การคัดแยกไวรัสคอมพิวเตอร์จากรหัสฐานสอง

นายประสิทธิ์ อุษาฟ้าพนัส

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2560

Classification of Computer Viruses from binary code

Mr. Prasit Usaphapanus

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2017

หัวข้อวิทยานิพนธ์	การคัดแยกไวรัสคอมพิวเตอร์จากรหัสฐานสอง
โดย	นายประสิทธิ์ อุษาฟ้าพนัส
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.นัทที นิภานันท์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา)

.....กรรมการ

(อาจารย์ ดร.พีรพล เวทีกูล)

.....กรรมการภายนอกมหาวิทยาลัย

(ดร.พงศ์ธวัช ชีพพิมลชัย)

ประสิทธิ์ อุษาฟ้าพนัส : การคัดแยกไวรัสคอมพิวเตอร์จากรหัสฐานสอง (Classification of Computer Viruses from binary code) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.เกริก ภิรมย์โสภา, 49 หน้า.

งานวิจัยนี้นำเสนอการใช้การเรียนรู้แบบมีผู้สอนเพื่อตรวจจับไฟล์ไวรัสคอมพิวเตอร์ที่ไม่เคยพบมาก่อนแบบ static ผู้วิจัยได้ทดสอบกับตัวแยกประเภทจำนวน 3 แบบ คือ random forest, multilayer perceptron และ extreme gradient boosting ชุดข้อมูลประกอบด้วย 6319 ไฟล์ executable แต่ละไฟล์ถูกสกัดด้วย objdump แล้วจัดเรียงตามคะแนน TF-IDF เพื่อหา feature ที่เหมาะสม ผลลัพธ์เปรียบเทียบด้วย F_1-score คือ สามารถใช้ตัวแยกประเภทแบบ random forest ร่วมกับข้อมูลที่มี 20 attribute ได้ 0.937 F_1-score ซึ่งมากกว่าบรรทัดฐานอยู่ 0.031 F_1-score และ สามารถใช้ตัวแยกประเภทแบบ extreme gradient boosting ร่วมกับข้อมูลที่มี 500 attribute ได้ 0.962 F_1-score ซึ่งมากกว่าบรรทัดฐานอยู่ 0.041 F_1-score จึงสรุปได้ว่าวิธีการในงานวิจัยนี้สามารถเพิ่ม precision และ recall ของการแยกประเภทได้

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2560

5870411921 : MAJOR COMPUTER ENGINEERING
KEYWORDS: COMPUTER SECURITY / DATA MINING

PRASIT USAPHAPANUS: Classification of Computer Viruses from binary code. ADVISOR: ASST. PROF. KRERK PIROMSOPA, Ph.D., 49 pp.

This thesis proposes a supervised machine learning model for detecting (unseen) viruses files. Our main focus is on static analysis approach. To find the best method, we experiment with difference types of feature extraction and three classifier algorithms including extreme gradient boosting, random forest and multilayer perceptron. Our data set contains 6,319 executable files. Each file is extracted with objdump and sorted with TF-IDF score to find best features. The F_1-score shows slightly better performance than those of the baselines. Random forest with 20 attributes yields 0.937 F1 score which is 0.031 more than that of the baseline . The extreme gradient boosting method with 500 attributes achieve 0.962 F1 score, 0.041 more than that of the baseline. We conclude that our approach can improve the precision and recall of the classification.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Engineering Advisor's Signature .....
Academic Year: 2017

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี มีองค์ประกอบมาจากส่วนสำคัญที่ขาดไม่ได้คือ ความรู้ทางวิชาการที่คณาจารย์ในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ทุกท่านที่ได้ประสิทธิ์ประสาทให้แก่ข้าพเจ้า อีกทั้งประกอบกับคำแนะนำต่างๆ ที่เป็นประโยชน์ต่อการดำเนินงานวิจัย โดยเฉพาะอย่างยิ่งการดูแล และให้ความช่วยเหลืออย่างสม่ำเสมอจาก ผศ. ดร. เกริก ภิรมย์โสภา อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า ซึ่งเป็นแบบอย่างทั้งในด้านการศึกษา และยังเป็นแบบอย่างสำคัญในด้านการดำเนินชีวิต ข้าพเจ้าจึงกราบแสดงความขอบพระคุณในความเมตตากรุณามา ณ ที่นี้

ขอขอบพระคุณคณาจารย์ และคณะกรรมการสอบวิทยานิพนธ์ทุกท่าน ได้แก่ ผศ. ดร. นัทที นิภานันท์ (ประธานกรรมการสอบวิทยานิพนธ์), อ. ดร. พีรพล เวทีกูล (กรรมการสอบวิทยานิพนธ์) และ อ. ดร. พงศ์ธวัช ชีพพิมลชัย (กรรมการจากภายนอกมหาวิทยาลัย) ที่ได้ชี้แนะแนวคิดในการพัฒนาการวิจัย แนวทางการปรับปรุงแก้ไข เพิ่มเติมในส่วนที่บกพร่อง และรวมถึงชี้แนะวิธีการนำเสนอ เพื่อให้งานวิจัยสำเร็จลุล่วงครบถ้วนตามเป้าหมาย

