JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 21, No. 3, pp. 569-577, Mar. 2020

Check for updates

# 서포트 벡터 머신을 이용한 유사 소프트웨어 분류 모델의 설계

**임 현 일**
경남대학교 컴퓨터공학부 부교수

# Design of Similar Software Classification Model through Support Vector Machine

**Hyun-il Lim**
Associate Professor, Department of Computer Engineering, Kyungnam University, Gyeongsangnam-do 51767, Korea

**[요 약]**

컴퓨팅 환경의 효율적인 활용을 위해서 소프트웨어의 분석 기술은 중요한 요소가 되고 있다. 본 논문에서는 기계 학습의 접근 방법 중 하나인 서포트 벡터 머신을 통해서 유사 소프트웨어를 분류할 수 있는 모델을 제안한다. 분석 모델의 설계를 위해서 소프트웨어의 특성을 입력 데이터로 표현하기 위한 코드 분석 방법을 제안하고 유사 소프트웨어 분류를 위한 서포트 벡터 머신의 분류 모델을 설계한다. 본 논문에서 제안한 방법의 정확성을 검증하기 위해서 실제 사용되는 자바 어플리케이션을 이용한 실험에서 93.7%의 유사 소프트웨어 분류 정확도를 보여주었다. 실험 결과로부터 본 논문에서 제안한 소프트웨어 코드 데이터는 소프트웨어의 특성 분류를 위한 데이터로 효과적으로 사용될 수 있으며, 서포트 벡터 머신을 이용한 유사 소프트웨어 분류 모델은 소프트웨어 분석에 적용될 수 있음을 확인할 수 있다. 소프트웨어를 분석하기 위한 명령 코드 정보는 소프트웨어 분석을 위한 빅데이터, 인공지능, 기계 학습 등의 분야에 적용할 수 있을 것이라 기대된다.

**[Abstract]**

For the efficient use of computing environments, software analysis is becoming an important factor. In this paper, we design and propose a model that can classify similar software through support vector machine. We propose a code analysis method for expressing the characteristics of the software as input data and design a similar software classification model of support vector machines. The accuracy of similar software classification was 93.7% in the evaluation experiments with real-world Java applications. From the results of the experiment, we can confirm that the software code data proposed in this paper can be effectively used as data for representing the characteristics of software and that the proposed similar software classification models using support vector machine can be applied in software analysis. The proposed code analysis method is expected to be applied to various areas such as big data, artificial intelligence and machine learning for software analysis.

**색인어 :** 코드 분석, 자바 바이트코드, 유사 소프트웨어 분류, 소프트웨어 분석, 서포트 벡터 머신
**Key word :** Code analysis, Java bytecode, Similar software classification, Software analysis, Support vector machine

**http://dx.doi.org/10.9728/dcs.2020.21.3.569**



**Received** 14 January 2020; **Revised** 15 March 2020
**Accepted** 25 March 2020

***Corresponding Author; Hyun-il Lim**

**Tel:** +82-55-249-2650
**E-mail:** hilim@kyungnam.ac.kr

## Ⅰ. 서 론

소프트웨어는 하드웨어와 더불어 정보 시스템의 운영에서 사용자에게 다양한 기능을 제공하고 있다. 컴퓨팅 환경에서 소프트웨어를 효율적으로 운영하기 위해서 소프트웨어가 가지는 정적, 동적 특성들을 체계적으로 이해하고 기술할 수 있다면 다양한 서비스를 제공하는 소프트웨어를 개발하고, 관리하는데 효율적으로 활용할 수 있을 것이다. 소프트웨어의 특성 분석에서 유사 소프트웨어를 분류하는 방법은 소프트웨어가 가지는 유사한 기능을 분석하는 것뿐만 아니라 중복된 코드 검색, 소프트웨어 도용 및 표절 탐지 [1, 2, 3], 악성 코드 분석 [4, 5] 등 다양한 응용 분야를 가지고 있다. 따라서 유사 소프트웨어 분류 기술은 소프트웨어 개발 및 운용에 기반이 되는 기술이다.

기계 학습 [6]은 인공 지능의 한 분야로서 주어진 문제를 풀기위한 다 수의 데이터를 이용해서 학습 과정을 통해 학습 데이터에 대한 결과를 찾아낼 수 있는 모델을 만들어가는 방법을 의미한다. 학습을 통해 만들어진 모델은 학습 과정에 사용되지 않은 새로운 데이터에 대해서도 효과적인 결과를 찾아낼 수 있으며, 기존의 컴퓨팅 문제 해결 방법에서 체계적인 해결책을 찾기 어려운 많은 문제들에 대해 기계 학습 방법이 적용되고 있다. 이런 기계 학습 분야는 선형 회귀 [7], 로지스틱 회귀, 서포트 벡터 머신, 신경망 등 다양한 방법으로 설계되고 있으며 실생활에 적용하기 위한 연구가 진행되고 있다.

본 논문에서는 기계 학습의 한 방법인 서포트 벡터 머신을 통해서 유사 소프트웨어를 분류할 수 있는 모델을 설계하고 제안된 방법을 이용해서 유사 소프트웨어 분류 모델의 정확성을 검증한다. 기존의 연구에서는 유사 소프트웨어 분류를 위해서 소프트웨어가 가지는 구조적 특성, 제어 흐름, 명령어 순서열, API 정보, 문법 특성 등을 직접 분석하고 비교하는 방법을 이용했다. 기존의 방법들은 유사 소프트웨어 분류를 위한 소프트웨어의 특성을 직접 분석하고 분석한 정보를 통해서 모델을 설계하였다. 최근에 주목 받고 있는 기계 학습 방법은 문제 해결을 위한 분석 과정을 주어진 입력으로부터 특성을 찾아가는 학습을 통해 스스로 모델을 찾아간다는 것이 중요한 특징이라고 볼 수 있다. 따라서 기계 학습 방법은 직접 분석 방법을 통해서 고려하기 어려운 미세한 입력 데이터의 특성들을 학습 과정을 통해서 결과에 반영할 수 있다는 장점이 있다.

