Faculty of Natural and Mathematical Sciences
Department of Informatics

King's College London
Strand Campus
London
United Kingdom

7CCSMPRJ
Individual Project Submission 2019/20

Name: Keying Zhang
Student Number: 1909300
Degree Programme: Data Science MSc
Project Title: Graph Similarity Computation
Supervisor: Loukides Grigorios
Word count: 7764

RELEASE OF PROJECT
Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☒ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: Keying Zhang

Date: 08/16/2020

Department of Informatics
King's College London
United Kingdom

7CCSMPRJ/7CCSMUIP MSc Project

GRAPH SIMILARITY COMPUTATION

Student Name: Keying Zhang
Student Number: 1909300
Degree Programme: Data Science MSc

Supervisor's Name: Loukides Grigorios

This dissertation is submitted for the degree of MSc in Data Science

Abstract

Graphs similarity computation problem is one of the most crucial graph-based issues. Although standard similarity measures, such as Graph Edit Distance (GED) and Maximum Common Subgraph (MCS) can calculate exact similarity scores, yet they are computationally costly and require polynomial time complexity over the number of nodes. In recent years, machine learning approaches show another possible solution to graph similarity calculation by converting it to a learning problem. Inspired by the success of graph embedding methods and machine learning models, in this work, I combine many state-of-the-art entire graph embedding approaches with some simple machine learning models and propose a novel framework to address the challenging graph similarity problem, aiming to speed up the graph similarity calculation while preserving satisfactory performances.

The proposed framework includes two strategies. In the first strategy, the similarity metric is transformed into a continuous variable ranging from 0 to 1. In the following stage, I pick several new entire graph embedding methods to encode graphs to feature vectors. Then these feature vectors are used as the input of Multi-Layer Perception (MLP) regression models. In the second strategy, the similarity measure is regarded as a discrete variable. The feature extraction process is the same as that in strategy one. Then I feed these feature vectors into several classification models, such as Decision tree, K-Nearest Neighbor (KNN), and MLP. Taking MCS as an example, experiments on two datasets demonstrate the effectiveness and efficiency of my proposed framework. Specifically, some combinations in my flamework achieve excellent evaluation scores while significantly reducing a great time in both strategies.

Keywords: Graph similarity Computation, Graph Embedding, Maximum Common Subgraph, Machine Learning

Acknowledgment

I would like to thank Dr. Grigorios Loukides for his expert advice and encouragement throughout this individual project.

I would also like to thank my family and friends whose moral support accompanied me in this tough period.

Furthermore, special thanks are also given to all teachers and staff of the Department of Informatics, King's College London.

Table of Contents

Nomenclature

GED Graph Edit Distance
MCS Maximum Common Subgraph
MCIS Maximum Common Induced Subgraph
MCES Maximum Common Edge Subgraph
CNN Convolutional Neural Network
LSTM Long short-term Memory Network
MLP Multi-Layer Perception
MSE Mean Squared Error
KNN K Nearest Neighbour

List of Figures and Tables

Table 7 Results on AIDS and IMDB (strategy two)
Table 8 Time evaluation (strategy two)

1. Introduction

1.1 Motivation

Data that are represented as graphs can be found in a wide range of domains, including social networks [1], biology [2], pharmacy [3], recommender systems [4], and network attacks analysis [5]. Constructing a similarity metric between two graphs is a crucial stage for further studies, such as graph/node classification [6], graph/node clustering [7], node retrieval [8], similarity search [9], link prediction [10], etc. Although in the past few decades, many methods and algorithms have been proposed to study the similarity of graphs, most of them rely on sophisticated mathematics and require high computational complexity such as Graph Edit Distance(GED) [11], Maximum Common Subgraph(MCS) [12], and Graph Isomorphism [13]. For example, even some heuristic approaches have been put forward to improve the performance of GED, it still needs to store enormous intermediate calculation results, which can only be applied to compute graphs with less than 16 vertices [11]. Both GED and MCS are known as NP-hard problem [14]. More recently, an increasing number of researchers began to formulate graph similarity estimation as a learning problem to give a pair of graphs a similarity score based on graph representations [13], such as Graph Kernels, Graph Neural networks, and Graph Embedding. Although these methods are still not capable of handling well on graphs of complex structures, they generally have lower computational costs. Specifically, in the field of Graph Embedding, a more significant number of approaches have emerged in the decade, which can be categorized into two groups roughly: node level embedding and graph level embedding [15]. So far, many works have concentrated on graph embeddings, but few works combine the effort with graph similarity computation. To improve the efficiency of graph similarity computation while preserving an excellent performance, I propose a new framework to compute similarity scores in a pair of graphs based on some state-of-art graph embedding algorithms and existed classification models in the machine learning field.

1.2 Aims & objectives

The project is aimed to design a novel framework for learning computing similarity scores with a low computational cost. Instead of calculating the precise similarity scores directly, my design turns the problem into a machine learning problem.

To achieve the aim, the objectives of this study are:
(1) Study the literature to find out how similarity is calculated in a pair of graphs and what are the possible methods to speed up the computation process.
(2) Design methods that can improve the efficiency of graph similarity computation theoretically.
(3) Implement the selected exact similarity computation algorithm MCS and the proposed fast approaches.
(4) Test and evaluate the performance of the proposed methods and the exact MCS to find out whether the proposed ways are better than the exact one.

1.3 Contribution

My contribution can be summarized as follows:
(1) I address the classical graph similarity computation problem by transforming it into

regression and classification machine learning problems in two strategies. At the feature extracting stage, five state-of-art methods like FEATHER [16], Gl2Vec [17], SF [18], FGSD [19] and NetLSD [52] are used to produce the whole graph feature as a multidimensional vector individually. At the training stage, obtained feature vectors from a pair of graphs are used as the input of the MLP Regression Model in strategy one and predicted numeric similarity scores are the output. The MLP Regression Model will learn to minimize the differences between the predicted values and the ground-truth scores. When using the original discrete number of MCS in strategy two, the same obtained vectors from a pair of graphs will be used as input of classification models like Decision Tree, KNN, and MLP. At the test stage, by feeding the learned MLP Regression Model or classification models with any pair of graphs features, I can get the predicted similarity score.

(2) To demonstrate and compare the effectiveness and efficiency of my proposed framework, I complete experiments on a widely used graph similarity metric, MCS. It is shown that the proposed methods obtain very competitive results compared to the original MCS even only using the simplest machine learning models. For example, in strategy one, The SF based model is 68996 and 382241 times faster than the exact MCS approach on AIDS and IMDB while obtaining R squared of 0.83 and 0.89, respectively. And in strategy two, the SF-DT method is 2727222 and 943655 times faster than the exact MCS method on AIDS and IMDB while achieving high accuracies of 0.98 and 0.88 individually.

1.4 Organization

The rest of the paper is organized as follows.
Chapter 2(Background) will describe some technical background for the design and implementation of my proposed approach, including traditional graph similarity metrics, Neural Network and Multi-Layer Perception, and Model Evaluation.

Chapter3(Related Work) will present a state-of-art literature review in Graph Embedding and classical graphs similarity metrics.

Chapter4(Design and Implementation) will describe the design and implementation of the proposed framework.

Chapter5(Experiments) will conduct experiments and evaluate the performances of my proposed framework, providing graphical demonstrations.

Chapter6(Conclusion) will provide a summary of the individual project and discuss the possible work which can be explored in the future to improve the algorithm.

2. Background

2.1 Graph Similarity Metrics

Graph similarity computation is closely related to many other research regions such as graph classification, graph match, graph clustering, etc. Therefore, many algorithms and similarity measures have been suggested in the decade. The proposed approaches can be categorized into three types: graph isomorphism (Graph Edit Distance and Maximum Common Subgraph), feature extraction, and iterative algorithms [54]. Among them, graph isomorphism-based measures can calculate the exact and explainable results, but they require exponential time as nodes increasing. Thus, they cannot be applied to large graphs. For feature extraction-based algorithms, they rely heavily on the features that are selected. It is very likely to get not intuitive and explainable results though they generally scale well and can be very fast. For example, even a pair of graphs consisting of very different numbers of nodes, feature extraction-based approach is possible to give the high similarity score, which is not desirable. The core idea behind iteration-based algorithms is that a pair of nodes are similar if their neighbors are similar. Most algorithms in this branch usually couple compute similarity scores of nodes and edges at first, then, the similarity score of a pair of graphs. In this paper, I choose to use MCS, a graph isomorphism-based algorithm, as the similarity score that is explainable and intuitional.

2.1.1 Graph and Graph Isomorphism

Graphs consist of a set of vertices and edges and can be represented as nodes and edges. There are many types of graphs according to different kinds of nodes and edges. For instance, in a graph, edges can be directed or undirected, weighted or unweighted. Likely, nodes can have attributes or not. In this paper, graphs are simple, undirected, and unweighted graphs without node attributes. A graph exists in various forms of representation having the same number of vertices, edges, and edge connectivity. Such graphs are named isomorphic graphs. Both the Maximum Common Subgraph (MCS) and Graph Edit Distance (GED) are developed based on this concept. For example, the graphs in Figure1 are isomorphic, despite their various visual representations. More formally, I use the definition and terminology from [21]: A pair of graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ are isomorphic if there is a bijection φ between V_1 and V_2 such that for every pair of nodes $i, j \in V_1$, $(i, j) \in E_1$ if and only if $(\varphi(i), \varphi(j)) \in E_2$. Symbolically we denote this by $G_1 \equiv G_2$. A bijection with this property is called an isomorphism. In the above definition, graphs are simple graphs that are unlabeled, undirected, and unweighted.

Figure 1 Example of isomorphic graphs

2.1.2 Maximum Common Subgraph

Maximum Common Subgraph (MSC) is a core and widely used measure for determining graph similarity [12]. The Subgraph Isomorphism problem is to find a copy of a small pattern graph inside a larger graph [22]. When the same pattern does not exist, we may want to be

given a set that contains as many vertices or edges of the pattern as possible. According to this, MCS can be classified as Maximum Common Induced Subgraph (MCIS) and Maximum Common Edge Subgraph (MCES). In this paper, I focus on MCIS and call it MCI in the rest paper for convenience. Maximum Common Subgraph (MCS) problem has been proven NP-hard and is far more computationally extensive than Subgraph Isomorphism problem. Figure 2 provides an illustration of MCIS of a pair of graphs. The maximum common induced subgraph of the below graphs contains 4 vertices.

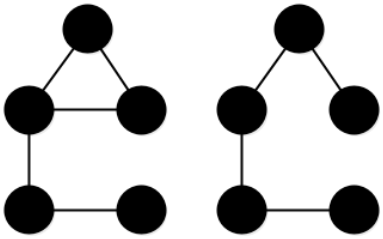

Figure 2 Example of MCS

Once the number of vertices in the maximum common induced subgraph is obtained, I transform it into a similarity score ranging from 0 to 1. More details about the transformation will be introduced in Chapter4.

2.1.3 Graph Edit Distance

Graph Edit Distance (GED) is the generalization of the graph isomorphism problem [54]. It is one of the most popular and well-established metrics to measure graph similarity by evaluating an error-correcting graph isomorphism [23]. It can be applied to various types of graphs, such as directed, undirected, labeled, unlabeled graphs. Formally, Given two graphs G and Q, the GED between them can be denoted as $ged(G,Q)$. GED is defined as the minimum number of edit operations that transform one graph to another [11]. An edit operation on a graph in this paper is an insertion or deletion of a vertex/edge. For example, Figure 3 demonstrates an example of GED. The graph edit distance between the left graph and the right graph is two, as the transformation needs 2 edit operations: (1) an edge deletion, (2) an edge insertion.

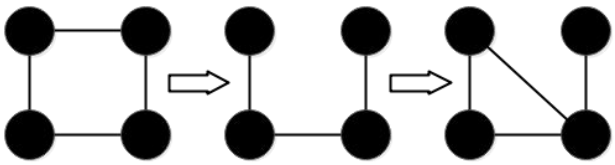

Figure 3 Example of GED

2.1.4 Relation Between MCS and GED

MCS and GED are both widely used similarity metrics in many fields, such as graph classification, graph clustering, graph matching, and graph embedding. It has been proved by researchers that under a particular cost function, GED computation is equivalent to the MCS problem [53].

2.2 Neural Network and Multi-Layer Perception

Neural Network has become extremely popular in the decade and is a form of bio-inspired machine learning models and can be used in both classification and regression tasks. It can classify data that is not linearly separable. The learning algorithm of a neural network can be supervised learning, unsupervised learning, and reinforcement learning according to different scenarios. In this paper, I only use supervised neural networks to calculate graph similarity. So far, there existed many neural network variants, such as Back Propagation (BP) Neural Network, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long short-term Memory Network (LSTM), etc. However, the most straightforward and original form of the neural network is Multi-Layer Perception (MLP).

The most typical MLP consists of three layers: an input layer, a hidden layer, and an output layer as it is shown in Figure 4. The MLP neural network is fully connected between different layers. Figure 5 shows the structure of the MLP. There is three essential elements in MLP: (1) weights, (2) Bias, (3) activation function. The hidden layer and output layer utilize a defined non-linear activation function that can combine input features with weights of the neurons and compute bias. The procedure is demonstrated in Figure 5.

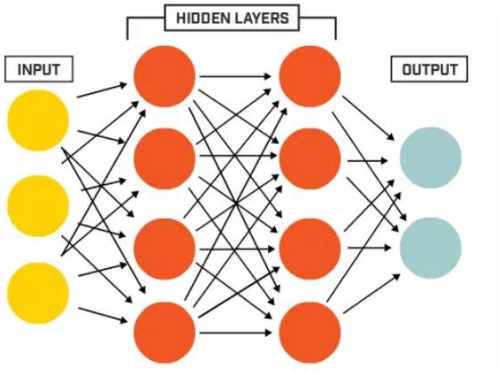

Figure 4 Structure of MLP [57]

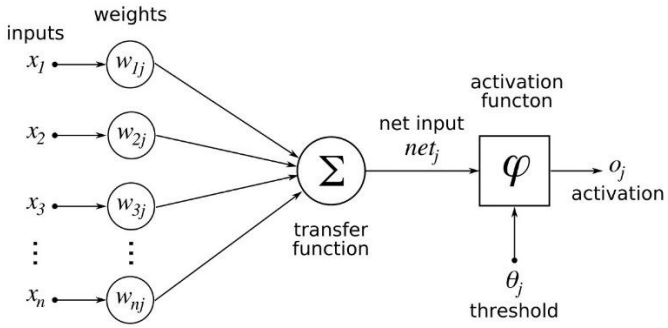

Figure 5 MLP Model [56]

2.3 Model Evaluation

2.3.1 Cross Validation

Cross Validation is a statistical approach used to split the dataset into folds to train and test models repeatedly using various combinations of folds. There are many types of cross validation methods, such as N-fold Cross Validation, Leave-One-Out Cross Validation, etc. In this paper, I choose N-fold cross validation to evaluate my proposed model. This method guarantees that the score of my model does not depend on the dataset division for picking the train and test set [24]. The steps are as follows.

