รายงานการวิจัยฉบับสมบูรณ์

เรื่อง

การเรียนรู้รวมแบบลูกผสมสำหรับการประมาณมูลค่าซอฟต์แวร์

(Hybrid Ensemble Learning for Software Cost Estimations)

โครงการวิจัยนี้ได้รับการสนับสนุนทุนวิจัย
จาก
สำนักงานคณะกรรมการวิจัยแห่งชาติ
ปีงบประมาณ พ.ศ. ๒๕๕๔

คณะผู้วิจัย

นางสาวสุวรรณา รัศมีขวัญ	หัวหน้าโครงการวิจัย
นายกฤษณะ ชินสาร	ผู้ร่วมวิจัย
นางสาวเบญจภรณ์ จันทรกองกุล	ผู้ร่วมวิจัย
นายภูสิต กุลเกษม	ผู้ร่วมวิจัย
นางสาวสุนิสา ริมเจริญ	ผู้ร่วมวิจัย

ศูนย์วิจัย Knowledge and Smart Technology
คณะวิทยาการสารสนเทศ มหาวิทยาลัยบูรพา

บทคัดย่อ

การพัฒนาซอฟต์แวร์เป็นกิจกรรมหลักซึ่งมีค่าใช้จ่ายสูงมากในเกือบทุกองค์กรและมีการดำเนินการในหลากหลายบริบท งานวิจัยนี้ได้นำเสนอวิธีการประเมินมูลค่าของซอฟต์แวร์ ซึ่งหมายถึงกระบวนการที่ใช้ในการพยากรณ์ความพยายามที่ต้องใช้ในการพัฒนาซอฟต์แวร์ ได้มีงานวิจัยด้านการประเมินมูลค่าซอฟต์แวร์ซึ่งแบ่งออกเป็น 2 กลุ่มหลัก คือ กลุ่มวิธีการแบบพาราเมติก (Parametric Models) และกลุ่มวิธีการแบบแมชชีนเลินนิ่ง (Machine Learning) มาแล้วไม่น้อยกว่า 30 ปี แต่ปรากฏว่าผลที่ได้รับนั้นยังไม่มีประสิทธิภาพเท่าที่ควร ในงานวิจัยนี้ได้นำเสนอวิธีการแบบเอนเซมเบิ้ลนิวรอนเน็ตเวอร์ค (Ensemble Neural Network) ซึ่งหมายถึงการใช้การผสมผสานวิธีการที่เหมาะสมจากกลุ่มของวิธีการแมชชีนเลินนิ่ง (Machine Learning) เข้าด้วยกัน เพื่อแก้ข้อบกพร่องของวิธีการแบบพาราเมตริก (Parametric Models) ในประเด็นที่ไม่สามารถจัดการกับเงื่อนไขพิเศษ เช่น บุคลากร ทีมงาน และ การจับคู่ระหว่างระดับความชำนาญกับงาน นอกจากนั้นในวิธีการที่มีอยู่เดิม ถ้าหากมีความจำเป็นในการปรับแต่งค่าในภายหลังก็ต้องจัดการด้วยตนเอง

สำหรับขั้นตอนในการประเมินมูลค่าซอฟต์แวร์นั้นประกอบด้วยกระบวนการหลัก 3 ขั้นตอน กล่าวคือ 1. การวัดขนาดซอฟต์แวร์ (Software Size) ซึ่งโดยทั่วไปดำเนินการด้วย 2 วิธีการ คือ วัดจากจำนวนชุดคำสั่ง (Code size metrics) และ วัดจากจำนวนฟังก์ชัน (Functionality metrics) 2. การวัดระดับค่าความพยายาม (Software Effort) ซึ่งทั่วไปจะวัดอยู่ในรูปของระยะเวลาที่ต้องใช้ต่อคน เช่น วัดเป็นหน่วยของจำนวนคนต่อหน่วยของเวลา ซึ่งต้องพิจารณาประเด็นของภาษาที่ใช้ในการพัฒนา เครื่องมือที่ใช้ในการพัฒนา ปริมาณขององค์ประกอบที่ได้จากระบบเดิม เวลาที่สามารถใช้ในการทำงานได้ ผลผลิตต่อบุคคล ความยากง่ายของงาน เป็นต้น 3.การคิดค่าใช้จ่าย (Software Cost) ซึ่งเป็นขั้นตอนที่นำผลที่ได้จากขั้นตอนที่ 1 และ 2 มาคำนวณกับค่าแรงมาตรฐานตามความชำนาญเฉพาะทางของบุคลากรในทีม

จากขั้นตอนการประมาณมูลค่าซอฟต์แวร์ที่กล่าวไว้ข้างต้น ผู้วิจัยคาดหวังว่าการวิจัยในโครงการนี้จะนำไปสู่การเลือกวิธีการและปัจจัยที่เหมาะสมในการประมาณค่าในขั้นตอนที่ 1 และขั้นตอนที่ 2 ซึ่งจะส่งผลต่อการประมาณค่าใช้จ่ายที่ใกล้เคียงกับค่าใช้จ่ายที่จะเกิดขึ้นจริงโดยมีค่าความคลาดเคลื่อนน้อยลงจากวิธีการที่มีอยู่ในปัจจุบัน

Abstract

Software Development is a crucial activity and a high cost one in any type of organization which has different context. Software cost estimation is a process to forecast effort needed for any software development. There are two main approaches in software cost estimation. The first one is Parametric Models and the second one is Machine Learning Models. For the last 30 years, there is a great number of researches that try to come-up with a more and more efficient method. In this research report, we have proposed "Ensemble Neural Network" as an alternative approach to estimate software cost. The proposed method will help to deal with factors on human resource skill, team work, skill matching and those automatic adjustments that Parametric Methods seem to have some difficulty.

In estimating software cost, there are three steps: 1. Measuring software size; in terms of code size metrics or functionality metrics 2. Forecasting effort need; in terms of time needed for each staff in performing their tasks and 3. Estimating software cost; in terms of amount of money. According to these steps, we proposed an alternative approach that will select more proper features in steps one and two. So that the software cost estimation in step no. 3 will be improved.

สารบัญ

บทที่ 1 บทนำ

1.1 ที่มาและความสำคัญของปัญหา

การพัฒนาซอฟต์แวร์เป็นกิจกรรมที่ถูกดำเนินการในหลากหลายบริบท ได้แก่ การพัฒนาซอฟต์แวร์เพื่อการปรับปรุงกระบวนการทำงานของส่วนราชการ การพัฒนาซอฟต์แวร์เพื่อการเพิ่มขีดความสามารถในการแข่งขันได้ของภาคธุรกิจ ตลอดจนการพัฒนาซอฟต์แวร์เพื่อสร้างการเรียนรู้ทั้งในระดับหน่วยงานย่อยขององค์กร ตลอดไปจนถึงการเรียนรู้ของนักเรียน นิสิต นักศึกษา และประชาชนโดยทั่วไป ซึ่งการสร้างซอฟต์แวร์สำหรับบริบทที่แตกต่างกันจำเป็นต้องอาศัยวิธีการ เครื่องมือ เทคโนโลยี ทักษะของบุคลากร และ ค่าใช้จ่ายในการพัฒนาที่แตกต่างกัน การที่โครงการพัฒนาซอฟต์แวร์ใดจะได้รับการพิจาณาให้ดำเนินการได้นั้น ประเด็นสำคัญอันดับต้นๆ คือ การประเมินความคุ้มค่าในการลงทุน ซึ่งผู้บริหารระดับสูงใช้เป็นเกณฑ์ในการพิจารณาการให้ลำดับความสำคัญของโครงการ การจะประเมินความคุ้มค่าในการลงทุนได้นั้นจำเป็นต้องมีการประเมินมูลค่าของซอฟต์แวร์ก่อนที่จะนำไปเปรียบเทียบกับมูลค่าของผลประโยชน์ที่จะได้รับ ซึ่งผลประโยชน์ที่จะได้รับนั้นอาจอยู่ในรูปของรายรับที่เพิ่มขึ้นหรือค่าใช้จ่ายที่ลดลง ในรูปของการได้ทำตามข้อบังคับหรือกฎหมาย หรือในรูปของการสร้างหรือรักษาชื่อเสียงและภาพพจน์ของหน่วยงาน