소프트웨어는 특성상 이진 정보들로 구성되고 있으며 분석기를 통해 직접 분석하기에는 구조가 복잡할 뿐만 아니라 실행 과정에서 반영되는 미세한 특징들을 직접적인 분석을 통해서 체계적으로 반영하기 어렵다는 특성이 있다. 기계 학습 방법은 이런 소프트웨어 분석에 어려운 점들을 학습 과정을 통해 많은 부분 보완해 줄 수 있는 방법이 될 수 있을 것이라 기대된다.

본 논문은 다음과 같이 구성된다. 2장에서는 본 연구에서 적용하는 서포트 벡터 머신을 이용한 학습에 대해서 소개하고, 3장에서는 소프트웨어의 특성을 서포트 벡터 머신의 학습 데이터로 표현하기 위한 분석 방법을 제안한다. 4장에서는 소프트웨어의 특성 데이터를 이용해서 서포트 벡터 머신의 분류 모델을 생성하기 위한 방법을 설계하고 5장에서는 본 논문에서 제안하는 방법을 구현하고 실험한 결과를 통해서 정확성을 검증한다. 마지막으로 6장에서 본 논문의 결론을 맺는다.

## Ⅱ. 서포트 벡터 머신을 이용한 학습

서포트 벡터 머신(support vector machine, SVM) [8, 9, 10, 11, 12]은 데이터가 가지는 특성을 모델링하고 특성에 따른 정보를 이용해 데이터를 분류하기 위한 방법으로 널리 사용되는 기계 학습 방법이다. SVM은 일반적으로 두 개의 카테고리로 구성된 데이터가 있을 때, 각 데이터를 어느 한쪽을 선택해서 분류하는 이진 분류 방법에서 널리 적용되고 있다. 이 방법은 분류 모델을 구축하기 위한 학습 데이터로부터 특성 데이터들을 습득하고, 습득한 정보를 통해 두 개의 카테고리를 분류하는 모델을 생성한다. 학습 데이터로부터 생성한 분류 모델은 학습에 사용되지 않은 새로운 데이터에 대해서도 그 데이터가 속하는 카테고리를 효과적으로 예측할 수 있고, 새로운 문제에 대해 적용할 수 있는 분류 모델이 된다. SVM의 분류 결과가 높은 변별력과 정확성을 가지기 위해서 학습 데이터의 특성들이 분류에 필요한 정보를 효율적으로 표현하고 있는 것이 중요하며, 두 개의 카테고리에 속하는 학습 데이터를 카테고리 별로 가장 큰 변별력을 가질 수 있는 분류 기준을 모델로 만듦으로써 분류 결과의 변별력을 높인다.

SVM을 이용한 분류 모델 데이터의 차원은 입력 데이터의 특성 값의 차원에 따라서 결정된다. 예를 들어 입력 데이터가 2차원의 특성 데이터로 표현된다면 2차원 평면의 데이터를 분할할 수 있는 1차원의 직선으로 표현된다. 만약 입력 데이터가 3차원 특성 데이터를 가지고 있다면 3차원 공간에 사상되는 데이터를 분할할 수 있는 2차원 평면으로 결정된다. 따라서 입력 데이터가 $n$차원을 가지는 특성 데이터로 표현될 때, SVM의 분류 모델은 $n$-1 차원의 초평면 (hyperplane)으로 표현된다. 이 초평면을 기준으로 데이터가 위치하는 방향이 입력 데이터에 대한 분류 기준이 된다. 따라서, 분류 기준이 되는 초평면의 변별력 있는 분류 정확성을 확보하기 위해서 초평면으로 부터 가장 가까운 위치에 있는 데이터에 대해서 높은 마진을 확보할 수 있도록 분류할 수 있는 초평면을 분류 기준으로 결정하는 것이 중요하다.

데이터의 특성을 이용해서 분류하는 문제는 기계 학습에서 널리 응용되고 있으며, 학습 데이터들을 통해 분류되는 데이터의 특성들을 분석하고 이를 기준으로 두 개의 클래스로 분류할 수 있다. SVM에서 입력으로 사용되는 데이터가 $n$차원의 특성 데이터로 표현된다면, 이 데이터를 분류하는 분류 모델은 $n$-1 차원의 초평면을 찾는다. 예를 들어 2차원 입력 데이터에 대한 SVM은 그림 1에서처럼 2차원 평면상에 입력 데이터가 위치하게 된다. 각 데이터는 그 속성에 따라서 클래스 A와 클래스 B

중 하나의 클래스에 속할 때 주어진 데이터를 가장 효과적으로 분류할 수 있는 1차원 초평면을 찾는 것이다. 본 사례에서 두 클래스의 데이터를 분류하는 세 개의 초평면 $H_1, H_2, H_3$를 비교해보면, $H_3$는 클래스 B에 속하는 데이터는 모두 정확하게 분류하고 있지만 클래스 A에 속하는 일부 데이터는 클래스 B로 잘못 분류하고 있다. 두 개의 클래스를 정확하게 분류하고 있는 두 초평면 $H_1$과 $H_2$를 보면 주어진 학습 데이터에 대해서는 두 개의 클래스를 정확하게 분류하고 있다. 하지만 학습에 사용되지 않은 새로운 데이터에 대해서 적용하기 위해서 학습 데이터를 보다 적은 오차로 분류할 수 있는 초평면을 선택하는 것이 바람직하다. 따라서, 초평면으로부터 가장 인접한 위치에 있는 데이터를 서포트 벡터라고 하고, 이 서포트 벡터와의 마진(margin)을 최대화하는 초평면을 분류 모델로 학습하는 방법이다.

