Software Design Specification

Bug Similarity in Heterogeneous Networks
(Research Project)

GROUP - Club_Elite

IIT2018114 - Harsh Goyal
IIT2018144 - Aaditya Gadhave
IIT2018149 - Sourabh Gupta
IIT2018158 - Meet Singh Gambhir
IIT2018159 - Tushar Atrey

Instructor - Amit Kumar, Ashutosh Kumar
Subject - Software Engineering

IV Semester, Department of Information Technology,
Indian Institute of Information Technology, Allahabad, Prayagraj.

Table of Content

Abstract

In this research project we have devised an algorithm to predict the relationship (any kind of relationship) between any pair of given bugs. This algorithm takes input from all the previous relationships between the bugs and predicts the extent of any new relationship between any new pairs of bugs.

INTRODUCTION

In this project suppose we consider two bugs which work on the same file then these two bugs are considered to be similar and there might be chances that they two have relation in future. This is just one kind of relationship. This relationship is generally denoted in the form of metapath that means that if two bugs are similar they have a path that connects these two bugs via any type of relationship. There are a total of 22 different metapath between any two given bugs. They are:

1) **B-T-B** : Bug reports having the same term.

2) **B-R-B** : Bug reports reported by same reporter.

3) **B-C-B** : Bug reports found in the same components.

4) **B-A-B** : Bug reports assigned to the same assignee.

5) **B-D-B** : Bug reports assigned to the same developer.

6) **B-F-B** : Two bugs changing the same file.

7) **B-T-B-D-B** : If B1 and B2 are related by the same term and B2 and B3 are related by the same Developer then B1 and B3 are related by this type of relationship.

8) **B-R-B-D-B** : If B1 and B2 are related by the same reporter and B2 and B3 are related by the same Developer then B1 and B3 are related by this type of relationship.

9) **B-C-B-D-B** : If B1 and B2 are related by the same component and B2 and B3 are related by the same Developer then B1 and B3 are related by this type of relationship.

10) **B-A-B-D-B** : If B1 and B2 are related by the same assignee and B2 and B3 are related by the same Developer then B1 and B3 are related by this type of relationship.

11) **B-F-D-F-B** : Two bug reports which are committed by the same developer i.e if B1 changes file F1 and is committed by developer D and if another bug B2 changes file F2 and is committed by same developer D then B1 and B2 are connected by this path.

12) **B-T-F-B** : If bug report B1 has term T which is also found in some path of file F and F has been changed by B2 then B1 and B2 are related by this path.

13) **B-C-F-B** : If bug report B1 has component C which is also found in some path of file F and F has been changed by B2 then B1 and B2 are related by this path.

14) **B-T-B-F-B** : If B1 and B2 are related by the same term and B2 and B3 change the same file then B1 and B3 are related by this type of relationship.

15) **B-R-B-F-B** : If B1 and B2 are related by the same reporter and B2 and B3 change the same file then B1 and B3 are related by this type of relationship.

16) **B-C-B-F-B** : If B1 and B2 are related by the same component and B2 and B3 change the same file then B1 and B3 are related by this type of relationship.

17) **B-A-B-F-B** : If B1 and B2 are related by the same assignee and B2 and B3 change the same file then B1 and B3 are related by this type of relationship.

18) **B-T-B-F-D-F-B** : If B1 and B2 are related by the same term and B2 and B3 are changing the same file which are committed by the same developer then B1 and B3 are related by this path.

19) **B-R-B-F-D-F-B** : If B1 and B2 are related by the same reporter and B2 and B3 are changing the same file which are committed by the same developer then B1 and B3 are related by this path.

20) **B-C-B-F-D-F-B** : If B1 and B2 are related by the same component and B2 and B3 are changing the same file which are committed by the same developer then B1 and B3 are related by this path.

21) **B-A-B-F-D-F-B** : If B1 and B2 are related by the same assignee and B2 and B3 are changing the same file which are committed by the same developer then B1 and B3 are related by this path.

22) **B-F-P-F-B** : If B1 and B2 are changing the same file which are located in the same package then they are related by this path.

PRODUCT SCOPE

There are two types of graph. They are:

1) Heterogeneous graphs
2) Homogeneous graphs

Heterogeneous graphs are those which have different kinds of nodes and edges i.e they are of different types.
Homogeneous graphs are those which have the same (or similar) kind of nodes and edges.
Heterogeneous graph is one of the latest concepts which has gained a lot of attention in recent times as due to the advantage of its over that of the homogenous one as it contributes very more of that in retrieving the information than the homogenous one.
The main reason for this being that it considers all types of possible elements of the network as nodes , and any edge between the two , ensures their connectivity , whether they being of different types of nodes , which is not being the case when we deal with the homogenous one. It will use that there will be some sort of similarities between two nodes based on the heterogeneous graph .It has various useful applications such as predicting relationships between various possibilities of two nodes possible and not only the two bugs only. Somewhat similar concept is also used in the link prediction and in the social media suggestion provided. Therefore we have worked on heterogeneous graphs for this project.

Definitions, Acronyms and Abbreviations

Acronyms and abbreviations:

- SRS: Software Requirement Specification
- SDS : Software Design Specification
- ML: MachineLearning
- SEOSS: Software Engineering in Open Source Systems

Definitions:

a. SEOSS Dataset: A systematically retrieved dataset consisting of 33 open-source software projects containing a large number of typed artifacts and trace links between them. The artifacts stem from the projects' issue tracking system and source version control system to enable their joint analysis.

b. Heterogeneous Graph: A heterogeneous graph is a type of graph that is comprising of various types of nodes (all nodes , i.e. possibility of all elements in the project, is to of the node) and edges (edges can be all , relating two nodes whether of same type or different types of nodes).

c. meta-path : A sequence of relations between object types, which defines a new composite relation between its starting type and ending type

d. HeteSim: A Meta path-based similarity measure.

Acknowledgments And References

- Bug Categorization
- Bug Categorization - 2
- Charu-Hete.pdf
- Heterecom-1
- Heterecom-2
- SEOSS
- Duplicate - 2
- SupervisedLink Prediction
- The Link Prediction Problem for Social Networks
- Time and Link prediction

OVERALL DESCRIPTION

Operating Environment

We are taking a shot at enormous information heterogeneous systems. We are given information from 33 distinct projects as sqlite datasets. We are figuring out individual relationships with the assistance of google sheets. We are submitting and pushing the progressions to our python application at Github.

User Characteristics

Clients ought to be comfortable with running projects utilizing command prompt. He/She ought to be comfortable with different criteria of similarity, similar to individual coefficient and path based importance measure.

Procedure

We started the project with the help of Amit sir who conducted meetings every week for an hour or two. The meeting started with Introduction of the heterogeneous graph and different types of relationship between different bugs. These relationships were denoted with the help of meta paths .We were provided with many sets of research papers that helped to build our own algorithm to count and predict the path between the bugs .

At first we started with the B-X-B kind of meta path and then with help of these we built algo for a bigger metapath. We used the test cases provided in the dataset by the instructor to run our algorithm. We used a database to store all the count of paths between every two bugs using algorithms similar to dfs (depth-first-search) and bfs (breadth-first-search).

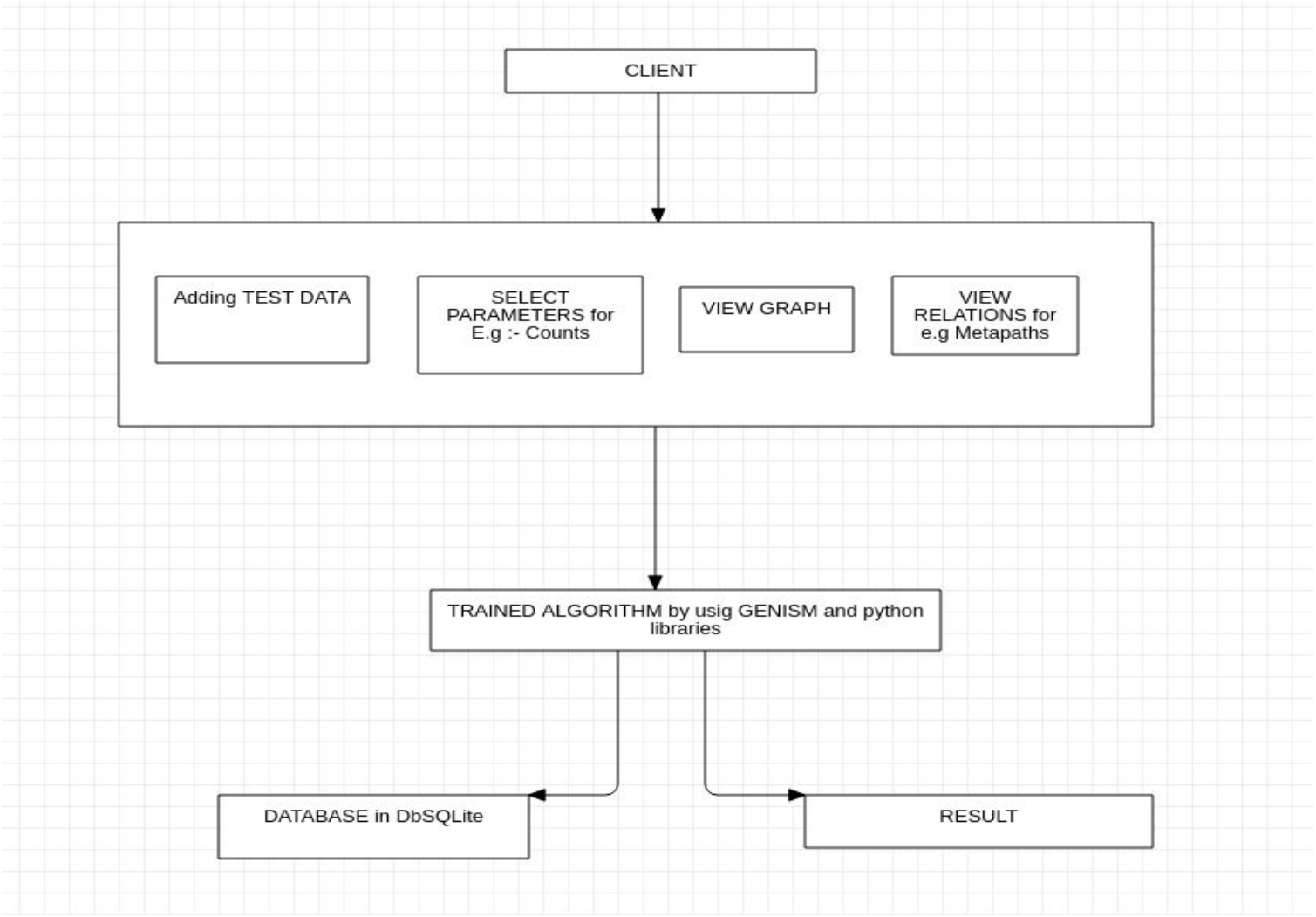
We used Python Language for this project as it is easy to use and has many libraries which are helpful for this algorithm. With python we can also implement Machine Learning Algorithm that helps to predict the future relationships between the bugs with the help of present relations between the bugs. I.E. if we use a test case consisting of a certain kind of relationship between the bugs, called the training data then with the help of the algorithm if there is a certain path between the bugs then the algo predicts the future possible relationship between the bugs and their probability of success.These were called as Testing dataset.

