Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Master Thesis Bioinformatics

The Graph-based gcForest Algorithm with Application to Biomedical Data Analysis

Zhang Junjie

August 9, 2020

Reviewers

Prof. Dr. Nico Pfeifer
(Bioinformatik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Prof. Dr. Manfred Claassen
(Biologie/Medizin)
Universitätsklinikum Tübingen
Universität Tübingen

Nachname, Vorname: Zhang, Junjie
Title of thesis
The Graph-based gcForest Algorithm with Application to Biomedical Data Analysis
Eberhard Karls Universität Tübingen
Thesis period: February 2020 - August 2020

Abstract

Ensemble learning is a machine learning approach where multiple base learners are trained to solve the same problem and their outputs are aggregated to obtain better results. Recently, a novel ensemble learning framework gcForest has been proposed. gcForest consists of two parts, Cascade Forest that is used to process the input features layer-by-layer and Multi-Grained Scanning that is used to perform representation learning and to enhance the Cascade Forest. However, the sliding window-based Multi-Grained Scanning in gcForest is not expressive enough to capture the structure or connectivity pattern when data has an internal graph structure relationship, such as cancer-related gene expression data.

In this thesis, we propose graph-based gcForest, which is an ensemble learning framework that is capable of scanning on a graph. In graph-based gcForest, inspired by the second-order random walk in Node2Vec, we propose a non-linear graph-based Multi-Grained Scanning method, which provides us with the flexibility of feature representation based on the intrinsic graph structure. And we argue that the added flexibility in Multi-Grained Scanning is the key to learning richer representations. We compare the graph-based gcForest with some other machine learning algorithms for the classification tasks on the YEAST dataset, Single Cell dataset, and a synthetic GAMETES dataset, to validate its effectiveness. In addition, we discuss some influential hyperparameters that are unique to graph-based gcForest.

Acknowledgements

First and foremost, I am grateful to my advisor, Dr. Ralf Eggeling, without encouragement and support from him, this thesis can not be finished. I benefit a lot from our weekly meetings and I am very fortunate to be able to work on this thesis under his guidance.

I am as well thankful to Prof. Dr. Nico Pfeifer for giving me the opportunity to start a machine learning project that I was really excited to participate in. Likewise, I would like to thank Prof. Dr. Manfred Claassen for his insightful comments and thoughtful feedback. I thank TCML-cluster that I often work on.

Finally, I deeply thank my friends and family. Without their support, I cannot finish this thesis.

This thesis is completed during the pandemic. I hope the coronavirus will disappear soon and people's lives will return to normal soon.

Contents

List of Abbreviations

DT	Decision Trees
RF	Random Forests
SVM	Support Vector Machine
BN	Bayesian Network
CF	Cascade Forest
LR	Linear Regression
DNN	Deep Neural Network
OOB	Out of bag
GBDT	Gradient Boosting Decision Tree
BFS	Breadth-first Sampling
DFS	Depth-first Sampling
Bnlearn	Bayesian Network learning and inference
SILGGM	Statistical Inference of Large-scale Gaussian Graphical Model
GGM	Gaussian graphical mode
B_NW_SL	The bivariate nodewise scaled Lasso
D-S_NW_SL	The de-sparsified nodewise scaled Lasso
D-S_GL	The de-sparsified graphical Lasso
GFC_SL	GGM estimation with false discovery rate control using scaled Lasso
GFC_L	GGM estimation with false discovery rate control using Lasso
ACCU	Accuracy
MCC	The Matthews Correlation Coefficient
SNP	Single Nucleotide Polymorphism

Chapter 1

Introduction

Over the last decade, we have seen explosive growth in the number of highly complex and variable biological data which mainly came from six different domains: genomics, proteomics, microarrays, system biology, evolution, and text mining[1]. As a result, the role of bioinformatics has been dramatically enhanced. However, with the development of high-throughput technologies, such as next-generation sequencing, the large amounts of data can be produced rapidly in a short time, which is too overwhelming for conventional bioinformatics tools. The problem is the challenge in computational biology, which requires the methods capable of extracting useful biological information from very complex biological data. In recent years, due to increased computer speed, machine learning methods have emerged in various fields and have been successfully applied to the biomedical field. For instance, machine learning models are used to annotate genes, including their untranslated regions(UTRs), introns and exons[2]; Some machine learning techniques aim to discover the mechanisms underlying gene expression[3]; As well as learning to discover the gene-gene interactions in disease data[4]. A more detailed discussion of machine learning methods applied to the biomedical field can be found elsewhere[5].

1.1 Biomedical Data

As mentioned above, the goal of learning a model is to use it to make predictions on unseen biomedical data. Thus, it is very necessary for us to understand the properties of biomedical data. For instance, biomedical data may contain the following properties:

Multi-source. In biomedical data, features could contain different types of data, such as gene expression profiles, protein-protein interactions, DNA methylation profiles, protein expression profiles, and a genomic sequence, etc.

Features could be continuous (e.g., gene expression values) or categorical (e.g., genome annotation). Labels could be continuous(e.g., the size of cells) or categorical(e.g., the stage of cancer)[6].

High-dimensional. Biomedical data is often high-dimensional data. For example, gene expression profiles often contain tens of thousands of genes. However, especially when biomedical data is presented with fewer samples and very high dimensions, it is very easy to cause the so-called 'curse of dimensionality'.

Graph structure. The features of most biomedical datasets could be modeled as network and their relationships can be captured as graph structure, such as metabolic pathways, gene regulatory networks.

1.2 Machine learning for Biomedical data

Machine learning techniques are increasingly becoming powerful research tools in various domains, bioinformatics is no exception. These tools can be classified as supervised or unsupervised. Broadly speaking, supervised learning algorithms involve two tasks, regression task, and classification task, respectively. And for unsupervised learning algorithms, the task is usually a clustering problem.

With the development of machine learning technology and the improvement of the ability to collect increasingly large and diverse data from different levels of biological systems, we can devise approaches that take advantage of the properties of biological datasets to build a more accurate and more robust model to analyze biological problems. Over the past decade, there are a very large number of well-known machine learning algorithms being applied to the biomedical field. For example, Decision Tree(DT)[7], Random Forest(RF)[8], Support Vector Machine(SVM)[9], Bayesian Network(BN)[10] etc. However, the performance of conventional machine learning algorithms relies heavily on data representations called features and feature engineering requires experts in the field to identify which features are useful for the given task, which remains difficult currently[11].

Recently, Deep learning(DL) has been successful in many fields, particularly in the field of speech and image recognition[12, 13]. Bioinformatics also benefits from the development of Deep Learning[14], such as Leung et al.[15], Mamoshina et al.[16] discussed the applications of Deep Learning in bioinformatics research. Although Deep Learning is very powerful, it also has drawbacks. First, Deep Learning is very hungry for data, however, in biomedical fields, data is sometimes very scarce and precious, which makes Deep Learning does not perform well on small-scale high-dimensional biomedical data.

Second, Deep Learning requires adjusting many parameters, which makes the Deep Learning model much more complex compared to conventional machine learning models.

1.3 gcForest

It is well known that representation learning ability is very critical for Deep Learning. Inspired by this recognition, Zhou, and Feng propose a novel ensemble tree approach, which is called gcForest[17]. As shown in Figure 2.6, its structure contains two ensemble components. One is the Multi-Grained Scanning, which potentially enables gcForest to be contextual or structural aware of high-dimensional data. The other one is the Cascade Forest, which consists of a multi-layer structure that is similar to Deep Neural Network(DNN), but each layer of gcForest is made up of different Random Forests instead of neurons. Unlike Deep Neural Network, in which multi-layer neural network learns representations by employing forward and backward propagation algorithms, Cascade Forest assembles many Random Forests to learn representations, which can be interpreted as an ensemble of ensembles but does not include the process of backpropagation.

As shown in Zhou and Feng's paper, there are some experiments showing that gcForest is able to gain good performance on different data from different domains. In particular, on small-scale datasets, gcForest achieves competitive performance to Deep Neural Network, whereas the training configuration of gcForest is much simpler and the training time of gcForest is lower than that of Deep Neural Network. Given its potential, we hope to apply gcForest to biological datasets.

1.4 Contributions

Nevertheless, the Multi-Grained Scanning of gcForest has the limitation that it can only linearly scan the raw input data. This linear scan does not produce a good feature representation when dealing with raw input data with an intrinsic graph structure. Considering the network features inherent in many biological datasets and inspired by the Node2Vec[18], we therefore proposed graph-based gcForest, which is a variation of gcForest that operates on graphs. Intuitively, our approach uses a second-order random walk(refer to Figure 2.10) to generate network neighborhoods(samples) of nodes[18], and then uses these node sequences to sample over the original input data, which is called graph-based nonlinear scan. In graph-based gcForest, we still employ the Cascade Forest, but the Multi-Grained Scanning uses a graph-based

nonlinear scan to obtain the original input feature's representations, enabling graph-based gcForest to have community awareness. Moreover, our approach gives the user more flexibility in choosing the scanning strategy. By choosing appropriate parameters governing our scanning strategy[18], we can have an intuitive interpretation of the results.

In our experiments, we focus on the classification task in biological datasets. The results show that graph-based gcForest achieves better performance compared to other benchmark algorithms.

1.5 Structure

This thesis is structured as follows: First there will be a brief introduction into gcForest and Node2Vec in general and graph inference. Then the graph-based gcForest will be explained in some more detail in Chapter 3. In Chapter 4, 5 and 6, the developed algorithm to analyze each test case is presented, followed by a comprehensive description of the used data or material. In Chapter 7, we attempt to answer some basic open questions and finish with some conclusions. In addition, we will describe future work and prospects.

Chapter 2

Background

In this chapter, we introduce the literature involved and the basis of the corresponding machine learning methods. First, we introduce the development of ensemble learning and the details of the emerging ensemble learning approach gcForest[17]. Secondly, we explain the representation learning on graphs and the related method Node2Vec[18]. Finally, we present three methods for graph structural learning on biomedical data, Bnlearn[19], SILGGM[20], and STRING database[21].

2.1 Tree-based methods

Decision Trees(DTs) are a non-parametric supervised learning method used for classification and regression, which was proposed by Breiman et al.[22] in 1984. Take the classification problem as an example, as illustrated in Figure 2.1[1], there are three classes and two X variables. The left panel shows the partitions of the data points and the right panel plots the corresponding decision tree structure[23].

In constructing the tree, the training datasets are recursively partitioned into smaller subsets. The goal is assigning to the split point a set of features based on the splitting criterion until every case remaining at a node is of the same class. These measurement metrics can express the homogeneity of the features within the training data.

The most popular splitting criteria are listed below:

Entropy Formally, Entropy H is described by

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{2.1}$$

[1]This figure is from Figure 1 of Loh's paper[23]

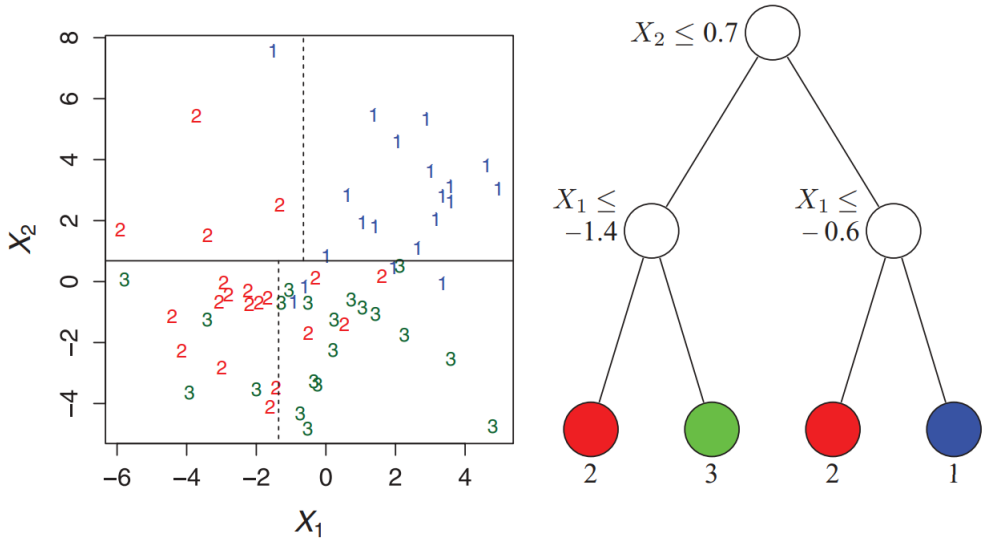


Figure 2.1: Illustrations of the partitions(lest) and decision tree structure(right) with three classes labeled 1, 2, and 3. At each intermediate node, a case goes to the left child node if and only if the condition is satisfied. The predicted class is given beneath each leaf node.

where p_i is the probability of the records that have class y_i and n is the number of possible classes. If all cases belong to the same class, entropy H will be 0, which indicates that the current node is terminal and a classification decision has been reached.

Gini impurity Gini impurity can be interpreted as the probability of an arbitrary record being wrongly classified when it is randomly selected, which can be formulated as

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2 \tag{2.2}$$

where p_i is again the probability of records that have class i and n is the number of possible classes. If the Gini impurity is 0, it means that all elements belong to one class. If the Gini impurity is 1, it means that all elements are randomly distributed across all classes. If the Gini impurity is 0.5, it means that all elements are evenly distributed in some classes.

Since the decision tree is able to obtain high accuracy while ensuring interpretability, it becomes a popular machine learning tool. However, it should be noted that the decision tree is easy to overfit due to its own nature or learn biased trees when some classes dominate. Therefore, ensemble methods are proposed to solve these problems and obtain performance improvements.

2.1.1 Ensemble tree methods

Ensemble tree methods is a supervised learning algorithm. The goal of ensemble tree methods is to construct an ensemble of individual decision trees that are diverse and yet accurate. There are two main ensemble tree methods: bagging[24], boosting[25].

Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The most representative algorithm is Random Forest[26]. Since we primarily use Random Forests in the thesis, the details of the Random Forests are presented as follows:

Random Forest

The general workflow of the Random Forest algorithm is depicted in Figure 2.2. Given a training set, decision trees are constructed on the bootstrap sample drawn with replacement from the training set. When growing the tree, not all cases will be used, and those that are not used are considered to be out-of-bag(OOB) data that is thus used to estimate the classification error. Also, not every feature will be used to grow the tree. After the forest is completed, the predictions of all trees are aggregated by taking a majority vote.

The next section describes the parameters of the Random Forest that were primarily used in the thesis.

Number of trees This hyperparameter in Random Forest means how many of trees compose a forest. As the number of trees goes up, the Random Forest may perform better.

Number of candidate features This hyperparameter in Random Forest means the number of features considered at each split. The default value is $\sqrt{n}$ for classification(with n as the total number of the features). In a completely Random Forest, this value is set to 1.

Boosting predictors is a method for training single decision tree, sequentially, each trying to correct its predecessor. There exists some methods such as AdaBoost[27], Gradient Boosting Decision Tree (GBDT)[28].

2.1.2 gcForest

Deep Learning methods are one of the most popular machine learning techniques recently. They have already surpassed human expert performance in some fields. But, one of the primary limitations of Deep Learning is that a large amount of data is required and a lot of hyperparameters tuning is needed. However, as proposed by Zhi-Hua Zhou[17], Deep Forest, henceforth referred to

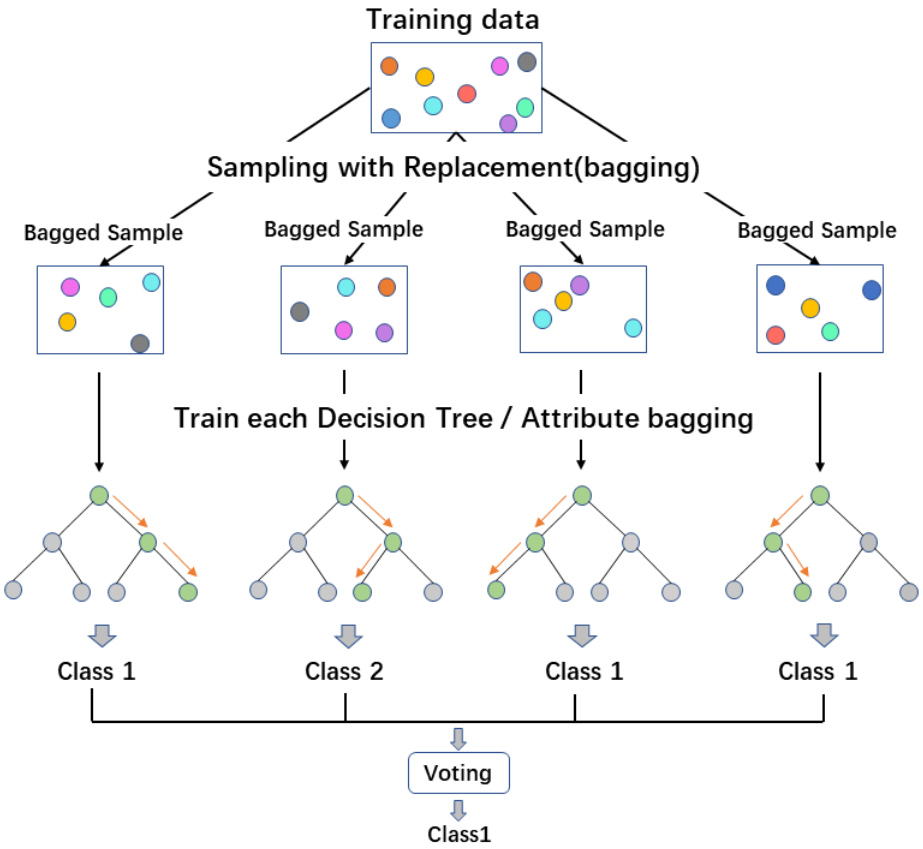


Figure 2.2: Illustration of the Random Forest algorithm

as gcForest, is a decision tree ensemble approach, which can work well on small scale dataset with much fewer hyperparameters than Deep Learning methods.

The overall architecture of gcForest includes two parts, the cascade forest structure, and the Multi-Grained Scanning.

Cascade Forest

Cascade Forest is inspired by the way Deep Neural Network represents features, that is, layer-by-layer processing of raw features. As illustrated in Figure 2.3[2], each level of Cascade Forest includes two Random Forests and two complete Random Forests. Each complete Random Forest contains 500 completely random trees[29], and the trees are generated by randomly selecting features to be partitioned at each node of the tree, and growing tree until each leaf node only contains instances of the same class. Each Random Forest contains 500 trees, and the trees are generated by randomly selecting $\sqrt{d}$ number of features as a candidate(d is the number of input features) and then

[2]This figure is from Figure 1 of Zhou's paper[17]

the feature with best *gini* value is selected as the segmentation. The number of trees in each forest and the depth of level is a hyperparameter. The type of base classifier(refers to Random Forest and completely Random Forest) is also a hyperparameter, which can be Support Vector Machine(SVM), boosting tree methods, and so on.