(1) Split the entire dataset into N approximately equal subsets randomly.
(2) Execute N times the model that you want to evaluate where one subset is employed for testing, and the rest subsets are used for training.
(3) Calculate the error rate after each repetition and average the results to yield an overall error rate.

2.3.2 Performance Measures of Numerical Predictive Models

(1) Mean Squared Error (MSE)

In numerical predictive models, $D=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_m,y_m)\}$ is the sample, y_i represents the ground-truth value for $\mathbf{x}_i$. To evaluate the performance of the model f, the predicted value $f(\mathbf{x})$ will be compared will ground true value y. The most commonly used performance measure in regression tasks is the Mean Squared Error (MSE).

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\left(f(\mathbf{x}_i)-y_i\right)^2$$

This measure is ranging from 0 to infinity. The larger MSE value indicates the worse the model. When MSE value is equal to 0, it is a perfect model [58].

(2) R Squared (R^2)

R Squared (R^2), also called the coefficient of determination, varies from 0 to 1 and is a normalized version of MSE. R^2 measures the proportion of the variance for a dependent variable's behavior that's explained by an independent variable's behavior. It can also be used to measure how well the data fit the regression model. An R^2 of 1 indicates that the dependent variable is entirely explained by the independent ones. Although high R Squared does not always denote the goodness of the regression model, it provides useful insights. A higher R squared reveals a better data fit for the regression model. Most times, people draw a conclusion about their model by using R squared together with other metrics in a statistical model [25].

$$\mathrm{R}^2=1-\frac{\text{Unexplained Variation}}{\text{Total Variation}}$$

2.3.3 Performance Measures of Classification Models

In the machine learning field, specifically the classification problems, a confusion matrix is widely used to describe the performance of models in a tabular way. A confusion matrix provides a summary of predicted values. Each entry in the matrix is the figure of predicted values that are obtained by the model. Classification accuracy is the most commonly used measure to evaluate the performance of classification models. It can be calculated by using the number of right predictions divided by the total number of predictions. Here is an example showing how to use a confusion matrix to calculate accuracy in a multi-class prediction problem [26].

		Predicted Class			
		a	b	c	Total
Actual Class	a	88	10	2	100
	b	14	40	6	60

	c	18	10	12	40
	Total	120	60	20	200

Table 1 Example of the confusion matrix in multiclass problems

Total Accuracy (Precision) = (88 + 40 + 12) / 200 = 70%

Most data are imbalanced in multiclass classification problems. With imbalanced classes, it is not hard to obtain a high accuracy score without actually predicting useful results. Therefore, accuracy should not be used as the only metric to evaluate the multiclass classification models in which class labels are not uniformly distributed. The confusion matrix is an excellent tool to supplement performance. By looking at the confusion matrix, we can clearly find out how well the model is in predicting each class.

2.3.4 Kendall Rank Correlation Coefficient

A rank correlation is used to measure an ordinal association, and Kendall rank correlation is a type of rank correlation. Kendall rank correlation, also commonly referred to as "Kendall's tau coefficient", is performed to test the similarities in the ordering of data when it is ranked by quantities [54]. Specifically, the correlation coefficient will return a value ranging from -1 to 1. Values close to 0 indicate that there is no relationship while values close to 1 or-1 denote that there exists a perfect positive or negative relationship. Another commonly used rank correlation is Spearman's rank correlation coefficient. Compared to Spearman's rank correlation coefficient, Kendall rank correlation is a non-parametric metric that requires no assumptions of the test. Formally, if we have two samples A and B and their sample size is N. The total combinations of pairings of the element in A and B is $\frac{1}{2}N(N-1)$. Kendall rank correlation can be computed using the formula below [59]:

$$\tau = \frac{N_c - N_d}{\frac{1}{2}N(N-1)}$$

N_c denotes the number of concordant

N_d denotes the number of discordant

3. Related Work

3.1 Graph Embedding

Graph analysis, such as graph classification, graph similarity computation, and node recommendation, provides people a better comprehension of the data from a wide diversity of scenarios in the world. Most proposed models to solve graph-based problems either works on graph adjacency matrix or on a derived vector space [27]. In recent years, Graph Embedding, an approach based on representing graphs in vector space while remaining the features, has been attracting increasing attention due to its effectiveness and efficiency compared with the traditional adjacency matrix [28]. There are multiple ways to categorize Graph Embedding, and details can be found in [13, 14, 15, 28]. In this Section, I roughly classify it into node level embedding and graph level embedding and briefly introduce some traditional and state-of-the-art methods in each classification as follows.

3.1.1 Node Level Embedding

There are a large number of methods that have been proposed for embeddings. Most of them can be categorized into two categories: (1) matrix factorization-based, and (2) random walk-based [29]. Matrix factorization, also called matrix decomposition, is a very sought-after approach used to reduce a complex matrix into constituent parts. It will be easier to operate on the constituent parts than the original complicated matrix [60]. Random walk on simple graphs is the process that starts in a chosen node then keeps moving to the neighbor of a current node randomly till reaching the number of specified steps [61]. Most works focused on matrix factorization-based algorithms are proposed early, including LLE [30], Laplacian Eigenmaps [31, 32], GraphRep [33], Hope [34], etc. In recent years, random walk-based approaches have become a popular branch to extract the properties of graphs combining with natural language processing models. There are three classical steps in the development of Random Walk-based node level embedding approaches in general. At the first stage, DeepWalk was proposed on the basis of Word2vec [35, 36] that is a method applying Word2vec in the NLP domain to Graph Embedding with low costs [37] in 2014. Then, in the following year. Tang Jian et al. improved DeepWalk and proposed Large-scale Information Network Embedding (LINE). LINE took different walking strategies in comparison with DeepWalk, which allows LINE to be used in a wide type of networks, such as directed graphs, undirected graphs, and weighted graphs. By introducing the first- and second-order neighbor relations into the objective function, the distribution of the node embedding learned at the end can be more balanced and smoother [38]. Then in 2016, Grover Aditya et al. proposed Node2vec [39]. The main difference between DeepWalk and Node2vec is that the latter conducts random walks biasedly that offers a trade-off between breadth-first (BFS) and depth-first (DFS) graph searches [14]. Recently, the field continues to thrive. In 2018, Alibaba proposed Enhanced Graph Embedding with Side Information (EGES) [40] to cope with the problem of cold start in embedding.

3.1.2 Graph Level Embedding

A large number of works concentrate on nodes embedding of graphs. However, some machine learning models, such as graph similarity computation, graph classification, and graph clustering, require the embeddings of entire graphs. Inspired by the success of Random Walk and word embedding models, such as Word2vec [35] and Doc2vec [41] related to node

level embedding, Graph2vec [20], Gl2vec [17] and FEATHER [16] are proposed in sequence. Graph2vec is an extension to Node2vec which is introduced in previous. Graph2vec is classical and based on Doc2vec. A document is made of sentences while a graph is made of subgraphs. Gl2vec improved the limitations of Graph2vec, which can handle edge labels and extracted better structural information on graphs. Benedek et al. proposed FEATHER this year. In the FEATHER algorithm, a newly defined characteristic function on graphs to describe the distribution of node features in several aspects shows better performance than most existed algorithms in both node classification tasks(DeepWalk, LINE, Node2Vec, GraRep) and graph classification tasks(GL2vec, Graph2Vec, etc.).

Apart from random walk-based node embedding approaches, some simple but fast embedding algorithms significantly outperform than many sophisticated ones in classification experiments, including FGSD [19] and, NetLSD [52], SF [18]. FGSD embeds the graphs based on the discovery of a family of graph spectral distances. SF and NetLSD both use spectral decomposition and graph Laplacian to perform graph embedding process.

3.2 Graph Similarity Metrics

3.2.1 GED-based Approaches

Computing GED directly is known as the NP-hard problem. It becomes infeasible as the number of vertexes and edges increases. Even most state-of-the-art approaches cannot guarantee to calculate reliable GED within polynomial time in a pair of graphs more than 16 vertexes [42]. Many works turn into computing approximate GED based on heuristic algorithms, including A*-GED [43], DF-GED [44], and CSI_GED [45], etc.

3.2.2 MCS-based Approaches

MCS has been proven NP-hard. Over the years, some works motivate to perform the algorithm in parallel [46, 47]. Some methods are proposed to approximate MCS to improve efficiency [48, 49, 50]. Specifically, genetic algorithms [49], branch and bound algorithms [46, 40] are combined to reduce the searching cost.

4. Design and Implementation

The previous chapter provided a review of recent literature on the area of graph embedding and classical graph similarity metrics. This chapter describes the development process of feature extraction, similarity metrics transformation, creating the machine learning models, and their parameter settings. In each step of the process, the implementation tools like software, related packages, and functions are presented.

4.1 Feature Extraction

4.1.1 Implementation Tools

NetworkX is a free Python package to manipulate and study complex graphs. It contains many standard graph algorithms, such as GED and ISMAGS. I can obtain MCS by operating on the ISMAGS algorithm. It is capable of seeking all subgraphs and MCS in a pair of graphs.

Karate Club [51] is an open-source Python framework on Github, consisting of many state-of-the-art approaches, such as Deepwalk, Node2vec, and Graph2Vec, etc., to do graph analysis. Implement approaches cover a wide range of conferences, workshops, and journals.

Centos and Windows operating systems are chosen to implement the algorithms. I installed Python3.7, then NetworkX, Karate Club, and other related libraries in the Centos7.4 virtual machine installed on Oracle VM VirtualBox and Windows10.

After completing the configuration of the environments, I can start coding to extract graph features.

4.1.2 FEATHER [16] (CIKM 2020)

FEATHER is a computationally efficient neighborhood-based graph embedding method that can be used in both node embedding level and graph embedding level. In this algorithm, the main contribution is that a characteristic function is defined as transition probabilities of random walks on graph vertices to describe node features based on random walks. In this paper, I will use ***FeatherGraph()*** API in the Karate Club library to implement FEATHER directly, and all graphs I feed into it are unweighted, unlabeled, and undirected graphs after standardization. And the output is stored in the format of the array. Each line of the array represents an embedded whole graph vector. Some parameter settings will be introduced in the experiment part.

4.1.3 GL2Vec [17] (ICONIP 2019)

As it is mentioned before, GL2Vec improves Graph2Vec by preserving more structural information and edge label information. Specifically, GL2Vec uses the line graphs to complement the missed edge label information based on the property of the line graph that can turn the edge label into the node label. There exists an API ***GL2Vec()*** to realize the algorithm in Karate Club. Some parameter settings will be introduced in the experiment part.

4.1.4 NetLSD [52] (KDD 2018)

NetLSD produces whole graph vectors by extracting properties of the Laplacian spectrum using the heat or wave kernel in a pair of graphs. The approach is designed for a fast comparison of large graphs based on three properties of the heat trace: (1) Permutation-invariance, (2) Scale-adaptivity, and (3) Size-invariance. According to the paper, NetLSD outperforms FGSD on various datasets in the area of graph classification. I achieve this approach using the API ***NetLSD()*** in the Karate Club. Some parameter settings will be introduced in the experiment part.

4.1.5 SF [18] (NeurIPS RRL Workshop 2018)

FEATHER and GL2Vec can be classified as random walk base methods while SF is different from them. SF is a very simple and fast approach to represent graphs based on the spectral decomposition of graph Laplacian. An API ***SF()*** in the Karate Club package can perform the algorithm straightly. Some parameter settings will be introduced in the experiment part.

4.1.6 FGSD [19] (NeurIPS 2017)

FGSD (Family of Graph Spectral Distances) is discovered to represent unlabeled graphs and demonstrates good uniqueness, stability, sparsity. Also, it can be computed very efficiently, and perform well in classification tasks. I use ***FGSD()*** API in the Karate Club library to realize the algorithm. Some parameter settings will be introduced in the experiment part.

4.2 Strategy One

Strategy One is a numerical prediction model based on the simplest neural network model MLP. I propose a graph similarity metric based on MCS ranging from 0 to 1 before training the model. Two graphs are believed identical if the proposed similarity measure is equal to 1.

4.2.1 Implementation Tools

Scikit-learn is a simple and efficient machine learning package in Python. I use it to perform MLP in strategy one. Scikit-learn is built on Numpy, SciPy, and Matplotlib and consists of a wide range of classification, clustering, regression, and evaluation algorithms, such as Decision Tree, Random Forrest, K-Means, and Logistical Regression.

4.2.2 Similarity Measure

$G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are a pair of graphs, and the Maximum Common Subgraph (MCS) based similarity metric is defined as,

$$MCS(G_1, G_2) = \frac{mcs(G_1, G_2)}{\max(G_1, G_2)}$$

In the equation above, $mcs(G_1, G_2)$ represents the common number of vertexes in two graphs and $\max(G_1, G_2)$ denotes the largest possible number of common vertexes. If a pair of graphs are the same, which means there exists an isomorphism between them, then the figure of $mcs(G_1, G_2)$ equals that of $\max(G_1, G_2)$, and the similarity score will be 1. On the opposite, if G_1 and G_2 are entirely different, the value of $mcs(G_1, G_2)$ will be 0 resulting in

the similarity score of 0. In other cases, the similarity score will lie between 0 to 1. The value of $\mathrm{MCS}(G_1, G_2)$ is regarded as a ground-truth similarity score to train the following prediction model.

Before starting to train the model, the designed ground-truth similarity scores need to be calculated, including all possible pair graph combinations. ***Networkx.number_of_nodes()*** denotes using ***number_of_nodes()*** in NetworkX to compute the number of nodes in a graph. ***Networkx.mcs()*** is used to find the number of common nodes of a pair of graphs. ***Dataframe()*** is the operation to transform lists to a dataframe. The output *msc_dataframe* stores the graph numbers of a pair graph and their similarity measure.