After this the instructor asked to use the concept of lemmatization. **Lemmatization** is the process where given a text we convert all verb form into present form. For topic modelling we were introduced with various tool like Gensism, LDA++ and Mailet tool. It also helps to remove stop words (is, the,

are,etc.) using **Spacy** Library.We then with the help of this developed an algorithm that took word n-gram from two different text files and compared these n-grams.

2. Conceptual Architecture/Architecture Diagram

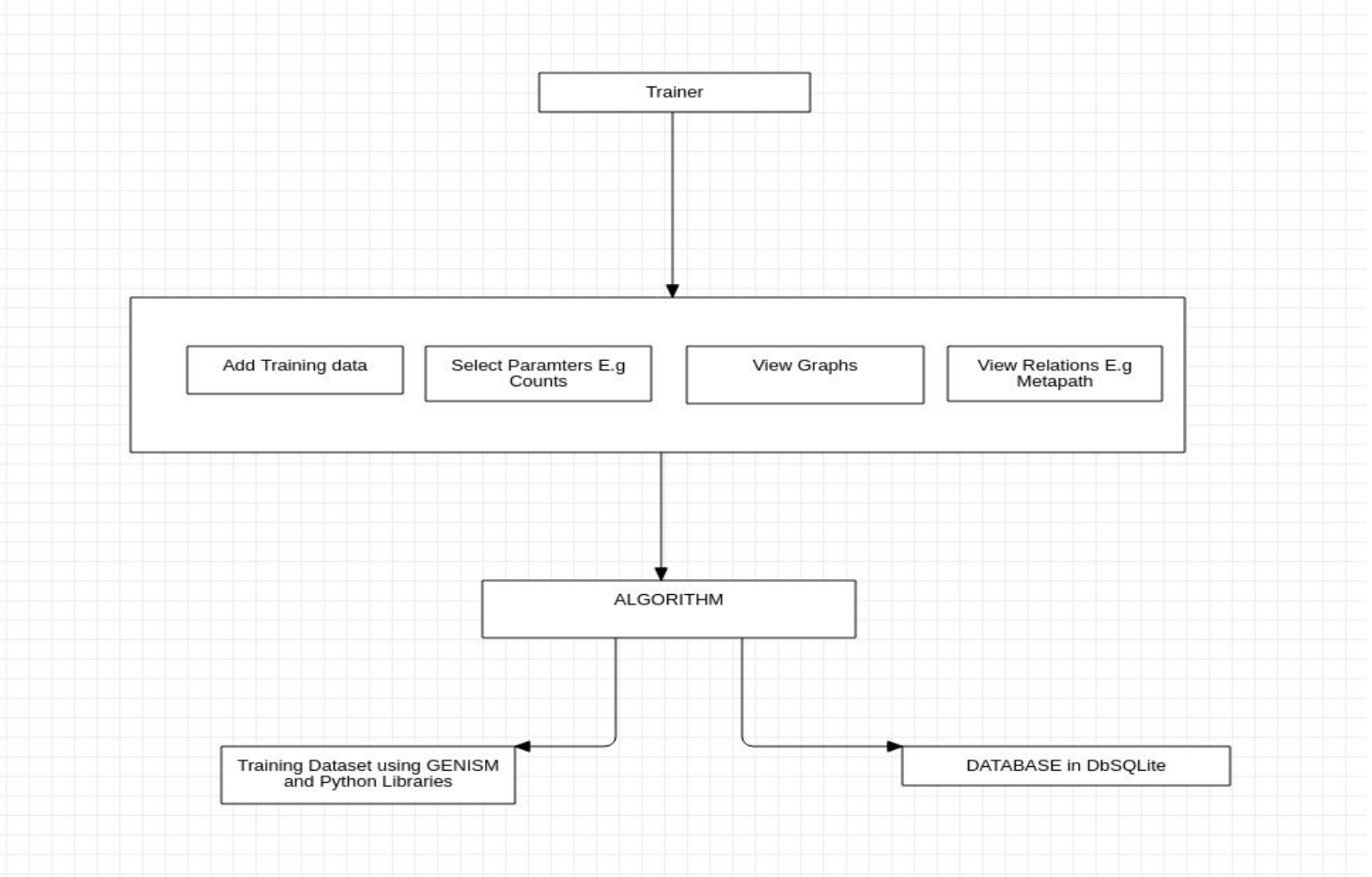
Architecture Diagram 1:



The above architecture diagram shows us how the software will interact with client and with the help of the algorithm trained by the software like Genism we can predict the count of metapath and the percentage of similarity between 2 bugs . The client will provide the test data for which we have to do these calculations . The algorithm needs to be need to be

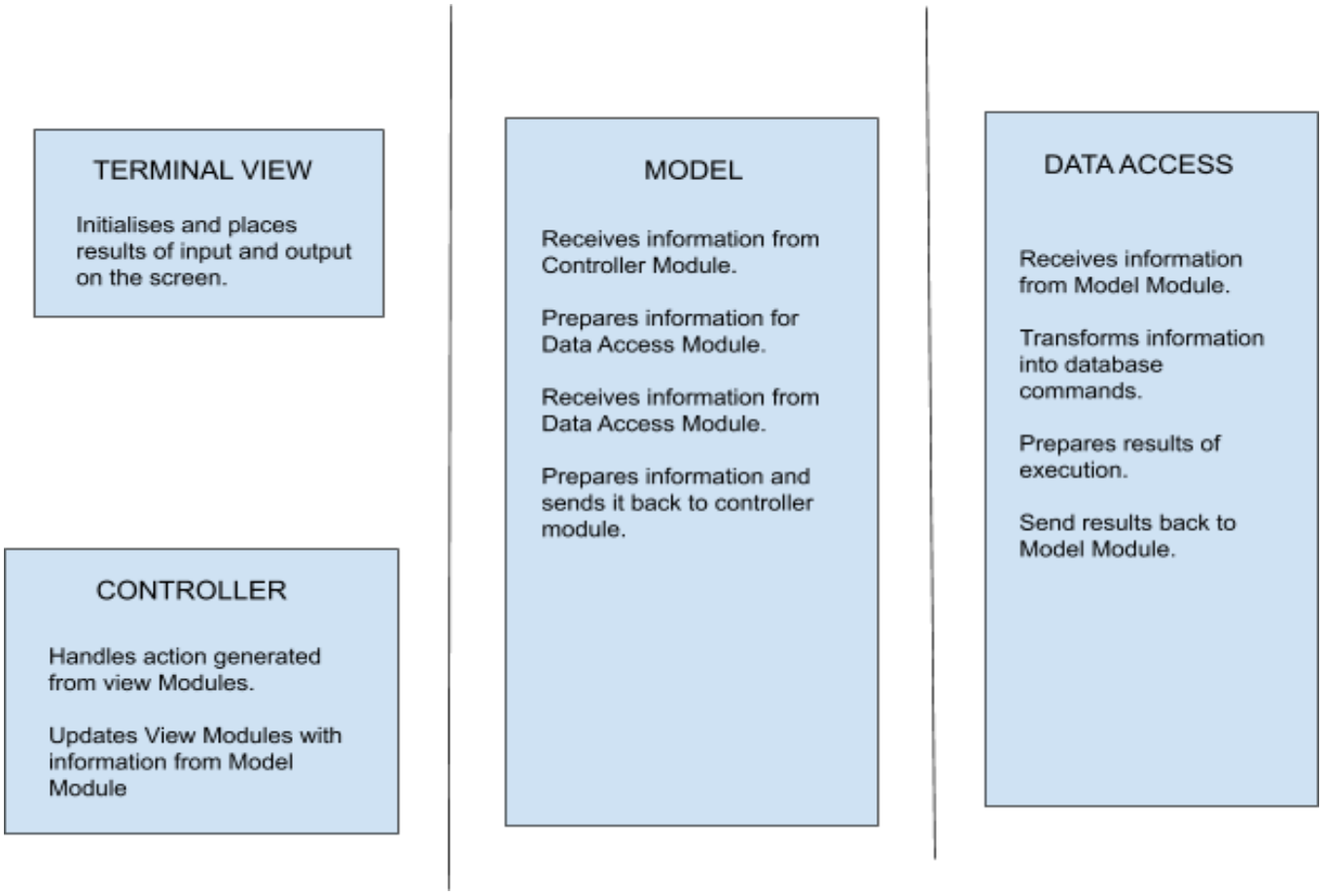
trained on some datasets so as it could make better and accurate predictions.

Architecture Diagram 2 :



The above architecture diagram depicts the training of algorithms. The trainer provides suitable Training data set for the algorithm to train itself and after an adequate amount of training we are ready to try it on the actual data set.