ขอขอบคุณทุนอัจฉริยะคืนรังจากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ซึ่งได้สนับสนุนค่าเล่าเรียนในภาคการศึกษาที่ลงทะเบียนตามจำนวนที่จ่ายจริงเป็นเวลา 4 ภาคการศึกษา นับตั้งแต่เริ่มเข้าศึกษา

สุดท้ายนี้ขอกล่าวคำขอบพระคุณ รายนามบุคคลดังต่อไปนี้ซึ่งมีคุโณปการสำคัญยิ่งต่องานวิจัยฉบับนี้ อันประกอบด้วย นายสมิทธ์ ธรรมบำรุง, นายภาสกร ทองสันตติ์ และนายธนชัย จิระจันทร์ ที่ได้เอื้อเฟื้อทั้งแรงกาย กำลังสติปัญญา ความรู้ ประสบการณ์ และคำแนะนำในการดำเนินงาน และการแก้ไขปัญหาต่างๆ ตลอดมา งานวิจัยนี้คงสำเร็จลงได้ยาก หากปราศจากความเอื้อเฟื้อจากทุกท่าน จึงขอเอ่ยนามเพื่อแสดงความขอบคุณสำหรับความกรุณาไว้ ณ ที่นี้

สารบัญ

หน้า

หน้า

บทที่ 1. บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ไวรัสคอมพิวเตอร์เพิ่มจำนวนขึ้นอย่างมากในปี 2015 [1] Kaspersky web antivirus ตรวจพบมัลแวร์ 121,262,075 ชนิด และทุกวันนี้ มีมัลแวร์เกือบ 1 ล้านตัว ถูกกระจายออกสู่โลกอินเทอร์เน็ต[2] โดยในจำนวนนี้ เป็นไวรัสถึง 57% และ โทรจัน 21% หรือประมาณได้ว่า มีไวรัสชนิดใหม่ถูกสร้างขึ้น 74000 ชนิด ทุกวัน

จากการสำรวจมากกว่า 30% ของผู้ใช้คอมพิวเตอร์ในสหรัฐอเมริกาติดมัลแวร์ และในสองปีที่ผ่านมา มีจำนวน 16 ล้านครัวเรือนที่ติดไวรัส ส่งผลเสียต่อเศรษฐกิจของสหรัฐอเมริกาถึง 4.55 พันล้านดอลลาร์สหรัฐ [3]

สถิติดังกล่าว แสดงให้เห็นถึงความเสียหายที่เกิดขึ้นต่อผู้ใช้คอมพิวเตอร์ ซึ่งแพร่หลายไปในธุรกิจทุกประเภท ส่งผลเสียต่อเศรษฐกิจของประเทศและประชากร นอกจากนี้ การที่จำนวนไวรัสเพิ่มจำนวนขึ้นอย่างมหาศาลในหนึ่งวัน ทำให้การระบุตัวตนของไวรัสจากลายเซ็นยากขึ้น จึงมีแนวคิดการระบุตัวตนของไวรัสจากปัจจัยอื่น ซึ่งจะนำเสนอในหัวข้อต่อไป

1.2 วัตถุประสงค์

1.2.1 เพื่อประเมินความสามารถในการแยกประเภทของไฟล์ทั่วไปและไฟล์ไวรัส

1.2.2 เพื่อพัฒนาระบบซึ่งสามารถแยกประเภทไฟล์ไวรัสคอมพิวเตอร์ที่ไม่เคยพบมาก่อนได้

1.3 ขอบเขตของการดำเนินงาน

1.3.1 ทดสอบการวิเคราะห์ไวรัสขณะที่ยังไม่ถูกสั่งให้ทำงาน

1.3.2 ไฟล์ไวรัสจำนวน 3000 ไฟล์ ไฟล์ทั่วไปจำนวน 3000 ไฟล์

1.3.3 จำกัดประเภทไฟล์ เฉพาะ executable เท่านั้น

1.3.4 ไฟล์ในระบบ Windows 32bit เท่านั้น

1.3.5 ไฟล์ซึ่ง objdump สามารถอ่านได้เท่านั้น

บทที่ 2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.4. ทฤษฎีที่เกี่ยวข้อง

2.4.1 การเรียนรู้แบบมีผู้สอน [4][5]

การเรียนรู้แบบมีผู้สอน (supervised learning) เป็นส่วนหนึ่งของการเรียนรู้ของเครื่อง (machine learning) โดยมีชุดข้อมูลสอน(training data set)เป็นตัวอย่างให้เรียนรู้พร้อมทั้งผลเฉลยการจำแนกประเภทที่ถูกต้อง ผลจากการเรียนรู้จะนำมาสร้างเป็นฟังก์ชันการถดถอย (regression) และนำฟังก์ชันนั้น ไปใช้คำนวณหาผลลัพธ์เพื่อทำนายจำแนกประเภท (classification)ต่อไป ฟังก์ชันที่ใช้จะมีหลากหลายแบบ ซึ่งใช้ทรัพยากรของคอมพิวเตอร์แตกต่างกัน และได้ผลลัพธ์ไม่เหมือนกัน ดังนี้