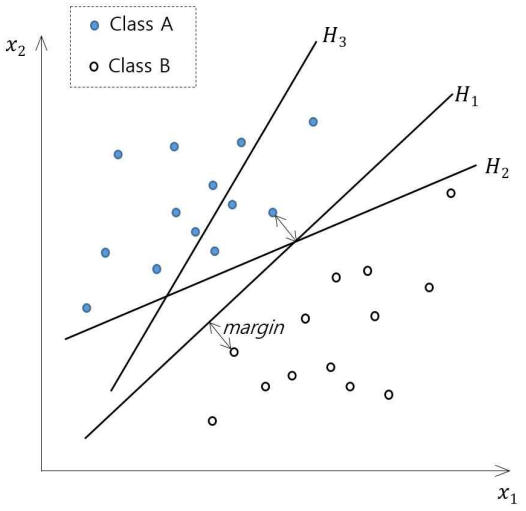



**그림 1.** 2차원 특성 데이터에 대한 서포트 벡터 머신 분류 모델 예제

**Fig. 1.** An example of support vector machine classification model for two dimensional data.

따라서, 학습 데이터를 정확하게 분류할 수 있는 벡터는 다수개가 존재할 수 있지만, 그 중에서 데이터 분류 결과의 오차를 최소화할 수 있도록 서포트 벡터와의 마진을 최대로 할 수 있는 초평면을 분류 모델로 학습 하는 방법이 SVM의 분류 모델이다.

## Ⅲ. 소프트웨어 특성 데이터 분석

컴퓨터 시스템에서 소프트웨어의 중요성은 점점 증가하고 있으며 시스템 운영에서 다양한 서비스를 제공하며 활용되고 있다. 소프트웨어의 분석을 통해 소프트웨어의 특성을 잘 이해할 수 있다면 소프트웨어 개발 및 효율적인 활용을 위해서 다양한 방법으로 활용될 수 있을 것이다.

소프트웨어 분석 기술은 지속적으로 발전하고 있으며, 소프트웨어 분석을 위한 특성을 추출하고 표현할 수 있는 분석 데이터 생성 기술은 효율적인 소프트웨어의 분석 및 이해를 위해서 기본이 되는 기술이다. 본 장에서는 소프트웨어의 특성을 추출하고 이를 분석 데이터로 활용할 수 있는 데이터 분석 방법에 대해서 기술한다.

### 3-1 소프트웨어의 특성 추출

소프트웨어의 분석을 위한 특성을 추출하기 위해서 소프트웨어의 근본적인 특성을 파악하고 반영하는 것이 중요하다. 소프트웨어는 컴퓨터 시스템의 동작에서 하드웨어의 다양한 기능을 구동하고 사용자와의 상호작용을 통해 다양한 기능의 서비스를 제공하는 무형의 데이터 및 정보들의 조합으로 구성된다. 소프트웨어는 하드웨어의 기종에 따라 정의된 이진(binary) 데이터들로 표현되며, 이런 이진 데이터들을 선처리 등의 사전 분석이 없이 직접 입력 데이터로 적용하기는 어렵다. 왜냐하면 소프트웨어의 이진 데이터로부터 소프트웨어가 가지는 논리적인 구조 및 특성들을 효과적으로 기술하는데 한계가 있기 때문이다. 따라서 이런 소프트웨어의 원형이 가지는 이진 데이터로부터 소프트웨어의 특성을 체계적으로 요약하거나 추출하는 분석 과정이 소프트웨어의 분석에 많은 도움이 된다.

본 논문에서는 소프트웨어의 동작을 구성하는 명령어 코드들을 체계적으로 추출하고 분석함으로써 소프트웨어의 특성을 표현할 수 있는 데이터 표현 방법으로 제안한다. 소프트웨어는 소프트웨어의 실행에 필요한 데이터들과 소프트웨어의 실행 명령어들의 집합으로 구성된다. 여기서 소프트웨어의 실행 명령어는 실제 소프트웨어가 하드웨어 상에서 동작할 때 실행되는 명령어들을 의미하고, 다 수의 실행 명령어들이 조합되어 컴퓨터에서 다양한 기능을 제공하는 소프트웨어가 동작한다. 이런 명령어들의 정보를 추출하고 다차원의 데이터로 표현함으로써 각 소프트웨어의 개별적인 특징을 요약하는 특성 데이터로 활용할 수 있다. 소프트웨어의 명령 코드들은 소프트웨어의 분석 과정에서 추출할 수 있으며, 소프트웨어의 특성을 표현하는 방법으로 제시한다.

### 3-2 소프트웨어의 다차원 데이터 표현

본 절에서는 소프트웨어의 분석을 통해서 소프트웨어 특성을 기계 학습의 학습 데이터로 활용하기 위한 데이터 표현 방법을 제안한다. 소프트웨어 개발을 위해서 다양한 프로그래밍 언어들이 활용되고 있으며 개발된 소스 코드는 실행을 위해서 컴파일 등의 과정을 거쳐 실행 가능한 이진 데이터로 구성된 실행 파일을 생성한다. 실행 파일은 이진 데이터로 표현되며, 실행에 필요한 데이터와 실행 명령을 기술하는 명령어 코드들로 구성된다.

```
class FactorialExample{
    public static void main(String args[]) {
        int i, fact=1;
        int number=5;
        for(i=1; i<=number; i++) {
            fact=fact * i;
        }
        System.out.println("Factorial of "
                                +number+" is: "+fact);
    }
}
```

**그림 2.** 자바 팩토리얼 예제 프로그램
**Fig. 2.** An example of Java factorial program.

예를 들어 그림 2에서는 팩토리얼을 계산하는 자바 프로그램을 보여주고 있다. 소스 코드에서는 반복문을 통해서 반복적으로 값을 곱하는 연산 과정 및 소프트웨어의 구조적 특성이 드러난다. 반면 이 프로그램을 실행하기 위해서는 컴파일을 거쳐서 실행 가능한 코드를 생성해야하는데, 실행 가능한 코드는 이진 데이터로 구성되기 때문에 이런 구조적인 특성이 쉽게 드러나지 않는다.

