INTERNATIONAL JOURNAL OF

Open Source Software and Processes

IGI PUBLISHING

Publisher of Peer-Reviewed, Timely, and Innovative Research Since 1988
www.igi-global.com

International Journal of Open Source Software and Processes (IJOSSP)

Volume 11, Issue 3
July - September 2020

Antonio Pecchia - Università degli Studi di Napoli Federico II, Italy

9781799806073

International Journal of Open Source Software and Processes

Volume 11 • Issue 3 • July-September 2020 • ISSN: 1942-3926 • eISSN: 1942-3934

IGI PUBLISHING
AN IMPRINT OF IGI GLOBAL
WWW.IGI-GLOBAL.COM

Call for Articles

International Journal of Open Source Software and Processes

Volume 11 • Issue 3 • July-September 2020 • ISSN: 1942-3926 • eISSN: 1942-3934

MISSION

The **International Journal of Open Source Software and Processes (IJOSSP)** publishes high-quality original research articles on the large field of open source software and processes. The primary mission is to enhance our understanding of this field and neighbouring areas by providing a focused outlet for rigorous research employing a multitude of approaches.

COVERAGE AND MAJOR TOPICS

The topics of interest in this journal include, but are not limited to:

Business models for open source and other community-created artifacts • Case studies of open source projects, their participants and/or their development process • Characteristics of open source software projects, products, and processes • Communication and coordination in open source projects • Customer co-creation and user participation in (software) design • Economic analyses of open source • Economics of a distributed innovation process • Evolution of both open source software artifacts and open source communities • Implications of open source software for functional areas like public administration or teaching • Legal issues of open source software • Motivation of participants in open source projects and other distributed development efforts • Open science and open knowledge • Open source adoption and quality • Open source software development processes • Usage and adoption of open source software in different application areas and/or countries • User-centered innovation processes

ALL INQUIRIES REGARDING IJOSSP SHOULD BE DIRECTED TO THE ATTENTION OF:

Antonio Pecchia, Editor-in-Chief • IJOSSP@igi-global.com

ALL MANUSCRIPT SUBMISSIONS TO IJOSSP SHOULD BE SENT THROUGH THE ONLINE SUBMISSION SYSTEM:

http://www.igi-global.com/authorseditors/titlesubmission/newproject.aspx

IDEAS FOR SPECIAL THEME ISSUES MAY BE SUBMITTED TO THE EDITOR(S)-IN-CHIEF

PLEASE RECOMMEND THIS PUBLICATION TO YOUR LIBRARIAN

For a convenient easy-to-use library recommendation form, please visit:
http://www.igi-global.com/IJOSSP

Table of Contents

International Journal of Open Source Software and Processes

Volume 11 • Issue 3 • July-September-2020 • ISSN: 1942-3926 • eISSN: 1942-3934

Research Articles

The *International Journal of Open Source Software and Processes* is indexed or listed in the following: ACM Digital Library; Bacon's Media Directory; Cabell's Directories; DBLP; GetCited; Google Scholar; INSPEC; JournalTOCs; MediaFinder; Norwegian Social Science Data Services (NSD); SCOPUS; The Index of Information Systems Journals; The Standard Periodical Directory; Ulrich's Periodicals Directory

International Journal of Open Source Software and Processes
Volume 11 • Issue 3 • July-September 2020

Efficient Algorithms for Cleaning and Indexing of Graph data

Santhosh Kumar D. K., Canara Engineering College, India
Demain Antony DMello, Canara Engineering College, Visvesvaraya Technological University (VTU), Belagavi, India

ABSTRACT

Information extraction and analysis from the enormous graph data is expanding rapidly. From the survey, it is observed that 80% of researchers spend more than 40% of their project time in data cleaning. This signifies a huge need for data cleaning. Due to the characteristics of big data, the storage and retrieval is another major concern and is addressed by data indexing. The existing data cleaning techniques try to clean the graph data based on information like structural attributes and event log sequences. The cleaning of graph data on a single piece of information alone will not increase the performance of computation. Along with node, the label can also be inconsistent, so it is highly desirable to clean both to improve the performance. This paper addresses aforesaid issue by proposing graph data cleaning algorithm to detect the unstructured information along with inconsistent labeling and clean the data by applying rules and verify based on data inconsistency. The authors propose an indexing algorithm based on CSS-tree to build an efficient and scalable graph indexing on top of Hadoop.

KEYWORDS

Big Data, Data Mining, Data Science, Graph Data Analytics, Graph Data Cleaning, Graph Data Indexing, Hadoop, Map-Reduce

DOI: 10.4018/IJOSSP.2020070101

1. INTRODUCTION

Big Data is a highly discussed phrase in the last decade. Due to the digitalization, social networks, IoT, Healthcare, automation, etc., The data is produced at a high rate, variety and volume; such data is categorized into five major types. A category called Data Store which categorizes the data based on how the data is stored. It comes with three types of data stores 1) Document oriented, 2) Column-oriented, and 3) Graph data (Santhosh Kumar D K and Demian Antony D'Mello, 2020). In many situations, the data generated from the earlier mentioned sources, are in numerical form or converted to numerical form. Such data is recorded in a matrix, and is used as a parameter to a predictive model. These data have relationships that can be captured in graphs as a collection vertices and edges i.e., G (V, E). This leads to many challenges, creates opportunities, and a need for graph data analytics or graph processing models (Kalashnikov and Mehrotra, 2006).

The graph data is collected from the varieties of sources, which are not in the form to analyze or feed to algorithms for analysis. Before analyzing the graph data, the essential part is to construct graph data and store it for further processing. To construct the graph data first extract and integrate data from multiple sources like social networks, Web pages, etc. Then, prepare the graph data. The quality of input drives the effectiveness of computing models in deriving accurate outcomes. Otherwise, the complete model works for the "garbage-in, garbage-out" principal, the dirty data results in unpredictable effects on the outcome of the model (Gani *et al.*, 2016). The cleansing can be done either manually or by ad-hoc routines, because cleaning is purely done on the interest in data that may vary from one model to another model (Heidari *et al.*, 2018). It's evident from the survey of last decades that more than 40% of the project time is spent by more than 80% of the researchers working on data analysis, the cleaning and preparation of the data (Kalashnikov and Mehrotra, 2006).

Graphs are being used to quickly link various kinds of associated information which have made graphs pervasive with high volume and diversity. Such a massive collection of graphs with billions of vertices(entities) and edges (relationships) generated by social networks and Web leads to the open challenges of managing and processing the graph data (Zomaya and Sakr, 2017). The graph data drives to the development of new powerful frameworks with a highly distributed parallel graph processing model. Examples of graph data are Social networks, customer interactions with enterprises network, biological networks, bibliographic citation networks. Graph data poses several challenges for suitable implementations to meet the requirements (Heidari *et al.*, 2018).

For examining big data, effective handling of the indexing strategy needs to be sketched. Current technologies fall off the vicinity of indexing big data; they are not entirely constructed to inspect such grouped, distributed, and multi-sized data scaled from terabytes to petabytes (Mittal, 2017). Indexing is employed in big data to accomplish retrieval from large, complicated sets of data with scalable, dispersed storage in the cloud. It is illogical to execute and manually explore such records. A

practical, indexing technique with high-throughput would optimize the operation of executing data queries. Hence, effective indexing strategies are necessary for efficiently accessing big data (Douze, Sablayrolles and Jegou, 2018).

1.1 Motivation and Contribution

The success of the graph analytics directly depends on the data quality fed to the analytics algorithm. This data quality also plays a significant role in deciding the efficiency of the algorithm. The time consumed to analyze is based on how fast data can be fetched and fed in every analytics stage. To speed up the retrieval of data, and efficient data indexing technique is required. Especially for the graph data, the survey highlights that the index constructed using tree data structures are highly effective.

Despite recent advances in data analytics, scalable graph analytics is still stimulating in multiple formats. The following are the key issues of the graph analytics.

1. The major challenge of detecting and cleaning dirty data originated from multiple sources.
2. Designing a Distributed graph indexing algorithm.
3. Merging the resultant index files form multiple splits.
4. Many graph indexing techniques are brittle concerning changes in the graph.

In this paper, two significant issues of graph data analytics are addressed. 1) Detecting the dirty data in the graph dataset and cleaning it. The data cleaning which involves detection of unstructured information, inconsistent labeling and cleaning the dataset by applying regular expressions, rules and verifying the structure, consistency of data. 2) For the cleaned graph dataset build an index using the algorithm CSS-Tree (Rao *et al.*, 1998) on top of Hadoop Map-Reduce paradigm to speed up the construction and handle the scaling data. In this paper, the authors intended to make contributions by addressing the following challenges.

1.2 Flexibility and Applicability

Usually, data cleaning algorithms or techniques are designed for a specific type of rule; by using such rules, it is challenging to address all the issues of data cleaning with different aspects. The proposed graph data cleaning algorithm addresses issues by incorporating three varieties of rules and data repair and corresponding complex logical expressions. The dynamic nature of the graph cleaning algorithm makes an algorithm more flexible and comprehensive for data cleaning. The proposed graph data indexing algorithm constructs index using the tree data structure, which makes the Algorithm flexible and the data structure creates index at the cache, which makes it applicable on any graph dataset.

1.3 Deployment and Extensibility

Many data cleaning algorithms proposed by researchers have limitations on real-world graph datasets without fusion customization. The previous algorithms are static, limiting adding new rules or reusing existing, which is inherited from other domains

during run time based on the configuration requirements. The proposed data cleaning algorithm supports dynamic rule generation and data repair, making the algorithm more extensible and deployable. The existing graph indexing algorithms are not built on top of the distributed platform, which limits their speed of indexing construction and fails with a large dataset. The proposed graph data indexing algorithm built on top of the famous distributed framework Hadoop, which makes the algorithm more extensible and deployable.

1.4 System Evolution

When data is fed to an algorithm, there is a huge need for fine tuning the rules and expressions strategies for data cleaning repair so that the updated strategies need proper adoption. The Graph data cleaning algorithm supports user intervention to achieve higher data quality. There is always a scope for improving and refining rules of cleaning algorithm to achieve the higher quality of data. The graph data indexing algorithm is built with three advantages, 1) Tree data structures which are best for indexing the graph dataset. 2) Index construction is done on cache memory which increases the processing speed. 3) Built on top of Hadoop Distributed Framework's Map-Reduce paradigm which supports colossal data size.

The organization of the paper is as follows. The related literature survey is presented in section 2; the proposed system is introduced in section 3 along with proposed Graph Cleaning and Graph Indexing algorithms. Section 4 presents the analysis of the significant results of proposed Graph Cleaning and Graph Indexing algorithms and the article is concluded in section 5.

2. LITERATURE SURVEY

In recent years the challenges of Graph dataset cleaning have engaged the academia and industry through several endeavors. Graph indexing is one more emerging area, also one of the critical tasks of graph analytics. To address these issues on the graph dataset authors has proposed the Graph data cleaning algorithm. Another issue of how fast data can be accessed; whose solution is by Indexing techniques. However, most of the indexing techniques are disk-based or RAM-based, which leads to much delay in accessing the data, especially when data is huge. This section discusses the overview of the current work most related to Data Cleaning and Indexing approaches proposed in this article.

The process of improving the quality of the data by identifying and decimating instance level contaminated data is called Data Cleaning. It is a process of recognizing and resolving issues. It includes locating, analyzing, and correcting the erroneous data. The prime application technologies comprise data identifying, inspecting, and data rectifying technologies. Data cleaning comprises of 4 phases; data pre-processing, anomaly recognition and resolution, and data verification. The technology employed in each phase of data cleaning is non-identical and can adopt machine learning, detection

link, data mining, and other algorithms to detect anomalous data (Kalashnikov and Mehrotra, 2006).

A particular approach can address many types of cleaning problems. Different types of data cleaning issues are identified and explored in the literature. Some of them are handling missing data, dealing with erroneous data, disambiguation and entities resolutions, etc. Many techniques are used to clean the extracted data. Some studies also proposed data cleaning support and finding repairs for the database and statistical methods. To guarantee the accuracy of the repair found, algorithm explicitly demands the user to check and verify the data repairs (Galhardas, Lopes and Santos, 2011).

Data cleaning is generally categorized into two categories: domain-specific and domain-independent data cleaning. The Domain-specific data cleaning involves manual cleaning, expertise in the respective area to interpret the data. Domain independent data cleaning primarily directs towards the extensive database users, substantially the relational database users (Mezzanzanica *et al.*, 2015). It is not about mastering precise knowledge in precise areas. Furthermore, it is straightforward to consolidate with DBMS and acceptable for varied business areas. Comparatively, this type of cleaning technology has broader span of applications and the lesser necessity for entrants (Cui, St and Potok, 2013).

The data pre-processing comprises of systematizing and assembling the user data set eligible for processing, attentively examining and comprehending the data set, structuring an instinctive depiction in the monitoring process, and straight elimination of the evident dis-qualified data, conclusively creating a resultant data set from the initial evaluation (Jin *et al.*, 2018). The collaboration of data management and business process has been accentuated for numerous workflow networks, which supports data and flow networks in domains like E-Governance, Healthcare, and Web applications. The existing studies highlight that execution performance can be accelerated or optimized (Fender *et al.*, 2016).

According to study on primary issues of redundant data traits, there exist multiple structural data quality issues caused due to a missing value, noise, and redundant and duplicate data. To handle data anomaly detection and repair, the PCA (Principal Component Analysis) dimensionality reduction algorithm is chiefly utilized for redundancy traits. The prime motive of dimensionality reduction is reconstructing the samples of high-dimensional space into low-dimensional space by transforming or mapping and then eliminating repetitive or insignificant traits by natural choice (Decastro-garcía *et al.*, 2018).

The data management community has conducted studies on processing data, mainly concentrating on query processing over workflow execution. A classical query loads a process specification and an execution pattern that tries to recognize a similar specified pattern out of all execution. The query may be already embedded with auxiliary conditions or condition can also be added, such as probability or type of information (Lee *et al.*, 2019). In addition to this as a first application, original queries which run over workflow are also explored. A traditional provenance query computes events transitive closure dependencies in the process data. A distinct study

shows the evaluating the constructional dissimilarities of two executions (Wang *et al.*, 2015). In this study, the work is implemented to clean the data rather than changing the structure. The proposed approach is to identify the dirty part of the data and clean it. Data storage and retrieval is a major concern in the big data like other data management system. For faster access to data, the indexing techniques are used. The disk-based indexing techniques are popular and are used in many scenarios; one such popular indexing technique is the hash table, which builds the index table with the help of a hash function to map key to values pair in the index bucket. These disk-based indexing techniques gobble up time to access data.

While loading, it is necessary to interrelate the graph data extracted from multiple sources and construct the combined graphs, which are sometimes called knowledge-graphs. The next challenge is how to store this data, which is heterogeneous and rapidly growing, a technique that supports faster retrieval and efficient management techniques. The indexing techniques are the best for such scenarios, but existing traditional indexing techniques become inefficient with highly thriving index size and seek time; there is a need for an optimized indexing scheme. Systematically made decisions help in acquiring prompt responses to the questions, and productive employment of calculating resources aids in advancing the potential of search exercises on big data. It brings the researchers in a state to advocate effective data analytics outcomes for the vast data (Keasavan and Kumar, 2019).

Researchers have implemented diverse indexing techniques with big data as its nucleus. The massive aggregated text's effective indexing in cloud is based on R-tree to facilitate multi-dimensional indexing of data in cloud (Shao *et al.*, 2019). Indexing with a supplement regulates an exchange between performance and power. As big data pilots a complicated clinical domain, reduced time and price are necessary for data inspection with strategies; it is evident that data indexing contributes effectively to narrowed time even though it is pricey, yet must be tolerated while establishing such strategies. According to the big data characteristics, an effective indexing technique is executed to fulfill big data demands(Zhang *et al.*, 2018).