Algorithm 1: MCS Numerical Similarity Metric

```
graphs_list – List of graphs imported by NetworkX
function MCS_Num (graphs_list):
  k ← 0
  mcs_num ← Initialize an empty list
  graphno1 ← Initialize an empty list
  graphno2 ← Initialize an empty list
  for i in 0: length(graphs_list) do
    for j in i: length(graphs_list) do
      true_num_of_common_nodes ← Networkx.mcs(graphs_list[i], graphs_list[j])
      if (Networkx.number_of_nodes(graphs_list[i]) > Networkx.number_of_nodes(graphs_list[i])) do
       max_num_of_common_node ← Networkx.number_of_nodes(graphs_list[i])
      else do
       max_num_of_common_node ← Networkx.number_of_nodes(graphs_list[j])
       mcs_score ← true_num_of_common_nodes/ max_num_of_common_node
       mcs_num[k] ← mcs_score
       graphno1[k] ← i
       graphno2[k] ← j
       k ++
  end
  mcs_dataframe ← Dataframe(graphno1, graphno2,msc_num)
  Output mcs_dataframe
end function
```

4.2.3 Numerical variables prediction model

The Multi-Layer Perceptron (MLP) algorithm is a simple and classical form of Neural Network that is widely used for both regressions as well as classification. The variable *embedding_vector* is the output of the previous feature extraction model in the form of an array. Assuming the shape of the array is M x N. The total number of rows M denotes the number of graphs and the total number of columns N represents the selected number of dimensions of an entire graph feature vector. I use the ***sklearn.MLP()*** to create the MLP regression model based on the training set. ***sklearn.MSE()*** is used to compute the Mean Squared Error (MSE) of the test dataset to evaluate the model. ***sklearn.r2()*** is used to obtain the R squared of the model and ***scipy.stats. kendalltau()*** is used to get Kendall rank correlation of the model.

Algorithm 2: MLP Regression Model

```
1: embedding_vectors – an array of all graph vectors
2: mcs_dataframe – the dataframe records the number a pair graphs and their score
3: function MLP (embedding_vectors, mcs_dataframe):
4: combine_vectors←Dataframe(embedding_vectors[mcs_dataframe.graphno1], embedding_vectors[mcs_dataframe.graphno2])
5:  scores ← mcs_dataframe.mcs_num
6:  model ← sklearn.MLP(combine_vectors, scores)
7:  mse ← sklearn.MSE(model)
8:  r2 ← sklearn.r2(model)
9:  kentau← scipy.stats. kendalltau (model)
10:  Output mse, r2, kentau
11: end function
```

4.3 Strategy Two

4.3.1 Implementation Tools

I also use Scikit-learn to achieve KNN, Decision Tree, and MLP Classification Model in strategy two.

4.3.2 Similarity Measure

In strategy two, inspired by the development of embedding techniques in recent years, I consider this graph similarity computation problem as a classification problem. The number of common nodes in two graphs found by the MCS approach is regarded as discrete categories. I test several simple classification models to speed up the calculation process while preserving accuracy.

Compared with Computing its numerical version, it only needs to find the number of common nodes in a pair graph using ***Networkx.mcs()*** with no more additional operations.

Algorithm 3: MCS Discrete Similarity Metric

```
1: graphs_list – List of graphs imported by NetworkX
2: function MCS_Dis (graphs_list):
3:    k ← 0
4:    mcs_dis ← Initialize an empty list
5:    graphno1 ← Initialize an empty list
6:    graphno2 ← Initialize an empty list
7:    for i in 0: length(graphs_list) do
8:      for j in i: length(graphs_list) do
9:        true_num_of_common_nodes ← Networkx.mcs(graphs_list[i], graphs_list[j])
10:       mcs_dis[k] ← true_num_of_common_nodes
11:       graphno1[k] ← i
12:       graphno2[k] ← j
13:       k ++
14:   end
15:   mcs_dataframe ← Dataframe(graphno1, graphno2, msc_num)
```

16: Output mcs_dataframe
27: end function

4.3.2 Classification Model

(1) KNN
K-nearest neighbors (KNN) algorithm is a non-parametric method that can be used for both classification and regression. I use ***sklearn.KNN()*** to achieve the KNN algorithm based on the training set. ***sklearn.accuracy()*** is used to compute the accuracy of the test dataset to evaluate the model. The definition of accuracy is specified in the Background. Common variables in Algorithm 4 have been explained in Algorithm 2. Few core parameter settings about KNN will be introduced in the next Experiments section.

Algorithm 4: KNN

1: embedding_vectors – an array of all graph vectors
2: mcs_dataframe – the dataframe records the number a pair graphs and their score
3: ***function*** KNN (embedding_vectors, mcs_dataframe):
4: combine_vectors←***Dataframe(***embedding_vectors[mcs_dataframe.graphno1], embedding_vectors[mcs_dataframe.graphno2]***)***
5: scores ← mcs_dataframe.mcs_num
6: model ← ***sklearn.KNN(***combine_vectors, scores***)***
7: accuracy ← ***sklearn.accuracy(***model***)***
8: Output accuracy
9: end function

(2) Decision Tree
A decision tree is a simple type of classification model that can be used to predict the category of the target variable based on prior training data. ***sklearn.Decision_Tree()*** represents using the function in sklearn package to implement the decision tree model. Common variables in Algorithm 5 have been explained in Algorithm 2 Few core parameter settings about the decision tree will be presented in the next Experiments section.

Algorithm 5: Decision Tree

1: embedding_vectors – an array of all graph vectors
2: mcs_dataframe – the dataframe records the number a pair graphs and their score
3: ***function*** Decision_Tree (embedding_vectors, mcs_dataframe):
4: combine_vectors←***Dataframe(***embedding_vectors[mcs_dataframe.graphno1], embedding_vectors[mcs_dataframe.graphno2]***)***
5: scores ← mcs_dataframe.mcs_num
6: model ← ***sklearn.Decision_Tree(***combine_vectors, scores***)***
7: accuracy ← ***sklearn.accuracy(***model***)***
8: Output accuracy
9: end function

(3) MLP Classification Model
This is the classification version of MLP. I use sklearn library to achieve this. Common variables in Algorithm 11 have been explained in Algorithm 2. Few core parameter settings about the model will be introduced in the next Experiments section.

Algorithm 6: MLP Classification Model

```
1: embedding_vectors – an array of all graph vectors
2: mcs_dataframe – the dataframe records the number a pair graphs and their score
3: function MLP_Classififer (embedding_vectors, mcs_dataframe):
4:  combine_vectors←Dataframe(embedding_vectors[mcs_dataframe.graphno1], embedding_vectors[mcs_dataframe.graphno2])
5:  scores ← mcs_dataframe.mcs_num
6:  model ← sklearn.MLP_Classififer(combine_vectors, scores)
7:  accuracy ← sklearn.accuracy(model)
8:  Output accuracy
9: end function
```

5. Experiments

5.1 Datasets

Two well-known graph datasets are used for the experiments. A summary to the datasets is shown in Table2.

Dataset	Graph Meaning	Num of graphs	Min Nodes	Max Nodes
AIDS	Chemical Compounds	697	2	10
IMDB	Social Networks	238	12	15

Table 2 A summary of datasets used for the experiment

(1) AIDS
The AIDS dataset is a set of 42687 chemical compound structures from the Developmental Therapeutics Program at NCI/NIH [12] and is widely used in many existed researches on the graph. I select 697 graphs, each of which has no more than 10 nodes and turn the labeled graphs into simple graphs that are unlabeled, unweighted, and undirected.

(2) IMDB
The IMDB dataset is a collection of 1500 ego-networks of actors or actresses. An edge between two persons means that they are in the same movie. I pick 238 graphs from the IMDB dataset, each of which has more than 12 nodes and less than 16 nodes, to test the scalability and efficiency of my model. All nodes are unlabeled in IMDB.

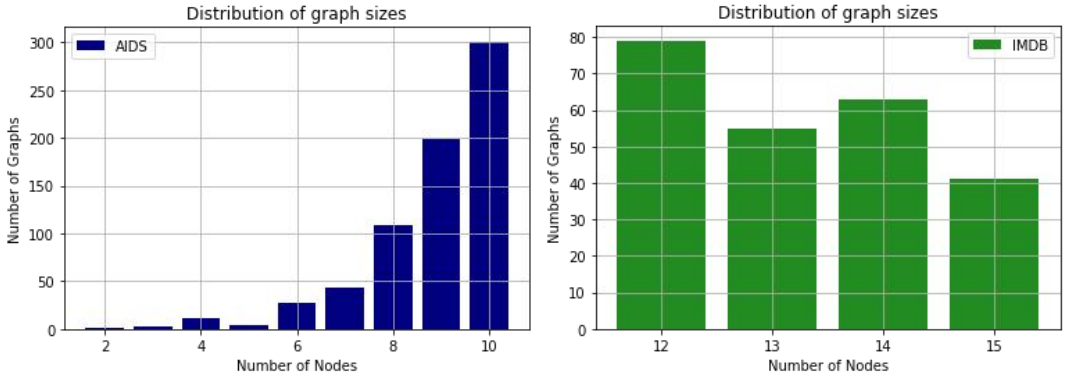

Figure 6 Distribution of graph sizes of AIDS and IMDB datasets

5.2 Data Preprocessing

For each dataset, I split dataset according to 4-fold cross validation methods before computing MSE, R squared, accuracy, and other metrics in the model evaluation stage. Specifically, I randomly partition the original dataset into 4 equal subsets. For 4 times, using one of them as a test set per time without repetition, the rest subsets are used for training. More specifically, 75 percent of the total dataset is used for training and 25 percent is used for testing per time.

Even graphs from AIDS are comparably small, performing GED on a significant number of graphs is still very challenging for my PC and HPC virtual machine. Therefore, I choose to use MCS to compute graph similarity scores. However, as the number of nodes and edges increases in the IMDB dataset, MCS becomes computationally infeasible for both my own

PC and HPC virtual machine. For traditional graph similarity metrics GED and MCS, no currently available algorithms are capable of giving reliable results of a pair of graphs with more than 16 nodes within acceptable time [12]. So, I only use graphs that have more than 12 nodes and less than 16 nodes to test in the IMDB dataset distinguishing from the selection in the AID dataset.

5.3 Baseline Methods

In this work, the baseline is based on the classical MCS algorithm. In the strategy one, I use my transformed MCS similarity metric as the target variable to train the model. In the strategy two, I simply use the number of common nodes obtained by MCS.

5.4 Parameter Settings

I conduct all the experiments on a single laptop with an Intel i7-7700HQ CPU and a 16G memory.

5.4.1 Feature Extraction

The output of the feature extraction step is an entire graph embedding vector each. The dimension of each graph embedding vector is 60 no matter what feature extraction algorithm is employed on AIDS in two strategies, for that reason that the dataset is vast, and my laptop can bear no larger dimension parameters.

When using the IMDB dataset in two strategies, I set different vector dimension parameters according to different feature extraction algorithms. It is shown in Table3.

Approaches	Dimension
FEATHER	100
GL2Vec	120
SF	100
FGSD	50
NetLSD	100

Table 3 Dimension Settings on IMDB

5.4.2 Strategy One

In the MLP regression model, the hidden layer size is set to 100. ReLU is used as an activation function. Adam is chosen to be the solver for weight optimization, and the maximum number of iterations is set to 200. Specifically, the selected solver Adam will not stop iterating until convergence or reaching this number of iterations. The batch size is set to be automatic.

5.4.3 Strategy Two

(1) KNN
In the KNN classification model, I set the number of neighbors to use to be 5. The distance metric is Minkowski distance.

(2) Decision Tree
I choose to use the CART decision tree and the entropy criterion is adopted to measure the quality of a split. I do not set the maximum depth of the tree.

(3) MLP
In the MLP classification model, the maximum number of iterations is set to 500. Other parameter settings are the same as the MLP regression model.

5.5 Evaluation Metrics

In strategy one, time, Mean Squared Error (MSE), R Squared, and Kendall Rank Correlation Coefficient are used to evaluate the model. Time is the wall time required to calculate the similarity score of two graphs. MSE measures the average squared difference between the model results and the actual scores. Since MSE does not have up and low limits, I also use R squared to return a score between 0 and 1. The larger figure means the better model generally. Kendall Rank Correlation Coefficient is also adopted to measure the ranking results.

In strategy two, I select time, accuracy, and confusion matrix to evaluate the models. Time is the runtime needed for each model to obtain the score for a pair of graphs. Accuracy is the proportion of predictions the model classified correctly. The confusion matrix is used to supplement the accuracy in the multiclass classification problem.

5.6 Results

5.6.1 Strategy One

(1) Effectiveness
The results on AIDS and IMDB datasets are demonstrated in Table 4 ad Table 5. r represents Kendall Rank Correlation Coefficient. SF reaches the best performance on all measures on two datasets while NetLSD performs worst on all metrics on two datasets. Interestingly, FEATHER and GL2Vec do not work well on the AIDS dataset. However, they both show excellent performance on the IMDB dataset that is a more complex dataset than the AIDS dataset. On the contrary, FGSD works better on the AIDS dataset than the IMDB dataset.

Embedding Methods	**MSE**	R^2	r
MSC(Baseline)	0	1	1
FEATHER	0.0061	0.51	0.42
GL2Vec	0.0115	0.24	0.40
SF	0.0026	0.83	0.66
FGSD	0.0032	0.82	0.65
NetLSD	0.0133	-0.0041	0.016

Table 4 Results on AIDS (strategy one)

Embedding Methods	**MSE**	R^2	r
MCS(Baseline)	0	1	1
FEATHER	0.0023	0.89	0.80
GL2Vec	0.0081	0.63	0.67
SF	0.0021	0.89	0.81

FGSD	0.0128	0.63	0.72
NetLSD	0.0158	0.23	0.40

Table 5 Results on IMDB (strategy one)

(2) Efficiency

Since calculating MCS is exceptionally time-consuming, I randomly choose a small sample from each dataset to compute the average time for the MCS of a pair of graphs individually, then multiply the average value by the number of the test dataset. The runtime comparison of the two datasets is demonstrated in Table 6 and Figure 7. The time measured for my proposed models does not include the time of the graph embedding stage. Since the selected IMDB dataset is comparably small, there is only a slight difference in all proposed models referring to time on this dataset. They all show excellent efficiency on IMDB. For instance, the SF based model is 382241 times faster than the exact MCS algorithm, and FGSD based model is 439888 times faster compared with original MCS on AIDS. Moreover, the SF and FGSD based models perform the best and the second-best results on the AIDS dataset. They significantly save more time than the other three models. Specifically, the SF based model is 68996 times faster than the exact MCS algorithm on AIDS while the figure for FEATHER is 10822.