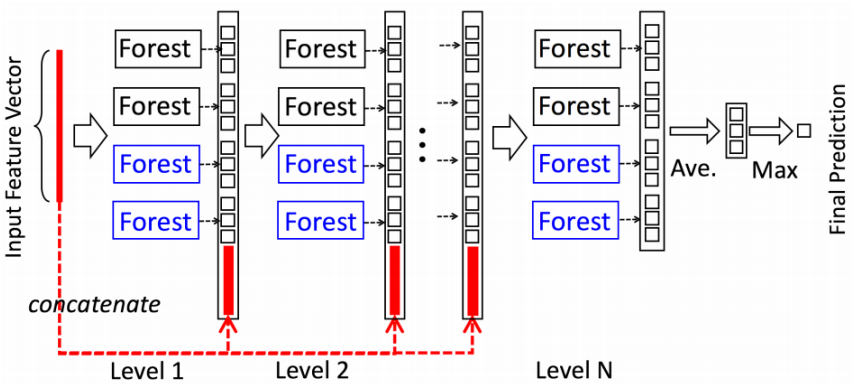


Figure 2.3: Illustration of a the cascade forest structure. Each level contains two Random Forests(black) and two complete Random Forests(blue), the red ones are the original features. Suppose there is triple classification problem; Therefore, each forest generates a three-dimensional class vector, which is then concatenated for re-representation of the original features.

Given an instance, each forest will estimate the class distribution, by counting the percentage of different classes of training examples at the leaf node where the concerned instance falls, and then Random Forest aggregates the decisions of each tree, as illustrated in Figure 2.4[3].

Suppose there is a triple classification problem, three-dimensional estimated class distribution outputs generated by each forest in the same layer form a class vector, which is then concatenated to the input feature to be input to the next layer of the cascade. For example, as illustrated in Figure2.3 [4], suppose there are 100 original features, which are highlighted in red color, then each of four forests will generate a three-dimensional class vector. Thus, the number of augmented features is (3×4 =) 12, and the number of input features to the next layer of the cascade is ($100+3\times4$ =) 112.

In order to reduce the risk of overfitting, the class vectors produced by each forest are generated through k-fold cross-validation. In the process of training, the training samples will be divided into two samples, growing samples and es-

[3]This figure is from Figure 2 of Zhou's paper[17]
[4]This figure is from Figure 1 of Zhou's paper[17]

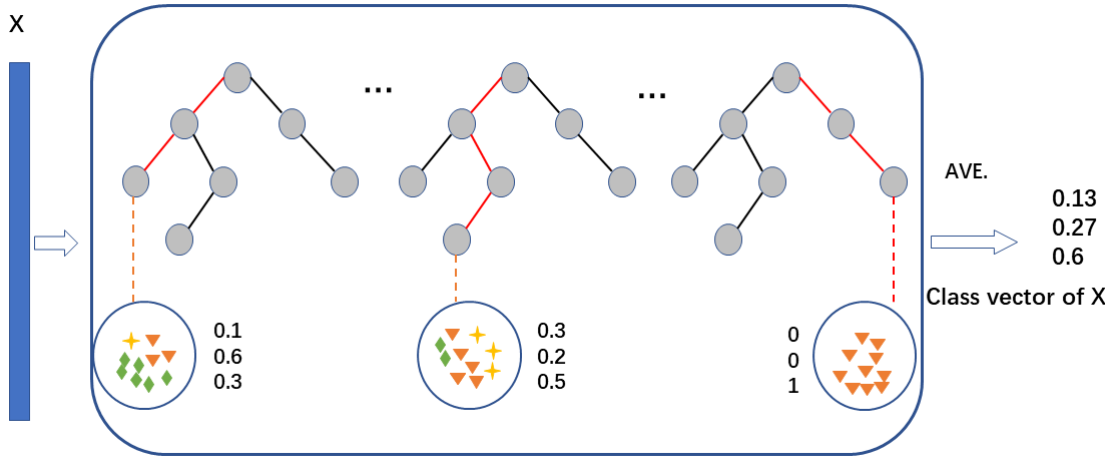


Figure 2.4: Illustration of a class vector generation. Different marks in leaf nodes imply different classes.

timating samples[5]. During the growth of the cascade, the overall performance of the current layer will be evaluated on the estimating sets and the growing progress will be automatically terminated once there is no performance gain, which is called adaptive determination. After the number of cascade layers being adaptively determined, the cascade is then retrained based on the growing and estimating samples.

Multi-Grained Scanning

Inspired by the way Deep Neural Networks process feature relationships, The author employs a Multi-Grained Scanning strategy to enhance Cascade Forest. In order to study the separate contribution of the Cascade Forest and Multi-Grained Scanning, The authors of gcForest compared gcForest with Cascade Forest on MNIST, GTZAN, and sEMG datasets. As shown in Table 2.1[6], the Multi-Grained Scanning procedure enhances the Cascade Forest[17].

Table 2.1: Results of gcForest w/wo Multi-Grained Scanning.

	MNIST	GTZAN	sEMG
gcForest	99.26%	65.67%	71.30%
CascadeForest	98.02%	52.33%	48.15%

[5]Some experimental datasets are given with training/validation sets. The author wants to avoid confusion, thus they call the subsets generated from training set as growing/estimating sets.

[6]This table is from Table 8 of Zhou's paper[17]

As illustrated in Figure 2.5[7], the sliding window extracts low-dimensional feature vectors by scanning raw input sequentially and then a series of the subset of raw input will be trained by a completely Random Forest and a Random Forest to obtain the class vectors which will then be concatenated as transformed features.

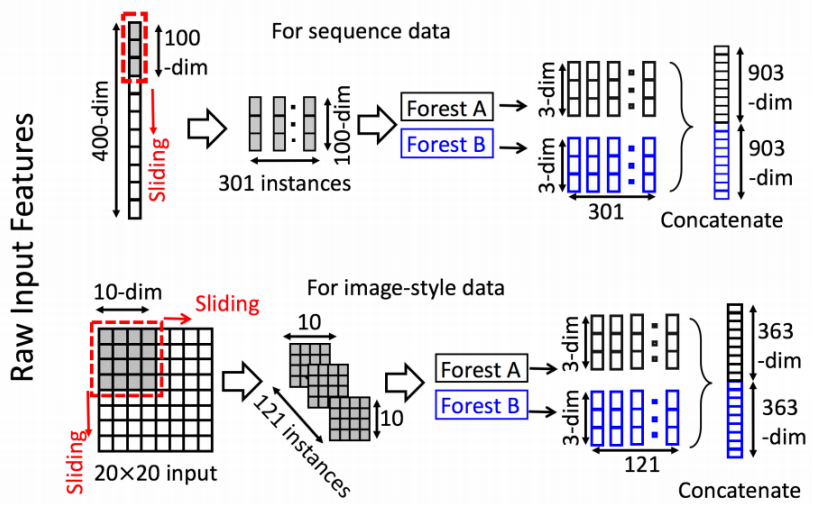


Figure 2.5: Illustration of the sliding window based scanning. Suppose there are three classes to be predicted, original features have 400 dimensions and silding window has 100 dimensions.

Suppose there are three classes to be predicted and a 100-dimensional window is used. For sequence data, sliding the window will produce 301 100-dimensional subsets of the raw input, leading to a $(301 \times 3 \times 2 =)1806$-dimensional transformed input corresponding to the original 400-dimensional raw input. For image data, such as a 20×20 panel of 400 image pixels, then a 10×10 window will produce 121 10×10 small images.

Since we don't believe gcForest will outperform Deep Neural Network methods when processing image data, we will ignore the discussion of image data. In this thesis, we mainly discuss the work on sequence data.

Overall structure of gcForest

As illustrated in Figure 2.6[8], gcForest merges the Multi-Grained Scanning and Cascade Forest structure. As described in Cascade Forest structure section and Multi-Grained Scanning section, suppose that raw input features have 400 dimensions, and there are three different sliding window sizes are used for

[7]This figure is from Figure 3 of Zhou's paper[17]
[8]This figure is from Figure 4 of Zhou's paper[17]

Multi-Grained Scanning. Given n training samples, When the window sliding size is 100, this scan produces 301 training subsets of 100 dimensions. These data will be firstly transformed by Random Forest and completely Random Forest. If there are three classes to predict, then 1806-dimensional transformed features will be obtained as described in the section of Multi-Grained Scanning and then will be fed into the Cascade Forest.

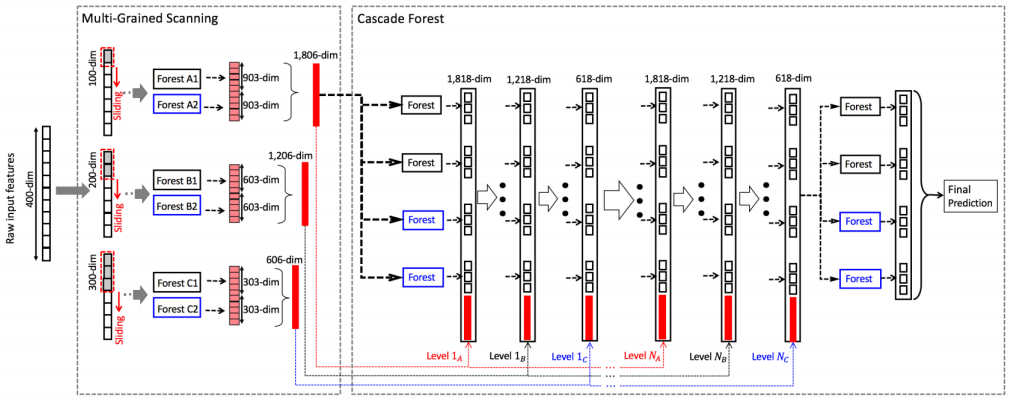


Figure 2.6: Illustration of the whole procedure of gcForest. Suppose there are three classes to predict, raw input features are 400 dimensional, and three sliding window sizes(100-dim, 200-dim, 300-dim) are used.

Similarly, 1206-dimensional and 606-dimensional transformed features, which are generated from a window sliding size of 200 and 300 respectively, will then be fed into the 2nd-grade and 3rd-grade of cascade forests structure respectively. This procedure is terminated when there is no gain of the improvement in model performance.

Given a testing sample, it will also be scanned by a Multi-Grained Scanning procedure to obtain its corresponding transformed features that then go through the cascade forest to the last level. The final prediction is an aggregation of the four 3-dimensional class vectors of the last layer, with the maximum aggregated value taken as the class.

Hyperparameters

As we mentioned earlier, the big advantage of gcForest over Deep neural networks is that it doesn't require too many fine hyperparameters tuning and will still be able to perform well especially on small datasets. The authors of Deep Forests also summarized the hyperparameters and default settings of gcForest and Deep neural networks, as shown in Table 1.1[9]. The next section

[9]This table is from Table 1 of Zhou's paper[17]

describes the hyperparameters of the gcForest that were primarily used in the thesis.

Number of Forests $n\_forests$ In gcForest, Multi-Grained Scanning employs one Random Forest and one completely Random Forest, and Cascade Forest employs four Random Forests and four completely Random Forests. It is recommended to set this hyperparameter based on the size of the dataset to reduce the runtime while getting the same performance.

Number of Trees n_trees The number of trees in the forests is a very useful parameter that can help us get a better representation of raw input features and a stronger predictive capability. But it should be noted that the number of trees is not a hyperparameter that a larger value always yields more reliable results than a smaller one.

Sliding window size $window\_size$ In gcForest, Multi-Grained Scanning employs feature windows with sizes of $[d/16]$, $[d/8]$, $[d/4]$, where d is the number raw input features. The author of gcForest pointed out that better performance can be obtained if this hyperparameter is task-specific tuned carefully.

Type of base classifiers In gcForest, the base classifiers are Random Forest and completely Random Forest. Note that the base classifiers here do not have to be a Random Forest or completely Random Forest, they can be replaced with other classifiers, such as, Gradient Boosting Decision Tree, Support Vector Machine, etc. But this is still an open question to be investigated.

Partitioning of the data set During the training phase, gcForest splits the training set into a growing set(80%) and an estimating set(20%). The ratio of growing set to estimating set can be adjusted for the specific data set.

Metrics In gcForest, Cascade Forest employs the accuracy for measuring the performance of each layer. It should be noted that some other metrics can be used to grow the layers depending on the task, e.g. F1 score, Matthews correlation coefficient(MCC), etc.

2.2 Vertex pairs generation

Networks are one of the most common and important data structures in the real world, such as social networks, communication networks, biological networks, and so on. Network embedding aims at seeking low-dimensional representation for the nodes or links while preserving the network structural information. These representations can be applied to a broad range of machine learning tasks, such as link prediction, classification, visualization, and so on. Network embedding techniques can be categorized into three groups: (1) random walk-based graph embedding, (2) matrix factorization-based graph

Table 2.2: Summary of default settings and hyperparameters. Bold font implies the relatively influential hyperparameters; "?" indicates default value is unknown or is sensitive to the type of task.

Deep neural networks (e.g., convolutional neural networks)	gcForest
Type of activation functions:	Type of forests:
Sigmoid, ReLU, tanh, linear, etc.	Completely-random tree forest, random forest, etc.
Architecture configurations:	Forest in multi-grained scanning:
No. Hidden layers: ?	**No. Forests**: {2}
No. Nodes in hidden layer: ?	**No. Trees in each forest**: {500}
No. Feature maps: ?	Tree growth: till pure leaf, or reach depth 100
Kernel size: ?	**Sliding window size**: $\{\lfloor d/16 \rfloor, \lfloor d/8 \rfloor, \lfloor d/4 \rfloor\}$
Optimization configurations:	Forest in cascade:
Learning rate: ?	**No. Forests**: {8}
Dropout: {0.25/0.50}	**No. Trees in each forest**: {500}
Momentum: ?	Tree growth: till pure leaf
L1/L2 weight regularization penalty: ?	
Weight initialization: Uniform, glorot_normal, glorot_uni, etc.	
Batch size: {32/64/128}	

embedding, and (3) deep learning-based graph embedding. For random walk-based graph embedding methods, a typical workflow includes the following steps: (1) update the graph, (2) generate random walks, (3) learn vertex representations(embedding), (4) train the downstream learning tasks[30]. In this thesis, we focus on the process of generating random walks in random walk-based graph embedding.

PRELIMINARIES

In this section, we give definitions required to introduce the graph structure.

Definition 2.2.1. *Graph. Given an undirected and unweighted graph G, $G = \{V, E\}$ has the set of vertices $V = \{v_1, v_2, ..., v_n\}$ and the set of edges $E = \{e_{ij} | 1 \leq i, j \leq n\} \subseteq V \times V$, if $e_{ij} \subseteq E$, then $e_{ji} \subseteq E$ and vice versa.*

2.2.1 Random Walk on graphs

Random walks arise in many models in mathematics and computer science. *Random walks on the graph* refers to a graph and a starting point, we randomly choose a neighbor of it and move to this neighbor; then we randomly choose a neighbor of this point, and move to it etc[31].

Random walks can be essentially viewed as k-th order Markov chain. Base on Markov property, the transition probability of a k-th order Markov chain only depends on previous k vertices visited by the walk. A random walk of length L starting from a vertex $v_{w(0)}$ consists of a sequence $W[w(0)] = \{w(0, ..., w(L-1))\}$, where $w(i) \in \{0, 1, ..., n\}$ represents the vertex index at the i-th position in the walk. Generally speaking, a k-th order random walk path is propagated by sampling a vertex $w(i)$ given the k previous vertices

$w(i-k,...,w(i-1))$ from the transition probability distribution :

$$p(w(i)|w(i-1),...,w(i-k)) \tag{2.3}$$

which is non-zero only if there is an edge between vertices $v_{w(i-1)}$ and $v_{w(i)}$.[30]

As illustrated in Figure 2.7, The random walk starts from the initial vertex $V_0 = 3$, randomly select the adjacent vertex $V_1 = 1$, and move to vertex V_1; Then randomly select the adjacent vertex $V_2 = 8$ and move to it, etc. The sequences of vertices $V_0, V_1, V_2, ..., V_k$ selected in this way is a simple random walk on graph G.

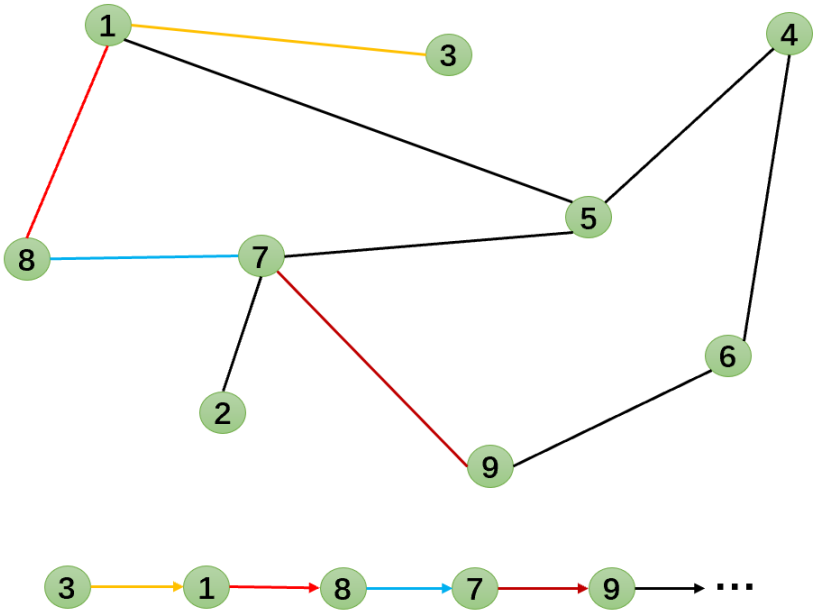


Figure 2.7: Illustration of a simple random walk on a graph. All the vertices in the graph are connected by black lines, and the color-coded lines indicate the order in which the vertices are visited. The line connected by arrows are the sequence of vertices generated by random walk on the above graph.

2.2.2 Node2Vec

Node2Vec[32] is a machine learning technique to find the vector representations of nodes on a graph. The Node2Vec learns low-dimensional representations for nodes in a graph by using biased random walks that can explore nodes in two ways: Breadth-first Sampling(BFS) and Depth-first Sampling(DFS). As illustrated in Figure 2.8, Node2Vec was inspired by Word2Vec[33] and the intuition of Node2Vec is that Random Walks on a graph can be treated like sentences in a corpus. Each node in a graph is viewed as an individual word,

and a random walk is viewed as a sentence; Then feeding these "sentences" into a skip-gram.