A detailed survey on indexing algorithm/techniques is presented. Survey has detailed the discussion of various type of dataset, most suitable indexing algorithms for a given dataset. It also outlined the taxonomies to illustrate the emerging area like cloud, big data, and machine learning. The study highlights that tree-structured algorithms are more suitable to index graph dataset. (Zoumpatianos, Idreos and Palpanas, 2014; Gani *et al.*, 2016). The updating of index which is constructed using tree data structures are faster and scalable (Bonnici *et al.*, 2010). The tree structured algorithms implemented on the distributed architectures adaptively tunes and automatically fits the workload on the fly (Zhang, Hu and Yang, 2007; Dathathri *et al.*, 2019)

A new indexing scheme based on frequent subtree is presented. The tree can preserve more structural information of graphs than a simple path and is also easier to manipulate. The size of the index can also be controlled by setting the threshold parameters. Hadoop Map-Reduce paradigm can be utilized for building the index

(Dittrich *et al.*, 2010). There are many leading technologies to operate on Big data available in the market. These technologies are cloud computing, distributed systems data warehouses, among others. The Hadoop is the popular distributed system mainly because they divide the task among the multiple nodes connected to the cluster, and tasks are computed in parallel to improve the performance with reduced cost, and Hadoop works on commodity hardware. It also has the added advantage of being an open-source framework. The distributed nature of Hadoop leverages the performance and improves efficiency. The Map-Reduce tasks are scan oriented, which helps to update the index frequently(Richter *et al.*, 2014).

The survey motivates the authors to address the two major issues highlighted in the study, i.e., data cleaning and data indexing. For any data analytics algorithm, the input dataset plays a major role in deciding the algorithm's performance and accuracy, and as it is highlighted that data cleaning is very important stage of data analytics. However, the present graph data cleaning and indexing techniques fail to handle large dataset and are static in nature.

3. PROPOSED SYSTEM

It is of prime importance to comprehend that all the data flow stages are prone to error invention, including the initial data cleaning itself. Figure 1 demonstrates the working of the proposed model for graph data cleaning and indexing. The graph data is extracted from a variety of data sources. While loading, it is necessary to interrelate the graph data extracted from multiple sources and construct the combined graphs; inconsistencies in the dataset is also caused during the process of data integration. Several of such origins leads to one or more of the error categories such as Measurement errors, Data entry error, Processing errors and Data integration errors (Fan and Geerts, 2007).

Data cleaning is performed mainly in two stages. Identify the erroneous dirty data and modifying the it. In this paper, the authors have highlighted many existing rule-based data cleaning techniques. These do not support complex logical operations and data repair functions. Duplication is also an important cause for erroneous data, that needs to be addressed by the data cleaning process. To address the above problems, authors propose to compare with other rule-based data repair and cleaning algorithms.

It provides a facility to configure the rules dynamically, where users will have the facility to update the rules during run time. While outlining the rules, the complex logic and minimization principles should comply with data cleaning rules and inheriting the interdisciplinary rules. It incorporates the configuration of the rules along with data transformation, by which the data cleaning can be done effectively along with necessary data repairs on the collected data from various sources. The rules are from regular expressions, Java functions (to repair the data); and DROOLS(Proctor, 2016) are embedded to support multi-domain rules rather than sticking to a single rule type.

Data cleaning may involve preliminary execution policies for preventing the error ahead of its occurrence. Though error-prevention policies can narrow regular

errors but not banish them, and several data errors will be confronted incidentally. Even after such policies, there is still a massive demand for the active and systematic search for recognizing and remedying issues/errors. Cleaning graph data comprises of many iterations of recognizing, evaluating, remedying, and recording this routine. The manual Cleaning approaches are removal, direct correction, scaling imputation, and flagging. Nevertheless, these are only effective with small structured datasets.

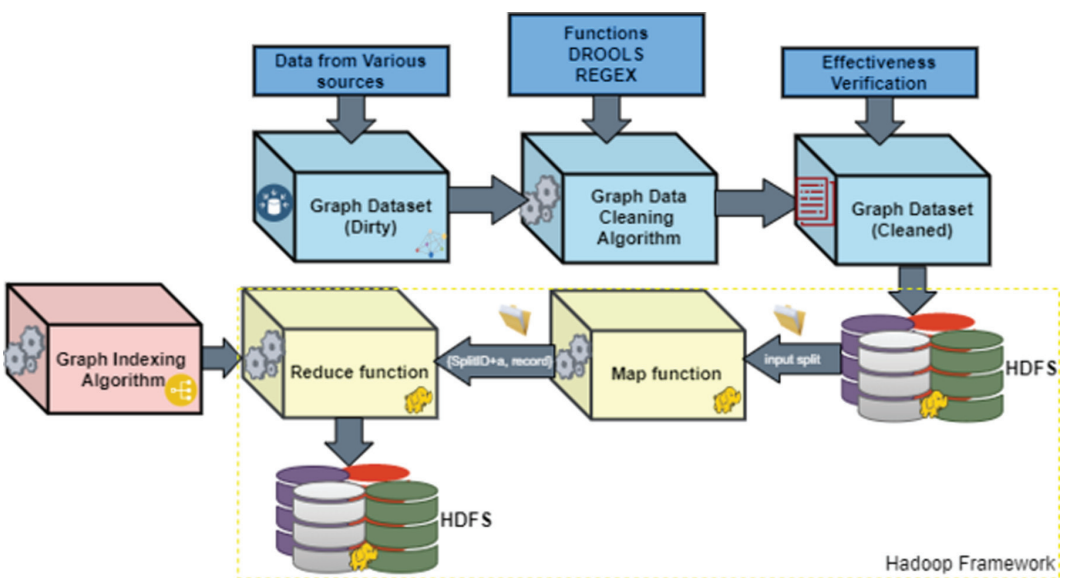
A large number of current graph indexes are fragile concerning graph changes, making it cost-ineffective for the dynamic graphs. Therefore, for dynamic graphs, it is essential to plot indexing strategies that are adaptable and resilient to changes in the graph.

3.1 Graph Cleaning Algorithm

Graph dataset cleaning is a result of numerous iterations; part of it involves manual tuning and transforming using the regular expressions, functions and DROOLS. Even though the implementation of the proposed algorithm is for a given problem, it also subjects to strict verification and validation process. For the real dataset, when the Graph Cleaning algorithm is executed, a minor tuning is required, like data transformations. This algorithm cleans the input graph dataset and also reads the collection of graph datasets that need manual data repair, like insert/delete instances.

Algorithm 1 demonstrates the procedure for graph dataset cleaning. The priority queue PQ is used in this algorithm. The element in the PQ represents either a state of branching or a node. The loop at line 2 fetches every element one by one from PQ. Each element is fetched by line 3 say E_{k-1} along with L' check for minimum lower bound by $LowerBound(E_i, L')$ and that element is added to PQ at line 4. Line 5 checks

Figure 1. Architecture of proposed system



for whether the extracted node exits in the specification if true directly it returns the result.

Otherwise line 8 extracts the k[th] transition from the execution trace and branches to generate a new bipartite graph and adds it to execution trace by line 9. The loop at lines 10-14 is responsible for determining each $a_i \in \Pr e_{R_E}(t_k)$, the preset function Pre() gives preset of a bipartite graph. Then verifies for the preset of transition and updates the PQ along with specification labelling and finally returns the result. This process is repeated for every node in the PQ. Finally cleaned graph dataset is moved to HDFS as illustrated in figure 1.

Algorithm 1: Graph_Cleaning (T_E, L, T_s)
Input: Graph dataset with an execution trace, Label and Specification (T_E, L, T_s)
Output: Clean Graph Data L' with $(T_E, L) \models T_s$
Begin
PQ $:= \{(\phi, L)\}$
Repeat PQ $\neq \phi$ then do
$(E_{k-1}, L') := \min(E_i, L') \in_{PQ} LowerBound(E_i, L')$
PQ := PQ - $\{(E_{k-1}, L')\}$
Check if $(T_E, L') \models T_S$ then
Return L'
Otherwise
t_k := E(k) the k[th] transition in E{ branch t_k to generate $L'(t_k)$}
$E_k := E_{k-1} \cup \{t_k\}$
Repeat for every $a_i \in \Pr e_{R_E}(t_k)$ then do
$A_i^C :=$ all valid labelling $L'(a_i)$ of a_i
Check if $L'(\Pr e_{R_E}(t_k)) = \Pr e_{R_S}(X)$ then
PQ := PQ $\cup \{(E_k, L')\}$
$L'(t_k) := L'_{min(t_k)}$
return L'
End

3.2 Graph Indexing Algorithm

The Graph indexing algorithm is implemented on top of the Hadoop 3.0.1 framework. In this algorithm, the index is constructed for each data split based on the Cache-Conscious Tree(CSS-Tree). Graph indexing consists of a sparse directory of indexes. For each data split, the covering index is added. The Index creation for a given task T in Map-Reduce for a user U is expressed as

Table 1. Notations and abbreviations of this article

CSS-Tree	Cache-Conscious Tree
HDFS	Hadoop Distributed File System
T	Bipartite graph with Petri net (A, P, R)
A	Finite set of User/people
P	Finite set of transitions
R	Follow relation
L and L'	Edge labelling
S	Process specification with triplet (A_S,P_S,R_S)
E	Execution trace (A_E, P_E, R_E)
PQ	Priority Queue
Pre()	Bipartite Graph Preset function
min()	Minimum function
$\oplus$	Concatenation operation

$$Index\_creation_{u_i}(U)\begin{cases} map(key,value) \rightarrow getSplitID() \oplus projection_{u_i}(key \oplus value), key \oplus value \\ reduce(key,vset) \rightarrow \left[vset \oplus IndexBuider_{u_i}(vset)\right] \end{cases}$$

As illustrated in figure 1, the cleaned graph dataset is stored in the Hadoop Distributed File System (HDFS). Provided the Map-Reduce first splits the given input into the number of partitions, then its map-function reads the pair of key-value {offset, record} and generates intermediate key-value pair, which consists of composite key concatenated with index attribute and a record with attributes of an index record. The key-value pair is given by {SplitID+a, record}. The CSS-Tree works on the sorted input values; to sort this, the Map-Reduce framework is used in creating interesting orders. As reduce-function works locally, the sorting task is completed faster. To sort these records, Map-Reduce is instructed with UDF comp instruction, which internally uses Map-Reduce compare function to compare composite keys.

```
Algorithm 2:  Graph_Indexing(job)
Input: Clean Graph data set job
Output: Index with key value pair
Begin
        Repeat for every f in GetFiles(job) then do
                p = f.GetPath()
```

```
                    s = GetInputStream(p)
                    flength = f.GetLength()
                          Repeate if flength > 0 then do
                                 s.seek(flength - FOOTER_
SIZE)
                                footer = ReadFooter(s)
                                 split_length =footer.
GetSplitSize()
                                 flength = flength - (split_
length + FOOTER_SIZE)
                                b = GetBlockLocations(p,
flength)
                                 new_split = CreateSplit (p,
flength, split_length, b)
                                 split.Add(new_split)
key = job.GetKey()
value = job.GetValue()
high_key = job.GetHighKey()
end_split = split.GetEnd()
While flength < end_split then
r = ReadNextRecord(split)
flength = flength + r.Size()
Check if r.key < high_key then
SetKeyValue(key, value, r)
Return
End
```

Since the index is built per partition split, the partition should be preserved in each reducer call; this is achieved with the grouping function which groups based on the split identifier. Finally, the reduce function creates an index with the help of the local Index Builder process, which builds the graph index using CSS-Tree on sorted data. The concatenated set of index values are emitted by Reduce-function and stored back on HDFS.

Algorithm 2 demonstrates the proposed Graph Indexing algorithm. Line 1 extracts and iterates over all the dataset files. Every file's path is scanned and placed in the stream and fetches file length from lines 2-4. Scanning the split footer by input stream, footer split size, and updating file length is performed by lines 6-9. Then next part extracts the block location to generate a logical split and adds it to the list by lines 10-12. This process is repeated for all the footers in the file. Finally, a list of logical split is ready to generate an index.

The key properties are fetched from lines 13-15 along with the end split from line 16. The Algorithm verifies for whether the file length is within the limit, if true reads each record from the split and updates the file length by the lines17-19. Lines 20-21 updates the new key-value pair for each record by verifying the record key with the high

key. This process iterates for all the records in the split. Finally, the index is created for each split and merged to generate a complete index for the given graph dataset.

4. EXPERIMENTATION AND RESULTS

In this section, the experimental results of proposed algorithms are reported. The accuracy and performance of the Graph Data Cleaning algorithm is compared with Graph Relable (Song, 2014) and Trace alignment (De Leoni, Maggi and Van Der Aalst, 2012) algorithms. Also the results validate the efficiency and effectiveness of the Graph Data Indexing algorithm which is compared with GDIndex (Williams, Huan and Wang, 2007) and C-Tree (He and Singh, 2006) algorithms. Authors have used twitter (MPI) (*Twitter (MPI) - Network analysis of Twitter (MPI) - KONECT*, no date) dataset for the experiments which is publically available on the Konect website and the purpose of selecting the twitter dataset is that, it is the largest dataset available for public access and also it meets the objective of the study.

4.1 Benchmark Setup

All the experiments were performed on PC running on intel i5 5th generation 2.3GHZ, 16GB memory, Ubuntu 18.04 64-bit platform, 256GB SSD, 1TB of disk space. The algorithms are implemented in java, and the Hadoop 3.0.1single node is configured to run the Graph Indexing algorithm.

4.2 Dataset

The data set details are given in table 2. The Twitter (MPI) graph dataset has 52,579,682 vertices (users) and 1,963,263,821 edges (follows). Based on the figures, only 44.12% is consistent, i.e., which means either node is structured, or labels are consistent. In particular, 15.25% content is irrelevant to the specification, i.e., with all the node names not from the specification domain. The remaining inconsistencies (40.63%), is fed to the proposed graph cleaning algorithm. There are 13,665 nodes with inconstant labelling that can be cleaned, and the remaining 511 nodes have no label.

4.3 Graph Cleaning Algorithm Performance Evaluation and Analysis

To examine the accuracy of the graph cleaning algorithm, we randomly imposed inconsistent labelling over 44.12% dataset, which was initially consistent and randomly changed node names as faults. The graph cleaning algorithm is composed of regular expressions used to remove the noisy data, apply imputation to calculate the correlation, and flags used to preserve some critical entries which have a missing value. The proposed cleaning algorithm's result's accuracy is measured by studying and comparing

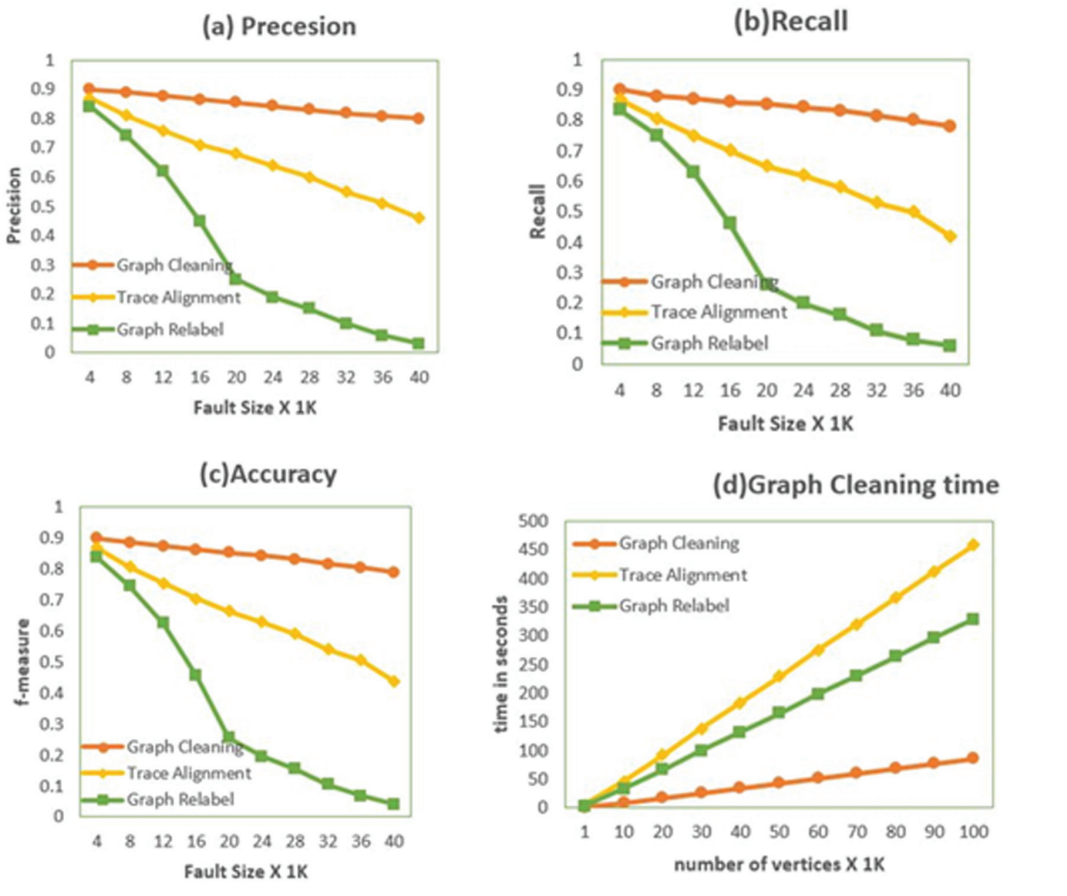
Table 2. Twitter (MPI) dataset

Dataset	Size	Volume	Average degree	Maximum degree	Source
Twitter (MPI)	52.5M	1.96B	74.678	3.6M	Konect

it with the original data (*truth*) of incorrect data replaced previously. On every trace, an experiment has been conducted by randomly inserting faults 100 times, and the final average accuracy is computed. Meantime, the time performance for cleaning is also reported, along with the cleaning over the remaining 40.63% of data. The *truth* is defined as the original collection of correct values given by $(t, L_0'(t))$ these are replaced randomly. The *found* is a set of cleaned results in L', defined as $(t, L'(t))$. The accuracy is evaluated using f-measure; the algorithm with higher f-measure is considered. The following equation gives the f-measure. Similarly, precision and recall are also calculated, and the equations are given below.