| Address | Opcode | Bytecode |
|---|---|---|
| 0: | 0x04 | iconst_1 |
| 1: | 0x3d | istore_2 |
| 2: | 0x08 | iconst_5 |
| 3: | 0x3e | istore_3 |
| 4: | 0x04 | iconst_1 |
| 5: | 0x3c | istore_1 |
| 6: | 0x1b | iload_1 |
| 7: | 0x1d | iload_3 |
| 8: | 0xa3 | if_icmpgt 21 |
| 11: | 0x1c | iload_2 |
| 12: | 0x1b | iload_1 |
| 13: | 0x68 | imul |
| 14: | 0x3d | istore_2 |
| 15: | 0x84 | iinc 1, 1 |
| 18: | 0xa7 | goto 6 |
| 21: | oxb1 | return |

**그림 3.** 팩토리얼 프로그램에 대한 자바 명령 코드 (자바 바이트코드)
**Fig. 3.** The Java bytecode for the Java factorial program.

그림 3은 그림 2에서 보여주고 있는 팩토리얼 프로그램의 실행 파일인 자바 클래스 파일에 포함된 바이트코드 [13]를 보여주고 있다. 이 명령어 코드는 프로그램이 실행하는 과정에서 수행하는 연산들을 보여주지만, 소스 코드에서와 같은 소프트웨어의 구조적인 특성들이 쉽게 노출되지 않는다. 반면 해당 아키텍처에서 실행되는 코드들은 그 소프트웨어의 실제 실행되는 연산들로 구성되며 실행 과정에서 드러나는 다양한 동작 특성들에 대한 정보를 가지고 있다. 따라서, 이 코드들의 분포 정보를 특성으로 하는 데이터로 요약하면 소프트웨어의 명령어 수준에서 실행되는 연산 특성들을 효과적으로 표현할 수 있을 것이다.

**표 1.** 자바 팩토리얼 프로그램에 대한 코드 분포 데이터
**Table 1.** Code distribution data for the Java factorial program.

| Name | Analyzed code data |
|---|---|
| SW Name | Java Factorial |
| Code Name in Factorial | [ iconst_1, istore_2, iconst_5, istore_3, istore_1, iload_1, iload_3, if_icmpgt, iload_2, Imul, iinc, goto, return ] |
| # of Code in Factorial | [2, 2, 1, 1,1, 2, 1, 1, 1, 1, 1, 1, 1] |

표 1은 그림 3에서 보여주는 자바 프로그램에서 분석한 명령 코드들로부터 코드의 분포 정보를 요약한 특성 정보를 보여준다. 이 코드 분포 데이터는 소프트웨어의 명령 코드에 대해서 각각의 코드들이 프로그램의 실행에 얼마나 빈번하게 실행되는지 정보를 표현하고 있다. 크고 복잡한 프로그램일수록 보다 다양하고 많은 명령어 코드들이 포함되기 때문에 명령어의 코드 특성을 데이터로 표현하면 소프트웨어의 특성이 보다 명확하게 드러날 수 있다.

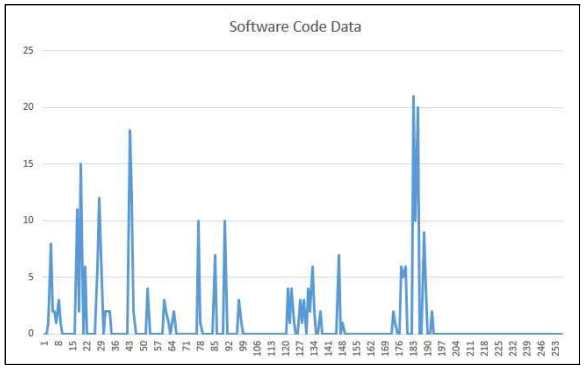



**그림 4.** AcornDB 프로그램에 포함된 한 클래스 파일로 부터 분석한 코드 분포 데이터
**Fig. 4.** Code distribution data analyzed from a Java classfile in AcornDB.

그림 4는 실제 자바 프로그램을 사용되는 AcornDB에 포함된 실행 프로그램에서 추출한 프로그램의 코드 데이터의 예를 보여주고 있다. 그림에서 수평 축은 자바 프로그램이 가지는 256개의 바이트코드를 표현하고 있으며 세로축은 분포 정보 특성을 표현한다. 이 데이터는 소프트웨어에서 특징적으로 사용되는 명령어의 구성 정보 및 특성을 쉽게 파악할 수 있고, 소프트웨어의 고유한 특성 정보로 표현될 수 있다.

소프트웨어에 포함되는 명령어 코드의 분포 정보는 소프트웨어를 실행하는데 필요한 복잡한 소프트웨어의 구조를 단순한 선형의 다차원 데이터 형태로 모델링할 수 있다. 소프트웨어의 다차원 데이터 표현은 실행 가능한 프로그램 코드의 특성을 완전하게 포함하지는 않지만 코드의 분포 정보로서 소프트웨어의 실행 과정에서 나타나는 명령들의 구성 및 사용 패턴 등의 특징들을 변별력 있게 요약할 수 있는 방법이다.

### 3-3 소프트웨어 특성 데이터 응용

소프트웨어의 특성을 분석하기 위해서 소프트웨어가 가지는 다양한 요소들을 고려할 수 있는데, 명령 코드 분석을 통한 다차원 특성 데이터는 소프트웨어를 구성하는 연산들의 분포 특성을 요약해서 표현할 수 있다. 이 데이터는 소프트웨어 분석을 위한 특성을 필요로 하는 다양한 응용 분야에서 데이터로 사용할 수 있으며, 특히 다양한 데이터를 필요로 하는 빅데이터, 인공지능 및 기계 학습과 같은 분야에서 소프트웨어를 다루는 데이터로 활용할 수 있을 것이다.