	Time on AIDS(s)	**Time on IMDB(s)**
MCS(Baseline)	23527.9387	18959.1922
FEATHER	2.174	0.0391
GL2Vec	3.961	0.0466
SF	0.341	0.0496
FGSD	0.422	0.0431
NetLSD	3.513	0.0436

Table 6 Time evaluation (strategy one)

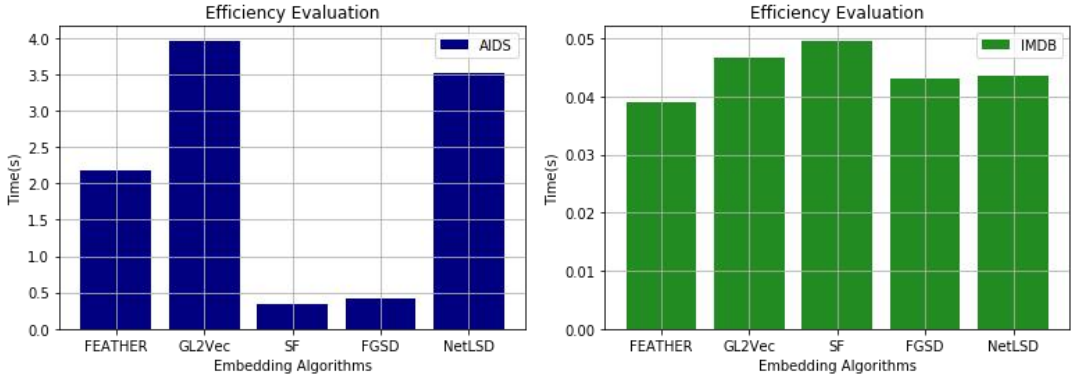

Figure 7 Time evaluation (strategy one)

In conclusion, it is reasonable to use my proposed flamework to speed up graph similarity computation, which is especially true for using SF based approach. It demonstrates the best efficiency on large graphs while preserving the lowest MSE, highest R Squared, and high Kendall Rank Correlation Coefficient.

5.6.2 Strategy Two

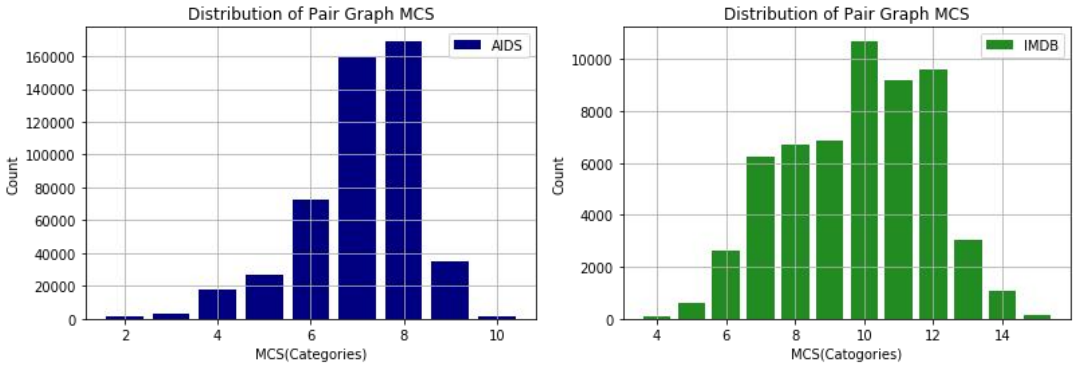

Figure 8 Distribution of MCS(Categories)

Figure 8 demonstrates that both AIDS and IMDB are imbalanced. Specifically, the classes (MCS) are not presented equally. Therefore, only using accuracy to measure the performance is not enough. I also use the confusion matrix as a supplement. The confusion matrix is normalized by row.

(1) Effectiveness
The results of AIDS and IMDB datasets are demonstrated in Table 7.

Models	Accuracy on AIDS	Accuracy on IMDB
MCS(Baseline)	1	1
FEATHER-KNN	0.86	0.84
FEATHER-DT	0.88	0.86
FEATHER-MLP	0.68	0.83
GL2Vec-KNN	Computationally infeasible	0.76
GL2Vec-DT	0.40	0.63
GL2Vec-MLP	0.44	0.67
SF-KNN	0.90	0.82
SF-DT	0.98	0.88
SF-MLP	0.79	0.81
FGSD-KNN	0.86	0.80
FGSD-DT	0.87	0.81
FGSD-MLP	0.81	0.77
NetLSD-KNN	0.37	0.79
NetLSD -DT	0.35	0.84
NetLSD -MLP	0.39	0.36

Table 7 Results on AIDS and IMDB (strategy two)

From the table above, we can see that FGSD and SF based models significantly achieve higher accuracy scores than other models on AIDS. FEATHER and NetLSD based models also demonstrate good performance. When we carefully look at their confusion matrixes, we see that almost all classes reach similar high accuracy scores. Only a few classes that have fewer samples respectively are indeed obtaining a slightly lower score (0.62 for class 3 in FGSD-KNN(AIDS), 0.65 for class 9 in SF-MLP(AIDS), and 0.64 for class 13 in SF-MLP(IMDB)).

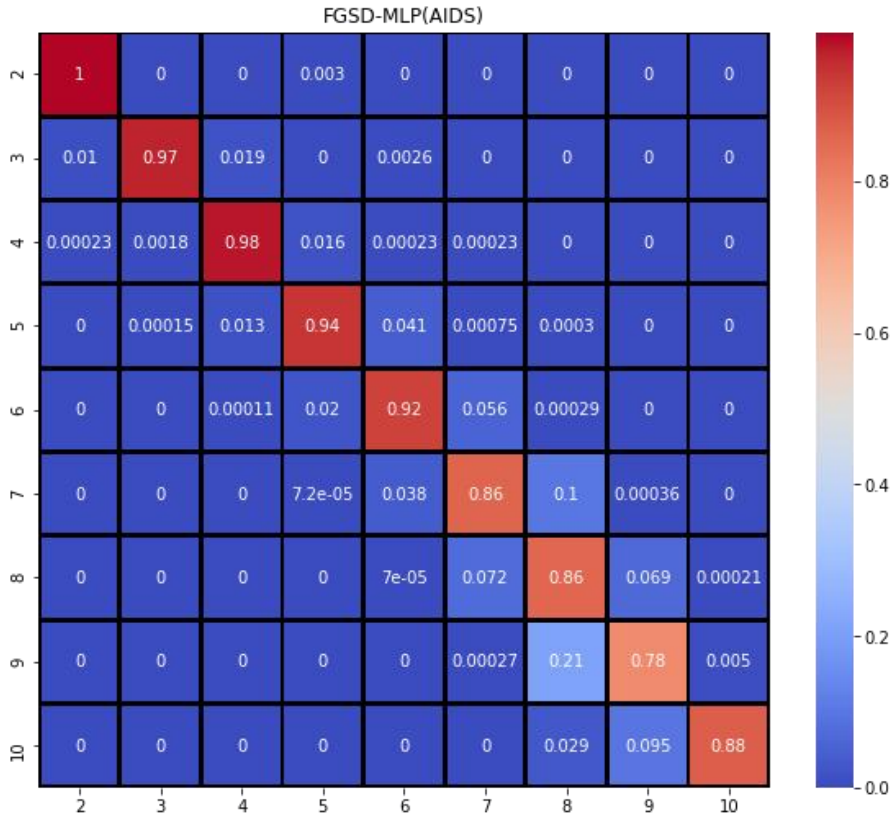

Figure 9 Confusion Matrix of FGSD-MLP(AIDS)

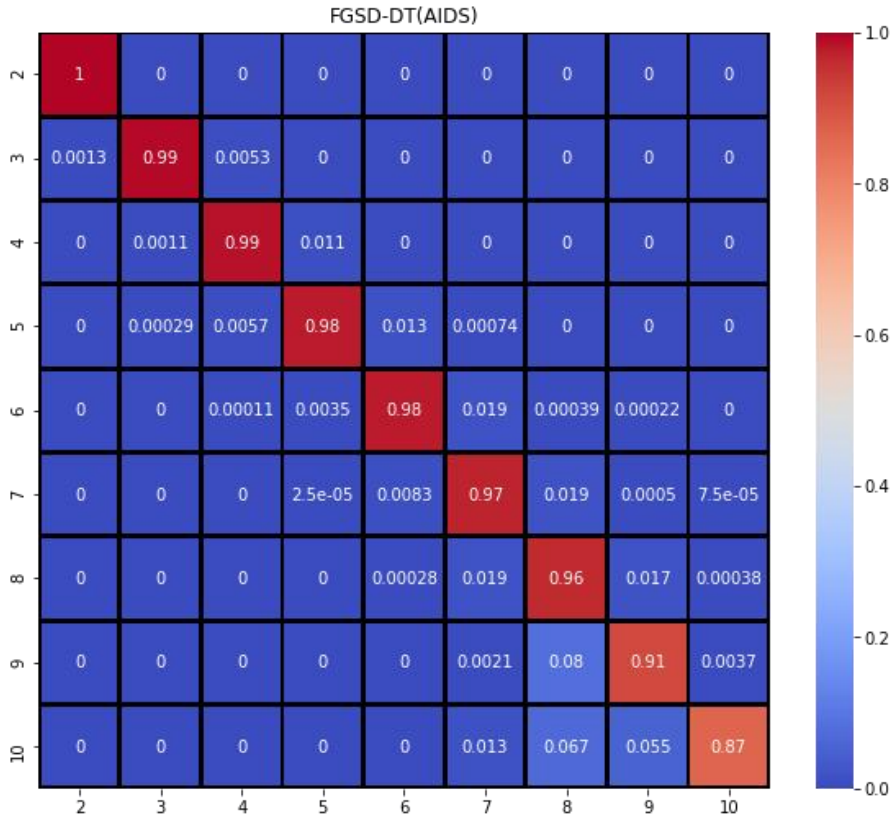

Figure 10 Confusion Matrix of FGSD-DT(AIDS)

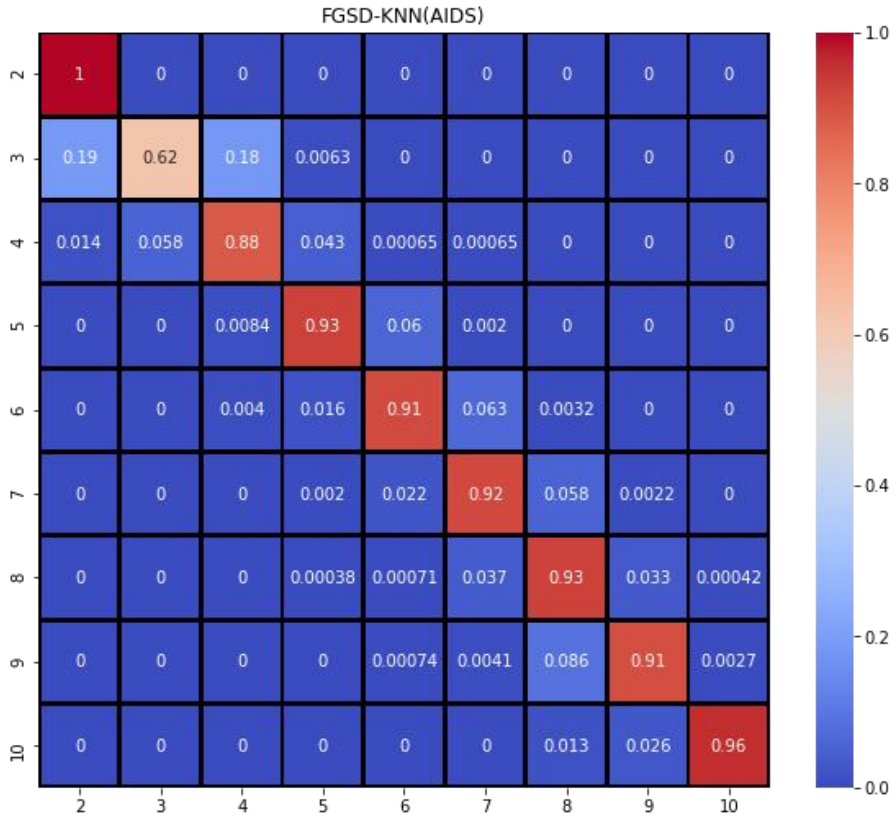

Figure 11 Confusion Matrix of FGSD-KNN(AIDS)

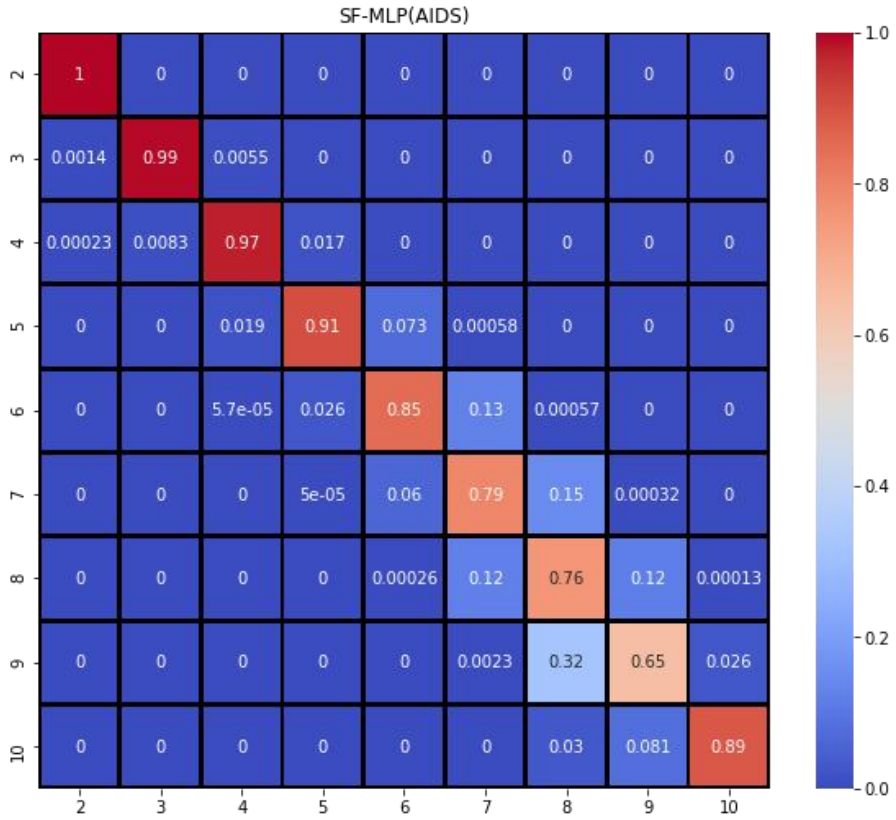

Figure 12 Confusion Matrix of SF-MLP(AIDS)