This experiment contrasts the proposed Graph cleaning algorithm with Trace Alignment and Graph Relabel algorithms. The Graph Relable has three methods namely, greedy heuristic method, contraction method, and hybrid approach. Graph Relable with hybrid approach achieves high effectiveness. For experimentation, the Graph Relable with hybrid approach is used on twitter dataset. Trace alignment uses heuristics to prune the process state-space. Trace algorithm is concentrates on pruning

Figure 2. Effectiveness (a) Precision, (b) recall, (c) accuracy and (d) Graph Cleaning time of proposed Graph Cleaning algorithm.



and heuristics approaches. For this study the Trace algorithm with heuristics approach is used on twitter dataset. Initially, the experiments were conducted on a small chunk of data ranging from 1K vertices to 100K vertices dataset. To verify the consistency of the proposed algorithm, the fault is injected to dataset. Where the fault size 4K means randomly 4K nodes are altered to check the consistency of graph repairing to eliminate the violations.

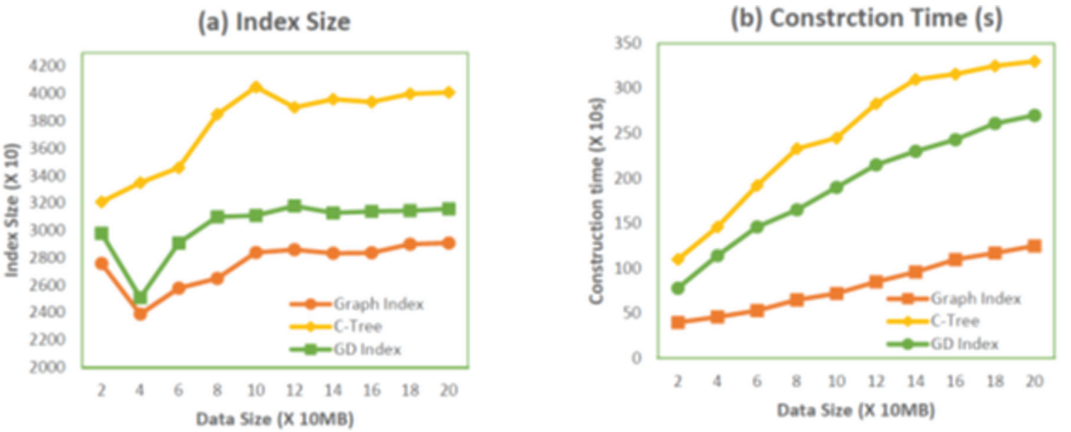
The comparison and the effectiveness of the proposed algorithm are presented based on precision, recall, and accuracy. The Figure 2(a) illustrates the precision of the Graph Cleaning Algorithm, with the least precision 0.8, and the highest precision of 0.9. Similarly, Figure 2(b) illustrates the recall of the Graph Cleaning Algorithm, with least precision not less than 0.78, and the highest precision of 0.9. Figure 2(c) shows the accuracy of the proposed algorithm, which is considerable, with least f-measure as 0.8 and remarkably with the highest f-measure as 0.9. Finally, Figure 2(d) shows the proposed algorithm's performance of scalability with increasing vertices size. The proposed graph cleaning algorithm can clean 100K vertices data in 85.4 seconds which is approximately 4 to 5 times faster than other two algorithms.

4.4 Graph Indexing Algorithm Performance Evaluation and Analysis

The Graph Indexing algorithm performance is compared with the GD Index and C-Tree algorithms. The GD Index algorithm is disk-based indexing technique which builds the index table with the help of a hash function to map key to values pair in the index bucket. The C-Tree does not support the graphs with continuous-valued attributes on vertices or edges. It constructs the tree based on the insertion order, which makes the graph lag with global optimization, and results in the demand for more insertion and splitting operations. However, these algorithms do not scale. The proposed algorithm performance scales well, which grows almost proportionally with the data size.

The same twitter dataset is used for the experimentation. The preliminary reports of indexing algorithms prove the efficiency and effectiveness of the Graph Indexing algorithm. First, the index size of the proposed Graph Index is compared with GD Index and C-Tree. Figure 3(a) depicts the number of indexes generated by each algorithm input dataset size varying from 20MB to 200MB. The proposed algorithms index size is smaller than the other two algorithms. The resulting graphs in figure 3(a) show that even though an increase in dataset size, irrespective of the data size, the size of the index generated by the proposed algorithm remains small and stable. Figure 3(b) presents the computing time required to construct the index patterns for all three algorithms. The index is built from scratch for dataset size varying from 20MB to 200MB. The index building time of all three algorithms is proportional to the dataset size, where the proposed Graph Indexing algorithm is relatively faster than the other two algorithms.

Figure 3. Effectiveness (a) index size and (b) construction time of proposed graph indexing algorithm



5. CONCLUSION

Graph data analytics is one of the significant areas of big data analytics. Graph data analytics leverage its application in Similarity networks, Healthcare, E-Governance, Web Applications, Business processes Workflow networks, Social Media, Interaction network etc.; the data quality is a severe concern in the analytics process. In this paper, the author's proposed Graph data cleaning algorithm addresses data linkage, standardization of the unstructured information and labelling issues in the given graph dataset. It detects the dirty data and cleans the data by enabling the configuration rules dynamically and verifying the structure of data simultaneously. The graph cleaning algorithm enables the run-time rule configuration and inherits the cross-domain rules. The experimental results on the twitter dataset show the accuracy and performance of the Graph cleaning algorithm is significantly higher and acceptable. Along with handling data quality, this paper also addresses how to store the enormous scaling graph data by proposing a Graph indexing algorithm. The graph's structural information is more preserved by tree data structures and also easier to manipulate. The proposed graph indexing algorithm uses CSS-Tree data structure to retrieve data quickly on top of the Hadoop Map-Reduce paradigm. The experimental results on the twitter dataset show significantly small index size and demonstrates that it outperforms other indexing algorithms with a significant margin.

REFERENCES

Bonnici, V. (2010). Enhancing graph database indexing by suffix tree structure. Lecture Notes in Computer Science, 6282, 195–203. doi:10.1007/978-3-642-16001-1_17

Cui, X., St, J., & Potok, T. (2013). GPU enhanced parallel computing for large scale data clustering. *Future Generation Computer Systems. Elsevier B.*, *29*(7), 1736–1741. doi:10.1016/j.future.2012.07.009

Dathathri, R. (2019). Gluon-async: A bulk-asynchronous system for distributed and heterogeneous graph analytics. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 15–28. doi:10.1109/PACT.2019.00010

De Leoni, M., Maggi, F. M., & Van Der Aalst, W. M. P. (2012). Aligning event logs and declarative process models for conformance checking. Lecture Notes in Computer Science, 82–97. doi:10.1007/978-3-642-32885-5_6

Decastro-garcía, N. (2018). *On Detecting and Removing Superficial Redundancy in Vector Databases.* Academic Press.

Dittrich, J., Quiané-Ruiz, J.-A., Jindal, A., Kargin, Y., Setty, V., & Schad, J. (2010). Hadoop++. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *3*(1–2), 515–529. doi:10.14778/1920841.1920908

Douze, M., Sablayrolles, A., & Jegou, H. (2018). Link and Code: Fast Indexing with Graphs and Compact Regression Codes. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 3646–3654. doi:10.1109/CVPR.2018.00384

Fan, W., & Geerts, F. (2007). *Improving Data Quality: Consistency and Accuracy*. Academic Press.

Fender, A. (2016). Accelerated hybrid approach for spectral problems arising in graph analytics. *Procedia Computer Science*, *80*, 2338–2347. doi:10.1016/j.procs.2016.05.435

Galhardas, H., Lopes, A., & Santos, E. (2011). Support for user involvement in data cleaning. Lecture Notes in Computer Science, 6862, 136–151. doi:10.1007/978-3-642-23544-3_11

Gani, A., Siddiqa, A., Shamshirband, S., & Hanum, F. (2016). A survey on indexing techniques for big data: Taxonomy and performance evaluation. *Knowledge and Information Systems*, *46*(2), 241–284. doi:10.1007/s10115-015-0830-y

He, H., & Singh, A. K. (2006). Closure-tree: An index structure for graph queries. *Proceedings - International Conference on Data Engineering*, *2006*, 38. doi:10.1109/ICDE.2006.37

Heidari, S., Simmhan, Y., Calheiros, R. N., & Buyya, R. (2018). Scalable graph processing frameworks: A taxonomy and open challenges. *ACM Computing Surveys*, *51*(3), 1–53. Advance online publication. doi:10.1145/3199523

Jin, C., Jin, R., Chen, K., & Dou, Y. (2018). A Community Detection Approach to Cleaning Extremely Large Face Database. *Computational Intelligence and Neuroscience*, *2018*, 1–10. Advance online publication. doi:10.1155/2018/4512473 PMID:29849547

Kalashnikov, D. V., & Mehrotra, S. (2006). Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems*, *31*(2), 716–767. doi:10.1145/1138394.1138401

Keasavan, V. T., & Kumar, B. S. (2019). Graph based indexing techniques for big data analytics: A systematic survey. *International Journal of Recent Technology and Engineering*, *7*(6), 641–647.

Lee, E. (2019). *Pre-Select Static Caching and Neighborhood Ordering for BFS-like Algorithms on Disk-based Graph Engines*. Academic Press.

Mezzanzanica, M., Boselli, R., Cesarini, M., & Mercorio, F. (2015). A model-based evaluation of data quality activities in KDD. *Information Processing and Management. Elsevier Ltd*, *51*(2), 144–166. doi:10.1016/j.ipm.2014.07.007

Mittal, M. (2017). Indexing Techniques and Challenge in Big Data. *International Journal of Current Engineering and Technology*, *7*(3), 1225–1228. http://inpressco.com/category/ijcet

Proctor, M. (2016). *Drools Documentation, JBoss Rules User Guide- Drools Documentation, Version 6.5.0*. Available at: https://docs.jboss.org/drools/release/6.5.0.Final/drools-docs/html_single/index.html

Rao, J. (1998). Cache conscious indexing for decision-support in main memory. *EuroSys*, *12*, 183. doi:10.1145/2168836.2168855

Richter, S., Quiané-Ruiz, J.-A., Schuh, S., & Dittrich, J. (2014). Towards zero-overhead static and adaptive indexing in Hadoop. *The VLDB Journal*, *23*(3), 469–494. doi:10.1007/s00778-013-0332-z

Santhosh Kumar & D'Mello. (2020). Strategies and Challenges in Big Data: A Short Review. In *Intelligent Systems Design and Applications. ISDA 2018 2018. Advances in Intelligent Systems and Computing*. Springer. doi: 10.1007/978-3-030-16660-1_4

Shao, Z. (2019). BlockGraphChi: Enabling Block Update in Out-of-Core Graph Processing. *International Journal of Parallel Programming*, *47*(4), 668–685. doi:10.1007/s10766-017-0532-z

Song, S. (2014). *Repairing Vertex Labels under Neighborhood Constraints*. Academic Press.

Twitter (MPI) - Network analysis of Twitter (MPI) – KONECT. (n.d.). Available at: http://konect.uni-koblenz.de/networks/twitter_mpi

Wang, J. (2015). Cleaning structured event logs: A graph repair approach. *Proceedings - International Conference on Data Engineering*, 30–41. doi:10.1109/ICDE.2015.7113270

Williams, D. W., Huan, J., & Wang, W. (2007). Graph database indexing using structured graph decomposition. *Proceedings - International Conference on Data Engineering*, 976–985. doi:10.1109/ICDE.2007.368956

Zhang, M. (2018). GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition. *Proceedings - International Symposium on High-Performance Computer Architecture*, 544–557. doi:10.1109/HPCA.2018.00053

Zhang, S., Hu, M., & Yang, J. (2007). TreePi: A novel graph indexing method. *Proceedings - International Conference on Data Engineering*, 966–975. doi:10.1109/ICDE.2007.368955

Zomaya, A. Y., & Sakr, S. (2017). *Handbook of Big Data Technologies.* doi:10.1007/978-3-319-49340-4

Zoumpatianos, K., Idreos, S., & Palpanas, T. (2014). Indexing for interactive exploration of big data series. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1555–1566. doi:10.1145/2588555.2610498

Santhosh Kumar D. K. is currently pursuing PhD program at VTU Belagavi, Karnataka and received his M.Tech in Computer Science and Engineering at NMAM Institute of Technology, Nitte, Karnataka, India and B E in Computer Science and Engineering at Dr. Ambedkar Institute of Technology, Bangalore, Karnataka, India. He is Assistant Professor at the Department of Computer Science and Engineering, Canara Engineering College Mangaluru, Karnataka, India. His research interests include Big Data Analytics, Machine Learning, Data Science and Block Chain Technology. He is author of research studies published at national and international journals, conference proceedings.

Demian Antony D'Mello is currently serving as the Professor & Head of Computer Science & Engineering Department at Canara Engineering College, Mangaluru, Karnataka, India. He received B.E. in Computer Engineering from Mangalore University and M.Tech & Ph.D from NITK Surathkal, Karnataka, India. He has over 20 years of experience in education and research. He has over 45 publications in reputed international conferences/journals. He is a regular reviewer for many reputed journals and conferences. He also serves as an editorial board member for reputed journals. His areas of interest are Web services, Cloud computing, Big data Analytics, Internet technologies and Digital image processing.

International Journal of Open Source Software and Processes
Volume 11 • Issue 3 • July-September 2020

Empirical Evaluation of Bug Proneness Index Algorithm

Nayeem Ahmad Bhat, Department of Computer Science, North Campus, University of Kashmir, India
Sheikh Umar Farooq, Department of Computer Science, North Campus, University of Kashmir, India

ABSTRACT

Researchers have devised and implemented different bug prediction approaches that use different metrics to predict bugs in software modules. However, the focus of research has been on proposing new approaches/models to predict bugs rather than on validating performance of existing approaches. In this paper, the authors evaluate and validate the findings of an algorithm that predicts the bug proneness index (bug score) of the software classes/modules. The algorithm uses normalized marginal R square values of software metrics as weights to the normalized metrics to compute bug proneness index (bug score). The experiment was performed on Eclipse JDT Core and reports significant improvements in F-measure of their algorithm as compared to the multiple linear regression. The authors found that there was no improvement in F-measure of evaluated algorithm compared to multiple linear regression. The use of marginal R square values as weights to the metrics in linear functions in the evaluated model instead of regression coefficients had no performance boost compared to the multiple linear regression.

KEYWORDS

Bug Prediction, Bug Proneness Index, Multiple Linear Regression, Marginal R Square

DOI: 10.4018/IJOSSP.2020070102

1. INTRODUCTION

Identifying and fixing bugs is one of the difficult and time-consuming tasks in software development lifecycle (SDLC). Our failure to manage the complexity and identify the bug prone modules results in the projects that are late, over budget and deficient in their implicit and stated quality requirements. One promising approach for detecting bugs in software before release is to use defect prediction techniques. The prediction techniques intend to predict bug score (probability of bugs, number of bugs) of different modules of the software. The aim is to get a ranked/prioritized list of different modules so that bug detection and fixing effort can be allocated in an optimal manner.