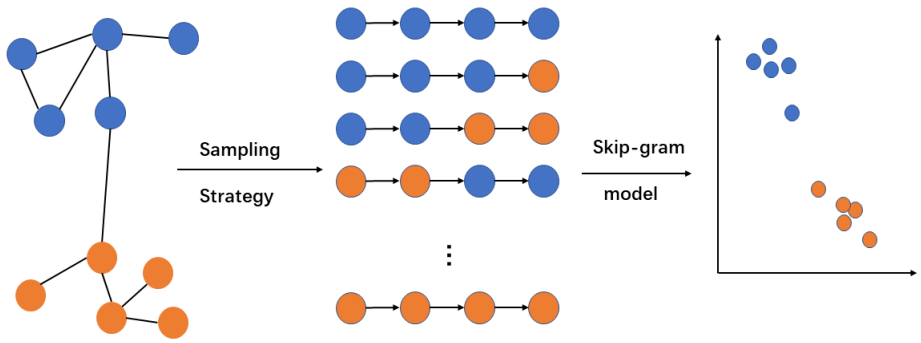


Figure 2.8: Illustration of the Node2Vec algorithm. Node2Vec employs adjustable sampling strategy to sample these directed acyclic subgraphs, and then these subgraphs will be fed into Skip-gram model.

Sampling strategy

In the Node2Vec algorithm, the most innovative part is the sampling strategy, and in this thesis, we only utilize this part, therefore we will in the following part only focus on introducing the sampling strategy.

The Node2Vec employs biased random walk sampling strategy that trades-off between Breadth-first sampling(BFS) and Depth-first sampling(DFS), between structural equivalence[34] and homophily[35]. Homophily hypothesis refers to nodes belong to a similar clusters should have similar embeddings(e.g., nodes N_0 and N_1 are in the same network cluster in Figure 2.9). The structural equivalence hypothesis means that nodes that have similar structural roles in the network should have similar embeddings(e.g., nodes N_0 and N_{12} are the hubs of their corresponding communities in Figure 2.9).

As illustrated in Figure 2.10, the trade-off in Node2Vec is achieved by 2 arguments, p and q, which controls how fast the walk explores and leaves the neighborhood of starting node u.

But how do p and q guide the walk? let us take Figure 2.10 as an example. Suppose the random walk is at node v and came from node t along the edge (t, v). The walk now has to decide on the next step thus it evaluates the transition probabilities π_{vx} on edges (v, x). The unnormalized transition

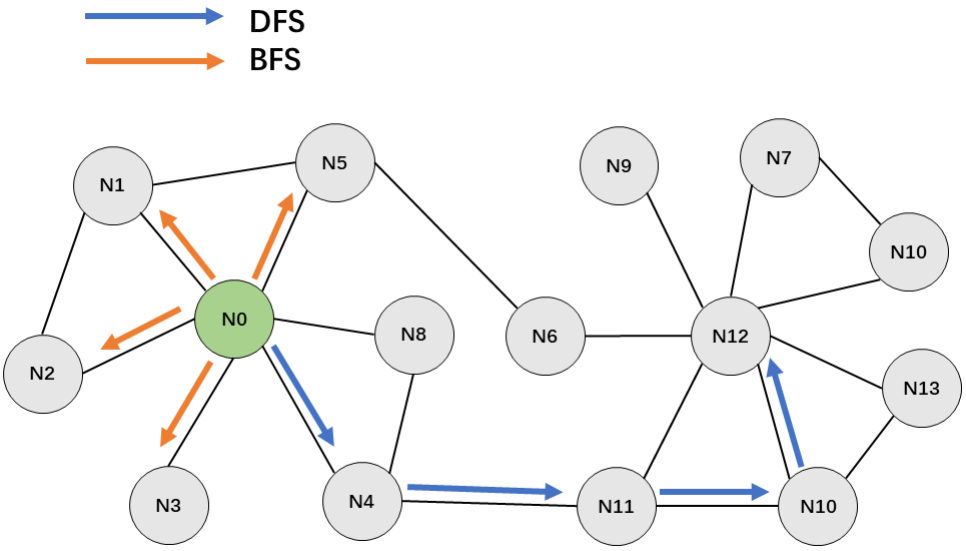


Figure 2.9: Illustration of the BFS and DFS search strategies from node N_0, the number of nodes equals 4.

probability is defined as $\pi_{vx} = \alpha_{pq}(t,x) \cdot \omega_{vx}$, where

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & if \quad d_{tx} = 0 \\ 1 & if \quad d_{tx} = 1 \\ \frac{1}{q} & if \quad d_{tx} = 2 \end{cases}$$

ω_{vx} is the edge weight between node v and x.

The Node2Vec's sampling strategy also accepts the other 2 parameters, $Number\_of\_Walks$ and $Walk\_length$, respectively. $Number\_of\_Walks$ refers to the number of random walks to be generated from each node in the graph and $Walk\_length$ means the number of nodes in each random walk.

2.3 Graph inference

Graph inference refers to the problem in which the information on the identity and the state of a system's elements is used to infer interactions or functional relationships among these elements and to construct the interaction graph underlying the system[36]. This problem has recently become an important topic in computational biology, where the reconstruction of various biological networks, such as gene or molecular networks from genomic data, is

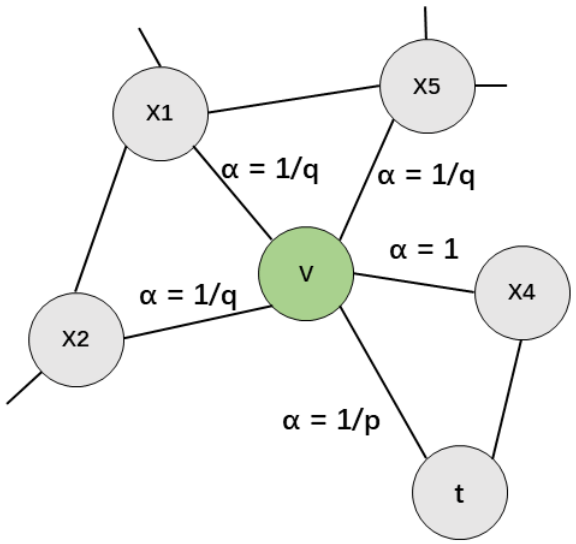


Figure 2.10: Illustration of the random walk procedure in Node2Vec. The walk is transitioned from t to v and now is evaluating its next step out of node v. Edge labels indicate search biases α.

a core prerequisite to the recent field of systems biology that aims at investigating the structures and properties of such networks[37]. We can simplify the elements of the system to the vertices of the graph, and the pairwise relationships to the edges of the graph. In the biological field, the elements in the cellular system can be genes, proteins, sites of action, etc. The edges imply interrelationships between elements, such as protein-protein interactions and gene-gene interactions, etc. As an example, with the development of high-throughput technologies, the reconstruction of protein interaction network[38], gene regulatory network[39] or metabolic network[40] from large-scale genomic data becomes an increasingly important topic.

Various approaches have been proposed to solve the network inference problem. The next section focuses on the three methods of graph inference that we used in the thesis.

2.3.1 Structural learning in graphical model

Bayesian networks are graphical models where nodes represent random variables and arrows represent probabilistic dependencies between them [41, 42]. The purpose of Bayesian methods is to find a directed, acyclic graph(hence the name of structure learning algorithms) describing the causal dependency relationships among components of a system and a set of local joint probability distributions that statistically convey this relationships[39].

There are two categories in bayesian network structure learning algorithms, constraint-based algorithms, and score-based algorithms. In the next section, we focus on introducing the score-based algorithms that were mainly used in the thesis.

Score-based algorithms assign a score to each candidate Bayesian network and try to maximize it with some heuristic search algorithm. Greedy search algorithms (such as hill-climbing or tabu search) are a common choice, but almost any kind of search procedure can be used. Bnlearn R[19] package provides us a free implementation of these structure learning algorithms.

2.3.2 STRING database

In molecular biology, STRING(Search Tool for the Retrieval of Interacting Genes/Proteins) is a biological database of known and predicted protein–protein interactions[21, 43]. It aims to collect, score, and integrate all publicly available sources of protein-protein interaction information that could be used to construct a comprehensive and objective global network.

As shown in Figure 2.11[10] , all the protein associations could be derived from seven independent domains. In addition to data from experimentally derived protein–protein interactions, STRING also store computationally predicted interactions from (i) text mining of scientific texts, (ii) interactions computed from genomic features, and (iii) interactions transferred from model organisms based on orthology. Therefore, in the thesis, we call the graph structure accessed from the STRING database ground-truth graph.

2.3.3 SILGGM

SILGGM(Statistical Inference of Large-scale Gaussian Graphical Model) is an extensive and efficient R package that includes four main approaches developed in estimating conditional dependence of genes using high-dimensional Gaussian graphical model(GGM) due to the intrinsically sparse structure of a gene network[20]: the bivariate nodewise scaled Lasso(B_NW_SL)[44], the desparsified nodewise scaled Lasso(D-S_NW_SL)[45], the de-sparsified graphical Lasso(D-S_GL)[46] and the GGM estimation with false discovery rate(FDR) control using scaled Lasso or Lasso(GFC_SL or GFC_L)[47].

The setup of SILGGM is very simple, which only takes an n(the number of genes) by p(the number of samples) gene expression data matrix. As shown in figure 2.12[11], the input data matrix is further centralized or standardized

[10]This figure is from Figure 1 of Szklarczyk's paper[21]
[11]This figure is from Figure 1 of Zhang's paper[20]

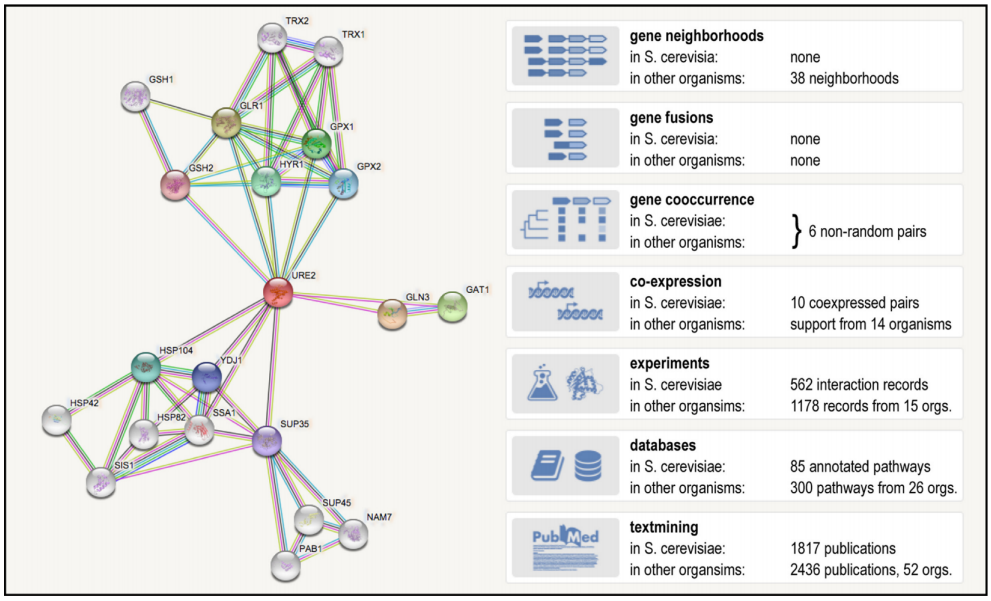


Figure 2.11: The left panel shows the typical protein-protein interaction network. The yeast prion-like protein URE2 is input. The color saturation of the edges represents the confidence score of each association. The right panel shows various evidence types in STRING.

and then it will be performed statistical inference by one of the four methods above. The final individual and global inference can be presented in a table format with the cytoscape_format argument.

Figure 2.13[12] gives a table with the 20 most significant gene pairs based on the rank of test statistics with hub gene CLK1. The first two columns show the names of the gene pair. "test_statistics" indicates the test statistics of gene pair. The last column "global_decision_0.05" shows the decision for conditional dependence between each gene pair under global inference with FDR control at the 0.05 level.

[12]This figure is from Figure 2 of Zhang's paper[20]

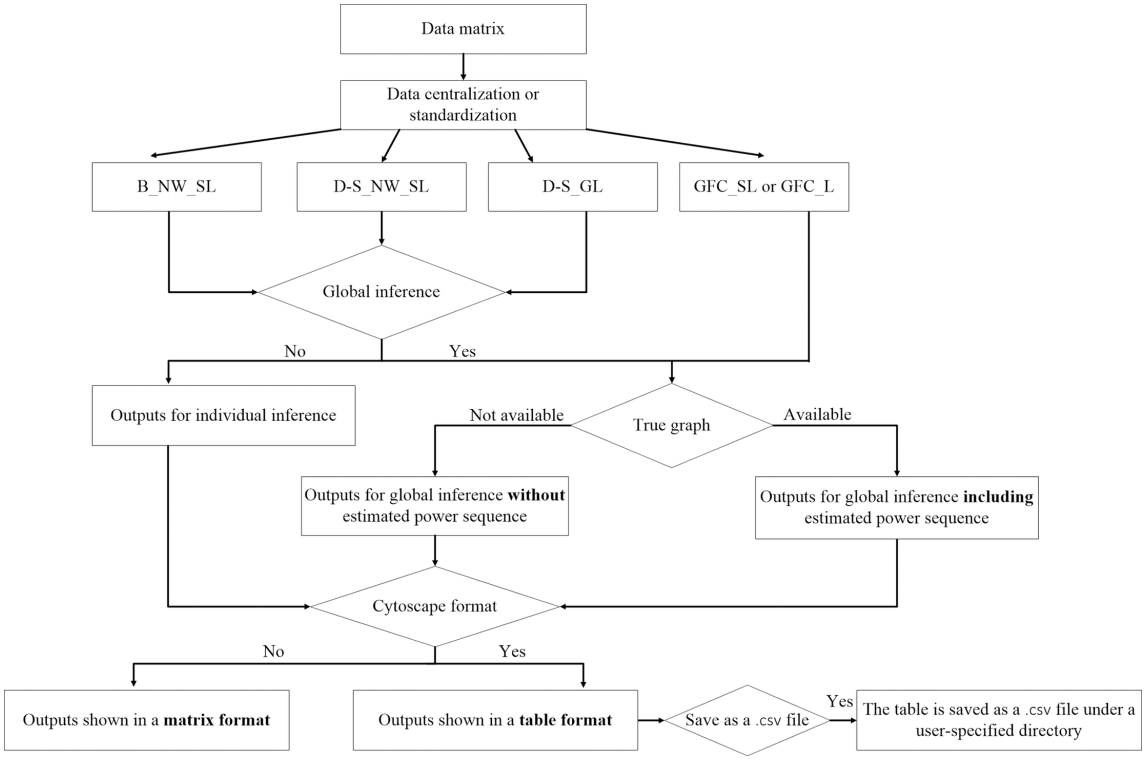


Figure 2.12: The workflow of SILGGM package.

(A)

gene1	gene2	test_statistic	global_decision_0.05
CLK1	C1ORF63	7.068319578	1
CLK1	DNAJB1	6.370053216	1
CLK1	RGS1	5.519126715	1
CLK1	RNF146	5.291264048	1
CLK1	POLQ	-5.191657692	1
CLK1	PIAS1	-5.131982503	1
CLK1	DNAJB9	4.986321414	1
CLK1	EGR1	-4.808012448	1
CLK1	ARMCX3	4.588409301	1
CLK1	DNAJB4	4.583513721	1
CLK1	HMMR	-4.55370168	1
CLK1	MIR21	-4.552792912	1
CLK1	TRA2A	4.481474871	1
CLK1	PPP4R2	4.441497616	1
CLK1	FAM76B	4.393816269	1
CLK1	C1ORF199	4.390922346	1
CLK1	PDE4B	-4.373685294	1
CLK1	SLC16A6	-4.318410531	1
CLK1	PIK3C2A	-4.136358875	1
CLK1	IER3	-4.11946634	1

(B)

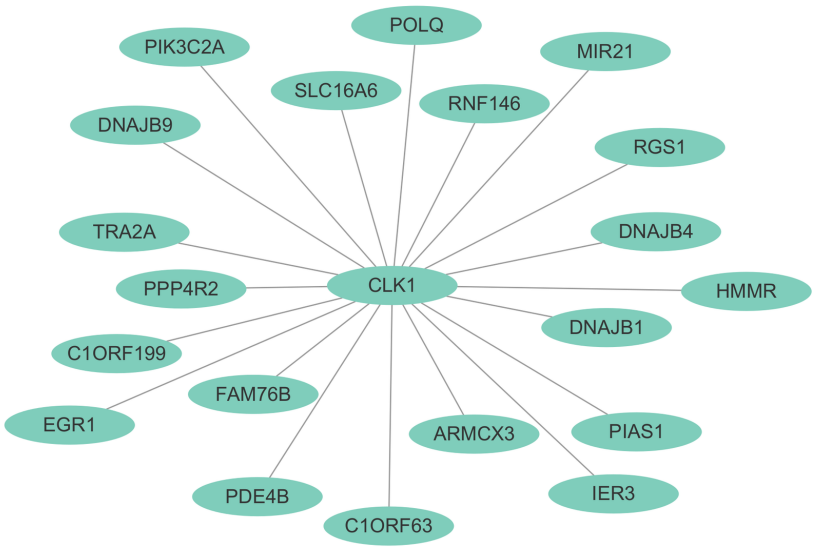


Figure 2.13: An example of table format output and the corresponding graph structure. (A) A table generated by using the method GFC_SL. (B) The corresponding graph structure.

Chapter 3

Methodology

Machine learning on the graph has recently turned into a trend. On the one hand, it is because graph structures are ubiquitous in our lives, such as protein-protein interaction networks, molecular graph structures, and social networks. On the other hand, it is because of the recent advancements in representation learning on graphs, including matrix factorization-based methods, random-walk based algorithms, and graph convolutional networks, where traditionally machine learning approaches relied on user-defined heuristics to extract features encoding structural information about a graph(e.g., degree statistics or kernel functions)[48].

In recent years, substantial research efforts have been devoted to allowing deep learning methods to exploit graph structures, resulting in many advances. We were inspired by this and then proposed a variant of gcForest, Graph-based gcForest, which can incorporate the information about the structure of the graph to scan the input data instead of scanning the input data linearly as in gcForest. The code of graph-based gcForest is available[1].