2.1 Overview of modules / Components

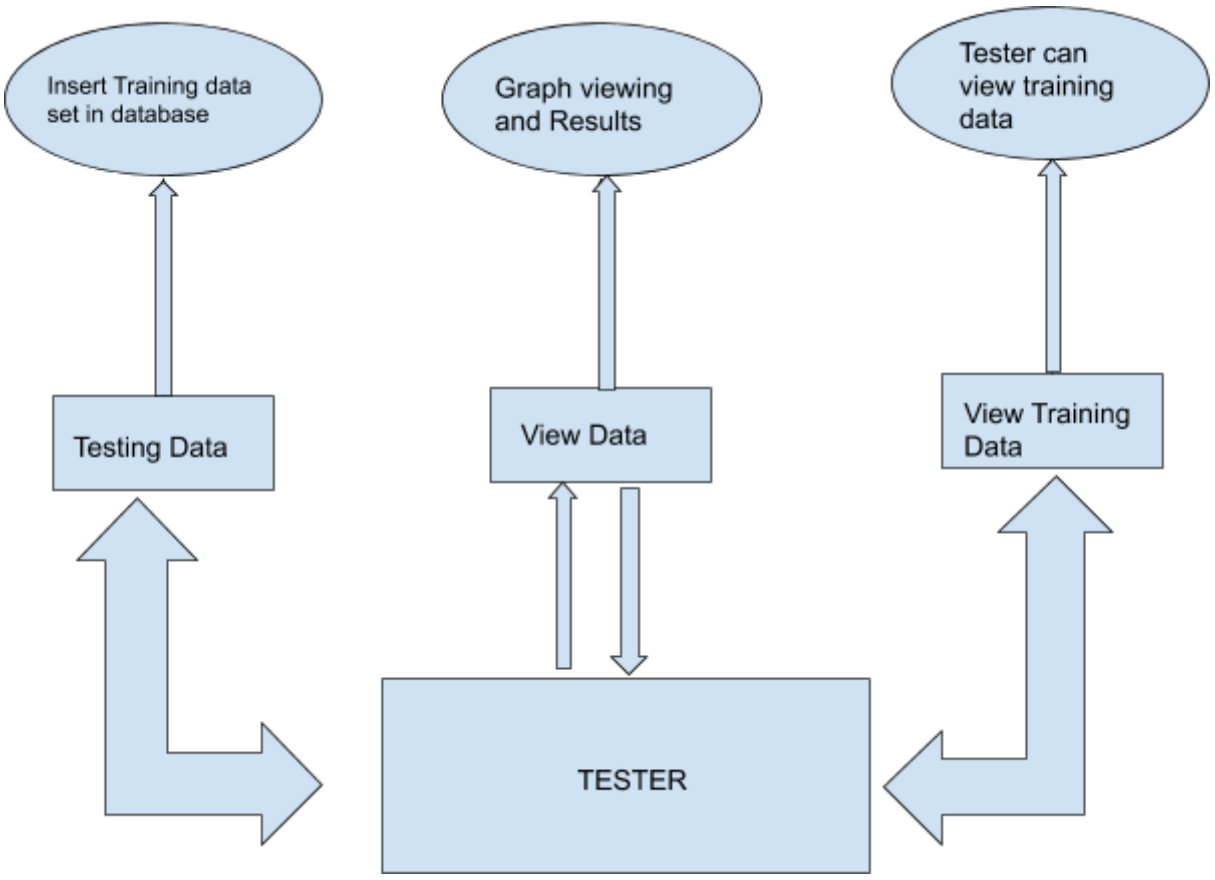


NOTE:

- The horizontal lines represent the separation of modules.
- More than one box within the same section represents sub-modules.

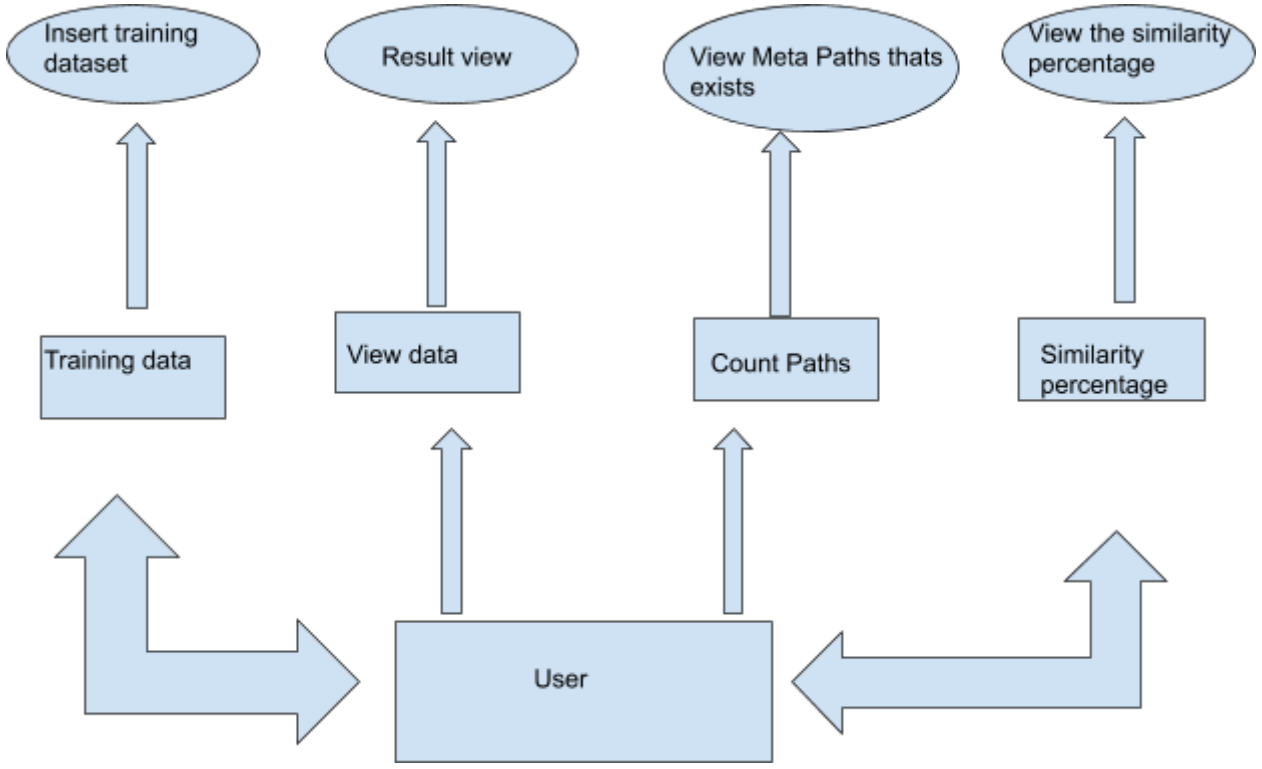
2.2 Structure and relationships

2.2.1 Tester's Side



The above diagram shows the structure and relationship of the tester with the software. Tester firstly inserts the training data to test and then he get back the results and the count of different metapath that exist and similarity between 2 bugs. Tester is the one which can see the training as well as testing data both whereas users can only see the testing data.