The bug prediction problem has been researched thoroughly (Bernstein et al., 2007; D'Ambros et al., 2010; El Emam et al., 2001; Graves et al., 2000; Gyimothy et al., 2005; Herzig, 2014; Kim et al., 2007; Nagappan & Ball, 2005b; Ratzinger et al., 2008). Using historical data (data extracted from Software Change Metric (SCM) and/or bug tracking systems like CVS/Subversion, Bugzilla/Jira) of the project, the task is to predict the bug score of the software modules in future releases. In the past, a variety of models have been designed to tackle the problem, relying on diverse information, such as code metrics (Basili et al., 1996; Hassan, 2009; Nagappan et al., 2006; Nagappan & Ball, 2005a; Zimmermann et al., 2007) (lines of code, response for class, coupling between objects, complexity), process metrics (Di Nucci et al., 2017, 2015; Graves et al., 2000; Mnkandla & Mpofu, 2016; Moser et al., 2008; Rahman & Devanbu, 2013; Van Rysselberghe, 2008; Zimmermann et al., 2007) (number of changes, number of refactorings) or past defects (Felix & Lee, 2017; Hassan & Holt, 2005; Joshi et al., 2007; Kim et al., 2007; Zimmermann et al., 2007). The major focus of research mostly has been on proposing new models/techniques to predict bugs rather than on validating performance of existing approaches. Validation of existing approaches is very important before their results are generalized and used in practice. This becomes more important when some method/technique claims to outperform any or all of the usual established problem-solving approaches in a problem domain.

In this paper, we evaluate an algorithm (Puranik et al., 2016) with respect to multiple linear regression for the bug prediction task. The aim is to validate the findings of the algorithm proposed by (Puranik et al., 2016) so that the findings are validated and used in practice. The original experiment reports improved prediction performance of the evaluated algorithm (Puranik et al., 2016) compared to the multiple linear regression over Eclipse JDT Core test set. For evaluation, we compare the performance of the algorithm with multiple linear regression model on two datasets i.e. *Eclipse JDT Core* and *Eclipse Pde UI datasets*. When F-measure, a composite measure of precision and recall, is used for evaluation the algorithm performs similar to multiple linear regression for both the training and test sets. The algorithm achieves an F-measure of 89% as compared to an F-measure of 89% achieved using multiple linear regression model, for Eclipse JDT Core test set. Similarly, the algorithm achieves an F-measure of 92% as compared to an F-measure of 92% achieved using multiple linear regression

model for Eclipse Pde UI test set. Similarly, when other performance measures like accuracy, precision, and recall are considered the algorithm has similar performance as compared to multiple linear regression. Thus, we conclude, the algorithm had no performance improvement over multiple linear regression as opposed to the conclusion drawn by experiment (Puranik et al., 2016). Our experiment endorses the point of validating results of prediction techniques by replicating the experiments before they are adopted in practice. Replications guarantee to help us to derive stronger conclusions about the phenomenon under investigation. A single experiment may not be able to answer all the questions under investigation confidently given the limitation of the empirical studies.

The organization of this paper is as follows. In section 2, a brief literature survey is presented. Section 3 provides the motivation for carrying out the work. Section 4 and 5 gives an overview of the datasets used in the study and detailed working of the evaluated algorithm respectively. Section 6 presents the results obtained when the algorithm is implemented. A comparison between the results of the evaluated algorithm and that of multiple linear regression is given in section 7. The threats to the validity of the experiment are reported in section 8 and finally the conclusions drawn from this study are presented in section 9.

2. LITERATURE SURVEY

More than 80% of software development costs are incurred in finding and fixing of bugs. The bug/defect prediction models are promised to reduce these costs significantly (Arar & Ayan, 2015; Ceylan et al., 2006; Fagan, 2002; Moser et al., 2008; Mullen & Gokhale, 2005; Rajbahadur et al., 2017; Shull et al., 2002). The first and foremost step in bug prediction is selection of metrics. Complexity of prediction process is increased when more metrics are used in the model construction (Shivaji et al., 2009). Moreover, the performance is dropped considerably with inclusion of irrelevant metrics. In the past, several bug prediction models that exploit diverse information, such as code metrics (Basili et al., 1996; Hassan, 2009; Nagappan et al., 2006; Nagappan & Ball, 2005a; Zimmermann et al., 2007) (lines of code, response for class, coupling between objects, complexity), process metrics (Di Nucci et al., 2017, 2015; Graves et al., 2000; Mnkandla & Mpofu, 2016; Moser et al., 2008; Rahman & Devanbu, 2013; Van Rysselberghe, 2008; Zimmermann et al., 2007) (number of changes, number of refactorings), past defects (Felix & Lee, 2017; Hassan & Holt, 2005; Joshi et al., 2007; Kim et al., 2007; Zimmermann et al., 2007) and metrics relating to developers have been proposed. In previous studies, process metrics are reported to perform well (D'Ambros et al., 2010; Madeyski & Jureczko, 2015; Nagappan et al., 2010). (D'Ambros et al., 2010) specifically report that predictions based on a single metric do not perform consistently well across different systems. (Madeyski & Jureczko, 2015) report certain process metrics like number of developers changing a file can significantly improve defect prediction. Appropriate metrics are selected through empirical verifications (Raman & Ioerger, 2003; Xu et al., 2016). Different classification techniques have

been applied in defect prediction. A detailed explanation of classification techniques is given by (Witten & Frank, 2005). (Hall et al., 2011) through an analysis of results from 19 defect prediction studies suggest Naïve Bayes and Logistic Regression perform best. Recently, we have seen the application of Ensemble of classifiers in prediction (Laradji et al., 2015; Minku & Yao, 2013; Petrić et al., 2016; Sun et al., 2012). The classification models can broadly be categorized into *discretized defect classifiers* and *regression-based defect classifiers*. In discretized defect classifiers the continuous defect count is discretized into "defective" and "non-defective" classes and used as response variable (Cataldo et al., 2009; Lessmann et al., 2008; Mockus, 2010) in model training. However, a significant amount of information is lost as a result of discretization of the response variable (Altman & Royston, 2006; Cohen, 1983; Royston et al., 2006) and undesired false positives are introduced (Austin & Brunner, 2004). To avoid the information loss, the continuous response variable can be used in training regression model to predict the defect score first, and then using a threshold, classification can be performed on regression model defect score – an idea called *regression-based classifier* (Hou et al., 2012; Xiang et al., 2010). An advantage of regression-based classifier is that the regression scores can be used to rank defective modules according to their defect proneness score, and thus support prioritizing their treatment (testing and code inspection) – more practical scenario for projects with a fixed quality assurance budget/resource. (Puranik et al., 2016) using regression-based classifier propose Bug Proneness Index (BPI) algorithm and reported the BPI algorithm has improved performance than multiple linear regression over Eclipse JDT Core datasets. We replicate their study with the aim of validating their finding by running the experiment on Eclipse JDT Core and Eclipse Pde UI datasets and found the Bug Proneness Index (BPI) algorithm has similar performance as multiple linear regression over the two datasets which is not in line with the findings of (Puranik et al., 2016).

3. MOTIVATION

A plethora of approaches have been proposed in literature to solve the problem of bug prediction. The major focus of the existing work has been on proposing new approaches rather than validating the findings of existing approaches. Moreover, the experiments are done on a small set of datasets. However, a general issue in software engineering is that the results emerging from single study cannot be generalized given the limitation of the empirical studies. Replications of the study is important to verify whether the obtained results were produced by chance or do exist in reality (Farooq, 2019). It is therefore, imperative to validate the findings of existing approaches for bug prediction under different conditions especially using same and other datasets before they are adopted in real practice in industry. Moreover, when some problem-solving approach claims to outperform the established problem-solving approaches in a problem domain, it becomes even more important to validate the claims before we endorse and use the results of that problem-solving approach in real practice. One such work (Puranik et al., 2016) reports results which suggest a better performance than

established linear regression. Therefore, it is very important to validate such results. We replicated the same experiment using same algorithm and compared its performance against multiple linear regression. However, we also used another dataset *Eclipse Pde UI* in addition to *Eclipse JDT Core*. The main aim of our work is to validate the findings of the already proposed Bug Proneness Index (BPI) algorithm (Puranik et al., 2016). We did not include a wide set of machine learning approaches in our study for evaluation. Besides being practically unfeasible to include all prediction techniques in a single study, that would have shifted the focus of our study from validation of results to comparison of results.

4. DATA COLLECTION

We use the Eclipse JDT Core and Eclipse Pde UI datasets (D'Ambros et al., 2010), freely available at *bug.inf.usi.ch*. The datasets contain source code metrics both CK metrics (Basili et al., 1996; Chidamber & Kemerer, 1994) and an additional set of OO metrics (number of methods, number of attributes, etc), previous defect data (Kim et al., 2007; Zimmermann et al., 2007) as bug metrics, change metrics (Moser et al., 2008), and complexity of code change as entropy (Hassan, 2009), churn and entropy of source code at the class level (D'Ambros et al., 2010). The choice of using *Eclipse JDT Core* and *Eclipse Pde UI* datasets is reinforced by the fact that the two datasets have same code structure (java code) and are from the same domain, the same company that might be following same bug tracking process. Besides original experiment had used *Eclipse JDT Core* dataset, so in order to verify the results it was imperative for us to use same dataset. In addition, to check the generalization ability of the evaluated algorithm we used *Eclipse Pde UI* dataset. Table 1 and 2 list the metrics we use from the two datasets as independent variables. Moreover, we also use *bugsFoundUntil* metric and *Entropy* metric (D'Ambros et al., 2010) from the two datasets as independent variables. Additionally, to set defect labels of software classes we add another metric *defective*. If the bug count of a software class is greater than 0, *defective* metric is set to a value of TRUE. Otherwise, *defective* metric is set to a value of FALSE.

5. DESCRIPTION OF THE ALGORITHM

In regression analysis, a curve is fitted to the data points in training data (known observations) in such a way that the sum of the squares of distances of data points from the curve is minimized. Based on the nature and number of predictor and dependent variables, an appropriate method of regression can be used. For bug prediction, linear model being the simplest of all the models known has been shown to closely approximate the bug prediction function (D'Ambros et al., 2010). The algorithm we evaluate use linear model for the predictions; but it uses the normalized marginal R square values as weights to the normalized metrics, instead of the actual regression coefficients, to predict the bug proneness index of software classes. The steps in the evaluated algorithm detailed are given in (Puranik et al., 2016)

Table 1. Class level source code metrics

Type	Metric	Description
CK	WMC	Weighted Method Count
CK	DIT	Depth of Inheritance Tree
CK	RFC	Response for Class
CK	NOC	Number of Children
CK	CBO	Coupling Between Objects
CK	LCOM	Lack of Cohesion in Methods
OO	FanIn	Number of other classes that reference the class
OO	FanOut	Number of other classes referenced by the class
OO	NOA	Number of attributes
OO	NOAI	Number of attributes inherited
OO	LOC	Number of lines of code
OO	NOM	Number of methods
OO	NOMI	Number of methods inherited

5.1 Metrics Selection

The first step in any regression analysis is to get a good set of independent variables which can determine the dependent variable to highest possible accuracy. The model performs simple linear regressions of normalized metrics (independent variables) with actual bug count (dependent variable) and determines the coefficient of determination (R^2) for every metric. The coefficient of determination (R^2) is the proportion of variation in the dependent variable accounted for by the independent variable. The model selects top n metrics having high value for the coefficient of determination (R^2) in the linear regression of normalized metric with actual bug count. The choice of n is project dependent, depending on the tolerance for the complexity of the prediction model, and is the number of metrics that will be used in the final prediction model. The experiment (Puranik et al., 2016) reports that the varying number of top n metrics were checked and top 4 metrics were reported to perform better. In our experiments, we also resort to the selection of top 4 metrics.

5.2 Multiple Linear Regression

The next step is to perform multiple linear regression of top n (top 4) metrics with normalized actual bug count and determine the coefficient of determination (R^2) value. The resultant coefficient of determination is used in next step to compute the marginal R square values for top n metrics.

5.3 Compute Marginal R^2 Values

For top n normalized metrics perform multiple linear regressions using (n−1) metrics (excluding the one selected) with the normalized bug count. Determine the coefficient of determination (R^2 s) from the n multiple linear regressions for all top n metrics. For

Table 2. Class level change metrics

Metric	Description
NR	Number of revisions
NREF	Number of refactorings
NFIX	Number of fixes
NAUTH	Number of authors
CHURN	Codechurn

all top n metrics compute the differences in the coefficient of determination (marginal R square) obtained in the current regressions and that obtained in the previous step 5.2. The values of marginal R square values are normalized in the range of 0 to 1 and used as weights to the metrics in the final prediction model. Higher marginal R square value of a metric indicates that the error is more because the metric was left out. This, in turn, means the contribution of the metric in the bug prediction is high. So, the algorithm uses the normalized marginal R square values as weights of the metrics in the linear functions.

5.4 Compute Bug Proneness Index

Once the weights of the top n selected metrics are computed, the next step is to compute the weighted sum of top n normalized metrics. The weighted sum of top n normalized metrics is defined as the *bug proneness index* of a software class. More the index close to 1, more buggy is the software class.

6. RESULTS

We use the Eclipse JDT Core and Eclipse Pde UI datasets (D'Ambros et al., 2010). With a split ratio of 0.5, we perform a random split of the datasets into training and test sets for the evaluation of the model (Puranik et al., 2016) with respect to multiple linear regression model. The split ratio of 0.5 is used to make sure we get an equivalent number of observations in the training and test sets for making the comparison of train and test set performance of the model. The results obtained when the above mentioned 4 steps are executed on the two datasets are shown below.

6.1 Selection of Metrics

When simple linear regressions were performed on the collected metrics (in section 4) with actual bug count, the top n optimal metrics in decreasing order of their coefficient of determination for Eclipse JDT Core and Eclipse Pde UI are shown in table 3 and table 4 respectively. The choice of n is project dependent. In our case, 4 metrics were selected for the two datasets.

6.2 Perform Multiple Regression

When multiple regression is performed on the 4 metrics in table 3 and 4 with actual bug count for Eclipse JDT Core and Pde UI respectively, the resultant coefficient of determination are 0.4872 and 0.63730 respectively.

6.3 Compute Marginal R Square Values

When multiple linear regressions are performed on 4 sets of (n − 1) metrics for the top 4 metrics with actual bug count, for Eclipse JDT Core and Eclipse Pde UI, the R square, marginal R square and normalized marginal R square values (weights) are listed in table 3 and 4 respectively.

6.4 Compute Bug Proneness Index

For all the classes in *Eclipse JDT Core* and *Eclipse Pde UI*, we calculate the weighted sum of normalized metrics for top 4 metrics, in above table 3 and table 4 respectively. Table shows the bug proneness index for two arbitrary chosen classes selected from *Eclipse JDT Core* dataset.

7. DISCUSSION AND COMPARISON OF THE RESULTS

We use multiple linear regression model for drawing a comparison for the effectiveness of the algorithm. In multiple linear regression the relationship between a response variable and predictor variables is approximated through estimation of regression weights, such that sum of squared error is minimized.

Table 3. Eclipse JDT Core Regression results

Metric	R square	R-1 square	Marginal R square	Weight
number of bugs found until	0.45610	0.46060	0.02660	1.00000
number of versions until	0.38070	0.48630	0.00090	0.00000
Entropy	0.36710	0.47860	0.00860	0.29961
LOC	0.34880	0.46560	0.02160	0.80545

Table 4. Eclipse Pde UI Regression results

Metric	R square	R-1 square	Marginal R square	Weight
number of attributes	0.57960	0.54830	0.08900	1.00000
number of bugs found until	0.54290	0.62510	0.01220	0.10280
number of versions until	0.36730	0.63390	0.00340	0.00000
number of fixes until	0.13880	0.63390	0.00340	0.00000

Table 5. Eclipse JDT Core bug proneness index

numberOfBugsFoundUntil	numberOfVersionsUntil	entropy	LOC	actualBugs	bugPronenessIndex
0.04673	0.05932	0.31539	0.01730	0.11111	0.15515
0.17290	0.23588	0.61582	0.06729	0.44444	0.41159

7.1 Estimated Multiple Linear Regression Equation

$$Y = b_{0+} b_1 * x_1 + b_{2*} x_2 + \ldots + b_n * x_n$$

The regression weight 'b_j' is the estimated change in response variable Y, as a result of a unit change in predictor variable 'x_j' when all other predictor variables are constant (Zumel et al., 2014).