3.1 Graph-based gcForest

GcForest consists of two parts. The first part is the Multi-Grained Scanning that linearly scans the original features and learns the representation of the original features through different Random Forests. Figure 2.5 shows the illustration of a Multi-Grained Scanning. The second part is the Cascade Forest that utilizes the layer-by-layer structure to process the input representations, thus gives more accurate predictions according to the ensemble of Random Forests. Figure 2.3 shows the illustration of a Cascade Forest.

[1]https://github.com/jzhanghzau/Graph_based_Deep-Forest

Graph-based gcForest also consists of two parts. The first part is the graph-based Multi-Grained Scanning, which is the modification of the Multi-Grained Scanning. We were inspired by Node2Vec, as described in Section 1.2.2, and proposed a graph-based Multi-Grained Scanning strategy to obtain the transformed features that incorporate the information of the graph corresponding to the original feature. Our contribution is focused on this part. The second part is the Cascade Forest. In Graph-based Deep Forest, we adopt the same Cascade Forest structure as in gcForest. We will ignore the description of the Cascade Forest structure, as details can be found in Section 1.1.2. In the next section, we will first introduce the graph-based Multi-Grained Scanning, followed by the overall architecture and remarks on hyperparameters.

3.1.1 Graph-based Multi-Grained Scanning

Multi-Grained Scanning inspired by Deep Neural Networks has been proposed to enhance Cascade Forest. Although it has been proved that Multi-Grained Scanning is able to recognize the local feature. However, when there exist graph structures between features, linear scanning does not capture and exploit their local structures well. Therefore we propose graph-based Multi-Grained Scanning, which exploits the information of graph structure for non-linear scanning.

The pseudocode for graph-based scanning is given in Algorithm 1. We simulate r random walks of fixed length starting from every node, control of the number of generated walks is then accomplished by s. At each step of the walk, sampling is done based on the transition probabilities π_{vx}. The transition probabilities π_{vx} for the 2^{nd} order Markov chain can be precomputed and hence, sampling of nodes while simulating the random walk can be done efficiently in $O(1)$ time using alias sampling[18]. All feature vectors extracted based on walks will then be used to generate transformed features like Multi-Grained Scanning described in Section 1.1.2: the subsets of the raw input data will be used to train a completely Random Forest and Random Forest, and then the class vectors are generated and concatenated as transformed features.

As shown in Figure 3.1, walks are used to scan the raw features. For sequence data, suppose there are 1000 raw input features, the corresponding graph contains 1000 nodes, the walk length used is 200 nodes and the number of generated walks is set to 300. Assume that there are three classes, 300 three-dimensional class vectors are then generated by each forest, resulting in 1800-dimensional transformed feature corresponding to the original 1000-dimensional raw input data.

In the graph-based Multi-Grained Scanning, both the walk and the sliding window in the Multi-Grained Scanning have the functionality used to extract

Algorithm 1 Graph-based Scanning

Input: Graph $G = (V, E, \pi)$, Walks per node r, Walk Length l, Scale s, Return p, In-out q, Raw Input D

function GRAPH-BASED SCANNING
  π = PreprocessModifiedWeights(G, p, q)
  Graph $G^{'} = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter \leftarrow 1$ to l **do**
    for all node $u \in V$ **do**
      $walk \leftarrow$ Node2VecWalk$(G^{'}, u, l)$
      Append $walk$ to $walks$
    end for
  end for
  $t \leftarrow$ TransformFeatures$(D, walks[: s])$
  return t
end function

Algorithm 2 Node2VecWalk

Input: Graph $G^{'} = (V, E, \pi)$, Start node u, Length l

function NODE2VECWALK
  Initialize $walk$ to $[u]$
  for $walk\_iter \leftarrow 1$ to l **do**
    $curr \leftarrow walk[-1]$
    $V_{curr} \leftarrow$ GetNeighbors$(curr, G^{'})$
    $s \leftarrow$ AliasSample(V_{curr}, π)
    Append s to $walk$
  end for
  return $walk$
end function

the subsets of the raw input data. The difference is that 1), the sliding window only scans the raw input data linearly, whereas walk can scan the raw input features non-linearly according to the graph structure; 2), in the sliding window, once the sliding size is set, the number of generated subsets of the raw input data is also determined, e.g. if there are 1000 raw input features and the sliding window size is 200 features, so the number of generated subsets of the raw input data is 801. However, in the walk, the walk length is not related to the number of generated walks, e.g. if there are 1000 raw input features, the walk length is 200 nodes. We can set the number of generated subsets of the raw input data, i.e. the number of generated walks, to any value.

Figure 3.1 shows only one length of the walk. By using multiple lengths of

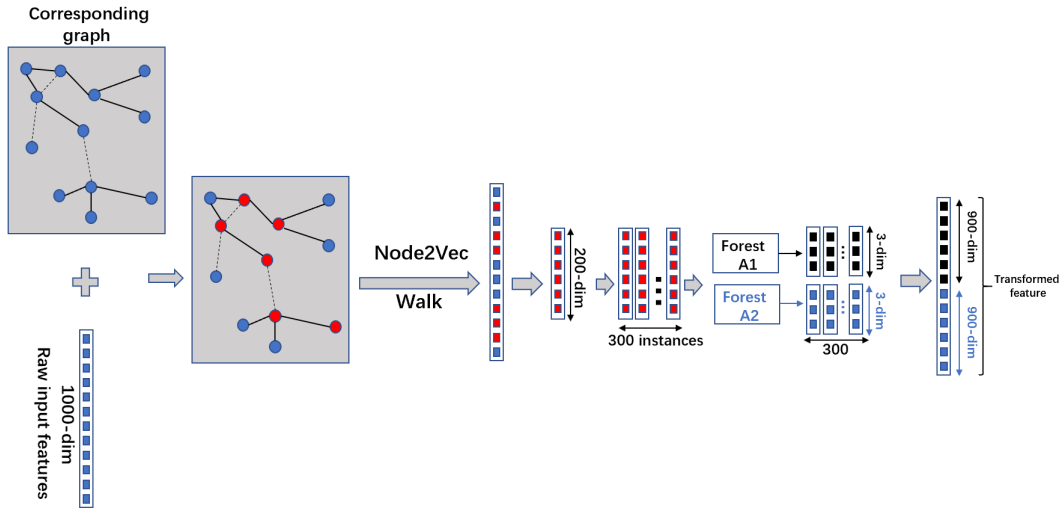


Figure 3.1: Illustration of graph-based Multi-Grained Scanning. Suppose there are three classes, raw input features are 1000-dim, the corresponding graph contains 1000 nodes, the walk length used is 200 nodes and the number of generated walks is set to 300.

walks, different grained feature vectors will be produced, as shown in Figure 3.2.

3.1.2 Overall structure of graph-based gcForest

As illustrated in Figure 3.2, graph-based gcForest merges the graph-based Multi-Grained Scanning and Cascade Forest structure. Suppose there are 1000 raw input features, the corresponding graph contains 1000 nodes, the walk length used is 400 nodes and the number of generated walks is set to 300, for m training samples, a data set of $300 \times m$ 200-dimensional training examples will thus be generated. These data will then pass through a Random Forest and completely Random Forest. Assume that there are three classes to be predicted, a 1800-dimensional transformed feature corresponding to the original 1000-dimensional raw input data will be obtained as described in the previous section. The transformed training data will then be used to train the 1st-grade of Cascade Forest.

Walks with walk length of 600 and 800 will go through the same process, leading to 2400-dimensional and 3000-dimensional transformed features, respectively. The transformed feature vectors, augmented with the class vector generated by the previous grade, will then be used to train the 2nd-grade and 3rd-grade of cascade forests, respectively. This procedure will be repeated until convergence of validation performance[17]. As shown in Figure 2.6, each

level of Cascade Forest contains multiple grades that correspond to grains of graph-based scanning.

Given a test instance, it is transformed into the corresponding transformed features by using graph-based scanning. Then, the transformed features are fed into the Cascade Structure to obtain its final decision by aggregating the four 3-dimensional class vectors at the last level and taking its class with the maximum aggregated value.

The pseudocode for graph-based gcForest is given in Algorithm 3.

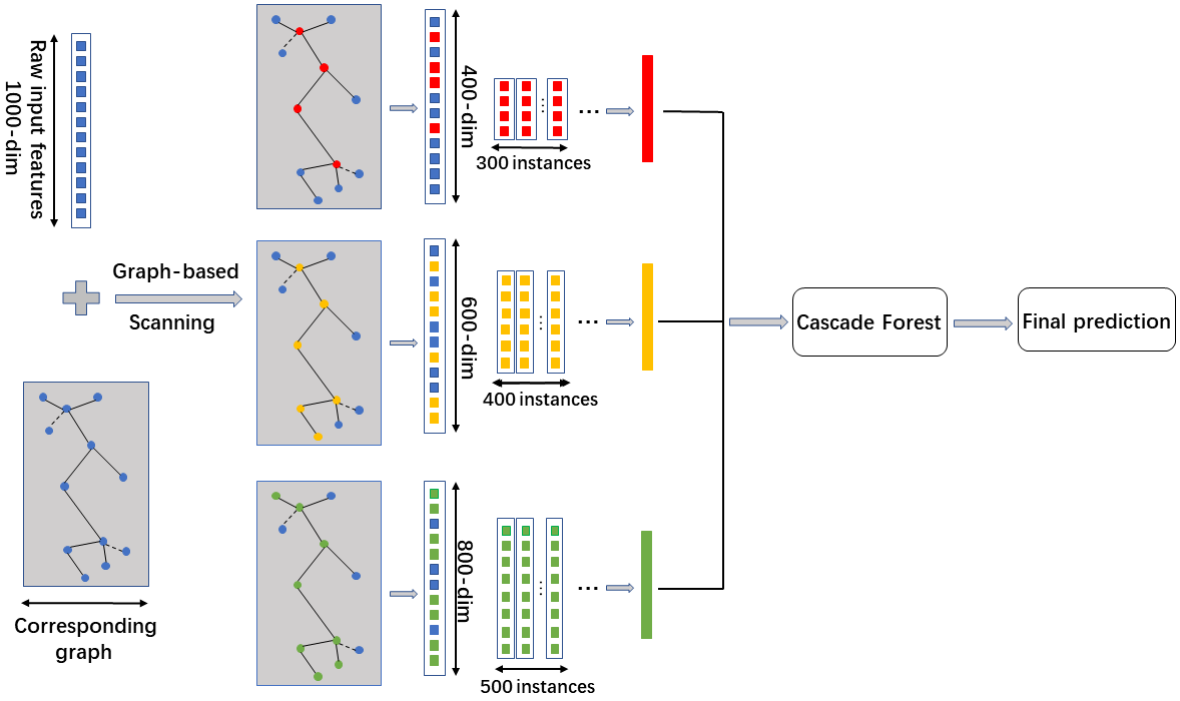


Figure 3.2: Illustration of overall procedure of graph-based gcForest. Suppose there are three classes to predict, raw input features are 1000 dimensional, corresponding graph contains 1000 nodes, three walk lengths(400 nodes, 600 nodes, 800 nodes) are used and the number of generated walks is set to 300, 400, 500, respectively.

3.1.3 Hyperparameters

Compared to the gcforest, the graph-based gcforest has several more hyperparameters which allow us to be more flexible in scanning the graph structure. The next section describes the hyperparameters of the graph-based gcForest that were primarily used in the thesis.

Algorithm 3 Graph-based gcForest

Input: Graph $G = (V, E, \pi)$, Walks per node r, Walk Length set L, Scale set S, Return p, In-out q, Raw Input D

1: **function** GRAPH-BASED GCFOREST
2: Initialize $Transformed \quad dataset \quad T$ to Empty
3: **for** all $l^{'} \in L$, all $s^{'} \in S$ **do**
4: $t \leftarrow$ Graph-basedScanning$(G, r, l^{'}, s^{'}, p, q, D)$
5: Append t to T
6: **end for**
7: $res \leftarrow$ Cascade Forest(T)
8: **return** res
9: **end function**

Number of Forests $n\_forests$ In graph-based gcForest, graph-based Multi-Grained Scanning employs one Random Forest and one completely Random Forest, and Cascade Forest employs four Random Forests and four completely Random Forests. It is recommended to set this hyperparameter based on the size of the dataset to reduce the runtime while getting the same performance.

Number of Trees n_trees The number of trees in the forests is a very useful parameter that can help us get a better representation of raw input features and a stronger predictive capability. But it should be noted that the number of trees is not a hyperparameter that a larger value always yields more reliable results than a smaller one.

Walk length $walk\_length$ In graph-based gcForest, graph-based Multi-Grained Scanning employs different lengths of walks that have the same functionality as the sliding windows in gcForest. Better performance can be obtained if this hyperparameter is task-specific tuned carefully.

Number of walks n_walks In gcForest, the number of generated subsets of the raw input data is automatically determined once the size of the sliding window has been set. In graph-based gcForest, the number of generated subsets of the raw input data, i.e., the number walks, can be freely set. a faster and equally good model can be obtained if this hyperparameter is task-specific tuned carefully.

Return parameter p Parameter p controls the probability of revisiting a node in the walk. As shown in Figure 2.10, if p is low, it would lead the walk to backtrack a step. In contrast, if p is high, we are less likely to revisit the already visited node[18].

In-out parameter q Parameter q is used to control the inward exploration(BFS strategy) and outward exploration(DFS strategy). Going back to

Figure 2.10, if q > 1, the random walk biased towards nodes which are close to node t. In contrast, if p < 1, the random walk prefers visiting nodes that are far away from the node t.

Graph *graph* The graph-based gcForest can accept different graph structures.

Type of base classifiers In graph-based gcForest, the base classifiers are Random Forest and completely Random Forest. Note that the base classifiers here do not have to be Random Forest or completely Random Forest, they can be replaced with other classifiers, such as Gradient Boosting Decision Tree, Support Vector Machine, etc. But this is still an open question to be investigated.

Partitioning of the data set During the training phase, graph-based gcForest splits the training set into a growing set(80%) and a estimating set(20%). The ratio of growing set to estimating set can be adjusted for the specific data set.

Metrics In graph-based gcForest, Cascade Forest employs the accuracy for measuring the performance of each layer. It should be noted that some other metrics can be used to grow the layers depending on the task, e.g. F1 score, Matthews correlation coefficient(MCC), etc.

Chapter 4

YEAST dataset

In this chapter, our proposed graph-based gcForest(Section 3.1), gcForest(Section 2.1.2), Cascade Forest(Section 2.1.2) and Random Forest(Section 2.1.1) will be applied to YEAST dataset to perform classification tasks. The aim is to confirm that graph-based gcForest is competitive compared to other models.

We use accuracy(Accu) to compare the performance of each model. The accuracy score is defined as follows:

$$Accu = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

where, TP stands for true positive, that is, a correctly identified instance; FP represents false positive, that is, an incorrectly identified instance; TN stands for true negative, that is, a correctly rejected instance; FN represents the false negative, that is, an incorrectly rejected instance.

In all experiments, the parameter settings of Random Forest and Cascade Forest are the same, which are listed in Table A.1 and A.2, respectively. The general configurations of gcForest and graph-based gcForest are shown in Table, it is noted that for different tasks some parameters will vary, referring to the settings in each task. For all experiments we use 3-fold cross-validation to generate class vector and use 80% of the training data as a growing set[1] and 20% as a estimating set[2].

For each individual task, the configurations of gcForest and graph-based gcForest are listed in each separate section. Preprocessing of the dataset, experimental procedures, and network inference are also presented below.

The list of materials used during the thesis is shown in Appendix A. For more detailed information see Table A.4.

[1]Refer to Section 2.1.2
[2]Refer to Section 2.1.2

4.1 Data description and preprocessing

OpenML is a collaboration platform through which scientists can automatically share, organize, and discuss machine learning experiments, data, and algorithms[49]. We use the yeast(4)[50] dataset from OpenML. The YEAST dataset contains micro-array expression data as well as phylogenetic profiles of yeast, including 2417 genes(samples) and 103 features. In total, 14 different labels can be assigned to a gene, but only 13 labels were used due to label sparsity. The first label is used to do the following study. Thus, in total the dataset consists of 2417 genes, of which 1655 are FASLE samples and 762 are TRUE samples.

4.2 Network Inference

Since the features of the YEAST dataset are anonymous, we thus use Bnlearn(Section 2.3.1) and SILGGM(Section 2.3.3) to infer its graph structure.

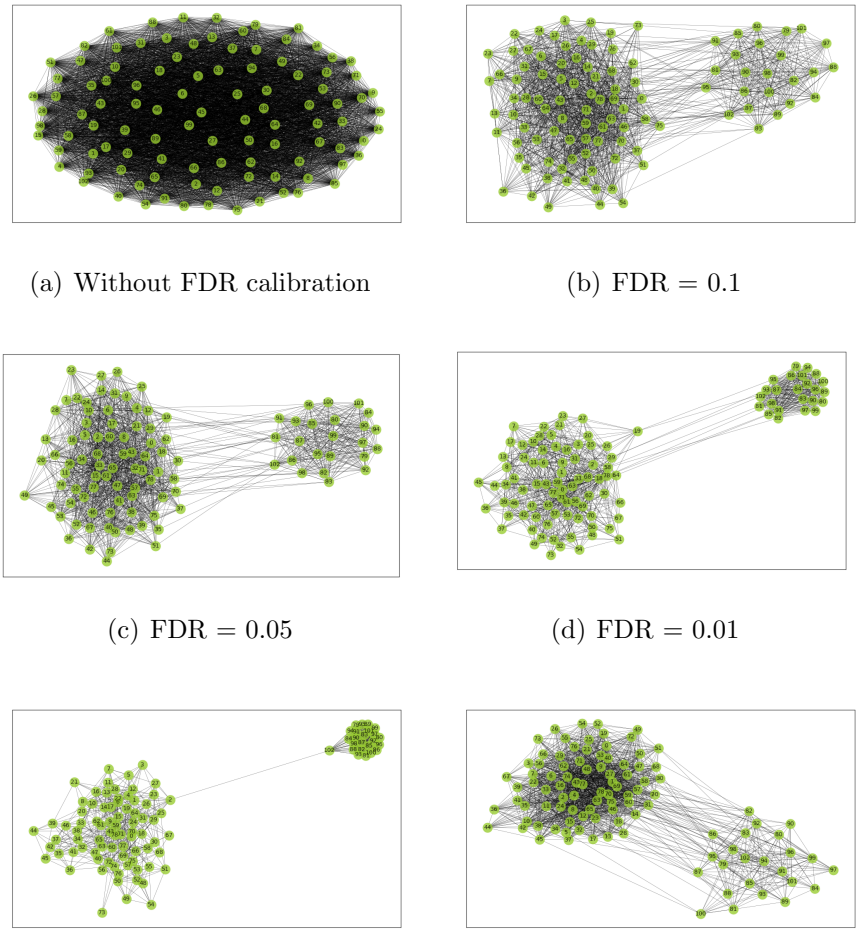
For Bnlearn, we use Hill-climbing(HC) algorithm that is a score-based structure learning algorithm to infer graph structure. The parameter settings are listed in Table A.5

For SILGGM, we use B_NW_SL method with default settings to infer graph structure. Outputs are shown with the different types of inference. For individual inference of gene i and gene j, SILGGM provides the associated p-value. We filtered out connections by different FDR-value to obtain different graph structures. The parameter settings are listed in Table A.6.

