INCREMENTAL BOUNDED MODEL CHECKING USING MACHINE LEARNING TECHNIQUES

DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF
MASTER OF SCIENCEIN ADVANCED COMPUTER SCIENCE WITH SPECIALIZATION IN ARTIFICIAL INTELLIGENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

YEAR OF SUBMISSION
2021

HUAJIE HE
10689523

SUPERVISED BY
Lucas Cordeiro

DEPARTMENT OF COMPUTER SCIENCE

Contents

Word Count: 17633

List of Tables

List of Figures

Abstract

INCREMENTAL BOUNDED MODEL CHECKING USING MACHINE LEARNING TECHNIQUES
Huajie He
A dissertation submitted to The University of Manchester
for the degree of Master of Science, 2021

This article mainly studies how to use machine learning technology to help ESBMC achieve better results in SV-COMP. ESBMC is an SMT-based verification tool. It can adjust the used SMT solvers, encoding methods and verification methods by selecting different parameters. Different parameter combinations have different effects when verifying C programs. In this article, we convert the C source code into AST, and then extract feature vectors from AST as samples. Similarly, we encode the verification result of ESBMC into the label of the sample. Four machine learning models are used in the experiment, and the data sets come from three sub-category data sets of SV-COMP. After a comprehensive evaluation, it is found that the decision tree model is more suitable because its training time is short and various evaluation benchmarks are better than other models.

In addition to evaluating the ability of a single model, this article also finds that the decision tree has a good performance on the generalization ability of the cross-category set, which can be used as a future research direction. Besides, we combined with the trained model and developed an ESBMC parameter recommender, which performs well in the simulation scenario of the SV-COMP competition. Compared with the ESBMC using the default parameters, it can help users save nearly 45% of the verification time. That is, it has a certain practical significance.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/`) and in The University's policy on presentation of Theses

Acknowledgements

I want to thank my mentor Lucas for providing very detailed guidance and a lot of help for this article. He is a patient and experienced scholar, and I am very happy to write this article under his supervision. In addition, I would also like to thank my parents for their support of my postgraduate studies at the University of Manchester. Finally, I would also like to thank my girlfriend Ruixiang and my cute puppy Lemon, who encouraged me to face and solve challenges.

Chapter 1

Introduction

1.1 Project description

This project is based on ESBMC and machine learning technology, and the data set comes from the benchmarks of SV-COMP over the years. ESBMC is a mature C program verification tool, based on satisfiability theory, can be used to test single-threaded or multi-threaded C/C++ programs[4]. This project mainly uses ESBMC with different parameter combinations to verify the C source files obtained from SV-COMP, and record the effects and time. Furthermore, we combine the verification results with the features extracted from the C source file to form a feature vector and label for machine learning model training.

Feature engineering is an important part of this project, which can be divided into three parts: feature selection, feature extraction and feature construction. The goal of feature selection is to obtain the representative attributes of the C source file. The method is to carry out manual screening based on expert experience. The purpose of feature extraction is to extract the corresponding features from the C source file. At first, we convert the C source file into an Abstract Syntax Tree(AST). After that, each child node is obtained from the AST. At last, the number of different types of child nodes is counted. The purpose of feature construction is to combine feature vectors obtained from C source files and ESBMC parameters into an overall feature vector for machine learning training.

This project uses three sub-category data sets obtained from SV-COMP, namely “Array”, “Floats” and “Loop”. For each sub-category data set, different ESBMC parameters must be used to generate training samples required for machine learning. In order to speed up this process, this experiment uses Python’s multi-process technology. For machine learning, four models of Support Vector Machine(SVM), Decision Tree(DT), K-nearest Neighbors(KNN) and Neutral Network(NN) are used. After testing, the DT’s performance on the three sub-category data sets is significantly better than the other three models. Therefore, the DT is used as the model of the ESBMC automatic parameter predictor.

1.2 Problem description

The Competition on Software Verification (SV-COMP) is an annual comparative evaluation of fully-automatic software verifiers for C and Java programs[5], and ESBMC is a context-bounded model checker used for the verification of C programs[4]. The purpose of this project is to discuss how to help ESBMC achieve good results in SV-COMP.

SV-COMP uses the verification time to evaluate the performance of different software verification tools. Therefore, we need to shorten the ESBMC verification time as much as possible to get a good result.

For each verification task, we can choose 5 types of SMT solvers, 3 types of encoding methods

and 3 types of verification methods, so ESBMC can choose more than 40 parameter combinations. The verification time will also be different when ESBMC uses different parameter combinations. According to our previous experimental results, the best parameter combination can get the result in milliseconds, while the worst parameter combination needed more than 1 hour.

Due to the time limitation in SV-COMP, ESBMC users cannot traverse all parameter combinations, they usually choose parameter combinations based on the experience of experts. However, most of the selected parameter combinations did not have a satisfactory result. Therefore, we need to develop a parameter prediction model based on machine learning to automatically help ESBMC users choose the appropriate parameter combination.

In order to build this model, our experiment will be divided into five parts:

1. Interface

   We have developed an interface to help users manage data sets visually.

2. Feature engineering

   Feature engineering is divided into three parts, they are feature selection, feature extraction, and feature construction.

3. Preparing and preprocessing data

   We obtain the relevant C source files which need to be tested from SV-COMP, and then let each C program run with different ESBMC parameter combinations, and record the verification results and verification time.

4. Training model

   In this experiment, we use four different machine learning models. After comparing the effects, we choose a model with excellent effects as the best model.

5. Automatic ESBMC parameter predictor

   We combine the model to develop an automatic ESBMC parameter predictor, which can predict the new C source file and give the ESBMC parameters with the highest execution efficiency.

1.3 Aim and Objectives

The goal of this project is to obtain a prediction program that can automatically match the optimal ESBMC parameters for different SV-COMP benchmarks. In order to achieve this goal, we have to complete these steps.

1. Build a data set for machine learning

   We extract three sub-category data sets from SV-COMP Each sub-data set has a large amount of C source files to be verified. In order to obtain the label for machine learning, we use ESBMC with different parameters to verify the C source file one by one and record the verification

results and time-consuming. Due to a large number of C source files and too many parameter combinations that need to be matched, we use python's multi-process technology to speed up this process.

2. Interface for managing data sets

   We develop a visual interface to help manage the SV-COMP data set used in this project. Users can intuitively observe the verification results and time-consuming of different C source files with different ESBMC parameters. Moreover, users can expand the data set by adding new C source files.

3. Feature Engineering

   First, we use the python-based pycparser library to convert the C source code into an AST. After that, we count the types and numbers of representative child nodes in the AST. Next, converting them into feature vectors. In the end, combining the different parameter combinations of ESBMC to construct a feature vector for machine learning.

4. Machine learning

   We use four machine learning models for training. By evaluating and comparing the performance of the four machine learning models on three sub-category data sets, we select the optimal model which used in this experiment.

5. Automatic prediction of ESBMC parameters

   For a new C source file to be verified, we use a machine learning model to predict its most suitable ESBMC parameters, then analyze and evaluate the result as a whole. In addition, we compare its actual performance to evaluate the effectiveness of the project.

1.4 Literature review

Model-checking[6] technology is normally used by chip design companies to integrate them into their quality assurance process. The term Model Checking was coined by Clarke and Emerson [7] in the early eighties. The symbolic model checking [8][9] is the first breakthrough. The combination of symbolic model checking and Binary Decision Diagrams(BDD) [9][10], makes it possible to verify a large number of real systems for the first time. Since then, this technology has been greatly developed recent years with the support of the industry[11]. However, due to the amount of memory required to store and manipulate BDD, comprehensive verification of many designs is still beyond the capabilities of BDD-based symbolic model checkers. So in 1994, Biere et al.[12] proposed a new technology called Bounded Model Checking (BMC). Although it does not solve the complexity problem of model checking, experiments have proved that it can solve many problems that cannot be solved based on BDD technology. The BMC problem can be effectively reduced to a proposition satisfiability problem, so it can be solved by the SAT method instead of BDD. However, the user

must provide the number of loop limits k that should be explored. The SAT program will not encounter the space explosion problem based on the BDD method. Modern SAT solvers can handle proposition satisfiability problems with hundreds of thousands or more variables. But with increasing system complexity, SAT solvers are increasingly being replaced by satisfiable modality theory (SMT) solvers. Combine these breakthroughs, Mikhail Ramalho et al. improved the k-induction algorithm and proposed an SMT-based verification tool named ESBMC[13].

1.5 The main contribution of our work

ESBMC, as an SMT-based tool, has been developed by leaps and bounds in recent years. However, the user is still required to participate in order to determine the appropriate parameters when faced with a new verification task. As the scale of the verification task grows, frequent user participation in this process will bring a lot of labour costs. Moreover, the model of relying on expert experience will bring the risk of more errors as the number of problems increases. We develop an ESBMC parameter predictor based on machine learning, which can automatically construct a predictive model based on prior information and complete parameter prediction. The best model has an accuracy rate of 88%. In addition, in experiments that simulate actual verification scenarios, our parameter predictor can save nearly 45% of verification time compared to ESBMC using default parameters. This proves that our predictor can help ESBMC to automatically complete the parameter selection without relying on expert experience. This predictor can better help ESBMC to approach its performance limit as much as possible in each verification task.

1.6 Structure of the Document

The overall structure of this article contains five chapters.

1. Introduction

   This chapter is mainly an overview of the project, including the project overall introduction, the project-specific description, the aiming and objectives, and the literature review related to the project.

2. Background

   This chapter is mainly an introduction to some background knowledge involved in the project, divided into two aspects. The first aspect is bounded model checking, and the second aspect is machine learning related content. Bounded model checking mainly involves the introduction of SV-COMP competition and related knowledge of ESBMC. Machine learning mainly involves an overview and analysis of the principles of four machine learning algorithms used in this experiment.

3. Research Methodology

This chapter mainly discusses in detail the overall design, architecture, selected technical solutions and specific technical implementation details of the experiment. The difficulties encountered in the experiment and the solutions adopted are discussed, and the specific experimental methods of each part are demonstrated in detail.

4. Results Evaluation and Discussion

   This chapter mainly discusses the evaluation plan of the experimental results and further analysis of the experimental results. It introduces in detail the current mainstream evaluation schemes in the industry and their specific applications in this experiment.

5. Conclusion and Future Work

   This chapter mainly summarizes the whole article and gives conclusions. And discussed the deficiencies in this experiment and the work that can be extended in the future.

Chapter 2

Background Knowledge

2.1 Background overview

Understanding of topic/problems The main goal of this project is to use machine learning to help ESBMC achieve better results in SV-COMP. ESBMC is a verification tool full of potential. Achieving better results in SV-COMP means that we can help ESBMC approach its performance limit, which is very important for a verification tool.

So this chapter will introduce experimental data sources, verification tools and machine learning algorithms. These introductions can help us better understand the details of the experiment and the challenges and difficulties that will be encountered.

Awareness of the solutions to the technical challenges

1. This experiment needs to extract C source files from SV-COMP, so we need to understand the composition and details of SV-COMP.

2. The main verification tool in this article is ESBMC. We will introduce the core theories of ESBMC, which are bounded model checking and satisfiability modulus theory. In addition, we also introduced LLBMC and CBMC, which are relatively well-known verification tools in the industry, for comparison with ESBMC.

3. Feature engineering is a necessary pre-part of machine learning. For this reason, we will introduce a structure that describes C source code, Abstract Syntax Tree, which is mainly used in feature engineering.

4. In the machine learning part, we respectively introduce the classification of three current mainstream machine learning methods, supervised learning, unsupervised learning and reinforcement learning. Secondly, we introduce the principles of decision tree, support vector machines, k-nearest neighbors algorithm and neural network respectively. Finally, we introduce solutions to the main problems faced by machine learning, stochastic gradient descent and GridSearchCV.

2.2 SV-COMP

The main goal of the International Software Verification Competition (SV-COMP) is to verify the accuracy and effectiveness of the system. It is currently one of the important standards in the field of software security verification. It contains a benchmark repository that can be used to verify system performance. The repository is suitable for loading ANSI C, Java language, and C language in SMT format. The repository of SV-COMP consists of sets of different questions, each set contains multiple benchmarks. The problem set contains the specification of the problem as well as the format and

different parameters of the program. *category*.prp represents the format, the mode of the program defined by *category*.set, and *category*.cfg describes the different parameters in the C program. Each benchmark includes procedures and parameters. Files with the suffix ".c" are the C programs waiting to be verified, files with the suffix ".i" are the preprocessed C source files, and files with the suffix ".yml" are the configuration files used to record the verification attributes and verification results. The main difference between files with ".c" and ".i" suffixes is that C source files with ".i" suffixes have been preprocessed. The preprocessing includes but is not limited to deleting comments, unnecessary blank lines, the value of various types of macro definitions, and storing the preset values of the macro definitions in relevant variables.[5]

During the verification process, SV-COMP defines terms to represent different sets of questions, including but not limited to "Arrays", "Floats", "Loop", "Mensafety", "Recursive" and "Termination". The main problem sets involved in this article are "Arrays", "Floats" and "Loops".

When the verification process is completed, the benchmark verification results include answers, witnesses, and time. Answer indicates whether the verification task conforms to the specification. True means that no counterexamples can be found under the current parameter configuration, which can be regarded as conforming to the specification. False means that counterexamples can be found under the current parameter configuration, which can be regarded as not conforming to the specification. Unknown means that the current parameter configuration cannot complete the verification task due to time or memory constraints.

SV-COMP has a great influence on software verification and is regarded as a benchmark in this field. Currently, SV-COMP can be used to follow the SMT format generated by ANSI C, Java, and C languages.[5]

In this experiment, we will use three sub-category data sets, "Array", "Loop" and "Floats". Each data set contains multiple benchmarks. We obtain the required C source files by identifying the files with the suffix '.c'.

2.3 Bounded model checking

2.3.1 SAT

Boolean satisfiability (SAT) problem has been continuously improved since the 1990s[14]. This is a well-known decision-making problem, including determining whether a proposition can satisfy a logical formula and assigning appropriate values to the variables of the formula[15]. Therefore, all SAT algorithms require the worst-case exponential time, unless P = NP, which means that none of the currently known algorithms can solve this problem in the worst case. When the problem exists, however, modern SAT algorithms are quite productive in dealing with large search spaces by using the structure of the problem. Many related optimization and decision-making problems have been extended from SAT, and these problems will be called extensions of SAT. At present, one of the most promising expansions of SAT is Satisfiability Modulo Theories (SMT)[16]. The expansion of SAT generally uses the same algorithm technology as SAT or directly uses SAT as the core engine.

A propositional formula is composed of a set of propositional variables, and a single propositional

variable can be expressed as x, y, z. Variables can be assigned a logical value of 0 or 1, or they can be unassigned. If all variables are assigned in 0, 1, the propositional formula called a complete assignment. Otherwise, it is a partial assignment. Most SAT algorithms require propositional formulas to be represented in Conjunctive Normal Form (CNF)[17]. Therefore, SAT mainly solves the satisfiability problem of CNF formulas. A CNF formula can also be regarded as a set of clauses, and each clause can be regarded as a set of literals. Given an assignment, clauses and CNF formulas can be expressed as unsatisfactory, satisfied, or unresolved. When the clause is not satisfied, then all the literals of the clause should be assigned the value 0. When the clause is satisfied, then the literal of at least one clause is assigned the value 1. When the clause is not resolved, the clause is not resolved. Not dissatisfied nor satisfied. A CNF formula is satisfied if all its clauses are satisfied, and when at least one of its clauses is not satisfied, it is not satisfied. Otherwise it cannot be resolved.[15]

SAT theory is the basic part of SMT theory, and SMT theory can be better understood by understanding SAT theory.

2.3.2 SMT

The satisfiability modulus theory (SMT) problem is a decision-making problem about the logic formula of the combination of the theory and the equation expressed in the classical first-order logic. SMT can be considered as a form of constraint satisfaction problem, therefore it is a formal method of constraint programming. Formally speaking, an SMT instance is a formula in first-order logic, in which some functions and predicate symbols have additional explanations, while SMT is a question of determining whether such formulas are satisfactory. Most SMT solvers only support their quantifier-less fragment logic. The SMT solver is a tool for solving SMT problems, it is a constraint solver. A constraint is a statement that specifies the properties of the solution to be found in this context.[18]

SMT works as follows. First of all, for the constraints that must be met, the SMT solver will try to find a solution that satisfies the formula. If there is a solution, then the formula is satisfactory. If the solution does not exist, then the formula is unsatisfiable. The SMT solver is an extension of the SAT solver. It solves the constraints related to (written) propositional logic. In addition, it can resolve constraints involving (written) predicate logic with quantifiers. Simply put, the SMT solver is a SAT solver plus a decision program. Generally, SMT solvers are more powerful than SAT solvers.[16] There are many SMT solvers, such as Z3[19], STP[20], Yices[21], Alt-Ergo[22] and Boolector[23].

The SMT solvers mainly involved in this experiment are Boolector, Z3, Yices, MathSAT[24] and CVC4[25]. They are implemented based on different architectures. Since the experiment does not involve the analysis and tuning of specific SMT solvers, we will not introduce them in detail.

2.3.3 Abstract syntax tree

The Abstract Syntax Tree(AST) is a tree constructed to represent the abstract syntax structure of the source code. Each node of the tree represents a structure of the source code. It has been widely used by various programming languages and software testing tools. [26]

The nodes of AST correspond to different structures and symbols. Compared with source code,

AST is more abstract and can represent the structure and characteristics of source code. This is because it ignores a lot of grammatical details, such as blank lines, punctuation and delimiters. For example, AST ignores the parentheses, they are not explicitly represented in the AST, so they are not a single node. However, the if-condition-then structure is represented as a single tree node with three branches. This is a significant difference between abstract syntax trees and concrete syntax trees. In this article, AST is mainly used to represent the specific structure of the source code, and it has some attributes that can significantly help us extract representative information of the source code. First of all, each node of the AST already contains various attributes, and this information can be further processed or modified during processing. Secondly, AST does not contain unnecessary symbols, and there is no need to pre-process them in the experiment. In addition, since the programming language is a natural language, the language is inherently ambiguous. AST can help us obtain contextual information to avoid such risks. Finally, the AST is a tree structure, which can be easily retrieved and traversed.[27]

Compared with parse trees, the height and the element number of AST is small, which means that its generation speed is very fast, and we don't need to spend a lot of time obtaining and parsing the AST tree.

In this experiment, AST is used in the feature engineering part, and feature vectors are constructed by extracting and counting the number of different types of child nodes in AST. Figure 2.1 is an example code, and figure 2.2 is the AST generated by code 2.1. This example can clearly tell us the correspondence between the code structure and the AST.

```
void fun1( )
{
    int a = 2;
    fun2(a);
}
int fun2(int x)
{
    printf("%d", x);
}
```

Figure 2.1: Example code[1]

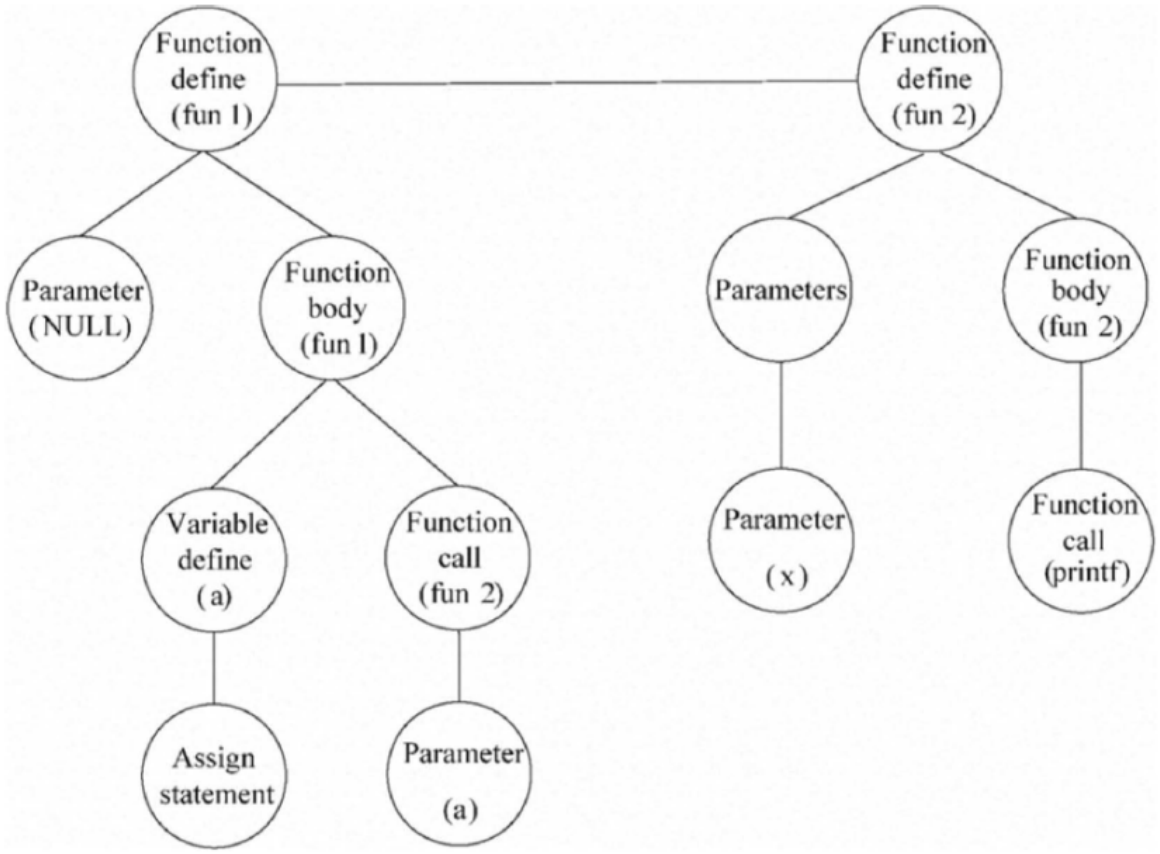

Figure 2.2: The AST of example code[1]

It can be seen from the figure 2.1 and figure 2.2 that a function define contains function parameters and function body. The function body contains specific variable defines and function calls. The variable define contains the assigned statement, and the function call contains the relevant parameters used by it.