ในงานวิจัยนี้ได้นำเสนอวิธีการประเมินมูลค่าของซอฟต์แวร์ ซึ่งการประเมินมูลค่าซอฟต์แวร์หมายถึงกระบวนการที่ใช้ในการพยากรณ์ความพยายามที่ต้องใช้ในการพัฒนาซอฟต์แวร์ ได้มีงานวิจัยด้านการประเมินมูลค่าซอฟต์แวร์ด้วยวิธีการที่หลากหลายซึ่งสามารถแบ่งออกเป็น 2 กลุ่มหลัก คือ กลุ่มวิธีการแบบพาราเมติก (Parametric Models) และกลุ่มวิธีการแบบแมชชีนเลินนิ่ง (Machine Learning) มาแล้วไม่น้อยกว่า 30 ปี แต่ปรากฏว่าผลที่ได้รับนั้นยังไม่บรรลุวัตถุประสงค์ที่ต้องการ เนื่องจากค่าใช้จ่ายที่เกิดจากการประมาณการ (Estimated Cost) ยังคงมีความแตกต่างจากค่าใช้จ่ายที่เกิดขึ้นจริง (Actual Cost) ค่อนข้างมาก ในงานวิจัยนี้ได้นำเสนอวิธีการแบบเอนเซมเบิ้ลนิวรอนเน็ตเวอร์ค (Ensemble Neural Network) ซึ่งหมายถึงการใช้การผสมผสานวิธีการที่เหมาะสมจากกลุ่มของวิธีการแมชชีนเลินนิ่ง (Machine Learning) เข้าด้วยกัน เนื่องจากผู้วิจัยมองเห็นว่าเป็นวิธีการที่จะช่วยแก้ข้อบกพร่องของวิธีการแบบพาราเมตริก (Parametric Models) ในประเด็นที่ไม่สามารถจัดการกับเงื่อนไขพิเศษ เช่น บุคลากร ทีมงาน และ การจับคู่ระหว่างระดับความชำนาญกับงาน นอกจากนั้นหากต้องการปรับแต่งค่าใดๆ ในภายหลังก็คงต้องจัดการด้วยตนเอง

สำหรับขั้นตอนในการประเมินมูลค่าซอฟต์แวร์นั้นประกอบด้วยกระบวนการหลัก 3 ขั้นตอน กล่าวคือ 1. การวัดขนาดซอฟต์แวร์ (Software Size) ซึ่งโดยทั่วไปดำเนินการด้วย 2 วิธีการ คือ วัดจากจำนวนชุดคำสั่ง (Code size metrics) และ วัดจากจำนวนฟังก์ชัน (Functionality metrics)

2. การวัดระดับค่าความพยายาม (Software Effort) ซึ่งที่นิยมทั่วไปจะวัดอยู่ในรูปของระยะเวลาที่ต้องใช้ต่อคน กล่าวคือ วัดเป็นหน่วยของจำนวนคนต่อหน่วยของเวลา ซึ่งต้องพิจารณาประเด็นของภาษาที่ใช้ในการพัฒนา เครื่องมือที่ใช้ในการพัฒนาที่สามารถสร้างองค์ประกอบสำเร็จรูปได้ (Component) ปริมาณขององค์ประกอบที่ได้จากระบบเดิม เวลาที่สามารถใช้ในการทำงานได้ ผลผลิตต่อบุคคล ความยากง่ายของงาน

3. การคิดค่าใช้จ่าย (Software Cost) ซึ่งเป็นขั้นตอนที่นำผลที่ได้จากขั้นตอนที่ 1 และ 2 มาคำนวณกับค่าแรงมาตรฐานตามความชำนาญเฉพาะทางของบุคลากรในทีม

จากขั้นตอนการประมาณมูลค่าซอฟต์แวร์ที่กล่าวไว้ข้างต้น พบว่าทั้งขั้นตอนที่ 1 และขั้นตอนที่ 2 นั้น ยังไม่มีมาตราฐานสากลที่เป็นข้อตกลงร่วมกันในการวัดค่าทั้ง 2 อย่างชัดเจน ดังนั้นผู้วิจัยคาดหวังว่าการวิจัยในโครงการนี้จะนำไปสู่การเลือกวิธีการและปัจจัยที่เหมาะสมในการประมาณค่าทั้ง 2 ชุด ดังกล่าว ซึ่งจะส่งผลต่อการประมาณค่าใช้จ่ายที่ใกล้เคียงกับค่าใช้จ่ายที่จะเกิดขึ้นจริงโดยมีค่าความคลาดเคลื่อนน้อยลงจากวิธีการที่มีอยู่ในปัจจุบัน

1.2 วัตถุประสงค์ของโครงการวิจัย

1. เพื่อศึกษาวิธีการวัดขนาดซอฟต์แวร์เพื่อใช้ในการหาปัจจัยที่ควรใช้ในการวัดขนาดซอฟต์แวร์ ที่จะนำไปสู่ขั้นตอนการประเมินการวัดค่าความพยายาม

2. เพื่อศึกษาวิธีการวัดค่าความพยายามที่จะทำให้ได้ค่าความพยายามที่ใกล้เคียงกับความเป็นจริงเพื่อนำไปคำนวณค่าใช้จ่าย

3. เพื่อศึกษาการคำนวณค่าใช้จ่ายที่เกิดจากการนำค่าความพยายามมาคำนวณกับค่าแรงมาตรฐานเพื่อให้ได้ค่าใช้จ่ายในการพัฒนาซอฟต์แวร์ที่มีค่าความคลาดเคลื่อนน้อยลงกว่าวิธีการในปัจจุบัน

4. เพื่อให้ผู้ที่สนใจสามารถนำแนวความคิดที่นำเสนอ ไปศึกษาเพื่อทำการพัฒนาหรือประยุกต์ใช้ในงานวิจัยของตนเองต่อไป

1.3 ขอบเขตของโครงการวิจัย

การวิจัยครั้งนี้มุ่งที่จะศึกษาและพัฒนาการนำเทคนิคการเรียนรู้แบบรวมมาทำการประมาณค่าของขนาดและค่าความพยายามในการผลิตซอฟต์แวร์ เพื่อนำไปคำนวณค่าใช้จ่ายในการพัฒนาซอฟต์แวร์ โดยมีขอบเขตดังนี้

1.งานวิจัยนี้จะสามารถรองรับการประมาณค่าขนาดและค่าความพยายามของซอฟต์แวร์ได้หลากหลายบริบท

2. งานวิจัยนี้สามารถรองรับการประมาณค่าขนาดและค่าความพยายามของซอฟต์แวร์ได้หลากหลายวิธีการและหลากหลายเครื่องมือที่ใช้ในการพัฒนาซอฟต์แวร์

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้ขั้นตอนวิธีในการประมาณมูลค่าซอฟต์แวร์ ที่สามารถนำไปใช้เพื่อประมาณค่าใช้จ่ายซอฟต์แวร์ได้หลากหลายบริบท

2. สามารถนำขั้นตอนวิธีข้างต้นไปพัฒนาระบบซอฟต์แวร์ เพื่อใช้ในการประมาณมูลค่าซอฟต์แวร์ที่ใกล้เคียงค่าความเป็นจริง ซึ่งจะส่งผลให้หน่วยงานสามารถบริหารทุนในการพัฒนาซอฟต์แวร์ได้อย่างเหมาะสม

3. ขั้นตอนวิธีที่นำเสนอจะสามารถใช้เป็นต้นแบบในการศึกษาขั้นสูงต่อไป

1.5 ระยะเวลาทำการวิจัยและแผนการดำเนินงานตลอดโครงการวิจัย

ดำเนินการวิจัยแบบ 2 ปีต่อเนื่อง (ปีงบประมาณ 2554-2555)

แผนการดำเนินงานปีที่ 1 (ปีงบประมาณ 2554-2555)

แผนการดำเนินงานวิจัย	เดือนที่											
	1	2	3	4	5	6	7	8	9	10	11	12
➢ การจัดเตรียมเครื่องมือ อุปกรณ์และวัสดุวิจัย	⇨											
➢ การศึกษาวิธีการประมาณมูลค่าซอฟต์แวร์	→	→	→	→	→	→						
➢ การจัดทำรายงานความก้าวหน้าครั้งที่ 1							⇨					
➢ การศึกษาการเรียนรู้รวม							→	→	→	→	→	→
➢ การจัดทำรายงานความก้าวหน้าครั้งที่ 2												⇨

แผนการดำเนินงานปีที่ 2 (ปีงบประมาณ 2555-2556)

แผนการดำเนินงานวิจัย	เดือนที่											
	1	2	3	4	5	6	7	8	9	10	11	12
➢ การศึกษาวิธีการแบบลูกผสมสำหรับการเรียนรู้รวมเพื่อใช้กับการประมาณมูลค่าซอฟต์แวร์	→	→	→	→	→	→						
➢ การจัดทำรายงานความก้าวหน้าครั้งที่ 3							⇨					
➢ พัฒนาเป็น Web-based Application สำหรับผู้ใช้ทั่วไป							→	→	→	→	→	→
➢ จัดส่งรายงาน												⇨

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ระบบโครงข่ายประสาทเทียม *(Neural Networks)*

