In [36]: split= StratifiedShuffleSplit(n_splits=1,test_size=.30,random_state=0)
```

```
In [37]: for train_index, test_index in split.split(data_scaled,data_scaled['classes']):
    strat_train_set = data_scaled.loc[train_index]
    strat_test_set = data_scaled.loc[test_index]
```

Random Forest

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
model1 = RandomForestClassifier(n_estimators=500,random_state=47,max_depth=9)
model1.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[38]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=9, max_features='auto', 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, n_estimators=500, n_jobs=None,
            oob_score=False, random_state=47, verbose=0, warm_start=False)
```

In [40]:
```
predict1 = model1.predict(strat_test_set.iloc[:,1:14])
from sklearn.metrics import accuracy_score,confusion_matrix
accuracy_score(strat_test_set['classes'],predict1) #checking accuracy
```

Out[40]:
```
0.8199052132701422
```

In [45]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict1))
```

Out[45]:

	0	1
0	144	8
1	30	29

Grid Search on Random Forest

In [42]:
```
import time
from sklearn.model_selection import GridSearchCV
parameters_grid={'n_estimators':[150,200,250,300,500],'max_depth':[5,8,10,20],'bootstrap':[True,False]}
grid=GridSearchCV(estimator=model1,param_grid=parameters_grid,cv=3,n_jobs=-1)
starttime=time.time()
grid_result_rf=grid.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
print(grid_result_rf.best_score_,grid_result_rf.best_params_)
time.time()-starttime
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: The default of th
e `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results
when test-set sizes are unequal.
  DeprecationWarning)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:740: DataConversionWarning: A column-vecto
r y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

```
0.7755102040816326 {'bootstrap': True, 'max_depth': 5, 'n_estimators': 200}
```

Out[42]:
```
48.74677586555481
```

In [47]:
```
predict_grid_rf=grid_result_rf.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict_grid_rf)
```

Out[47]:
```
0.8199052132701422
```

In [48]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict_grid_rf))
```

Out[48]:

Gradient Boosting

In [49]:
```
from sklearn.ensemble import GradientBoostingClassifier
model2 = GradientBoostingClassifier(random_state=89,learning_rate=0.1)
model2.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[49]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              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, n_estimators=100,
              n_iter_no_change=None, presort='auto', random_state=89,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

In [51]:
```
predict2 = model2.predict(strat_test_set.iloc[:,1:14])
from sklearn.metrics import accuracy_score,confusion_matrix
accuracy_score(strat_test_set['classes'],predict2)
```

Out[51]:
```
0.8151658767772512
```

In [52]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict2))
```

Out[52]:

	0	1
0	139	13
1	26	33

Grid Search on Gradient Boosting

In [53]:
```
import time
from sklearn.model_selection import GridSearchCV
parameters_grid={'n_estimators':[150,200,250,300,500],"learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2]}
grid=GridSearchCV(estimator=model2,param_grid=parameters_grid,cv=3,n_jobs=-1)
starttime=time.time()
grid_result_gb=grid.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
print(grid_result_gb.best_score_,grid_result_gb.best_params_)
time.time()-starttime
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
0.7408163265306122 {'learning_rate': 0.01, 'n_estimators': 250}
```

Out[53]:
```
37.95989012718201
```

In [54]:
```
predict_grid_gb=grid_result_gb.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict_grid_gb)
```

Out[54]:
```
0.8104265402843602
```

In [55]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict_grid_gb))
```

Out[55]:

	0	1
0	141	11
1	29	30

KNN

In [56]:
```
from sklearn.neighbors import KNeighborsClassifier
model4 = KNeighborsClassifier(n_neighbors=32)
model4.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[56]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=32, p=2,
           weights='uniform')
```

In [57]:
```
predict4 = model4.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict4)
```

Out[57]:
```
0.8056872037914692
```

In [58]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict4))
```

Out[58]:

	0	1
0	145	7
1	34	25

SVM

In [59]:
```
from sklearn.svm import LinearSVC
model3 = LinearSVC(tol=0.5,random_state=42,penalty='l2')
model3.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[59]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=42, tol=0.5, verbose=0)
```

In [60]:
```
predict3 = model3.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict3)
```

Out[60]:
```
0.8104265402843602
```

In [61]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict3))
```

Out[61]:

	0	1
0	141	11
1	29	30

Naive Bayes

In [62]:
```
from sklearn.naive_bayes import BernoulliNB,GaussianNB
```

In [63]:
```
model5=BernoulliNB()
model5.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[63]:
```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