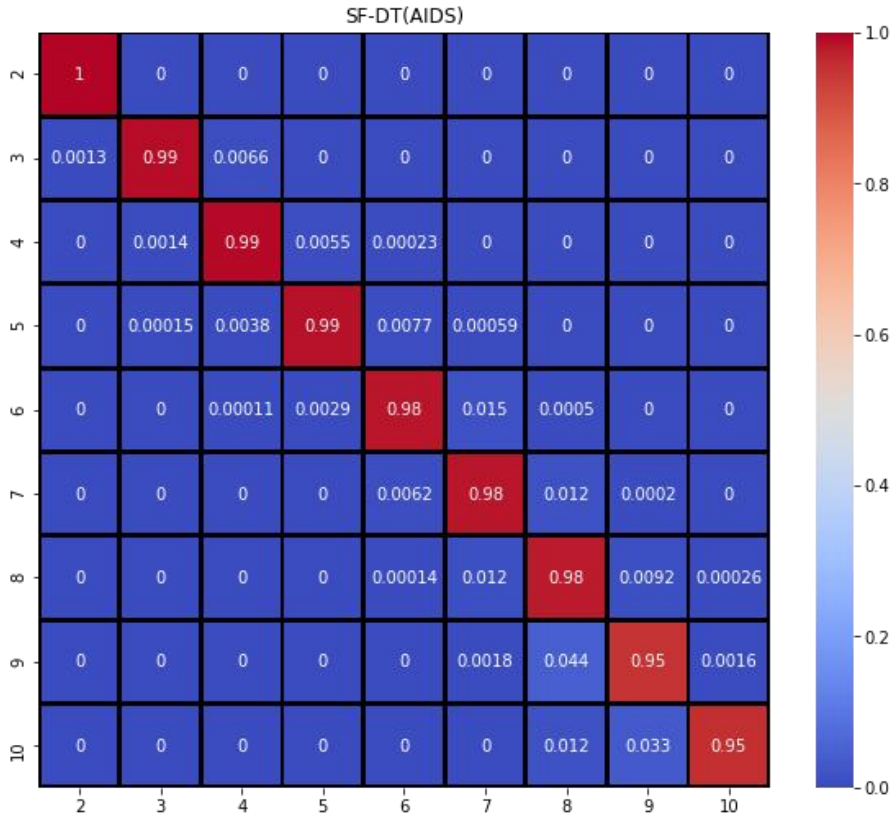

Figure 13 Confusion Matrix of SF-DT(AIDS)

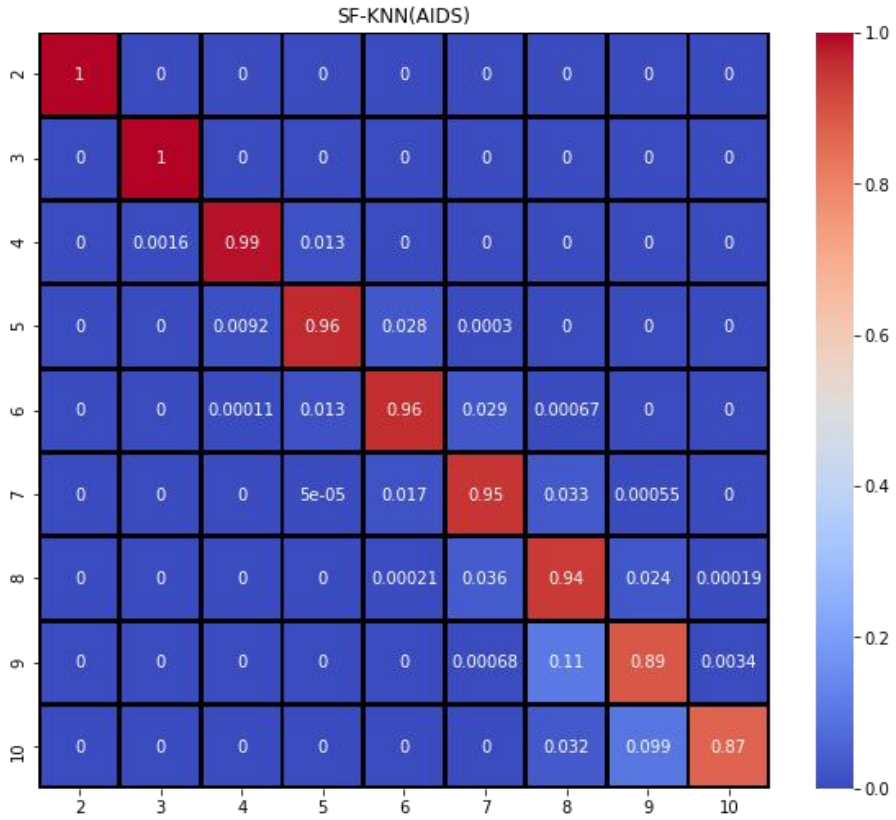

Figure 14 Confusion Matrix of SF-KNN(AIDS)

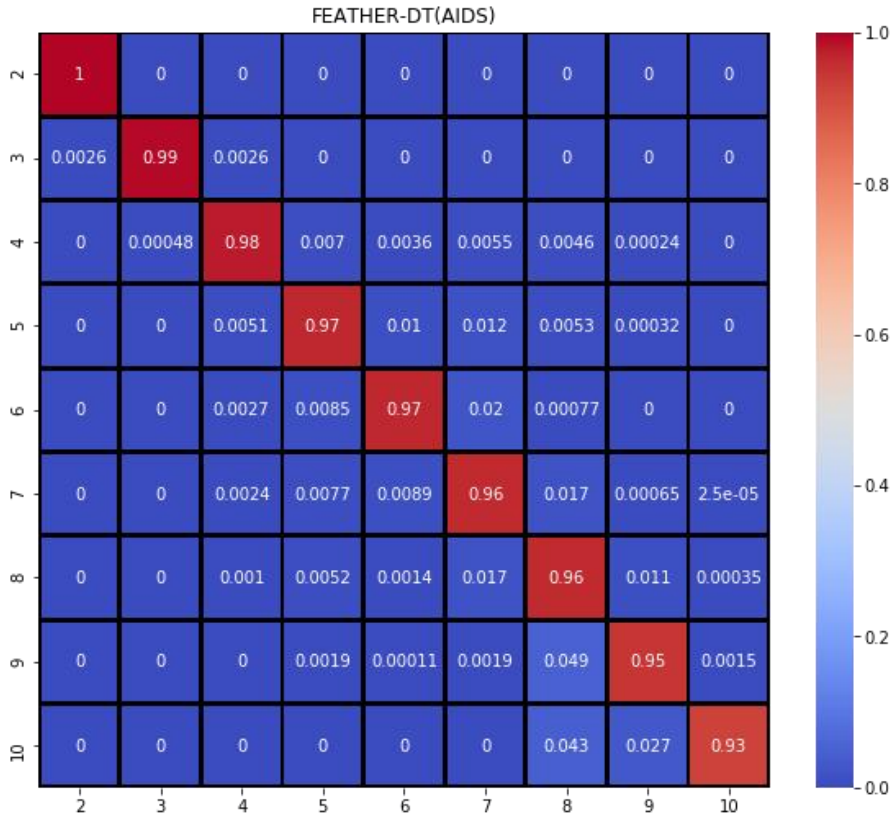

Figure 15 Confusion Matrix of FEATHER-DT(AIDS)

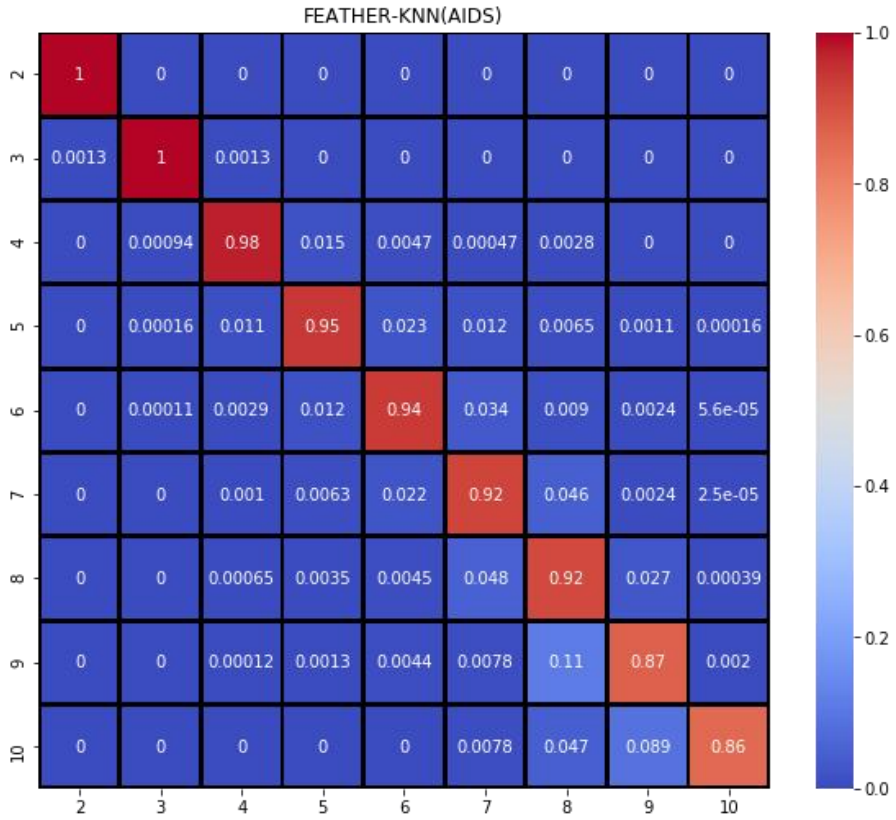

Figure 16 Confusion Matrix of FEATHER-KNN(AIDS)

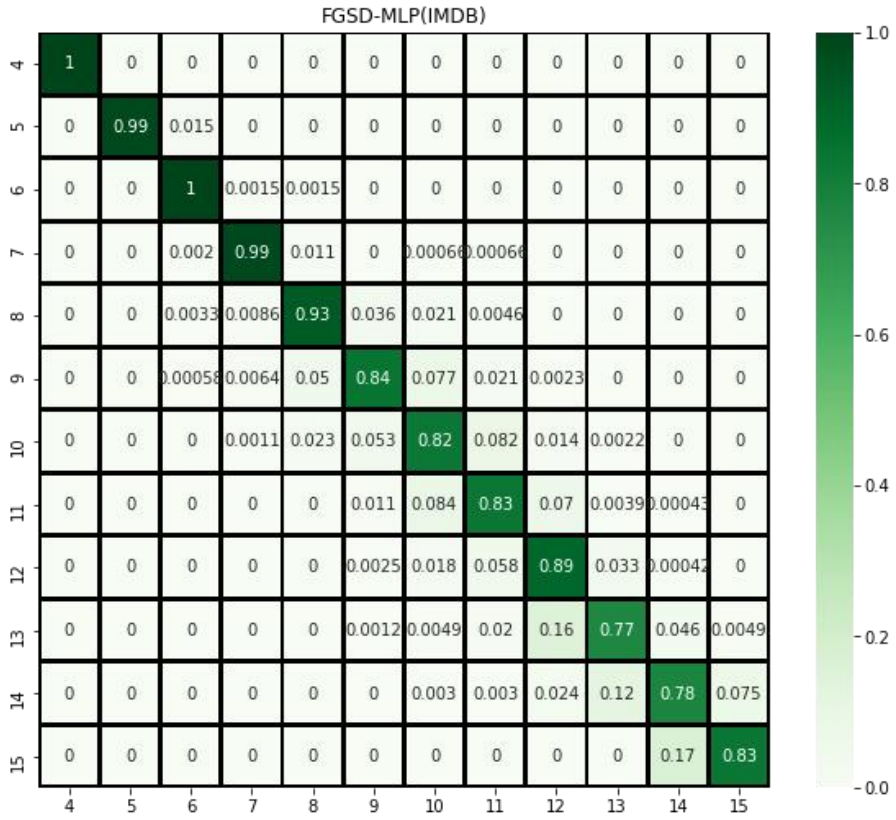

Figure 17 Confusion Matrix of FGSD-MLP(IMDB)

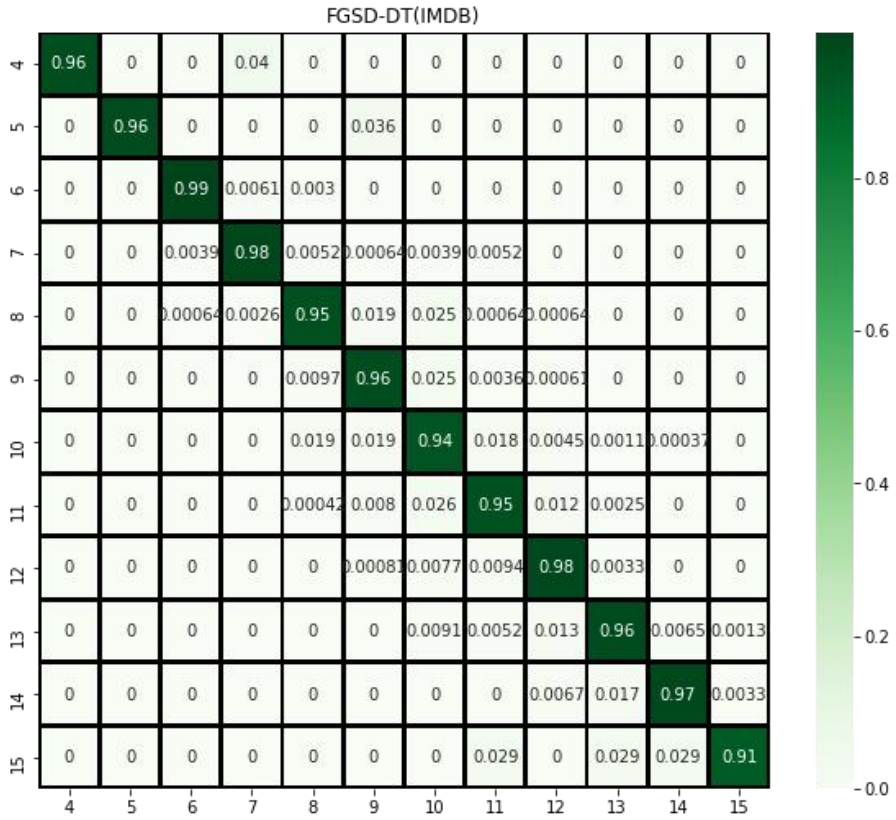

Figure 18 Confusion Matrix of FGSD-DT(IMDB)