The evaluated Bug Proneness Index algorithm use regression as intermediate steps for computation of predictor variable weights as explained in section 5. And the computed predictor variable weights 'w_i' are used in linear function with predictor values 'x_i' for computation of Bug Proneness Index 'I' of a software class.

$$I = w_1 * x_1 + w_{2*} x_2 + \ldots + w_n * x_n$$

We predict the bug score using multiple linear regression model and the bug proneness index model. The predicted bug scores are normalized in the range 0 to 1. To classify the software classes as defective or non-defective threshold value of 0.5 is used. Software classes whose predicted bug score is greater than 0.5 are classified as defective and those with predicted bug score less than or equal to 0.5 are classified as non-defective. We evaluate the performance using Accuracy, Precision, Recall and F-measure as shown in Figure 1. Accuracy is defined as, of the total classifications, how many are correct? Precision is defined as, of the classes classified as buggy, how many are actually buggy? Recall is defined as, of the classes that are actually buggy, how many are classified as buggy? F-measure is the harmonic mean of precision and recall.

The results for Eclipse JDT Core are shown in table 6 for bug proneness index model and table 7 for multiple linear regression model respectively.

The results for Eclipse Pde UI are shown in table 8 for bug proneness index model and table 9 for multiple linear regression model respectively.

The evaluated model achieves an F-measure of 89% and 89% as compared to an F-measure of 89% and 89% achieved using multiple linear regression model, for Eclipse JDT Core train and test sets respectively. Similarly, the algorithm achieves an F-measure of 93% and 92% as compared to an F-measure of 93% and 92% achieved using multiple linear regression model, for Eclipse Pde UI train and test sets respectively. F-measure is a comprehensive, composite measure of the quality of the model. It is evident from the results presented above that the bug proneness index

Figure 1.

$$Accuracy = \frac{No._{bb} + No._{cc}}{No._{bb} + No._{bc} + No._{cb} + No._{cc}} \quad (1)$$

$$Precision_{(bug)} = \frac{No._{bb}}{No._{bb} + No._{cb}} \quad (2)$$

$$Recall_{(bug)} = \frac{No._{bb}}{No._{bb} + No._{bc}} \quad (3)$$

$$F - measure_{(bug)} = \frac{2 * Precision_{(bug)} * Recall_{(bug)}}{Precision_{(bug)} + Recall_{(bug)}} \quad (4)$$

Table 6. Performance for Eclipse JDT Core using Bug proneness index technique

Measure	Overall	Training	Test
Accuracy	81%	80%	81%
Precision	80%	80%	81%
Recall	100%	100%	100%
f-measure	89%	89%	89%

Table 7. Performance for Eclipse JDT Core using multiple linear regression technique

Measure	Overall	Training	Test
Accuracy	81%	80%	81%
Precision	80%	80%	81%
Recall	100%	100%	100%
f-measure	89%	89%	89%

Table 8. Performance for Eclipse Pde UI

Measure	Overall	Training	Test
Accuracy	86%	87%	86%
Precision	86%	87%	86%
Recall	100%	99%	100%
f-measure	93%	93%	92%

Table 9. Performance for Eclipse Pde UI using Bug proneness index technique using Multiple linear regression technique

Measure	Overall	Training	Test
Accuracy	86%	87%	86%
Precision	86%	87%	86%
Recall	100%	99%	100%
f-measure	93%	93%	92%

algorithm performs similar to multiple linear regression model for train and test sets when the F-measure is considered. As shown in table 10, the original experiment reports that the evaluated algorithm performs better than the multiple linear regression. However, as shown in table 10, over two datasets of *Eclipse JDT Core* and *Eclipse Pde UI*, we find that results obtained in our experiment do not conform to the results of the original experiment as evaluated algorithm performs similar to multiple linear regression.

In figure 2, a comparison of metric weights computed by two algorithms is given for Eclipse JDT Core and Eclipse Pde UI training and overall datasets respectively. The metric weights are normalized to a common scale using min-max normalization (Jain & Bhandare, 2011; Patro & Sahu, 2015). From the visualizations, we find the two algorithm compute somewhat similar metric weights for estimation of bug score of software classes. The similarity in the metric weights used by the two algorithms is the reason for similar performance of the two algorithms.

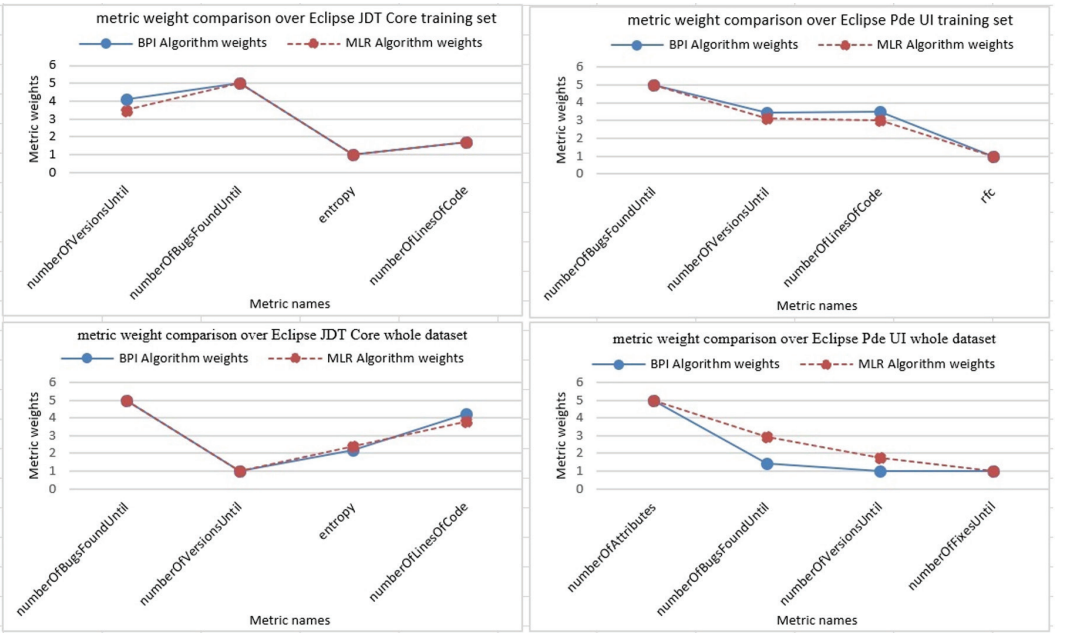
8. THREATS TO VALIDITY

The first threat concern with the implementation of the algorithm. The adoption of any preprocessing techniques or different set of parameters (e.g., number of top n metrics in model training) might adversely effect the performance of the model, hence, influence the conclusions drawn from the performance results. Therefore, we tried to implement the algorithm exactly as illustrated in (Puranik et al., 2016). In addition, other threat concern with the datasets selected for the experiment. In software engineering the results of empirical experiments are restricted to datasets used in the analysis. Therefore, in our evaluation experiment we tried to limit this

Table 10. F-measure comparison of evaluated Bug proneness index (BPI) algorithm and multiple linear regression (MLR)

Measure	Dataset	Experiment	Best performer		
			Training	Test	Overall
F-measure	Eclipse JDT Core	original experiment	BPI	BPI	BPI
		our experiment	BPI, MLR	BPI, MLR	BPI, MLR
	Eclipse Pde UI	our experiment	BPI, MLR	BPI, MLR	BPI, MLR

Figure 2. Metric weights computed by BPI algorithm and MLR algorithm over Eclipse JDT Core and Eclipse Pde UI datasets



threat by selecting the one more dataset of *Eclipse Pde UI* in addition to *Eclipse JDT Core* dataset which was used in the original experiment.

9. CONCLUSION AND FUTURE WORK

Different prediction algorithms are proposed to solve the problem of bug prediction and little attention is given to the validation of experimental results of these prediction algorithms. In this paper, we evaluated the experimental results of one such prediction algorithm (Puranik et al., 2016). We observed that the evaluated algorithm performed no different than the multiple linear regression for both training and test set for *Eclipse JDT Core* and *Eclipse Pde UI* datasets. The comparison of the algorithm results with that of multiple linear regression shows that the composite measure of precision and recall (F-measure) is same over datasets of Eclipse JDT Core and Eclipse Pde UI. Similarly, the model achieves same performance as multiple linear regression when performance measures like accuracy, precision, and recall are considered. Therefore, we recommend to validate the experimental results of the proposed approaches for bug prediction before their application in the real practice. Moreover, the experiment can be replicated under different conditions, using diverse measures and datasets to create an effective knowledge base.

REFERENCES

Altman, D. G., & Royston, P. (2006). The cost of dichotomising continuous variables. *BMJ (Clinical Research Ed.)*, *332*(7549), 1080. doi:10.1136/bmj.332.7549.1080 PMID:16675816

Arar, Ö. F., & Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, *33*, 263–277. doi:10.1016/j.asoc.2015.04.045

Austin, P. C., & Brunner, L. J. (2004). Inflation of the type I error rate when a continuous confounding variable is categorized in logistic regression analyses. *Statistics in Medicine*, *23*(7), 1159–1178. doi:10.1002/sim.1687 PMID:15057884

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, *22*(10), 751–761. doi:10.1109/32.544352

Bernstein, A., Ekanayake, J., & Pinzger, M. (2007). Improving defect prediction using temporal features and non linear models. *Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting*, 11–18. doi:10.1145/1294948.1294953

Cataldo, M., Mockus, A., Roberts, J. A., & Herbsleb, J. D. (2009). Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, *35*(6), 864–878. doi:10.1109/TSE.2009.42

Ceylan, E., Kutlubay, F. O., & Bener, A. B. (2006). Software defect identification using machine learning techniques. *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, 240–247. doi:10.1109/EUROMICRO.2006.56

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476–493. doi:10.1109/32.295895

Cohen, J. (1983). The cost of dichotomization. *Applied Psychological Measurement*, *7*(3), 249–253. doi:10.1177/014662168300700301

D'Ambros, M., Lanza, M., & Robbes, R. (2010). An extensive comparison of bug prediction approaches. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010),* 31–41.

Di Nucci, D., Palomba, F., De Rosa, G., Bavota, G., Oliveto, R., & De Lucia, A. (2017). A developer centered bug prediction model. *IEEE Transactions on Software Engineering*, *44*(1), 5–24. doi:10.1109/TSE.2017.2659747

Di Nucci, D., Palomba, F., Siravo, S., Bavota, G., Oliveto, R., & De Lucia, A. (2015). On the role of developer's scattered changes in bug prediction. *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 241–250. doi:10.1109/ICSM.2015.7332470

El Emam, K., Melo, W., & Machado, J. C. (2001). The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software*, *56*(1), 63–75. doi:10.1016/S0164-1212(00)00086-8

Fagan, M. (2002). Design and code inspections to reduce errors in program development. In *Software pioneers* (pp. 575–607). Springer. doi:10.1007/978-3-642-59412-0_35

Farooq, S. U. (2019). Gap between academia and industry: A case of empirical evaluation of three software testing methods. *International Journal of System Assurance Engineering and Management*, *10*(6), 1487–1504.

Felix, E. A., & Lee, S. P. (2017). Integrated Approach to Software Defect Prediction. *IEEE Access: Practical Innovations, Open Solutions*, *5*, 21524–21547. doi:10.1109/ACCESS.2017.2759180

Graves, T. L., Karr, A. F., Marron, J. S., & Siy, H. (2000). Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, *26*(7), 653–661. doi:10.1109/32.859533

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, *31*(10), 897–910. doi:10.1109/TSE.2005.112

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2011). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304. doi:10.1109/TSE.2011.103

Hassan, A. E. (2009). Predicting faults using the complexity of code changes. *2009 IEEE 31st International Conference on Software Engineering*, 78–88.

Hassan, A. E., & Holt, R. C. (2005). The top ten list: Dynamic fault prediction. *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 263–272. doi:10.1109/ICSM.2005.91

Herzig, K. (2014). Using pre-release test failures to build early post-release defect prediction models. *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 300–311.

Hou, C., Nie, F., Yi, D., & Wu, Y. (2012). Efficient image classification via multiple rank regression. *IEEE Transactions on Image Processing*, *22*(1), 340–352. PMID:22910112

Jain, Y. K., & Bhandare, S. K. (2011). Min max normalization based data perturbation method for privacy protection. *International Journal of Computer & Communication Technology*, *2*(8), 45–50.

Joshi, H., Zhang, C., Ramaswamy, S., & Bayrak, C. (2007). Local and global recency weighting approach to bug prediction. *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007),* 33. doi:10.1109/MSR.2007.17

Kim, S., Zimmermann, T., Whitehead, E. J. Jr, & Zeller, A. (2007). Predicting faults from cached history. *29th International Conference on Software Engineering (ICSE'07)*, 489–498. doi:10.1109/ICSE.2007.66

Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, *58*, 388–402. doi:10.1016/j.infsof.2014.07.005

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, *34*(4), 485–496. doi:10.1109/TSE.2008.35

Madeyski, L., & Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? An empirical study. *Software Quality Journal*, *23*(3), 393–422. doi:10.1007/s11219-014-9241-7

Minku, L. L., & Yao, X. (2013). Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, *55*(8), 1512–1528. doi:10.1016/j.infsof.2012.09.012

Mnkandla, E., & Mpofu, B. (2016). Software defect prediction using process metrics elasticsearch engine case study. *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 254–260. doi:10.1109/ICACCE.2016.8073757

Mockus, A. (2010). Organizational volatility and its effects on software defects. *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 117–126. doi:10.1145/1882291.1882311

Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *Proceedings of the 30th International Conference on Software Engineering*, 181–190. doi:10.1145/1368088.1368114

Mullen, R. E., & Gokhale, S. S. (2005). Software defect rediscoveries: a discrete lognormal model. *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05),* 10.

Nagappan, N., & Ball, T. (2005a). Static analysis tools as early indicators of pre-release defect density. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.,* 580–586.

Nagappan, N., & Ball, T. (2005b). Use of relative code churn measures to predict system defect density. *Proceedings of the 27th International Conference on Software Engineering*, 284–292.

Nagappan, N., Ball, T., & Zeller, A. (2006). Mining metrics to predict component failures. *Proceedings of the 28th International Conference on Software Engineering*, 452–461.

Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., & Murphy, B. (2010). Change bursts as defect predictors. *2010 IEEE 21st International Symposium on Software Reliability Engineering*, 309–318.

Patro, S., & Sahu, K. K. (2015). *Normalization: A preprocessing stage*. ArXiv Preprint ArXiv:1503.06462

Petrić, J., Bowes, D., Hall, T., Christianson, B., & Baddoo, N. (2016). Building an ensemble for software defect prediction based on diversity selection. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10. doi:10.1145/2961111.2962610

Puranik, S., Deshpande, P., & Chandrasekaran, K. (2016). A novel machine learning approach for bug prediction. *Procedia Computer Science*, *93*, 924–930. doi:10.1016/j.procs.2016.07.271

Rahman, F., & Devanbu, P. (2013). How, and why, process metrics are better. *2013 35th International Conference on Software Engineering (ICSE),* 432–441.

Rajbahadur, G. K., Wang, S., Kamei, Y., & Hassan, A. E. (2017). The impact of using regression models to build defect classifiers. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR),* 135–145.

Raman, B., & Ioerger, T. R. (2003). *Enhancing learning using feature and example selection*. Texas A&M University.