The basic information and visualizations of different graph structures are presented in Table 4.1 and Figure 4.1, respectively.

Table 4.1: The basic information about the six graphs. Without Calibration refers to the original graph learned from SILGGM. Different FDR values mean the graphs obtained from SILGGM with different FDR correction. Bnlearn indicates the graph learned by Bnlearn.

Graphs	Nodes	Edges
Without calibration	103	5253
FDR=0.1	103	1617
FDR=0.01	103	1018
FDR=0.05	103	1395
FDR=0.001	103	767
Bnlearn	103	1507



(a) Without FDR calibration

(b) FDR = 0.1

(c) FDR = 0.05

(d) FDR = 0.01

(e) FDR = 0.001

(f) Bnlearn

Figure 4.1: Visualizations of different graph structures of the YEAST dataset. (a), (b), (c), (d), (e) are learned through SILGGM and then calibrated by different FDR threshold. (f) is learned through Bnlearn.

4.3 Classification performance

We compare the graph-based gcForest with other benchmark algorithms on the YEAST dataset. It should be noted that for graph-based gcForest we only tune the hyperparameters p and q.

4.3.1 Experimental setup

We designed the following process to avoid data leakage or overfitting issues. As shown in Figure 4.2, the YEAST dataset was split into a training set(80%) as well as a testing set(20%) by using 5-fold cross-validation. Then the best p and q hyperparameters for graph-based gcForest were learned using 4-fold cross-validation(thus the validation set also takes up 20% of the total data) on the training set with a grid search over p, $q \in \{0.5, 1, 2, 4\}$. Finally, different benchmark algorithms were trained on the training set and compared on the testing set. The above process was repeated until each test fold generated from the outer loop is treated as testing set to obtain the average performance of each model.

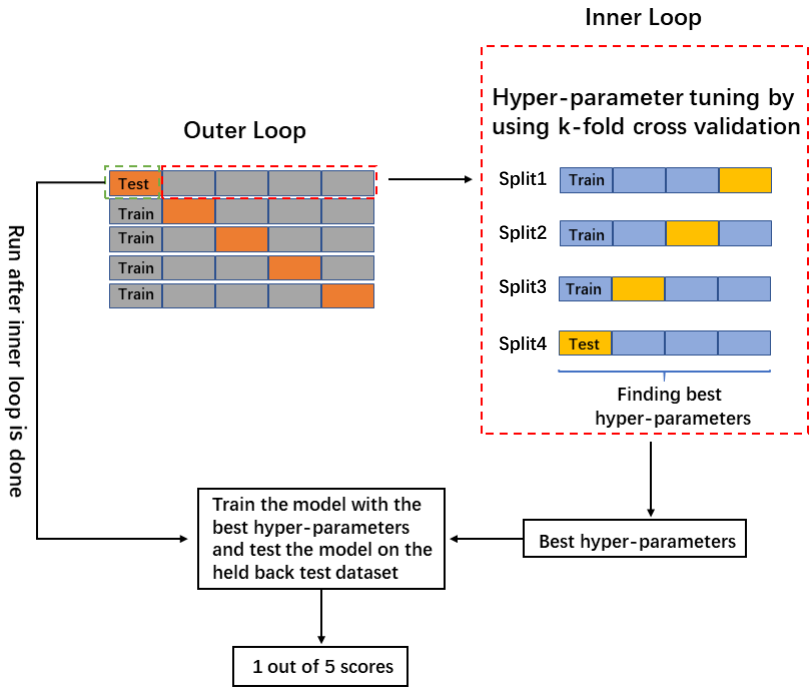


Figure 4.2: Illustration of the process of the experiment on YEAST dataset. For outer loop, 5-fold cross validation is used to split the training set and the testing set. For inner loop, 4-fold cross validation is used to obtain the best combination of p and q for graph-based gcForest.

In this task, some of the parameter settings of graph-based gcForest and gcForest have changed. We set the *window_size*(sliding window size) in Multi-Grained Scanning and the *walk_length*(walk length) in graph-based Multi-Grained Scanning to 6, 13 and 26, which are $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$ of the 103 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 98, 91, and 78, corresponding to each walk length. The parameter settings for gcForest and graph-based gcForest are listed in Table A.7 and Table A.8, respectively. The graph we used was from SILGGM and was then filtered out some connections using an FDR value of 0.05. This graph has 103 nodes and 1395 edges.

4.3.2 Experimental results

The comparison results for accuracy scores are summarized graphically in Figure 4.3. From the results, it is evident that graph-based gcForest not very significantly outperforms the other benchmark algorithms on the YEAST dataset. However graph-based gcForest are able to achieve boosts with more hyperparameter tuning, as shown in the subsequent experiments.

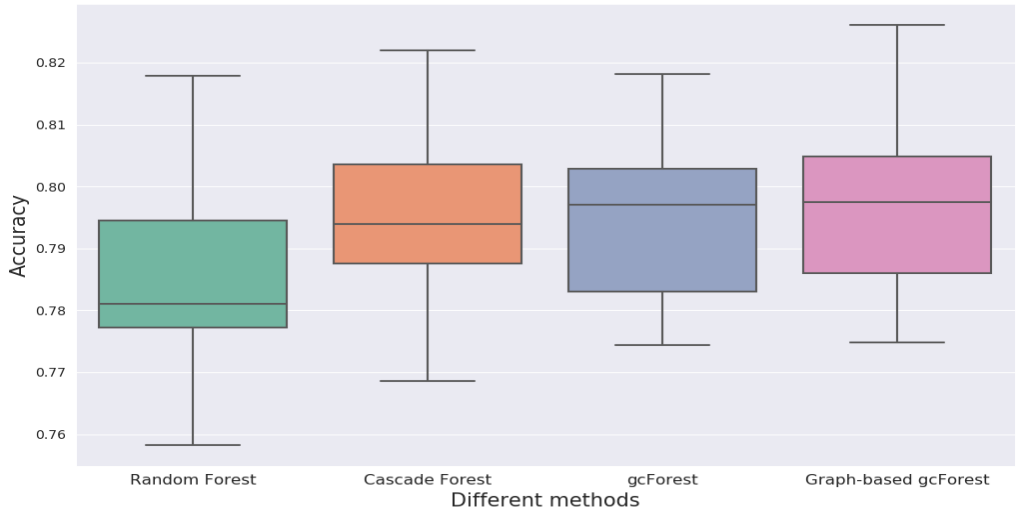


Figure 4.3: Performance evaluation of different methods on YEAST dataset.

4.4 Influence of different graph structures

For many biological datasets, we do not have access to accurate information about the graph structure. Therefore we examine how the different choices of graphs affect the performance of graph-based gcForest on the YEAST dataset.

4.4.1 Experimental setup

The YEAST dataset was first split into training data(80%) and testing data(20%), taking label imbalances into account. We trained graph-based gcForests by using different graphs on the training data and tested them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, only the graph structures of the graph-based gcForests vary(Section 4.2). In addition, for all graph-based gcForest we set the $walk\_length$(walk length) to 6, 13 and 26, which are $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$ of the 103 features, respectively. n_walks(The number of walks) was set to 98, 91, and 78, corresponding to each walk length. We set $p = 1$ and $q = 4$. The parameter settings for graph-based gcForests with different graphs are listed in Table A.9.

4.4.2 Experimental results

For graph-based gcForest, we have six graph options, of which five are from the SILGGM and the other is learned by Bnlearn(Section 4.2). The basic information about the graphs obtained through the SILGGM and Bnlearn refers to Table 4.1.

We then visualize the different graph structures with nodes assigned numbers based on the index of the features. As shown in Figure 4.1, subplot (a) is very dense, while the remaining five subplots all have two distinct clusters and are relatively sparse.

As shown in Figure 4.4, the performance of graph-based gcForest varies with the sparseness of the graph structure. Generally speaking, graph-based gcForest prefers a relatively sparse graph structure where possibly has less noise. In short, graph structure has an impact on the performance of the graph-based gcForest for the classification tasks.

4.5 Influence of the number of walks

The graph-based gcForest involves some hyperparameters that are not included in the gcForest, the most important of which is the number of walks. We examine how the number of walks affects performance.

4.5.1 Experimental setup

The YEAST dataset was first split into training data(80%) and testing data(20%), taking label imbalances into account. We trained graph-based gcForests by using a different number of walks on the training data and tested

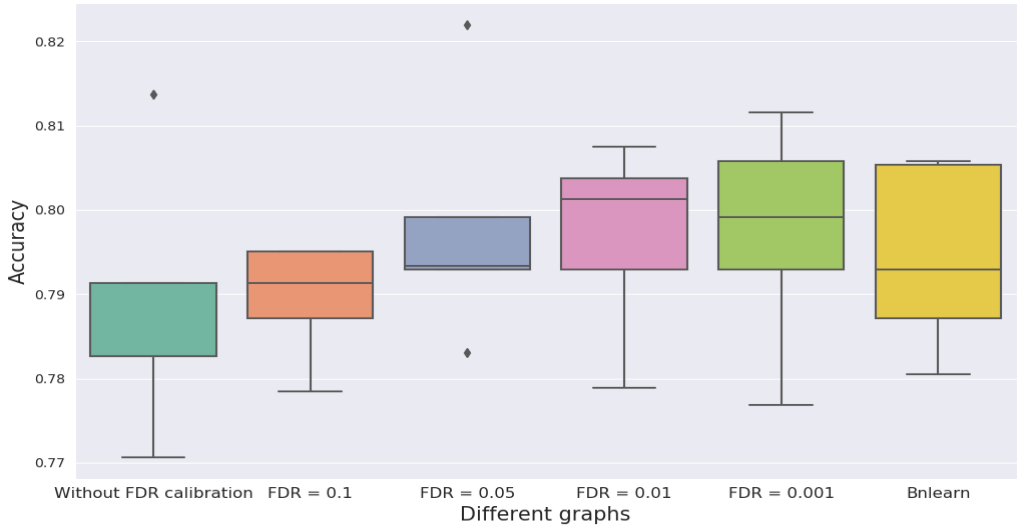


Figure 4.4: The performance of graph-based gcForest with different graphs on YEAST dataset. Without Calibration refers to the original graph learned from SILGGM. Different FDR values mean the graphs obtained from SILGGM with different FDR correction. Bnlearn indicates the graph learned by Bnlearn.

them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, only the number of walks of the graph-based gcForests vary. For all graph-based gcForest we set the *walk_length*(walk length) to 6, 13, and 26, which are $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$ of the 103 features, respectively. In gcForest, if we set the *window_size*(sliding window size) to 6, 13, and 26, the corresponding number of generated subsets of the original input is 98, 91, and 78. To maintain consistency with gcForest, we define a tuple of 98, 91, 78 as a w. For graph-based gcForests, we set *n_walks*(the number of walks) to different fractions of w. For instance, $0.1w$ indicates that the number of walks is 10, 9, 8. We set $p = 1$ and $q = 4$. The parameter settings for graph-based gcForests with different graphs are listed in Table A.10.

4.5.2 Experimental results

As shown in Figure 4.5, the performance of graph-based gcForest varies with the number of walks. In general, as the number of walks increases, the performance of graph-based gcForest also increases, and then stabilizes. We achieve large improvements at $2w$ at the cost of increasing time. Generally speaking, graph-based gcForest prefers a large number of walks, in particular for small-scale datasets. In short, the number of walks has an impact on the

performance of the graph-based gcForest for the classification tasks.

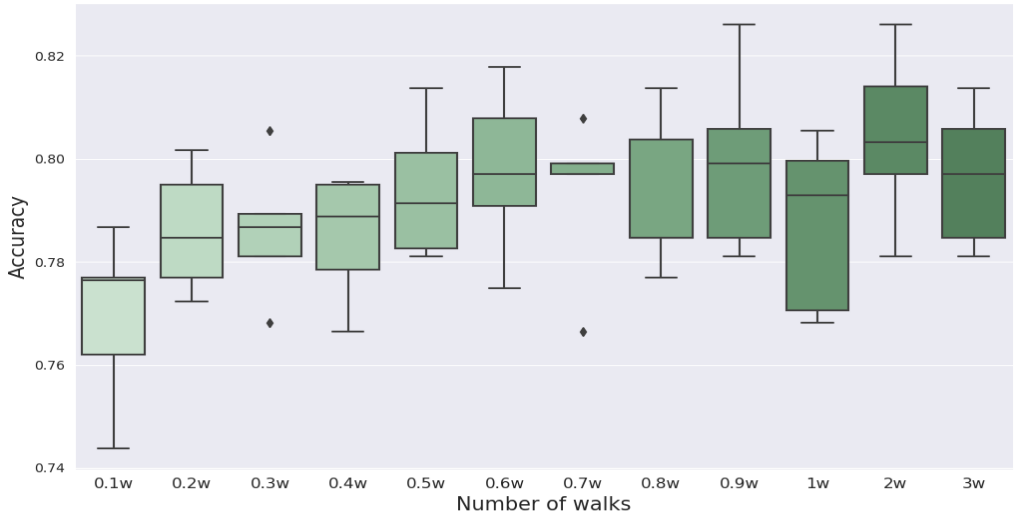


Figure 4.5: The performance of graph-based gcForest with different number of walks on YEAST dataset. The x axis denotes the fraction of w(Section 4.5.1), whereas the y axis denotes the accuracy score. The graph-based gcForest with $2w$ performs best.

4.6 Visualization of different scanning methods

In the above sections, we observed that graph-based nonlinear scanning can provide more benefits to Cascade Forest than sliding window-based linear scanning. We now aim to visualize the differences between the two different scanning methods.

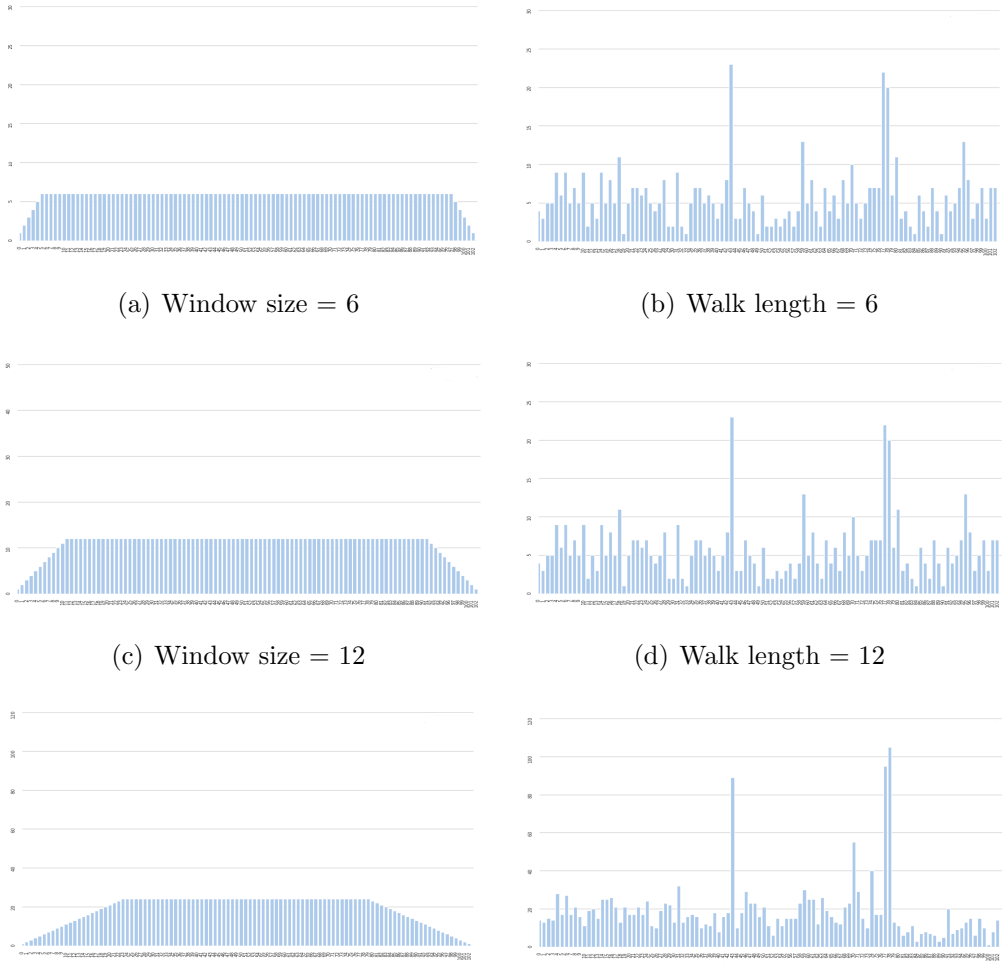
4.6.1 Experimental setup

In this task, we set the *walk_length*(walk length) to 6, 12, and 24 for graph-based gcForest and *window_size*(the sliding window size) to 6, 13, and 26 for gcForest. We only performed the scanning on the YEAST dataset, there was no training process.

4.6.2 Experimental results

As shown in Figure 4.6, in contrast to gcForest, nodes in the center of the graph or nodes with more connections, such as features 43 and 77, are more

likely to be sampled in the graph-based gcForest, which thus brings benefits.



(a) Window size = 6

(b) Walk length = 6

(c) Window size = 12

(d) Walk length = 12

(e) Window size = 24

(f) Walk length = 24

Figure 4.6: Schematic diagram of different scanning approaches on the YEAST dataset. The left panel is the sliding window-based linear scanning method, the *window_size* are 6, 12, 24, respectively. And the right panel is the graph-based nonlinear scanning method, the *walk_length* are 6, 12, 24, respectively. In the six subplots, the height of the bar represents the number of times each feature are selected, the X axis denotes the index of each feature.

Chapter 5

Melanoma dataset

In this chapter, our proposed graph-based gcForest(Section 3.1), gcForest(Section 2.1.2), Cascade Forest(Section 2.1.2) and Random Forest(Section 2.1.1) will be applied to melanoma dataset to perform classification tasks. The aim is to confirm that graph-based gcForest is competitive compared to other models.