소프트웨어의 특성을 분석하고 비교를 통한 유사 소프트웨어 분류 방법은 소프트웨어의 특성 정보를 효과적으로 표현할 수 있을 때, 특성 정보로부터 서로 다른 소프트웨어 사이에 차이점을 효율적으로 예측할 수 있을 것이다. 본 논문에서는 소프트웨어의 코드 명령 정보 및 분포 정보를 요약하는 다차원 데이터를 이용해서 소프트웨어의 특성을 모델링한다. 소프트웨어 특성 데이터는 유사 소프트웨어 분류를 위한 모델로 적용할 수 있도록 SVM 모델에 적용할 수 있도록 모델을 설계한다.

## Ⅳ. 서포트 벡터 머신을 이용한 소프트웨어 분류 모델 설계

본 장에서는 소프트웨어 특성 데이터를 이용해서 SVM의 분류 모델을 생성하기 위해서 SVM의 분류 알고리즘에 맞도록 알고리즘을 설계한다. 유사 소프트웨어 분류를 위한 소프트웨어 입력 데이터는 소프트웨어가 가지는 명령어의 종류 및 분포에 따라 다차원의 데이터 특성을 가진다. 자바 프로그램은 256개의 자바 바이트코드 [12]를 이용해서 소프트웨어의 클래스 파일이 만들어진다. 본 논문에서는 256개의 바이트코드를 명령어로 가지는 자바 프로그램에 대한 서포트 벡터 모델을 설계하고 평가한다. 자바 프로그램이 가지는 총 256개의 바이트코드 명령어에 대해서 각각의 자바 프로그램은 256개의 특성 값을 가지는 256차원 데이터로 $[x_1, x_2, \cdots, x_{256}]$와 같이 표현할 수 있다. 소프트웨어 특성 데이터에 대해서 학습 데이터는 256차원 데이터와 지도 학습에 필요한 데이터의 결과값 $y$의 쌍으로 다음 수식 (1)과 같이 표현할 수 있다.

$$([x_1, x_2, \cdots, x_{256}], y) \tag{1}$$

여기서 $y$는 소프트웨어 특성 데이터 $[x_1, x_2, \cdots, x_{256}]$가 속하는 클래스를 가지는 레이블로 학습에 사용되며, 입력 데이터가 속하는 클래스에 따라 1 또는 −1중에서 하나의 값을 가진다. 이 학습 데이터를 속하는 클래스에 따라 분류하는 초평면은 두 클래스의 결과 $y$가 1 또는 −1로 구분되므로 두 클래스를 양수와 음수로 구분할 수 있도록 다음 수식 (2)의 조건을 만족하는 초평면이 된다.

$$w \cdot [x_1, x_2, \cdots, x_{256}] - b = 0 \tag{2}$$

여기서 $\cdot$는 두 벡터의 내적 연산을 나타내고, $w$는 위의 식으로 표현되는 초평면의 법선 벡터가 된다. 이 초평면에 의해서 분류되는 두 개의 클래스에 대해서 각각 초평면과 가장 인접한 서포트 벡터 $X^+$와 $X^-$가 존재하는데, 서포트 벡터 $X^+$는 $y=1$ 인 클래스에 속하는 데이터 중에서 초평면과 가장 가까운 위치에 있는 데이터가 되고, 서포트 벡터 $X^-$는 $y=-1$인 클래스에 속하는 데이터 중에서 초평면과 가장 가까운 위치에 있는 데이터로 정의된다. 따라서 두 클래스를 분류하는 초평면과 같은 기울기를 가지면서 서포트 벡터 $X^+$를 지나는 초평면은 다음 수식 (3)과 같이 표현할 수 있다.

$$w \cdot [x_1, x_2, \cdots, x_{256}] - b = 1 \tag{3}$$

또한 두 클래스를 분류하는 초평면과 같은 기울기를 가지면서 서포트 벡터 $X^-$를 지나는 초평면은 다음 수식 (4)와 같이 표현할 수 있다.

$$w \cdot [x_1, x_2, \cdots, x_{256}] - b = -1 \tag{4}$$

수식 (2)의 초평면에 대해서 입력 데이터에 대한 분류 마진은 두 개의 서포트 벡터를 지나는 초평면 (2)와 (4) 사이의 거리 $\frac{2}{\|w\|}$로 구해진다. 따라서, SVM은 분류 마진을 최대로 하기 위해서 두 초평면 사이의 거리를 최대로 만드는 초평면을 분류 모델로 학습해 가는 것이다. 따라서 $w$의 노름(norm) $\|w\|$를 최소로 만드는 초평면을 찾으면 모델이 완성된다.

두 개의 서포트 벡터 $X^+$ 와 $X^-$는 각각 초평면에서 가장 가까이 있는 클래스의 데이터 이므로 이 두 데이터 사이에는 다른 데이터가 존재하지 않는다. 따라서, $y=1$ 에 속하는 모든 데이터는 다음 수식 (5)의 조건이 만족한다.

$$w \cdot [x_1, x_2, \cdots, x_{256}] - b \geq 1 \tag{5}$$

동일한 방법으로 $y=-1$에 속하는 모든 데이터는 다음과 같이 수식 (6)의 조건이 만족한다.

$$w \cdot [x_1, x_2, \cdots, x_{256}] - b \leq -1 \quad (6)$$

위 두 식 (5)와 (6)에서 양변에 $y$를 곱하면 다음과 같이 하나의 수식 (7)의 결과를 얻을 수 있다.

$$y \times (w \cdot [x_1, x_2, \cdots, x_{256}] - b) \geq 1 \quad (7)$$

따라서, 256차원의 명령 코드 특성 데이터를 입력으로 하는 SVM의 분류 모델은 두 클래스의 모든 데이터에 대해서 데이터를 최대의 마진으로 분류할 수 있는 다음 수식 (8)의 조건을 만족하는 초평면을 찾는 문제로 설계할 수 있다.