Ratzinger, J., Sigmund, T., & Gall, H. C. (2008). On the relation of refactorings and software defect prediction. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 35–38. doi:10.1145/1370750.1370759

Royston, P., Altman, D. G., & Sauerbrei, W. (2006). Dichotomizing continuous predictors in multiple regression: A bad idea. *Statistics in Medicine*, *25*(1), 127–141. doi:10.1002/sim.2331 PMID:16217841

Shivaji, S., Whitehead, E. J. Jr, Akella, R., & Kim, S. (2009). Reducing features to improve bug prediction. *2009 IEEE/ACM International Conference on Automated Software Engineering*, 600–604. doi:10.1109/ASE.2009.76

Shull, F., Basili, V., Boehm, B., Brown, A. W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., & Zelkowitz, M. (2002). What we have learned about fighting defects. *Proceedings Eighth IEEE Symposium on Software Metrics*, 249–258. doi:10.1109/METRIC.2002.1011343

Sun, Z., Song, Q., & Zhu, X. (2012). Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *42*(6), 1806–1817. doi:10.1109/TSMCC.2012.2226152

Van Rysselberghe, F. (2008). *Studying Historic Change Operations: Techniques and Observations*. Universiteit Antwerpen, Faculteit Wetenschappen.

Witten, I. H., & Frank, E. (2005). Data mining: practical machine learning tools and techniques (2nd ed.). Morgan Kaufmann.

Xiang, S., Nie, F., & Zhang, C. (2010). Semi-supervised classification via local spline regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(11), 2039–2053. doi:10.1109/TPAMI.2010.35 PMID:20847392

Xu, Z., Liu, J., Yang, Z., An, G., & Jia, X. (2016). The impact of feature selection on defect prediction performance: An empirical comparison. *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE),* 309–320.

Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for eclipse. *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007),* 9. doi:10.1109/PROMISE.2007.10

Zumel, N., Mount, J., & Porzak, J. (2014). *Practical data science with R*. Manning Shelter Island.

Nayeem Ahmad Bhat is a PhD student at Department of Computer Science, North Campus of University of Kashmir. His doctoral research focuses investigating cross project defect prediction mechanism. The research specifically focuses on developing effective models for cross project defect prediction using machine learning approach.

Sheikh Umar Farooq is an Assistant Professor at Department of Computer Sciences, North Campus of University of Kashmir. He received his PhD from the University of Kashmir. His research interests include Empirical Software Testing, Software Defect Prediction and Software Engineering Education.

International Journal of Open Source Software and Processes
Volume 11 • Issue 3 • July-September 2020

Open-Source Essential Protein Prediction Model by Integrating Chi-Square and Support Vector Machine

S. R. Mani Sekhar, REVA University, India & Ramaiah Institute of Technology, India
Siddesh G. M., Ramaiah Institute of Technology, India
Sunilkumar S. Manvi, REVA University, India

ABSTRACT

Identification and analysis of protein play a vital role in drug design and disease prediction. There are several open-source applications that have been developed for identifying essential proteins which are based on biological or topological features. These techniques infer the possibility of proteins to be essential by using the network topology and feature selection, which can ignore some of the features to reduce the complexity and, subsequently, results in less accuracy. In the paper, the authors have used selenium driver to scrap the dataset. Later, the authors integrated the chi-square method with support vector machine for the prediction of essential proteins in baker yeast. Here, chi-square is a test of dissimilarity used for altering the record, and afterward, the support vector machine is used to classify the test dataset. The results show that the proposed model Chi-SVM model achieves an accuracy of 99.56%, whereas BC and CC achieved an accuracy of 84.0% and 86.0%. Finally, the proposed model is validated using Statistical performance measures such as PPA, NPA, SA, and STA.

KEYWORDS

Amino Acid, Analytics, Baker Yeast, Chi-Square, Essential Protein, Machine Learning, Protein, Protein-Protein Interaction (PPI), Support Vector Machine (SVM), Yeast

DOI: 10.4018/IJOSSP.2020070103

1. INTRODUCTION

In the current computational domain, proteomics act as a sub-branch of bioinformatics, which studies the structure and functions of proteins. It also helps in identifying the procedure and encoding order used in the analysis of the large scale data set. Free and open-source software (FOSS) allows the developer to use, change, and distribute software without significant limitations. However, it also provides a complete software package to users freely. Now a day's requirement of open source software in bioinformatics and chem-informatics etc is increasing rapidly. cPath (Cerami, Bader, Gross, & Sander, 2006) an open-source software contains a web interface for gathering, storing, and querying bio-informatics data. It also supports the build-in mapping service and helps to link to external resources (Breitwieser & Colinge, 2013). Developed a tool to analysis proteins localization and processing of Tandem Mass Tags peptides. In addition it also supports statistical computation with user friendly reports generation. For study of protein function and sequence (Deng, Yuan, Huang, & Wang, 2013) created a SFAPS module by using spectrum function. This function helps in converting numerical sequences into the characteristic frequency of the protein interface. Subsequently helps in analyze sequence of proteins. ATGme (Daniel et al., 2015) designed an open-source web-based tool for optimizing the protein sequence by classifies organisms into rare and very high rare codons. After that, help the user to generate an individual custom optimized DNA sequence.

Proteins are the best adaptable macromolecules in living creatures and also play important roles in biological procedures. They are made of individual or several chains of amino acid residues. An amino acid is made up of an organic composite that has an essential carbon atom, called an alpha carbon (Cα). The bond is consists of four valences with a link to the carboxylic group, hydrogen atom, an amino group, and an adjacent chain. The amino acids are the strong structured chunks of protein. This sequence of amino acids in a given polymer helps in identifying protein structure and protein function. It can also be called as a polypeptide, though a small chain of amino acids. This chained bond starts with the amino cluster and ends with the carboxyl cluster. According to Bioinformatics researcher DNA, RNA, Protein, and other molecules do not work separately. They can be linked together to perform a specific biological action. Interaction is a process when two molecules linked to perform a certain function. These type of interaction are mainly divided into four groups by considering their molecule types as shown below

- *Protein-protein interactions*: collaboration between proteins to derive biological procedures.
- *Gene regulatory interactions*: interaction of genetic data to normalize protein expression level.
- *Metabolic interactions*: create support between enzyme proteins and helps in changing a substrate cell into product cell through numerous catalysis responses.

- *RNA-DNA interactions:* provide collaboration among RNA-RNA or RNA-DNA connections and also plays a vital role in critical diseases.

PPI performance numerous functions in a different cell, include metabolic pathways, DNA, replication, transduction, and transcription etc. In a PPI network, a vertical position signifies the proteins, and the linking line shows the communication link between proteins. This network can be noticed as a graph. Thus the closely connected protein clusters can be computed efficiently.

Essential Proteins are made up of a set of minimal genome, which can support cell survival with basic requirements. A living creature cannot breed and live without them (Kamath et al., 2003; Wang, X Peng, W Peng, & Wu, 2014). Proteins are mostly classified into two types Essential proteins and non-essential proteins. Essential proteins are the protein which cannot be produced by living organism and need to be taken as additional in the form of nutrition, whereas non-essential proteins can be produce and consume by living organism directly. Out of 20 amino acids, only nine amino acids are essential for living organisms they are valine, threonine, lysine, leucine, methionine, isoleucine, histidine, phenylalanine, and tryptophan (Young, 1994). Analysis and Identifying of essential proteins play a major focus in genomic research. Previous works focus only on the prediction of essential proteins either by using topological features or biological features, meanwhile some used feature selection methods to predict the essentiality of the protein by ignoring some of the features, thereafter, resulting in a reduction in prediction accuracy.

In the proposed experiment, Chi-square (Pearson, 1900) and Support Vector Mechanism (Cirtes, 1995) has been employed to predict essential proteins using computational techniques. The bakers yeast - saccharomyces ccrovisiao dataset (Snel, 2002) is first filtered through the Chi-square method, which is a test of dissimilarity to get a probability of dissimilarity for each protein, which is then used to rank the dataset. The subset of this ranked dataset is then used for training the machine for classification. Support vector machine (SVM) is being used for the classification of test data. It classifies each protein in one of the three class's namely essential, non-essential, or undefined (essential for some interaction and non-essential for others).

The rest of the paper is organized as follows: Section II presents a survey on different methods used to predict essential proteins with their drawbacks. Section III discuss the architecture of the proposed model. Subsequently, Chi-square and SVM algorithms implementation is discussed in Section IV. Furthermore, Section V illustrates the dataset used in computation and discussion of results. Finally, section VI discusses the conclusion and future work.

2. LITERATURE REVIEW

In (Zhang, Ruan, Gao, & Wu, 2019), uses a statistic methodology with gene information and subcellular information for essential protein prediction. Subsequently, greedy methodology is used for feature optimization. Whereas (Patel & Shah, 2013) proposed

a linear and non-linear support vector machine methodology for protein secondary structure prediction. They used the multi-class SVM method to increase the prediction accuracy. Here the focus is only on SVM. As the PPI dataset can have noise, in the proposed work author has incorporated filtering techniques like Chi-Square for more effective protein prediction.

In this work (Jiang et al., 2015) projected a computational method called integrated edge weights (IEW) for predicting the essential protein-based on protein-protein interactions (PPI). They use these methods on three organisms, namely Saccharomyces cerevisiae (yeast), escherichia coli (E. coli), and caenorhabditiselegans (C. elegans). Here the computation is purely based on edge weight. In essential protein prediction, (Sekhar, Matt, Manvi, Gopalalyenger, 2019) developed a computation model that uses two different methods mean weighted average and recursive feature elimination in prediction of essential and non-essential proteins and they also showed that which method can perform better than other. The accuracy can be further increased by incorporating supervise techniques.

(Kim, Tsoy, Persson, & Grailhe, 2017) developed open-source, user-friendly software called FLIM-FRET, which helps in an extract of essential information of protein-protein interaction (PPI) for life span based FRET analysis by using state fluorescence intensity information. They mainly focused on the PPI data. The OpenPPI_predictor tool (Pedamallu, & Posfai, 2010), helps in the prediction of human parasite roundworm by creating networks of protein-protein interface for specific genomes data by including connected orthologous interactive network information. For computation, they used Caenorhabditis genomics data of interactive network data. Genetic code analysis toolkit (Kraljic, Strüngmann, Fimmel, & Gumbel, 2018) computes the FASTA files for analysis nucleotide orders structure. It also provides a user-friendly workspace for the computation of genetic code and multi-sequence processing.

A Local Interaction Density (LID) method (Qi & Luo, 2016) uses topology centrality measures to predict essential proteins. LID computes the density of interaction among the neighbors of the protein, and the corresponding listed is generated in descending. (Jeong, Oltvai, & Barabasi, 2003) designed a predictive model to discover the effect of gene function in the presence of phenotypic, it allows forecasting the removal phenotype of untested yeast genes. The study shows that mRNA expression profile is concurrently associated with phenotypic of a gene. Whereas ECC (Ren, Wang, Li, Wang, & Liu, 2011) uses the process of joining subgraph centrality and protein complexes data for essential protein prediction on the yeast PPI network. Prediction of essential protein can be further increased by introducing a machine learning algorithm with more integrated proteins dataset.

3. PROPOSED SOLUTION

In the propose methodology, prediction of essential proteins is based on the link scores between each protein (Von et al., 2005), PPI network dataset of Baker's yeast

is used in prediction. The proposed work uses selenium driver to scrap the dataset and has implemented using Python language. Later different libraries has been used such as NumPy, SciPy, Itertools and Matplotlib.pylot etc. some of these libraries are briefed below:

- NumPy: It contains several features such as linear algebra capabilities and is used to execute several matrix operations. It is used to perform mathematical operations on arrays in the proposed work.
- SciPy: It includes modules for integration, linear algebra, statistics. It is used to perform statics and several complex mathematical operations on the data in the proposed work
- Itertools: It is used to count the number of occurrence of every distinct item in the array.
- Matplotlib.pyplot: provides a MATLAB-like plotting framework. Pylab combines pyplot with numpy into a single namespace which is convenient for interactive work. It is used to plot an accuracy-percentage and time-complexity graphs.

The proposed solution is classified into following steps:

- The dataset obtained from STRING Website (Snel, 2002) is converted to a matrix, where each matrix element represents the link score of two labeled proteins.
- The expected value matrix is constructed from the scores.
- The expected and observed value matrix is then passed to Chi-square (McHugh, 2013) matrix to obtain probability of dissimilarity.
- The probability of dissimilarity is then used to rank the proteins in order of dissimilarity.
- The machine is then trained with these ranked proteins for classification.
- The test dataset is then classified by the SVM (Cortes & Vapnik, 1995) to one of the three classes namely essential, non-essential or undefined proteins

Figure 1, demonstrates the system architecture, it is classified into two modules, filter module and classification module. Filter module is used to filter the unnecessary data, whereas SVM is used for classification.

Figure 2, illustrates the framework of Chi-SVM module. The protein protein Interactions (PPI) network dataset obtained from string-db organization used to compute the link score between proteins and used to construct a square matrix of link scores. Later the selenium driver is used to scrap the dataset and subsequently generate the supervised and test data.

Chi-square method is used to filter the dataset and to rank the data based on probability of dissimilarity. From this ranked dataset, a training dataset of desired number of tuples is created. Furthermore, machine is trained with created train dataset for classification, here SVM is used to classifying the test dataset to one of the three class's namely essential, non-essential or undefined data.

Figure 1. Proposed system architecture of Chi-SVM Model

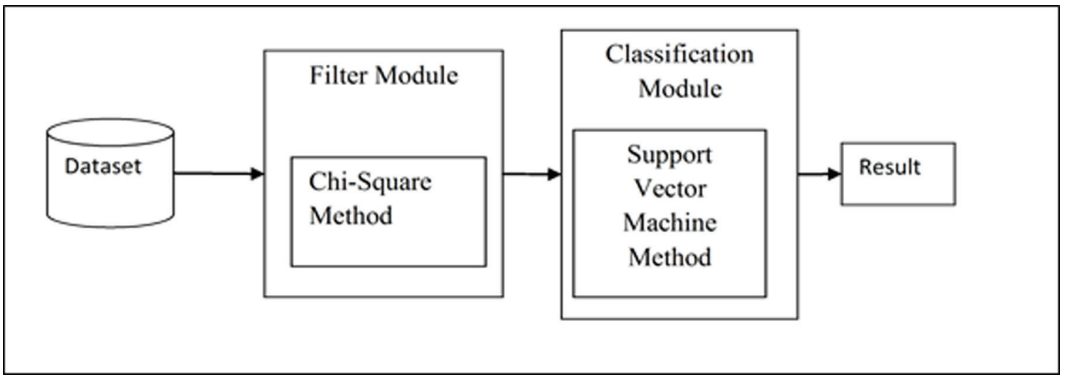


Figure 3, discuss the process of dataset filtering using Chi-square method. Here the input (Yates, 1934) to Chi-square method is a set of observed values taken from PPI network dataset and expected values is calculated from rest of the tuple. Furthermore the test method uses this value to return a probability set which contains the probability of dissimilarity with respect to other tuple. Thereafter this probability is further used to rank the proteins according to its uniqueness, this ranked set is then used to train the machine for classification.

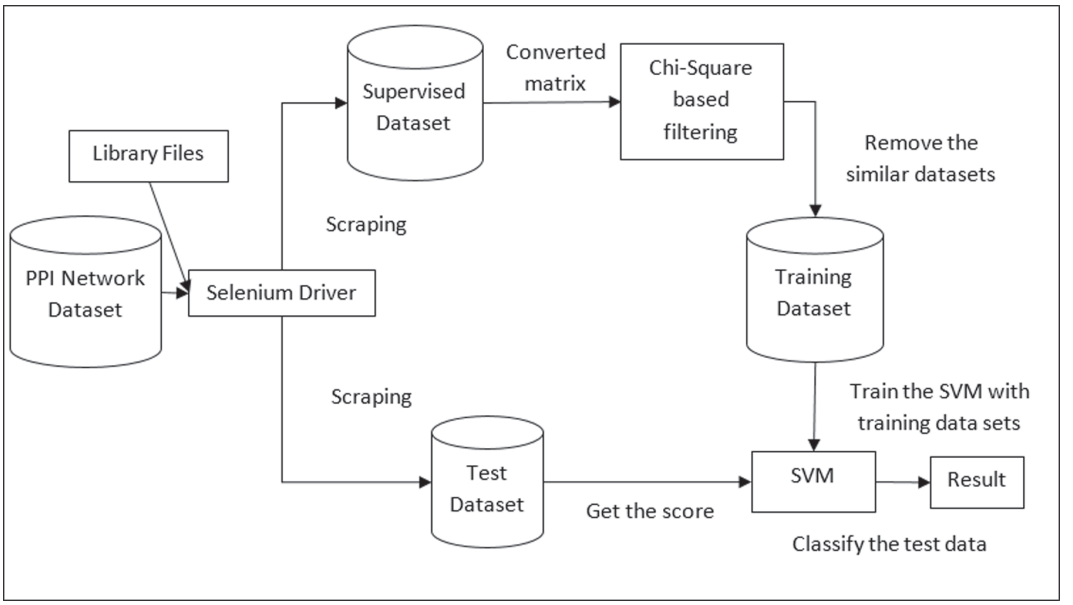
4. IMPLEMENTATION

In the proposed approach, Chi-SVM methodology is divided into two modules namely Chi-square (McHugh, 2013) based filtering and SVM (Cortes & Vapnik, 1995) based classification.