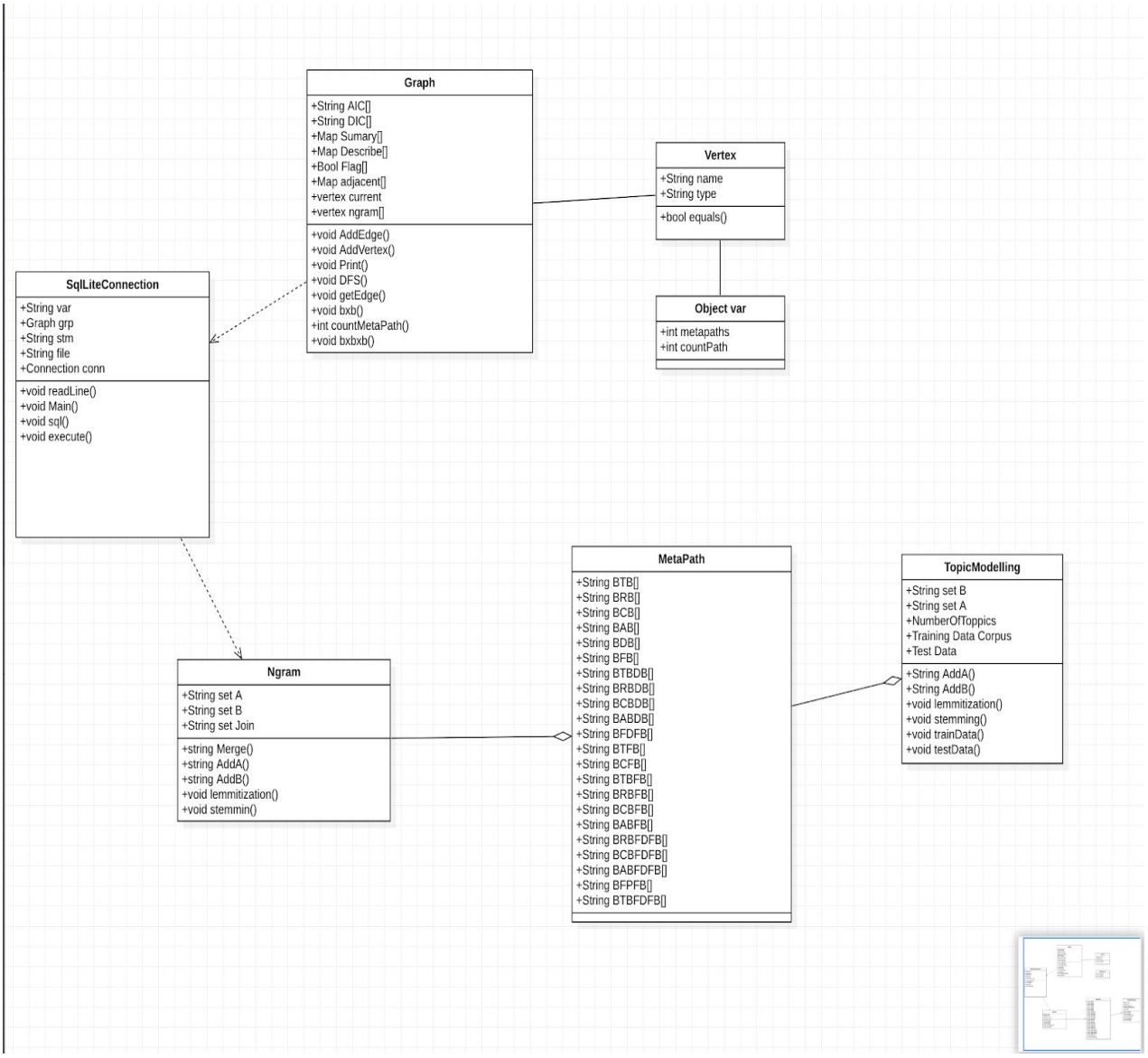
2.2.2 User's Side



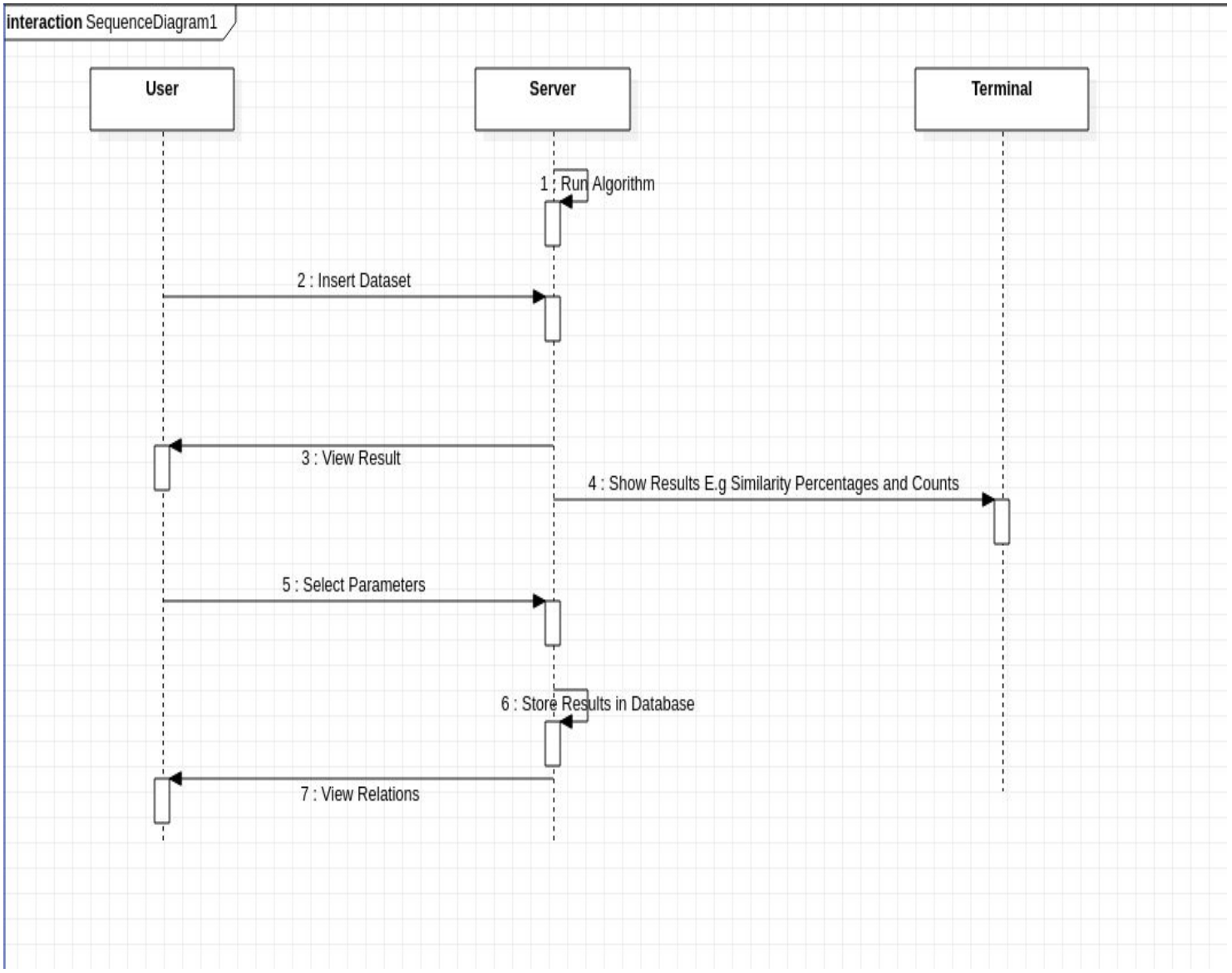
The above diagram shows the structure and relationship of user with the software. User firstly inserts the training data and then he gets back the results for different types of meta paths between that exists (all possible) and counts the paths that exist. And then the calculation is done on the basis of relations that are similar between 2 bugs.

3. Logical Architecture
(Class Diagram,Sequence Diagram,State Diagram)

- **Class Diagram:**



- **Sequence Diagrams:**



The sequence diagram below represents how our software is basically going to work. The User/Client will provide the required dataset and as an output of trained algorithm will produce the required results based on the parameters selected by the user.

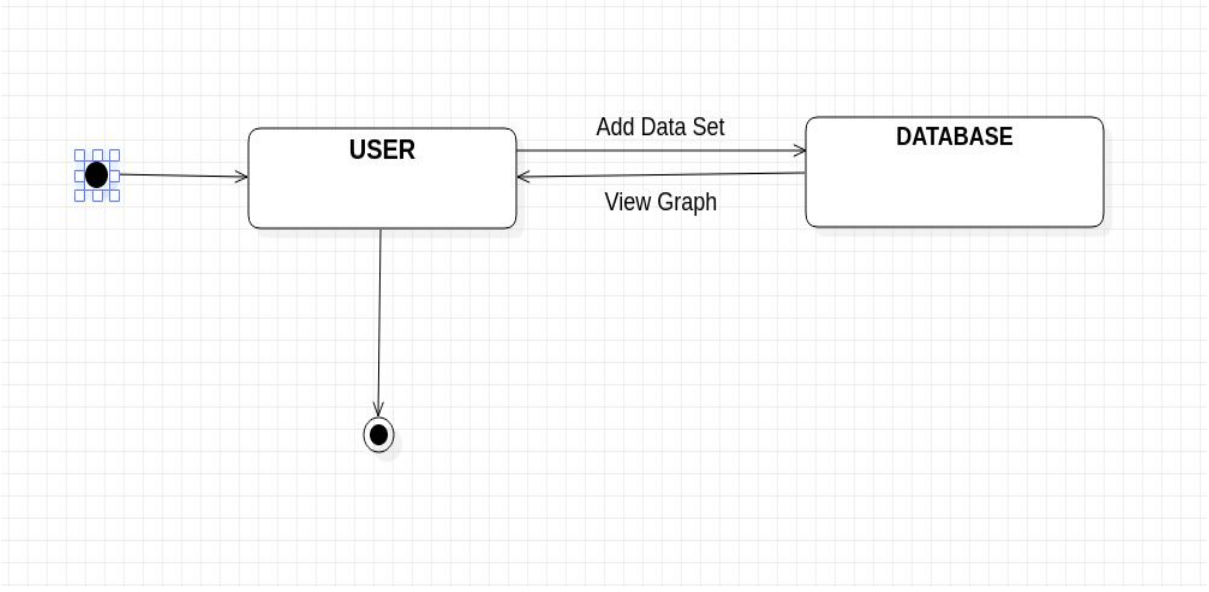
Basically, our sequence diagrams represent either how our software interacts with the user to produce the required results, how our algorithm

was trained by the trainer so that algorithm could become capable of Predicting accurate similarity between 2 bugs and calculate the percentage of similar text based on topic, understanding by using n gram similarity.

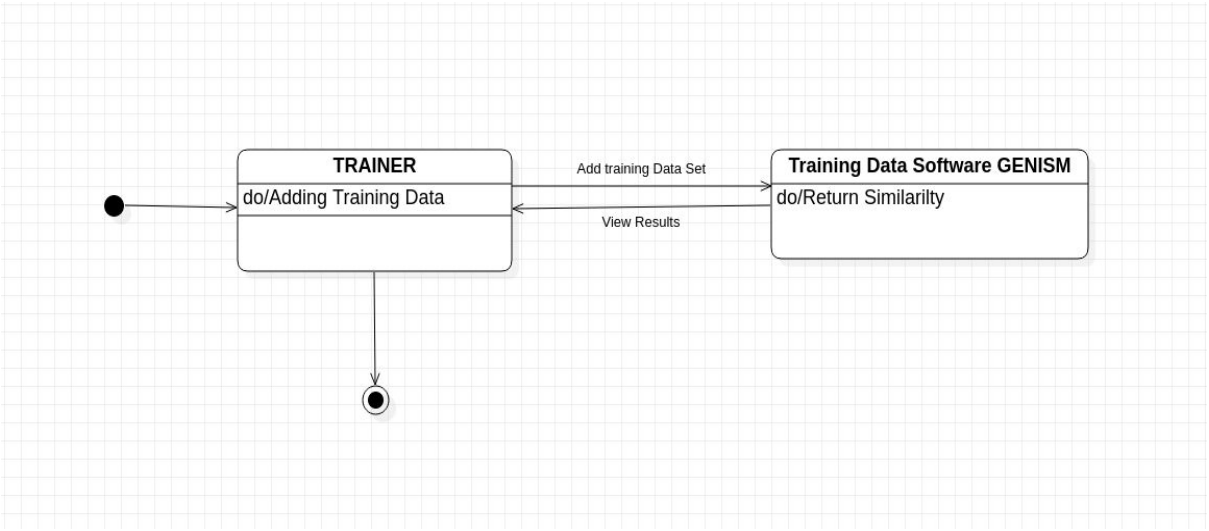
- **State Diagrams:**

State diagrams are mainly used to capture the behaviour of software system. State machine diagrams can be used to model the behavior of a class, a subsystem, a package, or even an entire system. It is also called a **Statechart or State Transition diagram.**

