클래스 간 메소드 위치 결정 방법의 비교

Comparative Analysis of Determination of Method Location between Classes

정영애, 박용범
단국대학교 전자계산학과

Young-Ae Jung(yajung@dankook.ac.kr), Young B. Park(yapark@dankook.ac.kr)

요약

객체지향 패러다임에서 객체의 속성, 동작, 객체사이의 관계를 표현하는 클래스의 구성요소들에 대한 연관관계를 측정하는 응집도는 다양하게 연구되어 왔다. 리팩토링 분야에서도 개발자의 경험이나 직감에 의한 수동분석에서 자동분석에 이르기까지 다양한 연구가 제안되어 왔다. 리팩토링을 자동으로 수행하기 위해서는 수행여부를 결정짓는 객관적 판단기준에 대한 검증이 필요하다. 본 논문에서는 참조관계를 고려한 여섯 개의 메소드 위치 결정 요인과 메소드 위치에 대한 관계를 분석하기 위한 방법으로 로지스틱 회귀분석과 신경망을 사용할 것을 제안하였다. 실험 결과, 로지스틱 회귀 분석은 97%, 신경망은 90% 이상의 예측율을 보였으며, 로지스틱 회귀분석이 신경망을 이용한 방법보다 더 우수한 예측결과를 보였다. 또한 두 방법 모두 90% 이상의 예측율로 여섯 개의 메소드 위치 결정 요인이 리팩토링 무브 메소드의 객관적 판단기준으로 적용될 수 있음을 보였다.

■ 중심어 : | 로지스틱 회귀분석 | 리팩토링□메소드 위치 결정□ 신경망 | 응집도 |

Abstract

In Object-Oriented Paradigm, various cohesion measurements have been studied taking into account reference relation among components - like attributes and methods - that belong to a class. In addition, a number of methods have taken into research utilizing manual analysis, that is performed by developer's intuition and experience, and automatic analysis in refactoring field. The verification of objective criteria is demanded in order to process automatic refactoring. In this paper, we propose a method exploiting logistic regression and neural network for analysis of the relationship between six factors considering reference relation and method location among classes. Experimental results demonstrate that the logistic regression predicts the results up to 97% and the neural network predicts the outcomes up to 90%. Hence, we conclude that the logistic regression based method is more effective to predict the method location. Moreover, more than 90% of experimental results from both methods show that the six factors used in Move Method in refactoring are suitable to be used as an objective criteria.

■ keyword : | Logistic Regression□Refactoring | Determination of Method Location□Neural Network | Cohesion |

I. 서 론

객체지향 패러다임에서 소프트웨어 품질이나 복잡도 등과 같은 다양한 속성들을 측정하기 위한 척도들이 제안되었다. 모듈의 독립성은 소프트웨어 복잡도를 감소시

접수번호 : #061027-002
접수일자 : 2006년 10월 27일
심사완료일 : 2006년 11월 23일
교신저자 : 정영애, e-mail : yajung@dankook.ac.kr

켜 소프트웨어의 품질과 신뢰성을 향상시킬 수 있는 핵심적인 요소로 간주되어 왔다. 모듈의 독립성을 보장하며 신뢰성과 유지보수성이 높은 소프트웨어를 생산하기 위하여 응집도라는 속성을 다양한 방법으로 측정해 왔다[1][2]. 응집도는 모듈의 구성요소들 사이의 연관성에 대한 소프트웨어의 속성으로 소프트웨어의 응집도가 높을수록 소프트웨어에 대한 이해도나 유지보수성이 높아진다.

객체지향 패러다임에서의 모듈은 프로시저나 함수가 모듈로 정의되었던 절차적 패러다임과는 달리 속성과 메소드를 포함한 추상적 데이터의 형태인 클래스로 정의할 수 있다. 클래스는 실제 존재하는 객체를 속성(attribute)이나 어떤 행위(behavior)를 수행하기 위한 메소드, 객체사이의 관계(relationship)등으로 표현한 것이다. 클래스는 클래스 간 속성과 메소드를 참조하는 관계를 가진다. 능동적인 행위의 개념인 메소드는 자신이 속한 클래스의 속성이나 메소드뿐만 아니라, 다른 클래스의 속성이나 메소드와 참조관계를 형성한다.

본 논문에서는 두 클래스 간의 참조관계가 존재할 때 참조관계에 형성하는 메소드의 위치를 결정하기 위하여 참조관계를 가지는 메소드를 기준으로 하는 여섯 가지의 참조관계를 정의하였다. 여섯 가지 참조관계는 리팩토링 기법 중에서 두 클래스간의 참조관계를 가지는 메소드의 위치를 그대로 둘 것인지, 자신이 속한 클래스가 아닌 다른 클래스에 옮길 것인지를 판단하기 위해 사용되는 무브 메소드 기법의 적용 요인들로 정의하였다. 두 클래스 간에 참조관계들은 정의된 여섯 가지 요인을 기초로 하여 참조 집합을 형성하게 된다. 정의된 여섯 가지 요인들은 메소드의 위치를 결정짓는데 유효한 영향을 미치는가를 평가하기 위하여 채택된 로지스틱 회귀분석과 신경망 학습모델에 알맞은 데이터로 생성되어 실험데이터로 사용되었다.

이 연구에서 논의하고자 하는 사항은 다음과 같다. 첫째, 일반적으로 프로그래머의 직관적인 수동분석에 의해 이루어지는 리팩토링 분야의 요인들이 신경망 모델과 같은 기계 학습 알고리즘이나 로지스틱 회귀분석과 같은 통계기법에 적용될 때 발생될 수 있는 문제에 대하여 논하고자 한다. 둘째, 클래스의 응집도 측정에 사용되는 참조 집합으로 표현되는 무브 메소드 요인 여섯 가지에 대한 각 속성 값을 신경망과 로지스틱 회귀분석에 적용하는 것이다. 셋째, 임의의 데이터가 아닌 실제 리팩토링이 필요한 데이터에 대해 제안된 실험방법들을 테스트하여 그 실용성을 파악하고자 한다.

논문의 구성은 2장에서 본 논문의 기초가 되는 리팩토링 기법과 응집도 척도, 로지스틱 회귀분석과 신경망 등의 관련연구에 대해 살펴본다. 3장에서는 메소드 위치 결정 요인과 참조집합에 대한 정의과정을 논한다. 4장에서는 신경망 학습 모델과 로지스틱 회귀분석에 적용할 데이터의 준비과정을, 5장에서는 제안된 방법을 적용한 실험 결과를 보인다. 마지막으로 6장에서 제안한 방법에 의한 예측에 있어 해결해야 하는 문제점을 지적하고, 앞으로 연구방향에 대하여 논한다.

II. 관련 연구

1. 리팩토링

리팩토링은 프로그램의 행위(behavior)를 보전하면서 프로그램의 가독성, 구조, 성능, 유지보수성, 추상성 등을 향상시키는 좋은 방법이다[3]. 소프트웨어 개발자는 리팩토링을 사용하여 프로그램을 이해하기 쉽고 변화를 포용할 수 있도록 프로그램을 단계적으로 개선할 수 있다. 리팩토링은 프로그램의 가독성을 높일 뿐만 아니라 개발 속도를 높일 수 있도록 효율적이고 통제된 방법을 제공하여 소프트웨어의 가치를 높인다는 장점을 가진다. 따라서 많은 개발자들은 리팩토링 자동화를 위하여 지속적인 연구를 하였고, 그 결과도 매우 다양하다.

리팩토링은 1990년대 초반부터 연구되기 시작하였고 그 후 소프트웨어 개발의 주요한 분야로 연구되어왔다[3-5]. 초기의 리팩토링은 수동분석에 의존하였으나 점차 자동화하려는 노력이 있었다[6][7]. 리팩토링 자동화는 리팩토링이 필요한 후보영역을 정의하는 분야와 개발자가 정의한 후보영역에 리팩토링을 자동으로 적용하는 분야로 양분화 되었다.

리팩토링의 후보 영역을 정의 하는 분야의 대표로는 프로그램 불변점(program invariants)을 이용한 접근방

법이 있다. 이 연구는 특정 리팩토링 기법에 대한 불변점 패턴 매칭(invariants pattern matching)을 사용하여 리팩토링이 필요한 부분을 찾아 나가는 방법을 사용한다. 또 다른 연구로 요구사항의 변화에 대하여 자바 프로그램의 후보영역(candidate spot)을 원하는 디자인 패턴 구조로 변환하는 방법이 제안되었다. 이 연구에서는 후보영역을 원하는 디자인패턴으로 자동변환하기 위하여 높은 유연성을 가지는 디자인 패턴을 기초로 하는 추론(inference) 규칙이 사용되었다[6]. 개발자 정의 후보영역에 자동으로 리팩토링을 적용하는 영역에서는, 프로그램 슬라이싱(Program slicing)을 사용하여 객체지향 프로그램의 메소드를 자동으로 리팩토링하는 방식[8], 가중치를 가지는 종속 그래프(weighted dependence graph)를 이용하여 객체지향 프레임워크에서 메소드를 자동으로 리팩토링 하는 메커니즘[9]등이 제안되었다. 더불어 결합도 매트릭스(cohesion matrix)를 이용한 거리를 측정하여 시각화(visualisation)하는 툴을 개발하여 리팩토링을 돕는 연구도 제안되었다[10].

리팩토링의 자동화에 대한 연구는 꾸준히 진행되고 있으나 자동화는 극히 일부분에서 진행중이며 자동 화 또한 개발자의 경험과 직관에 의존한다. 본 논문에서는 자동화를 하기 위하여 리팩토링의 무브 메소드 기법을 기초로 하여 적용 요인을 추출하고 그 요인에 대한 일반화 가능성을 검증한다.

2. 객체의 구성요소를 고려한 응집도 메트릭스

객체지향 시스템의 기본단위인 클래스는 객체의 특징을 속성(attribute)과 행위(behavior)로 표현한 단위이다. 클래스 구성요소들 간의 연관성을 나타낸 소프트웨어의 속성을 응집도라 하며 응집도에 의해 표현되는 클래스의 독립성은 소프트웨어의 복잡도를 감소시키며 소프트웨어의 품질과 신뢰성 향상의 핵심으로 평가되어 왔다. 소프트웨어의 응집도가 높을수록 소프트웨어의 가독성이나 유지보수성이 향상된다. 이러한 응집도의 척도를 구하기 위한 다양한 연구는 모듈이나 컴포넌트까지 그 영역이 확대되었다.

클래스 안에 존재하는 메소드의 수를 중심으로 한 매트릭스로는 WMC(Weighted Methods per Class)[11]와 RFC(Response For a Class)가 있다. WMC는 클래스당 가중 메소드의 수를 나타낸 것으로 클래스에 정의된 모든 메소드에 대한 복잡도의 합으로 표현된다. RFC는 클래스가 호출하는 메소드의 수로 클래스에 대한 반응도를 나타낸다. 호출되는 메소드가 많아질수록 객체의 복잡도가 증가하여 유지보수에 더 많은 노력이 필요하게 된다. CBO(Coupling Between Object classes)는 상속성이 없는 클래스 간의 결합도를 나타낸다. 하나의 클래스가 다른 클래스의 속성이나 메소드를 사용하여 결합한 수를 측정하며 이 값이 클수록 복잡도가 증가하며 모듈의 독립성 저하로 인하여 재사용과 변경이 어려워진다.

LCOM(Lack of Cohesion in Methods)은 같은 메소드에 의해 공용되는 인스턴스의 수와 공용되지 않 는 인스턴스의 수에 대한 차이로 클래스 내부의 메소드 응집 결여도를 나타낸다[11]. 이와 유사한 개념으로 TCC(Tight Class Cohesion)와 LCC(Loose Class Cohesion)가 있다[2]. TCC는 최대 연결 수에 대한 직접연결수의 비율로 LCC는 최대 연결 수에 대한 직접연결수와 간접연결수의 합에 대한 비율로 나타낸다. 그러나 같은 값을 가지는 경우에 메소드와 인스턴스 변수들 간의 연결 패턴의 차이를 구분하지 못한다[1].

기존의 응집도에 대한 연구는 그 범위를 하나의 클래스로 한정하고 있다. 본 논문에서는 그 범위를 두 클래스로 확장하고 두 클래스 간의 참조관계를 고려하여 메소드의 최적의 위치를 찾는 메커니즘을 제안한다.

3. 로지스틱 회귀분석

로지스틱 회귀분석이란 독립변수의 양적인 변수를 이용해서 종속변수인 이변량적 변수들간의 인과관계를 추정하는 기법이며, 한 개의 종속 변수와 여러 개의 독립변수간의 상호관련성에 대해 분석하려 할 때 가장 널리 사용되는 통계기법이다[12].

본 논문은 메소드의 위치를 결정하기 위하여 이산 변수에 대한 비선형적인 관계를 규명할 수 있는 로지스틱 회귀분석을 적용하고, 메소드 위치 결정요인이 객관적인 메소드의 위치 판단 기준으로 일반화될 수 있는지에 대하여 검증한다[12].

4. 신경망

신경망은 입출력 패턴 대한 반복적인 학습 수행 한 후 학습 패턴에 포함되지 않았던 입력 패턴의 목적 값을 예측하고, 일반화하는데 효과적인 학습 알고리즘이다. 1960년에 로잔브로트에 의해 제안된 퍼셉트론으로 시작되었으나 민스키와 페퍼트에 의해 퍼셉트론의 학습식별 능력의 한계가 밝혀졌다. 그러나 1986년 룸멜하트 등에 의해 백프로파게이션(Back prop- agation) 학습 알고리즘이 제안되어 신경망에 대한 관심이 높아졌고 학습 알고리즘의 성능 향상을 위한 연구와 응용이 활발하게 이루어지고 있다[13][14].

다층 네트워크는 일반적으로 은닉층에서 시그모이드(sigmoid) 함수를 사용한다. 시그모이드 함수는 입력값이 커지게 되면 함수의 기울기가 0에 가까워지게 되므로 가중치와 biases의 값이 작아지면서 기울기(gradient)도 작은 값을 가지기 때문에 가중치와 biases의 값은 최적의 값과는 멀어지게 된다. 이러한 편파적인 함수의 크기로 인한 부작용을 제거하여 역전파 알고리즘의 속도를 개선한 RPROP (Resilient Back Propagation) 학습 알고리즘을 신경망에 적용하였다[14].

III. 메소드 위치 결정 요인 정의

1. 메소드 위치결정 요인을 위한 용어 정의

두 클래스 간에 참조관계를 가지는 메소드의 위치를 결정하기 위한 요인을 정의하기 위한 용어들은 다음과 같다.

- 소스 클래스(Source Class): 무브 메소드를 적용할 메소드가 포함되어 있는 클래스
- 타겟 클래스(Target Class): 무브 메소드를 적용할 메소드가 이동하게 될 클래스
- 속성(attribute): 무브 메소드를 적용할 메소드가 참조하는 소스클래스의 필드 (타겟 클래스의 필드를 참조하기 위하여 get/set 메소드를 사용하는 것은 타겟 클래스의 필드를 참조하는 것으로 간주함)
- 대상 메소드: 무브 메소드 기법을 적용할 메소드
- 일반 메소드(general method): 타겟 클래스의 필드를 참조하기 위한 get/set 메소드를 제외한 모든 메소드

2. 무브 메소드에 기초한 메소드 위치 결정 요인

클래스 간의 참조관계가 형성된 경우, 무브 메소드의 적용여부에 대한 확신을 가지는 것은 쉽지 않다. 파울러는 무브 메소드 적용요인을 "메소드가 자신이 정의된 클래스보다 다른 클래스의 기능을 더 많이 사용하고 있는 경우" 정의하고 있다[3]. 메소드 위치 결정 요인은 무브 메소드에 대한 파울러의 기본정의에 기초하였으며, 수동 분석시 보편적으로 적용되는 상황들을 분석하여 반영하였다.

보편적으로 적용되는 상황으로는 대상 메소드가 타겟 클래스의 필드를 소스클래스의 필드보다 많이 사용하거나, 공동으로 일하는 부분이 너무 많아서 단단히 결합되어 있는 경우 등이 있다. 이런 경우 무브 메소드를 적용함으로써 클래스의 역할을 좀 더 명확하게 구분할 수 있게 된다.

메소드의 위치를 결정하기 위한 세 가지 요인[15]은 각 요인별로 소스 클래스의 속성이나 메소드를 참조하는 경우인 직접 참조와 타겟 클래스의 속성이나 메소드를 참조하는 경우인 간접 참조 관계를 가지게 된다. 따라서 참조관계는 각 요인 별로 두 개씩의 참조관계가 성립할 수 있어 총 여섯 가지의 참조관계를 가지며 다음과 같다.

- 메소드 위치 결정 요인 1

대상 메소드가 참조하는 소스클래스의 속성의 개수와 타겟 클래스의 필드 개수의 차이를 비교한다. 일반적으로 다른 클래스의 속성에 접근하기 위해서는 간접접근자인 get/set 메소드를 통하게 되므로 간접 접근자를 통한 타겟 클래스의 속성에 대한 참조도 간접적인 필드참조로 간주한다. 따라서 결정 요인1은 대상 메소드의 속성에 대한 직접 또는 간접적인 참조관계이다.

- 메소드 위치 결정 요인 2

대상 메소드의 일반 메소드에 대한 직접 또는 간접적

인 메소드 참조횟수의 차를 비교한다. 대상 메소드에서 소스클래스의 메소드보다 타겟 클래스의 메소드에 대한 호출이 많으면 메소드를 타겟 클래스로 옮겨야 한다.

• 메소드 위치 결정 요인 3

대상 메소드가 일반 메소드에 의해 직접 또는 간접적으로 참조되는 횟수의 차를 비교한다. 타겟 클래스에 의해 호출되는 횟수가 많은 경우 타겟 클래스와 강하게 결합하여 있으므로 무브 메소드를 수행한다.

3. 메소드 위치결정 요인을 위한 참조 집합

메소드 위치 결정을 위한 참조 집합은 응집도 매트릭스를 위해 제안된 참조 그래프[16]를 기반으로 정의되었다. 참조 집합은 두 클래스에 속한 속성과 메소드들의 참조관계를 잘 표현할 수 있는 방법으로 두 클래스의 참조관계를 시각화하는데 사용되었다.

참조그래프는 각 클래스에 속한 속성과 메소드는 노드로 참조관계는 방향성 있는 에지로 표현된다.

• 참조 그래프의 정의

□ 가 소스 클래스 □ 에서 정의된 데이터 요소와 메소드의 집합이고, □ 는 타겟 클래스에서 정의된 데이터 요소와 메소드의 집합이라 하자. sm(sm∈SC)이 sa(sm∈SC)을 사용하면 sm이 sa를 참조한다고 하고, sm → sa 로 나타낸다. 메소드 참조 그래프 G(V, E)는 방향성을 가지는 그래프이며, V={SC ∪ TC}이고, 참조 집합은 식(5)와 같다.

, (5)

식(5)에서 sm과 sm'은 소스 클래스에 존재하는 서로 다른 메소드이다.

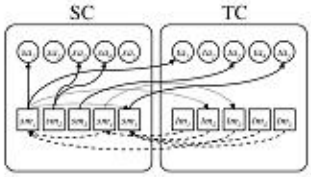

그림 1. 참조 그래프의 예

위치 결정의 대상이 되는 메소드가 소스 클래스 안의 속성이나 메소드와 가지는 참조관계는 ad, md, rd이다. 여기에서 d는 직접(direct)의 의미이다. 또한, 타겟 클래스의 구성요소와의 참조관계는 ai, mi, ri로 나타내며 여기에서 i는 간접(indirect)을 의미한다.

[그림 1]은 여섯 가지의 참조관계를 모두 가지는 경우이다. 속성에 관한 참조관계인 ad, ai는 실선으로, 메소드를 참조하는 경우인 md, mi는 둥근 점선으로, 다른 메소드에 의해 참조되는 경우인 rd, ri는 파선으로 표현한다.

• 메소드 위치 결정 참조 집합의 정의

무브 메소드를 적용할 대상 메소드인 sm(sm∈SC)이 가지는 참조 관계를 나타내는 "ad, md, ai, mi, rd, ri"들의 집합인 메소드 위치 결정 참조 집합은 식(6)와 같다.

□ □ □ □ (6) □

식(6)에서 i, j, k, l, m, n은 sm이 각 속성들과의 여섯 가지 각각의 참조관계에 대한 개수이다.

[그림 2]는 굵은 실선으로 표현된 대상 메소드 sm1에 대하여 SC와 TC의 참조관계를 하나씩 가지는 경우에 나타나는 메소드 결정 참조 집합의 예이다.

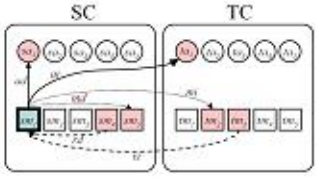

그림 2. 참조 집합의 예1

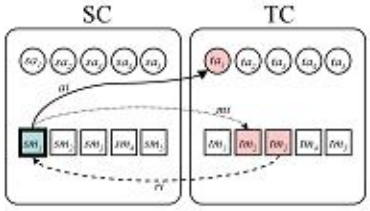

그림 3. 참조 집합의 예2

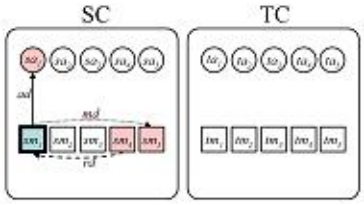

그림 4. 참조 집합의 예3

[그림 3]은 sm1이 가질 수 있는 간접적인 참조 관계만을 가지는 경우에 나타나는 경우이고, [그림 4]는 직접적인 참조 관계만을 가지는 경우이다. 여기에서, [그림 3]과 같은 경우는 간접적인 경우만 있다 하더라도 두 클래스 간의 참조 관계가 존재한다. 그러나 [그림 4]의 경우에는 직접적인 참조만 있기 때문에 SC와 TC 사이의 참조 관계는 없다. 그러나 본 논문에서는 간접적인 참조 관계가 없는 경우라고 하더라도 직접적인 참조관계가 존재하면 클래스의 참조관계가 존재하는 것으로 간주하였다.

IV. 실험을 위한 데이터 준비

1. 로지스틱 회귀분석을 위한 데이터 준비

실험을 위하여 무브 메소드를 적용할 수 있는 프로그램을 선별하여 실험에 이용하였다. 실험에 사용되는 데이터는 각 프로그램별로 참조 집합을 구성하는 요소인 ad, md, rd, ai, mi, ri 의 개수와 프로그래머가 무브 메소드를 수동분석한 결과 값으로 구성된다.

실험데이터의 ad, md, rd, ai, mi, ri는 로지스틱 회귀분석의 독립변수로 적용되기 위해 다음과 같이 데이터의 차를 구하였다.

- Factor 1 = ad의 개수 - ai의 개수
- Factor 2 = md의 개수 - mi의 개수
- Factor 3 = rd의 개수 - ri의 개수

2. 신경망 학습모델을 위한 데이터 준비

앞서 정의된 메소드의 위치를 결정짓는 적용요인들의 검증을 위해 전처리 과정을 통해 준비된 데이터에 대해 시뮬레이션을 수행하였다. 데이터는 적용요인1 ~적용요인 3이 가지는 두 가지씩의 경우에 해당되는 여섯 개의 입력 데이터와 하나의 디지트(digit)로 표현되는 목적 데이터로 구성된다. 여섯 개의 입력 데이터들의 속성 값의 범위는 실제 상황에서 다양할 수 있겠지만 본 논문에서는 0~5까지의 속성 값을 가지는 경우로 제한하였다.

학습 데이터는 각 입력 데이터의 속성 값을 0을 포함하는 자연수로 대치하고 digit-to-binary 벡터를 생성시켰다. 하나의 적용요인 속성 값은 다섯 개의 비트로 코드화 되었으며, 참조관계가 전혀 없는 경우는 0으로 코드화한다. 입력 데이터와 목적 값이 같은 패턴이 여러 번 나타난 경우는 학습에 중복적용이 되어도 학습효과에는 미치는 영향은 한 번을 적용하는 것과 다르지 않으므로, 이러한 데이터의 사용은 한번으로 제한하였다.

학습에 사용한 30×1의 구조를 가지는 입력 데이터의 수와 메소드의 위치 값은 출력 데이터의 수는 280개이다. 여기에서 입력 데이터는 하나의 메소드가 가질 수 있는 참조 관계의 수이며, 출력 데이터는 리팩토링 기법을 이용한 수동 분석 결과 값이다. 테스트 데이터의 수는 전체 데이터의 20%, 30%, 40% 에 해당되는 데이터들을 사용하였다. 20%의 경우는 다섯 가지, 30%의 경우는 세 가지, 40%의 경우는 두 가지의 경우로 구분하였다. 이때, 테스트 데이터에 해당되는 데이터들은 무작위 추출하였고, 각 경우의 테스트 데이터들의 중복은 없도록 하였다. 또한, 준비된 전체 데이터의 80%는 10-원 교차검증에 사용되었고, 나머지 20%는 테스트를 위하여 사용되었다.

V. 실험방법 및 결과

1. 로지스틱 회귀분석을 이용한 실험 및 결과

각 적용요인의 조합에 따라 그 결과가 어떻게 달라지

는지를 파악하기 위하여, 회귀분석을 위하여 준비된 세 가지 적용요인을 모두 적용한 경우, 세 가지 적용요인 중 두 가지씩 짝을 지은 세 가지의 경우, 개별적인 경우 세 가지에 대하여 분석을 수행하였다.

[표 1]은 두 개의 요인을 적용한 세 가지 경우에 대한 로지스틱 분석결과이다. [표 1]의 실험결과와 같이 실험결과는 Factor의 수와 종류에 영향을 받는다. 세 가지 요인을 모두 적용한 경우는 96.7%로 가장 낮은 예측률을 보였고, Factor 1, 2를 적용한 경우에는 예측률이 98.21%로 가장 높았다. 이러한 결과는 세 가지 적용요인들을 어떻게 조합하여 적용하느냐에 따라 다른 결과가 나올 수 있음을 보여준다.

이 결과로 미루어 Factor 2는 Factor 1이나 3보다 상대적으로 더 중요하게 작용함을 알 수 있다. 이는 메소드의 위치를 결정짓는데 대상 소드가 참조하는 필드의 개수보다는 대상 메소드가 메소드를 호출하거나 호출되는 횟수가 더 중요하다는 것을 의미한다.

본 논문에서는 리팩토링 기반연구를 통하여 메소드의 위치를 결정하는 요인들이 충분히 적용될 수 있는가에 대한 검증방법으로 이진 로지스틱 회귀분석을 사용하였다. 이 실험방식과 같이 리팩토링의 연구에 기초한 요인 추출과 그에 적합한 검증방법에 의한 검증이 제대로 이루어진다면, 추출된 요인들은 프로그램에서 멤버 변수나 멤버 메소드 등의 위치를 결정할 수 있는 일반화된 기준으로 사용할 수 있음을 보였다.

표 1. 회귀분석 실험결과

구 분	예측률
Factor 1,2,3	96.07%
Factor 1,2	98.21%
Factor 2,3	97.50%
Factor 1,3	96.43%
Factor 1	96.43%
Factor 2	97.50%
Factor 3	96.79%
평균 일치율	96.99±0.77%

2. RPROP를 이용한 실험 및 결과

학습할 데이터의 전처리 과정을 통해 준비된 학습용 데이터를 신경망에 적용시켰다. 학습율은 0.01, 최대 학습 반복수 10,000번, 최소 학습 율의 변화는 0.000001, 최대 가중치 변화는 50번, 가중치 변화 증가비율은 1.2, 가중치 변화 감소비율은 0.5이다. 학습에 사용된 시스템은 IBM PC Pentium IV 1.6GHz이다.

신경망 학습 구조에서 중간층의 뉴런 수는 실험을 통해 선택하였다. 신경망의 중간층 뉴런 수를 결정하기 위하여 중간층 뉴런의 수를 12개~24개까지 변화시켜 학습시켰고 각 학습의 경우에 대하여 학습율을 0.01 ~0.05까지 변화시켜 학습하였다. 그 결과, 중간층의 뉴런 수가 12개이면서 학습율이 0.01 또는 0.05인 경우, 뉴런 수가 14개이면서 학습율이 0.05인 경우, 뉴런 수가 16개이면서 학습율이 0.03인 경우가 학습효과가 다른 경우에 비해 비교적 높았다. 실험에서 뉴런 수가 12개이면서 학습율이 0.01인 경우에 가장 좋았기 때문에, 신경망 구조의 중간층 뉴런 수는 12개로, 학습율은 0.01로 채택하였다.

중간 층 뉴런의 수가 12개이고 학습율이 0.01인 학습 모델로 준비된 데이터로 20%의 테스트 데이터에 대한 실험을 하였다. 학습 데이터 대비 테스트 데이터의 비율이 80%: 20%의 비율로는 5가지 경우로 실험을 한 결과는 95% 이상의 예측율을 보였다.

[표 2]는 신경망 구조를 31×12×1로, 학습율은 0.01로 선택한 경우에 10-원 교차 검증의 실험 결과이다. 학습된 신경망에서 학습의 경우에는 100%, 교차검증에서는 95.18±2.52%의 학습 예측율, 그리고 테스트 데이터에 대해서는 88.46±3.65%의 예측율을 보였다.

VI. 결론 및 향후 과제

본 논문에서는 리팩토링을 자동화하기 위한 연구를 기반으로, 리팩토링 기법 중 하나인 무브 메소드의 적용여부를 판단할 수 있는 세 가지 적용요인을 정의하였다. 이렇게 정의된 적용요인들은 로지스틱 회귀분석의 독립변수로써 신경망 학습 모델의 입력 패턴으로 사용되어 분석되었다. 로지스틱 회귀분석의 결과는 평균 약 97%의 예측율을 보였고, 신경망을 이용한 분석결과에 따르면, 교차 검증에서는 약 95%의 예측율을 테스트 데이터에

대하여는 88%의 예측율을 보여 로지스틱 회귀분석이 예측율 면에 있어서 좀 더 좋은 결과를 보였다.

로지스틱 회귀분석의 경우에는 사용되는 적용요인 2를 포함하는 경우 좀 더 정확한 결과를 예측할 수 있었고, 필드 참조횟수보다 메소드 참조횟수가 메소드 위치 결정에 더 많은 영향을 준다는 것을 알 수 있다.

본 연구에서는 여러 클래스와 참조관계를 가지는 메소드를 대상으로 최적의 위치를 찾아줌으로써 소프트웨어의 응집도를 높여 안정성을 향상시키는데 목적이 있다. 개발자들의 직감에 의해 수행되는 수동분석보다는 통계기법이나 신경망과 같이 일반화에 용이한 기법에 의해 검증된 객관적인 판단기준이 보다 높은 신뢰성을 가질 수 있을 것으로 기대된다. 앞으로 여러 리팩토링기법들에서 다양한 요인을 추출하고, 다양한 경우의 데이터를 확보하여 소프트웨어의 안정성을 높일 수 있는 연구가 필요하다.

표 2. 학습 및 교차검증 예측결과 (임계값: 0.9)

case	구 분	학습 예측률
1	학 습	100.00±0.00
	교차검증	98.22±2.18
	테 스 트	92.50±4.50
2	학 습	100.00±0.00
	교차검증	95.51±3.52
	테 스 트	86.96±6.92
3	학 습	100.00±0.00
	교차검증	95.18±7.64
	테 스 트	92.14±8.23
4	학 습	100.00±0.00
	교차검증	94.66±5.51
	테 스 트	88.21±6.40
5	학 습	100.00±0.00
	교차검증	92.31±11.87
	테 스 트	82.50±6.36
평 균	학 습	100.00±0.00
	교차검증	95.18±2.52
	테 스 트	88.46±3.65

참 고 문 헌

[1] Bindu and Mehra, *A Critique of Cohesion Measures in the Object-Oriented paradigm*, Masters Thesis, Department of Computer Science, Michigan Technological university, 1997.

[2] J. M. Bieman and B. K. Kang, "Cohesion and reuse in an object-oriented paradigm," Proc. ACM Symposium on Software Reusability (SSR-95), pp.259-262, 1995.

[3] M Fowler, etc, *Refactoring Improving the Design of Existing Code*, Addison Wesley, 1999.

[4] W. G. Griswold, *Program Restructuring as an Aid to Software Maintenance*, PhD Thesis, Dept. of Computer Science & Engineering, University of Washington, 1991.

[5] W. F. Opdyke, *Refactoring object-oriented frameworks*, Ph.D.thesis, Computer Sciences Department, University of Illinois at Urbana-Champaign, 1992.

[6] S. U. Jeon, *An Approach to Automa- tically Identifying Design Structure for Applying Design Pattern*, MS. thesis of KAIST, 2003.

[7] S. Roock and A. Havenstein, *Refactoring Tags for automatic refactoring of framework dependent applications*, XP2002, 2002.

[8] K. Maruyama and K. Shima, "Automatic Method refactoring using weighted dependence graphs," Proceedings of the 21st international conference on Software engineering, 1999.

[9] K. Maruyama, "Automated Method extraction refactoring by using block-based slicing," Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, Vol.26, pp.236-245, 2001.

[10] F. Simon, F. Steinbrkner, and C. Lewerentz, "Metrics based refactoring," Proc. European Conf. Software Maintenance and Reengineering IEEE, Computer Society, pp.30-38, 2001.

[11] S. R. Chidamber and C. F. kemerer, "Towards a Metrics Suite for Object-Oriented Design," Proc. OOPSLA, '91, ACM, pp.197-211, 1991.

[12] 성웅현, *응용 로지스틱 회귀분석* 도서출판 탐진, 2001.

[13] A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis, "New globally convergent training scheme based on the resilient propagation algorithm," Nerocomputing 64, pp.253-270, 2005.

[14] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," Proceedings of the International Conference on Neural Networks, San Francisco, pp.586-591, 1993.

[15] 정영애, 박용범, "로지스틱 분석을 이용한 메소드 위치 결정 방법", 한국정보처리학회 논문지, 제 12-D권, 제7호, pp.1017-1022, 2005.

[16] D. W. Embley and S. N. Woodfield, "Accessing the quality of abstract data types written in Ada," Proc of Phoenix Conf. on Computers & Comm., pp.205-213, 1987.

저 자 소 개

정 영 애(Young-Ae Jung) 정회원

- 1995년 : 호서대학교 전자계산학과(학사)
- 2000년 : 호서대학교 대학원 전자계산학과(석사)
- 2003년 : 단국대학교 대학원 전자계산학과 박사수료

<관심분야> : SW품질평가, 패턴인식, 멀티미디어 콘텐츠

박 용 범(Young B. Park) 정회원

- 1985년 : 서강대학교 전자계산학과(학사)
- 1987년 : NY Polytechnic Univ. 대학원 전자계산학과(석사)
- 1991년 : NY Polytechnic Univ. 대학원 전자계산학과(박사)

- 1993년 ~ 현재 : 단국대학교 전자컴퓨터학부 컴퓨터과학 전공 교수

<관심분야> : Information Architecture, 패턴인식, 분산에이전트