$$\arg\min_{(w,b)} \|w\| \quad (8)$$
$$where \;\; y \times (w \cdot [x_1, x_2, \cdots, x_{256}] - b) \geq 1$$

소프트웨어의 특성 데이터에 대해서 조건을 만족하는 법선 벡터 $w$와 상수 $b$는 소프트웨어 분석을 위한 두 개의 클래스의 입력 데이터를 반복적으로 학습하면서 조정할 수 있다. 학습을 반복함에 따라 마진을 최대로 하는 법선 벡터 $w$와 상수 $b$를 찾을 수 있으며, 학습을 통해서 최적화된 결과의 $w$와 $b$에 대해서 초평면 $w \cdot [x_1, x_2, \cdots, x_{256}] - b = 0$는 유사 소프트웨어를 분류하는 SVM의 분류 모델이 된다. 본 장에서 설계한 SVM 분류 모델의 정확성은 실험을 통해서 결과를 분석한다.

## Ⅴ. 실험

본 장에서는 본 논문에서 제안한 SVM을 이용한 유사 소프트웨어 분류 모델을 구현하고 실제 사용되는 자바 프로그램에 대한 코드 분석 정보를 벤치마크 데이터로 수행한 실험 결과를 통해 유사 소프트웨어 분류 모델의 활용 가능성을 검증한다.

### 5-1 실험 환경 구성

그림 5는 본 논문에서 제안한 SVM을 이용한 유사 소프트웨어 분류 모델의 전체 구조를 보여주고 있다. SVM을 이용한 분류 모델을 구성하기 위해서 유사 소프트웨어와 상이한 소프트웨어에 대한 벤치마크 소프트웨어의 집합을 구성하고 SVM의 학습 데이터는 벤치마크 소프트웨어로부터 바이너리 코드 분석기를 통해서 명령어 집합을 추출한 후 각 소프트웨어의 특징을 포함하는 다차원 데이터를 생성함으로써 학습 데이터를 완성한다. 이 두 클래스의 소프트웨어 데이터 집합은 SVM을 통해서 학습 과정을 거치면 유사 소프트웨어의 분류를 위한 SVM 분류 모델을 생성한다. 생성된 모델은 정확성 및 효율성을 검증하기 위해서 테스트 데이터로 구성된 데이터 셋을 이용해 실험 결과를 확인하고 본 모델의 정확성을 확인할 수 있다.

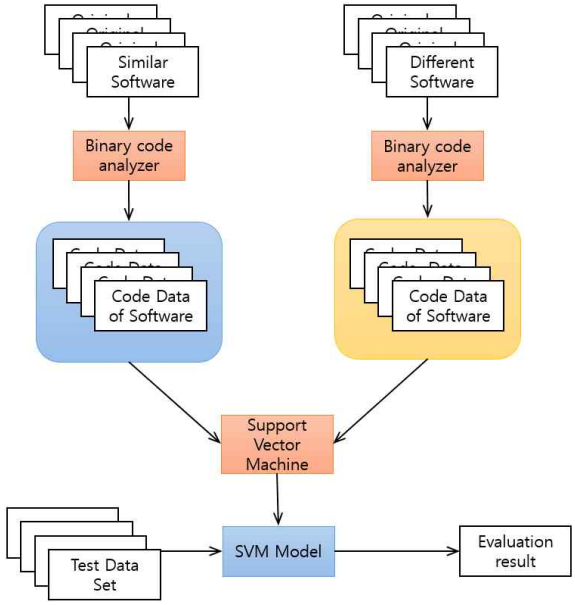



**그림 5.** 서포트 벡터 머신을 이용한 유사 소프트웨어 분류 모델의 구조
**Fig. 5.** The structure of classification model through support vector machine.

본 논문에서 제안한 분석 모델을 구현하기 위해서 파이썬과 자바 언어를 이용하였으며, LIBSVM 라이브러리 [14]를 이용해서 SVM을 이용한 학습 모델을 구현하였다. 구현된 분석 모델은 마이크로소프트 윈도우 7 환경에서 실험을 수행하였다.

**표 2.** 벤치마크 실험에 사용한 데이터 정보
**Table 2.** The specification of the benchmark data.

| Name | # of classfiles | Avg. # of bytecodes | Max. # of bytecodes |
|---|---|---|---|
| Aelfred | 7 | 290.4 | 1264 |
| JUnit | 100 | 79.6 | 980 |
| AcornDB | 218 | 154.6 | 3728 |
| Total | 225 | 134.4 | 3728 |

본 실험 환경에서 사용한 소프트웨어 데이터는 실제 환경에서 사용되는 자바 프로그램 The Aelfred XML Parser [15], JUnit [16], AcornDB [17]를 이용해서 학습 데이터와 테스트 데이터를 생성하였다. 표 2는 실험에 사용한 데이터의 정보를 보여주고 있다. 3개의 프로그램 중에서 The Aelfred XML Parser는 클래스파일의 수는 적지만 평균 코드의 사이즈가 큰 프로그램으로 구성되어 있고, 전체 프로그램의 평균 바이트코드 수는 134.4개 이다.

실험에 사용한 데이터는 자바 프로그램에 포함되어 있는 실행 파일인 클래스 파일들로부터 자바 바이트코드를 추출하고 추출된 바이트코드 명령어를 분석해서 소프트웨어 특성 데이터를 생성하였다. 생성한 데이터는 학습 데이터와 검증 데이터

로 나누어서 학습 및 검증 실험을 수행하였다. 그리고 유사한 소프트웨어 집합을 구성하기 위해서 자바 프로그램의 변환 도구인 Smokescreen 난독화 도구를 사용해서 변환한 유사 프로그램을 포함하였다. Smokescreen은 단순한 코드 내의 이름 변경뿐만 아니라 제어 흐름을 바꾸고 바이트코드 명령어를 바꾸는 변환을 수행하고, 코드의 변환을 통해서 유사 프로그램의 벤치마크 데이터를 생성하였다.