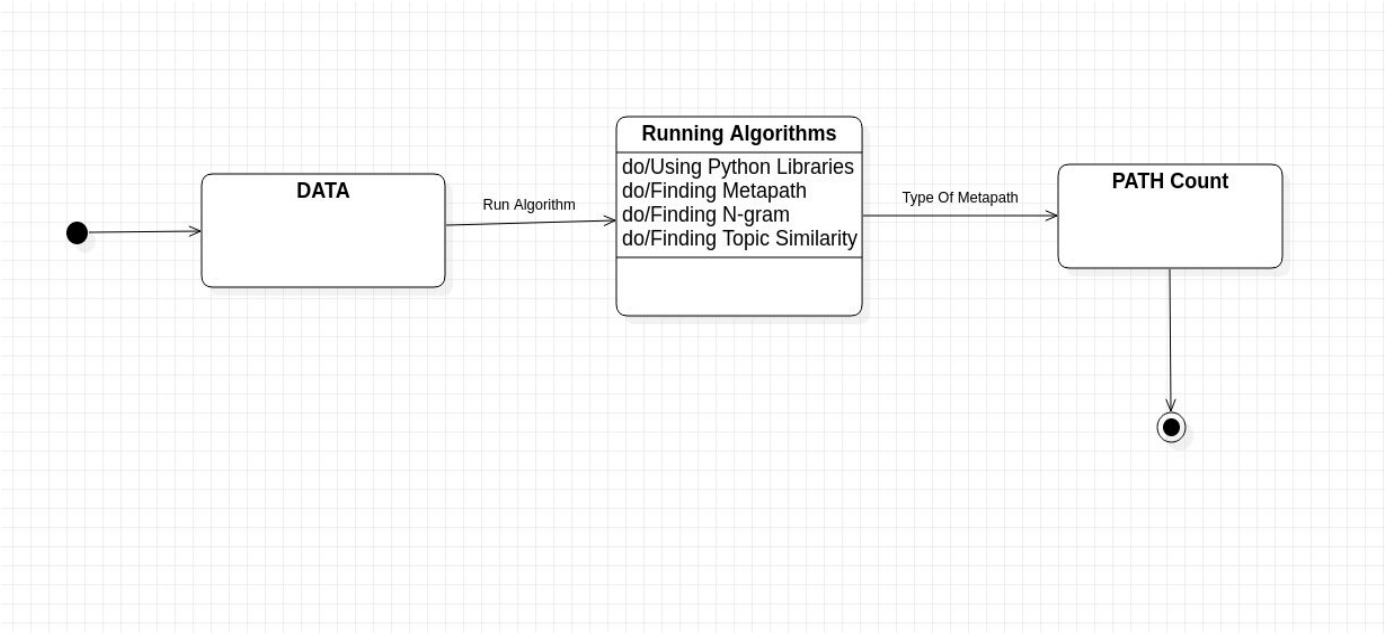
State diagram 1: Admin inserts data



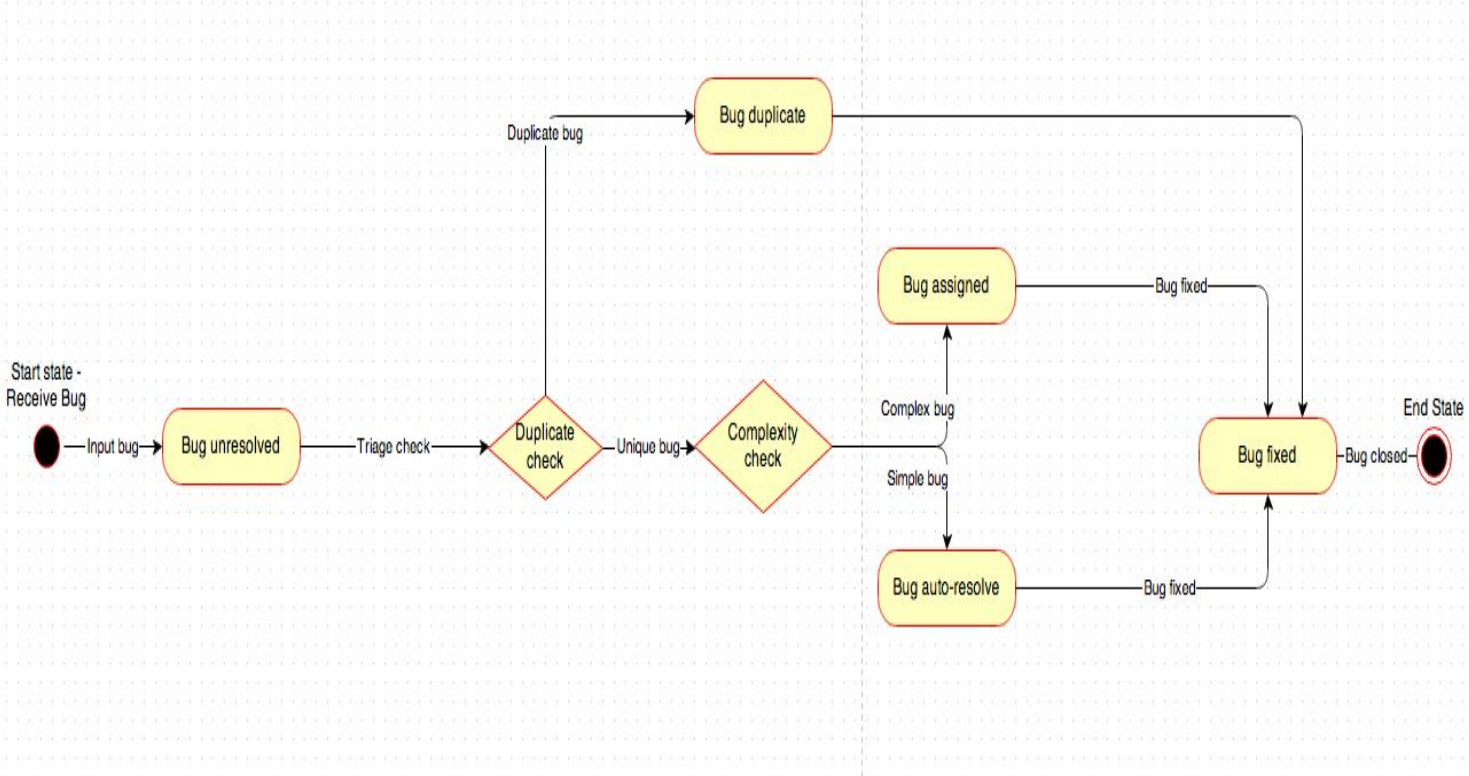
State diagram 2: Trainer inserts data



State diagram 3: Algorithm for path counts



- **State Diagram 4: Similarity between 2 bugs**



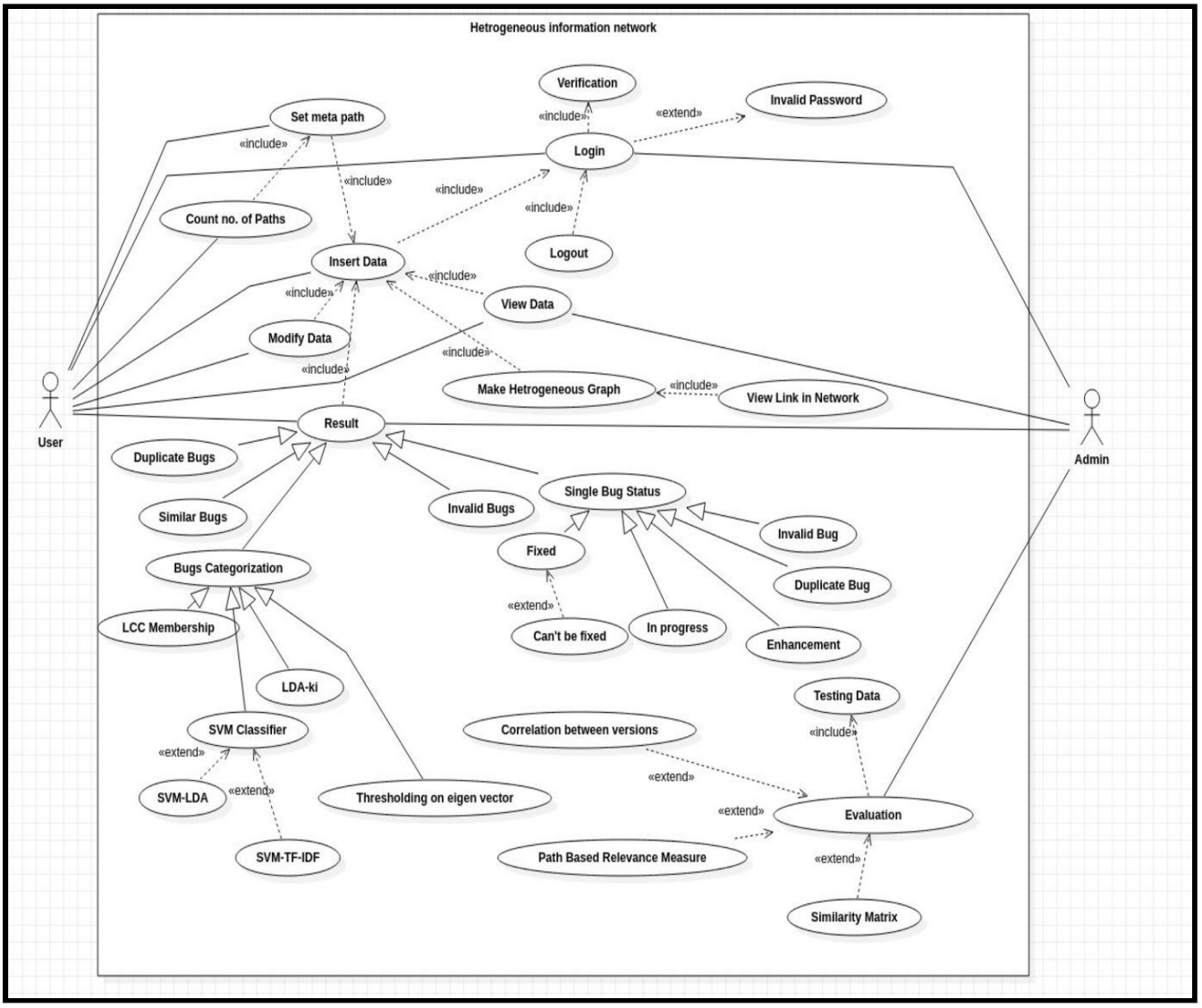
Design decisions and tradeoffs

Since our Software involves the prediction of links between any specified entities thus it does not have any frontend involved with it all it includes is the prediction of accurate links. Thus, this software has a wide range of applications and can be put to use with many other products such as social media platforms, performance analyzers, etc.

The Design decision to use a Relational Database rather than a Non-Relational Database is basically to improve the scalability and also because we have a defined Schema in case of SQL databases.

The End-user for our software are basically other product-based applications that can use our software to improve their product and increase its functionality.

Use Case Diagram



The functionality of the diagram is described as follows:

1) **Name :** Login
   Summary : Allows admin/user to login .
   Actors : Admin/User

   Main success scenario :
   • Admin/User clicks on the login button.
   • App checks for the Verification of login.

   Extension: Id or password incorrect. Shows error dialog box.

   Post-condition-: Admin/User can now access all features of the app.