We evaluate the multi-label classification ability of each model on the melanoma dataset based on the weighted F1 score that takes label imbalance into account. The F1 score is defined as follows:

$$Prec = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + TN} \tag{5.2}$$

$$F1 = 2 \times \frac{Prec \times Recall}{Prec + Recall} = \frac{2TP}{2TP + FN + FP} \tag{5.3}$$

where, TP stands for true positive, that is, a correctly identified instance; FP represents false positive, that is, an incorrectly identified instance; TN stands for true negative, that is, a correctly rejected instance; FN represents the false negative, that is, an incorrectly rejected instance.

In all experiments, we use the same Cascade Forest and Random Forest, which are listed in Table A.1 and A.2, respectively. Each level of Cascade Forest contains 4 Random Forests with 100 trees and 4 completely Random Forest with 100 trees. Random Forest contains 100 trees. For graph-based gcForest and gcForest, we use the following general settings, which are shown in Table A.3: During graph-based Multi-Grained Scanning and Multi-Grained Scanning , we use 1 Random Forest with 100 trees and 1 completely Random

Forest with 100 trees; For cascade structure, we use 4 Random Forests with 100 trees and 4 completely Random Forest with 100 trees.

It is noted that for different tasks some parameters of graph-based gcForest and gcForest will vary, referring to the settings in each task. For all experiments we use 3-fold cross-validation to generate class vector and use 80% of the training data as a growing set[1] and 20% as a estimating set[2].

Preprocessing of the dataset, experimental procedures, and network inference are also presented below. The list of materials used during the thesis is shown in Table A.4.

5.1 Data description and preprocessing

The Single Cell Portal was developed to facilitate sharing scientific results, and disseminating data generated from single-cell technologies. Through this web application, we retrieved the Melanoma intra-tumor heterogeneity dataset[51], henceforth referred to as the melanoma dataset, which is about the multicellular ecosystem of metastatic melanoma by single-cell RNA-seq. The melanoma dataset contains 4645 cells(samples), each of which was collected for 23,686 gene expression profiles. As Figure 5.1[3] shows, there is a multi-label classification problem. Single-cell expression profiles allow us to distinguish malignant and nonmalignant cell types and there are six subtypes of tumor cells.

Due to the sparseness and very high dimensionality of the single-cell dataset, we preprocessed the raw melanoma dataset. We selected a fraction of features for training based on the proportion of zero contained in each feature (gene).

5.2 Network inference

The melanoma dataset is characterized by genes, therefore we are able to use the STRING database to infer the gene-gene network. SILGGM is designed to infer large-scale gene networks, thus it's also perfect for our tasks.

The function *Getting the STRING network interactions* in the STRING API was used to retrieve the protein networks.

For SILGGM, we used the D-S_NW_SL method with default settings to infer gene networks. Outputs are shown with the different types of inference.

[1]Refer to Section 2.1.2

[2]Refer to Section 2.1.2

[3]This figure is from Figure 1 of Tirosh's paper[51]

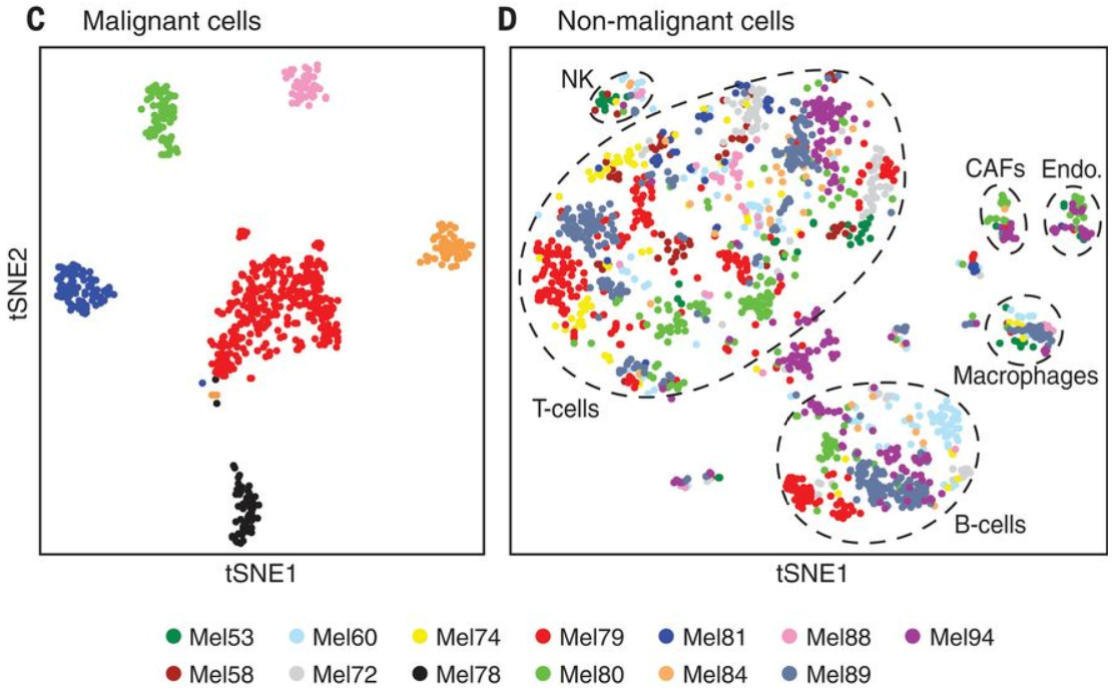


Figure 5.1: t-SNE plots of malignant(C) and nonmalignant(D) cells. Clusters of nonmalignant cells are marked by dashed ellipses and were annotated as T cells, B cells, macrophages, CAFs, and endothelial (Endo.) cells from preferentially expressed genes, NK, natural killer cells.

For individual inference of gene i and gene j, SILGGM provides the associated p-value. We filtered out connections with a p-value larger than 0.05 and thus obtained a more trustworthy gene network. The parameter settings are listed in Table A.11.

5.3 Case Study: MELANOMA40

In this task, the features containing less than 40% zeros in the MELANOMA dataset are retained, leaving a total of 1906 features, henceforth referred to as the MELANOMA40 dataset. This has two purposes, one is that the single-cell dataset is very sparse, as we mentioned in Section 5.1, and we can thus explore how much those sparse features affect the classification results. The second is that, due to computational resource limitations, we choose to start with a smaller dataset.

5.3.1 Experimental setup

The MELANOMA40 dataset was first split into training data(80%) and testing data(20%), taking label imbalances into account. Then the best p and q hyperparameters for graph-based gcForest were learned using 3-fold stratified cross-validation on the training data with a grid search over p, $q \in \{0.5, 4, 10\}$. Finally, we trained all models on the training data and tested them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, some of the parameter settings for graph-based gcForest and gcForest have changed. We set the *window_size*(sliding window size) in Multi-Grained Scanning and the *walk_length*(walk length) in graph-based Multi-Grained Scanning to 953, 1334 and 1715, which are 50%, 70%, 90% of the 1906 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 954, 573, and 193, corresponding to each walk length. The parameter settings for gcForest and graph-based gcForest are listed in Table A.12 and Table A.13, respectively.

5.3.2 Experimental results

The MELANOMA40 dataset contains 4645 samples and 1906 features. For graph-based gcForest we have two graph options, one is from the STRING database, which has 1290 nodes and 26264 edges. The other is learned by SILGGM, which has 1906 nodes and 167474 edges.

For graph-based gcForest, the best exploration strategy on the MELANOMA40 dataset is when p equals 4 and q equals 10.

The comparison results for the weighted F1 score are summarized graphically in Figure 5.2. From the results, it is evident that graph-based gcForest outperforms the other benchmark algorithms on the MELANOMA40 dataset without too much hyperparameter tuning.

5.4 Case Study: other MELANOMA

To avoid the outperformance of the graph-based gcForest due to the lucky preprocessing of the dataset, we also compare performance on the other preprocessed MELANOMA datasets with different ratios of zeros.

5.4.1 Experimental setup

We obtained the MELANOMA50, MELANOMA60, MELANOMA70, and MELANOMA80 datasets in the same way as we had obtained the

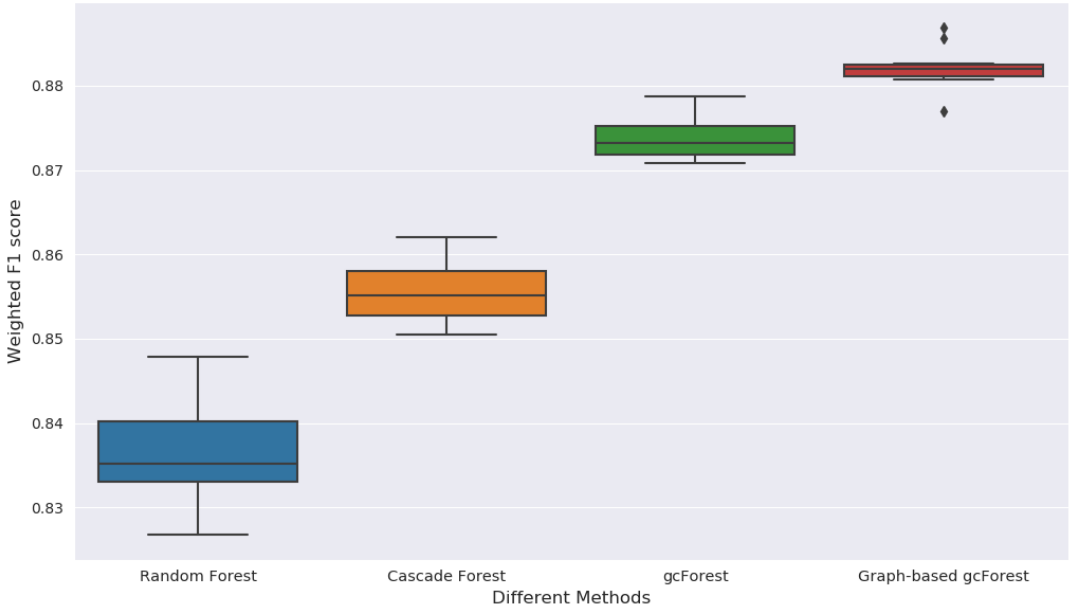


Figure 5.2: Performance evaluation of different methods on MELANOMA40 dataset.

MELANOMA40 dataset. These datasets were then split into training data(80%) and testing data(20%), taking label imbalances into account. Unfortunately, due to the computational resources limitations, we didn't do the hyperparameter tuning for these datasets. Finally, we trained all models on the training data and tested them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, some of the parameter settings for graph-based gcForest and gcForest have changed. For different datasets, the parameters were set as follows:

- MELANOMA50: We set *window_size*(the sliding window size) in Multi-Grained Scanning and *walk_length*(the walk length) in graph-based Multi-Grained Scanning to 1858, 2144 and 2429, which are 65%, 75%, 85% of the 2858 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 1001, 2144, and 2429, corresponding to each walk length.

  The parameter settings for gcForest and graph-based gcForest are listed in Table A.14 and Table A.15, respectively.

- MELANOMA60: We set *window_size*(the sliding window size) in Multi-Grained Scanning and *walk_length*(the walk length) in graph-based

Multi-Grained Scanning to 2834, 3238 and 3643, which are 70%, 80%, 90% of the 4048 features, respectively. In order to keep consistent with gcForest, we set the *n_walks*(the number of walks) to 1215, 811, and 406, corresponding to each walk length.

The parameter settings for gcForest and graph-based gcForest are listed in Table A.16 and Table A.17, respectively.

- MELANOMA70: We set *window_size*(the sliding window size) in Multi-Grained Scanning and *walk_length*(the walk length) in graph-based Multi-Grained Scanning to 5077, 5192 and 5307, which are 88%, 90%, 92% of the 5769 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 693, 578, and 463, corresponding to each walk length.

  The parameter settings for gcForest and graph-based gcForest are listed in Table A.18 and Table A.19, respectively.

- MELANOMA80: We set *window_size*(the sliding window size) in Multi-Grained Scanning and *walk_length*(the walk length) in graph-based Multi-Grained Scanning to 7638, 7802 and 7967, which are 93%, 95%, 97% of the 8213 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 576, 412, and 247, corresponding to each walk length.

  The parameter settings for gcForest and graph-based gcForest are listed in Table A.20 and Table A.21, respectively.

5.4.2 Experimental results

After preprocessing, the MELANOMA50 dataset contains 1906 features, the MELANOMA60 dataset contains 2858 features, the MELANOMA70 dataset contains 4048 features and the MELANOMA80 dataset contains 5769 features.

For graph-based gcForest we have two graph options, one is from the STRING database and the other is learned by SILGGM. The basic information about the graphs obtained through the STRING database and SILGGM refers to Table 5.1.

We use the weighted-F1 score for comparing performance in Table 5.2. Due to the computational resources limitations, we choose a large sliding window size and walk length for each different MELANOMA datasets and omit to optimize the combination of p and q parameter settings. Nevertheless, we can draw from the results is that graph-based gcForest with learned-graph

outperform the other benchmark algorithms. It should be noted that careful task-specific tuning could achieve better performance.

Table 5.1: The basic information about the graphs of the MELANOMA50, MELANOMA60, MELANOMA70 and MELANOMA80 datasets. Ground-truth graph means the graph is from the STRING database, learned graph means the graph is learned by SILGGM.

Datasets	Ground-truth Graph		Learned Graph	
	Nodes	Edges	Nodes	Edges
MELANOMA50	2014	46409	2858	353807
MELANOMA60	3022	87305	4048	574830
MELANOMA70	4540	159227	5769	1093548
MELANOMA80	6747	270595	7562	1755680

Table 5.2: Weighted-F1 score for multilabel classification on different MELANOMA datasets.

Datasets	Deep Forest		Random Forest	Graph-based Deep Forest	
	CascadeForest	gcForest		Ground truth Graph	Learned Graph
MELANOMA50	0.8384	0.8777	0.8358	0.8483	**0.8845**
MELANOMA60	0.8400	0.9047	0.8535	0.8684	**0.9104**
MELANOMA60	0.8386	0.9102	0.8611	0.8743	**0.9133**
MELANOMA70	0.8388	0.9132	0.8682	0.8778	**0.9159**

Chapter 6

Other datasets

In this chapter, our proposed graph-based gcForest(Section 3.1), gcForest(Section 2.1.2), Cascade Forest(Section 2.1.2) and Random Forest(Section 2.1.1) will be applied to Epithelium40, Dendritic40 and GAMETES dataset to perform classification tasks. The aim is to confirm that graph-based gcForest is competitive compared to other models.

We evaluate the multi-label classification ability of each model on the different datasets based on the weighted F1 score(refer to Eq.5.3) that takes label imbalance into account.

In all experiments, we use the same Cascade Forest and Random Forest, which are listed in Table A.1 and A.2, respectively. Each level of Cascade Forest contains 4 Random Forests with 100 trees and 4 completely Random Forest with 100 trees. Random Forest contains 100 trees. For graph-based gcForest and gcForest, we use the following general settings, which are shown in Table A.3: During graph-based Multi-Grained Scanning and Multi-Grained Scanning, we use 1 Random Forest with 100 trees and 1 completely Random Forest with 100 trees; For cascade structure, we use 4 Random Forests with 100 trees and 4 completely Random Forest with 100 trees.

It is noted that for different tasks some parameters of graph-based gcForest and gcForest will vary, referring to the settings in each task. For all experiments we use 3-fold cross-validation to generate class vector and use 80% of the training data as a growing set[1] and 20% as a estimating set[2].

Preprocessing of the dataset, experimental procedures, and network inference are also presented below. The list of materials used during the thesis is shown in Table A.4.

[1]Refer to Section 2.1.2
[2]Refer to Section 2.1.2

6.1 Airway epithelium dataset

6.1.1 Data description and preprocessing

Through Single Cell Portal we retrieve the Airway epithelium dataset, henceforth referred to as the Epithelium dataset, which contains 7494 epithelial cells. These cells are partitioned into seven distinct clusters by known marker gene expression[52]. As shown in Figure 6.1[3], each cluster is mapped to basal, club, ciliated, tuft, NE, goblet and ionocyte epithelial cell types.

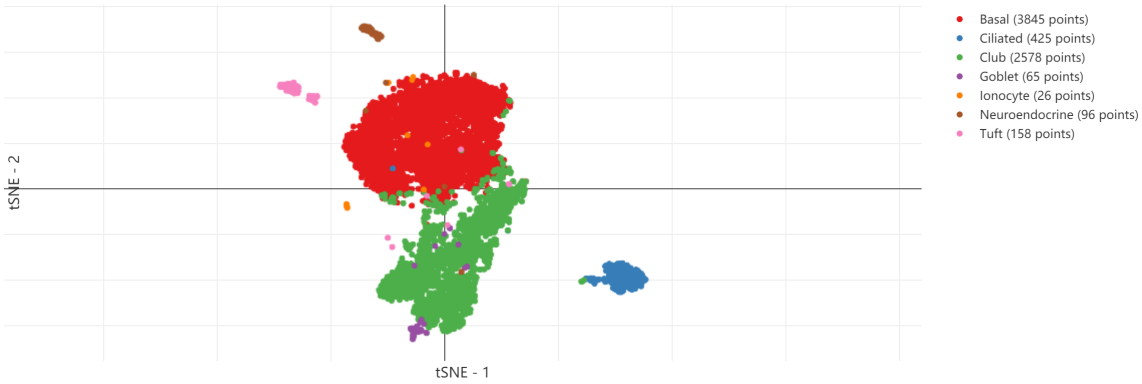


Figure 6.1: tSNE embedding of 7193 trachea epithelial cells (mouse).

6.1.2 Network inference

The Epithelium dataset is characterized by genes, therefore we are able to use the STRING database to infer the gene-gene network. SILGGM is designed to infer large-scale gene networks, thus it's also perfect for our tasks. We use the same setup as in Section 5.2.

6.1.3 Case Study: Epithelium40

For the same reasons as in Section 5.3, in this task, the features containing less than 40% zeros in the Epithelium dataset are retained, leaving a total of 558 features, henceforth referred to as the Epithelium40 dataset.

Experimental setup

The Epithelium40 dataset was first split into training data(80%) and testing data(20%), taking label imbalances into account. Unfortunately, due to the

[3]This figure is from the Airway epithelium dataset in Single Cell Portal.

computational resources limitations, we didn't do the hyperparameter tuning for this dataset. Finally, we trained all models on the training data and tested them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, some of the parameter settings for graph-based gcForest and gcForest have changed. The parameters were set as follows: We set *window_size*(the sliding window size) in Multi-Grained Scanning and *walk_length*(the walk length) in graph-based Multi-Grained Scanning to 279, 391 and 502, which are 50%, 70%, 90% of the 558 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 280, 168, and 57, corresponding to each walk length.

The parameter settings for gcForest and graph-based gcForest are listed in Table A.22 and Table A.23, respectively.