2.4.1.1 Decision tree [6]

ต้นไม้ตัดสินใจ นำเอาแนวคิดของการแบ่งแยกและปกครอง(divide-and-conquer) มาใช้แก้ปัญหา โดยแตกกิ่งออกไปจากราก แต่ละกิ่งมีเงื่อนไขขึ้นอยู่กับค่าของattribute และมีผลลัพธ์การทำนายอยู่ที่ใบ การทำงานจะเริ่มต้นจากจุดที่เป็นราก แล้วเปรียบเทียบค่าของ attribute กับเงื่อนไขของจุดต่อ(node)ว่าตรงกับกิ่งใดก็เลื่อนไปทำงานในจุดต่อในกิ่งนั้น จนกระทั่งถึงจุดที่เป็นใบก็จะได้ผลลัพธ์ออกมา ซึ่งลำดับการเลือก attribute ที่จะนำมาสร้างจุดต่อนั้น จะคำนวณด้วย information gain หรือ information gain ratio ดังจะกล่าวถึงต่อไปใน

2.4.1.2 Support vector machine [7]

Support vector machine (SVM) จะพยายามสร้าง hyper plane ขึ้นมาเพื่อแบ่งแยกข้อมูลของแต่ละประเภทออกจากกัน ซึ่งถ้ามี attribute จำนวน n attributes จะได้ hyper plane ที่เป็น n-1 มิติ เช่น ถ้าหากข้อมูลมี 2 attributes จะสามารถเขียนกราฟได้เป็น 2 มิติ และ hyper plane เป็นเส้น 1 เส้น ซึ่งการคำนวณ hyper plane ที่เหมาะสมนั้น สามารถเลือกวิธีคำนวณได้หลายวิธี ขึ้นอยู่กับการกำหนด kernel function และ kernel function ที่แตกต่างกันนี้ ส่งผลต่อทรัพยากรที่ใช้ในการประมวลผล และผลลัพธ์ของการทำนาย

2.4.1.3 Multilayer perceptron [5]

Multilayer perceptron คือ โครงข่ายประสาทเทียมแบบหนึ่ง ซึ่งมีโครงสร้างหลายชั้น เพื่อให้สามารถทำนายจำแนกประเภทที่มีความซับซ้อนได้ โครงข่ายประสาทเทียม พยายามจำลองการทำงานของเซลล์สมองมนุษย์ โดยมีค่าของ attributes ส่งเข้าไปเป็น input แล้วคำนวณร่วมกับน้ำหนัก ขีดเริ่มเปลี่ยน รวมเข้าด้วยกันในแต่ละnodeของlayerถัดไป สำหรับ multilayer perceptron นี้ จะมีโครงสร้างหลายชั้น เพื่อให้สามารถทำนายจำแนกประเภทที่มีความซับซ้อนได้

2.4.1.4 RIPPER [8]

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) เป็น classifier ประเภท rule based คือ จะมีตารางสำหรับการตัดสินใจในสิ่งต่าง ๆ โดยเงื่อนไขจะขึ้นอยู่กับค่าของ attribute เช่น ถ้า attribute x1 มีค่ามากกว่า 7 ให้ทำนายเป็นประเภท A เป็นต้น สร้างขึ้นด้วยวิธีการที่เรียกว่า covering approach โดยในแต่ละขั้น จะเลือกกฎ ที่ครอบคลุม instance บางส่วน

2.4.1.5 Naive Bayes [9]

Naive Bayes classifier ซึ่งใช้การตัดสินใจจากการคำนวณทางสถิติตามกฎของ Bayes คือ ถ้ามีสมมติฐาน H และข้อเท็จจริง E ซึ่ง H คือclass และ E เป็น attribute ของ instanceที่สนใจ จะคำนวณหาความน่าจะเป็นได้จากสูตร

$$\mathrm{P}(H|E) = \frac{\mathrm{P}(E|H) \times \mathrm{P}(H)}{\mathrm{P}(E)} \quad (1)$$

ในกรณีที่มีหลาย attribute จะแตกออกเป็น E_1, E_2, ... , E_n

$$\mathrm{P}(H|E) = \frac{\mathrm{P}(E_1|H) \times \mathrm{P}(E_2|H) \times \ \ldots \ \times \mathrm{P}(E_n|H) \times \mathrm{P}(H)}{\mathrm{P}(E)} \quad (2)$$

ซึ่งวิธีนี้จะสมมติว่า ในแต่ละ attribute นั้น ไม่ขึ้นต่อกัน เนื่องจากในกรณีที่มีการขึ้นต่อกันเกิดขึ้น ค่าที่คูณเข้าด้วยกันจะส่งผลอย่างมากจนอาจได้ผลลัพธ์ที่ผิดไป นอกจากนี้ยังมีการเพิ่มค่าคงที่ Laplace estimator เข้าไปไม่ให้ความน่าจะเป็นของบาง attribute เป็น 0 เพราะการมี 0 คูณกับค่าใด ๆ ก็ตาม ผลลัพธ์ย่อมเป็น 0 และปฏิเสธการทำนายนี้ไป