หลายกิจกรรมในชีวิตประจำวันของเราเกี่ยวข้องกับงานปัญญาประดิษฐ์ หรือการรู้จำ ซึ่งปัญหาการรู้จำเป็นปัญหาที่มีความยากและมีความซับซ้อนมากถ้าจะพัฒนาให้เป็นระบบอัตโนมัติในเครื่องคอมพิวเตอร์ แต่ในทางตรงกันข้ามงานดังกล่าวนี้กลับสามารถดำเนินการได้โดยง่ายโดยมนุษย์ กล่าวคือ มนุษย์สามารถรู้จำหรือแยกแยะวัตถุต่างๆ ได้อย่างมากมาย ทั้งที่วัตถุดังกล่าวนั้นกำลังอยู่ในสภาวะแวดล้อมที่มีความหลากหลาย ยกตัวอย่างเช่น นายขาวจะมีความสามารถในการจำเสียงของนายดำ และไม่ว่าในดำจะโทรศัพท์มาคุยกับนายขาวจากสภาพแวดล้อมใดๆ ก็ตาม นายขาวยังคงจำเสียงนายดำได้เสมอ เป็นต้น ดังนั้น จึงเป็นเหตุผลที่สำคัญที่เราต้องพัฒนาระบบการคำนวณ (Computing system) ที่สามารถเข้าใจและเลียนแบบการทำงานของมนุษย์ระบบดังกล่าวนี้ เราเรียกว่าระบบโครงข่ายประสาท (Neural Networks)

ระบบโครงข่ายประสาทเทียม (Artificial Neural Networks: ANN) หรือ เรียกสั้น ๆ ว่า ระบบโครงข่ายประสาท (Neural Networks) เกิดมาจากแรงบันดาลใจเกี่ยวกับความต้องการเลียนแบบการทำงานของสมองมนุษย์ด้วยเครื่องคอมพิวเตอร์ เพื่อสร้างระบบคอมพิวเตอร์แบบใหม่ที่มีความสามารถในการเรียนรู้ด้วยตนเอง ซึ่งแตกต่างจากระบบคอมพิวเตอร์ในปัจจุบัน ที่ต้องทำงานตามชุดคำสั่งที่มนุษย์เขียนสั่งไว้ล่วงหน้า (Software/Program) เท่านั้น และ ก็เป็นที่ทราบทั่วไปว่าการประมวลผลของสมองมนุษย์เรานั้น มีความซับซ้อน มีความไม่เป็นเชิงเส้น และ เป็นแบบขนานอย่างมาก นอกจากนี้ สมองมนุษย์เรานั้นยังมีความสามารถทางการคำนวณ (เช่น การรู้จำ การรับรู้ และ การสั่งการในการควบคุมเครื่องจักร) ได้อย่างรวดเร็วกว่าเครื่องคอมพิวเตอร์ที่เรามีใช้อยู่ในปัจจุบันเราเป็นอย่างมาก ยกตัวอย่างเช่น การมองเห็นของมนุษย์ ซึ่งถือว่าเป็นการประมวลผลสารสนเทศแบบหนึ่ง ในระบบการมองเห็นของมนุษย์เรานั้น จะประกอบด้วยขั้นตอนการแทนข้อมูลสภาพแวดล้อมต่างๆ ที่ได้รับ (Data Representation) และ รวมไปถึงความสามารถในการติดต่อกับสภาพแวดล้อมต่างๆ เหล่านั้นด้วย กล่าวคือ ทันทีที่เรามองเห็น เราจะสามารถแทนข้อมูลที่มองเห็นได้แทบจะทันที และหลังจากนั้นเราจะยังสามารถจะรู้จำสภาพแวดล้อมต่างๆ นั้นได้ (การรู้จำ หมายถึง ความสามารถในการจดจำ และ บรรยายสิ่งต่างๆ ที่มองเห็นให้กับผู้อื่นได้ทราบ) กล่าวกันว่า สมองมนุษย์เรานั้น มีความสามารถในการรู้จำสิ่งต่างๆ ที่มองเห็นภายในเวลาประมาณ 100 - 200 มิลลิวินาที ซึ่งไม่มีเครื่องคอมพิวเตอร์ใดสามารถทำได้

แบบจำลองอย่างง่ายของระบบโครงข่ายประสาทเทียมจะมีความคล้ายกับโครงสร้างทางชีววิทยาทางสมองของมนุษย์อย่างมาก ดังแสดงในตารางที่ 2-1 และ รูปที่ 2-1 โดยมีข้อกำหนดเบื้องต้น ดังนี้

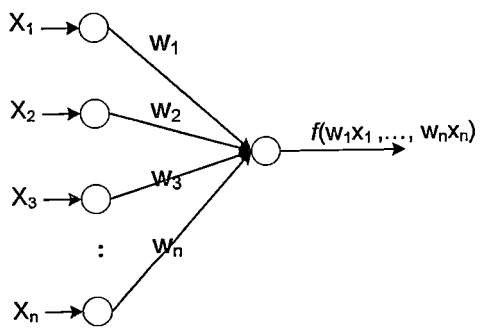
1. ตำแหน่งของค่าน้ำหนักบนโครงข่ายไม่มีความสัมพันธ์กัน

2. คำตอบของแต่ละโหนดจะมีเพียงค่าเดียว ซึ่งจะกระจายไปยังโหนดต่างๆ ที่มีการเชื่อมโยงถึง โดยตำแหน่งของการเชื่อมก็ไม่มีผลเช่นเดียวกัน

3. ข้อมูลที่เข้ามายังแต่ละโหนดในเวลาเดียวกันนั้นจะต้องคงสถานะเดิมไปจนกว่าการคำนวณของฟังก์ชัน $f(w_1x_1,\ldots,w_nx_n)$ จะเสร็จสิ้นลง

ตารางที่ 2-1 คำศัพท์เฉพาะเพื่อเทียบเคียง (ที่มา : K. Mehrotra et al., Element of Artificial Neural Network)

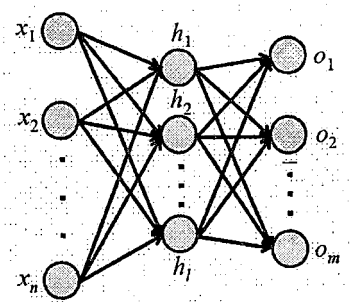
คำศัพท์เฉพาะชีววิทยาทางสมองของมนุษย์	คำศัพท์เฉพาะของโครงข่ายประสาทเทียม
Neuron	Node/Unit/Cell/Neurode
Synapse	Connection/Edge/Link
Synaptic Efficiency	Connection Strength/Weight
Firing Frequency	Node Output



รูปที่ 2-1 ระบบโครงข่ายประสาทเทียมอย่างง่าย

2.1.1 ระบบโครงข่ายประสาทเทียมแบบวิธีการแพร่กระจายย้อนกลับ

ขั้นตอนวิธีการแพร่กระจายย้อนกลับ (Back Propagation Algorithm) เป็นขั้นตอนวิธีที่ใช้ในการเรียนรู้ของเครือข่ายประสาทเทียมวิธีหนึ่งที่นิยมใช้ในโครงข่ายประสาทเทียมหลายชั้น (Multilayer Neural Networks) เพื่อใช้ในการปรับค่าน้ำหนักในเส้นเชื่อมต่อระหว่างโหนดให้เหมาะสม โดยการปรับค่านี้จะขึ้นกับความแตกต่างของค่าเอาต์พุตที่คำนวณได้กับค่าเอาต์พุตที่ต้องการ พิจารณารูปต่อไปนี้ประกอบ



รูปที่ 2-2 ตัวอย่างข่ายงานประสาทเทียมแบบหลายชั้น

ตัวอย่างในรูปด้านบนแสดงข่ายงานป้อนไปหน้าแบบหลายชั้นซึ่งประกอบไปด้วยชั้นอินพุต ชั้นฮิดเดนหรือชั้นซ่อน และชั้นเอาต์พุต ในรูปแสดงชั้นฮิดเดนเพียงชั้นเดียวแต่อาจมีมากกว่าหนึ่งชั้นก็ได้ เส้นเชื่อมจะเชื่อมต่อเป็นชั้น ๆ ไม่ข้ามชั้นจากชั้นอินพุตไปชั้นฮิดเดน ถ้ามีชั้นฮิดเดนมากกว่าหนึ่งชั้นก็เชื่อมต่อกันไป และสุดท้ายจากชั้นฮิดเดนไปชั้นเอาต์พุต

ในการปรับค่าน้ำหนักโดยขั้นตอนวิธีการแพร่กระจายย้อนกลับนั้น เราต้องนิยามค่าผิดพลาดสำหรับการเรียนรู้ของข่ายงาน $MSE(\vec{w})$ จากนั้นจะหาค่าน้ำหนักที่ให้ค่าผิดพลาดต่ำสุด นิยามค่าผิดพลาดดังนี้

$$MSE(\vec{w}) = \frac{1}{2}\sum_{p\in P}\sum_{k\in outputs}(d_{p,k} - o_{p,k})^2 \qquad ...(1)$$

โดยที่ *outputs* คือ เซตของเอาต์พุตโหนดในข่ายงานประสาทเทียม $d_{p,k}$ และ $o_{p,k}$ เป็นค่าเอาต์พุตเป้าหมายและเอาต์พุตที่ได้จากข่ายงานประสาทเทียมตามลำดับของเอาต์พุตโหนดที่ k ของตัวอย่างที่ p ขั้นตอนการแพร่กระจายย้อนกลับจะค้นหาค่าน้ำหนักที่ให้ค่าผิดพลาดกำลังสองเฉลี่ยต่ำสุด