Experimental results

The Epithelium40 dataset contains 7139 samples and 558 features. For graph-based gcForest we have two graph options, one is from the STRING database, which has 443 nodes and 7579 edges. The other is learned by SILGGM, which has 558 nodes and 20180 edges.

We use the Weighted-F1 score for comparing performance in Table 6.1. Due to the computational resources limitations, we choose a relatively large sliding window size and walk length for the Epithelium40 dataset and omit to optimize the combination of p and q parameter settings. Nevertheless, we can draw from the results is that graph-based gcForest with learned-graph outperform the other benchmark algorithms. It should be noted that careful task-specific tuning could achieve better performance.

6.2 Human blood dendritic cells and monocytes dataset

6.2.1 Data description and preprocessing

Through Single Cell Portal we retrieve the Atlas of human blood dendritic cells and monocytes dataset, henceforth referred to as the Dendritic dataset, which contains 1078 cells. By their single-cell profiles, we can identify and validate six dendritic cells and four monocyte subtypes, as shown in Figure 6.2[4].

[4]This figure is from the Atlas of human blood dendritic cells and monocytes dataset in Single Cell Portal.

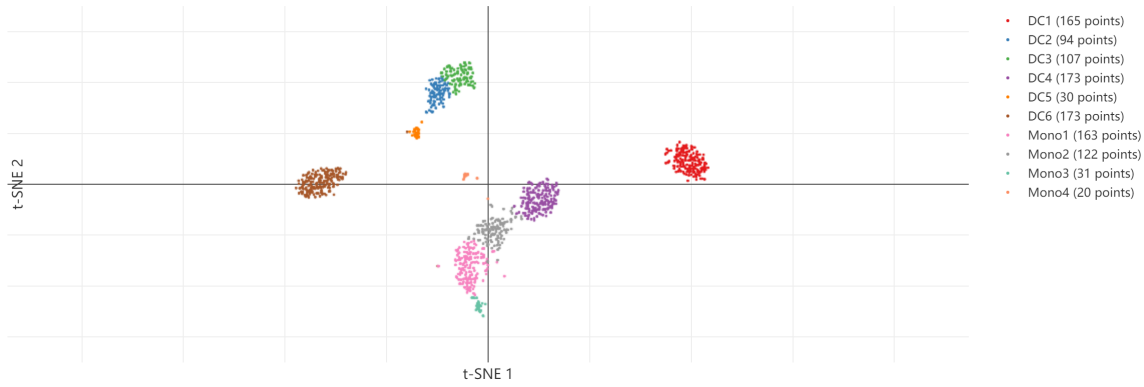


Figure 6.2: t-SNE analysis incorporating monocytes (n = 337) and dendritic cells (n = 742).

6.2.2 Network inference

The Dendritic dataset is characterized by genes, therefore we are able to use the STRING database to infer the gene-gene network. SILGGM is designed to infer large-scale gene networks, thus it's also perfect for our tasks. We use the same setup as in Section 5.2.

6.2.3 Case Study: Dendritic40

For the same reasons as in Section 5.3, in this task, the features containing less than 40% zeros in the Dendritic dataset are retained, leaving a total of 558 features, henceforth referred to as the Dendritic40 dataset.

Experimental setup

The Dendritic40 dataset was first split into training data(80%) and testing data(20%), taking label imbalances into account. Unfortunately, due to the computational resources limitations, we didn't do the hyperparameter tuning for this dataset. Finally, we trained all models on the training data and tested them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, some of the parameter settings for graph-based gcForest and gcForest have changed. The parameters were set as follows: We set $window\_size$(the sliding window size) in Multi-Grained Scanning and $walk\_length$(the walk length) in graph-based Multi-Grained Scanning to 1564, 2189 and 2814, which are 50%, 70%, 90% of the 3127 features, respectively. In order to keep consistent with gcForest, we set n_walk(the number of walks) to 1564, 939, and 314, corresponding to each walk length.

The parameter settings for gcForest and graph-based gcForest are listed in Table A.24 and Table A.25, respectively.

Experimental results

The Dendritic40 dataset contains 1078 samples and 3127 features. For graph-based gcForest we have two graph options, one is from the STRING database, which has 2339 nodes and 65856 edges. The other is learned by SILGGM, which has 3127 nodes and 325739 edges.

We use the weighted-F1 score for comparing performance in Table 6.1. Due to the computational resources limitations, we choose a relatively large sliding window size and walk length for the Dendritic40 dataset and omit to optimize the combination of p and q parameter settings. Nevertheless, we can draw from the results is that graph-based gcForest with learned-graph outperform the other benchmark algorithms. It should be noted that careful task-specific tuning could achieve better performance.

6.3 GAMETES dataset

6.3.1 Data description and preprocessing

The GAMETES dataset[53] is a synthetic data, which is about epistatic patterns of association in 'mock' single nucleotide polymorphism(SNP) data generated by the GAMETES genetic-data simulation software[54]. The GAMETES dataset contains 1600 samples, 1000 features, and there is a binary classification problem. We obtain this dataset through PMLB[55].

6.3.2 Network inference

Since the features of the GAMETES dataset are anonymous, we thus didn't have a ground-truth graph. In addition, because the size of the GAMATES dataset exceeds the Bnlearn's ability to handle, we didn't use Bnlearn to infer the graph structure either. Thus, in the following task, we have only one graph from SILGGM available.

In SILGGM, we used the D-S_NW_SL method with default settings to infer gene networks. Outputs are shown with the different types of inference. For individual inference of gene i and gene j, SILGGM provides the associated p-value. We filtered out connections with a p-value larger than 0.05 and thus obtained a more trustworthy gene network. Refer to Table A.11.

6.3.3 Case Study: GAMETES

Experimental setup

The GAMETES dataset was first split into training data(80%) and testing data(20%), taking label imbalances into account. Unfortunately, due to the computational resources limitations, we didn't do the hyperparameter tuning for this dataset. Finally, we trained all models on the training data and tested them on the testing data, repeating 10 times with 10 random seed initializations.

In this task, some of the parameter settings for graph-based gcForest and gcForest have changed. The parameters were set as follows: We set *window_size*(the sliding window size) in Multi-Grained Scanning and the *walk_length*(walk length) in graph-based Multi-Grained Scanning to 63, 125 and 250, which are $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$ of the 1000 features, respectively. In order to keep consistent with gcForest, we set *n_walks*(the number of walks) to 938, 876, and 751, corresponding to each walk length.

The parameter settings for gcForest and graph-based gcForest are listed in Table A.26 and Table A.27, respectively.

Experimental results

The GAMETES dataset contains 1000 samples and 1000 features. For graph-based gcForest, we have only one graph that is learned by SILGGM, which has 1000 nodes and 3848 edges.

We use the F1 score for comparing performance in Table 6.1. We omit to optimize the combination of p and q parameter settings. Nevertheless, we can draw from the results is that graph-based gcForest with learned-graph significantly outperform the other benchmark algorithms, achieving 11.8% gain over Cascade Forest. It should be noted that careful task-specific tuning could achieve better performance.

Table 6.1: F1 score for classification performance on Epithelium40, Dendritic40 and GAMETES datasets.

Dataset	Deep Forest		Random Forest	Graph-based Deep Forest	
	CascadeForest	gcForest		Ground truth Graph	Learned Graph
Epithelium40	0.9528	0.9504	0.9461	0.9356	**0.9559**
Dendritic40	0.8430	0.8600	0.8307	0.8178	**0.8636**
GAMETES	0.5074	0.4875	0.4827	-	**0.5669**

Chapter 7

Discussion and outlook

7.1 Summary

This thesis is about our proposed graph-based gcForest. The biggest difference between gcForest and graph-based gcForest is that the Cascade Forest in graph-based gcForest employs non-linear graph-based Multi-Grained Scanning, whereas the Cascade Forest in gcForest employs linear sliding-window-based Multi-Grained Scanning. Our aim is to explore whether Cascade Forest can derive greater benefits from non-linear scanning compared to linear scanning. Biomedical data often contain an intrinsic graph structure and are therefore well suited to our graph-based gcForest. In order to demonstrate the capabilities of graph-based gcForest, we compare them with other machine learning algorithms on biomedical data.

First of all, we implemented gcForest by ourselves. After obtaining the performance equivalent to the official gcForest, we began to modify the gcForest. We used the NODE2WALK algorithm of NODE2VEC and combined it with the Multi-Grained Scanning in the gcForest to enable it to perform Multi-Grained Scanning on the graph, which is called nonlinear graph-based Multi-Grained Scanning.

Second, in order to validate the capabilities of graph-based gcForest on biological data, we focused on finding some biologically relevant datasets. To make a quick validation, we used the YEAST dataset from openML, which only consists of 2417 samples and 103 features. After achieving some results, we applied graph-based gcForest to cancer-related datasets from the TCGA database, but even with a simple Random Forest, we were able to achieve very high classification accuracy, so that it is hard to evaluate the capabilities of graph-based gcForest, therefore we abandoned the use of these datasets. After that, we tried multiple single-cell datasets from Single Cell Portal, Melanoma dataset, Epithelium dataset, and Dendritic dataset, respectively. They all

have low-sample, high-dimensional characteristics, and because their features are genes at different loci, they have potential graph structures, making these datasets ideal for our task. In addition, for the diversity of the datasets, we used the GAMETES dataset from PMLB, which is a synthetic biologically relevant dataset.

Finally, regarding the inference of graph structure, since the YEAST dataset is small, we inferred its graph structure using Bnlearn and SILGGM. For the single-cell datasets with higher dimensionality, Bnlearn is not capable of handling it, thus we chose to obtain its ground-truth graph directly from the STRING database and use SILGGM to infer its graph structure. For the GAMETES dataset, since its features are anonymous, we can only use SILGGM to infer its graph structure.

In Chapter 1, we briefly describe the motivation as well as an overview of the thesis. In Chapter 2, we present the relevant literature and background. In Chapter 3, we present the core of the thesis, the graph-based deep forest, including its pseudo-code and illustrations and related hyperparameters.

In Chapter 4, graph-based gcForest was applied to the YEAST dataset for several classification problems. In Section 4.3, for comparison purposes, the other three ensemble tree methods, including Random Forest(RF), Cascade Forest(CF), and gcForest, are also run. We used nested cross-validation to avoid the risk of data leakage. After getting comparable results, we explore and visualize how the hyperparameters of graph-based gcForest work. In Section 4.4, since the quality of the graph has a great impact on the performance of the graph-based gcForest, we explored the impact of different graphs on the graph-based gcForest. As shown in Figure 4.4, the graph-based gcForest prefers a relatively sparse graph structure. In Section 4.5, we explored the effect of the number of walks on the graph-based gcForest. As illustrated in Figure 4.5, as the number of walks increases, the performance of graph-based gcForest also increases, and then stabilizes.

In Chapter 5 and Chapter 6, graph-based gcForest was used to accomplish the task of subclass mapping in three single-cell datasets, including Melanoma dataset, Epithelium dataset, and Dendritic dataset, and one synthetic GAMATES dataset. For each case study, experimental performance across different datasets demonstrate the effectiveness and robustness of our proposed graph-based gcForest.

7.2 Conclusion

Biomedical data often contain graph structures. We hope therefore that gcForest can exploit these properties and thus obtain better feature repre-

sentations of biomedical data. Inspired by this idea, we combine gcForest with node2vec and then propose graph-based gcForest. In this thesis, we apply graph-based gcForest to solve some problems arising from the biomedical field.

From the results section, it is clear that graph-based gcForest outperforms all other classifiers on biomedical datasets. The gains of graph-based gcForest can be explained on the basis of the data scanning strategy. Compared to the sliding window-based Multi-Grained Scanning, the graph-based Multi-Grained Scanning take the structural information of the data into account, which means that the scan of the data using the graph structure is non-linear and able to capture the local structure and features of the data, unlike in gcForest where only linear scans can be performed. Additionally, the two hyperparameters p and q allow the data scanning process to be adapted to different data types. This increases the flexibility and complexity of the Multi-Grained Scanning on the one hand; on the other hand, this increases the degree of interpretability, making it more likely that the feature space generated by the Multi-Grained Scanning will be consistent with the related biological knowledge. For instance, the BFS search strategy can explore the structural equivalences in networks based on the local structure of the nodes. On the other hand, the DFS search strategy is free to explore the network and discover homophilous communities at the cost of high variance[18].

We showed how the different graph structures affect the performance of the graph-based gcForest. Intuitively, there is a trade-off between a dense graph structure that contains more noise and a sparse graph structure that may lose more information. In general, graph-based gcForest prefer a relatively sparse graph structure(Section 4.4), but as shown in Chapter 5 and Chapter 6, a very sparse graph structure is fatal to graph-based gcForest.

In many tasks, the Multi-Grained Scanning in gcForest boosts Cascade Forest a lot. Its success lies in the fact that after the Multi-Grained Scanning, the raw features obtain a better quality feature representation. However, an obvious drawback of the Multi-Grained Scanning is that once we determine $window\_size$(the sliding window size), the number of transformed features produced is fixed, which means the Multi-Grained Scanning gives us no control over the number of transformed features. Obviously, both the quantity and quality of transformed features will have an impact on the performance of the model. In graph-based Multi-Grained Scanning, $walk\_length$(the walk length) and n_walks(the number of walks) can be set arbitrarily. The graph-based gcForest provides flexibility in the process of representation of raw features(Section 4.5).

The building blocks of Multi-Grained Scanning are a Random Forest and a completely Random Forest, and thus the essence of feature representation

in Multi-Grained Scanning is still ensemble learning, which requires that each classifier learns a diverse but good feature. The graph-based nonlinear scan will obviously produce more diverse data, compared to the sliding window-based linear scan. From this perspective, it can also explain why graph-based gcForest performs better than gcForest in our experiments(Section 4.6).

In conclusion, graph-based gcForest prefers high-dimensional data. As we can see in Chapter 4, we need to adjust some hyperparameters so that graph-based gcForest is able to outperform other algorithms. In Chapter 5 and Chapter 6, with simple configurations graph-based gcForest can outperform other algorithms. This may be due to the fact that the complex network contains more information, such as structural information, so the graph-based gcForest is more advantageous.

7.3 Challenges

As the authors of gcForest point out, gcForest has the ability to handle image data. However, it is difficult for a graph-based gcForest to learn the structure or network relationships between individual pixels, thus graph-based gcForest is not suitable for image data. In addition, graph-based gcForest prefers high-dimensional data and does not perform well on low-dimensional data. These two points limit the scope of application of graph-based gcForest.

As we mentioned above, the performance of the graph-based gcForest relies on the quality of the graph. In biology, one of the most common problems with the application of graphical models is that the dimensionality of the data is far greater than the size of the sample. In other words, there are many variables to be considered whereas the number of observations is very small. If the number of features is large, the parameters used to describe the A graphical model (e.g. edge probabilities) quickly outnumber the data points[56]. Therefore, it is difficult to obtain learn a graph structure for a high-dimensional low-sample biological dataset.

Regarding computational complexity, although we have implemented parallel graph-based gcForest, graph-based Multi-Grained Scanning still consumes a lot of memory. In addition, walking on the graph takes more time than sliding windows. Finally, the graph-based gcForest is an ensemble learning framework, If the base classifiers become complicated, for instance, the number of decision trees in the Random Forest is increased from 100 to 500, then the graph-based gcForest will consume several times the time increase.

A common problem of ensemble learning is that it is not highly interpretable, although it can achieve better performance than traditional machine

learning algorithms, such as Random Forest. But we can't output the importance of every feature like Random Forest, which makes it impossible for us to distinguish which features are beneficial and which are harmful in graph-based gcForest. In the biological field, the interpretability of a machine learning method is very important, otherwise, we cannot use it to guide experiments.

7.4 Outlook

GcForest has been noted in many papers, and now some variants of deep forest have appeared, such as BCDForest[57]. These variants focus on the modification of Cascade Forest, and our proposed graph-based gcForest mainly modifies the part of Multi-Grained Scanning. In the future, it will be interesting to combine our graph-based Multi-Grained Scanning with other gcForest variants.

As an ensemble learning framework, Graph-based gcForest provides us with high scalability. The base classifiers do not have to be Random Forests or completely Random Forests, which can be replaced by some other classic machine learning algorithms, such as, Support Vector Machine(SVM), Linear Regression model(LR), Gradient Boosting Decision Tree(GBDT), etc. As for sampling on the graph, we adopt the Node2walk sampling strategy(Section 2.9) in graph-based gcForest. However, this sampling strategy can also be replaced by some other sampling methods. We very much look forward to seeing variants in terms of the above two directions in the future.

The graph-based gcForest is a decision tree ensemble method, which combines multiple learners for the same task[58]. The core of constructing a good ensemble is to make each individual learner as accurate and diverse as possible, yet there is no widely accepted definition of diversity[58, 59]. In graph-based Multi-Grained Scanning, we use a Random Forest and a completely Random Forest. And in Cascade Forest, each cascade level contains multiple levels. The aim of doing this is to enhance diversity. In future work, we can use more and different base classifiers to transform the original features or make each layer of the Cascade Forest wider and more diverse, i.e., using more and different base classifiers.

Due to computational resources limitations, in all of the above experiments, we chose relatively simple configurations of graph-based gcForest, such as longer walk length and fewer number of walks, which helped us to reduce the complexity of calculations. However, we are also curious about what will happen if we use a shorter walk length and a larger number of walks. We think shorter walk lengths are able to capture more detail, but at the same time, the noise and the variation would be greater, which might be addressed by a larger number of walks.

We showed that graph-based gcForest improves compared to gcForest on several biomedical datasets. As future work, we would like to explore the reasons for its success and to make the parameters more interpretable. It would also be interesting to borrow some ideas from graph-related Deep Neural Networks.

Appendix A

Further Tables and Figures

Table A.1: The parameter settings for Random Forest in all experiments. Python implementation of random forests are used in scikit-learn.

Parameters	**Values**
n_estimators	100
max_features	If Random Forest, then **max_features = "sqrt"**; If completely Random Forest, then **max_features = 1**.
n_jobs	-1

Table A.2: The parameter settings for Cascade Forest in all experiments.

Parameters	**Values**
n_trees	100
n_forests	4 Random Forest, 4 completely Random Forest.
max_features	If Random Forest, then **max_features = "sqrt"**; If completely Random Forest, then **max_features = 1**.
n_jobs	-1

Table A.3: The general parameter settings of gcForest and graph-based gcForest in all experiments.

Parameters	**Values**
n_trees	100
n_forests	In Cascade Forest: 4 Random Forest, 4 completely Random Forest; In Multi-Grained Scanning and graph-based Multi-Grained Scanning: 1 Random Forest, 1 completely Random Forest
max_features	If Random Forest, then **max_features = "sqrt"**; If completely Random Forest, then **max_features = 1**.
n_jobs	-1

Table A.4: Applications and packages.