2.4.1.6 Random forest [10]

Random forest เป็น ensemble classifier ในกลุ่ม bootstrap aggregation แบบหนึ่ง ใช้ต้นไม้ตัดสินใจจำนวนมาก ร่วมกันเพื่อทำการ vote ผลลัพธ์ที่ต้องการ โดยแต่ต้นไม้ตัดสินใจแต่ละต้น จะได้ attribute ไปแตกต่างกันจากการสุ่มหยิบ สามารถทำงานได้อย่างรวดเร็ว ใช้ประโยชน์จาก multithread ได้อย่างเต็มที่ และมีความสามารถในการ predict โดยขึ้นอยู่กับจำนวนของต้นไม้ที่ใช้

2.4.1.7 Extreme gradient boosting [11]

Extreme gradient boosting หรือ XGBoost มีพื้นฐานอยู่บน gradient boosting ซึ่งเป็น ensemble classifier ในกลุ่ม boosting แบบหนึ่ง แต่ได้รับการปรับปรุงให้สามารถทำงานได้อย่างรวดเร็วและมีประสิทธิภาพมากขึ้นกว่าเดิม สามารถใช้ประโยชน์จาก multithread ได้อย่างเต็มที่ และเพิ่มตัวแปร regularization เข้าไปเพื่อลดการ overfit ปัจจุบันได้รับความสนใจอย่างมาก

หลังจากที่การแข่งขัน KDDCup ในปี ค.ศ.2015 เปิดเผยว่าผู้ชนะ 10 อันดับแรก ทุกทีมเลือกใช้ XGBoost

2.4.2 Feature selection [4]

Feature selection คือ หัวข้อหนึ่งของการแปลงข้อมูล (data transformation) โดยมีเป้าหมายเพื่อลดจำนวนของ attribute ที่ไม่จำเป็นลง ทำให้ machine learning algorithm สามารถทำนายผลลัพธ์ได้แม่นยำขึ้น และใช้เวลาในการทำงานน้อยลง การทำ feature selection สามารถแบ่งออกได้เป็นสองวิธี วิธีแรก จะพิจารณาแต่ละ attribute ทีละตัว แล้วคำนวณหาค่าผลลัพธ์ออกมาเรียกว่าค่า merit แล้วนำมาเรียงลำดับตามค่าmeritดังกล่าว เรียกว่า filter method และ อีกวิธีหนึ่ง จะแยกเป็นสองขั้นตอน ประกอบด้วยการค้นหา และการประเมินค่าความแม่นยำของการทำนาย เรียกว่า wrapper method ซึ่งในงานวิจัยนี้ เลือกใช้ filter method ในขั้นตอนของ feature selection แต่ละวิธี ซึ่งมีสูตรคำนวณค่าmeritแตกต่างกันไป ดังนี้

2.4.2.1 Correlation-based feature selection [12]

Mark A. Hall ในปี ค.ศ. 1999 ได้นำเสนอวิธีการ feature selection โดยที่มีสมมติฐานว่า feature ที่ดี จะมีความสัมพันธ์ร่วมกัน(correlate) กับ class แต่ไม่มีความสัมพันธ์ร่วมกันกับ feature อื่น ๆ งานวิจัยที่สร้าง correlation-based feature selection ขึ้นมา ได้ทดสอบกับทั้ง artificial dataset และ natural dataset ร่วมกับ classifier 3 แบบ คือ C4.5 decision tree learner , IB1 instance based learner และ naive Bayes ซึ่งผลการทดสอบ ได้แสดงให้เห็นว่า correlation-based feature selection สามารถกำจัด feature ที่ไม่ต้องการได้ตราบเท่าที่ feature นั้น ไม่ขึ้นอยู่กับ feature อื่น ๆ และ ผลลัพธ์ส่วนใหญ่ ให้ความแม่นยำเพิ่มขึ้นมากกว่าของเดิมที่ใช้ทุก feature และนอกจากนี้ ยังได้ทำการทดลองเปรียบเทียบกับ feature selection แบบ wrapper และได้ผลลัพธ์ใกล้เคียงกัน แต่ correlation-based feature selection สามารถทำงานได้เร็วกว่าแบบ wrapper มาก ซึ่งทำให้มีความเหมาะสมในการใช้งานกับชุดข้อมูลขนาดใหญ่

2.4.2.2 Information gain และ information gain ratio feature selection

Information gain นำเอาความรู้จาก information theory ของ Shannon ในส่วนของ Shannon Entropy มาใช้ คือ Entropy $\mathbf{H} = -\sum_i p_i \log(p_i)$ ซึ่ง function ค่า Entropy นี้จะถูกนำมาใช้ในการคำนวณเพื่อหาค่า information gain ซึ่งได้รับการยอมรับและนำมาใช้ในการสร้างต้นไม้ตัดสินใจแบบ ID3 แต่เนื่องจาก information gain มีความอ่อนไหวกับ feature ที่สามารถเป็นไปได้หลายค่า เช่น กรณีที่เป็นหมายเลขประจำตัวซึ่งไม่ซ้ำกัน เป็นต้น ดังนี้จึงได้มีการพัฒนา information gain ratio ขึ้นมาโดยเฉลี่ยค่าตาม intrinsic value และปรับใช้กับการสร้างต้นไม้แบบ C4.5

2.4.2.3 Symmetrical uncertainty feature selection [13][

Lei Yu และ Huan Liu ได้นำเสนอใน ปี ค.ศ. 2003 โดยมีพื้นฐานของการคำนวณจาก Information Theory คล้ายกับ Information Gain แต่มีการเพิ่มขั้นตอนในการnormalize เพื่อลด bias ของ Information Gain โดย symmetrical uncertainty ดังสูตรต่อไปนี้

$$\mathrm{SU}(X,Y) = 2 \times [\frac{\mathrm{IG}(X|Y)}{\mathrm{H}(X) + \mathrm{H}(Y)}] \quad (3)$$

ซึ่งจะทำให้ค่าที่ได้อยู่ในช่วง [0,1] โดยที่ 1 แสดงว่า feature นั้น สามารถทำนาย class ได้อย่างสมบูรณ์

2.4.2.4 Term frequency-inverse document frequency [14]

การให้น้ำหนักคำในเอกสารใด ๆ ในเอกสารทั้งหมด โดยดูจากผลคูณของ term frequency กับ inverse document frequency โดยผลลัพธ์ที่ได้จะมีน้ำหนักมากตามผลลัพธ์จากการคำนวณดังนี้

$$tfidf = tf \times idf \quad (4)$$

โดยค่าของ term frequency หรือ *tf* จะคำนวณจากสูตรดังนี้ โดย *w* คือ คำที่ต้องการคำนวณ และ *D* คือ เอกสารที่กำลังสนใจ

$$tf(w,D) = f_{w_D} \quad (5)$$

และค่าของ inverse document frequency หรือ *idf* สามารถคำนวณจากสูตรดังนี้ โดยที่ C แทนจำนวนเอกสารทั้งหมดที่มีอยู่ และ df(t) แทนจำนวนเอกสารที่มีคำ t ปรากฏอยู่ โดยใส่ 1 เข้าไปใต้ C เพื่อป้องกันปัญหาการหารด้วยศูนย์

$$idf(t) = 1 + log\frac{C}{1 + df(t)} \quad (6)$$

2.4.3 Ensemble [15]

เทคนิคเพื่อลดความแปรปรวนและเพิ่มความแม่นยำในการทำนายของ classifier ดังนี้

2.4.3.1 Vote

การโหวต สามารถทำได้โดยนำผลลัพธ์การทำนายจาก classifier ที่แตกต่างกัน มาพิจารณาร่วมกัน ซึ่งสามารถทำได้หลายแบบ เช่น โหวตโดยเสียงข้างมาก(Simple majority voting) และอาจใส่น้ำหนักเข้าไปในแต่ละ classifier ได้ด้วยหากทราบว่าclassifierสามารถเชื่อถือได้มากเพียงใด การโหวตนี้จะมีประสิทธิภาพมากในกรณีที่ผู้โหวต ไม่ขึ้นต่อกัน ตามทฤษฎีคณะลูกขุนของคอนดอร์ (Condorcet Jury theorem) ให้ p เป็น ความแม่นยำของผู้โหวตแต่ละคน Tเป็นจำนวนของผู้โหวต จะได้ดังสมการต่อไปนี้

$$\mathrm{p}_{ens} = \sum_{k=\frac{T}{2}+1}^{T} \binom{T}{k} p^k (1-p)^{T-k} \quad (7)$$

2.4.3.2 Bagging

Bagging หรือ ชื่อเต็มคือ Bootstrap Aggregation เป็นเทคนิคที่สุ่มหยิบข้อมูลในชุดฝึก (Training set) มาให้ Classifier เรียนรู้ซ้ำ ๆ เป็นชุด โดยการสุ่มหยิบนี้จะใส่คืนเพื่อให้ในรอบการสุ่มหยิบครั้งถัดไปสามารถได้ instance เดียวกันกับการสุ่มหยิบครั้งก่อนได้ การทำเช่นนี้ส่งผลให้ classifier เรียนรู้จากชุดฝึกเดิมหลาย ๆ รอบ ส่งผลให้ classifier มีความเสถียร และแม่นยำขึ้น นอกจากนี้ยังช่วยลดความแปรปรวน และหลีกเลี่ยงการเกิดoverfitting

2.4.3.3 Boosting

Boosting ใช้เพื่อลด bias และความแปรปรวน โดยมีแนวคิดให้มี weak learner ชุดหนึ่งทำงานร่วมกันจนสามารถเป็น strong learner ได้ การสร้าง weak learner แต่ละตัวสามารถทำได้โดยการปรับเพิ่มน้ำหนักของpenaltyของการทำนายที่ผิดพลาดให้มากขึ้นในแต่ละรอบ แล้วทำการเรียนรู้ใหม่ ซึ่งจะทำให้โมเดลของ classifier เปลี่ยนไปโดยให้ความสำคัญกับความผิดพลาดในรอบที่แล้วมากขึ้น เมื่อได้จำนวน weak learner มากพอแล้วจึงนำมารวมกันสร้างเป็น strong learner