In [64]:
```
predict5 = model5.predict(strat_test_set.iloc[:,1:14])
from sklearn.metrics import accuracy_score,confusion_matrix
accuracy_score(strat_test_set['classes'],predict5)
```

Out[64]:
```
0.7772511848341233
```

In [65]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict5))
```

Out[65]:

	0	1
0	123	29
1	18	41

In [66]:
```
model6=GaussianNB()
model6.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[66]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [67]:
```
predict6=model6.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict6)
```

Out[67]:
```
0.7867298578199052
```

In [68]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict6))
```

Out[68]:

	0	1
0	127	25
1	20	39

Xgboost

In [69]:
```
from xgboost import XGBClassifier
```

In [70]:
```
model7=XGBClassifier()
```

In [71]:
```
model7.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:219: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:252: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[71]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [72]:
```
predict7=model7.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict7)
```

Out[72]:
```
0.8199052132701422
```

In [73]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict7))
```

Out[73]:

	0	1
0	138	14
1	24	35

Stacking using RF,GB,XGBOOST algorithms

In [74]:
```
from mlxtend.classifier import StackingClassifier,StackingCVClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
```

In [75]:
```
lr = LogisticRegression()
main_model = StackingClassifier(classifiers=[model1,model2,model7], meta_classifier=lr)
```

In [76]:
```
for model, label in zip([model1,model2,model7,main_model],
                        [  'Random Forest',
                           'Gradient Bosting',
                           'Xgboost',
                           'StackingClassifier']):
    scores = model_selection.cross_val_score(model, strat_test_set.iloc[:,1:14],strat_test_set['classes'], cv=3, scoring='accura
cy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:528: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:528: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:528: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
```

```
Accuracy: 0.79 (+/- 0.03) [Random Forest]
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Multi Layer Perceptron(MLP)

In [78]:
```
from sklearn.neural_network import MLPClassifier
```

In [79]:
```
model_mp=MLPClassifier(activation='relu',solver='sgd',hidden_layer_sizes=(25,25),random_state=123)
```

In [80]:
```
model_mp.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:916: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarning: Stoc
hastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[80]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(25, 25), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=123, shuffle=True, solver='sgd', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
```

In [81]:
```
predict_mp=model_mp.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict_mp)
```

Out[81]: 0.7962085308056872

In [82]:
```
nd.DataFrame(confusion_matrix(strat_test_set['classes'],predict_mp))
```

Out[82]:

	0	1
0	141	11

Grid Search on Multi Layer Perceptron(MLP)

In [83]:
```
import time
from sklearn.model_selection import GridSearchCV
parameters_grid={'activation':['relu','identity','logistic','tanh'],'solver':['sgd','adam','lbfgs'],
                 'hidden_layer_sizes':[(25,25),(30,30),(25,15),(15,15),(10,15),(5,5),(10,10),(15,20)]}
grid=GridSearchCV(estimator=model_mp,param_grid=parameters_grid,cv=3,n_jobs=-1)
starttime=time.time()
grid_result_mlp=grid.fit(strat_train_set.iloc[:,1:14],strat_train_set['classes'])
print(grid_result_mlp.best_score_,grid_result_mlp.best_params_)
time.time()-starttime
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: The default of th
e `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results
when test-set sizes are unequal.
  DeprecationWarning)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:916: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

```
0.7755102040816326 {'activation': 'tanh', 'hidden_layer_sizes': (15, 15), 'solver': 'adam'}
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarning: Stoc
hastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[83]:
```
71.41214609146118
```

In [84]:
```
predict_grid_mlp=grid_result_mlp.predict(strat_test_set.iloc[:,1:14])
accuracy_score(strat_test_set['classes'],predict_grid_mlp)
```