2.3.4 BMC

The Bounded Model Checking Problem (BMC) based on Boolean Satisfaction (SAT) has been introduced as a technical supplement to the Binary Decision Diagram (BDD), and the main purpose is to alleviate the state explosion problem[28]. In short, it is a verification method that checks the negation of a given attribute in a certain step.

The basic idea is: firstly, use the satisfiability of the boolean formula to construct a boolean formula. Next, verify within the given boundary K. If there are counterexamples at the depth K and the depth is smaller, we will regard it as satisfiable. In this process, the SAT checker is used to verify whether it can be satisfied. In the BMC of a software system, the limit k limits the number of loop iterations and recursive calls in the program.

With the development of SMT technology, the SMT solver can be used as a backend to solve the

generated verification condition. Unlike SAT, SMT does not use propositional variables to encode various determinable theoretical predicates but saves them in boolean formulas. These problems are handled by special decision-making programs. In the BMC given to SMT, it finds the corresponding non-quantifier formulas from the decidable self of the first-order logic at first, and then use SMT to check their satisfiability.[29]

2.4 Verification tools based on BMC Theory

2.4.1 LLBMC

LLBMC is a program used to find C\C++ program errors and runtime errors. It uses SMT solver for the theory of bitvectors and arrays based on BMC, which makes it possible to achieve single bits of accuracy. It has two notable features. The first is that it can use store and load to support any type of conversion. This feature comes from its bit-precise memory model. The second is that it operates on a compiler intermediate representation (IR). This strategy has three advantages: Firstly, the syntax and semantics of IR are simpler than C\C++ and easier to implement; secondly, IR is closer to the specific execution compared with source code, this is because the compiler has solved most of the ambiguity problems; finally, LLBMC can analyze most programming languages as long as the IR generated by the compiler can be obtained.[2]

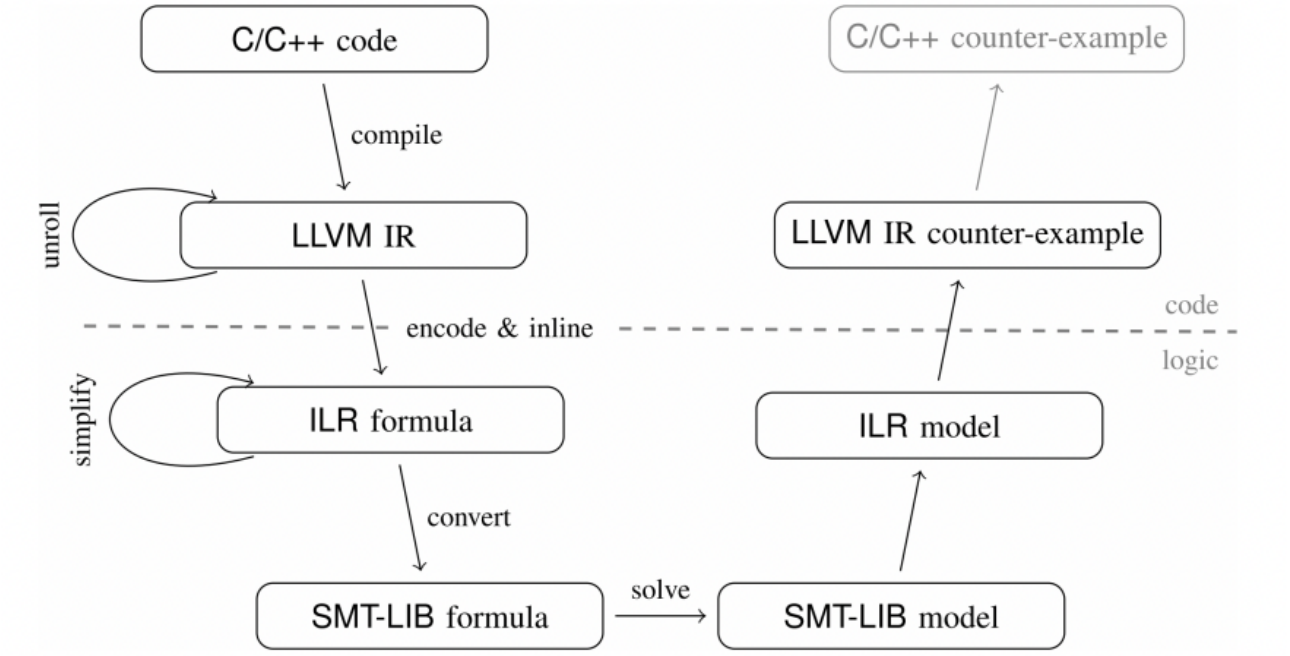

Figure 2.3: Overview of LLBMC's approach[2]

LLBMC runs on the IR[30] of LLVM, and LLVM is an abstract language based on SSA. Figure

2.3 is the approach of LLBMC. LLBMC does not require a lot of interaction with users, which means that it can automate the verification process, which saves users time and improves efficiency. In addition, LLBMC rarely generates false alarms due to its high accuracy.

2.4.2 CBMC

The C Bounded Model Checker (CBMC) currently uses MiniSat 2.2.0. as the backend to verify whether there is no violation of the assertion under the given loop unrolling boundary. CBMC uses GOTO programs as intermediate representation. First, CBMC generates a GOTO program for each C function found in the parse tree. In addition, it adds a new main function, which first calls the initialization function of the global variables, and then calls the original program entry function. At this stage, CBMC also converts the GOTO statement into static single assignment (SSA) form. In addition, CBMC can also supports SMT solver as a backend. The advantage of CBMC is that its performance is stable and it is suitable for most problem categories.[3] Figure 2.4 is the architecture of CBMC.[31]

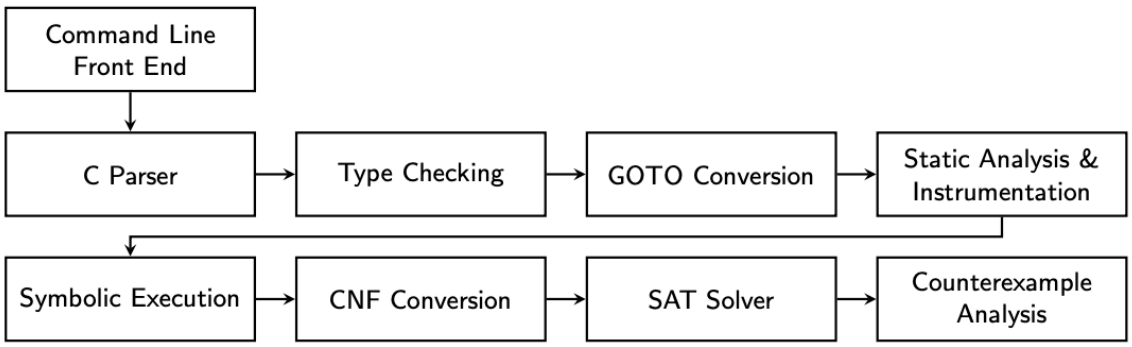

Figure 2.4: CBMC architecture[3]

2.4.3 ESBMC

ESBMC is a context-bounded model checker that can automatically verify the predefined security properties and user-defined program assertions in C programs. For each verification task, ESBMC can select different SMT solvers, encoding methods and verification methods. It will has three different results: verification success, verification failure and time out. When we use different parameter combinations to check the C program, the time taken for each parameter combination is different.[4]

When we use ESBMC to verify a C program, there are three parameters that can be configured. They are SMT solvers, encoding methods and verification methods. ESBMC provides five different SMT solvers: –z3, –boolector, –yices, –mathsat and –cvc. There are three encoding methods: –ir, –fixedbv and –floatbv. Verification methods have three types: –k-induction, –falsification and –incremental-bmc.

A standard ESBMC command line is as follows:

esbmc example.c –z3 –fixedbv –incremental-bmc –max-k-step 10 .

In the command line, example.c is the name of the C program that needs to be verified, –z3 represents the type of SMT solver to be called, –fixedbv uses the encoding method, and –incremental-bmc represents the verification method.

ESBMC currently uses clang[32] as the front-end component, and uses the Control-flow Graph (CFG) Generator to convert the AST into an equivalent GOTO program. Then ESBMC symbolically executes the GOTO program. The program can unroll the loop k times, generate the corresponding static single assignment (SSA) form, and derive all the security attributes to be checked by the SMT solver. Finally, ESBMC uses five solvers as backends, namely Boolector (default), Z3, MathSAT, CVC4 and Yices. In addition, ESBMC currently implements a Python API to help developers gain access to the inside of the verification process.[4] The whole workflow is shown as figure 2.5.

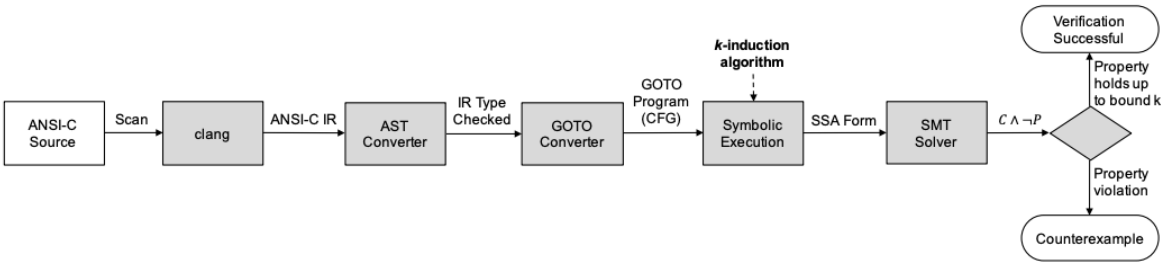

Figure 2.5: ESBMC workflow[4]

2.5 Machine learning

2.5.1 Machine learning overview

Machine learning is an important research part in the field of artificial intelligence, involving many disciplines such as Probability Theory, Statistics, Approximation Theory, Convex Analysis and Algorithm Complexity Theory. Machine learning is essentially an algorithm that obtains rules by analyzing data and uses the rules to predict unknown data. It can complete learning, reasoning, and decision-making without manual interaction. The machine learning algorithm will build a mathematical model based on sample data, and use the model to predict unknown data.[33]

The field of machine learning has significant changes in the past ten years. It has gone from a purely academic and research field to a wide range of applications in various fields, such as recommendation systems. Nowadays, companies are collecting user data, such as like types, clicks on links, responses. After that, they use machine learning to find what may interest their users.[33]

Machine learning is generally divided into supervised learning, unsupervised learning and reinforcement learning. The main method used in this experiment is supervised learning.

2.5.1.1 Supervised learning

The main idea of supervised learning is to predict unknown samples based on the knowledge of existing experience. More specifically, it uses labelled data sets to train models to classify results.

According to the different types of classification categories, supervised learning tasks are roughly divided into classification and regression.[34]

Classification is the most common supervised learning problem. Its most basic prototype is a binary classification problem, that is, the total number of classifications is two, and the model classifies the prediction results into two categories. When the number of classifications exceeds two, the problem becomes a multi-classification problem. Commonly used models include naive bayes and decision trees. The difference between classification and regression is the type of output variable. Qualitative output is called classification, or discrete variable prediction; quantitative output is called regression, or continuous variable prediction. Regression is often used for continuity data such as housing prices and rainfall. Commonly used models include support vector machines and k-nearest neighbours.

The data used in supervised learning can be divided into three categories: training set, validation set and test set. The data can be text, image or audio. At first, we need to extract the required features to form a feature vector. Next, use these feature vectors together with the corresponding labels as the input of the machine learning algorithm to train a predictive model. The same feature extraction method is applied to the train data set and test set data. Finally, the prediction model is trained on the training data set and tested on the test data set. The main purpose of the training set is to train a model to recognize potential patterns, then it can identify unlabeled samples in the test data set with the highest possible accuracy. The validation set is to ensure that the model is not overfitting. The testing process is used to evaluate and tune the effect of the model.[34]

However, supervised learning requires the use of labelled data for training, which places requirements on data acquisition and labelling. If the data is insufficient, it will lead to over-fitting problems[35].

2.5.1.2 Unsupervised learning

Unsupervised learning is a training method in machine learning. Compared with supervised learning, the criterion is whether there is supervision (supervised). If the input data has no labels, it is unsupervised learning. It is essentially a statistical method, whose main goal is to discover some potential structures in unlabeled data[36]. There are two main methods of unsupervised learning. The first is clustering. Clustering is an automatic classification method. After clustering, there is no obvious correspondence between the data and the label. The second is data dimensionality reduction. Dimensionality reduction is to reduce the complexity of the data while preserving the relevant structure as much as possible.[36] The main algorithm is PCA[37].

There are three main characteristics of unsupervised learning:

1. Uncertain training goal

   In unsupervised learning, the results cannot be predicted and therefore the training results obtained often deviate from cognition. This means that there is very large uncertainty in the result set.

2. No label is required for training data

Unsupervised learning uses unlabeled categories data and will automatically summarize the possible hidden patterns in the input data. After that, it will classify and label the original data. In actual engineering, unsupervised learning often has advantages. This is because the cost of manually labelling the input data may be very large. In addition, unsupervised learning does not need to understand the input data, that is, there is no need to do too much preprocessing on the original data, which greatly reduces the engineering cost.[38]

3. Training effect is harder to evaluate

   Firstly, compared with supervised learning, the results produced by unsupervised learning methods are not very accurate and reliable. This is because that the machine must look for hidden patterns in the original data, which may not be understood by humans. Secondly, because the category in unsupervised learning is unknown, it means that there is no prior knowledge to help the algorithm make corrections, which means that sometimes it is impossible to determine the results of the analysis.[36]

Unsupervised learning has a significant advantage, which can be used for real-time learning. Supervised learning basically requires offline learning. This gives people involved in machine learning the opportunity to interact with the machine when analyzing discrete data.[38]

2.5.1.3 Reinforcement learning

The goal of reinforcement learning is to make decisions by training machine learning models. In reinforcement learning, computers need to repeatedly experiment with problems to come up with different solutions. This process is similar to a game. Experimenters can interact with the computer through rewards or punishments. In this process, the computer will always try to maximize the total reward, which means that it will continuously optimize itself to meet the requirements of the experimenter. Similarly, it will learn from your own mistakes. The original model of this method comes from the human learning process.[39]

In the process of reinforcement learning, the computer will not get any hints and suggestions from the experimenters. The computer needs to perform a lot of calculations and simulations to optimize its own strategy to obtain more rewards. The computer will start from a completely randomized experiment and will continue to improve and expand its own strategy and methods until it obtains a result that satisfies the experimenter. Reinforcement learning is one of the most powerful algorithms at present. It can achieve better results when supported by equipment with sufficient computing power.[39]

The main difference between reinforcement learning and supervised learning is that only partial feedback about predictions needs to be provided to the computer. At the same time, experimenters can produce positive results by continuously influencing the future state of experimental results. Reinforcement learning technology has broad prospects in-game AI and autonomous driving.

Reinforcement learning also faces some challenges. The first is how to simulate a complex experimental environment. For game AI, the game scene is very simple and fixed; but for automatic driving, due to changes in the road environment, the experimental environment has a high degree of

complexity. Secondly, because the experimenter's main interaction with the computer is only rewards and punishments, this means that it cannot interfere with the training process. The computer has to store and memorize a large amount of information, so when storing new information, it will delete useful old information. Finally, like most machine learning algorithms, reinforcement learning will fall into a local optimal problem, which means that the computer will be terminated early before finding the optimal result.[40]

2.5.2 Decision tree

Decision tree is the simplest machine learning algorithm. It is easy to implement, highly interpretable, fully in line with human intuitive thinking, and has a wide range of applications. Decision tree is a tree structure. The internal nodes represent features and the leaf nodes represent labels[41]. A decision tree can be constructed based on these sample data. When new data are to be predicted, a certain feature value is used for judgment at the internal node of the tree, and the branch node to be entered is determined according to the judgment result. This process keeps going down until the leaf node is reached and the classification result is obtained.[42]

The three steps of decision tree algorithm:

1. Feature selection

   Feature selection determines which features are used to make judgments. In the training data set, there may be many features for each sample, and different features mean different effects. Therefore, the function of feature selection is to filter out the features that are more relevant to the classification results, that is, the features with strong classification ability.

2. Decision tree generation

   Decision tree generation. After the feature is selected, it is triggered from the root node. The information gain of all features is calculated for the node and the feature with the largest information gain is selected as the node feature, and then child nodes are established according to the different values of the feature. Each child node is generated in the same way until the information gain is small or there are no features to choose from.

3. Decision tree pruning

   The main purpose of pruning is to prevent over-fitting and reduce the risk of over-fitting by actively removing some branches.

2.5.2.1 ID3 algorithm

The core of the ID3 algorithm is to apply information gain to select features on each node of the decision tree, and build the decision tree recursively. Information gain indicates the degree to which the uncertainty of the class information is reduced by knowing the information of the feature. In information theory and probability statistics, entropy is a measure of the uncertainty of random variables. The greater the uncertainty, the greater the entropy, the lower the uncertainty, and the smaller

the entropy.[43] If X is a discrete random variable with a finite number of values, its probability distribution is

$$P(X = x_i) = p_i, i = 1, 2, \cdots, n \tag{2.1}$$

Then the entropy of the random variable X is defined as

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i \tag{2.2}$$

Where n represents n different discrete values of X.
For random variables (X, Y), the joint probability distribution is

$$P(X = x_i, Y = y_j) = p_{ij}, i = 1, 2, \cdots, n; j = 1, 2, \cdots, m \tag{2.3}$$

The conditional entropy of random variable Y is defined as

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i) \tag{2.4}$$

The difference between entropy H(Y) and conditional entropy H(Y|X) is called information gain.

$$Gain(Y, X) = H(Y) - H(Y|X) \tag{2.5}$$

The main steps of the ID3 algorithm:

1. Start from the root node, calculate the information gain of all possible features, and select the feature A with the largest information gain as the partitioning feature of the node
2. Select different values of feature A to create child nodes
3. Recursively call the above method on the child nodes to build a decision tree until the information gain of all features is small or there are no features to choose from

Although ID3 algorithm puts forward new ideas, there are still many areas worthy of improvement. For example, ID3 does not consider continuous features. If the length and density are continuous values, they cannot be used in ID3, which greatly limits the use of ID3. In addition, the information gain is used as the criterion for selecting the optimal partition feature, but the information gain will be biased towards those features with more values[44].

2.5.2.2 C4.5 algorithm

The C4.5 algorithm is an improvement of the ID3 algorithm. Different from ID3, C4.5 uses information gain ratio[45] as the feature selection criterion instead of information gain. The information gain Gain(D, A) of feature A to training data set D is defined as the difference between the empirical entropy H(D) of data set D and the empirical conditional entropy H(D|A) of D under the given condition of feature A:

$$GainRatio(D,A) = \frac{Gain(D,A)}{H_A(D)} \tag{2.6}$$

2.5.2.3 Classification and Regression Tree

Classification and Regression Tree (CART) algorithm uses the Gini index[46] to select features. The Gini index represents the impurity of the model. The smaller the Gini index, the lower the impurity and the better the features. For a given sample D, assuming there are K categories and the number of the k-th category is Ck, then the Gini index of sample D is:

$$Gini(D) = 1 - \sum_{i=1}^{n} p(x_i)^2 \tag{2.7}$$

If D is divided into two parts D1 and D2 according to a certain value a of feature A, then under the condition of feature A, the Gini index of D is:

$$GiniIndex(D|A = a) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2) \tag{2.8}$$

2.5.3 Support vector machine

Support vector machine (SVM) is a supervised machine learning model, which is very fast and reliable when the amount of data analyzed is limited. SVM is often used for binary classification problems. After providing a set of labelled training data for each category, the SVM model can effectively classify the training data. Compared with newer algorithms such as neural networks, SVM has a very fast training speed when the number of samples is small (thousands), and it tends to be effective. It makes this type of algorithm often exist as a baseline, which can be used to compare the effects of other algorithms. Furthermore, SVM can be used for multi-classification problems, but it often requires a longer training time.[47]

SVM is often used for linear classification, but kernel techniques can be used to make it a substantial non-linear classifier. The learning strategy of SVM is to maximize the interval, which can be formalized as a problem of solving convex quadratic programming, and it is also equivalent to the problem of minimizing the regularized hinge loss function. The learning algorithm of SVM is the optimal algorithm for solving convex quadratic programming.

2.5.3.1 Linear SVM

Firstly, we need to introduce the linear classifier. We will start with the classic binary classification problem. When given some data points, they belong to two different classes. Now we need to find a linear classifier to divide these data into two classes. If x is used to represent data points and y is used to represent categories (y can be 1 or -1, representing two different classes), the learning goal of a linear classifier is to find a hyperplane in an n-dimensional data space. The equation of this

hyperplane can be expressed as formula 2.9.

$$w^T x + b = 0 \tag{2.9}$$

We can find a lot of hyperplanes that can classify data by adjusting the values of w and b, then we need to determine from these hyperplanes a hyperplane that is most suitable for separating the two types of data. The criterion for judging the "best fit" is that the line has the largest interval between the data on both sides of the line. Therefore, it is necessary to find the hyperplane with the largest separation.

When the hyperplane $w^T x + b = 0$ is determined, we can obtain the distance between the point x and the hyperplane by calculating $\left|w^T x + b\right|$. When the point classification is correct, the sign of $w^T x + b$ should be consistent with the sign of the mark y, where $\left(y\left(w^T x + b\right)\right)$ can be used to indicate the correctness of the classification. Among them, we call formula 2.10 as functional margin.

$$\gamma = \left(y\left(w^T x + b\right)\right) \tag{2.10}$$

The minimum value of the function margin of the hyperplane (w, b) with respect to all sample points (xi, yi) in T (where x is the feature, y is the result label, and i represents the i-th sample), it is the hyperplane (w , b) about the function margin of the training data set T. As formula 2.11.

$$\gamma = min\gamma_i \, (i = 1 \cdots n) \tag{2.11}$$

However, when changing w and b proportionally, the value of the function margin will change when the hyperplane is unchanged. Therefore, geometrical margin is needed to define the distance from the point to the hyperplane. Regarding the geometric margin, we denote the point where the point x is projected vertically onto the hyperplane as x0, and the distance between x and x0 is the geometric margin of the point x. According to the knowledge of geometric plane, we get the formula 2.12.

$$x = x_0 + \gamma \frac{w}{\|w\|} \tag{2.12}$$

Where $\|w\|$ is the L^2-Norm of w and $\frac{w}{\|w\|}$ is the unit vector. Continuing to derive the formula, we can get 2.13:

$$\gamma = \frac{w^T x + b}{\|w\|} \tag{2.13}$$

From the formula 2.13, we can conclude that the geometric margin is the function margin divided by $\|w\|$.

Maximum Margin Classifier The greater the "margin" of the hyperplane from the data point, the greater the confidence of the classification. Therefore, in order to make the classification confidence as high as possible, the selected hyperplane needs to be able to maximize this "margin" value. From the previous analysis, we need to use geometric margins to measure the maximum margin.

In summary, we need to traverse all the points to find the sum of their geometrical margins to the hyperplane, and then find the hyperplane with the largest geometrical margin, which is the most suitable hyperplane.

Support Vector For each data point x, we can find the corresponding vertical projection point x0 on the hyperplane. A vector can be obtained from point x0 to point x. For all vectors, the points with the smallest absolute value on both sides of the hyperplane are the support vectors.

Hard margin The establishment of the hard-margin SVM model is based on the data being "linearly separabl". The so-called linearly separable means that there is no noise and the training data can be perfectly divided into two categories. The hard-margin SVM model is constructed based on this assumption, that is, assuming that the correctly classified sample points are on both sides of their respective support vectors.

Soft margin In the hard margin, it has been assumed that the training data is strictly linearly separable, that is, there is a hyperplane that can completely separate the two types of data. But the assumption of realistic tasks is often not exist. Therefore, a soft margin SVM is proposed, which allows SVM to make mistakes on a small number of samples, relaxes the previous hard margin maximization condition a bit, and allows a small number of samples to not meet the constraints.

2.5.3.2 Non-linear SVM

When the data cannot be linearly separable, we need to use some kernel techniques to implement nonlinear SVM[48].

Kernel techniques Essentially, the kernel function is an inner product function, which defines an inner product space. Therefore, the kernel function, like the ordinary inner product, can be regarded as a function to measure the similarity of two vectors.[49]

For non-linear situations, the processing method of SVM is to select a kernel function k, and solve the problem of linear inseparability in the original space by mapping the data to a high-dimensional space.

Specifically, in the case of linear inseparability, the support vector machine first completes the calculation in the low-dimensional space, then maps the input space to the high-dimensional feature space through the kernel function, and finally constructs the optimal separation hyperplane in the high-dimensional feature space. This can separate non-linear data that is not easily distinguishable on the plane.[49]

The kernel function has polynomial kernel(2.14), gaussian radial basis function kernel(2.15) and sigmoid kernel(2.16). Generally speaking, the gsaussian radial basis function kernel has a good effect on most nonlinear data. The main features of using kernel functions are:

1. Long training time

2. High accuracy

This is because that the ability to model very complex nonlinear decision boundaries is obtained through the kernel function SVM and the prediction ability can be improved. However, the calculation complexity is also increased, and then the time is increased. In addition, it also has a feature, that is, compared with traditional naive Bayes and decision tree algorithms, SVM can complete numerical predictions not just category predictions. This is very useful in many scenarios.

$$K\left(x_i, x_j\right) = \left(\gamma x_i^T x_j + b\right)^d \tag{2.14}$$

$$K\left(x_i, x_j\right) = exp\left(-\gamma \left\| x_i - x_j \right\|^2\right) \tag{2.15}$$

$$K\left(x_i, x_j\right) = tanh\left(\gamma x_i^T x_j + b\right) \tag{2.16}$$

2.5.4 KNN

The KNN algorithm is a classifier. It compares the features of the unknown data with the corresponding features in the training set and finds the top K data which are most similar, then the label of the unknown data is the most frequent label among these K data.[50]

The description of the algorithm is:

1. Calculate the distance between the unknown data and each training data
2. Sort ascending the distances
3. Select the K points with the smallest distance
4. Count the category frequency of these K points
5. Return the category with the highest frequency among the first K points as the predicted classification of the unknown data

The shortcomings of the traditional KNN method mainly include[51]:

1. Slow classification

   KNN is a lazy learning method. It stores all the training samples at first, and then temporarily performs calculation processing when classification is required. For high-dimensional samples or large sample sets, the time and space complexity are relatively high, and the time cost is O(mn), where m is the spatial feature dimension of the vector space model, and n is the size of the training sample set.

2. Strong dependence on sample library capacity

   The recognition error of KNN is large when there are too many categories. This is because it needs enough training samples.

3. Determination of K value

   The KNN algorithm must specify the K value. If the K value is incorrectly selected, the classification accuracy cannot be guaranteed. Choosing a smaller K value is equivalent to using training examples in a smaller field to make predictions. The "learning" approximation error will be reduced. In other words, the decrease of the K value means that the overall model becomes complicated and it is prone to overfitting. Choosing a larger K value is equivalent to predicting with training examples in a larger field. Its advantage is that it can reduce the estimation error of learning, but the disadvantage is that the approximate error of learning will increase. The increase of the K value means that the overall model becomes simple. In addition, it is completely unacceptable when k is equal to the number of samples.[50]

In terms of the amount of calculation, KNN is much slower than the supervised learning method. Because it requires a lot of calculations when performing classification. In response to this problem, most of the solutions so far are based on two considerations: reducing the sample size and speeding up the search for K nearest neighbours.

In order to change the defection of the same feature in the traditional KNN algorithm, the feature can be given different weights in the distance formula of similarity. For example, in the Euclidean distance formula, different weights are assigned to different features. The weight of the feature is generally set according to the role of each feature in the classification. The weight can be determined according to the role of the feature in the entire training sample library, and the weight can also be determined according to the classification role in the partial sample of the training sample.[52]

The training sample library is maintained to meet the needs of the KNN algorithm, including adding or deleting samples in the training sample library. The maintenance of the sample library is to use appropriate methods to ensure the size of the sample space. For example, samples that meet a certain condition can be added to the database. At the same time, the database can also delete samples when conditions are satisfied. Therefore, it can ensure that the samples in the training sample library provide the relatively uniform feature space required by the KNN algorithm.

In practical applications, the cross-validation method is used to select the optimal K value. Splitting the training data of the sample into different training data set and validation data set, then using training data set and validation data set to test the accuracy of the model respectively. Next, calculating the average value of the accuracy, which is the result of this cross-validation. Finally, applying cross-validation to different hyperparameters, and selecting the hyperparameter with the highest accuracy as the hyperparameter for model creation.[53]

2.5.5 Neural network

A neural network is a computational model with layers of connected nodes, and its hierarchical structure is similar to the structure of a neuron network in the brain. Due to neural networks can learn from data, they can be trained to recognize patterns, classify data and predict future events. [54]

The specific method of the neural network is to subdivide the input into multiple abstract layers, and the recognition pattern can be trained through a large amount of data. The behavior of the neural

network is determined by the connection mode of the nodes of each abstraction layer and the weights of these nodes. During training, the system will automatically adjust the relevant weights according to the specified learning rules until the neural network normally performs the required tasks.

Inspired by the biological nervous system, neural networks combine multiple processing layers through the parallel use of simple element operations. It consists of an input layer, one or more hidden layers and an output layer. The layers are connected to each other by nodes or neurons, and each layer uses the output of the previous layer as its input. Neural network application scenarios mainly include supervised learning and unsupervised learning, classification, regression, pattern recognition and clustering.[55]

2.5.5.1 Feed-forward neural network

A feedforward neural network [56] is a simple example of supervised learning. Neural networks have three kinds of nodes: input nodes, hidden nodes and output nodes. It has two obvious characteristics: the first is fully connected, which means that the neuron nodes of each layer use the output of the previous layer as input. In addition, the nodes of each adjacent layer are connected to each other. The second is that there is no cyclic connection between nodes, which means that there is no possibility of interconnection between nodes on the same layer. Moreover, the feedforward neural network can contain multiple hidden layers.[57] The neural network of this structure is mainly used in this article.

2.5.5.2 Features of neural network

1. High degree of parallelism

   The basic unit of the artificial neural network is the neuron. Although the function of the neuron is single and simple, when many of the same simple neurons are connected in parallel to form a neural network, the parallel processing capability of the neural network is amazing. Neural networks imitate the structure of the human brain. They are not only parallel in structure, but also parallel and synchronized in operation and processing. The processing units of each layer are operated at the same time. Due to the advantages of this structure, the computing unit of the neural network can be distributed. The working speed of neural networks with parallel processing capabilities is significantly higher than that of von Neumann computers with string structure.

2. Non-linear global action

   In a neural network, each neuron will receive input from other neurons. Combined with the parallel network structure, the neurons can control and influence each other. This process is essentially a non-linear mapping from the input state to the output state. From a global perspective, the overall performance of a neural network is a collective behavior rather than a superposition of the performance of individual sub-networks. Non-linear relationships are the reason why the human brain produces thoughts, and neural networks imitate this structure. Each neuron generally has two states of activation and inhibition. Mathematically, it is a nonlinear

neural network. At the same time, neurons can store thresholds, and the network composed of such neurons has better fault tolerance and storage capacity.

3. Associative memory function and good fault tolerance

   The artificial neural network can store the processed data in the weights between neurons, and combine its network structure to make it have associative memory. Additionally, due to this information is distributed among different neurons in a distributed storage form, which makes the neural network more fault-tolerant. Therefore, it is often used for feature extraction, cluster analysis, etc. pattern association, classification and recognition Work.

   Artificial neural networks can often learn from imperfect data or pictures. Since the weight data after training is stored in a distributed manner, when a certain proportion of neurons are not involved in the calculation, it will not have a particularly large impact on the function of the system. This feature makes it have good generalization and fault tolerance in the face of noise and missing data.

   Since the overall behavior of the neural network depends on the mutual influence and restriction of most of the neurons, it can simulate the human associative memory behavior.

4. Adaptive, self-learning function

   Artificial neural networks have a structure similar to the human brain, as well as self-learning and self-adaptive behaviors similar to humans. Self-learning means that as the amount of data learned by the neural network increases, it can continuously modify the weights between neurons to achieve the optimal strategy. Self-adaptation means that the neural network can find the hidden association pattern between the input data and the output data, and then start learning. This kind of learning does not require prior knowledge and pre-settings.

5. Distributed storage of knowledge

   The artificial neural network stores knowledge in a distributed manner. Its system does not have a specific storage structure, but stores information between neurons. Combined with its special structure, neurons can split and reassemble information. When a neural network needs to extract a set of information, it often requires multiple neurons to work together, and combined with the activation signals distributed on different neurons, the neural network can quickly identify similar patterns.

6. Non-convexity

   The neural network depends on the activation function. When the activation function is at the extreme value, the system will be relatively stable. Non-convexity refers to the existence of multiple extreme values of a certain function, which can make the system have multiple steady states and can make the system have good diversity.

2.5.6 Gradient Descent

Many machine learning algorithms need to solve optimization problems at the end. Among various optimization algorithms, the gradient descent method[58] is the simplest and most commonly used.

The essence of the optimization problem is to solve the extreme value of the function, which may be a maximum value or a minimum value. Combined with calculus, when the derivative of the function is 0, the extreme point can be found. Therefore, the essential method of gradient descent is to find the extreme value in combination with calculus.

This process is very similar to the process of descending a mountain. The slope of the slope is the derivative. In order to find the position of the bottom of the slope faster, we will choose the steepest position to move forward each time, and then move forward a certain distance each time. After repeated periods of time, enable reach to slopes. However, since there may be multiple extreme points, the lowest point we currently find is not necessarily the optimal solution, so there will be a local optimal problem.

The basic concept of gradient descent:

1. Learning rate: Learning rate is the length of progress in a certain direction in gradient descent.

2. Feature: Feature is the input part of the data sample.

3. Hypothesis function: In supervised learning, in order to fit the input sample, the hypothesis function is used.

4. Loss function: In order to evaluate the fit of the model, the loss function is usually used to measure the degree of fit. Minimization of the loss function means that the degree of fit is the best, and the corresponding model parameters are the optimal parameters. In linear regression, the loss function is usually the square of the difference between the sample output and the hypothesis function.

The main methods of gradient descent are algebraic method and matrix method. The specific tuning methods are:

1. Adjust the learning rate. In a specific algorithm implementation, the learning rate can take different values. The specific adjustment method depends on the loss function. When increasing the learning rate loss function becomes larger, we need to reduce the learning rate, and vice versa. In addition, when the step size is too large, it will cause the iteration to be too fast, and it may miss the optimal solution. On the contrary, if the step size is set too small, it will cause the iteration speed to be very slow and the training time will be long.

2. Initialization of parameters. The selection of the initial value of the parameter greatly affects the final result. Because gradient descent may only select local optimal values. Due to the risk of a local optimal solution, it is necessary to run the algorithm with different initial values multiple times and choose the initial value that minimizes the loss function.

3. Normalization. Because the value range of different features of the sample is different, it may lead to slow iteration. In order to reduce the impact of feature value, the feature data can be normalized, that is, for each feature x, its expectation and standard deviation can be calculated. Subtracting the expectation from x and put the standard deviation at the end. In this way, the new expectation of the feature is 0 and the new variance is 1, and the iteration speed can be greatly accelerated.

With the deepening of the research on gradient descent methods, some gradient descent algorithms have been proposed. The three main gradient descent methods are introduced below.

Batch Gradient Descent The batch gradient descent method is the most commonly used form of the gradient descent method. The specific method is to use all the samples to update the parameters.

Stochastic Gradient Descent The stochastic gradient[59] descent method is actually similar to the batch gradient descent method. The difference is that the data of all m samples is not used when calculating the gradient, but only one sample i is selected to find the gradient. It is at two extremes to the batch gradient descent method. One uses all data for gradient descent, and the other uses one sample for gradient descent. Naturally, their respective advantages and disadvantages are very prominent. Regarding the training speed, the stochastic gradient descent method only uses one sample to iterate at a time, and the training speed is very fast, while the batch gradient descent method cannot satisfy the training speed when the sample size is large. For accuracy, the stochastic gradient descent method is used to determine the direction of the gradient with only one sample, resulting in a solution that may not be optimal. For the convergence speed, because the stochastic gradient descent method iterates one sample at a time, the iteration direction changes greatly, and it cannot quickly converge to the local optimal solution.

Mini-batch Gradient Descent The biggest difference with batch gradient descent is that when updating parameters, all training samples are not taken into account, and then the sum is divided by the total. This method combines batch gradient descent and stochastic gradient descent, and can obtain more accurate results at a faster training speed.[60]

2.5.7 Resampling methods

Because the available sample capacity is limited, different random or non-random sample sets can be obtained through repeated sampling. Using these different sample sets to train the model can find the relatively best-performing model in all available training sample sets. Because general machine training requires a training set to train the best model, and then the test set is used to finally test the model (the best model in the training set does not mean that the test set has the best experimental results). At this time, we need to split the sample (generally 75-80% is the training set, and 25-20% is the test set). The following introduces two concepts that are only available in cross-validation: the true training set in the training set, which is the data set used to train the model. By continuously adjusting the parameters, the model fits best and the deviation is minimized. The rest of the training

set is used as the internal validation set. The test set is finally used to observe the predictive ability of the trained model. However, because the number of data sets is small in some cases, and the test set can only be used once. Therefore, the training set is divided into a true training set (for modeling) and a verification set (for verifying the model), so that the ability of the model can be tested during model training.[61]

The generalized data set grouping scheme is 3:1 (the training set is 3 times or more of the test set). Then the filtered training set is further divided into the true training set and the validation set for finding the best parameters for modeling and finding the best model.

2.5.7.1 Cross validation

The cross-validation family is divided into k-fold cross-validation (K-fold CV), leave-one-out cross-validation (LOOCV), repeated cross-validation (Repeated CV), and Monte Carlo (LOGCV). K-fold CV is mainly used in this article. K-fold CV divides the training set into k-parts, each time one is selected as the validation set, and the remaining k-1 is the true training set. K different training results are produced (that is, there are k different model parameters); leave-one-out cross-validation is a special case of k-fold validation, that is, the cross-validation method when k is exactly equal to the number of samples. Because the number of groups is very large, the operation efficiency is poor; repeated cross-validation is to repeat the k-fold cross-validation n times. Since it is performed n times more, the running efficiency is slower, but due to more groupings, the fitting ability and predictive ability of this model are marginally improved; Monte Carlo sampling is the standard for dividing the training set and the validation set each time. In the past, K-fold CV used a fixed data set as the verification set, but the number of groups set aside by Monte Carlo each time as the verification set is different. For example, a total of ten samples are taken and divided into five groups. The first time is 3:2, then the second set is not reserved, which means all are used for training, and the third time may be 4:1.[53]

2.6 Summary

In this chapter, we mainly include a brief introduction and development of some technologies involved in this article. It mainly contains four parts, namely SV-COMP, Bounded model checking, Verification tools based on BMC theory and Machine learning. Since the experiment in this article is based on SV-COMP, an international competition, we introduce the architecture and content components of SV-COMP at first. After that, we introduced the theory of Bounded model checking and three tools developed based on the theory, LLBMC, CBMC and ESBMC. ESBMC is the software verification tool used in this article. Finally, we introduced machine learning related technologies and four algorithms, namely K-nearest neighbours, Support vector machines, Decision Tree and Neural network. We focus on the basic principles, advantages and disadvantages of these four algorithms, and some techniques that can be used for optimization.

Chapter 3

Research Methodology

3.1 System overview

In order to discuss the experimental details in a better way, this article will discuss five aspects. In this project, they are the interface, preparing and preprocessing data set, feature engineering, machine learning models, and programs that can automatically predict the optimal parameter combinations. The pipeline of the system is as figure 3.1 shown.

The first part is the interface. The interface uses Java Web, React and Mysql technologies. The main target is to provide a visual interface to help users track benchmark training. Users can add a new benchmark as an expansion of the data set at any time. In addition, real-time prediction of the new benchmark provided by the user can be completed to determine the optimal parameter combination that matches the benchmark.

The second part is the preparation and preprocessing of the data set. We use the machine learning model with supervised learning in this experiment, as a result, we need to obtain data with labels. The main points of the experiment are as follows:

1. For the data in SV-comp, we extract the C source file with the suffix ".c" and use the different ESBMC parameter combinations in 239 to generate the benchmark for this experiment.

2. Divide the training set and test set. In order to get closer to the real situation, we use the SV-COMP year 2016-2020 data as the training data set, and SV-COMP year 2021 data as the test set.

3. For each C source file, we can verify with 239 ESBMC parameters. That is, each C source file can generate 239 corresponding benchmarks for this experiment.

4. In order to obtain the label, we used a multi-process-based python program, which uses ESBMC with different parameters to verify the C source file. We classify and record the results returned by the verification.

The third part is feature engineering. The experimental process is as follows:

1. We use pycparser to generate an AST from the C source file, after that, extracting features from the AST and convert them into feature vectors.

2. Generate parameter vectors corresponding to 239 different ESBMC parameters.

3. Combine the feature vector and parametric vector as the feature vector of the benchmark.

The fourth part is the machine learning model. This experiment uses four machine learning models, namely SVM, Decision Tree, KNN and Neural network. And use GridSearchCV technology to

automatically adjust the model parameters. In the end, we use the accuracy of cross-validation as the evaluation standard.

The fifth part is the program of automatically predicting the optimal parameter combination. The main idea of this part is to generate the corresponding 239 benchmarks for the C source file to be predicted, and then use the trained model to predict the verification results of these 239 benchmarks. At last, taking the best result as the recommended parameter output.

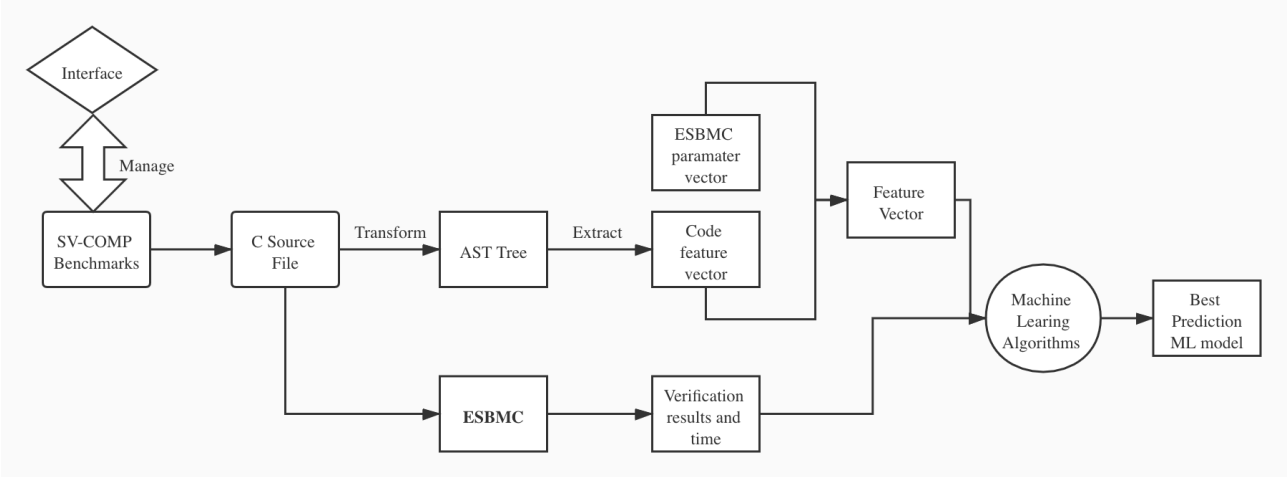

Figure 3.1: Pipeline of the system

As figure 3.1 shown, our system uses an interface to manage SV-COMP benchmarks and extracts C source files from those. For each C source file, we need to process two steps separately: The first step is to convert the C source file to AST, and then extract features from AST. At last, we combined it with ESBMC's parameter vector to become a feature vector for machine learning. The second step is to use ESBMC with different parameter combinations to verify the C source file and to classify the verification result and verification time. Moreover, we use a status code to indicate its specific classification. After completing these two steps, we use the feature vector obtained in the first step as the machine learning training sample, and the status code obtained in the second step as the label of the sample. Finally, the sample and label are used as the input of the machine learning model. After evaluating the performance of the four models on the three sub-category data sets, we choose the best performing model as our prediction model and build our parameter predictor.

3.2 Interface

This experiment provides users with an interface to facilitate users to manage all data sets visually. The main functions are as follows:

1. View the status of each benchmark in current data sets and the actual running results, as picture 3.2 shown:

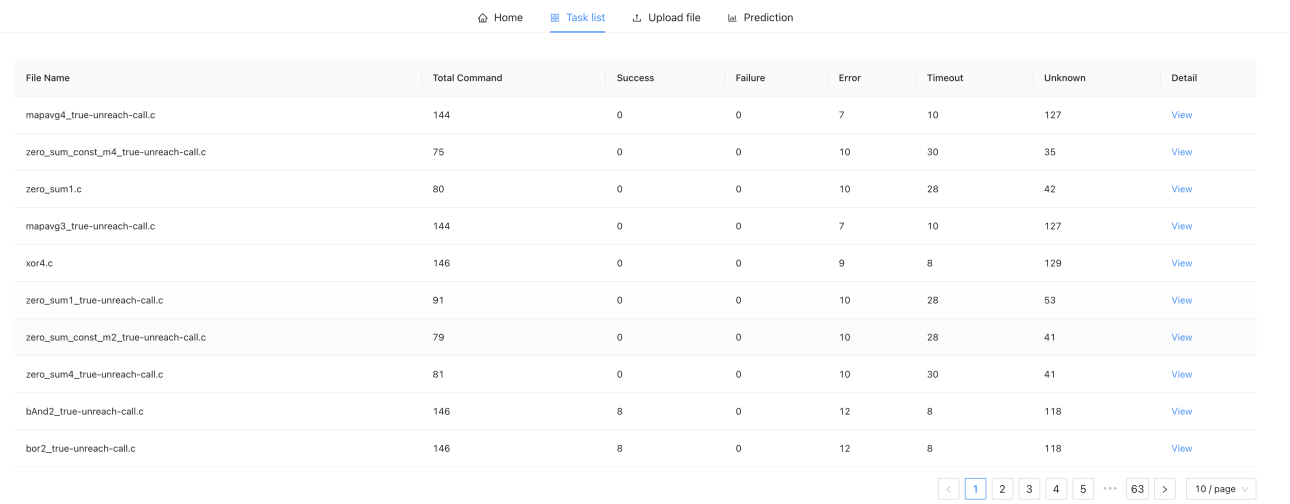

Figure 3.2: The interface of viewing benchmarks

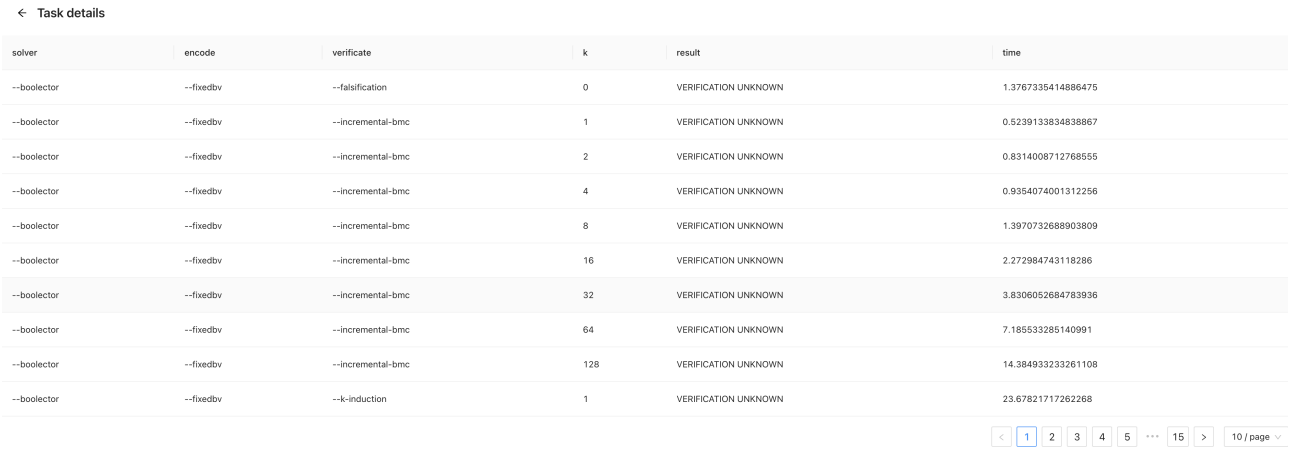

Figure 3.3: The interface of viewing details of benchmark

This page displays the C source files corresponding to all the benchmarks. When clicking on a single line, the user can see the verification results obtained by ESBMC with different ESBMC parameters for the benchmark, as shown in picture 3.3. The verification results contain two parts, one is return status, one is the specific running time (unit is seconds).

2. Add a new benchmark to the specific sub-question data set. As picture 3.4 shown:

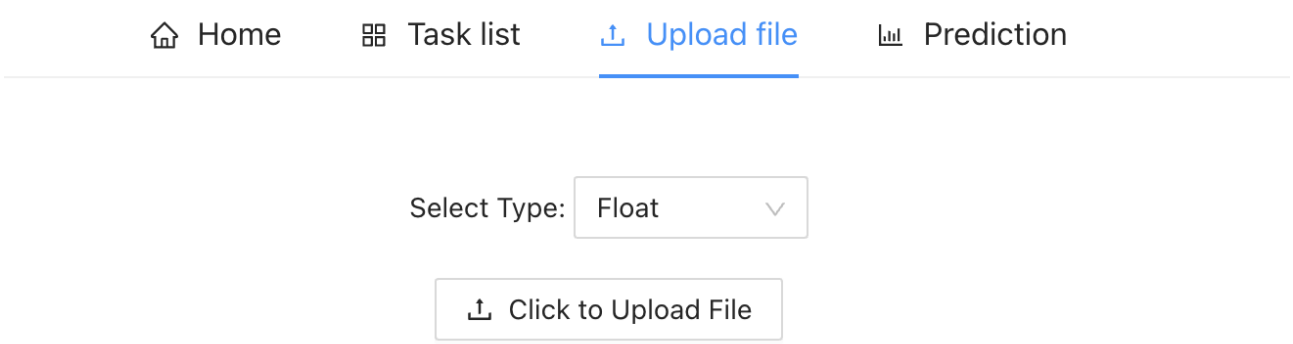

Figure 3.4: The interface of adding benchmarks

Users must specify the sub-category to which the uploaded meta-file belongs when uploading; then they can click the "+" button to add a folder or file locally; when the file is uploaded to the server, our system can automatically identify all uploaded files with ".c" as the source file suffix files. After that, adding them to the corresponding sub-category data set; finally, the system will sequentially verify the C source files with different ESBMC parameter combinations, and store the verification results into mysql database.

3. Predict the best parameters of the new benchmark, and give all the prediction results and the corresponding recommended parameters.

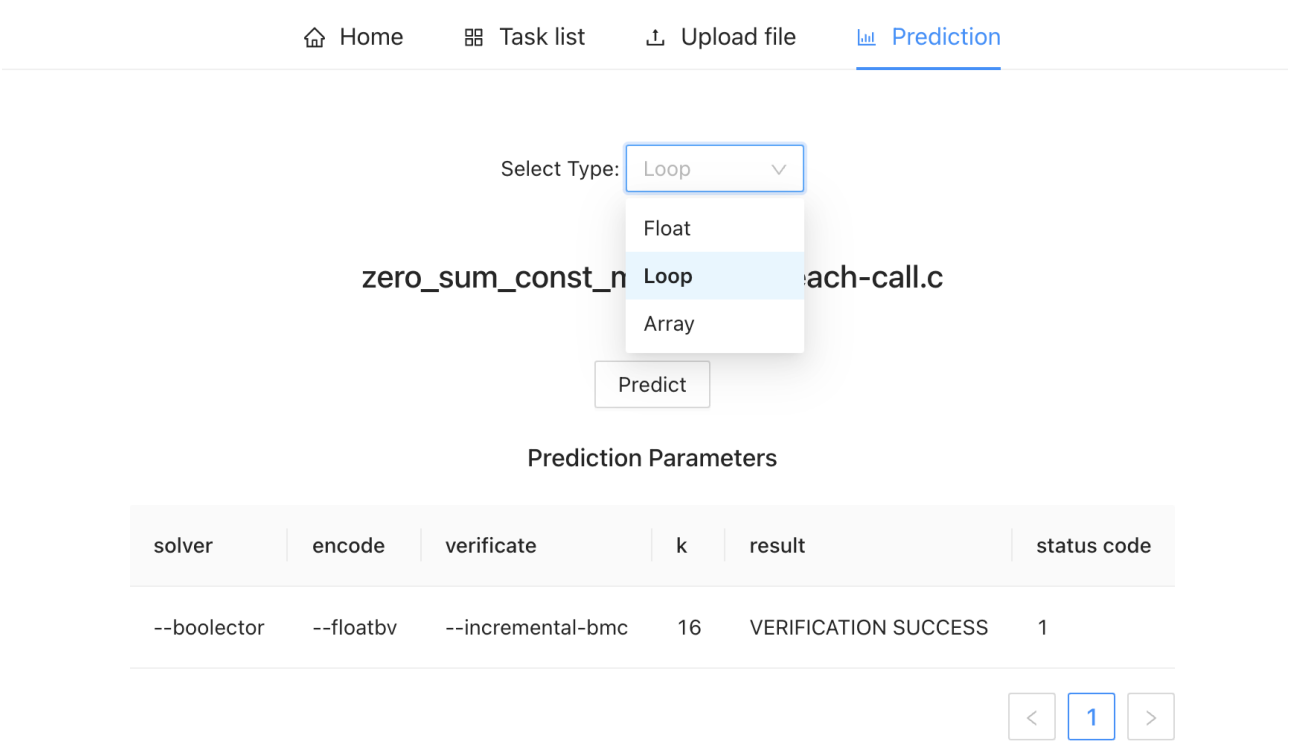

Figure 3.5: The interface of prediction

As shown in picture 3.5, when the user needs to predict the best parameter of a new C source file for the ESBMC, the system gives the prediction verification results of the C source file with different ESBMC parameter combinations after the model prediction. However, the specific result set will not be given. This is because our system uses a different status code to mark the possible runtime of the benchmark.

The entire interface project consists of two parts, one is the front-end project and the other is the back-end project. Their addresses are:

Front-end project: `https://github.com/Grassgod/msc-web`

Back-end project: `https://github.com/Grassgod/backend`

3.3 Benchmark of this experiment

For each C source file, ESBMC can generate multiple benchmarks with different parameters. The main adjustable parameters of ESBMC are divided into three categories:

1. SMT solver.

ESBMC supports five SMT solvers, namely z3, boolector, yices, mathsat and cvc4. Their corresponding parameters are “–z3”, “–boolector”, “–yices”, “–mathsat”, “–cvc”.

2. Encoding method.

   ESBMC supports three encoding methods, namely ir, fixedbv and floatbv. Their corresponding parameters are “–ir”, “–fixedbv”, “–floatbv”.

3. Verification method.

   ESBMC supports three verification methods, namely k-induction, falsification and incremental-bmc. Their corresponding parameters are “–k-induction”, “–falsification”, “–incremental-bmc”.

For a C source file, ESBMC can obtain 45 different parameter combinations by traversing these three types of parameters. For example, “–z3 –fixedbv –falsification” is a parameter combination. A benchmark in our experiment contains a C source file and a combination of parameters.

However, there is one exception. When the SMT solver of ESBMC is boolector and encoding mothed is ir, the verification method can only choose falsification. This is because the built-in boolector in ESBMC cannot support integer encoding mode.

In addition, when the verification method is determined to use k-induction or incremental-bmc, ESBMC can limit the maximum K rounds that the verification process can reach through the specified max-k-step, and specify k-step to set the increment limitation of each verification rounds.

In this experiment, in order to complete the verification process faster, 8 sets of max-k-step and k-step combinations are provided. They are (10, 1), (30, 2), (50, 4), (100, 8), (200, 16), (400, 32), (800, 64), (1600, 128) . i.e.when max-k-step is set to 10, its k-step will be automatically set to 1.

In summary, for a C source file, as table (3.1) shown, there are 5*3*2*8+5*3*1-1*1*2*8=239 kinds of parameter combinations that can be matched. i.e. for a C source file, 239 benchmarks can be generated.

Table 3.1: Parameter combinations of ESBMC

SMT solver	Encode method	Verification method	Max-k-step & K-step
z3	ir	k-induction	(10,1)
boolector	fixedbv	falsification	(30,2)
yices	floatbv	incremental-bmc	(50,4)
mathsat			(100,8)
cvc4			(200,16)
			(400,32)
			(800,64)
			(1600,128)

3.4 Feature engineering

In machine learning, data and features determine the upper limit of machine learning, and models and algorithms only approach this upper limit. It can be seen that feature engineering occupies a

very important position in machine learning. This chapter will introduce in detail the method of feature engineering in this experiment. Feature engineering is divided into three parts, namely feature selection, feature extraction and feature construction.

3.4.1 Data preprocessing

In this experiment, feature extraction cannot be performed due to the C source file belongs to unstructured data. Therefore, the C source file must be transformed into a structured data form. The formats we can choose from are AST (Abstract Syntax Tree), SSA (Static single assignment form), Go-to program and Symbol list. Considering comprehensively, this experiment uses AST as the conversion format.

The advantages of AST are:

1. The conversion speed is fast, less than 1 second.
2. It is a tree structure, similar to the code structure. Normally, it can retain the context structure and is easy to analyze and extract features.
3. It is supported by the python-based pycparser library and is easy to use.

The AST generated after transformation is a tree structure, each node of the tree represents a specific definition or expression. As listing (1) shown, this is a variable type definition of C, which is converted into an AST node object as shown in listing (2).

```
double x;
```

Listing 1: Example code

```
Decl(name='x',
     quals=[],
     storage=[],
     funcspec=[],
     type=TypeDecl(declname='x',
                   quals=[],
                   type=IdentifierType(names=['double'])
                   ),
     init=None,
     bitsize=None
   )
```

Listing 2: AST node example

When we analyze the AST node object shown in listing 2, we can see that outside the brackets is the type of the current node object. In this case, its type is “Decl”, which is the type definition. The definition of the remaining attributes is shown in listing 3.

```
# name: the variable being declared
# quals: list of qualifiers (const, volatile)
# funcspec: list function specifiers (i.e. inline in C99)
# storage: list of storage specifiers (extern, register, etc.)
# type: declaration type (probably nested with all the modifiers)
# init: initialization value, or None
# bitsize: bit field size, or None
#
Decl: [name, quals, storage, funcspec, type*, init*, bitsize*]
```

Listing 3: The explanation of AST example node

3.4.2 Feature selection

From the pycparser[62] document, there are 47 different AST node object types. After analysis, we selected 13 more important and common object types as features. They are:

'Decl','For','While','DoWhile','If','Switch','Assignment','FuncCall','Label','Constant','TypeDecl', 'IdentifierType','Goto'.

In order to further reduce the number of features, we carried out two operations to merge features:

1. Since the essence of 'For' loop, 'While' loop, and 'DoWhile' loop are the same, we combine them into one feature 'loop'.

2. 'Switch' can be regarded as composed of multiple 'If', so we merge 'Switch' into 'If'.

In addition, we found that recursion cannot be detected in the AST tree. However, recursion is an important feature that affects the verification time of ESBMC, so we developed an algorithm to detect recursion in the AST tree. The specific idea is that when the type of AST node is found to be 'FuncCall', which is a function call, we will compare it with the type of its father AST node. If it is found that the type of its father node is 'FuncDef' (function definition) and the naming of the two is consistent, It will be recognized as a standard recursion. The pseudo-code is shown in the algorithm (1).

Algorithm 1 Detecting recursion

Input: current_node, father_node
Output: Whether there is recursion

if $current\_node.class = FuncCall$ **and** $father\_node.class == FuncDef$ **and** $current\_node.name == father\_node.name$ **then**
  return True
else
  return False
end if

3.4.3 Feature extraction

For feature extraction, the method we adopt is to count the number of different AST node objects in the AST. i.e. If there are 10 “Decl” objects in the AST, the value of this feature is 10. This method is relatively simple to use and easy to explain. This is because there is a difference in the time it takes for ESBMC to parse different AST node objects, so quantitative statistics are helpful for us to analyze the time consumed by ESBMC for different C source codes.

3.4.4 Feature construction

When we traverse and parse the AST, we will obtain a feature vector of the C source file, but the feature vector represents only the current C source file. In order to generate the feature vector of the benchmark for our experiment, we also need to add parameter features.

As can be seen from the foregoing, a benchmark consists of a C source file and ESBMC parameters, then what we need is to convert the parameters into parameter features. In this experiment, we use different feature codes to represent different parameters. The specific parameters and corresponding feature codes are shown in the table (3.2)

Table 3.2: Parameter feature code

SMT solver	code	Encode method	code	Verification method	code	Max-k-step & K-step	code
z3	0	ir	0	k-induction	0	(10,1)	10
boolector	1	fixedbv	1	falsification	1	(30,2)	30
yices	2	floatbv	2	incremental-bmc	2	(50,4)	50
mathsat	3					(100,8)	100
cvc4	4					(200,16)	200
						(400,32)	400
						(800,64)	800
						(1600,128)	1600

When the feature vector of the C source file and the parameter feature vector of the benchmark are obtained, we splice them together to form the feature vector of the benchmark. At this point, we have completed the feature construction.

3.5 Preparing data

SV-comp has been held in 2012. So far, there have been more than 10 years of data sets and more than 10 sub-categories. The purpose of this experiment is to predict the performance of ESBMC in SV-COMP, so our data set uses the SV-COMP competition data set. Taking into account the performance of the existing experimental equipment and the training time of the machine learning model, we select the data set from 2016 to 2021, and the sub-categories select “array”, “loop” and “floats’ ’.

For each SV-COMP benchmark, it contains a C source file with suffix “.c” and a configuration file with suffix “.yml”. For this experiment, we only need the C source code files, so we extract all the C source files. The number of each sub-category data set is shown in the table (3.3). As mentioned

above, each C source file can generate 239 benchmarks for this experiment, so we can calculate the number of benchmarks for each sub-category. In order to simulate the actual situation, we use 16-20 years of data as the training set and 21 years of data as the test set. As shown in the table (3.3).

Table 3.3: Data distribution

	array	loop	floats
file number	622	627	1027
benchmark	148658	149853	245453
train file	422	427	727
benchmark	422*239	427*239	727*239
test file	200	200	300

3.6 Using ESBMC to get the benchmark' label

For each generated benchmark, we need to generate a sample for the machine learning model. A sample contains feature vectors and classes. For the feature vector, as mentioned above, we need to combine the feature vector of the C source file contained in it and the corresponding parameter vector to generate the feature vector of the benchmark.

For classes, we need to use ESBMC to complete the benchmark verification process to obtain actual verification results.

3.6.1 Using ESBMC to test a single benchmark

For ESBMC, it can support command line for testing. A standard ESBMC command is shown in the code (4).

```
esbmc c_filename.c --boolector --fixedbv --incremental-bmc
                   --max-k-step 400 --k-step 32 --timeout 15m
```

Listing 4: The example of ESBMC command

Among them, "esbmc" represents the actual address of esbmc after installation, and can also be replaced by a soft link which set in the shell; "c_filename.c" represents the address of the C source file that needs to be verified; "–boolector" represents the selected SMT solver parameter; "–fixedbv" represents the selected encoding method; "–incremental-bmc" represents the selected verification method; "—max-k-step" and "—k-step" represent the maximum number of iterations and the number of rounds added in each iteration, respectively. This parameter is only available when the verification method is k-induction or incremental-bmc. Finally, in order to ensure that ESBMC will not take too long to verify a single benchmark, we use "—timeout 15m" to limit the maximum running time of ESBMC. The result will be regarded as a timeout when a single test time exceeds 15 minutes.

3.6.2 Using multi-threading technology to speed up the testing process

Since the number of benchmarks to be completed is close to 450,000, we cannot simply process linearly and need to use some skills to speed up the verification process. In this experiment, we use multithreading to speed up the entire testing process. It is worth mentioning that due to the Global Interpreter Lock mechanism of the multi-threading technology supported by python itself, each CPU can only execute one thread at the same time, which is very unfriendly to CPU-intensive code. So we use python's multi-process mechanism, that is, open a separate python process for each benchmark, and an independent CPU completes the test. Finally, these python processes are managed through the process pool.

In this experiment, we used three Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz servers, each with 32 logical CPUs. This allows us to verify 32 benchmarks at the same time, which greatly speeds up the entire testing process.

3.6.3 Categorizing the results of ESBMC

For benchmarks, we need to classify the verification results they may return to facilitate classification in the machine learning process later.

After completing all benchmark tests, we found that the test results returned the following status codes:

1. ERROR: This means that there was an error in the verification process. This error is relatively rare. The main reason is that the ESBMC cannot complete the verification process during the process. In this experiment, the main reason is that the ESBMC version installed on the server cannot support the smt solver of the mathsat type. Since it has little impact on the experimental results and does not affect the evaluation process of the model, this type of result is retained.

2. VERIFICATION_UNKNOWN: This means that ESBMC cannot complete the test process within the specified number of iterations, and the result is a normal result.

3. Timed_out: This means that ESBMC cannot complete the verification process within 15 minutes.

4. VERIFICATION_FAILED: This means that ESBMC failed to verify the test result against the current benchmark, which is actually a valid result.

5. VERIFICATION_SUCCESS: This means that ESBMC has successfully verified the test result with the current benchmark, which is also a valid result.

When the return result of ESBMC is VERIFICATION_FAILED or VERIFICATION_SUCCESS, it means that they are valid conclusions, and the ESBMC verification time is also obtained. Therefore, we divide five-time intervals to define their test speeds. This step is mainly to facilitate the application of benchmark results to machine learning. The specific time interval definitions and corresponding speed states are shown in the table (3.4):

Table 3.4: Status code of benchmark

	VERIFICATION_FAILED or VERIFICATION_SUCCESS				Timed out	VERIFICATION UNKNOWN	ERROR
	Really Fast	Fast	Normal	Slow			
Time interval (seconds)	[0, 1)	[1, 60)	[60, 300)	[300, 900)			
Status code	0	1	2	3	4	5	6

Similarly, we also need to set a status code for other types of test results. As shown in the table 3.4, the status code of Timed out is 4, VERIFICATION_UNKNOWN is 5, and ERROR is 6.The purpose of setting the status code in this way is that we hope that the smaller the value of the status code can represent a more optimized result.

3.7 Machine learning methods

In machine learning, the choice of model is very important. Different models will get significantly different results because they are based on different principles. When selecting a model, in addition to the accuracy of model prediction, we also need to consider the difficulty and speed of model training, as well as the use of the model in actual scenarios. The following will introduce the tuning method of the model's hyperparameters and the detailed implementation of each model.

3.7.1 SVM

For SVM, we use sklearn.svm.SVC provided in sklearn as the model basis, and the main adjustment parameters are "C", "kernel", "gamma". "C" stands for regularization paramete, the strength of the regularization is inversely proportional to C, and it must be positive. "kernel" represents the kernel function selected by the SVM model, which can be 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'. "gamma" represents the kernel function parameters when the kernel function is 'poly', 'rbf', 'sigmoid', and 'auto' and 'scale' can be selected.

3.7.2 Decision tree

For the decision tree, we use the sklearn.tree.DecisionTreeClassifier provided in sklearn as the model basis. The main adjustment parameters are 'criterion', 'splitter', and 'max_features'.'criterion' can choose 'gini' or 'entropy', the former is the gini coefficient, The latter is information entropy, which represents the selection of different decision tree categories. 'Splitter' represents how to split the feature. You can choose 'best' and 'random'. When 'best' is selected, time-consuming may increase. 'Max_features' represents the maximum number of features, and the options are 'auto', 'log2', and 'sqrt'.

3.7.3 KNN

For KNN, we use sklearn.neighbors.KNeighborsClassifier in sklearn as the model basis, and the main adjusted parameters are 'n_neighbors' and 'weights'. 'N_neighbors' represents the number of selected neighbours. 'Weights' represents how to set the weight of the distance, you can choose 'uniform' and 'distance', 'uniform' means that the hyperparameter of distance weight is not considered, and 'distance' means that the distance weight is considered. When 'weights' is set to 'distance', the parameter 'P' can be set to specify which distance measurement method to use. When p = 1, this is equivalent to using manhattan_distance (L1), and euclidean_distance (L2) for p = 2. For arbitrary p, minkowski_distance (L_p) is used.

3.7.4 Neural network

For Neural network, we use sklearn.neural_network in sklearn as the model basis, and the main parameters adjusted are 'learning_rate_init' and 'max_iter'. 'learning_rate_init' represents the initial learning rate. And 'max_iter' represents the maximum number of iterations. The larger the value, the higher the probability of model convergence.

3.7.5 GridSearch CV

The purpose of GridSearchCV is to tune parameters automatically. The essential method is a greedy algorithm. Its idea is to tune the parameters that currently have the greatest impact on the model until the optimization, then take the next parameter tuning that has the most impact, and so on, until all parameters have been adjusted. The disadvantage of this method is that it may be adjusted to the local optimum instead of the global optimum, but it can save time. This experiment uses the GridSearchCV method in sklearn to systematically traverse multiple parameter combinations and determine the best effect parameters through cross-validation.

The following explains how to use GridSearchCV by introducing how to adapt GridSearchCV to the SVM model.

As shown in the code (5), we can set the value range of different parameters in the SVM by setting param_grid, and use it as an input to GridSearchCV. In addition, "accuracy'" is designated as the main indicator for evaluating the effect of the model, which means that GridSearchCV will automatically use the "accuracy" method of cross-validation to select the best model. The "cv" parameter specifies that the data set is divided into 5 groups for cross-validation.

```
1 param_grid = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
2               'gamma': ['scale', 'auto']}
3 grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
```

Listing 5: The example of GridSearch CV

3.8 Automatically predicting optimal parameters

This experiment uses the benchmark in the train data set to complete the training of the model, and four models can be obtained, which are based on SVM, Decision Tree, KNN and Neutral network models. Next, we need to predict the data in the test data set. Each piece of data in the test data set represents a C source file. As we can see from the foregoing, each C source file can be matched with 239 different ESBMC parameters. The specific process is as follows:

1. Perform feature extraction on the C source file to generate feature vectors.
2. Generate 239 benchmark feature vectors with the parameter vectors generated by 239 different ESBMC parameters combinations.
3. Use the model to predict 239 feature vectors, and obtain their prediction result status codes.
4. Traverse 239 types of status codes, select the smallest status code as the optimal prediction result and record the ESBMC parameters corresponding to the status code.
5. Output the parameter.

3.9 Summary

This chapter specifically elaborates the specific details and key content of the experiment. First, we develop a visual interface to manage all benchmarks from SV-CPMP. Then, we extracted the required C source files from the benchmarks of SV-COMP and combined the different parameters of ESBMC to form the benchmark for this experiment.

For each set of benchmarks used in this experiment, we need to perform feature engineering and label acquisition respectively. In the feature engineering part, we convert the C source file into AST and perform feature engineering from the AST to obtain the corresponding feature vector. Combined with the parameter vector formed by the ESBMC parameter combination corresponding to the benchmark, we can obtain the feature vector of the benchmark as a sample. For the label part, we combined Python's multi-process technology to use ESBMC with the benchmark's ESBMC parameter combination to verify the C source file to obtain the verification time and verification result. Finally, we use status codes to correspond to different verification results and times, which is the label.

So far, we have obtained samples and labels. After that, we used GridSeachCV technology to automatically tune the four models. Training, verification and testing are performed on three sub-classification data sets, and a total of 12 results can be obtained.

Chapter 4

Result Evaluation and Discussion

4.1 Description of the evaluation benchmarks

This experiment is a multi-classification problem. Compared with a simple binary classification problem, the corresponding evaluation scheme will be changed. The following will first discuss the commonly used indicators in the binary classification problem, and then we will extend these indicators to the multi-classification problem.

Before discussing these issues, let us first review the shortcomings of the most common indicator "Accuracy". Accuracy is the most commonly used indicator in binary classification problems, it calculates the ratio of the number of correct predictions to the total number of predictions. However, for unbalanced data sets (multi-classification problems), accuracy is not a good indicator. For example, suppose we have 100 pictures, of which 91 pictures are "dogs", 5 are "cats", and 4 are "pigs", we hope to train a three-classifier that can correctly identify the types of animals in the pictures. Among them, the dog category is the majority class. When the number of samples (dogs) in most classes far exceeds that of other classes (cats, pigs), if accuracy is used to evaluate the quality of the classifier, then even if the model performance is poor (for example, no matter what picture is input, it will be predicted as "Dogs"), we can also get a higher accuracy score (such as 91%). At this time, although the accuracy score is very high, it is of little significance. When the data is abnormally unbalanced, the flaws of the Accuracy evaluation method are particularly significant.

Therefore, we need to introduce Precision, Recall and F1-score evaluation indicators. Considering that the calculation methods of the evaluation indicators are slightly different in the two-class and multi-class models, we will discuss them separately.

4.1.1 Accuracy,Precision,Recall,F1-score

4.1.1.1 Common indicators of binary classification models

In the binary classification problem, suppose the sample has two categories: Positive and Negative. When the classifier's prediction is over, we can draw a confusion matrix. As shown in figure 4.1, the classification results are divided into the following categories:

		Prediction outcome		
		p	**n**	**total**
Actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

- True Positive (TP): The positive sample is successfully predicted as positive.
- True Negative (TN): Successfully predict negative samples as negative.
- False Positive (FP): falsely predict negative samples as positive.
- False Negative (FN): The false prediction of positive samples is negative.

Table 4.1: Binary confusion matrix

In the binary classification model, the definitions of Accuracy (4.1), Precision (4.2), Recall (4.3) and F1 score(4.4) are shown in the below:

$$Accuracy = \frac{TP+TF}{TP+TF+FP+FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP+FP} \tag{4.2}$$

$$Recall = \frac{TP}{TP+FN} \tag{4.3}$$

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN} \tag{4.4}$$

Among them, precision focuses on assessing how much real positive data occupies among all the data predicted to be positive. Recall focuses on evaluating all the positive data, that is to tell us how many data have been successfully predicted as positive. It should be noted here that we need to focus on different indicators according to different situations.

When the cost of False Negative (FN) is very high (the consequences are serious) and we want to avoid generating FN as much as possible, we should focus on improving the recall indicator. And when the cost of False Positive (FP) is very high (the consequences are serious), that is, when you want to avoid FP as much as possible, we should focus on improving the Precision index.

Take the spam blocking system as an example. Spam is positive, normal email is negative, and false positive is to identify normal email as spam. This situation should be avoided (we cannot tolerate an important work email directly into the spam box). We would rather mark spam as normal mail (FN) than let normal mail go directly to the trash can (FP). Here, the goal of the spam blocking system is to increase the Precision value as much as possible even at the expense of part of the recall. And F1-score is a combination of precision and recall.

4.1.1.2 Extending these indicators to multi-classification problems

Here we take the three classification problems as an example, as shown in figure 4.2, to see how to calculate the value of each indicator based on the confusion matrix.

Confusion Matrix		Prediction		
		Class A	Class B	Class C
Actual	Class A	a	b	c
	Class B	d	e	f
	Class C	g	h	i

Table 4.2: The confusion matrix of three classification problem

First, let's define TP, TN, FP and FN. At this time, we need to deal with each class separately. For class A, as shown in the table 4.3:

	label = A	label = B or label = C
predict = A	TP(A)	FP(A)
predict = B or predict = C	FN(A)	TN(A)

Table 4.3: Calculate TP, FP, TN, FN of class A

The calculations for other classes are very similar, and the details are not described here.

In the confusion matrix, the correct classification samples (Actual label = Predicted label) are distributed on the diagonal from the upper left to the lower right. Among them, accuracy is defined as the ratio of the number of samples correctly classified (on the diagonal) to the total number of samples. Accuracy measures the global sample prediction situation. For Precision and Recall, each class needs to calculate its Precision and Recall separately. Among them, the precision rate and recall rate of category A are as shown in formula 4.5 and 4.6, and the calculations of other classes are similar.

$$Precision(ClassA) = \frac{a}{a+d+g} \tag{4.5}$$

$$Recall(ClassA) = \frac{a}{a+b+c} \tag{4.6}$$

If we want to evaluate the overall function of the recognition system, we must consider the comprehensive predictive performance of the three classes. Generally speaking, we can use the following three methods:

1. Macro-average

   This method is the simplest. It directly adds up the evaluation indicators of different categories (Precision/ Recall/ F1-score) and averages them, giving all categories the same weight. This method can treat each category equally, but its value will be affected by the rare category.

$$Macro-Precision=\frac{P(A)+P(B)+P(C)}{3} \tag{4.7}$$

$$Macro-Recall=\frac{R(A)+R(B)+R(C)}{3} \tag{4.8}$$

2. Weighted-average

   This method gives different weights to different categories (the weight is determined according to the true distribution ratio of the category), and each category is multiplied by the weight and then added. This method takes into account the imbalance of categories, and its value is more susceptible to the influence of the majority class.

$$Weighted-Precision=P(A)\times W(A)+P(B)\times W(B)+P(C)\times W(C) \tag{4.9}$$

$$Weighted-Recall=R(A)\times W(A)+R(B)\times W(B)+R(C)\times W(C) \tag{4.10}$$

3. Micro-average

   This method adds up the TP, FP, and FN of each category, and then calculates it according to the two-category formula.

$$Micro-Precision=\frac{TP(A)+TP(B)+TP(C)}{TP(A)+TP(B)+TP(C)+FP(A)+FP(B)+FP_{(C)}} \tag{4.11}$$

$$Micro-Recall=\frac{TP(A)+TP(B)+TP(C)}{TP(A)+TP(B)+TP(C)+FN(A)+FN(B)+FN_{(C)}} \tag{4.12}$$

4.1.2 Cross-validation

Cross-validation is a common method used in machine learning to build models and optimize model parameters. The main method is to use the data repeatedly, divide the obtained sample data into different training sets and test sets, use the training set to train the model, and use the test set to evaluate the prediction of the model. On this basis, multiple sets of different training sets and test sets can be obtained. A sample in a certain training set may become a sample in the test set next time, so-called “crossover”. In this experiment, we use S-Folder Cross Validation. S-fold cross-validation will randomly divide the sample data into S parts, each time S-1 part is randomly selected as the training set, and the remaining 1 part is used as the test set. When this round is completed, randomly

select S-1 copies to train the data. After several rounds (less than S), the loss function is selected to evaluate the optimal model and parameters. In this experiment, S is 5, that is, 5-fold cross-validation.

4.2 Setup

For the software part, the visualization system is based on Java Web, Html, Javascript and CSS technology; python3 and pycparser are used in feature extraction to parse C source files and obtain feature vectors; In order to generate a data set that can be used for machine learning, we develop a multi-process python3 programs based on Linux. Its main function is to call ESBMC to verify different C source files. In the end, we use MySQL to store the verification results; Machine learning uses four models based on sklearn and uses cross-validation for evaluation. Optimization uses GridSearchCV technology to optimize the model. For the hardware part, we set up the visualization system on the Google Cloud server, the operating system is Linux, and Jenkins is used to implementing CI automatic release. The machine learning model runs on three Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz linux servers.

4.3 Objectives

Our goal is to comprehensively evaluate the performance of the four models on the three sub-categories data sets through Accuracy, Precision, Recall, and F1-score, and select a better model as our prediction model. In addition, the real training time is also under our consideration.

4.4 Results

In this experiment, we used four machine learning methods to build 12 models for three sub-category data sets. We will discuss them in detail separately.

4.4.1 Decision Tree for Array sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	8615	39	0	1	108	48	63
	1	40	74	0	0	15	47	1
	2	0	0	1	0	18	5	0
	3	0	0	0	1	0	0	0
	4	93	29	10	0	6064	379	81
	5	75	19	5	3	286	7604	69
	6	55	2	0	0	104	73	10389

Table 4.4: Confusion matrix of Decision Tree based on Array data set

4.4.2 SVM based on Array sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	4966	0	0	0	889	560	2455
	1	7	0	0	0	5	33	156
	2	0	0	0	0	5	2	12
	3	0	0	0	0	0	0	2
	4	526	0	0	0	3467	891	1782
	5	510	0	0	0	591	5685	1381
	6	1596	0	0	0	1162	1788	5945

Table 4.5: Confusion matrix of SVM based on Array data set

4.4.3 KNN based on Array sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	7971	28	0	1	121	99	617
	1	27	56	2	0	23	44	19
	2	0	0	1	0	20	6	1
	3	0	0	0	1	0	0	0
	4	96	37	8	1	5632	259	691
	5	44	16	1	0	258	7111	535
	6	576	8	0	0	792	829	8485

Table 4.6: Confusion matrix of KNN based on Array data set

4.4.4 Neural network based on Array sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	7543	15	0	0	713	281	285
	1	42	32	0	0	57	37	3
	2	4	4	0	0	18	2	0
	3	0	0	0	0	1	0	0
	4	446	20	13	0	5643	536	66
	5	205	2	0	0	752	6927	79
	6	474	0	0	0	543	323	9350

Table 4.7: Confusion matrix of neural network based on Array data set

4.4.5 Decision Tree based on Loop sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	3532	117	14	3	195	1003	52
	1	140	868	14	6	28	99	9
	2	23	11	82	3	7	9	1
	3	3	4	0	104	6	6	3
	4	289	46	13	9	2276	285	39
	5	1041	63	7	0	303	6599	72
	6	46	13	1	5	47	116	5332

Table 4.8: Confusion matrix of decision tree based on Loop data set

4.4.6 SVM based on Loop sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	510	38	0	0	244	2704	1320
	1	163	349	0	0	68	229	354
	2	34	7	0	0	33	28	31
	3	10	0	0	0	46	11	70
	4	411	47	0	0	721	783	1081
	5	460	3	0	0	115	5578	1981
	6	50	109	0	0	133	1618	3605

Table 4.9: Confusion matrix of SVM based on Loop data set

4.4.7 KNN based on Loop sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	2922	112	14	11	240	1077	540
	1	190	717	2	2	41	87	125
	2	28	14	51	4	17	5	17
	3	3	8	6	60	11	1	37
	4	270	40	16	13	1993	212	413
	5	977	55	4	0	185	6430	434
	6	631	147	11	20	492	745	3514

Table 4.10: Confusion matrix of KNN based on Loop data set

4.4.8 Neural network based on Loop sub-data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	1722	59	1	15	953	2146	20
	1	267	503	2	0	242	117	33
	2	44	19	8	3	39	17	6
	3	8	27	3	38	41	9	0
	4	630	80	0	18	1667	481	81
	5	1146	16	0	0	835	6034	54
	6	158	16	0	0	110	243	5033

Table 4.11: Confusion matrix of neural network based on Loop data set

4.4.9 Decision Tree based on Floats sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	9947	201	3	6	62	97	218
	1	305	1666	54	2	147	20	38
	2	0	63	231	10	59	0	8
	3	10	19	23	76	49	0	5
	4	109	201	39	23	1454	52	26
	5	112	19	4	0	53	784	54
	6	235	39	5	0	21	46	16530

Table 4.12: Confusion matrix of decision tree based on floats data set

4.4.10 SVM based on Floats sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	6100	0	0	0	11	1	4199
	1	1106	0	0	0	25	1	1217
	2	176	0	0	0	21	0	190
	3	116	0	0	0	4	0	39
	4	949	0	0	0	132	1	783
	5	600	0	0	0	21	5	464
	6	3157	0	0	0	37	0	13770

Table 4.13: Confusion matrix of decision tree based on floats data set

4.4.11 KNN based on Floats sub data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	9116	272	7	15	53	79	992
	1	477	1252	54	2	126	3	318
	2	24	94	131	13	68	0	41
	3	16	26	38	25	52	0	25
	4	113	274	61	36	1110	47	263
	5	103	7	2	0	34	763	117
	6	1825	280	22	9	320	157	14263

Table 4.14: Confusion matrix of KNN based on floats data set

4.4.12 Neural network based on Floats sub-data set

		Predict						
		0	1	2	3	4	5	6
Actual	0	8632	420	21	2	210	149	1100
	1	777	1036	31	2	274	16	96
	2	77	174	27	4	79	7	3
	3	28	49	18	0	87	0	0
	4	188	363	35	1	1046	45	226
	5	86	37	5	0	136	555	207
	6	190	9	1	0	53	40	16583

Table 4.15: Confusion matrix of neural network based on floats data set

4.5 Threats to validity

Based on the overall experimental results, we found that there are two factors that may threaten the validity of the experimental results.

1. Let's discuss our data below. As shown in table 4.16, the following are the statistical data of the returned results which come from our three data sets:

	array	loop	floats
VERIFICATION UNKNOWN	42879	49945	8264
Timed out	6687	5237	3179
ERROR	7214	6322	22094
VERIFICATION SUCCESS	216	3166	7057
VERIFICATION FAILED	5406	4516	7613

Table 4.16: Statistics of results of benchmarks

We will find that the number of different returned results is not very average, and there are large categories with very prominent sample numbers, so we need to use the value of macro-average.

2. As table 4.16 shows,the number of results classified as ERROR is very large. Considering that ERROR is not a valid result (because it is often caused by a mismatch between the verification tool and the input C source file), it will affect the actual prediction result. However, considering that the causes of ERROR are very diverse, so it is important whether a model can classify ERROR well. In the end, we do not choose to exclude this type of data.

   In addition, in order to eliminate as much as possible the threat of such results to the validity of the model, we will use the verification results and time obtained by using the default parameters of ESBMC as the baseline. That is, we can compare the baseline with the results generated by our parameter predictor. If we can save a certain amount of verification time compared to the baseline, it means that our model is effective.

4.6 Comparison and evaluation

4.6.1 Result analysis

	Decision Tree	SVM	KNN	Neural network
Best Cross validation Score	0.8838	0.5591	0.7554	0.7962
Micro F1 score	0.9403	0.5829	0.8501	0.8570
Macro F1 score	0.7245	0.3348	0.6173	0.5254
Weighted F1 score	0.9399	0.5810	0.8495	0.8578
Accuracy	0.9403	0.5829	0.8501	0.8570
Micro recall	0.9403	0.5829	0.8501	0.8570
Macro recall	0.7401	0.3346	0.6841	0.5177
Weighted recall	0.9403	0.5829	0.8501	0.8570
Training time(seconds)	3	5828	107	323

Table 4.17: The scores of the four models on the Array dataset

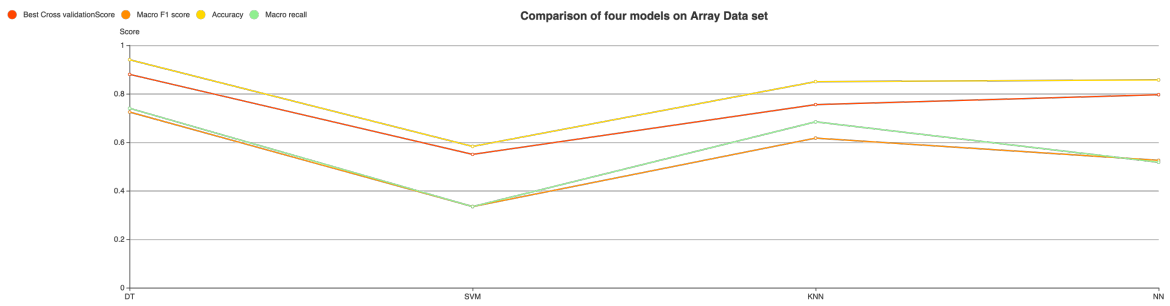

Figure 4.1: Comparison on Array data set

	Decision Tree	SVM	KNN	Neural network
Best Cross validation Score	0.6584	0.4401	0.5230	0.5829
Micro F1 score	0.8190	0.4690	0.6837	0.6539
Macro F1 score	0.7790	0.2847	0.6088	0.5046
Weighted F1 score	0.8194	0.4286	0.6824	0.6499
Accuracy	0.8190	0.4690	0.6837	0.6539
Micro recall	0.8190	0.4690	0.6837	0.6539
Macro recall	0.7767	0.2831	0.5946	0.4797
Weighted recall	0.8190	0.4690	0.6837	0.6539
Training time(seconds)	4	7714	95	381.1

Table 4.18: The scores of the four models on the Loop dataset

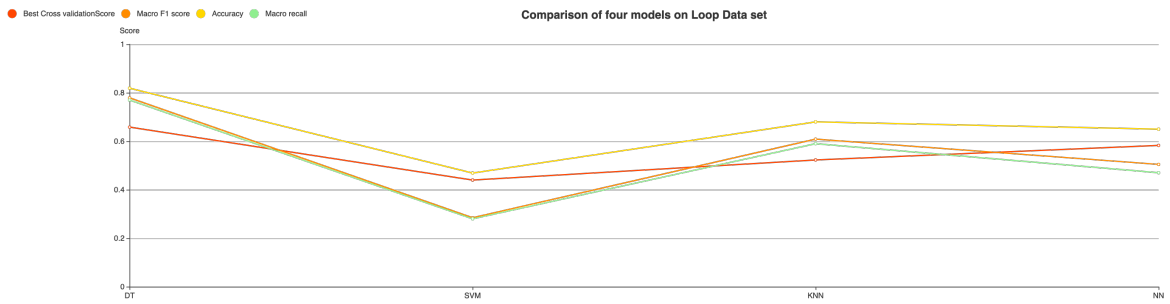

Figure 4.2: Comparison on Loop data set

	Decision Tree	SVM	KNN	Neural network
Best Cross validation Score	0.8234	0.5864	0.6991	0.7466
Micro F1 score	0.9264	0.6039	0.8048	0.8416
Macro F1 score	0.7653	0.2010	0.5932	0.5040
Weighted F1 score	0.9256	0.5508	0.8037	0.8331
Accuracy	0.9264	0.6039	0.8048	0.8416
Micro recall	0.9264	0.6039	0.8048	0.8416
Macro recall	0.7483	0.2112	0.5840	0.4899
Weighted recall	0.9264	0.6039	0.8048	0.8416
Training time(seconds)	6	22987	188.5	594.4

Table 4.19: The scores of the four models on the Floats dataset

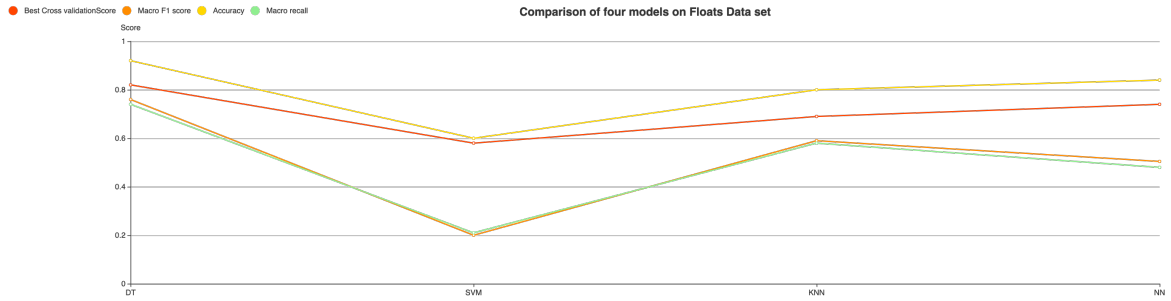

Figure 4.3: Comparison on Floats data set

Combining figure (4.1)(4.2)(4.3), we can intuitively see the scores of the four models on the three sub-categories data sets.

From the scores of the four models on three different data sets (4.17) (4.18) and (4.19), it can be seen that the scores of the decision tree are better than those of the other three models. So next we mainly discuss the decision tree. Considering the actual situation, we choose the decision tree as our optimal model. This is because it not only has good results but also can complete the training and prediction of the model in a short time, which is more in line with our usage scenarios.

4.6.2 Reusability

In addition to the performance of the model on a single data set, we also want to test the generalization ability of the model across data sets. If the model can have good generalization ability across data sets (that is, the model trained on the Array sub-data set can predict the data of the Loop sub-data set well), we can no longer consider the differences between the various sub-problems. This means we only need to combine all the data sets to form a super-large data set, and then complete the training of the model on this super-large data set to predict the benchmarks of all sub-categories.There is no doubt that this will be more convenient.

The details of the experiment we designed are as follows. First, we merge the data sets of Array and Loop, then use the decision tree algorithm to train on the new data set and finally use the data set of floats as the test set to test the effect.

		Predict						
		0	1	2	3	4	5	6
Actual	0	21023	518	23	8	189	208	699
	1	635	4048	100	16	463	38	114
	2	27	189	427	41	146	8	11
	3	36	45	25	215	127	0	7
	4	279	423	98	59	3776	83	89
	5	252	34	6	2	99	1953	135
	6	764	124	12	13	134	214	35701

Table 4.20: Confusion matrix of Decision tree on combined data set

	Decision Tree
Best Cross validation Score	0.6029612265209618
Micro F1 score	0.9118230213482535
Macro F1 score	0.7548317227763748
Weighted F1 score	0.9113396267656486
Accuracy	0.9118230213482535
Micro recall	0.9118230213482535
Macro recall	0.7413116581624051
Weighted recall	0.9118230213482535
Training time(seconds)	15

Table 4.21: Decision tree's ability to generalize across data sets

As can be seen from table 4.21, we can see that the performance of some models of the decision tree is very good, such as the Micro F1 score, but this does not mean that the generalization ability of the model is very good. This is because there are a large number of errors in the merged data set (category 6) (table 4.20). Due to a large number of error categories, the score of the entire model is improved. So we need to pay attention to Macro F1 score and Cross-validation score. We can see that these two scores are at a relatively average level, so we can conclude that the decision tree model has a certain generalization ability.

4.6.3 Baseline

In order to understand the application of the decision tree model in practice better, we compare the predicted results with the actual ESBMC verification time to determine whether our prediction program can help users save time and how much time has been saved.

The details of the experiment are as follows. For the C source files in the Array and Loop data sets, we use the test set (SV-comp2021 data) as the data used in this test. First, we have to establish a baseline, that is, to obtain the verification time and results of the C source file using ESBMC without any parameters (using the default parameters of ESBMC). After that, we use our automatic parameter prediction program to get the best parameter for each C source file and records the test time and test results. It is worth noting that when the running time exceeds 15 minutes, we will default the test time for the result to 15 minutes.

The results are shown in the table 4.22:

	Array	Loop
Baseline(seconds)	535958	407456
Automatically select parameters(seconds)	301287	223730
Total time saved(seconds)	234671	183726

Table 4.22: Time-consuming comparison

We can see that compared to the baseline, our program can save 43.7% of the time on the Array data set and 45% of the time on the Loop data set. This is a very good result, which means that our program can select more excellent parameters on most of the benchmarks.

4.7 Summary

After evaluation, we found that the decision tree model is better than the other three models in four important scores: Cross-validation score, Macro F1 score, Accuracy, and Macro recall. So we choose decision tree as our prediction model.

In addition, we also tested the generalization ability of the decision tree model on cross-subcategory data sets and its actual performance compared with the baseline. We found that the decision tree model has a certain generalization ability, and it can save nearly 45% of the verification time compared to the baseline.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

In this experiment, we constructed the feature vector by manually selecting features, combined with python's multi-process technology to use ESBMC to verify all the benchmarks of the three sub-category data sets of Array, Loop and Floats and recorded their verification results and time. In addition, a corresponding interface is provided to help users manage the data set visually. For machine learning, we compared the predictive abilities of the four models decision tree, KNN, SVM, and neural network on three sub-category data sets.

Summary of achievements As a result, we believe that the decision tree model with the shortest training time and excellent effect is the most suitable model. In order to test the generalization ability of the decision tree model across category data sets, we trained the model on the combined dataset of Array and Loop, then tested the model on the Floats dataset. The conclusion is that the decision tree model has a certain generalization ability and can be used as an extended research direction. In addition, we tested the effect of this system in actual scenarios, that is, comparing verification time of the ESBMC default parameters with the verification time of the system automatically selecting the optimal parameters, we found that when using the recommended parameters of this system, it can save nearly 45% of the total test time. This is a very good result, which proves that the system is effective and has practical value.

Reflection, identification of improvements For this experiment, we believe that there are deficiencies in four aspects and can be further improved:

1. Feature Engineering

   The feature analysis designed in this experiment is relatively intuitive and simple, and there are other feasible feature engineering schemas. For example, using Static Single Assignment form or Goto program as the internal representation of C source code instead of abstract syntax tree. In coming experiments, corresponding comparative experiments should be designed to demonstrate the advantages and disadvantages of different feature engineering schemas.

2. Neural Network

   As a current popular machine learning algorithm, the neural network has great potential. The neural network used in this article is a relatively basic feedforward neural network, and the structure is relatively simple. In the future, we can consider expanding the complexity of neural networks to improve the performance of this type of model.

3. The number of sub-category data sets

This experiment only conducts experiments on three sub-category data sets. However, the sub-categories included in SV-COMP are much more than that. Adding more sub-category data sets or even using all SV-COMP data sets is a potential solution to further evaluate the effectiveness of the model.

4. Model evaluation

   For model evaluation, this experiment can achieve a more refined evaluation by filtering out some useless classifications. In addition, we found that the data set of SV-COMP over the years have a certain overlap with the data set of SV-COMP in 2021, which may lead to certain over-fitting problems. In further plans, this situation should be avoided to ensure that the experimental results are more precise.

5.2 Future work

We believe that the work that can be continued in the future includes:

1. Expand the data set

   At present, we are working on three sub-category data sets from SV-COMP, but there are still a large number of other category data sets that have not been considered and discussed. We should complete the testing and discussion of all sub-category data sets in the future.

2. Verify the cross-problem generalization ability of the decision tree

   We proved that the decision tree has a certain generalization ability, but it is also limited to the three sub-category data sets that this article focuses on.

3. Optimize feature engineering

   The feature engineering scheme adopted in this paper is to select features artificially and does not take all the candidate features into consideration, so in the future, we should expand the selected feature set and add more features to test whether it can have a better effect. In addition, we should also consider selecting features automatically by using deep learning.

Bibliography

[1] B. Cui, J. Li, T. Guo, J. Wang, and D. Ma, "Code comparison system based on abstract syntax tree," in *2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, 2010, pp. 668–673.

[2] S. Falke, F. Merz, and C. Sinz, "The bounded model checker llbmc," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 706–709.

[3] D. Kroening and M. Tautschnig, "Cbmc–c bounded model checker," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 389–391.

[4] M. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole, "Esbmc 5.0: An industrial-strength c model checker," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 888–891. [Online]. Available: https://doi.org/10.1145/3238147.3240481

[5] D. Beyer, "Software verification: 10th comparative evaluation (sv-comp 2021)," *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 12652, p. 401, 2021.

[6] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[7] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Workshop on Logic of Programs*. Springer, 1981, pp. 52–71.

[8] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang, "Symbolic model checking: 1020 states and beyond," *Information and computation*, vol. 98, no. 2, pp. 142–170, 1992.

[9] O. Coudert and J. C. Madre, "A unified framework for the formal verification of sequential circuits," in *The Best of ICCAD*. Springer, 2003, pp. 39–50.

[10] K. L. McMillan, "Symbolic model checking," in *Symbolic Model Checking*. Springer, 1993, pp. 25–60.

[11] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," 2003.

[12] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 1999, pp. 193–207.

[13] M. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole, "Esbmc 5.0: an industrial-strength c model checker," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 888–891.

[14] J. Marques-Silva, "Practical applications of boolean satisfiability," in *2008 9th International Workshop on Discrete Event Systems*. IEEE, 2008, pp. 74–80.

[15] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[16] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani, "A sat based approach for solving formulas over boolean and linear mathematical propositions," in *International Conference on Automated Deduction*. Springer, 2002, pp. 195–210.

[17] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.

[18] L. De Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[19] ——, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[20] V. Ganesh and D. L. Dill, "A decision procedure for bit-vectors and arrays," in *International conference on computer aided verification*. Springer, 2007, pp. 519–531.

[21] B. Dutertre, "Yices 2.2," in *International Conference on Computer Aided Verification*. Springer, 2014, pp. 737–744.

[22] S. Conchon, A. Coquereau, M. Iguernlala, and A. Mebsout, "Alt-ergo 2.2," in *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, 2018.

[23] R. Brummayer and A. Biere, "Boolector: An efficient smt solver for bit-vectors and arrays," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2009, pp. 174–177.

[24] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani, "The mathsat 4 smt solver," in *International Conference on Computer Aided Verification*. Springer, 2008, pp. 299–303.

[25] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "Cvc4," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 171–177.

[26] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 783–794.

[27] I. Neamtiu, J. S. Foster, and M. Hicks, "Understanding source code evolution using abstract syntax tree matching," in *Proceedings of the 2005 international workshop on Mining software repositories*, 2005, pp. 1–5.

[28] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

[29] J. Stevenson, "The treatment of the long-term sequelae of child abuse," *The Journal of Child Psychology and Psychiatry and Allied Disciplines*, vol. 40, no. 1, pp. 89–111, 1999.

[30] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* IEEE, 2004, pp. 75–86.

[31] E. Clarke, D. Kroening, and F. Lerda, "A tool for checking ansi-c programs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2004, pp. 168–176.

[32] B. C. Lopes and R. Auler, *Getting started with LLVM core libraries*. Packt Publishing Ltd, 2014.

[33] T. O. Ayodele, "Machine learning overview," *New Advances in Machine Learning*, vol. 2, 2010.

[34] S. B. Kotsiantis, I. Zaharakis, P. Pintelas *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[35] S. O'Hara, Y. M. Lui, and B. A. Draper, "Unsupervised learning of human expressions, gestures, and actions," in *2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG)*. IEEE, 2011, pp. 1–8.

[36] H. B. Barlow, "Unsupervised learning," *Neural computation*, vol. 1, no. 3, pp. 295–311, 1989.

[37] J. Shlens, "A tutorial on principal component analysis," *arXiv preprint arXiv:1404.1100*, 2014.

[38] Z. Ghahramani, "Unsupervised learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 72–112.

[39] S. B. Thrun, "Efficient exploration in reinforcement learning," 1992.

[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[41] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[42] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, "An introduction to decision tree modeling," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, no. 6, pp. 275–285, 2004.

[43] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[44] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[45] M. Bramer, *Principles of data mining*. Springer, 2007, vol. 180.

[46] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[47] S. Suthaharan, "Support vector machine," in *Machine learning models and algorithms for big data classification*. Springer, 2016, pp. 207–235.

[48] J. A. Suykens, "Nonlinear modelling and support vector machines," in *IMTC 2001. proceedings of the 18th IEEE instrumentation and measurement technology conference. Rediscovering measurement in the age of informatics (Cat. No. 01CH 37188)*, vol. 1. IEEE, 2001, pp. 287–294.

[49] R. Schaback and H. Wendland, "Kernel techniques: from machine learning to meshless methods," *Acta numerica*, vol. 15, pp. 543–639, 2006.

[50] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[51] H. Parvin, H. Alizadeh, and B. Minaei-Bidgoli, "Mknn: Modified k-nearest neighbor," in *Proceedings of the world congress on engineering and computer science*, vol. 1. Citeseer, 2008.

[52] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, no. 1, pp. 273–314, 1997.

[53] M. W. Browne, "Cross-validation methods," *Journal of mathematical psychology*, vol. 44, no. 1, pp. 108–132, 2000.

[54] S.-C. Wang, "Artificial neural network," in *Interdisciplinary computing in java programming*. Springer, 2003, pp. 81–100.

[55] K. Gurney, *An introduction to neural networks*. CRC press, 2018.

[56] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[57] D. Jurafsky, *Speech & language processing*. Pearson Education India, 2000.

[58] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[59] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[60] S. Khirirat, H. R. Feyzmahdavian, and M. Johansson, "Mini-batch gradient descent: Faster convergence under data sparsity," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2880–2887.

[61] P. I. Good, *Resampling methods*. Springer, 2006.

[62] E. Bendersky, "Pycparser ast nodes," [EB/OL], https://github.com/eliben/pycparser/blob/master/pycparser/_c_ast.cfg Accessed August 1, 2021.

Appendix A

Code

A.1 Use ESBMC to generate verification results

```
1
#!/usr/bin/env python
3 # coding: utf-8

5 # In[1]:


7
import os
9 import time
import subprocess, datetime, signal
11 import multiprocessing
from multiprocessing import Process,Pool
13 from dbutils.pooled_db import PooledDB
from dbutils.persistent_db import PersistentDB
15 import random
import MySQLdb
17


19 # In[28]:


21
esbmc_dir = "/home/huajie.he/esbmc/bin/esbmc"
23 source_dir = "/home/huajie.he/sourceFile/loop/"
database_dir = "/home/huajie.he/esbmc.db"
25
cores = 30
27
# esbmc_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/
    ESBMC_Project/release/bin/esbmc"
29 # source_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/sourceFile/
    Array/"
# database_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/esbmc.db"
31
SMT = ["--z3", "--boolector", "--yices" , "--mathsat", "--cvc"]
33 Encode = ["--ir", "--fixedbv", "--floatbv"]
Verification = ["--k-induction", "--falsification", "--incremental-bmc"]
35
max_steps = [10,30,50,100,200,400,800,1600]
37 steps     = [ 1, 2, 4,  8, 16, 32, 64, 128]

39 VERIFICATION_FAILED = "VERIFICATION FAILED"
```

```
41 VERIFICATION_UNKNOWN = "VERIFICATION UNKNOWN"

43 VERIFICATION_SUCCESS = "VERIFICATION SUCCESS"

45 Timed_out = "Timed out"

47 ERROR = "ERROR"

49 #                                    Lucas
   status_code = ["unknown","error","timeout","success","VERIFICATION FAILED"]
51
   timeout_time = 15 * 60
53
   poolDB = PersistentDB(MySQLdb, host='localhost',user='root',passwd='ESbmc_2021',
       db='esbmc',port=3306)
55


57 # In[6]:


59
   def insert_db(sql):
61     conn = poolDB.connection()
       try:
63         c = conn.cursor()
           c.execute(sql)
65         conn.commit()
       except Exception as ex:
67         conn.rollback()
           print(ex)
69     conn.close()


71
   # In[7]:
73


75 def select_db(sql):
       conn = poolDB.connection()
77     try:
           c = conn.cursor()
79         c.execute(sql)
           conn.commit()
81         results = c.fetchall()
       except Exception as ex:
83         conn.rollback()
           print(ex)
85     conn.close()
       return results
87
```

```
89 # In[8]:

91
   def check_file_exist(fileName):
93     results = select_db("select * from programC p where p.fileName = '{}'".
           format(fileName))
       if len(results) == 1:
95         return False
       return True
97

99 # In[9]:

101
   def check_task_exist(command):
103    results = select_db("select * from task t where t.command = '{}'".format(
           command))
       if len(results) == 1:
105        return False
       return True
107

109 # In[10]:

111
   def insert_task_output(taskID,status,time,output):
113    insert_db("UPDATE task SET status = '{}', time = {}, output = '{}' where
           taskID = {}".format(status,time,output,taskID))

115
   # In[38]:
117

119 def create_programC():
       insert_db('''CREATE TABLE programC(
121    fileID INT auto_increment NOT NULL,
       fileName varchar(100) NOT NULL unique,
123    fileAddress varchar(300) NOT NULL,
       KEY(fileID),
125    CONSTRAINT programC_PK PRIMARY KEY (fileID))ENGINE=InnoDB DEFAULT CHARSET=
           utf8 COLLATE=utf8_general_ci;''')

127 def create_task():
       insert_db('''CREATE TABLE task(
129    taskID INT auto_increment NOT NULL,
       fileID INT NOT NULL,
131    command TEXT NOT NULL,
       solver varchar(50) NOT NULL,
```

```
133     encode varchar(50) NOT NULL,
        verificate varchar(50) NOT NULL,
135     max_k INT NOT NULL,
        k INT NOT NULL,
137     time DOUBLE,
        status varchar(100) NOT NULL,
139     output TEXT,
        KEY(taskID),
141     CONSTRAINT task_PK PRIMARY KEY (fileID,solver,encode,verificate,k))ENGINE=
            InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;''')

143 def create_base():
        insert_db('''CREATE TABLE base(
145     fileID INT NOT NULL,
        fileName varchar(100) NOT NULL unique,
147     fileAddress varchar(300) NOT NULL,
        time DOUBLE,
149     status varchar(100) NOT NULL,
        output TEXT,
151     PRIMARY KEY (fileID))ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=
            utf8_general_ci;''')


153
    # In[12]:
155


157 def get_all_file(file_dir):
        name_list = []
159     for root,dirs,files in os.walk(file_dir):
            for file in files:
161             if os.path.splitext(file)[1] == ".c":
                    name_list.append({"prefix":root,"name":os.path.splitext(file)
                        [0]+".c"})
163
        return name_list
165


167 # In[13]:


169
    # "unknown","error","timeout","success"
171 def file_to_tasks(file):
        cmd_list = []
173     for solver in range(len(SMT)):
            for encoder in range(len(Encode)):
175             for verificate in range(len(Verification)):
                    if SMT[solver]=="--boolector" and Encode[encoder]=="--ir" and (
                        Verification[verificate]=="--incremental-bmc" or Verification
                        [verificate]=="--k-induction"):
```

```
                    continue
                row = {"command":"","solver":"","encode":"","verificate":"","max
                    -k":0,"status":"",'k':0}
                row['status'] = "unknown"
#                       esbmc
                cmd = esbmc_dir + " "
#
                cmd += file['prefix'] + "/" + file['name'] + " "
#                     solver
                cmd += SMT[solver] + " "
                row["solver"] = SMT[solver]
#                      encode
                cmd += Encode[encoder] + " "
                row["encode"] = Encode[encoder]
                if Verification[verificate] == "--falsification":
                    cmd += Verification[verificate]
                    row["verificate"] = Verification[verificate]
                    row['max-k'] = 0
                    row['k'] = 0
                    row["command"] = cmd
                    cmd_list.append(row)
                else:
                    row = dict(row)
                    row["verificate"] = Verification[verificate]
                    row['max-k'] = max_steps[0]
                    row['k'] = steps[0]
                    row["command"] = cmd + Verification[verificate] + " --max-k-
                        step " + str(row['max-k'])+" --k-step " + str(row['k'])
                    cmd_list.append(row)
    return cmd_list
# len(file_to_tasks(get_all_file(source_dir)[0]))


# In[14]:


def produce_command(fileID,solver,encode,verificate,max_k,k):
    cmd = esbmc_dir + " "
    cmd += select_db("select p.fileAddress from programC p where p.fileID = {}".
        format(fileID))[0][0] + " "
    cmd += solver + " " + encode + " " + verificate + " "
    if verificate != "--falsification":
        index_k = 0
        for k in range(len(max_steps)):
            if max_steps[k] == max_k:
                index_k = k
        if index_k == len(max_steps)-1:
            return None,None,None
        else:
```

```
223             cmd += " --max-k-step " + str(max_steps[index_k+1])+" --k-step " +
                    str(steps[index_k+1])
                return cmd, max_steps[index_k+1], steps[index_k+1]
225     else:
            return cmd
227
def create_new_task(fileID,command,solver,encode,verificate,max_k,k):
229     if check_task_exist(command):
            insert_db("INSERT INTO task (fileID,command,solver,encode,verificate,
                max_k,k,status) VALUES ({},'{}','{}','{}','{}',{},{},'unknown')".
                format(fileID,command,solver,encode,verificate,max_k,k))
231
233
def create_new_file(filePrefix,fileName):
235     if check_file_exist(fileName):
            insert_db("INSERT INTO programC(fileName,fileAddress) VALUES ('{}',
                '{}')".format(fileName, filePrefix+"/"+fileName))
237
        results = select_db("SELECT fileID from programC where programC.fileName =
            '{}'".format(fileName))
239     fileID = results[0][0]
        return fileID
241
243
def get_unknown_taks():
245     results = select_db("SELECT t.taskID, t.command from task t where t.status='
            unknown'")
        return results
247
249
def file_manage(file_list):
251     for file in file_list:
            fileID = create_new_file(file['prefix'],file['name'])
253
255 def task_manage(file):
        fileID = create_new_file(file['prefix'],file['name'])
257     task_list = file_to_tasks(file)
        for task in task_list:
259         if check_task_exist(task['command']):
                create_new_task(fileID, task['command'], task['solver'], task['
                    encode'],task['verificate'],task['max-k'],task['k'])
261
263
```

```
265 def run_cmd(task):
267     task_id = task[0]
        cmd = task[1]
269
        cmd = cmd + " --timeout 15m"
271
        print("Task %d starts!"%(task_id))
273
        startTime  = time.time()
275     result = None
277     process = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=
            subprocess.PIPE)
279     while process.poll() is None:
            line = process.stdout.readline()
281         line = line.strip().decode()
            if Timed_out in line:
283             result = Timed_out
            if VERIFICATION_FAILED in line:
285             result = VERIFICATION_FAILED
            if VERIFICATION_SUCCESS in line:
287             result = VERIFICATION_SUCCESS
            if VERIFICATION_UNKNOWN in line:
289             result = VERIFICATION_UNKNOWN
291     runtime = time.time()-startTime
293     if Timed_out == result or runtime>888:
            insert_task_output(task_id,Timed_out,runtime,"")
295         print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,Timed_out))
            return
297
        if VERIFICATION_FAILED == result:
299         insert_task_output(task_id,VERIFICATION_FAILED,runtime,"")
            print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,
                VERIFICATION_FAILED))
301         return
303     if VERIFICATION_SUCCESS == result:
            insert_task_output(task_id,VERIFICATION_SUCCESS,runtime,"")
305         print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,
                VERIFICATION_SUCCESS))
            return
307
        if VERIFICATION_UNKNOWN == result:
309         insert_task_output(task_id,VERIFICATION_UNKNOWN,runtime,"")
```

```
            print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,
                VERIFICATION_UNKNOWN))
311         _task = select_db("select fileID, solver,encode,verificate,max_k,k from
                task t where t.taskID={}".format(task_id))[0]
            if _task[4] == 0:
313             return
            else:
315             new_command, new_max_k, new_k = produce_command(_task[0],_task[1],
                    _task[2],_task[3],_task[4],_task[5])
                if new_command == None:
317                 return
                else:
319                 create_new_task(_task[0],new_command,_task[1],_task[2],_task[3],
                        new_max_k,new_k)
                    new_task_id = select_db("SELECT taskID from task where task.
                        command = '{}'".format(new_command))[0][0]
321                 run_cmd([new_task_id,new_command])
                    return
323
        insert_task_output(task_id,ERROR,runtime,"")
325     print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,ERROR))
        return
327
329
    def insert_base_output(taskID,status,time,output):
331     insert_db("UPDATE base SET status = '{}', time = {}, output = '{}' where
            fileID = {}".format(status,time,output,taskID))
333
335 def run_cmd_base(fileID, cmd):
337     task_id = fileID
339     cmd = cmd + " --timeout 15m"
341     print(cmd)
343     print("Task %d starts!"%(task_id))
345     startTime  = time.time()
        result = None
347
        process = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=
            subprocess.PIPE)
349
        while process.poll() is None:
351         line = process.stdout.readline()
```

```
        line = line.strip().decode()
353     if Timed_out in line:
            result = Timed_out
355     if VERIFICATION_FAILED in line:
            result = VERIFICATION_FAILED
357     if VERIFICATION_SUCCESS in line:
            result = VERIFICATION_SUCCESS
359     if VERIFICATION_UNKNOWN in line:
            result = VERIFICATION_UNKNOWN
361
    runtime = time.time()-startTime
363
    if Timed_out == result or runtime>898:
365     insert_base_output(task_id,Timed_out,runtime,"")
        print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,Timed_out))
367     return

369 if VERIFICATION_FAILED == result:
        insert_base_output(task_id,VERIFICATION_FAILED,runtime,"")
371     print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,
            VERIFICATION_FAILED))
        return
373
    if VERIFICATION_SUCCESS == result:
375     insert_base_output(task_id,VERIFICATION_SUCCESS,runtime,"")
        print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,
            VERIFICATION_SUCCESS))
377     return

379 if VERIFICATION_UNKNOWN == result:
        insert_base_output(task_id,VERIFICATION_UNKNOWN,runtime,"")
381     print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,
            VERIFICATION_UNKNOWN))
        return
383
    insert_base_output(task_id,ERROR,runtime,"")
385 print("Task %d finished, %f, %s!!!!!!"%(task_id,runtime,ERROR))
    return
387



389
def main():
391 print(cores)

393 create_programC()
    create_task()
395
    st = time.time()
397
```

```
    file_list = get_all_file(source_dir)
    pool = Pool(cores)
    print("We have %d files"%len(file_list))

    for file in file_list[:10]:
        pool.apply_async(func=task_manage, args=(file,))

    for file in file_list:
        pool.apply_async(func=task_manage, args=(file,))

    pool.close()
    pool.join()

    print(time.time()-st)
    print("Already got all tasks, ready to run!!!")

    pool = Pool(cores)

    task_pool = get_unknown_taks()

    print("Starting esbmc")
    for i in task_pool:
        pool.apply_async(func=run_cmd, args=(i,))

    pool.close()
    pool.join()

def baseline():
    create_base()
    sql = "select * from programC"
    programs = select_db(sql)
    for program in programs:
        fileID = program[0]
        fileName = program[1]
        fileAddress = program[2]
        insert_db("INSERT INTO base (fileID,fileName,fileAddress,status) VALUES
            ({},'{}','{}','unknown')".format(fileID,fileName,fileAddress))
#         run_cmd_base(fileID, esbmc_dir+" "+ fileAddress + " ")

    pool = Pool(cores)
    for program in programs:
        fileID = program[0]
        fileName = program[1]
        fileAddress = program[2]
        cmd = esbmc_dir + " " + fileAddress
        pool.apply_async(func=run_cmd_base, args=(fileID,cmd + " ",))

    pool.close()
    pool.join()
```

```
447 if __name__=='__main__':
        main()
449     baseline()
```

A.2 Feature extraction

```
2  from __future__ import print_function

4  import json
   import sys
6  import re

8  # This is not required if you've installed pycparser into
   # your site-packages/ with setup.py
10 #
   sys.path.extend(['.', '..'])
12
   from pycparser import parse_file, c_ast
14 from pycparser.plyparser import Coord


16
   RE_CHILD_ARRAY = re.compile(r'(.*)\[(.*)\]')
18 RE_INTERNAL_ATTR = re.compile('__.*__')


20
   class CJsonError(Exception):
22     pass


24
   def memodict(fn):
26     """ Fast memoization decorator for a function taking a single argument """
       class memodict(dict):
28         def __missing__(self, key):
               ret = self[key] = fn(key)
30             return ret
       return memodict().__getitem__
32


34 @memodict
   def child_attrs_of(klass):
36     """
       Given a Node class, get a set of child attrs.
38     Memoized to avoid highly repetitive string manipulation
```

```
    """
    non_child_attrs = set(klass.attr_names)
    all_attrs = set([i for i in klass.__slots__ if not RE_INTERNAL_ATTR.match(i)
        ])
    return all_attrs - non_child_attrs


def to_dict(node):
    """ Recursively convert an ast into dict representation. """
    klass = node.__class__

    result = {}

    # Metadata
    result['_nodetype'] = klass.__name__

    # Local node attributes
    for attr in klass.attr_names:
        result[attr] = getattr(node, attr)

    # Coord object
    if node.coord:
        result['coord'] = str(node.coord)
    else:
        result['coord'] = None

    # Child attributes
    for child_name, child in node.children():
        # Child strings are either simple (e.g. 'value') or arrays (e.g. '
            block_items[1]')
        match = RE_CHILD_ARRAY.match(child_name)
        if match:
            array_name, array_index = match.groups()
            array_index = int(array_index)
            # arrays come in order, so we verify and append.
            result[array_name] = result.get(array_name, [])
            if array_index != len(result[array_name]):
                raise CJsonError('Internal ast error. Array {} out of order. '
                    'Expected index {}, got {}'.format(
                    array_name, len(result[array_name]), array_index))
            result[array_name].append(to_dict(child))
        else:
            result[child_name] = to_dict(child)

    # Any child attributes that were missing need "None" values in the json.
    for child_attr in child_attrs_of(klass):
        if child_attr not in result:
            result[child_attr] = None
```

```
86      return result
88
def to_json(node, **kwargs):
90      """ Convert ast node to json string """
        return json.dumps(to_dict(node), **kwargs)
92
94  def file_to_dict(filename):
        """ Load C file into dict representation of ast """
96  #       ast = parse_file(filename, use_cpp=True)
        ast = parse_file(filename, use_cpp=True,
98                  cpp_path='gcc',
                    cpp_args=['-E', r'-I/Users/grassgod/opt/anaconda3/envs/bmc/lib/
                        python3.7/site-packages/pycparser/utils/fake_libc_include'])
100     return to_dict(ast)
102
def file_to_json(filename, **kwargs):
104     """ Load C file into json string representation of ast """
        ast = parse_file(filename, use_cpp=True)
106     return to_json(ast, **kwargs)
108
def _parse_coord(coord_str):
110     """ Parse coord string (file:line[:column]) into Coord object. """
        if coord_str is None:
112         return None

114     vals = coord_str.split(':')
        vals.extend([None] * 3)
116     filename, line, column = vals[:3]
        return Coord(filename, line, column)
118

120 def _convert_to_obj(value):
        """
122     Convert an object in the dict representation into an object.
        Note: Mutually recursive with from_dict.
124     """
        value_type = type(value)
126     if value_type == dict:
            return from_dict(value)
128     elif value_type == list:
            return [_convert_to_obj(item) for item in value]
130     else:
            # String
132         return value
```

```
134
def from_dict(node_dict):
136     """ Recursively build an ast from dict representation """
    class_name = node_dict.pop('_nodetype')
138
    klass = getattr(c_ast, class_name)
140
    # Create a new dict containing the key-value pairs which we can pass
142     # to node constructors.
    objs = {}
144     for key, value in node_dict.items():
        if key == 'coord':
146             objs[key] = _parse_coord(value)
        else:
148             objs[key] = _convert_to_obj(value)

150     # Use keyword parameters, which works thanks to beautifully consistent
    # ast Node initializers.
152     return klass(**objs)


154
def from_json(ast_json):
156     """ Build an ast from json string representation """
    return from_dict(json.loads(ast_json))
```

A.3 Feature analysis

```
2 const_feature_name = ['Decl','Loop','Recursion','If','Assignment','FuncCall','
    Label','Constant','TypeDecl','IdentifierType','Goto','solver','encode','
    verificator','max_k','class']
feature_list = {i:0 for i in const_feature_name}
4
def get_object_name(obj):
6     return obj.__class__.__name__

8 def for_node(nodes,father):
    for node in nodes:
10         extract_feature_vec(node,father)

12 def FileAST(ast_node,father):
    for_node(ast_node.ext,father)
14
def FuncDef(ast_node,father):
16     for decl in ast_node.decl:
```

```
            Decl(decl,ast_node.decl.name)
    if ast_node.param_decls:
        for decl in ast_node.param_decls:
            Decl(decl,ast_node.decl.name)
    for_node(ast_node.body,ast_node.decl.name)

def FuncCall(ast_node,father):
    if ast_node.name.name==father:
        feature_list['Recursion']+=1
    feature_list['FuncCall']+=1

def Decl(ast_node,father):
    feature_list['Decl']+=1

def If(ast_node,father):
    feature_list['If']+=1
    if ast_node.iftrue:
        for_node(ast_node.iftrue,father)
    if ast_node.iffalse:
        for_node(ast_node.iffalse,father)

def Label(ast_node,father):
    feature_list['Label']+=1
    if ast_node.stmt:
        for_node(ast_node.stmt,father)

def For(ast_node,father):
    feature_list['Loop']+=1
    if ast_node.stmt:
        for_node(ast_node.stmt,father)

def Compound(ast_node,father):
    if ast_node.block_items:
        for_node(ast_node.block_items,father)

def Assignment(ast_node,father):
    feature_list['Assignment']+=1
    if ast_node.lvalue:
        for_node(ast_node.lvalue,father)
    if ast_node.rvalue:
        for_node(ast_node.rvalue,father)

def Return(ast_node,father):
    feature_list['Return']+=1

def Constant(ast_node,father):
    feature_list['Constant']+=1

def Typename(ast_node,father):
```

```
66      if ast_node.type:
            for_node(ast_node.type,father)
68
    def TypeDecl(ast_node,father):
70      feature_list['TypeDecl']+=1
        if ast_node.type:
72          for_node(ast_node.type,father)

74  def Typedef(ast_node,father):
        if ast_node.type:
76          for_node(ast_node.type,father)

78  def While(ast_node,father):
        feature_list['Loop']+=1
80      if ast_node.stmt:
            for_node(ast_node.stmt,father)
82
    def Cast(ast_node,father):
84      if ast_node.to_type:
            for_node(ast_node.to_type,father)
86      if ast_node.expr:
            for_node(ast_node.expr,father)
88
    def IdentifierType(ast_node,father):
90      feature_list['IdentifierType']+=1

92  def Goto(ast_node,father):
        feature_list['Goto']+=1
94
    def DoWhile(ast_node,father):
96      feature_list['Loop']+=1
        if ast_node.stmt:
98          for_node(ast_node.stmt,father)

100 def Switch(ast_node,father):
        if ast_node.stmt:
102         for_node(ast_node.stmt,father)

104 def Case(ast_node,father):
        feature_list['If']+=1
106     if ast_node.stmts:
            for_node(ast_node.stmts,father)
108


110 def extract_feature_vec(ast_node,father):

112     global feature_list
        """ Extract feature from ast """
114     node_name = get_object_name(ast_node)
```

```
    if node_name == 'FileAST':
116         FileAST(ast_node, father)
            return
118
    if node_name == 'Decl':
120         Decl(ast_node, father)
            return
122
    if node_name == 'FuncDef':
124         FuncDef(ast_node, father)
            return
126
    if node_name == "If":
128         If(ast_node, father)
            return
130
    if node_name == "Label":
132         Label(ast_node, father)
            return
134
    if node_name == "For":
136         For(ast_node, father)
            return
138
    if node_name == "Compound":
140         Compound(ast_node, father)
            return
142
    if node_name == "Assignment":
144         Assignment(ast_node, father)
            return
146
    if node_name == "ID":
148         return

150 if node_name == "FuncCall":
            FuncCall(ast_node, father)
152         return

154 if node_name == "Return":
#           Return(ast_node, father)
156         return

158 if node_name == "BinaryOp":
            return
160
    if node_name == "UnaryOp":
162         return
```

```
164     if node_name == "ArrayRef":
            return
166
        if node_name == "Constant":
168         Constant(ast_node, father)
            return
170
        if node_name == "Typename":
172         Typename(ast_node, father)
            return
174
        if node_name == "While":
176         While(ast_node, father)
            return
178
        if node_name == "Cast":
180         Cast(ast_node, father)
            return
182
        if node_name == "TypeDecl":
184         TypeDecl(ast_node, father)
            return
186
        if node_name == "IdentifierType":
188         IdentifierType(ast_node, father)
            return
190
        if node_name == 'ExprList':
192         return

194     if node_name == 'Goto':
            Goto(ast_node, father)
196         return

198     if node_name == 'Typedef':
            Typedef(ast_node, father)
200         return

202     if node_name == 'Break':
            return
204
        if node_name == 'DoWhile':
206         DoWhile(ast_node, father)
            return
208
        if node_name == 'Switch':
210         Switch(ast_node, father)
            return
212
```

```
    if node_name == 'Case':
214         Case(ast_node,father)
            return
216
#       print(get_object_name(ast_node))
218
def extract_feature_func(ast):
220     global feature_list
        feature_list = {i:0 for i in const_feature_name}
222     extract_feature_vec(ast,None)
        return feature_list
```

A.4 Model training and evaluation

```
1
#!/usr/bin/env python
3 # coding: utf-8

5 # In[1]:


7
from dbutils.pooled_db import PooledDB
9 from dbutils.persistent_db import PersistentDB
import MySQLdb
11
import os
13 import pandas as pd
import re
15
from sklearn import svm
17 from sklearn.model_selection import GridSearchCV
import numpy as np
19 from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
21 from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
23 from pycm import *

25 from sklearn.neural_network import MLPClassifier


27
# ### prepare db connection
29
# In[2]:
31
```

```
array_poolDB = PersistentDB(MySQLdb, host='localhost',user='root',passwd='
    ESbmc_2021',db='esbmc_array',port=3306)
loop_poolDB = PersistentDB(MySQLdb, host='localhost',user='root',passwd='
    ESbmc_2021',db='esbmc_loop',port=3306)
float_poolDB = PersistentDB(MySQLdb, host='localhost',user='root',passwd='
    ESbmc_2021',db='esbmc_floats',port=3306)


# ### db_utils

# In[3]:


def insert_db(poolDb, sql):
    conn = poolDB.connection()
    try:
        c = conn.cursor()
        c.execute(sql)
        conn.commit()
    except Exception as ex:
        conn.rollback()
        print(ex)
    conn.close()


# In[4]:


def select_db(poolDB, sql):
    conn = poolDB.connection()
    try:
        c = conn.cursor()
        c.execute(sql)
        conn.commit()
        results = c.fetchall()
    except Exception as ex:
        conn.rollback()
        print(ex)
    conn.close()
    return results


# ### set file location

# In[5]:


def search_file(file_dir, file_name):
```

```
    for root,dirs,files in os.walk(file_dir):
        for file in files:
            if file == file_name:
                return True
    return False


# In[6]:


def get_all_fileNames(poolDB):
    files = []
    items = select_db(poolDB, "select * from programC")
    for file in items:
        files.append(file[1])
    return files


# In[7]:


def split_files(file_dir, fileList):
    svcomp21_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
        evaluation/dataset/DataSource/svcomp21/"
    train = []
    test = []
    for file in fileList:
        if search_file(file_dir, file):
            test.append(file)
        else:
            train.append(file)
    return train,test


# #### all files

# In[8]:


forbidden_list = []

array_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
    evaluation/dataset/array/"
loop_dir  = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
    evaluation/dataset/loop/"
float_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
    evaluation/dataset/float/"

#          arrayfiles
```

```
123 array_files = get_all_fileNames(array_poolDB)

125 #           loopfiles
    loop_files = get_all_fileNames(loop_poolDB)
127
    #           floatfiles
129 float_files = get_all_fileNames(float_poolDB)


131
    # In[9]:
133


135 print(len(array_files))


137
    # #### split train files and test files
139
    # In[10]:
141


143 data_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/evaluation
        /dataset/DataSource/"

145 array_svcomp21_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
        evaluation/dataset/svcomp21/array/"
    loop_svcomp21_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
        evaluation/dataset/svcomp21/loop/"
147 float_svcomp21_dir = "/Users/grassgod/Documents/Manchester/MSCProject/2021/Code/
        evaluation/dataset/svcomp21/float/"

149 #             train

151 train_array_files, test_array_files = split_files(array_svcomp21_dir,
        array_files)

153 train_loop_files, test_loop_files = split_files(loop_svcomp21_dir, loop_files)

155 train_float_files, test_float_files = split_files(float_svcomp21_dir,
        float_files)


157
    # ### 1. extract_features
159
    # In[11]:
161


163 def alert_file(file_address):
        with open(file_address, 'r', encoding='utf-8') as f:
165         c = f.read()
```

```
    with open(file_address, 'r', encoding='utf-8') as f:
        c_line = f.readlines()
        if "__attribute__" in c and "#define __attribute__(x)" not in c:
            c_line.insert(0,"#define __attribute__(x)\n")

        if "__extension__" in c and "#define __extension__" not in c:
            c_line.insert(0,"#define __extension__\n")

#         if "size_t" in c and "typedef unsigned long size_t;" not in c:
#             c_line.insert(0,"typedef unsigned long size_t;\n")

    comments = False
    for line in range(len(c_line)):
        if "/*" in c_line[line] and "*/" not in c_line[line]:
            comments = True
        if "/*" not in c_line[line] and "*/" in c_line[line]:
            comments = False
        if "__attribute__" in c_line[line]:
            c_line[line] = re.sub("__attribute__\s+","__attribute__",c_line[line
                ])
#         if "#include" in c_line[line] and "/*" not in c_line[line]:
#             c_line[line] =  "/* " + re.sub("\n","",c_line[line]) + " */\n"
        if "//" in c_line[line] and "/*" not in c_line[line] and "http" not in
            c_line[line] and not comments:
            index = c_line[line].find("//")
            c_line[line] =  c_line[line][:index] + "\n/* " + c_line[line][index
                :] + "*/\n"
        if c_line[line].strip().startswith("//") and not comments:
            c_line[line] =  "/* " + re.sub("\n","",c_line[line]) + " */\n"

    with open(file_address, 'w', encoding='utf-8') as f:
        f.writelines(c_line)


# In[12]:


const_feature_name = ['Decl','Loop','Recursion','If','Assignment','FuncCall','
    Label','Constant','TypeDecl','IdentifierType','Goto','solver','encode','
    verificator','max_k','class']
feature_list = {i:0 for i in const_feature_name}


# In[13]:


def get_file_address(file_dir,file_name):
    for root,dirs,files in os.walk(file_dir):
```

```
        for file in files:
            if file == file_name:
                return root+"/"+file


# In[14]:


get_ipython().run_line_magic('run', 'extract_feature.py')
get_ipython().run_line_magic('run', 'analysis_feature.py')


# In[15]:


from imp import reload
import extract_feature
reload(extract_feature)

import analysis_feature
reload(analysis_feature)


# In[16]:


def get_features(file_address):
    feature_list = {i:0 for i in const_feature_name}
    ast_dict = file_to_dict(file_address)
    ast = from_dict(ast_dict)

    return extract_feature_func(ast)


# In[17]:


forbidden_list = []


# In[18]:


def get_dateset_features(file_dir,name_list):
    feature_set = {}
    for file in name_list:
        if file in forbidden_list:
            continue
        file_address = get_file_address(file_dir,file)
```

```
#           alert_file(file_address)
        try:
            feature_list = get_features(file_address)
            feature_set[file] = feature_list
        except Exception as ex:
            print(ex)
            forbidden_list.append(file)
    return feature_set


# In[19]:


array_feature = get_dateset_features(data_dir,array_files)


# In[20]:


loop_feature = get_dateset_features(data_dir, loop_files)


# In[21]:


float_feature = get_dateset_features(data_dir, float_files)


# ### 2 format feature csv

# In[22]:


SMT = ["--z3", "--boolector", "--yices" , "--mathsat", "--cvc"]
Encode = ["--ir", "--fixedbv", "--floatbv"]
Verification = ["--k-induction", "--falsification", "--incremental-bmc"]

max_steps = [10,30,50,100,200,400,800,1600]
steps     = [ 1, 2, 4,  8, 16, 32, 64, 128]

VERIFICATION_FAILED = "VERIFICATION FAILED"

VERIFICATION_UNKNOWN = "VERIFICATION UNKNOWN"

VERIFICATION_SUCCESS = "VERIFICATION SUCCESS"

Timed_out = "Timed out"

ERROR = "ERROR"
```

```
309
# In[23]:
311

313 def identify_status(time, status):
    if status == ERROR:
315     return 6

317 if status == VERIFICATION_UNKNOWN:
        return 5
319
    if status == Timed_out:
321     return 4

323 if status == VERIFICATION_FAILED or status == VERIFICATION_SUCCESS:
        if time<2:
325         return 0
        if time<60:
327         return 1
        if time<300:
329         return 2
        if time<900:
331         return 3


333
# In[24]:
335

337 def identify_parameters(parm):
    SMT = ["--z3", "--boolector", "--yices" , "--mathsat", "--cvc"]
339 Encode = ["--ir", "--fixedbv", "--floatbv"]
    Verification = ["--k-induction", "--falsification", "--incremental-bmc"]
341 for i in range(len(SMT)):
        if SMT[i] == parm:
343         return i
    for i in range(len(Encode)):
345     if Encode[i] == parm:
            return i
347 for i in range(len(SMT)):
        if Verification[i] == parm:
349         return i


351
# In[25]:
353

355 def process_feature(poolDB, feature_base, solver, encode, verificate, max_k,
    file):
```

```
    feature_list = feature_base

    fileID = select_db(poolDB,"select * from programC where fileName = '{}'".
        format(file))[0][0]
    task = select_db(poolDB, "select * from task where fileID = {} and solver =
        '{}' and encode = '{}' and verificate = '{}' and max_k = {}".format(
        fileID,solver,encode,verificate,max_k))
    if len(task) == 0:
        return False

    feature_list['solver'] = identify_parameters(solver)
    feature_list['encode'] = identify_parameters(encode)
    feature_list['verificator'] = identify_parameters(verificate)
    feature_list['max_k'] = max_k

    feature_list['class'] = identify_status(task[0][8], task[0][9])

    return feature_list


# In[26]:


def roll_back(poolDB, feature_base, solver, encode, verificate, max_k, file):
    feature_list = feature_base

    fileID = select_db(poolDB,"select * from programC where fileName = '{}'".
        format(file))[0][0]

    for k in range(7,-1,-1):
        temp_k = max_steps[k]

        if temp_k>= max_k:
            continue
        else:
            task = select_db(poolDB, "select * from task where fileID = {} and
                solver = '{}' and encode = '{}' and verificate = '{}' and max_k =
                {}".format(fileID,solver,encode,verificate,temp_k))
            if len(task) == 0:
                continue

            feature_list['solver'] = identify_parameters(solver)
            feature_list['encode'] = identify_parameters(encode)
            feature_list['verificator'] = identify_parameters(verificate)
            feature_list['max_k'] = max_k

            feature_list['class'] = identify_status(task[0][8], task[0][9])
            return feature_list
```

```
399
# In[27]:
401

403 const_feature_name = ['Decl','Loop','Recursion','If','Assignment','FuncCall','
    Label','Constant','TypeDecl','IdentifierType','Goto','solver','encode','
    verificator','max_k','class']
def get_all_feature(poolDB, feature_set, file_list):
405     df = pd.DataFrame(columns = const_feature_name)
        for file in file_list:
407         if file in forbidden_list:
                continue
409         for smt in SMT:
                for encode in Encode:
411                 for verificate in Verification:
                        if smt=="--boolector" and encode=="--ir" and (verificate=="
                            --incremental-bmc" or verificate=="--k-induction"):
413                         continue
                        if verificate == '--falsification':
415                         feature_base = feature_set[file]

417                         feature_list = process_feature(poolDB, feature_base, smt
                                , encode, verificate, 0, file)

419                         feature_list =  pd.Series(feature_list, name=file+"_"+
                                str(identify_parameters(smt))+"_"+str(
                                identify_parameters(encode))+"_"+str(
                                identify_parameters(verificate))+"_"+str(0))

421                         df = df.append(feature_list)
                        else:
423                         for k in range(8):
                                feature_base = feature_set[file]
425
                                feature_list = process_feature(poolDB, feature_base,
                                    smt, encode, verificate, max_steps[k], file)
427
                                if feature_list == False:
429                                 feature_list = roll_back(poolDB, feature_base,
                                        smt, encode, verificate, max_steps[k], file)

431                             feature_list =  pd.Series(feature_list, name=file+"_
                                    "+str(identify_parameters(smt))+"_"+str(
                                    identify_parameters(encode))+"_"+str(
                                    identify_parameters(verificate))+"_"+str(
                                    max_steps[k]))

433                             df = df.append(feature_list)
    return df
```

```
435
437 # In[28]:
439
# df_train_array = get_all_feature(array_poolDB, array_feature,
    train_array_files)
441 # df_train_array.to_csv("df_train_array.csv")
443 # df_test_array = get_all_feature(array_poolDB, array_feature, test_array_files)
# df_test_array.to_csv("df_test_array.csv")
445
447 # In[29]:
449
# df_train_loop = get_all_feature(loop_poolDB, loop_feature, train_loop_files)
451 # df_train_loop.to_csv("df_train_loop.csv")
453 # df_test_loop = get_all_feature(loop_poolDB, loop_feature, test_loop_files)
# df_test_loop.to_csv("df_test_loop.csv")
455
457 # In[30]:
459
# df_train_float = get_all_feature(float_poolDB, float_feature,
    train_float_files)
461 # df_train_float.to_csv("df_train_float.csv")
463 # df_test_float = get_all_feature(float_poolDB, float_feature, test_float_files)
# df_test_float.to_csv("df_test_float.csv")
465
467 # In[31]:
469
df_train_array = pd.read_csv("df_train_array.csv", index_col=0)
471
df_test_array = pd.read_csv("df_test_array.csv", index_col=0)
473
df_train_loop = pd.read_csv("df_train_loop.csv", index_col=0)
475
df_test_loop = pd.read_csv("df_test_loop.csv", index_col=0)
477
df_train_float = pd.read_csv("df_train_float.csv", index_col=0)
479
df_test_float = pd.read_csv("df_test_float.csv", index_col=0)
481
```

```
# ### 2 Evaluation

# ### 2.1 build model

# In[43]:


def SVM_model(train_data,train_label,test_data, test_label):
    # SVM
    clf = svm.SVC()

    param_grid = {'C': [0.1,1,10,100]}
    grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
    grid.fit(train_data, train_label)

    print('-----------------------------------------------------------')
    print('SVM:')
    print('Best cross-validation score:', grid.best_score_)
    print('Best parameters:', grid.best_params_)
    clf = grid.best_estimator_
    y_pred = clf.predict(test_data)

    cm = ConfusionMatrix(actual_vector=test_label.values, predict_vector=y_pred)
    cm2 = confusion_matrix(test_label,y_pred)
    print(cm)
    print(classification_report(test_label,y_pred,target_names=["Really fast","
        Fast","Normal","Slow","Timed out","Unknown","ERROR"],digits=2))
    print("----------------")

    print("accuracy score", accuracy_score(test_label,y_pred))
    print("correct number", accuracy_score(test_label,y_pred,normalize=False))
    print("-------------------")


    print("micro_f1",f1_score(test_label,y_pred,average='micro'))
    print("macro_f1",f1_score(test_label,y_pred,average='macro'))
    print("weighted_f1",f1_score(test_label,y_pred,average='weighted'))

    print("micro_recall",recall_score(test_label,y_pred,average='micro'))
    print("macro_recall",recall_score(test_label,y_pred,average='macro'))
    print("weighted_recall",recall_score(test_label,y_pred,average='weighted'))


    return clf


# In[33]:

```

```
531 def DT_model(train_data,train_label,test_data, test_label):
    clf = DecisionTreeClassifier()
533
    param_grid = {'criterion': ['gini', 'entropy'],
535                  'splitter': ['best', 'random'],
                     'max_features': ['auto', 'sqrt', 'log2']}
537 grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
    grid.fit(train_data, train_label)
539
    print('-----------------------------------------------------------')
541 print('DT:')
    print('Best cross-validation score:', grid.best_score_)
543 print('Best parameters:', grid.best_params_)

545 clf = grid.best_estimator_
    y_pred = clf.predict(test_data)
547
    cm = ConfusionMatrix(actual_vector=test_label.values, predict_vector=y_pred)
549 cm2 = confusion_matrix(test_label,y_pred)
    print(cm)
551 print(classification_report(test_label,y_pred,target_names=["Really fast","
        Fast","Normal","Slow","Timed out","Unknown","ERROR"],digits=2))

553 print("----------------")

555 print("accuracy score", accuracy_score(test_label,y_pred))

557 print("correct number", accuracy_score(test_label,y_pred,normalize=False))
    print("-------------------")
559

561 print("micro_f1",f1_score(test_label,y_pred,average='micro'))
    print("macro_f1",f1_score(test_label,y_pred,average='macro'))
563 print("weighted_f1",f1_score(test_label,y_pred,average='weighted'))

565 print("micro_recall",recall_score(test_label,y_pred,average='micro'))
    print("macro_recall",recall_score(test_label,y_pred,average='macro'))
567 print("weighted_recall",recall_score(test_label,y_pred,average='weighted'))


569
#     print(cm)
571
    return clf
573

575 # In[34]:

577
```

```
def KNN_model(train_data,train_label,test_data, test_label):
    # KNN
    clf = KNeighborsClassifier()

    param_grid = {'n_neighbors': [1, 2, 3, 4, 5],
                  'weights' : ['uniform', 'distance']}
    grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
    grid.fit(train_data, train_label)

    print('-----------------------------------------------------------')
    print('KNN:')
    print('Best cross-validation score:', grid.best_score_)
    print('Best parameters:', grid.best_params_)
    clf = grid.best_estimator_
    y_pred = clf.predict(test_data)

    cm = ConfusionMatrix(actual_vector=test_label.values, predict_vector=y_pred)
    cm2 = confusion_matrix(test_label,y_pred)
    print(cm)

    print(classification_report(test_label,y_pred,target_names=["Really fast","
        Fast","Normal","Slow","Timed out","Unknown","ERROR"],digits=2))

    print("----------------")

    print("accuracy score", accuracy_score(test_label,y_pred))

    print("correct number", accuracy_score(test_label,y_pred,normalize=False))
    print("-------------------")

    print("micro_f1",f1_score(test_label,y_pred,average='micro'))
    print("macro_f1",f1_score(test_label,y_pred,average='macro'))
    print("weighted_f1",f1_score(test_label,y_pred,average='weighted'))

    print("micro_recall",recall_score(test_label,y_pred,average='micro'))
    print("macro_recall",recall_score(test_label,y_pred,average='macro'))
    print("weighted_recall",recall_score(test_label,y_pred,average='weighted'))

    return clf


# In[35]:


def NN_model(train_data,train_label,test_data, test_label):
#     neural network
    clf = MLPClassifier(alpha=1e-5, max_iter = 1000)

    param_grid = {'learning_rate_init' : [0.01,0.02,0.05,0.1,0.2,0.5]}
```

```
    grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
627 grid.fit(train_data, train_label)

629 print('-----------------------------------------------------------')
    print('NN:')
631 print('Best cross-validation score:', grid.best_score_)
    print('Best parameters:', grid.best_params_)
633 clf = grid.best_estimator_
    y_pred = clf.predict(test_data)
635
    cm = ConfusionMatrix(actual_vector=test_label.values, predict_vector=y_pred)
637 cm2 = confusion_matrix(test_label,y_pred)
    print(cm)
639 print(classification_report(test_label,y_pred,target_names=["Really fast","
        Fast","Normal","Slow","Timed out","Unknown","ERROR"],digits=2))

641 print("----------------")

643 print("accuracy score", accuracy_score(test_label,y_pred))

645 print("correct number", accuracy_score(test_label,y_pred,normalize=False))
    print("-------------------")
647


649 print("micro_f1",f1_score(test_label,y_pred,average='micro'))
    print("macro_f1",f1_score(test_label,y_pred,average='macro'))
651 print("weighted_f1",f1_score(test_label,y_pred,average='weighted'))

653 print("micro_recall",recall_score(test_label,y_pred,average='micro'))
    print("macro_recall",recall_score(test_label,y_pred,average='macro'))
655 print("weighted_recall",recall_score(test_label,y_pred,average='weighted'))

657 return clf


659
# ### prepare the data
661
# In[36]:
663


665 def balance_data(df_train, df_test, frac):
    temp = df_test.sample(frac=0.7)
667 train_data = pd.concat([df_train,temp],axis = 0)
    test_data = df_test[~df_test.index.isin(temp.index)]
669
    train_label = train_data['class']
671 test_label = test_data['class']

673 train_data = train_data.drop(columns=['class'])
```

```
    test_data = test_data.drop(columns=['class'])

    train_label = train_label.astype('int')
    test_label = test_label.astype('int')

    return train_data, test_data, train_label, test_label


# In[37]:


train_data_array, test_data_array, train_label_array, test_label_array =
    balance_data(df_train_array, df_test_array, 0.2)

train_data_loop, test_data_loop, train_label_loop, test_label_loop =
    balance_data(df_train_loop, df_test_loop, 0.2)

train_data_float, test_data_float, train_label_float, test_label_float =
    balance_data(df_train_float, df_test_float, 0.4)


# ### 2.2 Test model individually

# In[38]:


import time
from sklearn.metrics import confusion_matrix,classification_report,
    accuracy_score,precision_score,recall_score,f1_score,roc_auc_score


# In[39]:


start_time = time.time()
DT_array = DT_model(train_data_array, train_label_array, test_data_array,
    test_label_array)
print(time.time()-start_time)


# In[40]:


start_time = time.time()
DT_loop  = DT_model(train_data_loop, train_label_loop, test_data_loop,
    test_label_loop)
print(time.time()-start_time)
```

```
# In[41]:


start_time = time.time()
DT_float = DT_model(train_data_float, train_label_float, test_data_float,
    test_label_float)
print(time.time()-start_time)


# In[42]:


start_time = time.time()
KNN_array = KNN_model(train_data_array, train_label_array, test_data_array,
    test_label_array)
print(time.time()-start_time)


# In[43]:


start_time = time.time()
KNN_loop  = KNN_model(train_data_loop, train_label_loop, test_data_loop,
    test_label_loop)
print(time.time()-start_time)


# In[44]:


start_time = time.time()
KNN_float = KNN_model(train_data_float, train_label_float, test_data_float,
    test_label_float)
print(time.time()-start_time)


# In[40]:


start_time = time.time()
SVM_array = SVM_model(train_data_array, train_label_array, test_data_array,
    test_label_array)
print(time.time()-start_time)


# In[41]:


start_time = time.time()
```

```
761 SVM_loop = SVM_model(train_data_loop, train_label_loop, test_data_loop,
        test_label_loop)
    print(time.time()-start_time)
763

765 # In[42]:

767

    start_time = time.time()
769 SVM_float = SVM_model(train_data_float, train_label_float, test_data_float,
        test_label_float)
    print(time.time()-start_time)
771

773 # In[45]:

775

    start_time = time.time()
777 NN_array = NN_model(train_data_array, train_label_array, test_data_array,
        test_label_array)
    print(time.time()-start_time)
779

781 # In[46]:

783

    start_time = time.time()
785 NN_loop = NN_model(train_data_loop, train_label_loop, test_data_loop,
        test_label_loop)
    print(time.time()-start_time)
787

789 # In[47]:

791

    start_time = time.time()
793 NN_float = NN_model(train_data_float, train_label_float, test_data_float,
        test_label_float)
    print(time.time()-start_time)
795

797 # ### 2.3 Combine the models of array and loop to float to test the reusability
        of the model

799 # In[39]:

801

    train_data_merge = pd.concat([df_train_array, df_test_array, df_train_loop,
        df_test_loop],axis = 0)
```

```
803
test_data_merge = pd.concat([df_train_float, df_test_float],axis = 0)
805
train_data_2, test_data_2, train_label_2, test_label_2 = balance_data(
    train_data_merge, test_data_merge, 0)
807

809 # In[40]:

811
start_time = time.time()
813 DT_model(train_data_2, train_label_2, test_data_2, test_label_2)
print(time.time()-start_time)
815

817 # ### 2.4 Set a baseline, compared to automatically choosing parameters.(Train
    with the data set of previous years and predict on SV-comp2021)

819 # In[99]:

821
def get_params(num):
    count = -1
    for smt in SMT:
        for encode in Encode:
            for verificate in Verification:
                if smt=="--boolector" and encode=="--ir" and (verificate=="--
                    incremental-bmc" or verificate=="--k-induction"):
                    continue
                if verificate == '--falsification':
                    count += 1
                    if count == num:
                        return smt,encode,verificate,0
                else:
                    for k in range(8):
                        count += 1
                        if count == num:
                            return smt,encode,verificate,max_steps[k]

839
# In[100]:
841

843 def roll_back_time(poolDB, solver, encode, verificate, max_k, fileID):

845     for k in range(7,-1,-1):
        temp_k = max_steps[k]
847

        if temp_k>= max_k:
```

```
849             continue
        else:
851             task = select_db(poolDB, "select * from task where fileID = {} and
                    solver = '{}' and encode = '{}' and verificate = '{}' and max_k =
                    {}".format(fileID,solver,encode,verificate,temp_k))
                if len(task) == 0:
853                 continue
                return task[0][8]
855


857 # In[101]:


859

def predict(poolDB, model, file, feature_set):
861     test_data = get_all_feature(poolDB, feature_set, [file])
        test_label = test_data['class']
863     test_data = test_data.drop(columns=['class'])
        y_pred = model.predict(test_data)
865
        status = 8
867     record = -1
        for i in range(len(y_pred)):
869         if status>y_pred[i]:
                status = y_pred[i]
871             record = i

873     sql = "select * from base where fileName = '{}'".format(file)
        default_status = identify_status(select_db(poolDB, sql)[0][3],select_db(
            poolDB, sql)[0][4])
875     default_time = select_db(poolDB, sql)[0][3]

877     solver, encode, verificate, max_k = get_params(record)

879     fileID = select_db(poolDB,"select * from programC where fileName = '{}'".
            format(file))[0][0]
        sql = "select * from task where fileID = '{}' and solver = '{}' and encode =
             '{}' and verificate = '{}' and max_k = {}".format(fileID,solver,encode,
            verificate,max_k)
881     results = select_db(poolDB,sql)
        if len(results) != 0:
883         time = results[0][8]
        else:
885         time = roll_back_time(poolDB,solver,encode,verificate,max_k,fileID)


887
#       print('-----------------------------------------')
889 #       print(file)
#       print("default status is %s"%(default_status))
891 #       print("predict status is %d"%(status))
```

```
#     print(get_params(record))
    return default_status,status,default_time,time


# In[102]:


predict(array_poolDB, DT_array, array_files[0], array_feature)


# In[103]:


def process_predict(poolDB, file_list, model, feature_set):
    total_save = 0
    total_increase = 0
    mispredict = 0

    predict_success = 0
    for file in file_list:
        if file in forbidden_list:
            continue
        defalut_status,status,default_time,time = predict(poolDB, model, file,
            feature_set)
        if defalut_status == 4:
            if status>4:
                print("prediction error!!!!")
                mispredict += 1
            if status<4:
                total_save += 900 - time
        elif defalut_status<4:
            if status>4:
                print("prediction error!!!!")
                mispredict += 1
            if time<default_time:
                total_save += default_time - time
            else:
                total_increase += abs(default_time - time)
        elif defalut_status == 6:
            if status!=6:
                print("predict success")
                predict_success+= 1
#         print('----------------------------')
#         print("total_save", total_save)
#         print("total_increase", total_increase)
#         print(defalut_status, status)
    print("mispredict", mispredict)
    print("change error", predict_success)
    print("total_save", total_save)
```

```
    print("total_increase", total_increase)
941     print("True save", total_save-total_increase)


943
# In[104]:
945


947 process_predict(array_poolDB, test_array_files, DT_array, array_feature)


949
# In[105]:
951


953 process_predict(loop_poolDB, test_loop_files, DT_loop, loop_feature)


955
# In[ ]:
```