ขั้นตอนของ Back-propagation Algorithm มีดังนี้

Algorithm Backpropagation;

Start with randomly chosen weights;

while MSE is unsatisfactory

and computational bounds are not exceeded, do

for each input pattern x_p, $1 \leq p \leq P$,

Compute hidden node inputs ($net_{p,j}^{(1)}$);

Compute hidden node outputs ($x_{p,j}^{(1)}$);

Compute inputs to the output nodes ($net_{p,k}^{(2)}$);

Compute the network outputs ($o_{p,k}$);

Modify outer layer weights:

$$\Delta w_{k,j}^{(2,1)} = \eta(d_{p,k} - o_{p,k})\mathcal{S}'(net_{p,k}^{(2)})x_{p,j}^{(1)}$$

Modify weights between input & hidden nodes:

$$\Delta w_{j,i}^{(1,0)} = \eta \sum_k \left((d_{p,k} - o_{p,k})\mathcal{S}'(net_{p,k}^{(2)})w_{k,j}^{(2,1)}\right)\mathcal{S}'(net_{p,j}^{(1)})x_{p,i}$$

end-for

end-while.

Note: if S is a logistic function, then
$S'(x) = S(x)(1 - S(x))$

2.2 การเรียนรู้ด้วยระบบโครงข่ายประสาทเทียมแบบ

การเรียนรู้ด้วยระบบโครงข่ายประสาทเทียมแบบผสม ผสม (Neural Network Ensemble Learning) เป็นวิธีการเรียนรู้ที่นำผลลัพธ์จากการใช้วิธีการเรียนรู้หลายๆ แบบมารวมกัน เพื่อเพิ่มความแม่นยำของผลลัพธ์ ซึ่งมักจะนำมาใช้ในการเพิ่มประสิทธิภาพของการจำแนกข้อมูล การพยากรณ์ผลลัพธ์ การประมาณค่าของฟังก์ชัน เป็นต้น การรวมผลลัพธ์ที่ได้มาจากขั้นตอนวิธีการเรียนรู้ที่มากกว่า 1 แบบ ทำให้เพิ่มความถูกต้องว่าผลการตัดสินใจจากขั้นตอนวิธีเหล่านั้นให้ความมั่นใจได้มากขึ้น ตัวอย่างเช่น ในปัญหาการจำแนกข้อมูล มีงานวิจัยจำนวนมากที่นำเสนอขั้นตอนวิธีในการแก้ปัญหานี้ ในการใช้อัลกอริทึมที่ต่างกัน ผลลัพธ์ที่ได้จากการเรียนรู้ในแต่ละครั้งก็มักจะให้ผลลัพธ์ที่แตกต่างกันด้วย ทำให้เกิดปัญหาตามมาว่า ถ้าในกรณีที่ตัวเรียนรู้แต่ละตัวให้ผลไม่ตรงกัน ควรจะเลือกผลลัพธ์ใดไปเป็นคำตอบ

จากปัญหาดังกล่าว แนวความคิดในการสร้างวิธีเรียนรู้แบบ Ensemble จึงเกิดขึ้นเพื่อเพิ่มประสิทธิภาพของคำตอบด้วยการรวมผลลัพธ์จากหลายๆ การเรียนรู้ ซึ่งขั้นตอนหลักของการเรียนรู้แบบ ensemble จะแบ่งเป็น 2 ขั้นตอนคือ

1. ทำการเรียนรู้โดยใช้ตัวเรียนรู้หลายๆ ตัว ซึ่งตัวเรียนรู้เหล่านี้อาจจะเป็นอัลกอริทึมเดียวกันที่ใช้ parameter ต่างกัน หรือว่าจะเป็นคนละอัลกอริทึมกันก็ได้

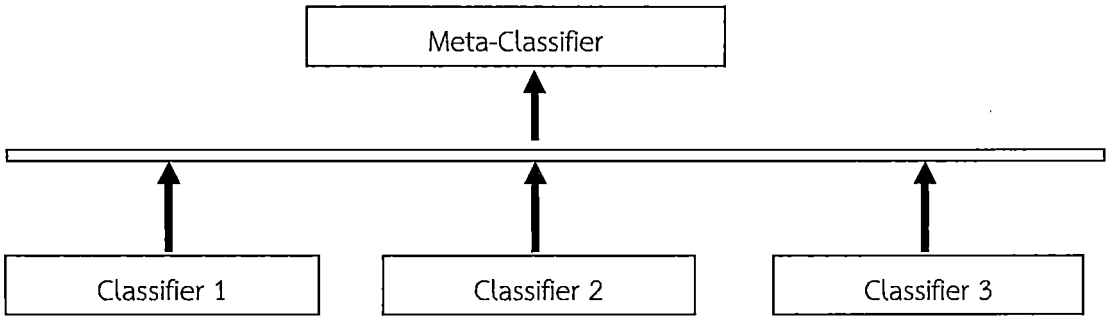
2. นำผลลัพธ์ที่ได้จากการเรียนรู้ที่มาจากหลายๆ แบบมารวมกัน แล้วทำให้จากหลายผลลัพธ์กลายเป็นคำตอบเดียวที่เหมาะสมที่สุด ซึ่งวิธีการรวมก็สามารถทำได้หลายวิธี เช่น อาจจะใช้การเฉลี่ยหรือการหาคะแนนนิยม เป็นต้น

ตัวอย่างของวิธีการเรียนรู้แบบผสม

1. Boosting (Schapire, 1990) เป็นการทำ ensemble โดยใช้ผลลัพธ์จากตัวเรียนรู้ตัวที่ให้ผลลัพธ์ยังไม่ค่อยดีเพื่อสร้างตัวเรียนรู้ที่ดีกว่า โดยการให้น้ำหนักของผลลัพธ์ที่เรียนรู้ได้ แล้วส่งให้ตัวเรียนรู้ตัวใหม่ใช้ข้อมูลนั้นในการเพิ่มประสิทธิภาพในการเรียนรู้ต่อไป โดยค่าน้ำหนักนี้จะเพิ่มขึ้นถ้าตัวเรียนรู้ไม่สามารถให้ผลลัพธ์ที่ถูกต้องได้ และค่าน้ำหนักจะลดลงถ้าตัวเรียนรู้สามารถเรียนรู้ข้อมูลที่ถูกสอนได้ถูกต้อง สาเหตุที่ต้องมีการกำหนดค่าน้ำหนักเช่นนี้เนื่องจากวิธีนี้ต้องการมุ่งเน้นไปที่การเรียนรู้ตัวอย่างยากที่ไม่สามารถให้ผลลัพธ์ที่ถูกต้องได้จากการเรียนรู้ในครั้งก่อนหน้า ขั้นตอนการเรียนรู้แบบนี้จะทำซ้ำหลายๆ รอบจนได้ตัวเรียนรู้ที่ให้ผลลัพธ์ที่ดีที่สุดออกมา วิธีการนี้เป็นวิธีการที่เสนอมาในยุคแรกๆ และเป็นที่นิยมใช้มาจนถึงทุกวันนี้ โดยอัลกอริทึมที่ถูกพัฒนาขึ้นมาในภายหลังและเป็นที่นิยมใช้กันในวงกว้างคือ อัลกอริทึม AdaBoost หรือ Adaptive boosting (Freund and Schapire, 1996)

2. Bagging (Breiman, 1996) วิธีการ Bagging หรือ Boostrap Aggregating จะใช้ข้อมูลสอนหลายๆ ชุดที่มาจากการสุ่มแบบใส่กลับคืน เพื่อสร้างชุดข้อมูลสอนที่แตกต่างกัน และข้อมูลสอนเหล่านี้จะถูกนำมาใช้ในการเรียนรู้ของตัวเรียนรู้แต่ละตัวแล้วจะนำผลลัพธ์ที่ได้มาจากตัวเรียนรู้แต่ละตัว ซึ่งผลลัพธ์อาจจะได้ออกมาเหมือนหรือต่างกัน มาทำการเฉลี่ยหรือ vote โดยกำหนดค่าน้ำหนักของผลลัพธ์ที่มาจากแต่ละตัวเรียนรู้ด้วยน้ำหนักที่เท่ากัน

3. Stacked generalization (Wolpert, 1992) เป็นวิธีการที่นำหลักการของ Meta Learner เข้ามาใช้ โดยการแบ่งตัวเรียนรู้ออกเป็นลำดับชั้น ตัวเรียนรู้ที่เป็น Master Model จะทำหน้าที่เป็นตัวตัดสินใจเพื่อให้คำตอบสุดท้ายออกมา ซึ่งการตัดสินใจนี้จะขึ้นอยู่กับผลลัพธ์จากตัวเรียนรู้ที่เหลือ ซึ่งลักษณะการทำงานแสดงดังรูปที่ 2-3