Applications and Packages	**Version**
Python	3.7.7
scikit-learn	0.22.1
Numpy	1.18.1
Matplotlib	3.1.3
Pandas	1.0.2
Networkx	2.4
R	4.0
SILGGM	1.0.0 in CRAN
Bnlearn	4.5 in CRAN
OpenML	latest
Single Cell Portal	latest
STRING database	latest

Table A.5: The parameter settings for Bnlearn on YEAST dataset.

Parameters	**Values**
Score-based structure learning algorithms	Hill-climbing(HC)

Table A.6: The parameter settings for SILGGM on YEAST dataset.

Parameters	**Values**
Method	B_NW_SL
FDR value	FDR value $\in$ [0.1, 0.05, 0.01, 0.001]

Table A.7: The parameter settings for gcForest on YEAST dataset. d denotes the dimension of YEAST dataset, i.e., 103.

Parameters	**Values**
window size	{[d/16] = 6, [d/8] = 13, [d/4] = 26)}

Table A.8: The parameter settings for Graph-based gcForest on YEAST dataset. d denotes the dimension of YEAST dataset i.e., 103. Assume the walk length is 6, the corresponding number of the walk is 103-6+1=98. We use the graph learned by SILGGM and corrected with FDR value equals 0.05.

Parameters	**Values**
walk length	{[d/16] = 6, [d/8] = 13, [d/4] = 26)}
n_walks	{98, 91, 78}
p and q	In validation phase: p ∈ [0.5, 1, 2, 4], q ∈ [0.5, 1, 2, 4]

Table A.9: The parameter settings for Graph-based gcForest on YEAST dataset. d denotes the dimension of YEAST dataset, i.e., 103. Assume the walk length is 6, the corresponding number of the walk is 103-6+1=98. There are six graph options, refer to Table A.5 and Table A.6.

Parameters	**Values**
walk length	{[d/16] = 6, [d/8] = 13, [d/4] = 26)}
n_walks	{98, 91, 78}
p and q	p:1 q:4

Table A.10: The parameter settings for Graph-based gcForest on YEAST dataset. d denotes the dimension of YEAST dataset i.e., 103. We use the graph learned by SILGGM and corrected with FDR value equals 0.05.

Parameters	**Values**
walk length	{[d/16] = 6, [d/8] = 13, [d/4] = 26)}
n_walks	{$0.1w$ = (10, 9, 8); $0.2w$ = (20, 18, 16); $0.3w$ = (29, 27, 23); $0.4w$ = (39, 36, 31); $0.5w$ = (49, 46, 39); $0.6w$ = (59, 55, 47); $0.7w$ = (69, 64, 55); $0.8w$ = (78, 73, 62); $0.9w$ = (88, 82, 70); $1w$ = (98, 91, 78); $2w$ = (196, 182, 156); $3w$ = (294, 273, 234)}
p and q	p: 1, q: 4.

Table A.11: The parameter settings for SILGGM on MELANOMA40 dataset.

Parameters	**Values**
Method	D-S_NW_SL
p value	0.05

Table A.12: The parameter settings for gcForest on MELANOMA40 dataset. d denotes the dimension of MELANOMA40 dataset, i.e., 1906.

Parameters	**Values**
window size	{[0.5d] = 953, [0.7d] = 1334, [0.9d] = 1715)}

Table A.13: The parameter settings for Graph-based gcForest on MELANOMA40 dataset. d denotes the dimension of MELANOMA40 dataset, i.e., 1906. Assume the walk length is 953, the corresponding number of the walk is 1906-953+1=954.

Parameters	**Values**
walk length	{[0.5d] = 953, [0.7d] = 1334, [0.9d] = 1715)}
n_walks	{954, 573, 193}
p and q	In validation phase: p ∈[0.25, 4, 10], q ∈ [0.25, 1, 4, 10]; In testing phase: p: 4, q: 10.

Table A.14: The parameter settings for gcForest on MELANOMA50 dataset. d denotes the dimension of MELANOMA50 dataset, i.e., 2858.

Parameters	**Values**
window size	{[0.65d] = 1858, [0.75d] = 2144, [0.85d] = 2429)}

Table A.15: The parameter settings for Graph-based gcForest on MELANOMA50 dataset. d denotes the dimension of MELANOMA50 dataset, i.e., 2858. Assume the walk length is 1858, the corresponding number of the walk is 2858-1858+1=1001.

Parameters	**Values**
walk length	{[0.65d] = 1858, [0.75d] = 2144, [0.85d] = 2429)}
n_walks	{1001, 715, 430}
p and q	p: 100 ; q: 100

Table A.16: The parameter settings for gcForest on MELANOMA60 dataset. d denotes the dimension of MELANOMA60 dataset, i.e., 4048.

Parameters	**Values**
window size	{[0.7d] = 2834, [0.8d] = 3238, [0.9d] = 3643)}

Table A.17: The parameter settings for Graph-based gcForest on MELANOMA60 dataset. d denotes the dimension of MELANOMA60 dataset, i.e., 4048. Assume the walk length is 2834, the corresponding number of the walk is 4048-2834+1=1215.

Parameters	**Values**
walk length	{[0.7d] = 2834, [0.8d] = 3238, [0.9d] = 3643)}
n_walks	{1215, 811, 406}
p and q	p: 100 ; q: 100

Table A.18: The parameter settings for gcForest on MELANOMA70 dataset. d denotes the dimension of MELANOMA70 dataset, i.e., 5769.

Parameters	**Values**
window size	{[0.88d] = 5077, [0.9d] = 5192, [0.92d] = 5307)}

Table A.19: The parameter settings for Graph-based gcForest on MELANOMA70 dataset. d denotes the dimension of MELANOMA70 dataset, i.e., 5769. Assume the walk length is 5077, the corresponding number of the walk is 5769-5077+1=693.

Parameters	**Values**
walk length	{[0.88d] = 5077, [0.9d] = 5192, [0.92d] = 5307)}
n_walks	{693, 578, 463}
p and q	p: 100 ; q: 100

Table A.20: The parameter settings for gcForest on MELANOMA80 dataset. d denotes the dimension of MELANOMA80 dataset, i.e., 8213.

Parameters	**Values**
window size	{[0.93d] = 7638, [0.95d] = 7802, [0.97d] = 7967)}

Table A.21: The parameter settings for Graph-based gcForest on MELANOMA80 dataset. d denotes the dimension of MELANOMA80 dataset, i.e., 8213. Assume the walk length is 7638, the corresponding number of the walk is 8213-7638+1=576.

Parameters	**Values**
walk length	{[0.93d] = 7638, [0.95d] = 7802, [0.97d] = 7967)}
n_walks	{576, 412, 247}
p and q	p: 100 ; q: 100

Table A.22: The parameter settings for gcForest on Epithelium40 dataset. d denotes the dimension of Epithelium40 dataset, i.e., 558.

Parameters	**Values**
window size	{[0.5d] = 279, [0.7d] = 391, [0.9d] = 502)}

Table A.23: The parameter settings for Graph-based gcForest on Epithelium40 dataset. d denotes the dimension of Dendritic40 dataset, i.e., 558. Assume the walk length is 279, the corresponding number of the walk is 558-279+1=280.

Parameters	**Values**
walk length	{[0.5d] = 279, [0.7d] = 391, [0.9d] = 502)}
n_walks	{280, 168, 57}
p and q	p: 100 ; q: 100

Table A.24: The parameter settings for gcForest on Dendritic40 dataset. d denotes the dimension of Dendritic40 dataset, i.e., 3127.

Parameters	**Values**
window size	{[0.5d] = 1564, [0.7d] = 2189, [0.9d] = 2814)}

Table A.25: The parameter settings for Graph-based gcForest on Dendritic40 dataset. d denotes the dimension of Epithelium40 dataset, i.e., 3127. Assume the walk length is 1564, the corresponding number of the walk is 3127-1564+1=1564.

Parameters	**Values**
walk length	{[0.5d] = 1564, [0.7d] = 2189, [0.9d] = 2814)}
n_walks	{1564, 939, 314}
p and q	p: 100 ; q: 100

Table A.26: The parameter settings for gcForest on GAMETES dataset. d denotes the dimension of GAMETES dataset, i.e., 1000.

Parameters	**Values**
window size	{[$\frac{1}{16}$d] = 63, [$\frac{1}{8}$d] = 125, [$\frac{1}{4}$d] = 250)}

Table A.27: The parameter settings for Graph-based gcForest on GAMETES dataset. d denotes the dimension of GAMETES dataset, i.e., 1000. Assume the walk length is 63, the corresponding number of the walk is 1000-63+1=938.

Parameters	**Values**
walk length	{[$\frac{1}{16}$d] = 63, [$\frac{1}{8}$d] = 125, [$\frac{1}{4}$d] = 250)}
n_walks	{938, 876, 751}
p and q	p: 100 ; q: 100

Bibliography

[1] P. Larrañaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J. A. Lozano, R. Armañanzas, G. Santafé, A. Pérez, and V. Robles, "Machine learning in bioinformatics," *Briefings in Bioinformatics*, vol. 7, pp. 86–112, 03 2006.

[2] E. Picardi and G. Pesole, "Computational methods for ab initio and comparative gene finding," *Methods in molecular biology (Clifton, N.J.)*, vol. 609, pp. 269–84, 10 2010.

[3] R. Karlic, H.-R. Chung, J. Lasserre, K. Vlahoviček, and M. Vingron, "Histone modification levels are predictive for gene expression," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 107, pp. 2926–31, 02 2010.

[4] R. Upstill-Goddard, D. Eccles, J. Fliege, and A. Collins, "Machine learning approaches for the discovery of gene–gene interactions in disease data," *Briefings in Bioinformatics*, vol. 14, pp. 251–260, 05 2012.

[5] M. Libbrecht and W. Noble, "Machine learning applications in genetics and genomics," *Nature reviews. Genetics*, vol. 16, 05 2015.

[6] D. M. Camacho, K. M. Collins, R. K. Powers, J. C. Costello, and J. J. Collins, "Next-generation machine learning for biological networks," *Cell*, vol. 173, no. 7, pp. 1581 – 1592, 2018.

[7] G. Stiglic, S. Kocbek, I. Pernek, and P. Kokol, "Comprehensive decision tree models in bioinformatics," *PLOS ONE*, vol. 7, pp. 1–13, 03 2012.

[8] X. Chen and H. Ishwaran, "Random forests for genomic data analysis," *Genomics*, vol. 99, no. 6, pp. 323 – 329, 2012.

[9] E. Byvatov and G. Schneider, "Support vector machine applications in bioinformatics," *Applied bioinformatics*, vol. 2, pp. 67–77, 02 2003.

[10] S. Isci, H. Dogan, C. Ozturk, and H. H. Otu, "Bayesian network prior: network analysis of biological data using external knowledge," *Bioinformatics*, vol. 30, pp. 860–867, 11 2013.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[14] S. Min, B. Lee, and S. Yoon, "Deep learning in bioinformatics," *Briefings in Bioinformatics*, vol. 18, pp. 851–869, 07 2016.

[15] M. K. K. Leung, A. Delong, B. Alipanahi, and B. J. Frey, "Machine learning in genomic medicine: A review of computational problems and data sets," *Proceedings of the IEEE*, vol. 104, pp. 176–197, 2016.

[16] P. Mamoshina, A. Vieira, E. Putin, and A. Zhavoronkov, "Applications of deep learning in biomedicine," *Molecular Pharmaceutics*, vol. 13, no. 5, pp. 1445–1454, 2016. PMID: 27007977.

[17] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3553–3559, 2017.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

[19] M. Scutari, "Learning bayesian networks with the bnlearn R package," *Journal of Statistical Software*, vol. 35, no. 3, pp. 1–22, 2010.

[20] R. Zhang, Z. Ren, and W. Chen, "Silggm: An extensive r package for efficient statistical inference in large-scale gene networks," *PLoS Computational Biology*, vol. 14, 2018.

[21] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. von Mering, "String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets," *Nucleic Acids Research*, vol. 47, pp. D607–D613, 2019.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[23] W.-Y. Loh, "Classification and regression trees," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 1, pp. 14–23, 2011.

[24] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[25] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, (San Francisco, CA, USA), p. 148–156, Morgan Kaufmann Publishers Inc., 1996.

[26] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, p. 5–32, Oct. 2001.

[27] R. E. Schapire, "A brief introduction to boosting," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, (San Francisco, CA, USA), p. 1401–1406, Morgan Kaufmann Publishers Inc., 1999.

[28] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *Ann. Statist.*, vol. 29, pp. 1189–1232, 10 2001.

[29] F. T. Liu, K. Ting, Y. Yu, and Z.-H. Zhou, "Spectrum of variable-random trees," *J. Artif. Intell. Res. (JAIR)*, vol. 32, pp. 355–384, 05 2008.

[30] H. Peiro Sajjad, A. Docherty, and Y. Tyshetskiy, "Efficient representation learning using randomwalks for dynamic graphs." QC 20190208.

[31] L. Lovász, L. Lov, and O. Erdos, "Random walks on graphs: A survey," pp. 1–46, 01 1996.

[32] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 855–864, Association for Computing Machinery, 2016.

[33] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 3111–3119, Curran Associates, Inc., 2013.

[34] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: Structural role extraction mining in large graphs," in *KDD'12 - 18th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1231–1239, Sept. 2012. 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2012 ; Conference date: 12-08-2012 Through 16-08-2012.

[35] J. Yang and J. Leskovec, "Overlapping communities explain core–periphery organization of networks," *Proceedings of the IEEE*, vol. 102, no. 12, pp. 1892–1902, 2014.

[36] R. Albert, "Network inference, analysis, and modeling in systems biology," *Plant Cell*, vol. 19, pp. 3327–3338, Nov. 2007.

[37] J. philippe Vert and Y. Yamanishi, "Supervised graph inference," in *Advances in Neural Information Processing Systems 17* (L. K. Saul, Y. Weiss, and L. Bottou, eds.), pp. 1433–1440, MIT Press, 2005.

[38] R. Jansen, H. Yu, D. Greenbaum, Y. Kluger, N. J. Krogan, S. Chung, A. Emili, M. Snyder, J. F. Greenblatt, and M. Gerstein, "A bayesian networks approach for predicting protein-protein interactions from genomic data," *Science*, vol. 302, no. 5644, pp. 449–453, 2003.

[39] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using bayesian networks to analyze expression data," *Journal of computational biology : a journal of computational molecular cell biology*, vol. 7, pp. 601–20, 02 2000.

[40] M. Kanehisa, "Prediction of higher order functional networks from genomic data," *Pharmacogenomics*, vol. 2, no. 4, pp. 373–385, 2001. PMID: 11722287.

[41] K. B. Korb and A. E. Nicholson, *Bayesian Artificial Intelligence, Second Edition*. USA: CRC Press, Inc., 2nd ed., 2010.

[42] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. 01 2009.

[43] C. von Mering, L. J. Jensen, B. Snel, S. D. Hooper, M. Krupp, M. Foglierini, N. Jouffre, M. A. Huynen, and P. Bork, "STRING: known and predicted protein–protein associations, integrated and transferred across organisms," *Nucleic Acids Research*, vol. 33, pp. D433–D437, 01 2005.

[44] Z. Ren, T. Sun, C.-H. Zhang, and H. Zhou, "Asymptotic normality and optimalities in estimation of large gaussian graphical model," *The Annals of Statistics*, vol. 43, 09 2013.

[45] J. Janková and S. Geer, "Honest confidence regions and optimality in high-dimensional precision matrix estimation," *TEST*, 07 2015.

[46] J. Jankova and S. Geer, "Confidence intervals for high-dimensional inverse covariance estimation," *Electronic Journal of Statistics*, vol. 9, 03 2014.

[47] W. Liu, "Gaussian graphical model estimation with false discovery rate control," *The Annals of Statistics*, vol. 41, 06 2013.

[48] W. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 09 2017.

[49] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo, "Openml: Networked science in machine learning," *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013.

[50] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," vol. 14, pp. 681–687, 01 2001.

[51] I. Tirosh, B. Izar, S. M. Prakadan, M. H. Wadsworth, D. Treacy, J. J. Trombetta, A. Rotem, C. Rodman, C. Lian, G. Murphy, M. Fallahi-Sichani, K. Dutton-Regester, J.-R. Lin, O. Cohen, P. Shah, D. Lu, A. S. Genshaft, T. K. Hughes, C. G. K. Ziegler, S. W. Kazer, A. Gaillard, K. E. Kolb, A.-C. Villani, C. M. Johannessen, A. Y. Andreev, E. M. Van Allen, M. Bertagnolli, P. K. Sorger, R. J. Sullivan, K. T. Flaherty, D. T. Frederick, J. Jané-Valbuena, C. H. Yoon, O. Rozenblatt-Rosen, A. K. Shalek, A. Regev, and L. A. Garraway, "Dissecting the multicellular ecosystem of metastatic melanoma by single-cell rna-seq," *Science*, vol. 352, no. 6282, pp. 189–196, 2016.

[52] D. Montoro, A. Haber, M. Biton, V. Vinarsky, B. Lin, S. Birket, F. Yuan, S. Chen, H. Leung, J. Villoria, N. Rogel, G. Burgin, A. Tsankov, A. Waghray, M. Slyper, J. Waldman, L. Nguyen, D. Dionne, O. Rozenblatt-Rosen, and J. Rajagopal, "A revised airway epithelial hierarchy includes cftr-expressing ionocytes," *Nature*, vol. 560, 08 2018.

[53] R. J. Urbanowicz, J. Kiralis, J. M. Fisher, and J. H. Moore, "Predicting the difficulty of pure, strict, epistatic models: metrics for simulated model selection," *BioData Mining*, vol. 5, p. 15, September 2012.

[54] R. J. Urbanowicz, J. Kiralis, N. A. Sinnott-Armstrong, T. Heberling, J. M. Fisher, and J. H. Moore, "Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures," *BioData Mining*, vol. 5, p. 16, October 2012.

[55] R. Olson, W. La Cava, P. Orzechowski, R. Urbanowicz, and J. Moore, "Pmlb: A large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, 03 2017.

[56] M. Scutari and K. Strimmer, "Introduction to graphical modelling," *Handbook of Statistical Systems Biology*, 05 2010.

[57] Y. Guo, L. Shuhui, Z. Li, and X. Shang, "Bcdforest: A boosting cascade deep forest model towards the classification of cancer subtypes based on gene expression data," *BMC Bioinformatics*, vol. 19, 04 2018.

[58] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Chapman Hall/CRC, 1st ed., 2012.

[59] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Mach. Learn.*, vol. 51, p. 181–207, May 2003.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift