

MAPPING THE EFFECTIVENESS OF AUTOMATED TEST SUITE GENERATION TECHNIQUES

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

**Gyanendra Raz, Mayank Kumar (REG.NO.
38110188, 38110313)**



**DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

APRIL - 2020



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai,
Chennai - 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Gyanendra Raz, Mayank Kumar (REG.NO. 38110188, 38110313)** who carried out the project entitled "**MAPPING THE EFFECTIVENESS OF AUTOMATED TEST GENERATION TECHNIQUES**" under our supervision from **Nov 2019** to **April 2020**.

Internal Guide

Dr.J.Albert Mayan, M.E., Ph.D

Head of the Department

Submitted for Viva voice Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **Gyanendra Raz,Mayank Kumar** (REG.NO. **38110188,38110313**)
hereby declare that the Project Report entitled **“MAPPINGTHE EFFECTIVENESS
OF AUTOMATED TEST GENERATION TECHNIQUES”**
done by us under the guidance of **Dr.J.Albert Mayan, M.E., Ph.D.**, Associate
professor, Department of CSE at Sathyabama Institute of Science and Technology is
submitted in partial fulfillment of the requirements for the award of Bachelor of
Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI-119.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr.T.SASIKALA, M.E., Ph.D., Dean, School of Computing and Dr.S.VIGNESWARI, M.E., Ph.D., and Dr.L.LAKSHMANAN, M.E., Ph.D., Heads of the Department of Computer Science and Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Dr.J.Albert Mayan Ph.D.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

In Software Engineering, the single point is to deliver great yield while improving the expense and the time expected to finish the application improvement. To accomplish this objective, programming groups will play out the test on their application before live creation. Experiments unite the entire testing process. Experiments are pivotal ancient pieces that function as a not too bad wellspring of truth for how a structure and a particular segment of programming works. This paper center around robotize experiment age dependent on accessible test assets, troubles which could be conveyed by methods for precise practice upgrade and characterizing test system for any product application. Automated Test Suite Generation is a significant subject in Software Engineering with a wide scope of procedures and apparatuses being utilized in the scholarly world and industry. While their helpfulness is broadly perceived, because of the work serious nature of the undertaking, the viability of the various strategies in consequently creating experiments for various programming frameworks isn't completely comprehended. In spite of numerous examinations presenting different ATSG strategies, much stays to be scholarly, be that as it may, about what makes a specific strategy function admirably (or not) for a particular programming system. This proposed application guarantees productive test inclusion where key usefulness won't be missed in the programmed experiment age.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	i
	LIST OF FIGURES	iv
	LIST OF ABBREVIATIONS	v
1.	INTRODUCTION	1
2.	LITERATURE SURVEY	4
3.	METHODOLOGY	7
	3.1 SYSTEM SPECIFICATION	7
	3.1.1 Back-end Technologies	7
	3.1.2 Front-end Technologies	7
	3.2 SOFTWARE DESCRIPTION	7
	3.2.1 About Python	7
	3.2.2 Python Line Structure	9
	3.2.3 Selection of IDE	11
	3.2.4 Anaconda	12
	3.2.5 Pandas	12
	3.2.6 Numpy	14
	3.3 EXISTING SYSTEM	16
	3.3.1 Disadvantages Of Existing System	17
	3.4 PROPOSED SYSTEM	17
	3.4.1 Advantages Of Proposed System	18
	3.5 META FRAMEWORK	19
	3.6 CUT SPACE	19
	3.6.1 Classes Under Test	20
	3.6.2 Cut Features	21
	3.7 ATSGT SPACE	22
	3.8 ATSGT EFFECTIVENESS SPACE	23

4.	RESULTS AND PERFORMANCE ANALYSIS	26
5.	CONCLUSION AND FUTURE WORK	29
	REFERENCES	30
	APPENDIX	34
	A.SOURCE CODE	34
	B.PAPER WORK	39

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
3.1	PYTHON ENVIRONMENT	7
3.2	PYTHON FUNCTIONALITIES	8
3.3	META FRAMEWORK	21
4.1	TEST GENERATION	22
4.2	LOGIN PAGE	26
4.3	DASHBOARD	27
4.4	FILES UPLOADING	27
4.5	UPLOADED SUCCESSFULLY	28
4.6	FINAL OUTPUT	29

LIST OF ABBREVIATIONS

ATSGT	-	AUTOMATED TEST SUITE GENERATION TECHNIQUES
CUT	-	CLASSES UNDER TEST
WSA	-	WHOLE SUITE WITH ARCHIVE
LDA	-	LATENT DIRICHLET ALLOCATION
IDE	-	INTEGRATED DEVELOPMENT ENVIRONMENT
CLI	-	COMMAND LINE INTERFACE

CHAPTER 1

INTRODUCTION

Programming deformity expectation has been one of the most appealing exploration points in the field of programming building. As per normal reports, semantic bugs bring about an expansion of use costs, inconvenience to clients, and even genuine results during sending. In this manner, confining and fixing absconds in beginning periods of programming advancement are pressing necessities. Different methodologies have been proposed to build models which can foresee whether a source code contains surrenders. These investigations can be partitioned into two bearings: one is applying AI systems on information of programming measurements that are physically intended to extricate highlights from source code; the other is utilizing projects' tree portrayals and profound figuring out how to naturally learn imperfection highlights.

ATSG assumes a basic job in the testing procedure because of the trouble of manual experiment age. These days, programming frameworks are progressively mind boggling and enormous, thus physically creating experiments that test all pieces of the framework is mentally requesting and tedious. Besides, ongoing programming advancement practices, for example, consistent conveyance and combination interest for compelling robotized ways to deal with testing. Because of its critical job in guaranteeing sans bug programming, ATSG methods began in the mid 70 then the quantity of works in the zone takes expanded from that point forward as appeared by these complete studies. While their helpfulness is broadly perceived, because of the work serious nature of the product testing, the appropriateness of the various methods in consequently creating experiments for various programming frameworks isn't completely comprehended.

Research presenting new methods or test examines exploring the presentation of various ATSGTs for the most part depends on a little arrangement of CUT. It present little data about the CUTs attributes and how they are chosen offering little understanding into the outside legitimacy of the discoveries. Intriguingly 33% of the papers consider solely compartment classes that are not normal in mechanical frameworks. Besides, most papers are just depicting the advantages of

the recently presented system and the advancement completed during improvement, while only a couple of notice the restrictions or present negative outcomes. It brings up issues as far as how ATSGTs are assessed, and whether the datasets utilized in exploratory examinations are sufficiently various to thoroughly quantify the capacities of the various procedures. Results asserting the better execution of an Automated Test Suite Generation over different strategies on a chose set of CUTs may not be generalizable than untested datasets. In a perfect world, a trial study would show a depiction of the conditions under which an ATSG system can be required to succeed or come up short, be that as it may, this is once in a while remembered for distributed examinations.

Here proposed toward seal this hole through presenting META structure to assess the adequacy of Automated Test Suite Generation through giving data around their reasonableness as indicated by the highlights of the product framework, which as an outcome, empowers the choice of the best strategy for a specific programming framework. The thought takes motivation from an imaginative system that has been effectively applied for assessing calculations for enhancement issues [14], [15]. Our proposed system, not at all like the usually utilized procedure in surveying the viability of ATSGTs, intends to describe the two qualities and shortcomings of the broke down ATSGTs utilizing existing and recently created programming highlights removed from the code and control stream diagram of 'CUTs'. The system additionally gives an away from of the calculations' conduct by indicating a 2D portrayal of the viability of ATSGTs on the component space through the use of dimensionality decrease strategies. Fundamentally the META system presented right now be utilized to describe the highlights of CUT occurrences that have sway on the ATSGT adequacy.

Generally, the commitments of this paper are as per the following:

- We present the META system, which is another methodology for surveying qualities and shortcomings of ATSGTs.
- Using the META structure, we recognize CUT highlights that affect ATSGT adequacy.

- We utilize the META structure to imagine the ATSGT impressions all through the CUT space.
- Finally, the META structure is utilized to manufacture a choice tree, which bolsters the choice of the best ATSGT to naturally produce tests for new programming frameworks dependent on their highlights. We lead a lot of trials on 211 non-unimportant JAVA classes chose from 6,432 classes and three ATSGTs: Random Testing (RT), Whole Suite with Archive (WSA) and Many-Objective Sorting Algorithm (MOSA). Experiments show that the META system can precisely recognize territories of solidarity and shortcoming in the CUT space for the distinctive ATSGTs. While in the analyses we utilize just three systems, which don't cover every single imaginable methods for naturally producing test suites, the META structure isn't strategy subordinate, and can be handily reached out with different ATSGTs.

CHAPTER 2

LITERATURE SURVEY

“Visual Classifier Training for Text Document Retrieval” (Florian Heimerl ; Steffen Koch ; Harald Bosch ; Thomas Ertl), here they developed Performing exhaustive searches over a large number of text documents can be tedious, since it is very hard to formulate search queries or define filter criteria that capture an analyst's information need adequately. Classification through machine learning has the potential to improve search and filter tasks encompassing either complex or very specific information needs, individually. Unfortunately, analysts who are knowledgeable in their field are typically not machine learning specialists. Most classification methods, however, require a certain expertise regarding their parametrization to achieve good results. Supervised machine learning algorithms, in contrast, rely on labeled data, which can be provided by analysts. However, the effort for labeling can be very high, which shifts the problem from composing complex queries or defining accurate filters to another laborious task, in addition to the need for judging the trained classifier's quality. We therefore compare three approaches for interactive classifier training in a user study. All of the approaches are potential candidates for the integration into a larger retrieval system. They incorporate active learning to various degrees in order to reduce the labeling effort as well as to increase effectiveness. Two of them encompass interactive visualization for letting users explore the status of the classifier in context of the labeled documents, as well as for judging the quality of the classifier in iterative feedback loops. We see our work as a step towards introducing user controlled classification methods in addition to text search and filtering for increasing recall in analytics scenarios involving large corpora.

“User-Driven Topic Modeling Based on Interactive Non-negative Matrix Factorization”. (Jaegul Choo ; Changhyun Lee ; Chandan K. Reddy ; Haesun Park), their approach is on Topic modeling has been widely used for analyzing text document collections. Recently, there have been significant advancements in various topic modeling techniques, particularly in the form of probabilistic graphical

modeling. State-of-the-art techniques such as Latent Dirichlet Allocation (LDA) have been successfully applied in visual text analytics. However, most of the widely-used methods based on probabilistic modeling have drawbacks in terms of consistency from multiple runs and empirical convergence. Furthermore, due to the complicatedness in the formulation and the algorithm, LDA cannot easily incorporate various types of user feedback. To tackle this problem, we propose a reliable and flexible visual analytics system for topic modeling called UTOPIAN (User-driven Topic modeling based on Interactive Non negative Matrix Factorization). Centered around its semi-supervised formulation, UTOPIAN enables users to interact with the topic modeling method and steer the result in a user-driven manner. We demonstrate the capability of UTOPIAN via several usage scenarios with real-world document corpuses such as InfoVis/VAST paper data set and product review data sets.

“A Multidimensional Brush for Scatter plot Data Analytics” (MichaelAupetit Nicolas Heulot ; Jean-Daniel Fekete), Brushing is a fundamental interaction for visual analytics. A brush is usually defined as a closed region of the screen used to select data items and to highlight them in the current view and other linked views. Scatter plots are also standard ways to visualize values for two variables of a set of multidimensional data. We propose a technique to brush and interactively cluster multidimensional data navigating through a single of their scatterplot projection.

“Progressive Learning of Topic Modeling Parameters: A Visual Analytics Framework.(Mennatallah El-Assady ; Rita Sevastjanova ; Fabian Sperrle ; Daniel Keim ; Christopher Collins), Topic modeling algorithms are widely used to analyze the thematic composition of text corpora but remain difficult to interpret and adjust. Addressing these limitations, we present a modular visual analytics framework, tackling the understandability and adaptability of topic models through a user-driven reinforcement learning process which does not require a deep understanding of the underlying topic modeling algorithms. Given a document corpus, our approach initializes two algorithm configurations based on a parameter space analysis that enhances document separability. We abstract the model complexity in an interactive visual workspace for exploring the automatic matching results of two models, investigating topic summaries, analyzing parameter distributions, and reviewing

documents. The main contribution of our work is an iterative decision-making technique in which users provide a document-based relevance feedback that allows the framework to converge to a user-endorsed topic distribution. We also report feedback from a two-stage study which shows that our technique results in topic model quality improvements on two independent measures.

“Security and Privacy in Decentralized Energy Trading through Multi-Signatures, Block-chain and Anonymous Messaging Streams”,(Nurzhan Zhumabekuly Aitzhan and Davor Svetinovic), this papaer is step towards Smart grids equipped with bi-directional communication flow are expected to provide more sophisticated consumption monitoring and energy trading. However, the issues related to the security and privacy of consumption and trading data present serious challenges. In this paper we address the problem of providing transaction security in decentralized smart grid energy trading without reliance on trusted third parties. We have implemented a proof-of-concept for decentralized energy trading system using block chain technology, mufti-signatures, andanonymous encrypted messaging streams, enabling peers to anonymously negotiate energy prices and securely perform trading transactions. We conducted case studies to perform security analysis and performance evaluation within the context of the elicited security and privacy requirements.

CHAPTER 3

METHODOLOGY

3.1 SYSTEM SPECIFICATION

3.1.1 Backend Technologies

- Python
- Numpy
- Sci-learn
- Eclipse IDE

3.1.2 Frontend Technologies

- Web Technologies
- Bootstrap

3.2 SOFTWARE DESCRIPTION

3.2.1 About Python

Python is a free, open-source programming language. Therefore, all you have to do is install Python once, and you can start working with it. Not to mention that you can contribute your own code to the community. Python is also a cross-platform compatible language. So, what does this mean? Well, you can install and run Python on several operating systems. Whether you have a Windows, Mac or Linux, you can rest assured that Python will work on all these operating systems.

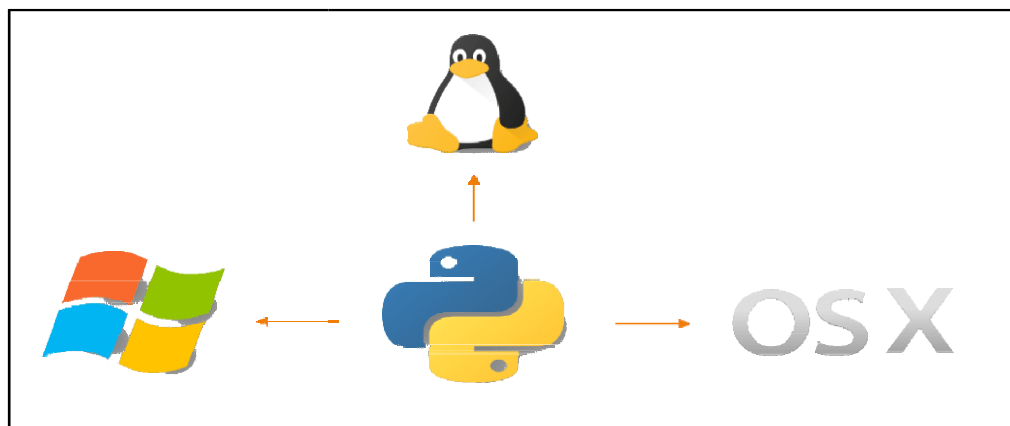


Fig.3.1: Python Environment

Python is also a great visualization tool. It provides libraries such as Matplotlib, seaborn and bokeh to create stunning visualizations.

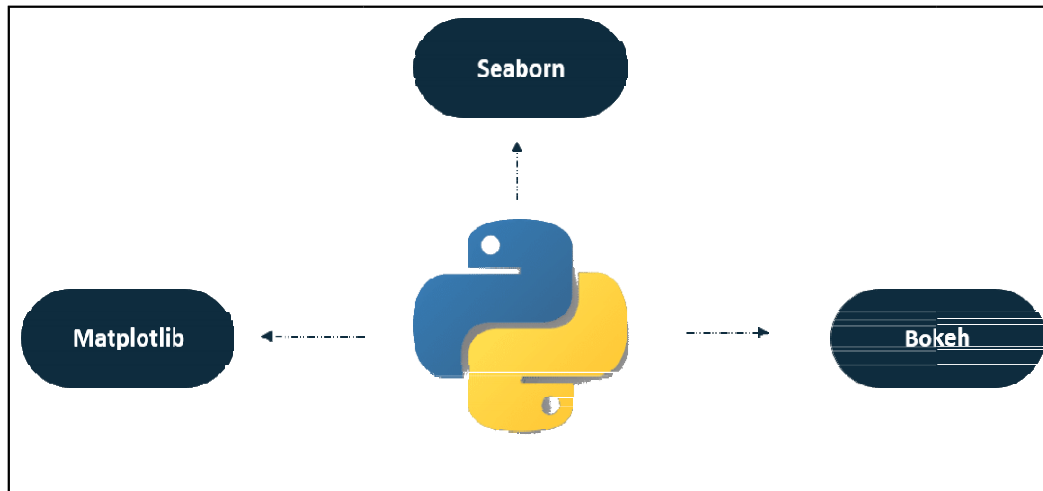


Fig.3.2: Python Functionalities

In addition, Python is the most popular language for machine learning and deep learning. As a matter of fact, today, all top organizations are investing in Python to implement machine learning in the back-end.

3.2.2 Python Line Structure

Python coding style comprises physical lines as well as logical lines or statements. A physical line in a Python program is a sequence of characters, and the end of the line terminates the line sequence as opposed to some other languages, such as C and C++ where a semi-colon is used to mark the end of the statement. A logical line, on the other hand, is composed of one or more physical lines. The use of a semi-colon is not prohibited in Python, although it's not mandatory. The NEWLINE token denotes the end of the logical line. A logical line that only contains spaces, comments, or tabs are called blank lines and they are ignored by the interpreter. As we saw that in Python, a new line simply means that a new statement has started. Although, Python does provide a way to split a statement into a multiline statement or to join multiple statements into one logical line. This can be helpful to increase the readability of the statement. Following are the two ways to split a line into two or more lines:

- Explicit Line Joining
- Implicit Line Joining

Statements that reside inside [], {}, or () parentheses can be broken down into two or more physical lines without using a back slash. Multiple Statements on a Single Line In Python, it is possible to club multiple statements in the same line using a semi-colon; however, most programmers do not consider this to be a good practice as it reduces the readability of the code. Whitespaces and Indentation Unlike most of the programming languages, Python uses indentation to mark a block of code. According to Python coding style guideline or PEP8, we should keep an indent size of four. Most of the programming languages provide indentation for better code formatting and don't enforce to have it. But in Python it is mandatory. This is why indentation is so crucial in Python. Comments in any programming language are used to increase the readability of the code. Similarly, in Python, when the program starts getting complicated, one of the best ways to maintain the readability of the code is to use Python comments. It is considered a good practice to include documentations and notes in the python syntax since it makes the code way more readable and understandable to other programmers as well, which comes in handy when multiple programmers are simultaneously working on the same project.

The code can only explain how it does something and not why it does that, but Python comments can do that. With Python comments, we can make documentations for various explanations in our code itself. Comments are nothing but tagged lines of codes which increase the readability of a code and make it self-explanatory. There are different ways of creating comments depending on the type of comment we want to include in our code. Following are different kinds of comments that can be included in our Python program:

- Single Line Comments
- Doc string Comments
- Multiline Comments

Single line Python comments are marked with # character. These comments end at the end of the physical line, which means that all characters starting after the # character (and lasts till the end of the line) are part of the comment. Python has the documentation strings (or docstrings) feature which is usually the first statement included in functions and modules. Rather than being ignored by the Python

Interpreter like regular comments, docstrings can actually be accessed at the run time using the dot operator. It gives programmers an easy way of adding quick notes with every Python module, function, class, and method. To use this feature, we use triple quotes in the beginning of the documentation string or comment and the closing triple quotes at the end of the documentation comment. Docstrings can be one-liners as well as multi-liners.

Unlike some programming languages that support multiline comments, such as C, Java, and more, there is no specific feature for multiline comments in Python. But that does not mean that it is totally impossible to make multiline comments in Python. There are two ways we can include comments that can span across multiple lines in our Python code.

Python Block Comments can use several single line comments for a whole block. This type of comment is usually created to explain the block of code that follows the Block comment. Python Block comment is the only way of writing a real comment that can span across multiple lines. It is supported and preferred by Python's PEP8 style guide since Block comments are ignored by Python interpreter or parser. However, nothing is stopping programmers from using the second 'non-real' way of writing multiline comments in Python which is explained below.

Docstrings are largely used as multiline comments in Python by many programmers since it is the closest thing to having a multiline comment feature in Python. While it is not wrong to use docstrings when we need to make multiline comments, it is important to keep in mind that there is a significant difference between docstrings and comments. Comments in Python are totally ignored by the Python Interpreter, while docstrings, when used inside the Python function, can be accessed at the run time.

3.2.3 Selection Of Integrated Development Environment

IDE- Integrated Development Environment. It is software suite that contains different different instruments for web application software engineers to compose and test applications. Programming web application software engineers utilize wide scope of devices habitually all through application source code creation, building and application testing. Programming Development apparatuses frequently incorporate word processors for source code creation, code libraries for reuse, compilers and test stages.

An Integrated Development Environment(IDE) is a product application that gives a programming domain to streamline creating and troubleshooting programming. Instead of playing out every one of the means required to make an executable program as disconnected individual assignments, it brings every one of the devices required into one application and workspaces. Every one of the devices has a familiarity with the earth, and they cooperate to display a consistent improvement set for the designer. Without an IDE, a product web application software engineer must choose, send, incorporate and deal with these instruments independently. An IDE brings huge numbers of those improvement related instruments all together system, application of administration. The incorporated toolset is intended to streamline programming advancement and can distinguish and limit coding and grammatical errors.

- ✓ Komodo
- ✓ Net Beans
- ✓ PyCharm
- ✓ Python for Visual Studio.NET
- ✓ PyDev
- ✓ PyStudio

3.2.4 Anaconda

Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator as a graphical alternative to the command line interface (CLI).The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

3.2.5 Pandas

Pandas is a popular Python package for data science, and with good reason: it offers powerful, expressive and flexible data structures that make data manipulation and analysis easy, among many other things. The DataFrame is one of these structures.Pandas is a high-level data manipulation tool developed by Wes

McKinney. It is built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables. Pandas is built on top of the NumPy package, meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed statistical analysis in SciPy, plotting functions from Matplotlib, and machine learning algorithms in Scikit-learn.

Jupyter Notebooks offer a good environment for using pandas to do data exploration and modeling, but pandas can also be used in text editors just as easily. Jupyter Notebooks give us the ability to execute code in a particular cell as opposed to running the entire file. This saves a lot of time when working with large datasets and complex transformations. Notebooks also provide an easy way to visualize pandas' DataFrames and plots. As a matter of fact, this article was created entirely in a Jupyter Notebook. There are two types of data structures in pandas: Series and DataFrames.

1. Series: a pandas Series is a one dimensional data structure (“a one dimensional ndarray”) that can store values — and for every value it holds a unique index, too.

2. DataFrame: a pandas DataFrame is a two (or more) dimensional data structure – basically a table with rows and columns. The columns have names and the rows have indexes.

Those who are familiar with R know the data frame as a way to store data in rectangular grids that can easily be overviewed. Each row of these grids corresponds to measurements or values of an instance, while each column is a vector containing data for a specific variable. This means that a data frame's rows do not need to contain, but can contain, the same type of values: they can be numeric, character, logical, etc. Now, DataFrames in Python are very similar: they come with the Pandas library, and they are defined as two-dimensional labeled data structures with columns of potentially different types.

In general, you could say that the Pandas DataFrame consists of three main components: the data, the index, and the columns.

Firstly, the DataFrame can contain data that is:

- a Pandas DataFrame

- a Pandas Series: a one-dimensional labeled array capable of holding any data type with axis labels or index. An example of a Series object is one column from a DataFrame.
- a NumPy ndarray, which can be a record or structured
- a two-dimensional ndarray
- dictionaries of one-dimensional ndarray's, lists, dictionaries or Series.

Some of the key features of Python Pandas are as follows:

It provides DataFrame objects with default and customized indexing which is very fast and efficient.

- There are tools available for loading data of different file formats into in-memory data objects.
- It is easy to perform data alignment and integrated handling of missing data in Python Pandas.
- It is very simple to perform pivoting and reshaping of data sets in Pandas.
- It also provides indexing, label-based slicing, and sub-setting of large data sets.
- We can easily insert and delete columns from a data structure.
- Data aggregation and transformations can be done using group by.
- High-performance merging and joining of data can be done using Pandas.
- It also provides time series functionality.
- Inserting and deleting columns in data structures.
- Merging and joining data sets.
- Reshaping and pivoting data sets.
- Aligning data and dealing with missing data.
- Manipulating data using integrated indexing for DataFrame objects.
- Performing split-apply-combine on data sets using the group by engine.
- Manipulating high-dimensional data in a data structure with a lower dimension

using hierarchical axis indexing.

- Subsetting, fancy indexing, and label-based slicing data sets that are large in size.
- Generating data range, converting frequency, date shifting, lagging, and other time-series functionality.
- Reading from files with CSV, XLSX, TXT, among other formats.
- Arranging data in an order ascending or descending.
- Filtering data around a condition.
- Analyzing time series.
- Iterating over a data set.

3.2.6 Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy. A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension. NumPy is, just like SciPy, Scikit-Learn, Pandas, etc. one of the packages that you just can't miss when you're learning data science, mainly because this library provides you with an array data structure that holds some benefits over Python lists, such as: being more compact, faster access in reading and writing items, being more convenient and more efficient.

NumPy is a Python library that is the core library for scientific computing in Python. It contains a collection of tools and techniques that can be used to solve on a computer mathematical models of problems in Science and Engineering. One of these tools is a high-performance multidimensional array object that is a powerful data structure for efficient computation of arrays and matrices. To work with these arrays, there's a vast amount of high-level mathematical functions operate on these

matrices and arrays. an array is basically nothing but pointers. It's a combination of a memory address, a data type, a shape, and strides:

- The data pointer indicates the memory address of the first byte in the array,
- The data type or dtype pointer describes the kind of elements that are contained within the array,
- The shape indicates the shape of the array, and
- The strides are the number of bytes that should be skipped in memory to go to the next element. If your strides are (10,1), you need to proceed one byte to get to the next column and 10 bytes to locate the next row.

Or, in other words, an array contains information about the raw data, how to locate an element and how to interpret an element.

With NumPy, we work with multidimensional arrays. We'll dive into all of the possible types of multidimensional arrays later on, but for now, we'll focus on 2-dimensional arrays. A 2-dimensional array is also known as a matrix, and is something you should be familiar with. In fact, it's just a different way of thinking about a list of lists. A matrix has rows and columns. By specifying a row number and a column number, we're able to extract an element from a matrix. We can create a NumPy array using the `numpy.array` function. If we pass in a list of lists, it will automatically create a NumPy array with the same number of rows and columns. Because we want all of the elements in the array to be float elements for easy computation, we'll leave off the header row, which contains strings. One of the limitations of NumPy is that all the elements in an array have to be of the same type, so if we include the header row, all the elements in the array will be read in as strings. Because we want to be able to do computations like find the average quality of the wines, we need the elements to all be floats.

NumPy has several advantages over using core Python mathematical functions, a few of which are outlined here:

- ✓ NumPy is extremely fast when compared to core Python thanks to its heavy use of C extensions.
- ✓ Many advanced Python libraries, such as Scikit-Learn, Scipy, and Keras, make

extensive use of the NumPy library. Therefore, if you plan to pursue a career in data science or machine learning, NumPy is a very good tool to master.

NumPy comes with a variety of built-in functionalities, which in core Python would take a fair bit of custom code.

3.3 EXISTING SYSTEM

The CUT space is stressed over the item structures and their characteristics. Programming systems are considered at a class level, and features are evaluated for each class. The segment extraction procedure showed up. It very well may be propelled with existing instruments for feature extraction, for instance, jCT and CKJM. At the present time, present new estimations that depend on CFG incorporates and energized from outline speculation. The part extraction framework for these estimations can be performed using the item group NetworkX. In the present system, tests are delivered truly. Analysis age frameworks for the most part produce tests from the going with resources: Requirement assurance document, UML blueprint and Source code. Examination Generation Techniques has two methodology: Define and Design.

3.3.1 Disadvantages of Existing System

- Less Accuracy
- Lacks named ground truth information
- No ideal way to a base expense
- High Computational Cost
- Require complex drug designs

3.4 PROPOSED SYSTEM

The principle objective of Control Flow Analysis (CFA) is utilized to decide the execution stream of a program articulations or guidelines. The hubs of a stream diagram are fundamental squares. Essential square is an immediate arrangement of program directions having one section point and one leave point. A Path fragment is subsequence of hubs along the way. Follow is a progression of directions performed during execution. A Control Flow Graph (CFG) is the graphical portrayal of control

stream or calculation during the execution of undertakings or applications. Control stream charts are commonly used in static assessment similarly as compiler applications, as they can decisively address the stream inside a program unit. In a control stream diagram, two exceptionally allotted squares exist: passage and leave squares. The passage square empowers the control to go into the control stream diagram, however the control stream leaves through the leave square. Toward the day's end, the control stream chart is contained all the structure squares drew in with a stream graph, for instance, the starting center point, end center point and streams/round sections between the centers.

3.4.1 Advantages Of The Proposed System

- Low Computational Cost
- Low memory prerequisite
- Prediction stage may be quick
- High exactness (moderately) – it is really high
- Versatile valuable for characterization

3.5 META FRAMEWORK

The proposed META framework has three main components:

- 1.The CUT space, which includes the CUTs and the structural-based complexity features extracted from the CUTs;
- 2.The ATSGT space, which contains the selected ATSGT and the metrics used to assess their performance (e.g., generation time, branch coverage, mutation coverage, etc.);
- 3.The ATSGT effectiveness space, which identifies CUT features that have an impact on the effectiveness of ATSGTs and maps the strengths and weaknesses of the ATSGTs as footprints in a 2-D space (as we will explain in the following sections). The META framework is inspired from the algorithm selection problem introduced in the area of optimization and machine learning and the No Free Lunch theorem, which tell us that there does not exist a single algorithm that can be expected to outperform all other algorithms on all problem instances. Hence, if

method A is superior over method B in solving a particular set of problems, then one may claim that there exist other untested problems where method B may outperform method A. Empirical studies in the area of ATSG should focus on identifying conditions, under which an algorithm is expected to succeed or fail instead of claiming superiority of a method over another. We have extended and adapted these ideas to assess the strengths and weaknesses of ATSGTs and built the META tool. We have built the META tool, which can be used to predict which ATSGT from a portfolio of different techniques is likely to perform best based on measurable features of a collection of CUTs.

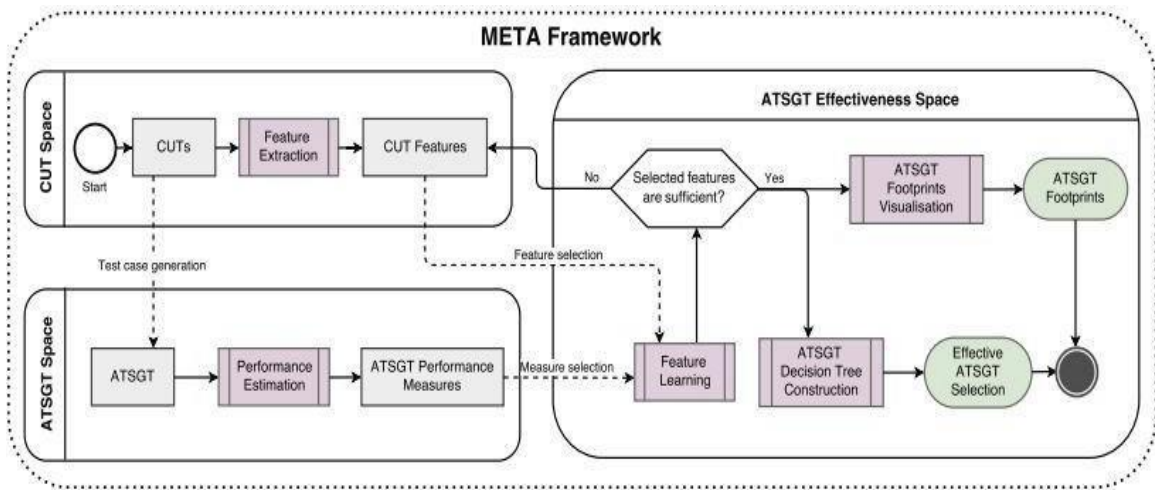


Fig:3.3:Meta Framework

3.6 CUT SPACE

The CUT space is concerned with the software systems and their characteristics. Software systems are considered at a class level, and features are measured for each class. The feature extraction procedure can be instantiated with existing tools for feature extraction, such as jCT and CKJM. In this work, we introduce new metrics that are based on CFG features and inspired from graph theory. The feature extraction procedure for these metrics can be performed using the software package NetworkX.

3.6.1 Classes Under Test

The CUT space considers the CUTs, which are the input to our META framework. Our framework does not impose restrictions on the programming language and can

be applied to other programming languages for which automated test case generation approaches exist. For the purpose of this paper, however, we focus on JAVA programming language.

3.6.2 CUT Features

Features are problem dependent and must be chosen so that the varying complexities of the CUT instances are exposed, any known structural properties of the CUTs are captured, and any known advantages and limitations of the different ATSGTs are related to features. The most common measures and metrics used to characterize features of CUTs are extracted from the code. Other software features are extracted from the CFG, which represents the structure of code. The most common metric that is based on the CFG is the CC, which analyzes the branching complexity of the CFG. Our approach uses well-established software measures, such as CC, and introduces new measures inspired from graph theory to analyze different aspects of the CFG. These metrics can provide important information about graph complexity. In essence, the software features considered in this work can be classified into code-based and graph-based features. The first group of objected oriented code metrics were proposed based in measurement theory and expertise of experienced software developers. The set includes simple features that simply count the number of methods or lines of code to more elaborated features that measure the interaction between methods and the depth of inheritance tree.

The selected set includes features used in graph theory and also features created specifically for software systems like CC [52]. The CFG is a directed graph; however, we also use undirected graph features by removing the direction restriction. The CFG is generated per method; therefore, to generate values for the whole CUT, metrics are reported as average, standard deviation, sum, maximum, and minimum values. The full list of CFG-based features is presented As an illustration, A CFG with three graph measurements. The first graph, from left to right, presents the degree of the nodes, i.e., the number of adjacent edges for each. The second graph presents the nodes eccentricity that is the maximum distance between a node and any other node in the graph. The last graph shows the cluster

coefficient of the nodes, i.e., the degree to which the nodes tend to cluster together. The darker the shade, the higher are the values of the measure.

3.7 ATSGT SPACE

The ATSGT space includes the automated software test generation techniques and the measures used to evaluate their performance.

1.ATSGs: Three techniques were selected to be part of the study. The first technique is called WSA and uses an SBST approach that is considered one of the most promising and has been exhaustively investigated by the community. The second method is also part of the SBST category and is known as MOSA . It introduces a novel preference criterion based on the Pareto optimality algorithm. The third method is RT, which is recommended as a comparison baseline to evaluate new techniques. RT is generally regarded as not only a simple but also an intuitively appealing technique . In RT, test cases are randomly generated based on a uniform distribution or according to the operational profile. A pool of randomly generated inputs is maintained, and a probability is used for either reusing a generated test case or creating a new one. The main merits of RT include the availability of efficient algorithms to generate test cases and its ability to infer reliability and statistical estimates . It has been shown that RT can be cost effective in many cases, providing a very high segment and branch coverage . For more complex software or more sophisticated criteria, RT may simply fail to generate appropriate data in any reasonable time frame. Due to its simplicity and ease of implementation, RT is a desirable technique for automated test case generation. There are no clear guidelines, however, to inform practitioners about its effectiveness on particular software systems with particular features. Furthermore, there is still a strong lack of understanding of how exactly the performance of RT depends on software characteristics. WSA is an SBST method. SBST techniques employ search algorithms, such as hill climbing , simulated annealing and GAs, to automate or partially automate a testing task, e.g., the automatic generation of test data. Key to the optimization process is a problem-specific fitness function. The role of the fitness function is to guide the search to high quality solutions from a potentially very large search space, within a practical time limit, which is why these methods are often known as fitness-guided automated testing. Work on SBST dates back to

1976 , with interest in the area beginning to gather pace in the 1990s. WSA is implemented in EvoSuite and uses monotonic GA. To measure the quality of a solution, WSA uses branch and method coverage as a fitness function . The aim is to cover as many control structures, such as if, while statements, and methods as possible, by evaluating the logical predicates that result in both true and false. In order for all branches to be covered, each predicate must be executed at least twice, resulting in true and false values. In the considered SBST method, the branch distance $d(b, T)$ for each branch b in test suite T is defined as

$$f(b, T) = a_1 + \text{norm}(d(b, T)).$$

0, if a branch has been covered d

b, T

$\min$, if a predicate is executed at least twice 1, otherwise.

The distance $d(b, T)$ is 0 if at least one of the branches has been covered, and > 0 otherwise. The fitness function that is minimized by WSA is $f(T) = |M| - |MT| + \sum_{b \in B} d(b, T)$

$d(b, T)$

where $|M|$ is the total number of methods, $|MT|$ is the number of executed methods in test suite T , and B is the set of branches in the program. WSA starts by generating a set of solutions, which are uniformly randomly initialized. Formally, let t denote a test case, which consists of a sequence of statements $t = s_1, s_2, \dots, s_l$ of length l . A statement s_i can be a constructor, a field, a primitive, a method, or an assignment. A solution is defined as a test suite T , which is a collection $T = \{t_1, t_2, \dots, t_n\}$ of test cases. An optimal solution $T_{\square}$ is a test suite that covers all possible branches and lines of code, i.e., 100% coverage. Since the number of test cases in a test suite and the number of statements in a test case may vary, the solution representation is of a variable size. The solutions are evolved in iterations until a stopping criterion is achieved, which usually is a predefined number of function evaluations. The GA used by EvoSuite has four genetic operators that are applied to solutions at every iteration: crossover, mutation, selection, and replacement. Crossover creates two new solutions by combining test cases from two test suites in the population. The mutation operator is applied after the crossover operator at a test suite level and

at a test case level. Test suites are mutated by changing each of the test cases with a probability $1/n$, where n is the number of test cases in the test suite. In addition, new test cases are added to the test suite at random. Mutation of test cases is performed by either adding, changing, or removing statements from a test case with a probability $1/l$, where l is the number of statements in a test case. A rank-based selection procedure is employed to select the parent solutions that will undergo recombination and mutation procedures. Solutions are ranked based on the fitness function. When there is a tie between solutions, shorter test suites are assigned better ranks. As a result, solutions with better branch coverage and shorter length have a higher chance of projecting their “genes” to the next generation. Similar to the original studies with EvoSuite, an elitist strategy is used as a replacement procedure. The elitist strategy selects the best solutions to create the next generation. MOSA is a many-objective GA used for automated test case generation. MOSA starts with an initial set of randomly generated test cases that evolve via crossover and mutation. The selection scheme considers both the nondominance relation and the proposed preference criterion, which determines the test case with the lowest objective score (branch distance + approach level) for each uncovered branch. Test cases are ranked according to the nondominated sorting algorithm used by the Nondominated Sorting Genetic Algorithm II. Next, the crowding distance is used in order to decide which test case to select: the test cases with a higher distance from the rest of the population have a higher probability of being selected. Test cases are added to the next generation based on their ranks. MOSA is implemented in Evosuite [17], and the objective function $f(b, T)$ for each branch $b \in B$ in test suite T is defined as follows:

$$f(b, T) = a_1 + \text{norm}(d(b, T)).$$

The branch distance $d(b, T)$ is computed using (2). The overall objective function of a test suite T with respect to all branches is computed as $f(T) = |M| - |MT| + \sum_{b \in B} f(b, T)$

$$f(b, T)$$

where M is the set of methods in T and MT is the set of methods executed during the test.

ATSGT Performance Measure: There are some commonly used metrics to evaluate the quality of test suites. The most common is the code coverage, i.e., the degree to which a software system is tested by a particular test suite. Mutation testing is another technique commonly used to evaluate test suites. It involves changing a program slightly with mutation operators. Each new version generated is called a mutant, and tests detect and reject mutants (kill mutants) by causing the behavior of the original version to differ from the mutant. Test suites are measured by the percentage of mutants that they kill. For the purpose of this study, we use branch coverage, also known as decision coverage, which quantifies the percentage of branches/decisions that have been executed by the test suite. The term branch/decision means that two or more exits are possible from statements like an IF or a CASE, generating always two outcomes: either true or false. The branch coverage is the most commonly used metric to evaluate ATSGT performance.

3.8 ATSGT EFFECTIVENESS SPACE

The ATSG effectiveness space models the interaction between the CUT space and ATSGT space. This is the part of our framework that enables us to answer our research question: What are the software features that are likely to impact ATSGT effectiveness? Once the critical features have been identified through the feature learning procedure, they are used to analyze and visualize the footprints of the ATSGTs, which help understand strengths and weaknesses of algorithms. The feature learning procedure identifies features that capture the difficulty of CUTs. A training set of CUTs is used to learn critical features. This helps us answer the question: Which ATSGTs in a portfolio are likely to be best for a relevant set of CUTs, and which ones are likely to produce suboptimal results? To answer this question, the META framework visualizes the footprints of the different ATSGTs across the CUT space using dimensionality reduction techniques. Finally, the META framework creates a DT based on the most significant CUT features, which can be used to predict and select the most effective ATSGT for generating test cases for new software projects. Its accuracy is evaluated in a validation set of CUTs.

CHAPTER 4

RESULT AND PERFORMANCE ANALYSIS

As can be seen, the overall performances of the three ATSGTs are very similar, with a median coverage between 0.41 and 0.5. The standard deviation of the different techniques is also very similar. These statistics indicate that, overall, there is a little difference between the effectiveness of the three techniques in generating high-quality test suites. The question, however, remains whether for particular CUTs, a certain ATSGT is better than the others.

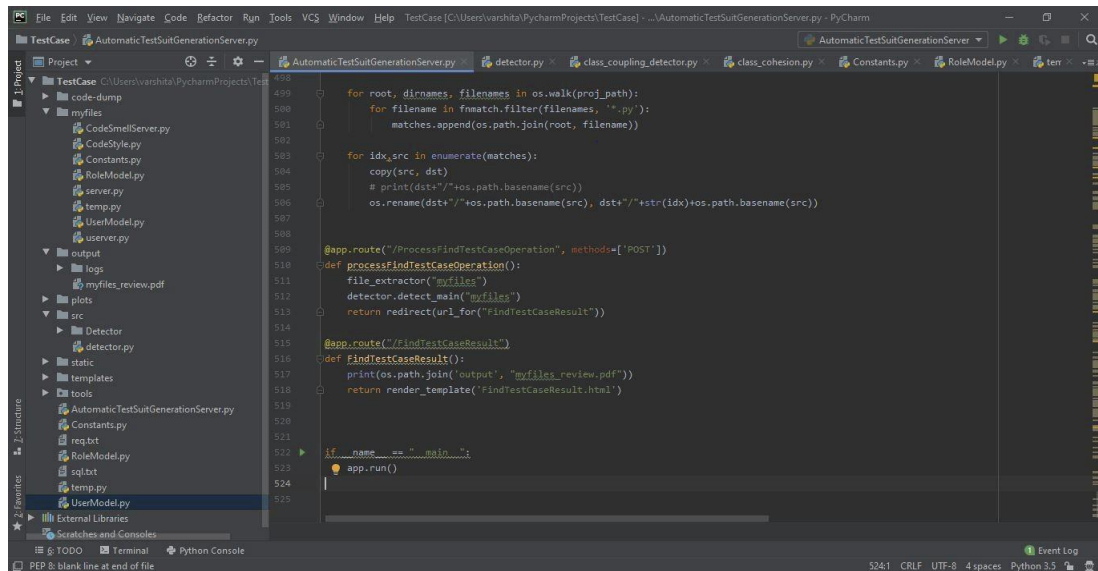


Fig.4.1:Test Generation

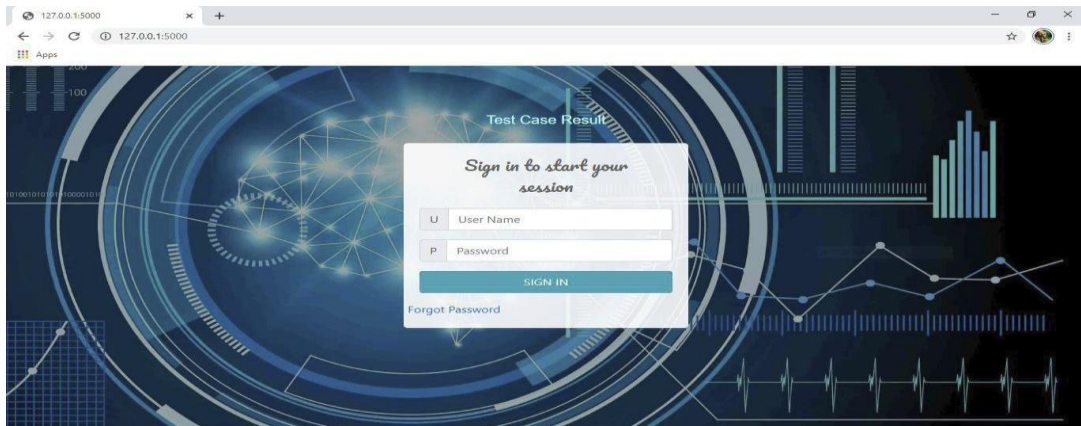


Fig.4.2:Login Page

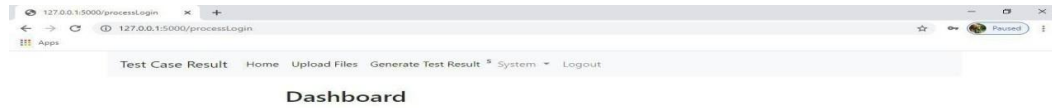


Fig.4.3:Dashboard

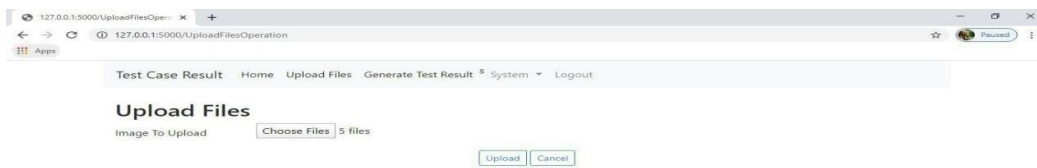


Fig.4.4:Files Uploading

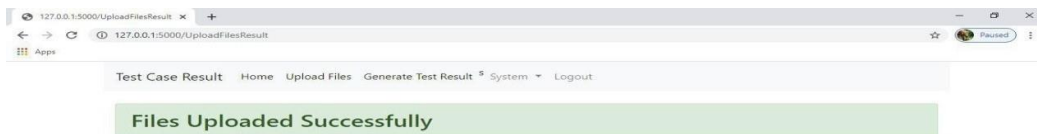


Fig.4.5:Uploaded Successfully

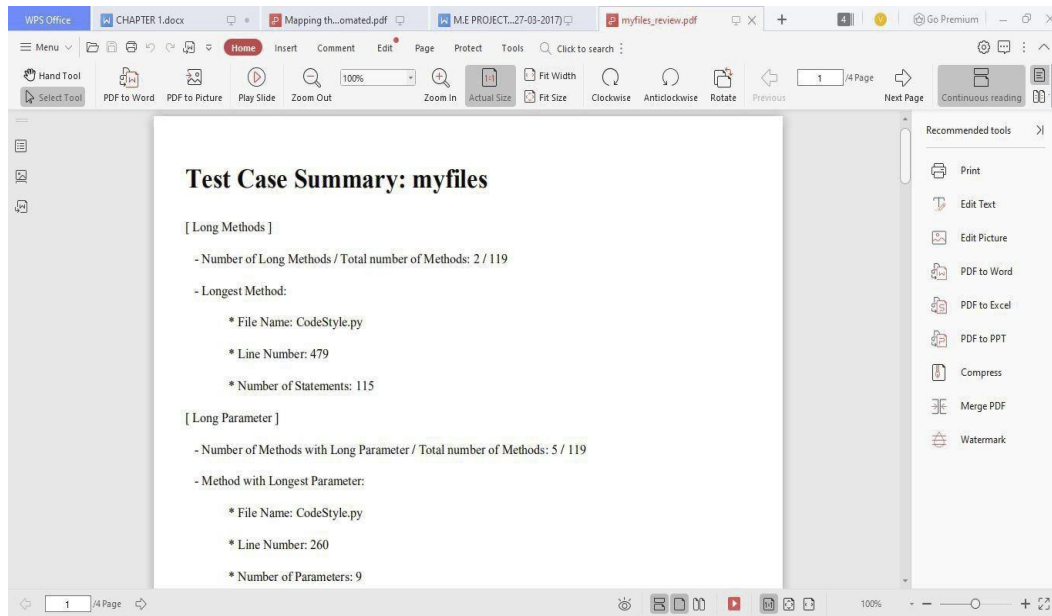


Fig.4.6:Final Output

CHAPTER 5

CONCLUSION AND SUMMARY

Given a portfolio of ATSGTs, the META framework can be used to select the technique that is likely to be best for a relevant set of CUT instances. Using the framework, we uncovered relationships between features of CUTs and the effectiveness of ATSGTs. We identified three significant features that impact ATSGT effectiveness, namely the response for a class, the coupling between object classes, and the number of methods in a class. Using these three significant features, we constructed an ATSGT DT, which identified the correct technique in more than 88% of the cases. In the future, including dynamic symbolic execution as a fourth technique is a priority. Furthermore, we will investigate more datasets from open-source projects and industrial case studies and create a repository of CUTs and their features that can be used by other researchers to test the effectiveness of new ATSGTs.

Quality and reliability are the two most important parameter in a software development field. In this paper we describe the theoretical aspects of various software testing techniques. We analyze both the manual and automation testing, both are preferable in order to reduce bugs in software but different prospects such that if same test cases using repeatedly number of time, then instead of using manual testing we go through test automation to create scripts or programs. Through this paper we present an evaluation of our testing techniques with the help of example. In future, we believe that it is important to improve the research efforts in software testing field for improving the quality and efficiency of software and emphasis the enhancement of open source testing software.

REFERENCES

- [1] Perez J.A., Poon C.Y., Merrifield R.D., Wong S.C., Yang G.Z. Big data for health. *IEEE J. Biomed. Health Inform.* 2015;19:1193-1208. doi: 10.1109/JBHI.2015.2450362.
- [2] Erdman A.G., Keefe D.F., Schiestl R. Grand challenge: Applying regulatory science and big data to improve medical device innovation. *IEEE Trans. Biomed. Eng.* 2013;60:700-706. doi: 10.1109/TBME.2013.2244600.
- [3] Esteban R.R., Logging W.T. Quantifying the complexity of medical research. *Bioinformatics.* 2013;29:2918-2924. doi: 10.1093/bioinformatics/btt505.
- [4] Köpcke F., Trinczek B., Majeed R.W., Schreiweis B., Wenk J., Leusch T. Evaluation of data completeness in the electronic health record for the purpose of patient recruitment into medical trials: A retrospective analysis of element presence. *BMC Med. Inform. Dec. Mak. Med. Syst.* 2013;13:1-8. doi: 10.1186/1472-6947-13-37.
- [5] Zhu Q., Tao C., Ding Y. Time-related patient data retrieval for the case studies from the pharmacy genomics research network. *J. Med. Syst.* 2012;36:37-42. doi: 10.1007/s10916-012-9888-1.
- [6] Madkour M., Benhaddou D., Tao C. Temporal data representation, normalization, extraction, and reasoning: A review from medical domain. *Comput. Methods Program Biomed.* 2016;128:52-68. doi: 10.1016/j.cmpb.2016.02.007.
- [7] Karystianis G., Sheppard T., Dixon W.G., Nenadic G. Modeling and extraction of variability in free-text medication prescriptions from an anonymised primary care electronic medical record research database. *BMC Med. Inform. Decis. Mak.* 2016;16:1-8.
- [8] Shah D., Martinez C., Hemingway H. The free text matching algorithm: A computer program to extract diagnoses and causes of death from unstructured text in electronic health records. *BMC Med. Inform. Decis. Mak.* 2012;12:2010-2026. doi: 10.1186/1472-6947-12-88.
- [9] Onitilo A.A., Stankowski R.V., Berg R.L. A novel method for studying the temporal relationship between type 2 diabetes mellitus and cancer using the electronic

medical record. BMC Med. Inform. Decis. Mak. 2014;14:1661-1670. doi: 10.1186/1472-6947-14-38.

[10] Reeves R.M., Ong F.R., Matheny M.E., Denny J.C., Gobbel G.T., Montella D. Detecting temporal expressions in medical narratives. Int. J. Med. Inform. 2013;82:118-127. doi: 10.1016/j.ijmedinf.2012.04.006.

[11] NiklasNorén G., Hopstadius J., Bate A. Temporal pattern discovery in longitudinal electronic patient records. Data Min. Knowl. Discov. 2010;20:361-387.

[12] Hanauer D.A., Ramakrishnan N. Modeling temporal relationships in large scale medical associations. J. Am. Med. Inform. Assoc. 2013;20:332-341. doi: 10.1136/amiajnl-2012-001117.

[13] Zhou L., Melton G.B., Parsons S., Hripcsak G. A temporal constraint structure for extracting temporal information from medical narrative. J. Biomed. Inform. 2006;39:424-439. doi: 10.1016/j.jbi.2005.07.002.

[14] Zhang Y., Gou L., Tian Y., Li T., Zhang M., Li J. Design and development of a sharable medical decision support system based on a semantic web service framework. J. Med. Syst. 2016;40:1-14. doi: 10.1007/s10916-016-0472-y.

[15] Singh A., Nadkarni G., Gottesman O., Ellis S.B., Gutttag J.V. Incorporating temporal EHR data in predictive models for risk stratification of renal function deterioration. J. Biomed. Inform. 2015;53:220-228. doi: 10.1016/j.jbi.2014.11.005.

APPENDIX

A. SOURCE CODE

```
Import numpy as np
```

```
Import pandas as pd
```

```
From plotly import tools
```

```
Import plotly.figure_factory as ff
```

```
Import plotly.graph_objs as go
```

```
From plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
```

```
Init_notebook_mode(connected=True)
```

```
Import matplotlib.pyplot as plt
```

```
Import seaborn as sns
```

```
From string import ascii_letters
```

```
From scipy.stats import norm
```

```
From scipy.stats import skew
```

```
From scipy.stats.stats import pearsonr
```

```
From scipy import stats
```

```
Import statsmodels.api as sm
```

```
From statsmodels.sandbox.regression.predstd import wls_prediction_std
```

```
Import warnings
```

```
Warnings.filterwarnings("ignore")
```

```
raw_data = pd.read_csv('Datasets.csv')
```

```
raw_data.head()
```

```
raw_data.info()
```

```

raw_Data.columns

4678+from sklearn.preprocessing import LabelEncoder

Cat_cols = ['sex', 'smoker' , 'region']

For var in cat_cols:

Number = LabelEncoder()

Raw_Data[var] = number.fit_transform(raw_data[var].astype('str'))

Raw_data.head()

X = raw_data.iloc[:, :-1]

Y = raw_data.iloc[:, -1]


From sklearn.preprocessing import MinMaxScaler

Scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(x)

From sklearn.model_selection import train_test_split

X_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2)

Print('x_train', len(x_train))

Print('y_train:', len(y_train))

Print("x_test:", len(x_test))

Print('y_test:', len(y_test))

From tensorflow.python.keras.models import Sequential

From tensorflow.python. Keras .layer import dense

From tensorflow.python . keras.wrappers.scikit_learn import kerasRegressor

Cnnmodel  = Sequential()

Cnnmodel.add(dense(12, input_dim=6, activation= 'relu'))

Cnnmodel .add(dense(8, activation='relu'))

```

```

Cnnmodel.add(dense(4, activation= 'relu'))

Cnnmodel.add(dense(1, activation= 'linear'))

Cnnmodel.Summary()

Cnnmodel.compile(loss='mse', optimizer='adam', metrics=['mse', 'mae', 'accuracy' ])

X_train  = np.asarray(x_train)

Y_train  = np .asarray(y_train)

History = cnnmodel.fit(x_train,y_train,  epochs      = 300,  batch_size  =  50,
verbose =1 , validation_split=0.2)

Import matplotlib.pyplot as plt

%matplotlib inline

Print(history.history.keys())

#"Loss"

Plt.plot(history.history['loss'])

Plt.plot(history.history['val_loss'])

Plt.title('model loss')

Import matplotlib.pyplot as plt

%matplotlib inline

Print ( history.history.keys())

#"Loss"

Dict_keys(['loss'      , 'mse'      ,      'accuracy', ' val_loss',      'val_mse',      'val_mae',
.'val_accuracy'])

"source": [

"charge_dist = df["charges"].values\n",

"logcharge = np.log(df["charges"])\n",

```

```

"\n",

"\n",

"\n",

"trace0 = go.Histogram(\n",
"    x=charge_dist,\n",
"    histnorm='probability',\n",
"    name=\"Charges Distribution\",\n",
"    marker = dict(\n",
"        color = '#FA5858',\n",
"    )\n",
")\n",

"trace1 = go.Histogram(\n",
"    x=logcharge,\n",
"    histnorm='probability',\n",
"    name=\"Charges Distribution using Log\",\n",
"    marker = dict(\n",
"        color = '#58FA82',\n",
"    )\n",
")\n",

"\n",

"fig = tools.make_subplots(rows=2, cols=1,\n",
"                            subplot_titles=('Charge Distribution','Log Charge Distribution'),\n",
"                            print_grid=False)\n",

"\n",

"\n",

```

```

"\n",

"fig.append_trace(trace0, 1, 1)\n",

"fig.append_trace(trace1, 2, 1)\n",

"\n",

"\n",

"fig['layout'].update(showlegend=True, title='Charge Distribution', bargap=0.05)\n",

"iplot(fig, filename='custom-sized-subplot-with-subplot-titles')"

]

},

"data": [

    {

"hoverinfo": "label+percent",

"labels": [

"Young Adult",

"Senior Adult",

"Elder"

"textinfo": "value",

"type": "pie",

"values": [

574,

548,

Preds = cnnmodel.predict(x_test)

Prediction = pd.DataFrame(preds)

Actual = pd.DataFrame(y_test)

Predictions.head()

```