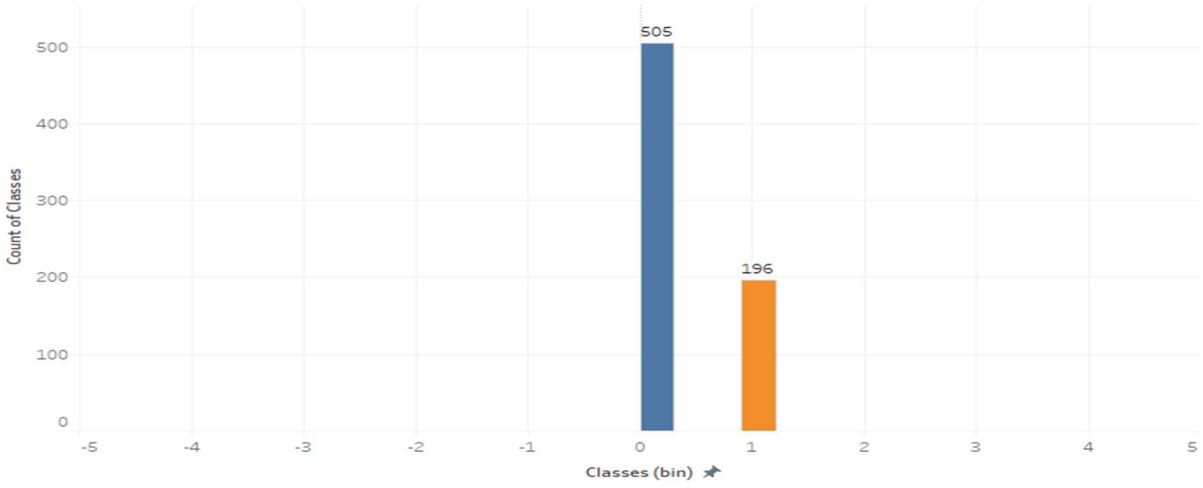
Out[84]:
```
0.7772511848341233
```

Stacking using all Grid Search algorithms(Grid_RF,Grid_GB,Grid_MLP)

In [86]:
```
lr1 = LogisticRegression()
main_model = StackingClassifier(classifiers=[grid_result_rf,grid_result_gb,grid_result_mlp], meta_classifier=lr1)
```

In [87]:
```
for model, label in zip([grid_result_rf,grid_result_gb,main_model,grid_result_mlp],
                        [  'Grid_Random Forest',
                           'Grid_Gradient Bosting',
                           'Grid_MLP',
                           'StackingClassifier']):
    scores = model_selection.cross_val_score(model, strat_test_set.iloc[:,1:14],strat_test_set['classes'], cv=3, scoring='accura
cy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: The default of th
e `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results
when test-set sizes are unequal.
  DeprecationWarning)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:740: DataConversionWarning: A column-vecto
r y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: The default of th
e `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results
when test-set sizes are unequal.
  DeprecationWarning)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:740: DataConversionWarning: A column-vecto
r y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: The default of th
e `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results
when test-set sizes are unequal.
  DeprecationWarning)
C:\Users\Freeware Sys\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:740: DataConversionWarning: A column-vecto
```



CHAPTER 8
Results

8.1 Results:

Sl.No	Algorithm	Accuracy
1	Random Forest	81.9
2	Gradient Boosting	81.5
3	K-Nearest Neighbours	80.5
4	Support Vector Machine	81.0
5	Gaussian Naive Bayes	78.6
6	XGboost	81.9
7	Multi-Layered perceptron	79.6
8	Stacking on RF , GB, XGboost	82.0
9	Grid search on RF	81.9
10	Grid Search on GB	81.0
11	Grid Search on MLP	77.7
12	Stacking on Grid Search algorithms	80.0

8.2 Discussion:

On applying some of the machine learning algorithm, we found the accuracy for each model as shown in the table above and each algorithm approximately provided accuracy of 81%, on applying stacking on all the algorithms we found the accuracy to be slightly improved .

FUTURE DIRECTION

Protein-protein interaction network is highly dynamic [43] and studying the evolution of protein-protein interaction networks is one of the central problems of systems biology, the results of such researches are crucial for a better understanding of the evolution of living systems and could be used for protein interaction and function prediction.

The efficiency in drug Target Protein prediction is crucial part of drug Development Programs which will help in making effective drugs for diseases.

Different Machine Learning algorithms giving different accuracy, better accuracy can be obtained by implementing more efficient algorithms.

REFERENCES

1. Hopkins AL, Groom CR (2002) The druggable genome. Nature Rev Drug Discovery 1: 727–730.

2. Xu H, Xu HY, Lin MZ, Wang W, Li ZM, et al. (2007) Learning the drug target-likeness of a protein. Proteomics 7: 4255–4263. PMID: 17963289

3. Protein-protein interactions (PPIs) handle a wide range of biological processes, including cell-to-cell interactions and metabolic and developmental control [1].

4. A. Zhang, Protein Interaction Networks-Computational Analysis, Cambridge University Press, New York, NY, USA, 2009.

5. J. De Las Rivas, C. Fontanillo, and F. Lewitter, “Protein-protein interactions essentials: key concepts to building and analyzing interactome networks,” PLoS Computational Biology, vol. 6, no. 6, Article ID e1000807, 2010.