รูปที่ 2-3 การทำงานของการเรียนรู้แบบผสมแบบ Stacked generalization

การประมาณต้นทุนการพัฒนาซอฟต์แวร์ (Estimating Software Development Costs)

วิธีการในการประมาณมูลค่าซอฟต์แวร์แบ่งออกเป็นสองชนิดหลักได้แก่ วิธีการแบบ Non-algorithmic ซึ่งหมายถึงการวิธีการคิดที่ไม่สามารถระบุเป็นขั้นตอนวิธีได้อย่างชัดเจน และ วิธีการแบบ Algorithmic ซึ่งสามารถระบุขั้นตอนวิธีได้อย่างชัดเจน โดยสร้างเป็นตัวแบบอัลกอริทึมที่มีพื้นฐานในการใช้สูตรทางคณิตศาสตร์ บางตัวแบบใช้วิธีการคำนวณเชิงสถิติ เช่น การคำนวณหาค่าเฉลี่ยและค่าเบี่ยงเบนมาตรฐาน และบางตัวแบบมีพื้นฐานมาจากตัวแบบการถดถอย สมการ-อนุพันธ์ เป็นต้น ตัวแบบเหล่านี้ไม่สามารถนำมาใช้งานได้ทันทีแต่จำเป็นต้องมีการปรับค่าและประยุกต์ให้เข้ากับสถานการณ์เฉพาะที่เพื่อเพิ่มความเที่ยงตรงของตัวแบบ

Non-algorithmic Methods

วิธีนี้ใช้ข้อมูลและความรู้จากผู้เชี่ยวชาญในการประมาณมูลค่าซอฟต์แวร์ ตัวแบบการประมาณมูลค่าซอฟต์แวร์ โดยไม่มีการนำตัวแบบเชิงคณิตศาสตร์มาเป็นพื้นฐานแต่อย่างใด วิธีการประมาณค่าซอฟต์แวร์ที่จัดอยู่ในกลุ่มนี้ได้แก่

วิธี Analogy Costing วิธีนี้จำเป็นต้องมีโครงการที่สมบูรณ์ก่อนหน้านี้อย่างน้อย 1 โครงการที่มีลักษณะคล้ายคลึงกับโครงการใหม่เพื่อใช้ในการหาค่าใช้จ่ายที่เกิดขึ้นจริง การประมาณค่าใช้จ่ายโดยอาศัยความคล้ายคลึงสามารถทำได้ทั้งในระดับภาพรวมหรือส่วนใดส่วนหนึ่งของโครงการ ระดับภาพรวมมีประโยชน์ คือ องค์ประกอบทุกด้านของระบบถูกนำเข้ามาพิจารณา ขณะที่ระดับย่อยมีประโยชน์ในการให้รายละเอียดในความคล้ายคลึงและความแตกต่างระหว่างโครงการใหม่กับโครงการเก่าได้ดีกว่า จุดเด่นของวิธีนี้คือ การประมาณอยู่บนพื้นฐานของประสบการณ์ที่เคยเกิดขึ้นจริง อย่างไรก็ตาม โครงการที่เดิมเคยทำมาแล้วมักจะไม่ชัดเจนพอที่จะใช้เป็นตัวแทนของเงื่อนไข สภาพแวดล้อม และการทำงานต่าง ๆ ของโครงการใหม่ทั้งหมด

วิธี Expert Judgment วิธีนี้เกี่ยวข้องกับผู้เชี่ยวชาญอย่างน้อยหนึ่งคนขึ้นไป ผู้เชี่ยวชาญจะใช้วิธีและประสบการณ์ของตนเองในการประมาณค่า กลไกในการดำเนินการของวิธีการนี้ไม่ว่าจะเป็นวิธีการแบบ Delphi หรือ หลักการแบบ PERT จะใช้ในการแก้ปัญหาความไม่แน่นอนในการประมาณค่า

วิธีการ Delphi มีขั้นตอนโดยย่อดังนี้

1. ผู้ประสานงานส่งมอบข้อมูลทางเทคนิคและแบบฟอร์มให้แก่ผู้เชี่ยวชาญแต่ละคนเพื่อบันทึกการประมาณค่า
2. ผู้เชี่ยวชาญแต่ละคนกรอกข้อมูลลงในแบบฟอร์มโดยอิสระหากมีข้อสงสัยผู้เชี่ยวชาญสามารถถามรายละเอียดกับผู้ประสานงานได้
3. ผู้ประสานงานทำการสรุปข้อมูลการประมาณทั้งหมดที่ได้มาจากผู้เชี่ยวชาญ (ด้วยวิธีหาค่ากลาง เช่น ค่าเฉลี่ย หรือ ค่ามัธยฐาน) ลงบนแบบฟอร์มเพื่อให้ผู้เชี่ยวชาญแต่ละคนประมาณค่าอีกครั้งและให้เหตุผลประกอบ
4. ทำซ้ำขั้นตอนที่ 2 และ 3 หลาย ๆ ครั้งตามความเหมาะสม กล่าวคือ ค่าที่ได้ค่อนข้างคงที่ หรือมีการเปลี่ยนแปลงค่าเพียงเล็กน้อย

Boehm และ Fahquhar (Boehm, 1981) เป็นผู้เสนอให้มีการปรับปรุงแก้ไขวิธี Delphi ให้มีประสิทธิภาพมากขึ้น โดยก่อนการประมาณค่า ผู้ประสานงานและผู้เชี่ยวชาญต้องเข้าร่วมประชุมเพื่ออภิปรายประเด็นต่างๆของการประมาณค่า และในขั้นตอนที่ 3 ผู้เชี่ยวชาญไม่จำเป็นต้องให้เหตุผลของการประมาณ แต่จะมีการนัดประชุมเพื่อให้ผู้เชี่ยวชาญอภิปรายถึงประเด็นต่างๆในกรณีที่มีความแตกต่างของการประมาณค่าเกิดขึ้น

วิธี Parkinson ในวิธีนี้มีแนวคิดที่ว่า “งานจะถูกขยายออกไปจนเต็มความสามารถของทรัพยากรที่มีอยู่” มูลค่าจะถูกกำหนด (ไม่ใช่เกิดจากการประมาณค่า) โดยทรัพยากรที่มีอยู่ทั้งหมด เช่น ถ้าซอฟต์แวร์ต้องส่งมอบภายใน 12 เดือน และมีบุคลากร 5 คน แต่ผลที่ได้จะถูกประมาณค่าเป็น 60 คน-เดือน ซึ่งในกรณีนี้ไม่สามารถเกิดขึ้นได้จริง อย่างไรก็ตามแม้ว่าบางครั้งการประมาณค่าจะให้ผลการประมาณค่าที่ดี แต่ไม่แนะนำให้ใช้เพราะอาจจะให้การประมาณค่าไม่สมจริง และไม่เป็นวิธีการที่ดีในทางปฏิบัติในแง่ของวิศวกรรมซอฟต์แวร์

วิธี Price-to-win มูลค่าซอฟต์แวร์ถูกประมาณค่าด้วยราคาที่ดีที่สุด การประมาณค่าขึ้นอยู่กับงบประมาณของลูกค้ามากกว่าฟังก์ชันการทำงานของซอฟต์แวร์ ตัวอย่างเช่น ถ้าการประมาณค่าที่สมเหตุสมผลสำหรับโครงการหนึ่งมีมูลค่าเท่ากับ 100 คน-เดือนแต่ลูกค้าสามารถจ่ายได้ 60 คน-เดือน ผู้ประมาณค่าจะทำการปรับค่าการประมาณค่าให้เท่ากับ 60 คน-เดือน เพื่อให้ได้รับงานนี้ ในทาง

ปฏิบัติแล้ววิธีการนี้ไม่ใช่วิธีการที่ดีเนื่องจากอาจเป็นเหตุให้เกิดความล่าช้าในการส่งมอบหรือทำให้ทีมงานพัฒนาต้องทำงานล่วงเวลามากขึ้น

วิธี Bottom-up วิธีการนี้แต่ละองค์ประกอบของระบบซอฟต์แวร์จะถูกทำการประมาณค่าแยกกันและผลลัพธ์ของแต่ละส่วนจะถูกนำมารวมเพื่อทำการประมาณค่าระบบทั้งหมด หลักการที่สำคัญของวิธีการนี้ คือ จะต้องมีการออกแบบเบื้องต้นไว้แล้ว เพื่อเป็นตัวบ่งชี้ว่าระบบจะถูกแบ่งออกเป็นองค์ประกอบย่อยที่แตกต่างกันได้อย่างไรบ้าง

วิธี Top-down วิธีการนี้ตรงข้ามกับวิธี Bottom-up การประมาณมูลค่าทั้งหมดของระบบได้มาจากคุณสมบัติโดยรวม มูลค่ารวมสามารถแบ่งย่อยไปยังองค์ประกอบย่อยได้ วิธีการนี้เหมาะสมสำหรับการประมาณมูลค่าในระยะเริ่มต้นของโครงการ