2.4.4 การพิจารณาและประเมินผลการทดสอบ

2.4.4.1 Confusion matrix [4]

Confusion matrix แสดงจำนวนของการทำนายถูกและผิด โดยการเปรียบเทียบระหว่างผลการทำนาย และผลลัพธ์จริง ๆ ที่ทราบอยู่แล้ว ใส่ลงในตารางขนาด NxN โดยที่ N เป็นจำนวนของคลาสที่เป็นไปได้ ในงานวิจัยนี้ใช้ N=2 ดังตัวอย่างต่อไปนี้

		Predicted class	
		Yes	No
Actual class	Yes	TP	FN
	No	FP	TN

ตาราง 1 confusion matrix

Accuracy คือ ค่าแสดงอัตราความแม่นยำในการทำนาย จากสูตรดังนี้

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (8)$$

Sensitivity, recall หรืออีกชื่อหนึ่งคือ true positive rate แสดงอัตราความถูกต้องในการทำนายว่าเป็นจริง จากสูตรดังนี้

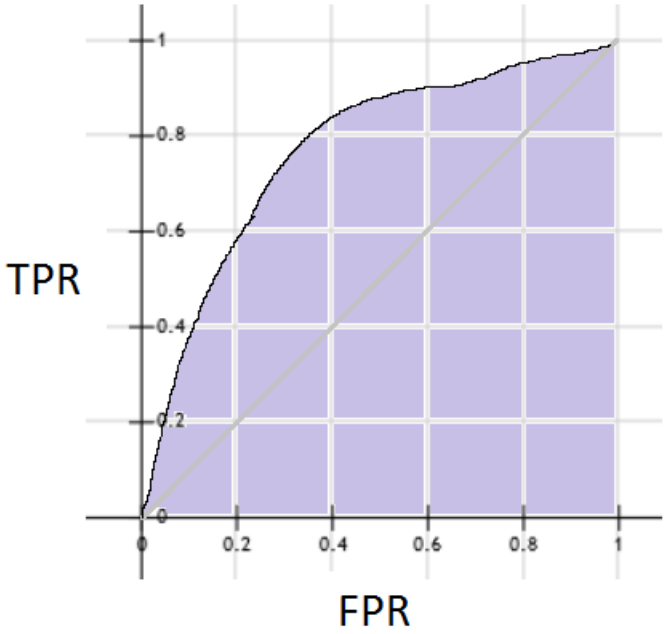
$$Recall = \frac{TP}{TP + FN} \quad (9)$$

F-measure หรือ F_1 score คำนวณจาก Accuracy และ Recall ร่วมกันเป็นค่าเดียว

$$F_1 = 2 \times \frac{Accuracy \times Recall}{Accuracy + Recall} \quad (10)$$

2.4.4.2 พื้นที่ใต้กราฟ Receiver operating characteristic (ROC) [4]

กราฟนี้เกิดจากการพลอตโดยมีแกน Y เป็น $\text{True positive rate} = 100 \times \frac{TP}{TP+FN}$ และแกน X เป็น $\text{False positive rate} = 100 \times \frac{\text{FP}}{\text{FP+TN}}$ ทั้งสองแกนแสดงเป็นเปอร์เซ็นต์ตั้งแต่ 0-100 โดยนับให้กราฟทั้งภาพมีพื้นที่ 1 หน่วย เมื่อวาดกราฟออกมาแล้ว หาพื้นที่ใต้กราฟ จะได้ค่าตั้งแต่ 0-1 หน่วย แสดงถึงสมรรถภาพของ binary classifier หรือ classifier ที่มี outputเป็นไปได้สองแบบ ค่าพื้นที่ใต้กราฟมากแสดงถึงสมรรถภาพที่สูงตามไปด้วย



รูป 1 ตัวอย่างแสดงพื้นที่ใต้กราฟ ROC

2.5. งานวิจัยที่เกี่ยวข้อง

งานวิจัยการใช้ Data mining เพื่อแยกประเภทของไฟล์เริ่มได้รับความสนใจจากงานของ 2001, Schultz, M.G. et al. หลังจากนั้นจึงมีงานวิจัยอื่น ๆ ในทำนองเดียวกันนี้ตีเผยแพร่ออกมา โดยในแต่ละงานวิจัยจะมีปัจจัยที่แตกต่างกัน เช่น featureที่เลือกใช้ วิธีการเลือกfeature (feature selection) classifierที่เลือกใช้ และ malwareที่นำมาพิจารณา โดยสามารถแบ่งงานวิจัยออกเป็นกลุ่มตามลักษณะfeatureที่เลือกใช้ ดังนี้