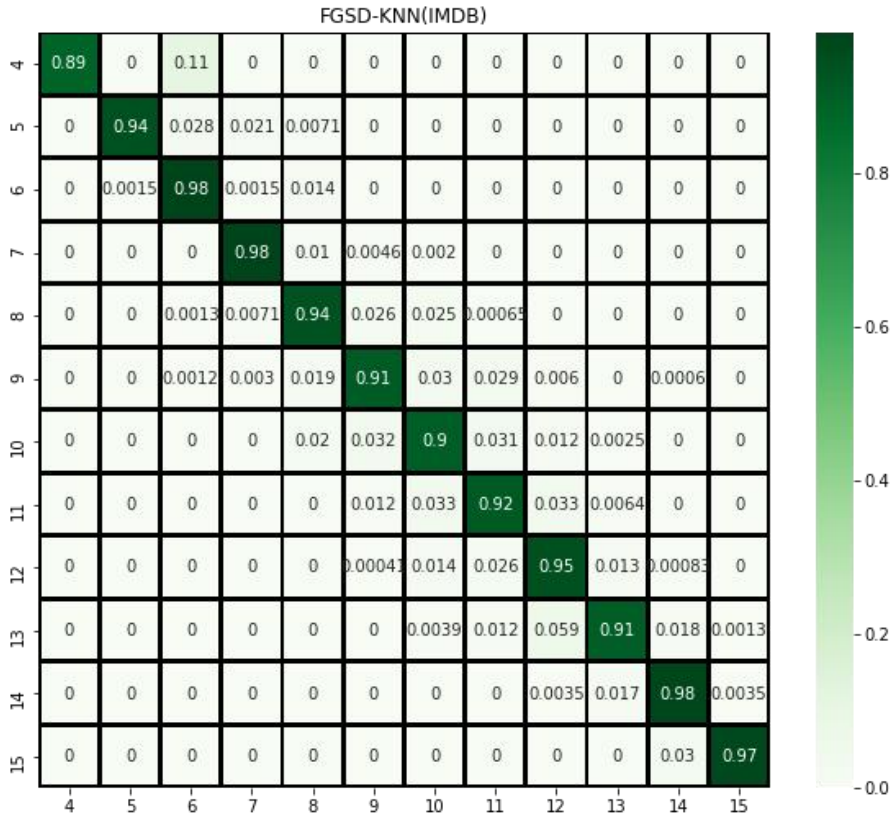

Figure 19 Confusion Matrix of FGSD-KNN(IMDB)

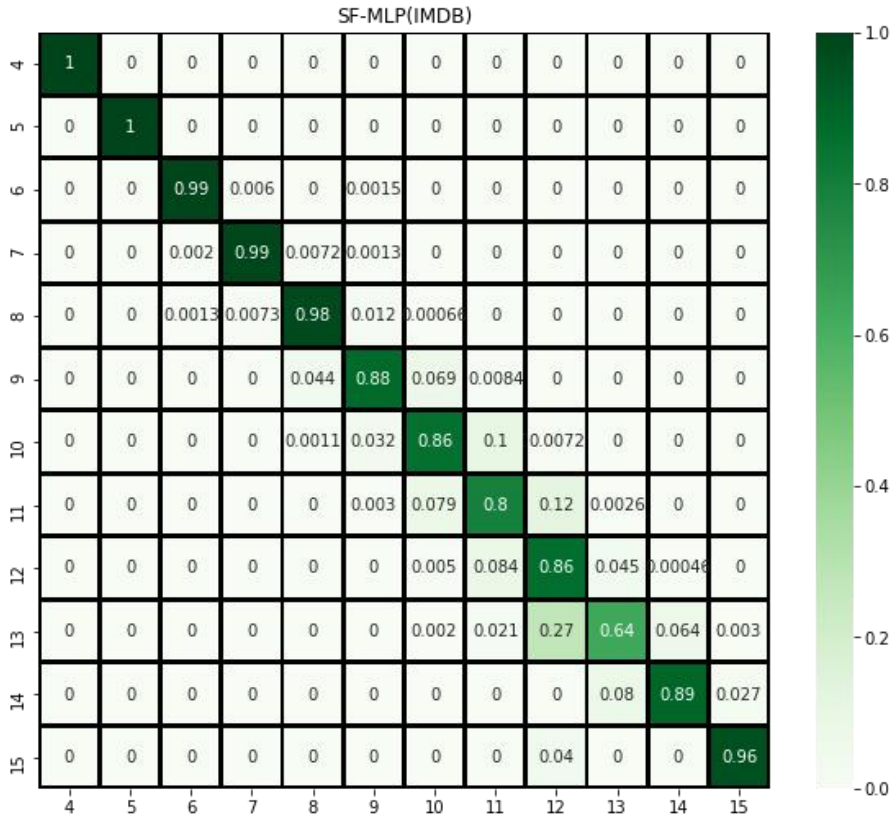

Figure 20 Confusion Matrix of SF-MLP(IMDB)

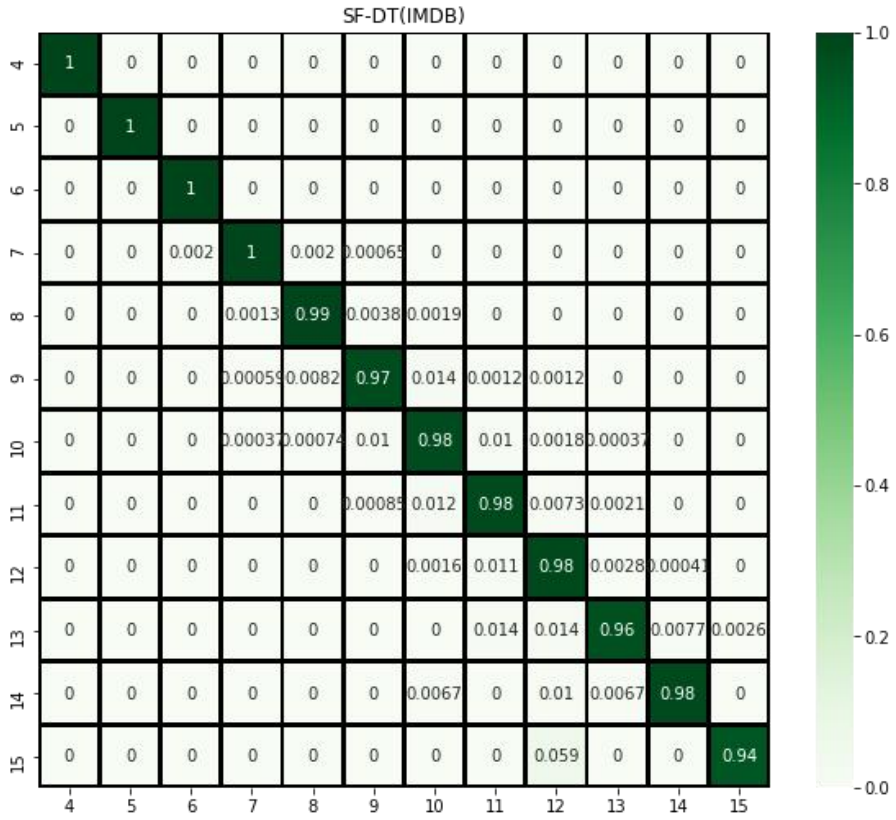

Figure 21 Confusion Matrix of SF-DT(IMDB)

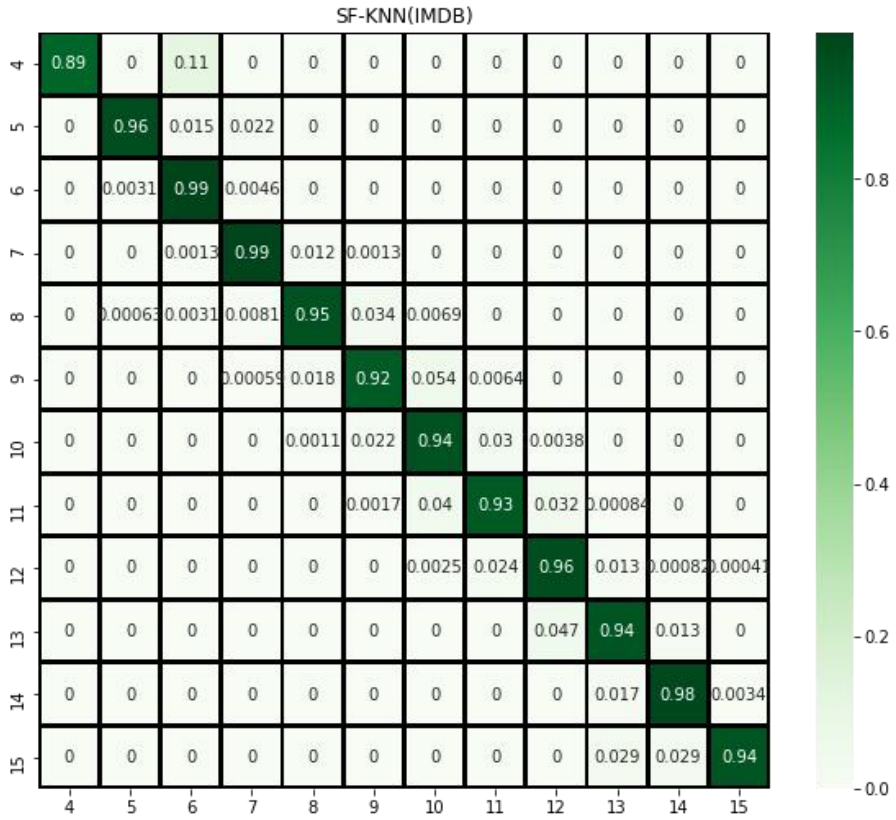

Figure 22 Confusion Matrix of SF-KNN(IMDB)

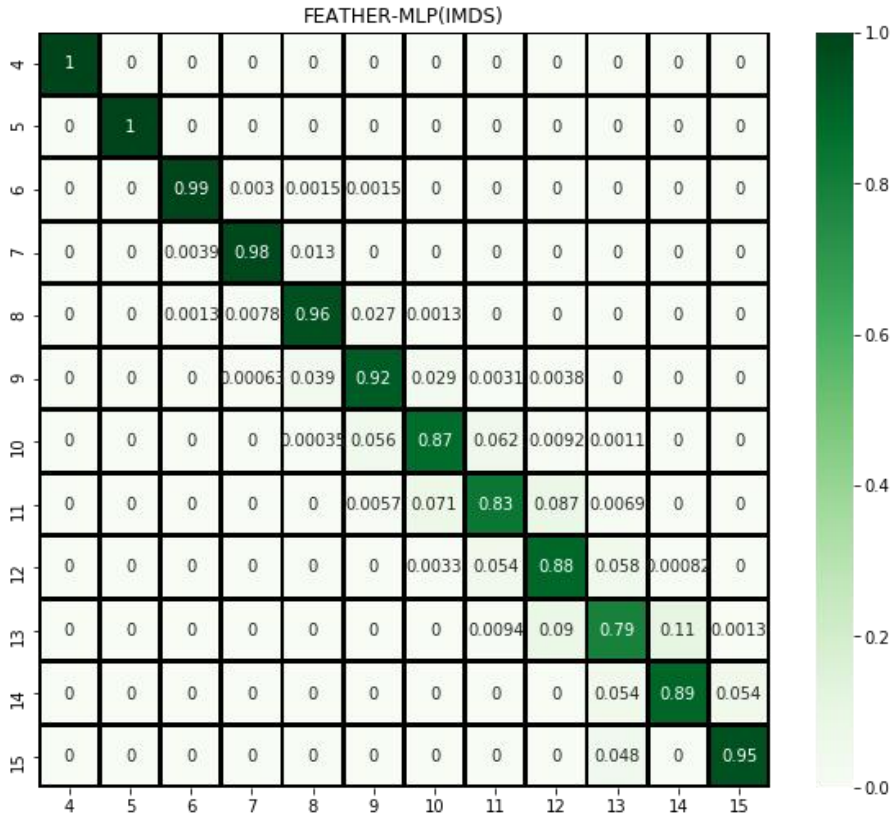

Figure 23 Confusion Matrix of FEATHER-MLP(IMDB)