Algorithmic methods

วิธีนี้มีพื้นฐานจากตัวแบบเชิงคณิตศาสตร์ซึ่งทำการประมาณค่าด้วยฟังก์ชั่นของตัวแปรต่างๆ ที่เป็นปัจจัยหลักของมูลค่าที่จะเกิดขึ้น ตัวแบบอัลกอริทึมจะอยู่ในรูปของ

$$Effort = f(x1,x2,...,xn)$$

โดยที่ *{x1,x2,...,xn}* ใช้แทน *cost factors* วิธีการอัลกอริทึมที่มีอยู่ในปัจจุบันมีข้อแตกต่างอยู่สองจุด คือ รูปแบบของฟังก์ชั่น *f* และการเลือก *cost factors* ตลอดสามทศวรรษที่ผ่านมาตัวแบบการประมาณมูลค่าซอฟต์แวร์เชิงปริมาณได้มีการพัฒนาอย่างต่อเนื่อง เริ่มจากตัวแบบ Empirical ไปจนถึงตัวแบบ Analytical (Putnam, 1978; Parr, 1980, Cantone et al., 1986)โดยตัวแบบ Empirical จะใช้ข้อมูลจากโครงการก่อนหน้าเพื่อประเมินมูลค่าโครงการปัจจุบันและได้สูตรพื้นฐานมาจากการวิเคราะห์ระบบฐานข้อมูลเฉพาะทางที่มีอยู่ ตัวอย่างเช่น COCOMO ของ Boehm (Boehm, 1981) ในขณะที่ตัวแบบ Analytical นั้นจะใช้สูตรที่มีพื้นฐานมาจากสมมุติฐานรวม เช่น อัตราการแก้ปัญหาของนักพัฒนาและจำนวนปัญหาที่มีอยู่

ตัวแบบการประมาณมูลค่าซอฟต์แวร์ด้วยวิธีอัลกอริทึม

- Linear models อยู่ในรูปแบบ

$$Effort = a_0 + \sum_{i=1}^{n} a_i x_i$$

โดยที่ *a1,..., an* ถูกเลือกให้เหมาะสมกับข้อมูลของโครงการที่สมบูรณ์ งานของ Nelson ได้ใช้ตัวแบบนี้

- Multiplicative Models อยู่ในรูปแบบ

$$Effort = a_0 \prod_{i=1}^{n} a_i^{x_i}$$

โดยที่ *a1,..., an* ถูกเลือกให้เหมาะสมกับข้อมูลของโครงการที่สมบูรณ์ งานของ Walston-Felix ได้ใช้ตัวแบบนี้ และ *xi* มีค่าที่เป็นไปได้ 3 ค่าคือ -1, 0, +1 ตัวแบบ Doty จัดอยู่ในตัวแบบนี้เช่นเดียวกันโดย *xi* มีค่า 2 ค่าคือ 0 และ +1

- Power function models อยู่ในรูปแบบ

  Effort = $a \times S^b$

โดยที่ *S* เป็น code-size และ *a, b* เป็นฟังก์ชั่นของ cost factor อื่น ตัวแบบชนิดนี้ที่มีชื่อเสียงและเป็นที่นิยมได้แก่ COCOMO และ Putnam & SLIM

- Model Calibration โดยการใช้ linear regression ตัวแบบที่กล่าวมาข้างต้นทั้งหมดไม่ได้พิจารณาถึงสถานการณ์ท้องถิ่น อย่างไรก็ตามสามารถปรับ cost factors โดยใช้ข้อมูลท้องถิ่นและวิธีการถดถอยเชิงเส้น
- Discrete Models อยู่ในรูปแบบของตารางซึ่งโดยปกติจะเกี่ยวข้องกับ ความพยายาม (Effort) ระยะเวลา ความยาก และ cost factors อื่นๆ ตัวแบบในกลุ่มนี้ได้แก่ตัวแบบ Aron (Aron, 1969) ตัวแบบ Wolverton (Wolverton, 1974) และ ตัวแบบ Boeing (Black et al., 1977) ตัวแบบเหล่านี้ได้รับความนิยมในยุคแรกๆเนื่องจากใช้งานง่าย
- Price-S เป็นตัวแบบที่ประมาณมูลค่าซอฟต์แวร์กรรมสิทธิ์พัฒนาและบำรุงรักษาโดย RCA, New Jersey (Park, 1988) เริ่มจากการประมาณขนาด ชนิดและความยากของโครงการ ตัวแบบคำนวณมูลค่าโครงการและตารางการทำงาน
- SoftCost ใช้ขนาด ความพยายาม และระยะเวลาเพื่อระบุความเสี่ยงโดยใช้รูปแบบของการกระจายความน่าจะเป็นของ Rayleigh (Tausworthe, 1981) ตัวแบบนี้มี Heuristics เพื่อใช้เป็นแนวทางให้กับผู้ประมาณในการจัดการกับเทคโนโลยีใหม่ๆและความสัมพันธ์ที่ซับซ้อนระหว่างพารามิเตอร์ต่างๆที่เกี่ยวข้อง

ตารางที่ 2-2 การแบ่งประเภทของตัวแบบอัลกอริทึม

ตัวแบบอัลกอริทึม					
	Linear	Multiplicative	Power function	Discrete	Others
Empirical	Nelson	Walston-Felix Herd et al.	COCOMO(s)	Aron Boeing Wolverton	Price-S
Analytical			Putnam		SoftCost

นอกจากขนาดของซอฟต์แวร์ยังมี Cost Factors ประเภทอื่นๆ ซึ่งกลุ่มของ Cost Factors ที่ใช้กันอย่างกว้างขวางและถูกเสนอและใช้โดย Boehm et al. ในตัวแบบ COCOMO II (Boehm et al. ,1996) cost factors เหล่านี้แบ่งออกเป็น 4 ชนิด คือ ปัจจัยทางด้านผลิตภัณฑ์ (Product factors) ปัจจัยทางด้านคอมพิวเตอร์ (Computer factors) ปัจจัยส่วนตัว (Personal factors) และปัจจัยทางด้านโครงการ (Project factors)

2.3 การทบทวนวรรณกรรม/สารสนเทศที่เกี่ยวข้อง

B. Tirimula Rao (Rao, 2009) และคณะ นำเสนองานวิจัยเรื่อง A Novel Neural Network Approach for Software Cost Estimation Using Functional Link Artificial Neural Network (FLANN) โดยในงานวิจัยนี้นำเสนอระบบโครงข่ายประสาทเทียมสำหรับการพยากรณ์มูลค่าของซอฟต์แวร์ซึ่งช่วยลดความซับซ้อนในการคำนวณและสามารถพยากรณ์มูลค่าซอฟต์แวร์แบบออนไลน์ได้ สถาปัตยกรรมของโครงข่ายประสาทเทียมที่นำเสนอในงานวิจัยนี้เป็นแบบที่ไม่มีชั้นซ่อน (Hidden Layer) กระบวนการสอนจะไม่ได้ใช้แบบแพร่ย้อนกลับแบบสมบูรณ์ คำตอบที่ใช้ในการเปรียบเทียบของการสอนจะใช้ค่าที่ได้จากการประมาณมูลค่าซอฟต์แวร์ด้วยวิธี COCOMO (Constructive Cost Model) จากผลการทดลองพบว่า วิธีการที่นำเสนอให้ผลการทดลองที่มีประสิทธิภาพสูงกว่าวิธีการโครงข่ายประสาทเทียมทั่วไป

N. Tadayon (Tadayon, 2005) นำเสนองานวิจัยเรื่อง Neural Network Approach for Software Cost Estimation โดยใช้ระบบโครงข่ายประสาทเทียมแบบเปอร์เซปตรอนหลายชั้น (Multi Layer Perceptron) ซึ่งมีชั้นซ่อนได้มากกว่าหนึ่งชั้น โดยใช้การคำตอบของการสอนที่ได้จากวิธี COCOMO การปรับค่าน้ำหนักของโครงข่ายประสาทเทียมจะใช้วิธี Gradient Descent Learning Rule

K. Vinay Kumar และคณะ (Kumar, 2008) นำเสนองานวิจัยเรื่อง Software Development Cost Estimation using Wavelet Neural Networks งานวิจัยนี้นำเสนอการประยุกต์ใช้ Wavelet Neural Networks สำหรับการพยากรณ์มูลค่าซอฟต์แวร์ โดย K. Vinay Kumar แบ่งการทำงานออกเป็นสองขั้นตอนใหญ่ๆ คือ 1.) การใช้ฟังก์ชัน Morlet และ ฟังก์ชันเกาส์เป็นฟังก์ชันสำหรับการทราสเฟอร์และ 2.) นำเสนอขั้นตอนวิธีสำหรับการหาค่าThreshold ที่สามารถยอมรับได้สำหรับการสอนของ Wavelet Neural Networks (TAWNN) จากผลการศึกษาเปรียบเทียบผลระหว่าง WNN กับวิธี MLP, RBF, MLR, DENFIS และ SVM พบว่า WNN ที่นำเสนอให้ผลการทดลองที่ดีกว่าบางวิธีการที่กล่าวไว้ข้างต้น