6. De Las Rivas J, Fontanillo C (June 2010). "Protein-protein interactions essentials: key concepts to building and analyzing interactome networks". PLoS Computational Biology. 6 (6): e1000807. Bibcode:2010PLSCB...6E0807D. doi:10.1371/journal.pcbi.1000807. PMC 2891586. PMID 20589078

7. Linton C. Freeman: Centrality in networks: I. Conceptual clarification. Social Networks 1:215-239, 1979. http://leonidzhukov.ru/hse/2013/socialnetworks/papers/freeman79-centrality.pdf

8. BRANDES, U. & ERLEBACH, T. 2005. Network Analysis: Methodological Foundations, U.S. Government Printing Office.

9.In our system, ϵ is set to be 1.7, which is close to 1.67, the ϵ-value as we assume the neighborhood sub-network has a four-community. [LIN, C.-Y. 2008]

10. 3.1.2 R: R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.

11. 03. Ben Hur A, Ong CS, Sonnenburg S, Schölkopf B, Rätsch G. Support Vector Machines and Kernels for Computational Biology. PLoS. comput. biol. 2008;4(10):e1000173. [PMC free article] [PubMed]

12. 70. Guo Y, Yu L, Wen Z, Li M. Using support vector machine combined with auto covariance to predict proteinprotein interactions from protein sequences. Nucleic Acids Res. 2008;36(9):3025–3030. [PMC free article] [PubMed]

13. 104. Lo S, Cai C, Chen Y, Chung M. Effect of training datasets on support vector machine prediction of protein-protein interactions. Proteomics. 2005;5(4):876 – 884. [PubMed]

14. 105. Dohkan S, Koike A, Takagi T. Improving the Performance of an SVM-Based Method for Predicting Protein-Protein Interactions. In. Silico Biol. 2006;6:515–529. [PubMed]

15. 106. Rashid M, Ramasamy S, PS Raghava G. A simple approach for predicting protein-protein interactions. Curr.Pro.Pept. Sci. 2010;11(7):589–600. [PubMed]

16. 107. Kuncheva LI. Combining Pattern Classifiers Methods and Algorithms (Kuncheva LI 2004) [book review]. IEEE Transactions on Neural Networks. 2007;18(3):964–964.

17. Ben Hur A, Ong CS, Sonnenburg S, Schölkopf B, Rätsch G. Support Vector Machines and Kernels for Computational Biology. PLoS. comput. biol. 2008;4(10):e1000173

18. Witten I H, Frank E, Hall M A. Data Mining Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques. Morgan Kaufmann. 2011

19. Chen X, Wang M, Zhang H. The use of classification trees for bioinformatics. Wiley Interdisciplinary Reviews. J of Data Mini and Know Disc. 2011;1(1):55–63. [PMC free article] [PubMed]

20. Xia JF, Han K, Huang DS. Sequence-based prediction of protein-protein interactions by means of rotation forest and autocorrelation descriptor. Protein Pept Lett. 2010;17(1):137. [PubMed]

21. Chen XW, Liu M. Prediction of protein-protein interactions using random decision forest framework. Bioinformatics. 2005;21 (24):4394–400. [PubMed]

22. Qi Y, Klein-Seetharaman J, Bar-Joseph Z. Random forest similarity for protein-protein interaction prediction from multiple sources. Pac Symp Biocomput. 2005:531–542. [PubMed]

23. Cover, T.M. Nearest Neighbour Pattern Classification. IEEE Trans. Inf. Theory 1967, 13, 21–27. [Google Scholar] [CrossRef]

24. Hu, S.S.; Chen, P.; Wang, B.; Li, J. Protein binding hot spots prediction from sequence only by a new ensemble learning method. Amino Acids 2017, 49, 1–13. [Google Scholar] [CrossRef] [PubMed]

25. Mark J van der Laan, Eric C Polley, and Alan E Hubbard. "Super Learner." Journal of the American Statistical Applications in Genetics and Molecular Biology. Volume 6, Issue 1. (September 2007).

26. Leo Breiman. "Stacked Regressions." Machine Learning, 24, 49-64 (1996)

27. CHEN, S.-H., CHIN, C.-H., WU, H.-H., HO, C.-W., KO, M.-T. & LIN, C.-Y. cyto-Hubba: A Cytoscape plug-in for hub object analysis in network biology. 20th International Conference on Genome Informatics, 2009.