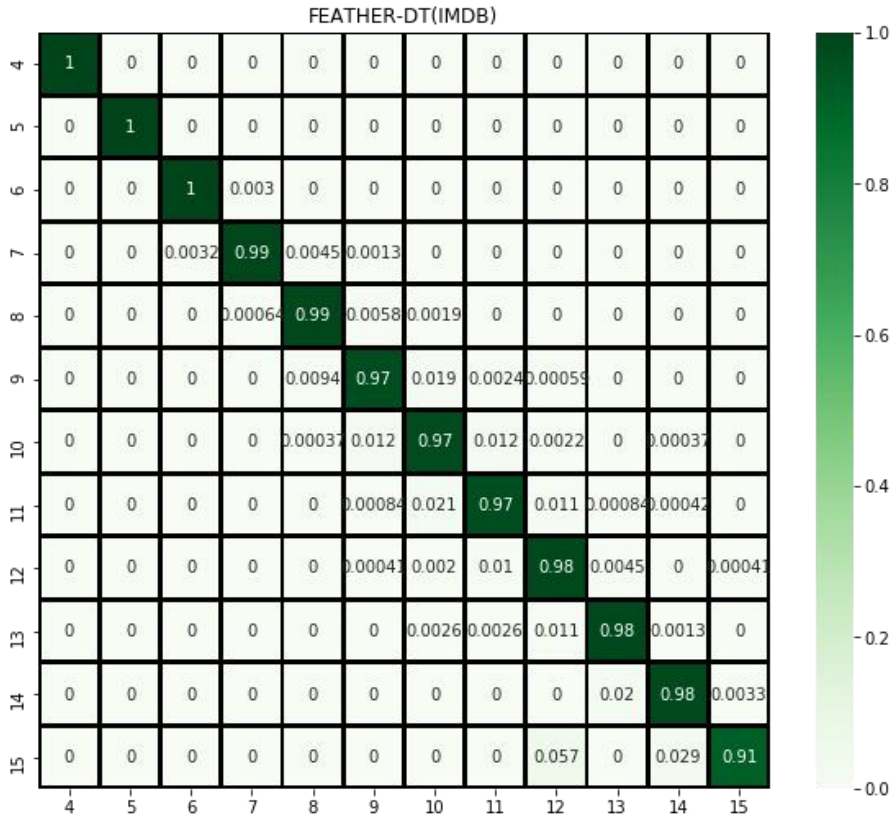

Figure 24 Confusion Matrix of FEATHER-DT(IMDB)

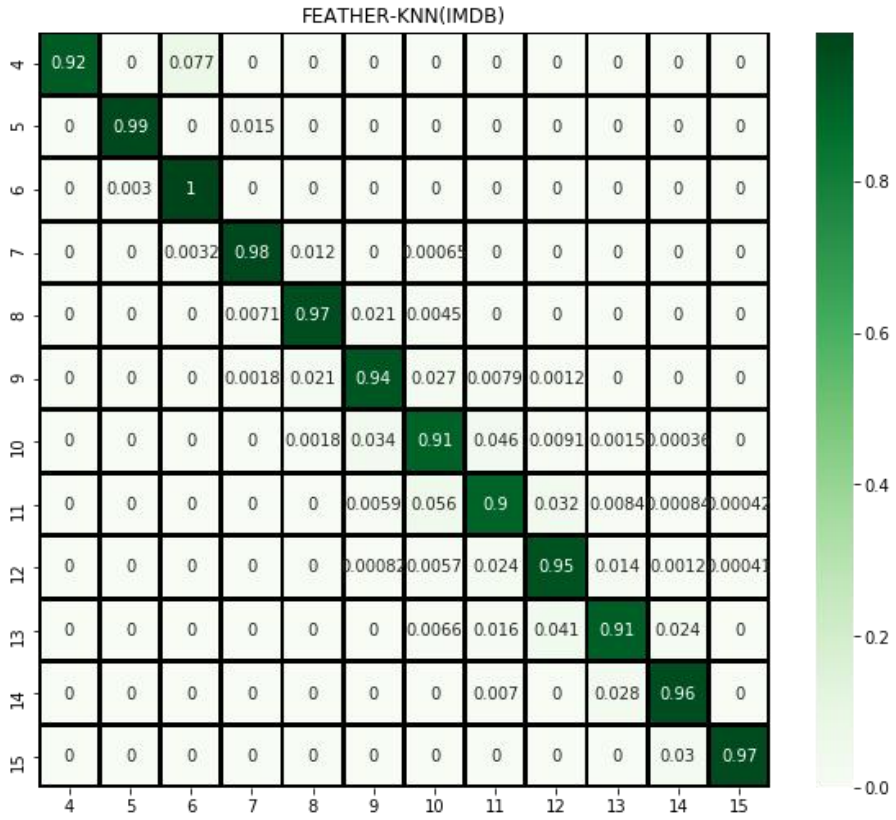

Figure 25 Confusion Matrix of FEATHER-KNN(IMDB)

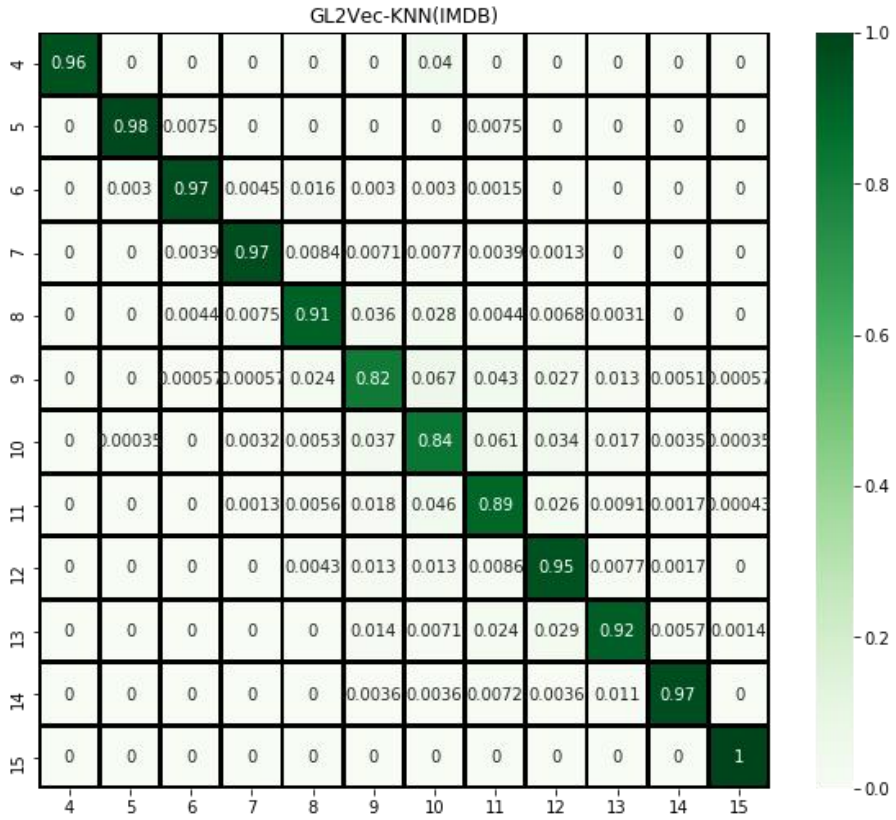

Figure 26 Confusion Matrix of GL2Vec-KNN(IMDB)

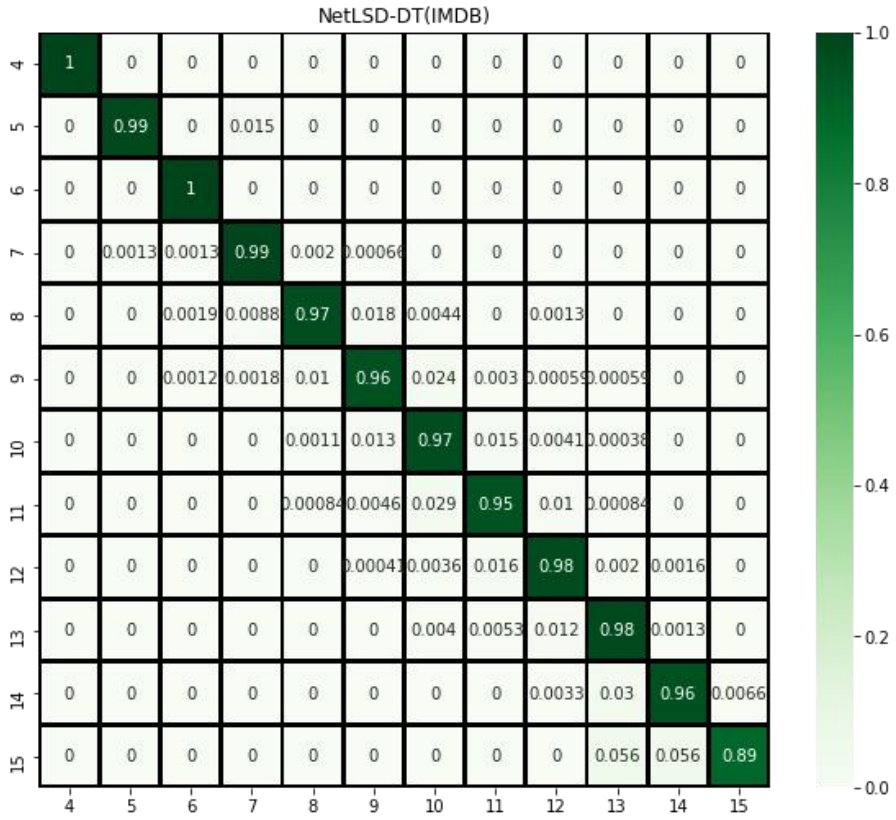

Figure 27 Confusion Matrix of NetLSD-DT(IMDB)

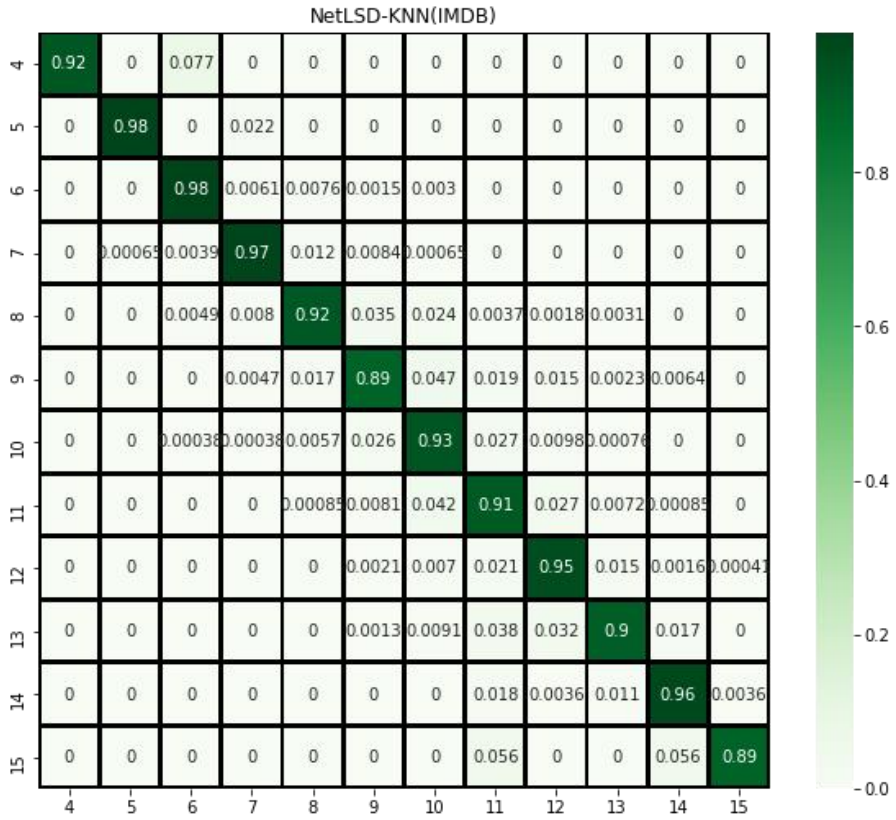

Figure 28 Confusion Matrix of NetLSD-KNN(IMDB)

(2) Efficiency

The prediction time comparison of the two datasets is demonstrated in Table 8 and Figure 29.

Models	Time on AIDS(s)	Time on IMDB(s)
MCS(Baseline)	244631.8242	18967.4739
FEATHER-KNN	77.3871	3.5382
FEATHER -DT	0.1003	0.0313
FEATHER-MLP	0.4094	0.0682
GL2Vec-KNN	Computationally infeasible	7.5472
GL2Vec -DT	0.1326	0.0201
GL2Vec-MLP	0.3459	0.0631
SF-KNN	78.1183	6.2201
SF -DT	0.0897	0.0156
SF-MLP	0.4418	0.0737
FGSD-KNN	593.0306	2.8504
FGSD-DT	0.1003	0.0106
FGSD-MLP	0.4748	0.0587
NetLSD-KNN	20.8463	2.7790
NetLSD -DT	0.1273	0.0175
NetLSD -MLP	0.3936	0.0526

Table 8 Time evaluation (strategy two)

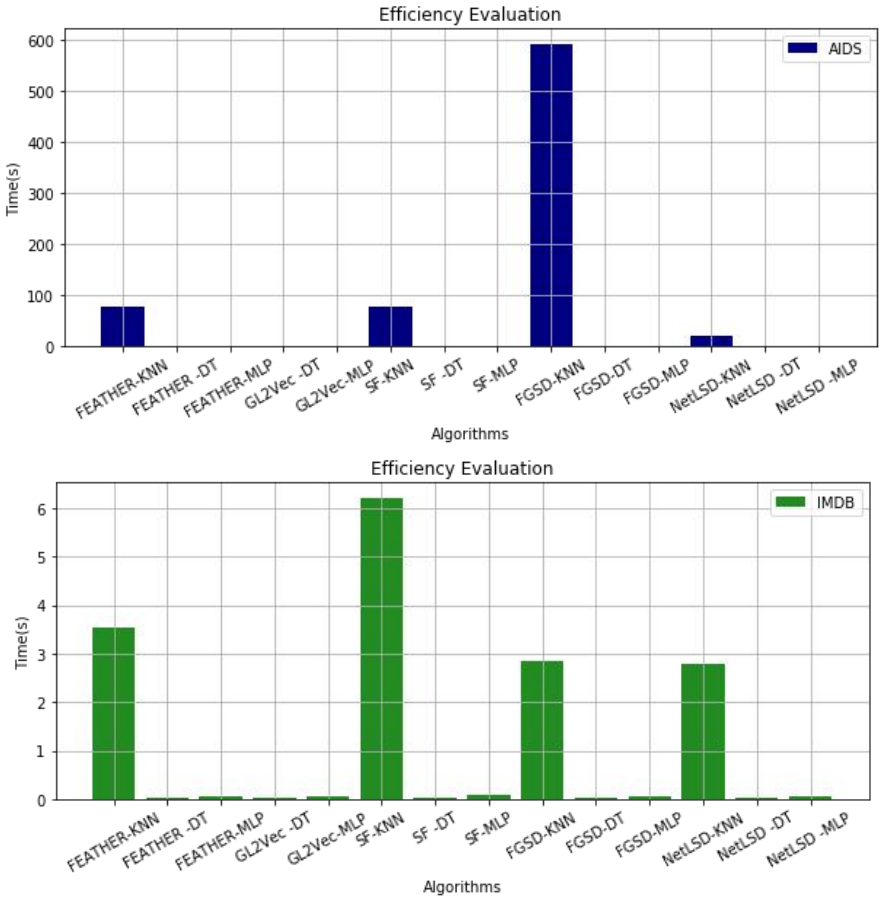

Figure 29 Time evaluation (strategy two)

The comparison of prediction time on the two datasets is shown in Table 8 and Figure 29. All approaches except GL2Vec-KNN have shown excellent efficiency on both datasets compared to the exact MCS algorithm. Even the slowest FGSD-KNN is 412 times faster than the precise MCS. Among them, the decision tree-based approached generally saving more time in comparison with other proposed approaches while the KNN based algorithms always consume more time. For instance, the SF-DT method is 870 times faster than the SF-KNN and also obtain higher accuracy scores than SF-KNN on AIDS. Moreover, the SF-DT approach achieves the highest accuracies (0.98 on AIDS and 0.88 on IMDB) and costs the least and the second least time on two datasets (0.0897(s) on AIDS and 0.0156(s) on IMDB), which performs the best among all proposes approaches. Apart from SF-DT, SF-KNN, SF-MLP, FGSD-KNN, FGSD-DT, FGSD-MLP, FEATHER-KNN, FEATHER-DT also show excellent performance on both datasets.