Y. Kultur และ คณะ (Kultur, 2009) นำเสนองานวิจัยเรื่อง Ensemble of Neural Networks with Associative Memory (ENNA) for Estimating Software Development Costs ในงานวิจัยนี้นำเสนอวิธีการเรียนรู้แบบผสมโดยใช้เทคนิค Associative Memory สำหรับการประมาณมูลค่าซอฟต์แวร์ ผลจากการนำเสนอในงานวิจัยนี้ทำให้ได้วิธีการที่สามารถเพิ่มประสิทธิภาพในการปรับเทียบค่าอย่างอัตโนมัติแทนการจัดการด้วยมือ นอกจากนี้ วิธีการที่นำเสนอยังให้ค่าความถูกต้องที่ดีกว่าวิธีโครงข่ายประสาทเทียมแบบมาตรฐาน

บทที่ 3 วิธีดำเนินการวิจัย

ในงานวิจัยนี้ได้นำเสนอวิธีการประเมินมูลค่าของซอฟต์แวร์ที่ใช้กลุ่มวิธีการแบบแมชชีนเลินนิ่ง (Machine Learning) แบบลูกผสมนิวรอนเน็ตเวอร์ค (Ensemble Neural Network) ซึ่งหมายถึงการใช้การผสมผสานวิธีการที่เหมาะสมจากกลุ่มของวิธีการแมชชีนเลินนิ่ง (Machine Learning) เข้าด้วยกัน

สำหรับขั้นตอนในการประเมินมูลค่าซอฟต์แวร์นั้นประกอบด้วยกระบวนการหลัก 3 ขั้นตอน กล่าวคือ

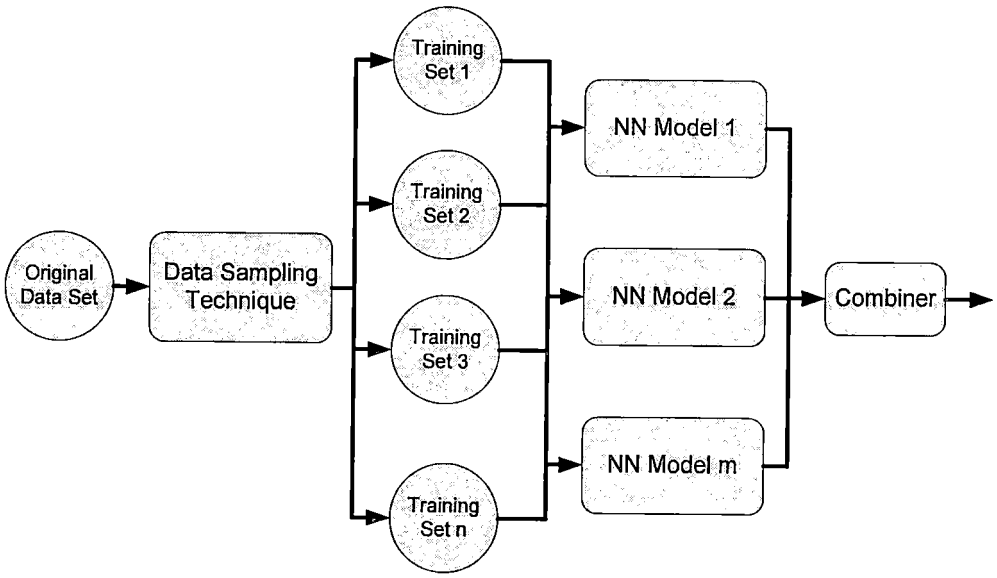
1. การวัดขนาดซอฟต์แวร์ (Software Size) ซึ่งโดยทั่วไปดำเนินการด้วย 2 วิธีการ คือ วัดจากจำนวนชุดคำสั่ง (Code size metrics) และ วัดจากจำนวนฟังก์ชัน (Functionality metrics)

2. การวัดระดับค่าความพยายาม (Software Effort) ซึ่งที่นิยมทั่วไปจะวัดอยู่ในรูปของระยะเวลาที่ต้องใช้ต่อคน เช่น วัดเป็นหน่วยของจำนวนคนต่อหน่วยของเวลา ซึ่งต้องพิจารณาประเด็นของภาษาที่ใช้ในการพัฒนา เครื่องมือที่ใช้ในการพัฒนาที่สามารถสร้างองค์ประกอบสำเร็จรูปได้ (Component) ปริมาณขององค์ประกอบที่ได้จากระบบเดิม เวลาที่สามารถใช้ในการทำงานได้ผลผลิตต่อบุคคล ความยากง่ายของงาน

3. การคิดค่าใช้จ่าย (Software Cost) ซึ่งเป็นขั้นตอนที่นำผลที่ได้จากขั้นตอนที่ 1 และ 2 มาคำนวณกับค่าแรงมาตรฐานตามความชำนาญเฉพาะทางของบุคลากรในทีม

จากขั้นตอนการประมาณมูลค่าซอฟต์แวร์ที่กล่าวไว้ข้างต้น พบว่าทั้งขั้นตอนที่ 1 และขั้นตอนที่ 2 นั้น ยังไม่มีมาตรฐานสากลที่เป็นข้อตกลงร่วมกันในการวัดค่าทั้ง 2 อย่างชัดเจน ในงานวิจัยนี้จึงนำเสนอวิธีการเรียนรู้แบบลูกผสมเพื่อการประมาณมูลค่าซอฟต์แวร์ โดยโครงข่ายประสาทเทียมสำหรับการเรียนรู้ในครั้งนี้จะเลือกใช้ Multi Layer Perceptron ที่มีสถาปัตยกรรมแตกต่างกัน ใช้การปรับค่าน้ำหนักแบบแพร่กระจายย้อนกลับ และผลลัพธ์จากตัวเรียนรู้เหล่านี้จะถูกนำมารวมกัน เพื่อให้ได้คำตอบที่ประเมินมูลค่าของซอฟต์แวร์ได้ใกล้เคียงที่สุด โดยขั้นตอนวิธีที่ใช้ตัดสินใจในขั้นสุดท้ายจะเป็นขั้นตอนวิธีแบบผสม ซึ่งจะมีการเลือกคุณลักษณะต่างๆ ที่เหมาะสมมาประมวลผลรวมกัน แล้วทำการปรับค่าให้ได้ผลลัพธ์ที่ดีที่สุด โดยค่าคำตอบที่ต้องการของการเรียนรู้ (Desired output) เพื่อการเปรียบเทียบนั้น จะได้จากค่าการประมาณมูลค่าซอฟต์แวร์ด้วยวิธี COCOMO และ ข้อมูลจริงของ NASA ดังนั้นผู้วิจัยคาดหวังว่าการวิจัยในโครงการนี้จะนำไปสู่การเลือกวิธีการและปัจจัยที่เหมาะสมในการประเมินค่าทั้ง 2 ชุด ดังกล่าว ซึ่งจะส่งผลต่อการคิดค่าใช้จ่ายที่ใกล้เคียงกับค่าใช้จ่ายที่จะเกิดขึ้นได้จริงโดยมีค่าความคลาดเคลื่อนน้อยลง รูปที่ 3-1 แสดงภาพรวมของขั้นตอนวิธีที่นำเสนอ ซึ่งมีหลักการคือ ทำการเลือกข้อมูลจากชุดข้อมูลที่มีอยู่ โดยแบ่งชุดข้อมูล

ออกเป็น *n* ชุด แล้วนำข้อมูลที่เลือกมาได้เข้าสู่กระบวนการเรียนรู้ซึ่งจะใช้ Multi Layer Perceptron ที่มีสถาปัตยกรรมแตกต่างกันจำนวน *m* ตัวแบบ และใช้การปรับค่าน้ำหนักแบบแพร่กระจายย้อนกลับ เพื่อทำการเรียนรู้ ผลจากการเรียนรู้จากแต่ละโมเดลจะถูกนำเข้าสู่กระบวนการรวมคำตอบ ซึ่งจะใช้วิธีการแบบผสมเป็นตัวปรับค่าจากผลลัพธ์ที่ได้มาจากหลายตัวเรียนรู้และทำการตัดสินใจในขั้นสุดท้ายก่อนที่จะให้ผลลัพธ์ที่ดีที่สุดออกไป