***Chi-Square based filter module*,** filtering of the training dataset is based on the probability of the dissimilarity. The module takes the dataset from the PPI network dataset. This method then converts the link score between proteins to a square matrix with values representing the link score between two proteins. The converted matrix is then examined for any zero tuples if any it removes that tuple as the Chi-square method works for only non-zero values. The obtained matrix acts as a matrix of observed values, while the expected matrix is constructed for each element in the observed matrix. Both these matrixes are passed to the Chi-square method as a parameter to get the probability of the dissimilarity matrix. This probability matrix is used to rank the tuples, which are most dissimilar. Based on input, the method creates a training dataset with most dissimilar training examples.

Algorithm 1 summarizes the detail of dataset filtering using the Chi-square method. Initially, the matrix of size n * n is supplied as input to the filtering algorithm. In the first three lines, we compute the row-wise sum value for the given input data. Similarly, line number four to six used for computation of column-wise sum. Line number 7 performs the overall sum for the given data. Subsequently, from line number 8 to 12 used to generate expected values for each n * n matrix. Meanwhile, from line

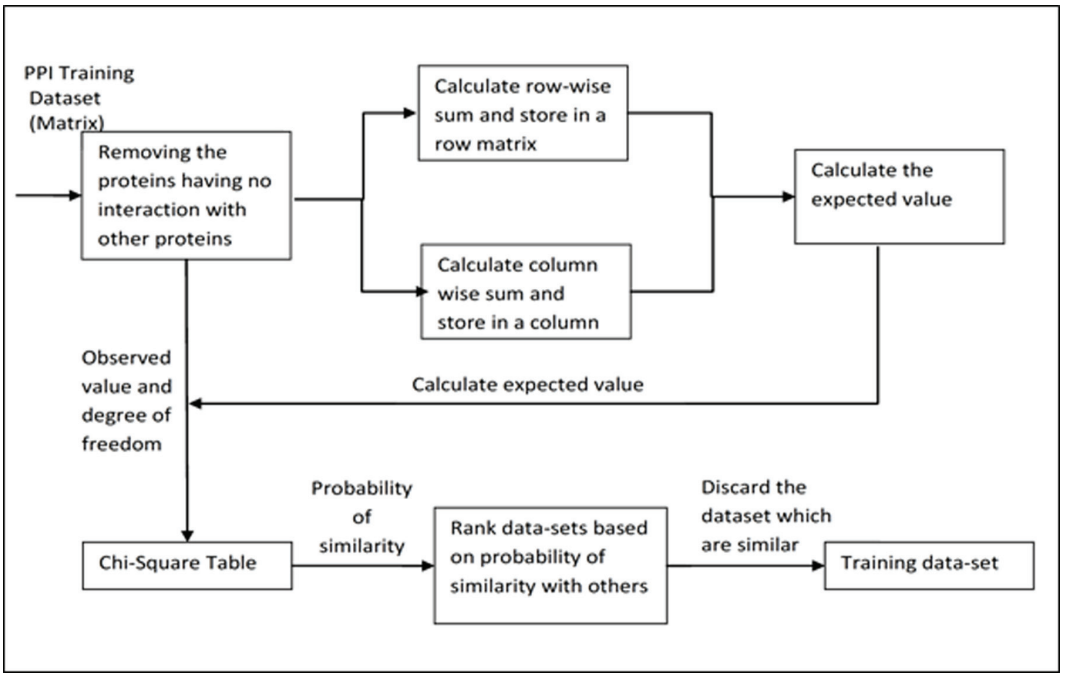
Figure 2. Chi-SVM Framework for essential protein prediction



number 13 to 15 is used for computation of chi-square measure for each row. Then we compute the degree of freedom using line number 16. Furthermore, from line number 17 to 21, calculate the probability of similarity for each row with other. Line number 22 to 24 appends the probability of each protein with its score. Arranging and rank of proteins is based on the probability of similarity, as shown in line number 25 to 27. Finally, in the remaining lines, we calculate the number of necessary tuples "t".

Algorithm 1: Chi-Square Based Filtering
Input: Matrix of size n * n
Output: Reduced Matrix M of size n1 * n1
(1) **for** each_row r ϵ Row [1, n]
(2) RSum[r] $= \sum_{i=1}^{n} M[r][i]$
(3) **endfor**
(4) **for** each_column c ϵ Column [1, n] **do**
(5) RSum [c] = $\sum_{i=1}^{n} M[i][c]$
(6) **endfor**
(7) Total= $\sum_{i=1}^{n} RSum[i]$
(8) **for** each_row r ϵ Row [1, n] **do**
(9) **for** each_column c ϵ Column [1, n] **do**
(10) E [r][c] = (RSum [r] * CSum[c]) / Total
(11) **endfor**
(12) **endfor**
(13) **for** each_row e ϵ Row [1, n] **do**

Figure 3. Chi-Square based PPI data filtering



(14) Chi [ex]= $\sum_{i=1}^{n} \frac{\left(M\left[ex\right]\left[i\right]-E\left[ex\right]\left[i\right]\right)^2}{E\left[ex\right]\left[i\right]}$

(15) **endfor**
(16) DF(Degree_of_freedom)=(n-1) * (n-1)
(17) **for** each_rowiϵ Row [1,n] **do**
(18) **for** each_column j ϵ Column [1, n] **do**
(19) Prob[i][j] = $X^2 table\left(Chi\left[i\right]+Chi\left[j\right],DF\right)$
(20) **endfor**
(21) **endfor**
(22) **for** each_row j ϵ Prob **do**
(23) M = M.append (Prob)
(24) **endfor**
(25) **for** each_row k ϵ Prob **do**
(26) M= M [M [:,-1].argsort]
(27) **end for**
(28) Count = 0
(29) **for** each_row r ϵ Row [1, n], r ϵ [1, n] **do**
(30) **If** Count ^{1} 0 **then**
(31) M[ ][ ](n1 * n1) = M [1: t] (n * n)
(32) **else**

```
(33)        M[ ][ ](n1 * n1)= ∅
(34)  endif
(35)  endfor
```

Classification Module is used to classify proteins with respect to essential, non-essential and undefined proteins by using SVM (Cortes & Vapnik, 1995), the obtained training dataset from Chi-Square based filter is used to train the SVM for classification this test data is then passed to SVM module, and here SVM classifies the proteins based on the scores of the test data.

The prediction efficiency of SVM purely depends on the hyper plane created from the training dataset, if the training examples are as such which clearly separates the classes, the hyper plane width will reduce and if it does not hyperplane width will increase. The efficiency of classification purely depends on the hyper plane, more the hyper plane width more the number of noise point and hence lesser efficiency of classification, with less hyper plane width classification efficiency increases. Here the hyperplane g(x), is needed to be calculated as shown in equation 1.

$$g\left(x\right) = w_t \, \mathrm{x} + w_0 \tag{1}$$

Meanwhile, w_t is a vector, perpendicular to hyperplane and x_t is input vector which needs to be classified.

Classification of proteins is based on following condition:

- If $r_t = 1$, then $w_t \, \mathrm{x} + w_0$ will be between 0 and 1.
- If $r_t = 0$, then $w_t \, \mathrm{x} + w_0$ will be equal to 0.
- If $r_t = -1$, then $w_t \, \mathrm{x} + w_0$ will be between-1 and 0.

The value of g(x) defines the class in which input x_t belongs, if this value is greater than zero it belongs to essential protein class and if the value is zero it labeled in non-essential protein class and if minus one its belong to undefined class.

Algorithm 2, summarize the detail classification process using SVM. Initially pruned dataset is supplied as an input to the classifier. The sample vector value is supplied for classification process. In line number 3 if computed value for r^t is +1 then the corresponding vector value will be consider as essential protein, whereas if the computed value will be -1 then it will be consider as a non-essential proteins. Subsequently the remaining lines classify the data into essential and non-essential vector class.

```
Algorithm 2: Support vector machine based classification
Input: Pruned dataset after chi-square based pruning
Output: classification of proteins
(1)   Sample X = (x^t, r^t)
(2)   x^t = sample vector
```

(3) $$r^t = \begin{cases} +1 \ if essential protein \\ -1 \ if non-essential protein \end{cases}$$

(4) w^t = vector perpendicular to hyperplane

(5) $g(x) \leftarrow w^t x + w_0$

(6) **if** $r^t = 1$ **then**

(7) $w^t x^t + w_0 \geq 1$

(8) **endif**

(9) **if** $r^t = -1$ **then**

(10) $w^t x^t + w_0 \leq -1$

(11) **endif**

(12) **if** g(x) > 0 **then**

(13) Choose C_1

(14) **else**

(15) Choose C_2

(16) **endif**

Here C1 is the class of vector which represents essential proteins whereas C2 is the class of vector which represents non-essential proteins

5. RESULT AND DISCUSSION

The proposed model uses a PPI network dataset of Baker's Yeast (Saccharomyces cerevisiae), which contains 5,976 proteins with 2,10,914 interactions downloaded from string-db.org database (snel, 2002). This PPI-network dataset, has 2031 organism's interaction dataset having 9.6 mio proteins with 1380 mio interactions. Table 1, shows the snapshot of the sample dataset. It consists of five columns. Column first and second display the names of the proteins, here the columns are named as "Item_id_A ", " Item_id_B ". Third column so-called "mode action" represents the way of interaction like binding, catalyses, react etc between proteins. The fourth column termed as "a_ is_acting" represents whether 'a' is the actor or not. If it does, then it takes the value of 1 else 0. The last column named "score" display the score values, by considering various evidences, and confidence of each interaction with every considered value contributes to the score calculation of each PPI.

The score are computed by considering various evidences. The evidences are listed below with the number of interactions considered for each evidences.

- Gene neighborhood, normal (7,851 interactions)
- Gene neighborhood, transferred (11177 interactions)
- Gene fusion (514 interactions)
- Gene co-occurrence (35497 interactions)
- Gene co-expression, normal (12376 interactions)
- Gene co-expression, transferred (3154 interactions)
- Experiments / biochemistry, normal (5301 interactions)

Table 1. Sample PPI Network Dataset- Baker Yeast (Snel, 2002)

ItemIdA	ItemIdB	mode action	aIsActing	Score
4932.Q0032	4932.YNL182C	Binding	0	683
4932.Q0045	4932.Q0050	Binding	0	900
4932.Q0045	4932.Q0085	Binding	0	735
4932.Q0045	4932.YEL024W	Binding	0	839
4932.Q0045	4932.YEL039C	Catalysis	1	538
4932.Q0045	4932.YGR033C	Binding	0	749
4932.Q0045	4932.YGR183C	Binding	0	720
4932.Q0045	4932.YJR048W	Catalysis	1	538
4932.Q0045	4932.YKR016W	Binding	0	538
4932.Q0045	4932.YLR038C	Binding	0	900
4932.Q0045	4932.YLR067C	Expression	0	800
4932.Q0045	4932.YJR048W	Catalysis	1	538
4932.Q0045	4932.YDR231C	Binding	0	730
4932.Q0045	4932.YDR529C	Binding	0	720
4932.Q0045	4932.YDR529C	Binding	0	746
4932.Q0045	4932.YEL024W	Binding	0	720
4932.Q0045	4932.YEL024W	Binding	0	839

- Experiments / biochemistry, transferred (4113 interactions)
- Annotated pathways, normal (6726 interactions)
- Annotated pathways, transferred (1727 interactions)
- Text mining, normal (27445 interactions)
- Text mining, transferred (7119 interactions)
- Combined-score, total (210914 interactions)

The input to Chi-Square method is a square matrix of observed values. The score shown in table 1, is used to convert the interaction between each protein into matrix as shown in table 2. Here first row and first column showed the proteins names with their corresponding values presented in different columns. If there is an interaction equivalent score value will get updated or else zero will get assign to it.

After the machine is trained for classification, it starts classifying each protein in test dataset to one of two class's namely essential, non-essential or undefined proteins. Table 3, show the accuracy of prediction of proteins based on different training dataset, it is divided into two columns, first column list the different training dataset used for computation whereas second column list the corresponding accuracy of protein prediction in percentage. For 100 training dataset the accuracy of prediction of proposed model is 74.06%, BC is 60% and for CC is 61%. For 300 and 600 training datasets the accuracy is 75.74% & 78.79% for Chi-SVM, 62%, 63% for BC and for CC is 62% & 62.5% . When the training dataset is of 1800 size the prediction is 83.58%

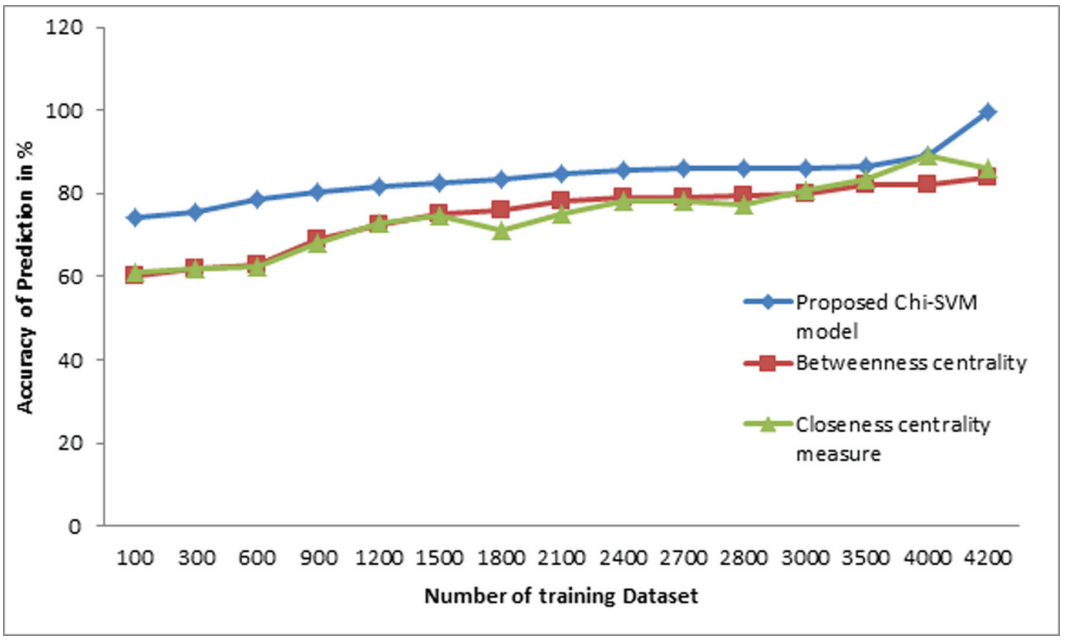
Table 2. Sample Matrix form of PPI Network Dataset

Protein	4932. Q0032	4932. Q0045	4932. Q0050	4932. Q0055	4932. Q0060	4932. Q0065	4932. Q0070	4932. Q0075	4932. Q0080	4932. Q0085
4932. Q0045	0	0	0	0	0	0	0	0	0	0
4932. Q0050	0	800	900	0	0	0	0	0	0	0
4932. Q0055	0	900	800	0	0	0	0	0	0	0
4932. Q0060	0	0	0	800	0	0	0	0	0	0
4932. Q0065	0	0	0	0	0	0	0	0	0	0
4932. Q0070	0	0	0	0	0	0	0	0	0	0
4932. Q0075	0	0	0	0	0	0	0	0	0	0
4932. Q0080	0	0	0	0	0	0	0	0	0	1000
4932. Q0085	0	735	0	0	0	0	0	0	1000	0

Table 3. Predicted essential proteins based on Chi-SVM model, BC, and CC Measures

Number of training dataset	Accuracy of predicted (%)		
	Chi-SVM model	Betweenness centrality measure	Closeness centrality
100	74.06	60	61
300	75.74	62	62
600	78.79	63	62.5
900	80.15	69	68
1200	81.76	72.5	73
1500	82.74	75	74.5
1800	83.58	75.8	71
2100	84.67	78.2	75
2400	85.5	79	78
2700	85.93	79	78
2800	86.16	79.5	77.5
3000	86.24	80	81
3500	86.64	82	83.5
4000	89.19	82	89
4200	99.56	84	86

Figure 4. Predicted accuracy for proposed Chi-SVM Model, BC and CC in percentage



whereas for BC is 75.8% and for CC is 71%. Furthermore it shows that chi-SVM gives an accuracy of 86.64% whereas for BC & CC is 82% % 83.5 when the machine was trained with 3500 (58.56% of dataset) tuples. Finally the accuracy is 99.56% for Chi-SVM model whereas 84% for BC & 86% for CC when the machine was trained with 4200 (70.28%) tuple(s).