### 5-2 실험 결과 및 분석

표 2에서 기술한 세 개의 벤치마크 프로그램은 학습 데이터 셋과 검증 데이터 셋으로 분리하여 데이터를 생성하였다. 학습에 사용한 데이터는 세 프로그램으로부터 유사한 클래스의 파일과 다른 클래스의 파일을 구성하였으며 생성한 학습 데이터의 개수는 총 90,796개였다. 이 학습 데이터로부터 SVM을 학습하고 유사 소프트웨어 분류를 위한 SVM 모델을 생성하였다.

**표 3.** 벤치마크 데이터를 이용한 SVM 모델의 실험 결과
**Table 3.** The experimental results of SVM model with the benchmark data.

| | |
|---|---|
| # of Data for Training SVM | 90,796 |
| # Data for Evaluating SVM Model | 12,321 |
| # of Correct classification through SVM Model | 11,541 |
| Evaluation Accuracy | 93.7% |

학습을 통해 생성한 SVM 모델의 정확성 검증을 위해 사용한 테스트 데이터는 프로그램에서 테스트를 위해 별도로 구성한 클래스 파일로부터 생성하였다. 유사한 클래스와 다른 클래스의 프로그램을 생성한 후 검증에 사용한 데이터의 개수는 12,321개의 소프트웨어 데이터 파일로 구성된다. 검증 데이터는 학습을 통해 생성된 SVM 모델을 통해 분류 실험을 수행하였으며, 12,321 개의 데이터 클래스 중에서 11,541개의 소프트웨어 클래스를 올바르게 예측하였으며, 93.7%의 정확도를 보여주었다. 표 3은 본 실험의 결과를 정리해서 보여주고 있다.

본 논문에서 제안한 소프트웨어의 코드 데이터를 사용하는 SVM 분류 모델은 소프트웨어 코드가 가지는 각 소프트웨어의 특징들이 특성 데이터를 통해 효과적으로 표현되었기 때문에 가능하다. 그림 6과 7은 표 2에서 보여주고 있는 본 SVM의 유사 소프트웨어 분류 실험에서 사용한 벤치마크 프로그램의 특성을 보여주고 있다.

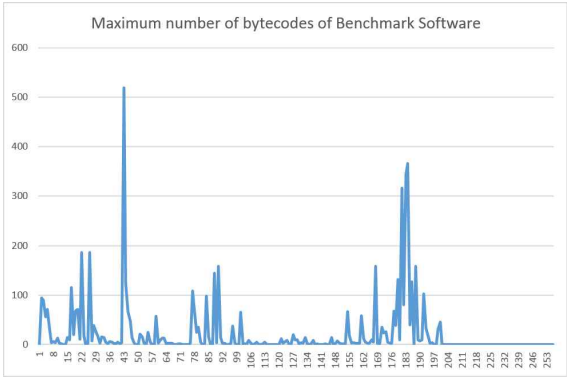



**그림 6.** 벤치마크 프로그램의 자바 바이트코드 최대 빈도
**Fig. 6.** Maximum number of Java bytecodes of the benchmark programs.

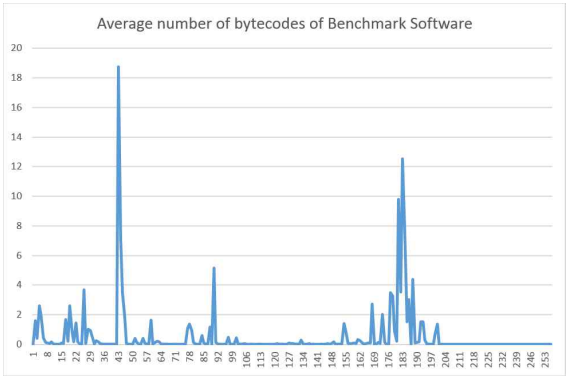



**그림 7.** 벤치마크 프로그램의 자바 바이트코드 평균 빈도
**Fig. 7.** Average number of Java bytecodes of the benchmark programs.

그림 6은 벤치마크 프로그램에서 사용된 자바 프로그램별로 포함된 자바 바이트코드 명령어의 최대 개수 정보를 보여주고 있고, 그림 7은 프로그램에서 사용된 바이트코드 명령어의 평균 개수를 보여준다. 자바 프로그램의 바이트코드는 총 256개로 구성되고 각각의 명령어에 대해서 사용 빈도를 분석했을 때, 하나의 프로그램에서 최대 빈도를 가지는 5개의 명령어는 aload_0, invokespecial, invokevirtual, getfield, iload 순서로 나타났으며, 평균 사용 빈도가 높은 5개의 명령어는 aload_0, invokevirtual, getfield, invokespecial, aload_1 명령어 순서로 나타났다.

소프트웨어에서 명령어 코드들의 사용 패턴 및 사용 빈도는 각각의 소프트웨어의 작업 내용 또는 기능에 따라 명령어의 특성 분포가 달라진다. 소프트웨어의 특성 정보들은 SVM의 분류 모델 생성을 위한 입력 데이터로 변별력을 가질 수 있기 때문에 높은 정확도로 유사 소프트웨어를 분류할 수 있다고 평가된다. 예를 들어, 평균적인 명령어 코드 사용 패턴에 비해서 특정 명령어 코드들이 많이 사용되거나 적게 사용되는 소프트웨어의 특성은 개별적인 소프트웨어를 구별할 수 있는 변별력 있는 특성 정보로 사용됨을 알 수 있다.