28. LIN, C.-Y., CHIN, C.-H., WU, H.-H., CHEN, S.-H., HO, C.-W. & KO, M.-T. 2008. Hubba: hub objects analyzer—a framework of interactome hubs identification for network biology. Nucleic acids research, 36, W438-W443

29. LIN, C.-Y., CHIN, C.-H., WU, H.-H., CHEN, S.-H., HO, C.-W. & KO, M.-T. 2008. Hubba: hub objects analyzer—a framework of interactome hubs identification for network biology. Nucleic acids research, 36, W438-W443.

30. Wolfram Research, Inc., Mathematica, Version 10.0, Champaign, IL (2014). http://reference.wolfram.com/language/ref/RadialityCentrality.html_

31. R language and environment

Hornik, Kurt (2017-10-04). "R FAQ". The Comprehensive R Archive Network. 2.1 What is R? Retrieved 2018-08-06.

R Foundation

Hornik, Kurt (2017-10-04). "R FAQ". The Comprehensive R Archive Network. 2.13 What is the R Foundation? Retrieved 2018-08-06.

The R Core Team asks authors who use R in their data analysis to cite the software using:

R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/

32. Szklarczyk D, Franceschini A, Wyder S, Forslund K, Heller D, Huerta-Cepas J, Simonovic M, Roth A, Santos A, Tsafou KP, Kuhn M, Bork P, Jensen LJ, von Mering C (2015). "STRING v10: protein-protein interaction networks, integrated over the tree of life". Nucleic Acids Res. 43 (Database issue): D447–52. doi:10.1093/nar/gku1003. PMC 4383874. PMID 25352553.

33. Franceschini A, Szklarczyk D, Frankild S, Kuhn M, Simonovic M, Roth A, Lin J, Minguez P, Bork P, von Mering C, Jensen LJ (2013). "STRING v9.1: protein-protein interaction networks, with increased coverage and integration". Nucleic Acids Res. 41 (Database issue): D808–15. doi:10.1093/nar/gks1094. PMC 3531103. PMID 23203871.

34. Szklarczyk D, Franceschini A, Kuhn M, Simonovic M, Roth A, Minguez P, Doerks T, Stark M, Muller J, Bork P, Jensen LJ, von Mering C (2011). "The STRING database in 2011: functional interaction networks of proteins, globally integrated and scored". Nucleic Acids Res. 39 (Database issue): D561–8. doi:10.1093/nar/gkq973. PMC 3013807. PMID 21045058.

35. Jensen LJ, Kuhn M, Stark M, Chaffron S, Creevey C, Muller J, Doerks T, Julien P, Roth A, Simonovic M, Bork P, von Mering C (2009). "STRING 8—a global view on proteins and their functional interactions in 630 organisms". Nucleic Acids Res. 37 (Database issue): D412–6. doi:10.1093/nar/gkn760. PMC 2686466. PMID 18940858.

36. von Mering, C; Jensen, LJ; Kuhn, M; Chaffron, S; Doerks, T; Kruger, B; Snel, B & Bork, P (2007). "STRING 7—recent developments in the integration and prediction of protein interactions". Nucleic Acids Res. 35 (Database issue): D358–62. doi:10.1093/nar/gkl825. PMC 1669762. PMID 17098935.

37. Snel, B; Lehmann, G; Bork, P & Huynen, MA (2000). "STRING: a web-server to retrieve and display the repeatedly occurring neighbourhood of a gene". Nucleic Acids Res. 28 (18): 3442–4. doi:10.1093/nar/28.18.3442. PMC 110752. PMID 10982861.

38. https://www.drugbank.ca/about

39. UniProt, Consortium. (January 2015). "UniProt: a hub for protein information". Nucleic Acids Research. 43 (Database issue): D204–12. doi:10.1093/nar/gku989. PMC 4384041. PMID 25348405.

40. https://cytoscape.org/what_is_cytoscape.html

41. http://hub.iis.sinica.edu.tw/cytoHubba/

42. https://www.python.org/doc/essays/blurb/

43. Plant protein-protein interaction network and interactome. Zhang Y, Gao P, Yuan JS Curr Genomics. 2010 Mar; 11(1):40-6.[PubMed] [Ref list]

44. "GitHub project webpage".

45. "Python Package Index PYPI: xgboost". Retrieved 2016-08-01

46. "CRAN package xgboost". Retrieved 2016-08-01.

47. ^"Julia package listing xgboost". Retrieved 2016-08-01