รูปที่ 3-1 การเรียนรู้แบบผสมที่นำเสนอ

บทที่ 4 ผลการทดลอง

จากการวางแผนการทดลองที่ได้ออกแบบไว้ในบทที่ 3 ดังแสดงในรูปที่ 3-1 และคณะผู้วิจัยได้รวบรวมข้อมูลจากฐานข้อมูลมาตรฐานในการพยากรณ์ราคาซอฟต์แวร์ของ NASA โดยข้อมูลที่รวบรวมมานี้เป็นข้อมูล 16 มิติ จำนวน 93 ชุด โดยจะทำการทดลองทั้งหมด (ยังไม่มีการสุ่มเลือกข้อมูล) และในการทดลองขั้นตอนนี้จะแบ่งข้อมูลออกเป็น 3 กลุ่มย่อย ตามมูลค่าจริงของซอฟต์แวร์นั้น (ซึ่ง NASA กำหนดไว้) แล้วทำการประมาณราคาซอฟต์แวร์โดยใช้โครงข่ายประสาทเทียมแบบแพร่ย้อนกลับ ซึ่งมี 5 โครงสร้างที่แตกต่างกัน คือ 16x32x1, 16x64x1, 16x32x16x1, 16x32x32x1, และ 16x400x1, โดยกำหนดพารามิเตอร์สำหรับการเรียนรู้ของโครงสร้างทั้ง 5 โครงสร้างเหมือนกัน ดังนี้

LearningRate=0.4

Momentum=0.2

TrainingTime=50000

Epsilon = Depends on Number of patterns

Training/Testing = 100%

Activation functions (Sigmoid function) = 1 / (1 + exp(-x))

4.1 ผลการพยากรณ์ราคาซอฟต์แวร์

ตารางที่ 4-1 นำเสนอค่าความคลาดเคลื่อนที่เกิดจากการพยากรณ์ราคาซอฟต์แวร์ ตามโครงสร้างที่กำหนดไว้ในก่อนหน้านี้ จากการทดลองพบกว่า การแบ่งข้อมูลออกเป็นชุดย่อยให้ผลการทดลองในการพยากรณ์ที่ดีเร็วกว่า (อย่างมีนัยสำคัญ) และผลการพยากรณ์มีความถูกต้องสูงกว่าเมื่อทำการพยากรณ์โดยใช้ข้อมูลทั้งหมด (93 patterns) โดยคณะผู้วิจัยใช้โครงสร้างโครงข่ายประสาทเทียมแบบ 16 x 400 x 1 ได้ค่า MAE เป็น 5.9355e+007 ยกเว้นในข้อมูลชุดย่อยที่ 3 พบว่ายังให้คำตอบที่ยังไม่เป็นที่น่าพอใจ ซึ่งคณะผู้วิจัยจะศึกษาเพื่อหาสาเหตุ และดำเนินการปรับปรุงต่อไป

ตารางที่ 4-1 ค่าความคลาดเคลื่อนสัมบูรณ์ (MAE) ของการพยากรณ์ราคาซอฟต์แวร์

BPNN Network Structures	MAE (Mean Absolute Error)		
	ชุดข้อมูลย่อยที่ 1 P1 – P31	**ชุดข้อมูลย่อยที่ 2** P32 – P77	**ชุดข้อมูลย่อยที่ 3** P78 – P93
16 x 32 x 1	1.4363e+003	9.3266e+005	1.1068e+295
16 x 64 x 1	1.5394e+003	9.3202e+005	2.5650e+007
16 x 32 x 16 x 1	1.4240e+003	9.5665e+005	2.5650e+007
16 x 32 x 32 x 1	1.5922e+003	9.3202e+005	2.5650e+007
16 x 400 x 1	2.0308e+003	1.4391e+004	2.5650e+007

บทที่ 5 สรุปผลการทดลอง

5.1 สรุปผลการทดลอง

ในงานวิจัยนี้ (ปีงบประมาณ 2554) ได้นำเสนอผลการทดลองเบื้องต้นในส่วนของการพยากรณ์ราคาซอฟต์แวร์ของฐานข้อมูลมาตรฐานของ NASA โดยแบ่งข้อมูลสำหรับการสอนออกเป็น 3 ชุดย่อยตามราคาของซอฟต์แวร์ จากนั้นนำเสนอขั้นตอนวิธีการรู้จำแบบแพร่ย้อนกลับโดยใช้โครงสร้างของโครงข่ายที่แตกต่างกันจำนวน 5 โครงสร้าง ผลที่ได้จากการทดลองแสดงให้เห็นว่า การแบ่งข้อมูลออกเป็นชุดย่อยให้ผลการทดลองในการพยากรณ์ที่เร็วกว่าและดีกว่าเมื่อทำการพยากรณ์โดยใช้ข้อมูลทั้งหมด ดังนั้น เราจึงสามารถสรุปในขั้นต้นได้ว่า การพยากรณ์แบบแบ่งข้อมูลออกเป็นชุดย่อยตามที่คณะผู้วิจัยนำเสนอวันในข้อเสนอโครงการวิจัยนั้นถูกต้อง และมีความเป็นไปได้ที่จะเพิ่มประสิทธิภาพในการพยากรณ์ในการทำวิจัยในปีงบประมาณถัดไป และนอกจากนี้เรายังสามารถเพิ่มประสิทธิภาพของการรู้จำได้โดยใช้เทคนิคการรู้จำแบบอื่น ๆ หรือการสกัดคุณลักษณะเด่นเพื่อลดเวลาในการประมวลผลเพื่อการพยากรณ์มูลค่าซอฟต์แวร์

5.2 งานที่ต้องทำต่อไปในปีงบประมาณ พ.ศ. 2555

1. ศึกษาและพัฒนาขั้นตอนวิธีสำหรับการสกัดคุณลักษณะเด่นสำหรับการพยากรณ์มูลค่าซอฟต์แวร์
2. ศึกษาและพัฒนาขั้นตอนวิธีเพื่อการรู้จำแบบอื่น ๆ รวมทั้งการเรียนรู้แบบผสมสำหรับการพยากรณ์มูลค่าซอฟต์แวร์ให้มีประสิทธิภาพสูงขึ้น
3. ศึกษาและปรับปรุงฟังก์ชันการกระตุ้น (Activation Function) ที่เหมาะสมสำหรับโครงข่ายประสาทเทียม
4. ตีพิมพ์ผลงานวิจัยในประชุมวิชาการระดับนานาชาติ และ/หรือ วารสารวิจัย

บรรณานุกรม

B. Tirimula Rao, B. Sameet, G. Kiran Swathi, K. Vikram Gupta, Ch. RaviTeja, and S.Sumana, (2009), A Novel Neural Network Approach For Software Cost Estimation Using Functional Link Artificial Neural Network (FLANN), *International Journal of Computer Science and Network Security*, VOL.9 No.6, pp.126-131.

Charles.W.Therrien, (1989), *Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics,* John Wiley & Sons.

D. H. Wolpert, (1992), Stacked Generalization. *Neural Networks*, 5(2), pp. 241–259.

K. Vinay Kumar, V. Ravi *, Mahil Carr, and N. Raj Kiran, (2008), Software development cost estimation using wavelet neural networks, *The Journal of System and Software*, Vol. 81, pp. 1853-1867.

L. Breiman, (1996), Bagging Predictors, *Machine Learning*, 24(2), pp. 123–140.

N. Tadayon, (2005), Neural Network Approach for Software Cost Estimation, *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, Vol.02, pp. 815 – 818.

R. E. Schapire, (1990), The Strength of Weak Learnability, *Machine Learning*, 5(2), pp. 197–227.

S. Haykin, (1998), *Neural Networks: A Comprehensive Foundation (2nd edition)*, Prentice Hall.

Y. Kultur, B. Turhan and A. Bener (2009), Ensemble of neural networks with associative memory (ENNA) for estimating software development costs, *Knowledge-Based Systems,* Vol. 22, pp. 395-402.

Y. Freund and R. E. Schapire, (1996), Experiments with a New Boosting Algorithm., *Proceedings of the13th International Conference on Machine Learning (ICML'96).* San Francisco, CA: Morgan Kaufmann, pp.148–156.

J. D. Aron, (1969) *Estimating Resource for Large Programming Systems*, NATO Science Committee, Rome, Italy.

R.K.D. Black, R. P. Curnow, R. Katz and M. D. Gray, (1977) , BCS Software Production Data, *Final Technical Report, RADC-TR-77-116*, Boeing Computer Services, Inc.

B. W. Boehm, (1981), *Software engineering economics*, Englewood Cliffs, NJ: Prentice-Hall.

B.W. Boehm et al., (1996), *The COCOMO 2.0 Software Cost Estimation Model*, American Programmer, pp.2-17.

G. Cantone, A. Cimitile and U. De Carlini, (1986) A comparison of models for software cost estimation and management of software projects, in *Computer Systems: Performance and Simulation*, Elsevier Science Publishers B.V.

R. E. Park, (1988), PRICE S: The calculation within and why, *Proceedings of ISPA Tenth Annual Conference*, Brighton, England.

N. A. Parr, (1980), An alternative to the Raleigh Curve Model for Software development effort, *IEEE on Software Engineering*.

R. Tausworthe, (1981), *Deep Space Network Software Cost Estimation Model*, Jet Propulsion Laboratory Publication 81-7.

R. W. Wolverton, (1974), The cost of developing large-scale software, *IEEE Trans. Computer*, pp.615-636.