SVM의 학습을 통해 변별력 있는 유사 소프트웨어 분류 모델을 생성하기 위해서 입력 데이터에서 소프트웨어 특성 정보를 효과적으로 표현하는 것이 중요하다. 본 논문에서는 소프트웨어의 명령어 코드 특성 정보 분석을 통한 소프트웨어 데이터를 분석하고 이를 이용한 입력 데이터를 이용해 SVM의 분류 모델을 생성할 수 있었다. 또한 본 논문에서 제안한 방법의 정확성을 검증하기 위해서 실제 사용되는 자바 어플리케이션을 벤치마크 프로그램으로 실험을 하였다. 검증 데이터를 통한 유사 소프트웨어 분류 모델 실험 결과에서 93.7%의 정확도를 보여줌으로써 소프트웨어 코드 데이터를 통해 SVM을 이용한 유사 소프트웨어 분류 모델이 효과적으로 사용될 수 있음을 확인할 수 있다. 또한, 소프트웨어 명령 코드 정보는 유사 소프트웨어 분류를 위한 모델 생성 뿐 만 아니라 소프트웨어를 대상으로 하는 빅데이터, 인공지능, 기계 학습 등의 분야에 적용할 수 있는 분석 및 입력 데이터로 효과적으로 활용될 수 있을 것이라 기대된다.

## Ⅵ. 결 론

최근의 컴퓨팅 환경에서 소프트웨어의 중요성은 더욱 증가하고 있으며, 소프트웨어의 분석 기술은 효율적인 소프트웨어 개발 및 운영을 위해서 중요한 요소가 되고 있다. 본 논문에서는 기계 학습의 접근 방법 중 하나인 SVM을 통해서 유사 소프트웨어를 분류할 수 있는 모델을 설계하고 제안하였다. 분석 모델의 설계를 위해서 학습 데이터의 변별력 있는 특성 정보 표현을 위한 코드 분석 방법을 제안하였고, 코드 분석 결과를 입력 데이터로 이용하는 SVM의 분류 모델을 설계하였다. 본 논문에서 제안한 방법의 정확성을 검증하기 위해서 실제 사용되는 자바 어플리케이션을 벤치마크 프로그램으로 실험을 하였고, 유사 소프트웨어 분류 모델 실험 결과에서 93.7%의 정확도를 보여주었다. 실험 결과로부터 소프트웨어 코드 데이터는 소프트웨어의 특성 분류를 위한 데이터로 효과적으로 사용될 수 있으며, SVM을 이용한 유사 소프트웨어 분류 모델은 소프트웨어 분석에 적용될 수 있음을 확인할 수 있었다. 소프트웨어를 분석하기 위한 명령 코드 정보는 본 논문에서 제안한 유사 소프트웨어 분류를 위한 모델 생성 뿐 만 아니라 소프트웨어 분석을 위한 빅데이터, 인공지능, 기계 학습 등의 분야에 적용할 수 있을 것이라 기대된다.

## 감사의 글

이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2017R1D1A1B03034769).

## 참고문헌

[1] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam and B. Maqbool, A Systematic Review on Code Clone Detection, in *IEEE Access*, vol. 7, pages 86121-86144, 2019.

[2] Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh, Software clone detection: A systematic review, *Information and Software Technology*, Volume 55, Issue 7, Pages 1165-1199, July 2013.

[3] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiaofeng Wang, Effective and efficient malware detection at the end host, *Proceedings of the 18th conference on USENIX security symposium*, pages 51-366, August 2009.

[4] Andrew Walenstein and Arun Lakhotia, The Software Similarity Problem in Malware Analysis, Dagstuhl Seminar Proceedings, Duplication, *Redundancy, and Similarity in Software*, 2006.

[5] Abdullah Sheneamer and Jugal Kalita, A Survey of Software Clone Detection Techniques, *International Journal of Computer Applications*, Vol. 137, No. 10, pages 1-21, March 2016.

[6] Kevin P. Murphy, *Machine Learning: A Probabilistic Perspective*, The MIT Press. 2012.

[7] Hyun-il Lim, Similarity Analysis of Programs through Linear Regression of Code Distribution, *Journal of Digital Contents Society*, Vol. 19, No. 7, pp.1357-1363, 2018.

[8] Corinna Cortes and Vladimir Vapnik, Support-vector network, *Machine Learning* 20, pages 273-297. 1995.

[9] Steve R. Gam, Support Vector Machines for Classification and Regression, Technical Report, University of Southampton, May 1998.

[10] Christopher J.C. Bruges, A Tutorial on Support Vector Machines for Pattern Recognition, *Data Mining and Knowledge Discovery* , Vol. 2, 121-167. 1998.

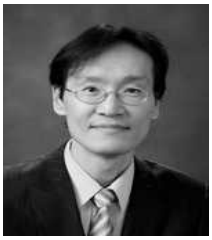
[11] Dustin Boswell, Introduction to Support Vector Machines, 2002

[12] Jaehak Yu, Hansung Lee, Younghee Im, Myung-Sup Kim, and Daihee Park, Realtime Classification of Internet Application Traffic using a Hierarchical Multiclass SVM, *KSII Transactions on Internet and Information Systems*, Vol. 4, No. 5, 2010.

[13] Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley, *The Java Virtual Machine Specification,* Java SE 8 Edition, Oracle, March 2015.

[14] Chih-Chung Chang and Chih-Jen Lin, LIBSVM: A Library for Support Vector Machines, National Taiwan University, 2013.

[15] The Ælfred XML Parser, http://saxon.sourceforge.net/aelfred.html
[16] JUnit, https://junit.org/junit4/
[17] AcornDB, https://sourceforge.net/projects/acorndb/



**임현일**(Hyun-il Lim)

1995년 : KAIST 전산학과 (공학사)
1997년 : KAIST 전산학과 (공학석사)
2009년 : KAIST 전산학과 (공학박사)

2009년～2010년: KAIST 전산학과 연구원
2010년～현 재: 경남대학교 컴퓨터공학부 부교수
※관심분야 : 소프트웨어 분석, 소프트웨어 보안, 인공 지능, 기계 학습, 소프트웨어 공학, 프로그래밍 언어 등