กลุ่มแรก ใช้ bytes code เป็น feature เช่น งานวิจัยของ Schultz, M.G. et al. [16] ได้ซึ่งเป็นงานวิจัยที่จุดประกายให้กับการใช้ machine learning เพื่อแยกประเภทไฟล์ โดยในงานนี้ได้ทดสอบโดยเปรียบเทียบระหว่าง Signature method, RIPPER กับ DLL, Naïve Bayes กับ Strings และ Multi-Naïve Bayes กับ n-gram ของ bytes code โดยในที่นี้ n=2 bytes และ shift = 2 และเสนอแนวทางการใช้งานบนระบบ e-mail [17] จากนั้นจึงมีงานวิจัยของ Kolter, J Zico และ Maloof, Marcus a [18] ออกมาต่อยอดในมุมของการใช้ bytes code เป็น feature นี้ โดยเพิ่มความหลากหลายของ ค่า n ใน n-gram ให้มากขึ้นและเพิ่ม classifier ที่ใช้ให้หลากหลายมากขึ้นด้วย

กลุ่มที่สอง ใช้ Opcode (operation code) เป็น feature เช่น งานวิจัยของ Siddequi, Muazzam , et al. [19] นำเอา Opcodeมาต่อกัน แล้วตัดด้วย Opcode ที่มีผลในการเปลี่ยนแปลงทิศทางการทำงานของโปรแกรม เรียกอีกอย่างหนึ่งว่า ใช้ feature เป็น basic block อีกแบบหนึ่งคือ Raja Khurram Shahzad [20] ใช้ Opcode แบบ n-gram, basic block และ sliding window ร่วมกัน เพื่อแยกไฟล์ benign กับ scareware

นอกจากfeatureสองกลุ่มข้างต้นแล้ว ยังมีงานวิจัยอื่น ๆ ที่มีลักษณะคล้ายกันนี้ เช่น Zhang, Bo-yun, et al. [21] ซึ่งเป็นการตรวจจับแบบ dynamic คือรันไฟล์จริง ๆ แล้วเก็บข้อมูลของลำดับ API function call มาใช้เป็น feature หรือ Wang, Tzu-yen [22] ใช้ API function call เช่นกัน แต่เป็นการตรวจจับแบบ static

เนื่องจากปัจจัยที่แตกต่างกันข้างต้น ร่วมกับการที่ไม่มีชุดข้อมูลกลาง ในแต่ละงานวิจัยจึงได้ผลลัพธ์ที่ต่างกัน และไม่สามารถเปรียบเทียบกันได้โดยตรง โดยจะสรุปได้ดังตารางต่อไปนี้

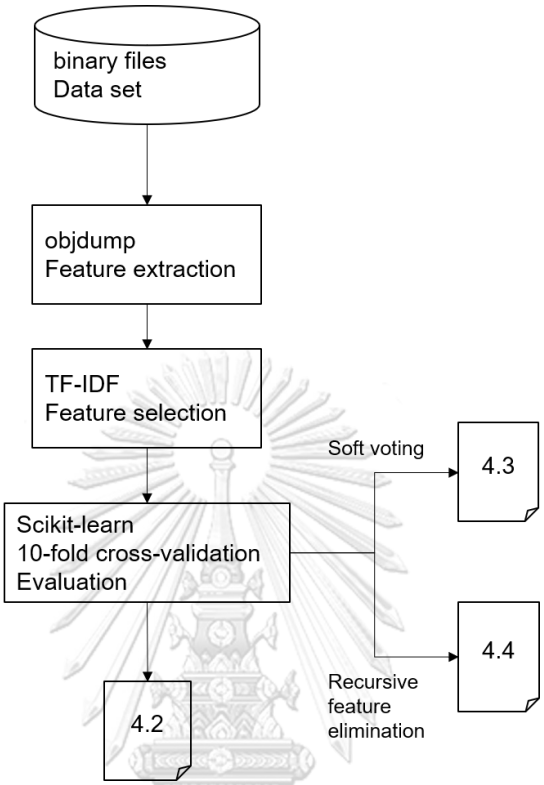
Author(s)	Features	Classifiers	Scope	Best result %Acc
Matthew G.Schultz Eleazar Eskin Zadok, Erez (2011) [16]	DLL function calls Strings Hexadecimal byte code	RIPPER Naive Bayes Multi-Naive Bayes	Windows or MS-DOS format 1001 benigns 3265 malicious	97.11
Kolter, J Zico Maloof, Marcus a (2006) [18]	Hexadecimal bytes code	SVM IBk J48 NaiveBayes Boosted	1971 benigns 1651 malicious	99.58
Bo-yun Zhang, et al (2006) [21]	API function calls (dynamic)	SVM	423 benigns 209 malicious 50 Benign + 50 Malicious Training 373 Benign + 159 Malicious Training	91.13
Tzu-Yen Wang Chin-Hsiung Wu (2008) [22]	DLL function calls	SVM with RBF kernel	1758 benigns 846 viruses	95.62
Siddequi, Muazzam Wang, Morgan C. Lee, Joohan (2008) [19]	Opcode basic block	Random forest Bagging Decision Tree	1330 benigns 1444 worms With control files size.	96
Dragos¸ Gavrilut¸ (2009) [23]	Binary	Perceptrons	~3M malwares ~180M benigns	96.16
Raja Khurram Shahzad (2010) [24]	Hexadecimal bytes code	ZeroR NaiveBayes SMO J48 Tree RandomForest JRipper	119 benigns 18 spywares	90.54