In conclusion, it is reasonable to use my proposed framework as a fast method to calculate graph similarity. It works well on both strategies.

6. Conclusion

6.1 Overview

In this work, a framework combining many state-of-the-art graph embedding methods and simple machine learning algorithms is proposed to compute graph similarity efficiently. The framework is designed very flexibly. The core idea is to make good use of a large number of existed works in graph embedding and machine learning areas prompting the development of traditional similarity computation. Some combinations of my proposed framework show extremely satisfactory performance. Compared to the calculation of classical similarity metric MCS, my proposed flamework runs very efficiently while preserving good accuracy.

6.2 Limitation and Future Work

Though my proposed framework has shown effectiveness and efficiency, there are several limitations as follows.

(1) My proposed model can only measure simple graphs that are unlabelled, undirected, and unweighted. Many graph applications are based on complex graphs that exist some attributes on nodes or edges. For example, nodes of chemical compounds are labeled, and edges of social networks can be weighted and directed. Therefore, it is beneficial to extend my proposed framework.

(2) I only pick several entire graph embeddings methods and machine learning algorithms. It will be fascinating to combine more techniques, such as node embedding methods and graph attention mechanism, boosting, and bagging.

(3) Since computing the exact MCS or GED for a pair of large and complex graphs is impossible, it will be promising to train the model on the approximate MCS or GED algorithms between large graphs.

(4) My framework is designed to be adaptive to all sorts of similarity metrics theoretically. In this paper, I only use MCS to test my framework. In the future, I can use GED or other classical similarity metrics to test.

(5) My proposed framework is the combination of many state-of-the-art algorithms, and it takes some time to complete feature extraction and model training stages. Thus, it will be very promising and convenient to design a distributed version in the future.

7. References

[1] Boldi, Paolo & Rosa, Marco & Vigna, Sebastiano. (2013). Robustness of social and web graphs to node removal. Social Network Analysis and Mining. 3. 10.1007/s13278-013-0096-x.
[2] A. Theocharidis, S. Van Dongen, A. Enright, and T. Freeman, “Network visualization and analysis of gene expression data using biolayout express3d,” Nature protocols, vol. 4, pp. 1535–1550, 2009.
[3] Syan, Sabrina & Smith, Mara & Frey, Benicio & Remtulla, Raheem & Kapczinski, Flavio & Hall, Geoffrey & Minuzzi, Luciano. (2018). Resting-state functional connectivity in individuals with bipolar disorder during clinical remission: A systematic review. Journal of psychiatry & neuroscience : JPN. 43. 298-316. 10.1503/jpn.170175.
[4] Fang, Minghong & Yang, Guolei & Gong, Neil & Liu, Jia. (2018). Poisoning Attacks to Graph-Based Recommender Systems. 10.1145/3274694.3274706.
[5] YUAN, Bin-tao & PAN, Zu-lie & SHI, Fan. (2018). A Review on Network Attack Graph Technology. DEStech Transactions on Engineering and Technology Research. 10.12783/dtetr/ecar2018/26351.
[6] Markovich, Natalia & Ryzhov, Maxim. (2019). Random Graph Node Classification by Extremal Index of PageRank. 10.1007/978-3-030-36625-4_34.
[7] Wang, Haiyan & Wang, Feng & Xu, Kuai. (2020). Clustering of Online Social Network Graphs. 10.1007/978-3-030-38852-2_4.
[8] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, “Scalable graph embedding for asymmetric proximity,” in AAAI, 2017, pp. 2942–2948.
[9] Choi, Ryan & Chung, Chin-Wan. (2015). Efficient processing of graph similarity search. World Wide Web. 18. 633-659. 10.1007/s11280-014-0274-4.
[10] Zhou, Yujing & Pei, Yang & He, Yuanye & Mo, Jingjie & Jiong, Wang & Gao, Neng. (2019). Dynamic Graph Link Prediction by Semantic Evolution. 1-6. 10.1109/ICC.2019.8761688.
[11] Chen, Xiaoyang & Huo, Hongwei & Huan, Jun & Vitter, Jeffrey. (2017). Fast Computation of Graph Edit Distance.
[12] Bai, Yunsheng & Ding, Hao & Bian, Song & Chen, Ting & Sun, Yizhou & Wang, Wei. (2019). SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. WSDM '19: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. 384-392. 10.1145/3289600.3290967.
[13] Ma, Guixiang & Ahmed, Nesreen & Willke, Theodore & Yu, Philip. (2019). Deep Graph Similarity Learning: A Survey.
[14] Goyal, Palash & Ferrara, Emilio. (2017). Graph Embedding Techniques, Applications, and Performance: A Survey. Knowledge-Based Systems. 10.1016/j.knosys.2018.03.022.
[15] Cai, Hongyun & Zheng, Vincent & Chang, Kevin. (2017). A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. IEEE Transactions on Knowledge and Data Engineering. 10.1109/TKDE.2018.2807452.
[16] Rozemberczki, Benedek & Sarkar, Rik. (2020). Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models.
[17] Chen, Hong & Koga, Hisashi. (2019). GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features. 10.1007/978-3-030-36718-3_1.
[18] Lara, Nathan & Pineau, Edouard. (2018). A Simple Baseline Algorithm for Graph Classification.
[19] Verma, Saurabh & Zhang, Zhi-Li. (2017). Hunt For The Unique, Stable, Sparse And Fast Feature Learning On Graphs.

[20] Narayanan, Annamalai & Mahinthan, Chandramohan & Venkatesan, Rajasekar & Chen, Lihui & Liu, Yang & Jaiswal, Shantanu. (2017). graph2vec: Learning Distributed Representations of Graphs.
[21] Xu, Haoyan & Chen, Runjian & Bai, Yunsheng & Feng, Jie & Duan, Ziheng & Luo, Ke & Sun, Yizhou & Wang, Wei. (2020). Hierarchical and Fast Graph Similarity Computation via Graph Coarsening and Deep Graph Learning.
[22] Köbler, Johannes & Schöning, Uwe & Torán, Jacobo. (1993). The Graph Isomorphism Problem: its structural complexity. 10.1007/978-1-4612-0333-9.
[23] Sanfeliu, A. and K.-S. Fu, A Distance measure between attributed relational graphs for pattern recognition.IEEE transactions on systems, man, and cybernetics, 1983. 13(3): p. 353-362
[24] https://www.mygreatlearning.com/blog/cross-validation/
[25]https://corporatefinanceinstitute.com/resources/knowledge/other/rsquared/#:~:text=RSquared%20%28R%C2%B2%20or%20the%20coefficient%20of%20determination%29%20is,fit%20the%20regression%20model%20%28the%20goodness%20of%20fit%29.
[26] Witten, Ian & Frank, E. & Hall, M.A. & Pal, C.J.. (2016). Data Mining: Practical Machine Learning Tools and Techniques.
[27] Goyal, Palash & Ferrara, Emilio. (2017). Graph Embedding Techniques, Applications, and Performance: A Survey. Knowledge-Based Systems. 10.1016/j.knosys.2018.03.022.
[28] Cui, Peng & Wang, Xiao & Pei, Jian & Zhu, Wenwu. (2017). A Survey on Network Embedding. IEEE Transactions on Knowledge and Data Engineering. PP. 10.1109/TKDE.2018.2849727.
[29] Hamilton, William & Ying, Rex & Leskovec, Jure. (2017). Representation Learning on Graphs: Methods and Applications.
[30] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (5500) (2000) 2323–2326.
[31] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in: NIPS, Vol. 14, 2001, pp. 585–591.
[32] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A. J.17 Smola, Distributed large-scale natural graph factorization, in: Proceedings of the 22nd international conference on World Wide Web, ACM, 2013, pp. 37–48.
[33] S. Cao, W. Lu, Q. Xu, Grarep: Learning graph representations with global structural information, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 891–900.
[34] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: Proc. of ACM SIGKDD, 2016, pp. 1105–
[35] Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.
[36] Mikolov, Tomas & Sutskever, Ilya & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems. 26.
[37] Perozzi, Bryan & Al-Rfou, Rami & Skiena, Steven. (2014). DeepWalk: Online Learning of Social Representations. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 10.1145/2623330.2623732.
[38] Tang, Jian & Qu, Meng & Wang, Mingzhe & Zhang, Ming & Yan, Jun & Mei, Qiaozhu. (2015). LINE: Large-scale Information Network Embedding. Line: Large-Scale Information Network Embedding. 10.1145/2736277.2741093.

[39] Grover, Aditya & Leskovec, Jure. (2016). node2vec: Scalable Feature Learning for Networks. KDD : proceedings. International Conference on Knowledge Discovery & Data Mining. 2016. 855-864. 10.1145/2939672.2939754.
[40] Wang, Jizhe & Huang, Pipei & Zhao, Huan & Zhang, Zhibo & Zhao, Binqiang & Lee, Dik. (2018). Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. 839-848. 10.1145/3219819.3219869.
[41] Le, Quoc & Mikolov, Tomas. (2014). Distributed Representations of Sentences and Documents. 31st International Conference on Machine Learning, ICML 2014. 4.
[42] Blumenthal, David & Boria, Nicolas & Gamper, Johann & Bougleux, Sébastien & Brun, Luc. (2019). Comparing heuristics for graph edit distance computation. The VLDB Journal. 10.1007/s00778-019-00544-1.
[43] K. Riesen, S. Fankhauser, and H. Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In MLG, pages 21–24, 2007.
[44] Z. Abu-Aisheh, R. Raveaux, J. Y. Ramel, and P. Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In ICPRAM, pages 271–278, 2015.
[45] K. Gouda and M. Hassaan. CSI GED: An efficient approach for graph edit similarity computation. In ICDE, pages 256–275, 2016.
[46] Marcelli, Andrea & Quer, Stefano & Squillero, Giovanni. (2019). The Maximum Common Subgraph Problem: A Portfolio Approach.
[47] Quer, Stefano & Marcelli, Andrea & Squillero, Giovanni. (2020). The Maximum Common Subgraph Problem: A Parallel and Multi-Engine Approach. Computation. 8. 48. 10.3390/computation8020048.
[48] Chen, Alan & Elhajj, Ahmed & Gao, Shang & Sarhan, Abdullah & Afra, Salim & Kassem, Ahmad. (2015). Approximating the maximum common subgraph isomorphism problem with a weighted graph. Knowledge-Based Systems. 85. 10.1016/j.knosys.2015.05.012.
[49] Valenti, Cesare. (2019). A genetic approach to the maximum common subgraph problem. CompSysTech '19: Proceedings of the 20th International Conference on Computer Systems and Technologies. 98-104. 10.1145/3345252.3345272.
[50] McCreesh, Ciaran & Prosser, Patrick & Trimble, James. (2017). A Partitioning Algorithm for Maximum Common Subgraph Problems. 712-719. 10.24963/ijcai.2017/99.
[51] Rozemberczki, Benedek & Kiss, Oliver & Sarkar, Rik. (2020). An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs.
[52] Tsitsulin, Anton & Mottin, Davide & Karras, Panagiotis & Bronstein, Alex & Müller, Emmanuel. (2018). NetLSD: Hearing the Shape of a Graph. 10.1145/3219819.3219991.
[53] Bunke, H.. (1997). Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recogn. Lett. 18(8), 689-694. Pattern Recognition Letters. 18. 689-694. 10.1016/S0167-8655(97)00060-3.
[54] https://towardsdatascience.com/kendall-rank-correlation-explained-dee01d99c535
[55] https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient
[56] https://thenextweb.com/neural-basics/2020/05/27/everything-you-need-to-know-about-artificial-neural-networks/
[57] https://zhuanlan.zhihu.com/p/101224584
[58] https://towardsdatascience.com/regression-an-explanation-of-regression-metrics-and-what-can-go-wrong-a39a9793d914
[59] https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/#:~:text=Kendall%20rank%20correlation%3A%20Kendall%20rank%20correlation%20is%20a,measures%20the%20strength%20of%20dependence%20between%20two%20variables.

[60] https://machinelearningmastery.com/introduction-to-matrix-decompositions-for-machine-learning/
[61] https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007#:~:text=DeepWalk%20uses%20random%20walks%20to%20produce%20embeddings.%20The,Sampling%3A%20A%20graph%20is%20sampled%20with%20random%20walks.

8. Appendices

```
Readme.txt
Statement.docx

├─7.21-IMDBMulti
│   AIDS_Gra_Seq.csv
│   Compute_MCS_and_MCS_TIME.py
│   Data_Visualization.py
│   IMDB-Strategy1.py
│   IMDB-Strategy2.py
│   strategy2_time.csv
│
│   ├─dataset
│   ├─HTML Version for codes with Results
│   │    Compute_MCS_and_MCS_TIME.html
│   │    Data_Visualization.html
│   │    IMDB-Strategy1.html
│   │    IMDB-Strategy2.html
│   │
│   ├─mcs_result
│   └─mcs_result_discrete
└─7.28-AIDS
    AIDS-Strategy1.py
    AIDS-Strategy2.py
    Gra_Seq.csv
    │
    ├─dataset
    ├─dataset_net
    ├─HTML Version for codes with Results
    │    AIDS-Strategy1.html
    │    AIDS-Strategy2.html
    │
    ├─mcs_discrete_result
    └─mcs_result
```