2) **Name:** Insert Data
   Summary: Allows users to insert data about the bugs of the project.
   Actors: User

   Main success scenario:
   • Admin provides a dataset in the form of a sqlite database.
   • App connects the sqlite database with the project and includes it for predicting Bug Similarities.

   Extension: Here we are working with sqlite databases for this project, but we can also work with other databases after performing database migrations such as MySQL.

   Post-condition: Project will now use the given database for predicting Bug Similarities.

3) **Name:** View Data
   Summary : Allows admin/user to view the SQL database to our application
   Actors : Admin/User

   Main success scenario:
   • Actor connects to the database successfully and verifies the data.

   Post-condition: Admin can now access the data from different projects in our heterogeneous graph.

4) **Name:** Modify Data.
   Summary: User can modify the existing database.
   Actors: User

Main Success Scenario:
• Users cross-verifies the database and make required changes.

Post Condition: Now User can expect Correct Result.

5) **Name:** Make Heterogeneous Graph. **Summary:** From the given database Admin creates the required Heterogeneous Graph.
Actors: Admin

Main Success Scenario:
• Successful formation and display of the required graph.

Extension: Now the link between the nodes can be viewed in the network.

Post Condition: Now we can Predict the similarity between bugs by using this graph.

6) **Name:** Number of the paths for metapath. **Summary:** Calculate total number of paths with the given type of metapaths **Actors:** Admin

Main Success Scenario:
• There are various types of meta paths in the dataset and calculation of total number of paths of the specified metapath between bugs.

Post Condition: Prediction of similarity between bugs by counting total number of paths of all the meta paths.

7) **Name:** Choose meta paths.
Summary: Admin gives different types of meta paths to find the links between Bugs. **Actors:** User

Main success scenario:
• Admin provides various metapaths which will be studied as a criteria for prediction of Bug similarity.
• App measures the number of various metapaths to use a path based relevance measure algorithm.

Post-condition: Project will now measure all the various kinds of metapaths, so that path based relevance measure algorithms can be applied.

8) **Name:** Result.
Summary: From the given database Admin concludes the outcome from Heterogeneous Graph

Actors: User

Post Condition: Now the User can see the details of the result.

9) **Name:** Duplicate Bugs.
Summary: From the result user can view all the duplicate bugs of the data.
Actors: User

Main success scenario:
• App measures all the Duplicate Bugs and shows it to the user.

Post-condition: All the Duplicate Bugs can be discarded.

10) **Name:** Similar Bugs.
Summary: From the result the user can view all the Similar bugs of the data.
Actors: User

Main success scenario:
• App measures all the Similar Bugs and shows it to the user.

Post-condition: All the Similar Bugs can be categorised into one category.

11) **Name:** Invalid Bugs.
Summary: From the result the user can view all the Invalid bugs of the data.
Actors: User

Main success scenario:
• App measures all the Invalid Bugs and shows it to the user.

Post-condition: All the Invalid Bugs will be discarded and not considered into the data.

12) **Name:** Single Bug Status
Summary: From the result the user can view the status of any particular bug.
Actors: User

Main success scenario:
• App measures all the Status of the particular chosen bug and shows it to the user.

Extension: A particular bug can be fixed , or in progress , is invalid or can be among the duplicate ones or even can be an enhancement.

13) **Name:** Bug Categorisation.
Summary: From the result the user can view all the classification of the bugs of the data.
Actors: User

Main success scenario:
• App measures all the classification of the Bugs and shows it to the user.

Extension: Different algorithms such as LCC membership , SVM classifier , LDA-Ki , Thresholding on eigenvectors are employed to categorise the bug

Post-condition: All the similar Bugs will be categorised in the same category.

14) **Name:** Path based Relevance Measure
Summary: Finding the relatedness of an object pair depending on the given relevance path.
Actors: Admin Software Requirements

Main success scenario:
• Admin calculates the relatedness of bugs connected in the heterogeneous graph using the Path based Relevance measure algorithm given in HeteSim: A General Framework for Relevance Measure in Heterogeneous Networks.

Post-condition: Project will provide correlation between users(Developers) in a range from 0 to 1, using a path based relevance measure algorithm.

15) **Name:** Testing data.
Summary: A dataset is added to our application for testing of our algorithm and checking its correctness.
Actors: Admin

16) **Name:** Design similarity matrix
Summary: Similarity Matrix is created by the training data which will further help in generating links.
Actors: Admin

Main success scenario:

• The similarity matrix between different types of nodes is created and it helps in link prediction techniques using this matrix.

Post-condition: Now the application contains a similarity matrix which can be further used for link prediction in our heterogeneous graph .

17) **Name:** Evaluation
Summary: Calculates accuracy between predicted links using different projects
Actors: Admin

Main success scenario:
•Based on previous predictions we can calculate our accuracy percent by calculating the pearson coefficient.

Post-condition: We can know the accuracy of predicting a new link.

18) **Name:** Correlation between different versions of a project.
Summary: Prediction of similarity between bugs in different versions of the project.
Actors: Admin

Main success scenario:
• The project checks newer versions of any given project to check if the link predicted by the app does actually exist in the future.

Post-condition: This will help to research to what extent does heterogeneous graph predicting links works.

19) **Name:** Training data
Actor : Admin

Main success scenario:
•.Training data is quite essential for generating appropriate results from the algorithm.
• A separate graph is created based on the training dataset in which we can test our algorithm to increase the accuracy of our algorithm.

Extension: The given dataset should be valid i.e it should contain all the required fields for our algorithm to work else an error message will be shown.

Post-condition: The algorithm is now trained and can be seen working correctly if not the necessary changes can be made so that it works fine on big datasets.

THE TRAVERSAL AND COUNTING OF THE PATHS

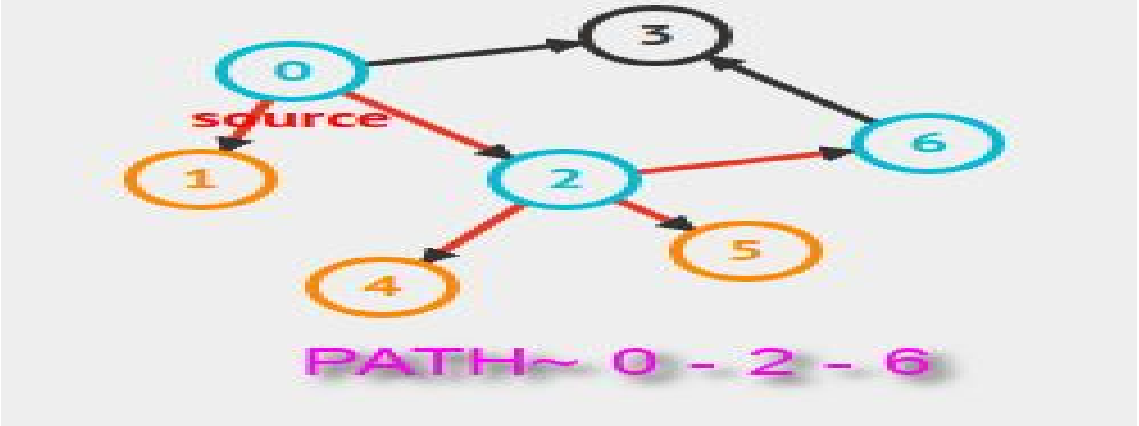
The graph constitutes , of each possible element of the project (viz Bug,term , component , file , Developer) as a vertex in the graph , and any two of the vertices , being related , by a relation (say a bug , changing contents of a particular file) ,are related , and this constitutes as an edge in our graph.

There are different types of vertices , in the graph , any thus , many different types of edge, i.e. relating any two types of vertex.Now this makes the graph heterogeneous in nature.

As the graph is not homogenous , the concepts now should be somewhat modified , in order to apply to the heterogenous one, comprising considering different types of vertices and edges differently among themselves, which is sufficient in order to do operations for the heterogeneous one.

Now from each vertex as the starting one, each type of already mentioned path's frequency is needed to be calculated .The algorithm used to do so is dfs, i.e. the depth first traversal.

WHY DFS ?



Just like an iteration in an array ,to traverse the array , there are two possibilities to do in case of graphs, dfs and bfs.

The dfs algorithm begins at a starting node, and proceeds to all other nodes that are reachable from the starting node using the edges of the graph.
Depth-first search always follows a single path in the graph as long as it finds new nodes. After this, it returns to previous nodes and begins to explore other parts of the graph. The algorithm keeps track of visited nodes, so that it processes each node only once.
Whereas Bfs visits the nodes in increasing order of their distance from the starting node.
Breadth-first search goes through the nodes one level after another. First the search explores the nodes whose distance from the starting node is 1, then the nodes whose distance is 2, and so on. This process continues until all nodes have been visited.
The time complexity for both the algorithms is the same , each being O(n+m) , n being no. of nodes and m , being the no. of edges . because each edge will only be traversed, twice , one time from each of the two connecting vertices, for each algorithm(bfs or dfs whichever you are proceeding) .So the time factor is nullified from sides.
Coming to the reason that we are opting for dfs , is that we want to store the path from the initial source , to each vertex ,so as to check which type of path is this from the source , and in each iteration in the dfs algorithm , the path is changed by at most 1 value.
The steps in dfs are:

a.) Picking a starting node and pushing all its adjacent nodes using edges from it , into a stack.
(This will add the starting node in the path 1 iteration and the path from the starting to this vertes , is the current content in the path).

b.) Pop a node from stack to select the next node to visit (popping from top a node , from the stack is a one iteration process) and push all its adjacent nodes into a stack (this step will not change the path).

While in case of obtaining the path (i.e. the order of vertices visited from the source vertex ,, to reach any vertex) , we will need additional O(n) , time , which will make it worse than dfs.

PSEUDOCODE

The task is restricted to find only the mentioned paths and also the length of the paths , is small, so we can go for them explicitly as well
. i.e. say for finding count for the paths B-T-B , for each possible pair of B , we increase their count by one, if they share the same T.

Pseudocode->

```
issue_id <-  set()
component <- set()
link <- dict()
ans <- dict()
edge <- []
file <- input() // the data file
file_handle <- open(file,'r')
file_handle.readline()
for line in file_handle:
    a,b=line.split(",",1)

    issue_id.add(a)
    component.add(b)
    edge.append([b,a])

    if(a in link.keys()):
        link[a].add(b)

    else:
        link[a]=set()
        link[a].add(b)

for i in issue_id:
    for j in issue_id:
        if i!=j:     // as both B can't be same
            check=len(link[i].intersection(link[j]))
```

// similar is done with all the meta paths , with the length 3 .