The graph shown in figure 4, tells the accuracy of classification on test dataset with different number of training tuples for both Chi-SVM model, Betweenness centrality (BC) measure (Freeman, 1977; Joy, Brock, Ingber, & Huang, 2005) and Closeness centrality (G Li, M Li, Wang, Y Li, & Pan, 2018). BC & CC value has been computed by using jackknife methodology (Holman, Davis, Foster, Carlow, & Kumar, 2009). Here x-axis of graph depicts the number of dataset used as training tuples whereas the y-axis of the graph depicts the percentage of test data classified correctly.

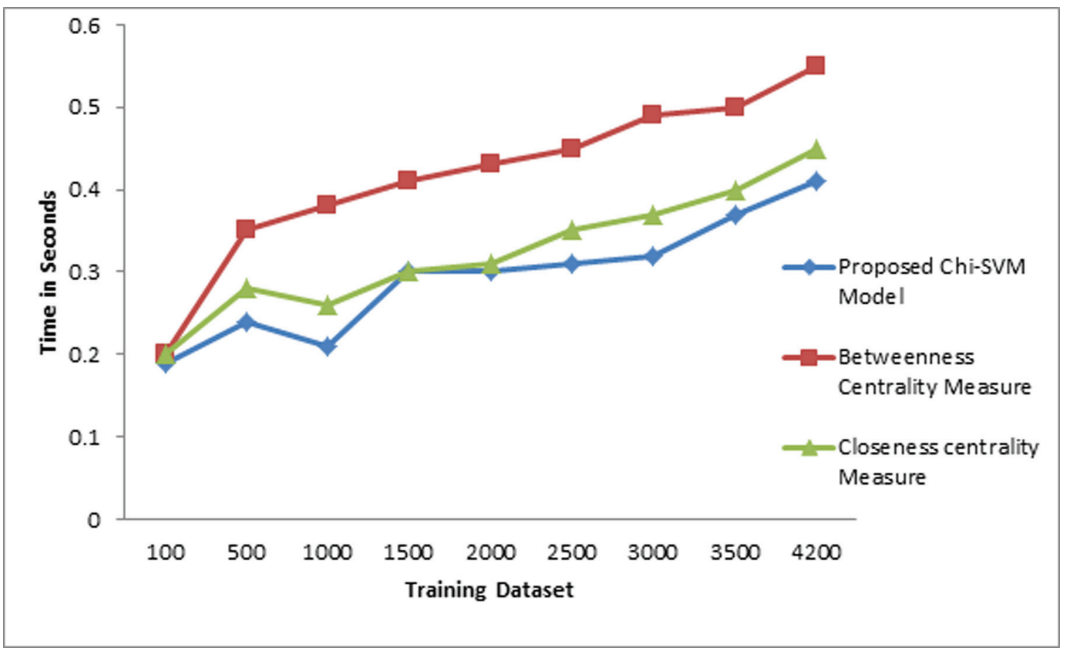
Table 4, show the execution time for given trained dataset. It consists of two columns, which displays the time taken by the training dataset to finish the computation for proposed model, BC & CC.

Figure 5, display the Time-Complexity graph for proposed Chi-SVM model, BC measure and CC measures, here the time taken to classify the test dataset with different number of training tuple(s) taken in percentage of dataset for the given intervals of time. The x-axis of graph depicts the number of dataset used as training tuples whereas the y-axis of the graph depicts the time taken to classify the test data. We are able to classify 5976 proteins in 0.41 seconds when our machine was trained with 4200 (58.56%) tuples using proposed Chi-SVM model, whereas BC has taken 0.55 second and CC has taken 0.45 seconds.

Table 4. Execution time for predicted essential proteins using Proposed Chi-SVM, BC and CC Measures

Number of training dataset	Proposed Chi-SVM model	Betweenness centrality measure	Closeness centrality Measure
100	0.19	0.2	0.2
500	0.24	0.35	0.28
1000	0.21	0.38	0.26
1500	0.3	0.41	0.3
2000	0.3	0.43	0.31
2500	0.31	0.45	0.35
3000	0.32	0.49	0.37
3500	0.37	0.5	0.4
4200	0.41	0.55	0.45

Figure 5. Time-complexity graph with number of tuples for proposed Chi-SVM, BC and CC measures



Statistical Performance Measures

To validate the performance of the proposed Chi-SVM model with BC & CC measures, different Statistical measurement parameters have been incorporated like positive predictive assessment (PPA), negative predictive assessment (NPA), specificity assessment (SA) & Sensitivity assessment (STA). Here top 1500 proteins are considered as essential proteins; meanwhile, the reset is considered as non-essential proteins.

Now, True positive is used in the identification of correctly predicted essential proteins. Similarly, True negative is used in the finding of correctly non-essential

protein prediction. Meanwhile, false-positive classify the erroneously predicted non-essential proteins. Later, false-negative represent the unnoticed essential proteins.

Positive predictive assessment is a ratio of correctly predicted essential protein with computed values of true and false assessment, equation 2 illustrated the same.

$$PPA=(TP)/(TP+FP) \quad (2)$$

Equation 3, uses the true negative and false negative values for the assessment of negative predictively.

$$NPA=(TN)/(TN+FN) \quad (3)$$

Specificity assessment describe the fraction of correctly anticipated non-essential protein (TN) versus total essential protein (TN+FP), equation 4 discuss the same.

$$SA=(TN)/(TN+FP) \quad (4)$$

Sensitivity assessment computes the ration between correctly known essential protein (TP) and sum of identified essential proteins (TP + FN), equation 5 define the same.

$$STA=(TP)/(TP+FN) \quad (5)$$

Table 5, illustrates the various Statistical Performance Measures for proposed Chi-SVM, BC & CC models. The result shows that statistical assessment values of PPA, NPA, STA & SA is far better for the proposed Chi-SVM model when compared with the BC & CC measures.

6. CONCLUSION

Essential proteins are composed of a set of minimal genome. They are amino acids that are required for the existence of an organism but cannot be synthesized within. Several open-source software has been developed for the prediction of essential proteins which includes, works based on biological, topological, and several other features. The majorities of these open-source software include the prediction technique that

Table 5. Statistical Performance Measures for proposed Chi-SVM, BC & CC models

Methodology\ Statistical Measures	PPA	NPA	STA	SA
Proposed Chi-SVM model	0.61	0.982	0.491	0.881
Betweenness centrality measure	0.501	0.721	0.434	0.832
Closeness centrality	0.511	0.691	0.42	0.84

uses network topology and feature selection for prediction of proteins by ignoring some of the features to reduce complexity, resulting in less accuracy.

In the proposed solution, the selenium driver is used to scrap the data and subsequently generate the supervised and test datasets. Later these dataset is first filtered through the Chi-Square Method, which is a test of dissimilarity used to rank the protein dataset based on probability values. Subsequently, the subset of this ranked dataset is then used for training the machine for classification. In addition, SVM is being used for the classification of the test dataset. After that, each protein is classified as one of the two classes, namely essential, non-essential, or undefined.

The result shows that the proposed method classified 86.64% proteins correctly when the machine was trained with 3500 (58.56% of the dataset) tuple's whereas the accuracy of BC & CC is only 82% & 89%. Later 99.56% protein is classified correctly when the machine was trained with 4200 (70.28%) tuples whereas, for BC & CC, it shows an accuracy of 84% & 86%. Finally, the model is able to classify 5976 proteins in 0.55 seconds when our machine was trained with 3500 (58.56%) tuples by incorporating the proposed Chi-SVM model. As the future work efficiency of essential proteins can be further improved by performing the computation on a multi-node environment.

REFERENCES

Breitwieser, F. P., & Colinge, J. (2013). IsobarPTM: A software tool for the quantitative analysis of post-translationally modified proteins. *Journal of Proteomics*, *90*, 77–84. doi:10.1016/j.jprot.2013.02.022 PMID:23470796

Cerami, E. G., Bader, G. D., Gross, B. E., & Sander, C. (2006). cPath: Open source software for collecting, storing, and querying biological pathways. *BMC Bioinformatics*, *7*(1), 7. doi:10.1186/1471-2105-7-497 PMID:17101041

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. doi:10.1007/BF00994018

Daniel, E., Onwukwe, G. U., Wierenga, R. K., Quaggin, S. E., Vainio, S. J., & Krause, M. (2015). ATGme: Open-source web application for rare codon identification and custom DNA sequence optimization. *BMC Bioinformatics*, *16*(1), 16. doi:10.1186/s12859-015-0743-5 PMID:26391121

Deng, S., Yuan, J., Huang, D., & Wang, Z. (2013). SFAPS: An R package for structure/function analysis of protein sequences based on informational spectrum method. *International Conference onBioinformatics and Biomedicine (BIBM), IEEE*, 29-34.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, *40*(1), 35–41. doi:10.2307/3033543

Holman, A. G., Davis, P. J., Foster, J. M., Carlow, C. K., & Kumar, S. (2009). Computational prediction of essential genes in an unculturable endosymbiotic bacterium, Wolbachia of Brugia malayi. *BMC Microbiology*, *9*(1), 243. doi:10.1186/1471-2180-9-243 PMID:19943957

Jeong, H., Oltvai, Z. N., & Barabasi, A. L. (2003). Prediction of protein essentiality based on genomic data. *Complexus*, *1*(1), 19–28. doi:10.1159/000067640

Jiang, Y., Wang, Y., Pang, W., Chen, L., Sun, H., Liang, Y., & Blanzieri, E. (2015). Essential protein identification based on essential protein–protein interaction prediction by integrated edge weights. *Methods (San Diego, Calif.)*, *83*, 51–62. doi:10.1016/j.ymeth.2015.04.013 PMID:25892709

Joy, M. P., Brock, A., Ingber, D. E., & Huang, S. (2005). High-betweenness proteins in the yeast protein interaction network. *BioMed Research International*, *2005*(2), 96–103. PMID:16046814

Joy, M. P., Brock, A., Ingber, D. E., & Huang, S. (2005). High-betweenness proteins in the yeast protein interaction network. *BioMed Research International*, *2005*(2), 96–103. PMID:16046814

Kamath, R. S., Fraser, A. G., Dong, Y., Poulin, G., Durbin, R., Gotta, M., & Welchman, D. P. et al. (2003). Systematic functional analysis of the Caenorhabditis elegans genome using RNAi. *Nature*, *421*(6920), 231–237. doi:10.1038/nature01278 PMID:12529635

Kim, J., Tsoy, Y., Persson, J., & Grailhe, R. (2017). FLIM-FRET analyzer: Open source software for automation of lifetime-based FRET analysis. *Source Code for Biology and Medicine*, *12*(1), 7. doi:10.1186/s13029-017-0067-0 PMID:29142589

Kraljic, K., Strungmann, L., Fimmel, E., & Gumbel, M. (2018). Genetic Code Analysis Toolkit: A novel tool to explore the coding properties of the genetic code and DNA sequences. *SoftwareX, 7*, 12-14.

Li, G., Li, M., Wang, J., Li, Y., & Pan, Y. (2018). United neighborhood closeness centrality and orthology for predicting essential proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1. doi:10.1109/TCBB.2018.2889978 PMID:30596582

Patel, M., & Shah, H. (2013, December). Protein Secondary Structure Prediction Using Support Vector Machines (SVMs). In *2013 International Conference on Machine Intelligence and Research Advancement* (pp. 594-598). IEEE.

Pearson, K. (1900). X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, *50*(302), 157–175. doi:10.1080/14786440009463897

Pedamallu, C. S., & Posfai, J. (2010). Open source tool for prediction of genome wide protein-protein interaction network based on ortholog information. *Source Code for Biology and Medicine*, *5*(1), 5. doi:10.1186/1751-0473-5-8 PMID:20684769

Qi, Y., & Luo, J. (2015). Prediction of essential proteins based on local interaction density. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *13*(6), 1170–1182. doi:10.1109/TCBB.2015.2509989 PMID:26701891

Ren, J., Wang, J., Li, M., Wang, H., & Liu, B. (2011, May). Prediction of essential proteins by integration of PPI network topology and protein complexes information. In *International Symposium on Bioinformatics Research and Applications* (pp. 12-24). Springer. doi:10.1007/978-3-642-21260-4_6

Sekhar, M., Sivagnanam, R., Matt, S. G., Manvi, S. S., & Gopalalyengar, S. K. (2019). Identification of Essential Proteins in Yeast Using Mean Weighted Average and Recursive Feature Elimination. *Recent Patents on Computer Science*, *12*(1), 5–10. doi:10.2174/2213275911666180918155521

Snel. (2002). *Search Tool for the Retrieval of Interacting Genes/Proteins*. https://stringb.org/cgi/download.pl? UserId= dEmo8h1aD NP0&sessionId =Lo4gfXCEqTv7

Von Mering, C., Jensen, L. J., Snel, B., Hooper, S. D., Krupp, M., Foglierini, M., ... Bork, P. (2005). STRING: known and predicted protein–protein associations, integrated and transferred across organisms. *Nucleic Acids Research, 33*(suppl_1), D433-D437.

Wang, J., Peng, X., Peng, W., & Wu, F. X. (2014). Dynamic protein interaction network construction and applications. *Proteomics*, *14*(4-5), 338–352. doi:10.1002/pmic.201300257 PMID:24339054

Yates, F. (1934). Contingency tables involving small numbers and the χ 2 test. *Supplement to the Journal of the Royal Statistical Society*, *1*(2), 217–235. doi:10.2307/2983604

Young, V. R. (1994). Adult amino acid requirements: the case for a major revision in current recommendations. *The Journal of nutrition, 124*(suppl_8), 1517S-1523S.

Zhang, Z., Ruan, J., Gao, J., & Wu, F. X. (2019). Predicting essential proteins from protein-protein interactions using order statistics. *Journal of Theoretical Biology*, *480*, 274–283. doi:10.1016/j.jtbi.2019.06.022 PMID:31251944

International Journal of Open Source Software and Processes

Volume 11 • Issue 3 • July-September 2020 • ISSN: 1942-3926 • eISSN: 1942-3934

MISSION

The **International Journal of Open Source Software and Processes (IJOSSP)** publishes high-quality original research articles on the large field of open source software and processes. The primary mission is to enhance our understanding of this field and neighbouring areas by providing a focused outlet for rigorous research employing a multitude of approaches.

SUBSCRIPTION INFORMATION

The International Journal of Open Source Software and Processes (IJOSSP) is available in print and electronic formats and offers individual or institution-level pricing. Full subscription information can be found at www.igi-global.com/IJOSSP.

IJOSSP is also included in IGI Global's InfoSci-Journals Database which contains all of IGI Global's peer-reviewed journals and offers unlimited simultaneous access, full-text PDF and XML viewing, with no DRM. Subscriptions to the InfoSci-Journals Database are available for institutions. For more information, please visit www.igi-global.com/infosci-journals or contact E-Resources at eresources@igi-global.com.

CORRESPONDENCE AND QUESTIONS

EDITORIAL

Antonio Pecchia, Editor-in-Chief • IJOSSP@igi-global.com

SUBSCRIBER INFO

IGI Global • Customer Service
701 East Chocolate Avenue • Hershey PA 17033-1240, USA
Telephone: 717/533-8845 x100 • **E-Mail:** cust@igi-global.com

The *International Journal of Open Source Software and Processes* is indexed or listed in the following.

ACM Digital Library; Bacon's Media Directory; Cabell's Directories; DBLP; GetCited; Google Scholar; INSPEC; JournalTOCs; MediaFinder; Norwegian Social Science Data Services (NSD); SCOPUS; The Index of Information Systems Journals; The Standard Periodical Directory; Ulrich's Periodicals Directory