Now for the 5 length paths , say B-T-B-C-B , we can use similar logic , i.e. finding explicility , if two bugs relate to one bug , one with the T related and other with C related then , increase their count by one.We can also use the 3 length paths .e.g. B-T-B and B-C-B ,to save the calculations.

Pseudocode ->

```
issue_id <- set()
component <- set()
link <- dict()
edge <- []
file <- input() // the database file of first relation here B-T file.
file_handle <- open(file,'r')
file_handle.readline()
for line in file_handle:
    a,b <- line.split(",")

    issue_id.add(a)
    component.add(b)
    edge.append([b,a])

    if(a in link.keys()):
        link[a].add(b)
    else:
        link[a] <- set()
        link[a].add(b)

ans1 <- dict()

for i in issue_id:
    for j in issue_id:
        if i!=j:  // not the same i and j
            check <- len(link[i].intersection(link[j]))
            if check:
                if i in ans1.keys():
                    ans1[i][j] <- int(check)
                else:
                    ans1[i] <- dict()
                    ans1[i][j] <- int(check)
```

```
file1 <- input()  // the database file of second relation here B-C file.
file_handle <- open(file1,'r')
file_handle.readline()
edge1 <- []
link1 <- dict()
component1 <- set()
issue_id1 <- set()

for line in file_handle:
    a,b=line.split(",")

    issue_id1.add(a)
    component1.add(b)
    edge1.append([b,a])

    if(a in link1.keys()):
        link1[a].add(b)
    else:
        link1[a] <- set()
        link1[a].add(b)

ans2 <- dict()

for i in issue_id1:
    for j in issue_id1:
        if i!=j:
            check <- len(link1[i].intersection(link1[j]))
            if check:
                if i in ans2.keys():
                    ans2[i][j] <- int(check)
                else:
                    ans2[i] <- dict()
                    ans2[i][j] <- int(check)

inter <- issue_id.intersection(issue_id1)

for i in issue_id:
    for j in issue_id1:
        if i==j:
            continue
        sum <- 0
        for k in inter:
            if i==k or j == k :
```

```
    continue
temp <- 0
if i in ans1.keys():
    if k in ans1[i].keys() and ans2.keys():
        if j in ans2[k].keys():
            a <- ans1[i][k]
            b <- ans2[k][j]
            temp <- a*b
sum  <- sum +temp
```

// this is done , to get the count of all length 5 paths.

FUNCTIONS ON META PATH

We are also interested to have four measures along the lines of topological features in homogeneous networks.These are path count, normalized path count, random walk, and symmetric random walk, which are defined as follows.

1• Path count.

Path count means the same as the word suggests.When following a particular meta path, the no. of instances found between any two vertices of the graph, is stored as path count, denoted as PCR(ai,aj),
where R is the relation between the two vertices of the meta path which relates the two vertices. Path count is the product of adjacency matrices associated with each relation in the meta path.

2• Normalized path count.

No. of paths in the overall connectivity for the two vertices in the graph, is the normalised path count, and is defined as:
NPCR(ai , aj) = PCR(ai,aj)+PCR~ (aj ,ai) PCR(ai,·)+PCR(·,aj) , where
R~ denotes the inverse relation of R,
PCR(ai , ·) is the total number of paths following relation R starting with ai .i.e. Covering all the possibilities for the end vertex. , and
PCR(·, aj) is the total number of paths following relation R ending with aj i.e. covering all the possibilities for the starting vertex..

3• Random walk.

Random walk for a particular path is ->

RWR(ai , aj) = PCR(ai,aj) PCR(ai,·) ,where PCR is the same as Path count , following the relation R.

4• Symmetric random walk.

Symmetric random walk considers both way random walk along the meta path->
SRWR(ai , aj) = RWR(ai , aj) + RWR~ (aj , ai).

Using Topic Modelling ToolKit in Python (Gensim)

1. Install the requirements using **pip3 install -r requirements.txt.**
2. First we will do lemmatization and stemming of training and test data using **Spacy.**
3. Select the train data (summary + description of Bug) and give it as input (insert into the corpse) to the Program.
4. Now Select the test data and number of topics
5. Run the Program you will get the first **k** mentioned topics from trained data (corpse).

Sample Data and Results

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: merged | New Record | Delete Record

	source	dest	BCB_ans.csv	BCBCB_ans.csv	BCBDB_ans.csv	BCBFB_ans.csv	BDB_ans.csv	BDBCB_ans
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	B2	B3	1	1	1	4	2	2
2	B2	B4	1	3	0	1	0	1
3	B2	B1	3	3	0	6	0	3
4	B2	B6	1	5	1	8	2	3
5	B2	B7	1	4	0	11	1	3
6	B3	B2	1	1	2	1	2	1
7	B3	B6	1	1	2	1	1	2
8	B4	B2	1	3	1	4	0	0
9	B4	B1	1	3	0	4	0	0
10	B4	B5	1	0	1	4	0	0
11	B1	B2	3	3	3	2	0	0
12	B1	B4	1	3	0	1	0	0
13	B1	B6	1	4	6	5	0	0
14	B1	B7	1	4	3	4	0	0
15	B5	B4	1	0	0	0	0	1
16	B5	B7	1	0	0	0	1	1

Combined Matrix of Sample Data

Table: temp_file

New Record | Delete Record

	issue_id	file
	Filter	Filter
1	B4	F1
2	B6	F1
3	B2	F2
4	B5	F2
5	B5	F3
6	B6	F3
7	B1	F4
8	B5	F4
9	B1	F5
10	B3	F5
11	B1	F6
12	B2	F6
13	B1	F3
14	B2	F4
15	B3	F5
16	B5	F4
17	B6	F4

Sample data for B-F edge

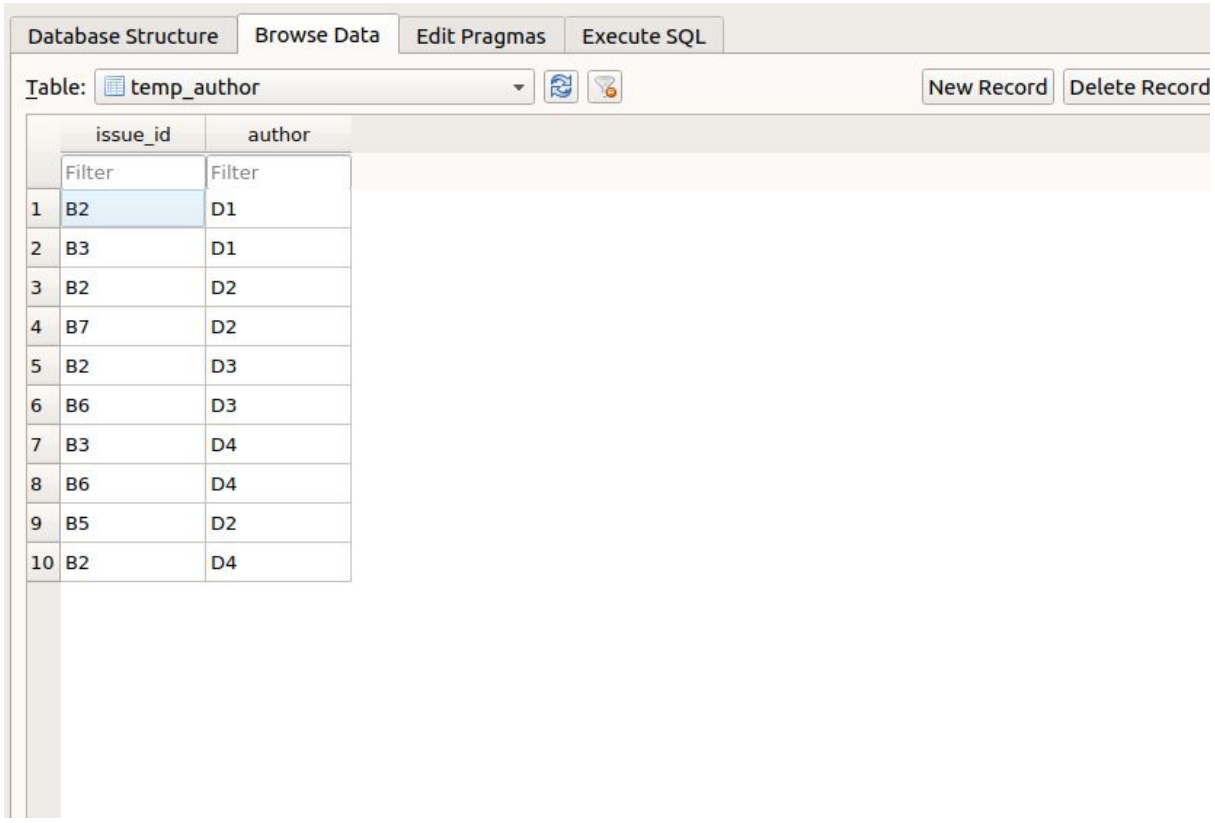
Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: temp_component

New Record | Delete Record

	issue_id	component
	Filter	Filter
1	B1	C1
2	B2	C1
3	B2	C2
4	B3	C2
5	B2	C3
6	B4	C3
7	B4	C4
8	B5	C4
9	B3	C5
10	B6	C5
11	B5	C6
12	B7	C6
13	B6	C7
14	B7	C7
15	B1	C7
16	B2	C7
17	B1	C3

Sample data for B-C edge



	issue_id	author
	Filter	Filter
1	B2	D1
2	B3	D1
3	B2	D2
4	B7	D2
5	B2	D3
6	B6	D3
7	B3	D4
8	B6	D4
9	B5	D2
10	B2	D4

Sample Data for B-D edge

Minutes Of Meetings

Meeting - 1

20 January 2020 / 7:00PM / BIG DATA ANALYTICS LAB,CC3

Attendees

Amit Kumar , Project Instructor
Harsh Goyal (IIT2018114)
Aaditya Gadhave (IIT2018144)
Sourabh Gupta (IIT2018149)
Meet Singh Gambhir (IIT2018158)
Tushar Atrey (IIT2018159)

Agenda

1. Introduction and briefing of the project.
2. Briefing on the manner of working.
3. Introduction to research paper and bugzilla.
4. Assigning task.

Notes

- The meeting started with introductions.
- The instructor informed about the manner of working and it was decided that 2 meetings per week, one each Monday and each Friday will take place.
- Then the task is assigned to our group.
- The instructor give introduction about the assigned task and what we have to do.
- The instructor informed about the heterogeneous network and its advantages over homogeneous network.
- Instructor explained the different type of meta paths with the help of author and research paper example.
- The instructor shows us the site bugzilla and tell us how it work.
- An overview of the dataset on which the work is to be done was given.
- Tasks to be completed until the next meeting were assigned

Action Items

1. Team is required to download the dataset and familiarize with it thus setting the environment of the project.
2. The team is required to study the research papers on heterogenous graph network.(to be circulated by the instructor).

Meeting - 2

29 January 2020 / 6:30PM/BIG DATA ANALYTICS LAB,CC3

Attendees

Amit Kumar ,Project Instructor
Harsh Goyal (IIT2018114)
Aaditya Gadhave (IIT2018144)
Sourabh Gupta (IIT2018149)
Meet Singh Gambhir (IIT2018158)
Tushar Atrey (IIT2018159)

Agenda

1. Briefing of the project
2. Implementation of the project
3. Introduction to meta paths
4. Different type of meta paths

Notes

- The meeting started with the briefing of the project
- The instructor informed about the heterogeneous network and its advantages over homogeneous network
- Instructor explained the different type of meta paths with the help of author and research paper example
- Instructor assigned the task i.e identifying the relationship among the bugs over the heterogeneous network using statistics , algorithms, testing sets etc.
- Tasks to be completed until the next meeting were assigned

Action Items

1. Team is required the find the meta paths between the bugs
2. In this stage of the project the direct paths relating to the bugs and enhancing their similarities . such as bug - file -bug , bug - comment - bug , etc were needed to be calculated , and keep their account.

Meeting - 3

03 February 2020 / 5:00PM / BIG DATA ANALYTICS LAB,CC3

Attendees

Amit Kumar , Project Instructor
Harsh Goyal (IIT2018114)
Aaditya Gadhave (IIT2018144)
Sourabh Gupta (IIT2018149)
Meet Singh Gambhir (IIT2018158)
Tushar Atrey (IIT2018159)

Agenda

1. Briefing of the work to be done next.
2. Count of meta paths.
3. Inspection of the work done.
4. Assigning task.

Notes

- The meeting started with the inspection of the work that is already done.
- Count the path that exist between bugs using algorithm like bfs or dfs.
- Instead of counting the path between two bug we have counted the path that exist from one bug so instructor has said it has to be corrected till the last meeting.
- Tasks to be completed until the next meeting were assigned.

Action Items

1. Team is required to update the MOM and correct the work that is already done.
2. Count the path of all types (in terms of related) that exists between two bugs.

Meeting - 4

10 February 2020 / 7:00PM/BIG DATA ANALYTICS LAB,CC3

Attendees

Amit Kumar ,Project Instructor
Harsh Goyal (IIT2018114)
Aaditya Gadhave (IIT2018144)
Sourabh Gupta (IIT2018149)
Meet Singh Gambhir (IIT2018158)
Tushar Atrey (IIT2018159)

Agenda

1. How to make sample graph on complex meta paths
2. Introduction to symmetric and asymmetric meta paths
3. Working on Hadoop database

Notes

- The meeting started with the briefing of the work we did earlier
- The instructor told us to make sample graphs on complex meta paths so that he can verify whether our program is working or not.
- Instructor explained the difference between symmetric and asymmetric meta paths.
- Instructor told us to work on hadoop database for our next task
- Instructor assigned the task i.e to make a table in which we have to store the count of different kinds of meta path between two bugs.
- Tasks to be completed until the next meeting were assigned

Action Items

1. Team is required to make a table in which we have to store the count of different kinds of meta paths between two bugs which was reported between 2010 and 2013(Training Data).
2. Team is required to make another table in which we have to store the count of all kinds of meta paths between the two bugs(one bug was reported between 2010 and 2013 and one bug was reported between 2014 and 2015 (Testing Data).

Meeting - 5

19 February 2020 / 7:00PM/BIG DATA ANALYTICS LAB,CC3

Attendees

Amit Kumar ,Project Instructor
Harsh Goyal (IIT2018114)
Aaditya Gadhave (IIT2018144)
Sourabh Gupta (IIT2018149)
Meet Singh Gambhir (IIT2018158)
Tushar Atrey (IIT2018159)

Agenda

1. Main agenda of this meeting was B - T - B metapath
2. Introduction to stemming and lemmatization
3. Instructor introduces us to Word n-gram , character n-gram and topic modeling.
4. Introduction to NLTK toolkit

Notes

- The meeting started with the briefing of the work we did earlier
- The instructor told us how to implement the B - T - B metapath
- Instructor told us that in stemming we remove stop words (is ,are,and ,etc) and in lemmatization we convert all the verbs into present form.
- In word n-gram we took n words subarray from two different text files and compared them . Ex. in word 1-gram we took 1 word subarray from files.
- In character n-gram we took n characters subarray from two different text files and compared them . Ex. in character 3-gram we took 3 characters subarray from files.
- Instructor gave an introduction to topic modeling.
- Tasks to be completed until the next meeting were assigned

Meeting - 6

4 March 2020 / 7:00PM/BIG DATA ANALYTICS LAB,CC3

Attendees

Amit Kumar ,Project Instructor
Harsh Goyal (IIT2018114)
Aaditya Gadhave (IIT2018144)
Sourabh Gupta (IIT2018149)
Meet Singh Gambhir (IIT2018158)
Tushar Atrey (IIT2018159)

Agenda

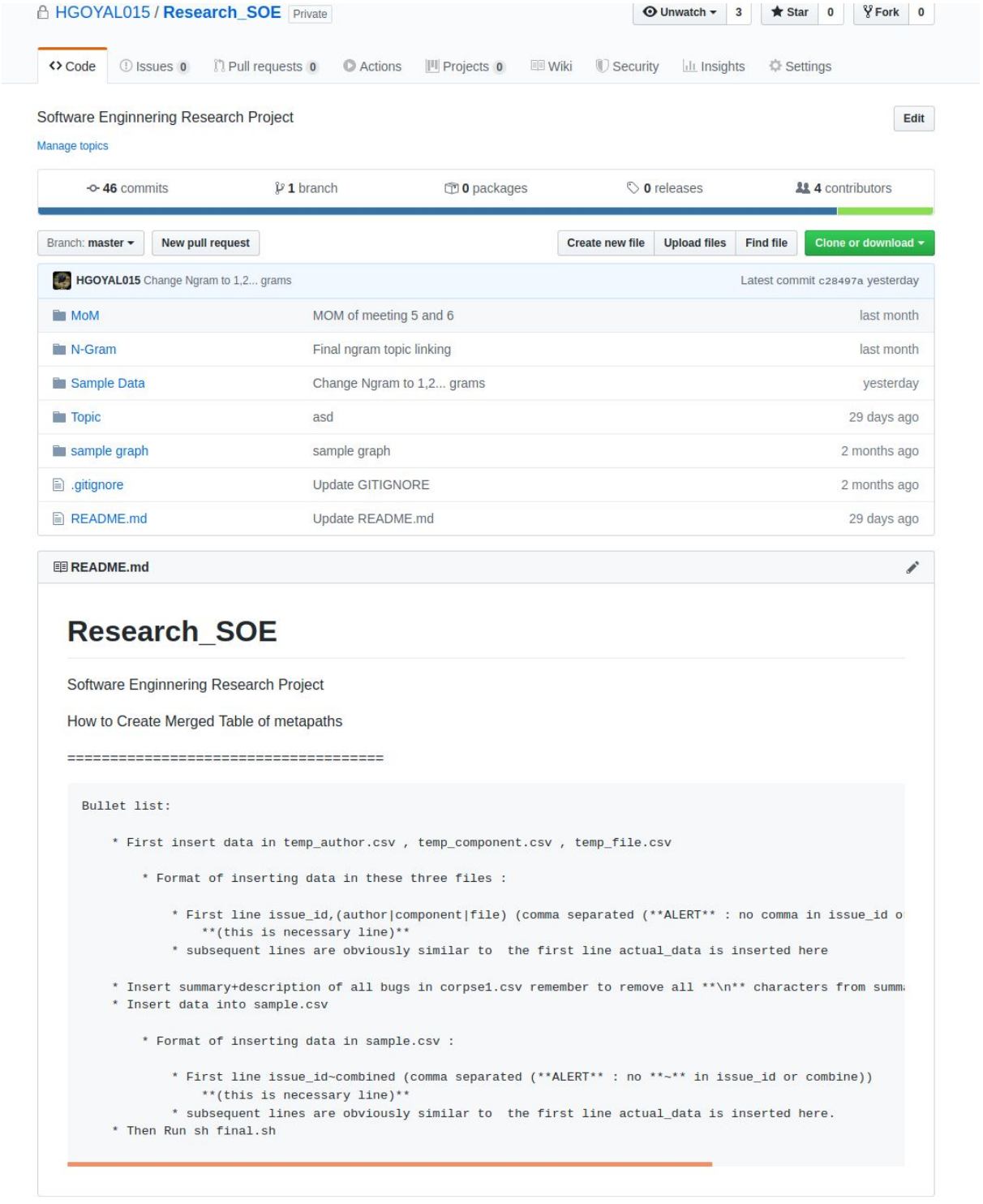
1. Main agenda of this meeting was topic modeling.
2. Getting familiar with Gensim , LDA++ and Mailet tools used for topic modeling
3. Introduction to WEKA and ARFAA file
4. Making the corpus and training the data

Notes

- The meeting started with the briefing of the work we did earlier
- The instructor told us how to make a corpus for the topic modeling tools.
- Instructor told us to divide the data into two parts :
  I) training data (70%)
  II) testing data (30%)
- Instructor told us how to train our data and how to test whether it is working correct or not
- Instructor assigned the task i.e to make the table in which rows will be the pairs of bugs and in column there will be different kinds of paths(mentioned in the meeting) . Each index value means the competitive measures between a bug pair.
- We have to make such four matrix because there are four methods of competitive measures(path count,normalized path count,random walk, symmetric random walk)

Appendices

SOE Project Github-Repository:



Further details and data about the project can be found under these links:

1. https://github.com/HGOYAL015/Research_SOE