Raja Khurram Shahzad (2011) [25]	Opcode n-gram	NaiveBayes SMO IBk J48 JRip	Windows 300 benigns 300 adwares file size < 512 KB	94.9
Raja Khurram Shahzad (2011) [20]	Opcode basic block	SMO NaiveBayes IBk JRipper J48 Tree RandomForest	550 scarewares 250 benigns	97.2
Raja Khurram Shahzad (2012) [26]	Opcode n-gram	JRipper J48 Tree IBk	250 scarewares 250 benigns	98.6
Gil Tahan (2012) [27]	Hexadecimal bytes code	MAL-ID	849 malwares 2627 benigns	98.6
Dmitry Komashinskiy and Igor Kotenko (2012) [28]	Opcode (dynamic)	Naive Bay Decision Tree k Nearest Neighbors	11958 benigns 13120 malicious Bitfros, Lmir, Magania, OnlineGames, Poison, Vapsup	84.73
Raja Khurram Shahzad (2012) [29]	Strings Opcode bi-grams	JRipper J48Tree IBk Veto(Vote) Majority(Vote)	250 scarewares 250 benigns	95.8
Shanmugam, Gayathri Low, Richard M. Stamp, Mark (2013) [30]	Opcode (static)	Hill climbing similarity	50 NGVCK 50 G2 100 MWOR 30 benigns	100
Singh, Tanuvir Di Troia, Fabio Corrado, Visaggio Aaron	Opcode Graph based	SVM	Harebot 50 NGVCK 200 Security Shield 50	100

Austin, Thomas H. Stamp, Mark (2015) [31]			Smart HDD 50 Winwebsec 200 Zbot 200 ZeroAccess 200 Benign 40	
Vatamanu, Cristina Cosovan, Doina Gavrilu, Drago Luchian, Henri (2015) [32]	Static+Dynamic information	OSC algorithm+Ensemble	242811 malicious 2105896 benigns	76.46
Abhijit Yewale Maninder Singh (2016) [33]	Opcode 1-gram	Random forest	100 malwares 100 benigns	97

ตาราง 2 สรุปงานวิจัยที่เกี่ยวข้อง

บทที่ 3. การออกแบบการทดลอง



รูป 2 ออกแบบงานวิจัย

3.1 การรวบรวมชุดข้อมูล

ชุดข้อมูลแบ่งออกเป็น 2 ชุด คือ benign และ virus ซึ่งเป็นไฟล์ executable 32 bit ทั้งหมด ในส่วนของ benign นั้น ได้จากหลายแหล่งที่มา ดังนี้ ได้จากการติดตั้งระบบปฏิบัติการ Windows XP, Windows Vista, Windows 7 ไฟล์จากแฟ้มของ Cygwin และไฟล์จากการดาวน์โหลด ทั้งนี้ ไฟล์ benign ทั้งหมดได้ผ่านการตรวจสอบจาก Microsoft security essential แล้ว สำหรับไฟล์ virus ได้จากโครงการ Virus Heaven collection [34] ซึ่งได้มีการจัดเก็บรวบรวมและแยกประเภท malicious ต่าง ๆ ไว้แล้ว

3.2 การทำ feature extraction

Feature ที่จะนำมาใช้ในงานวิจัยมี 4 แบบ ต่อไปนี้

3.2.1 n-gram ของ Hexadecimal bytes code

ใช้คำสั่ง objdump -s filename โดยแทน filename ด้วย ชื่อไฟล์ที่ต้องการ จะได้ผลลัพธ์เป็น bytes code ของไฟล์นั้น ซึ่งในการทดลองนี้ จะทดสอบการทำ n-gram ที่ n = 1-4 byte ตัวอย่างเช่น ผลลัพธ์จากการ extract bytes code "ff ff ff ff a3 75" ดังนี้

```
40fd50 ffffffff a3750a0a 0a0a0a0a 0a0a0a0a
40fd60 0a0a0a0a 0a0a0a0a ffffffff ffffffff
40fd70 ffffffff ffffffff ffffffff f003ffdf
40fd80 fc380f8f ff3a2f0f fffb2f27 fffbee77
40fd90 fff80ff3 fff007fb ffe003fd ffc001ff
40fda0 ffc001ff ff8000ff f80000ff fb80000f
40fdb0 fb8000ef fb8000ef f8c001ef f940018f
```

รูป 3 ตัวอย่าง objdump -s

n-gram	ff	a3	75
จำนวน	4	1	1

ตาราง 3 ตัวอย่าง Hexadecimal n-gram, n=1

n-gram	ffff	ffa3	a375
จำนวน	3	1	1

ตาราง 4 ตัวอย่าง Hexadecimal n-gram, n = 2

n-gram	ffffff	ffffa3	ffa375
จำนวน	2	1	1

ตาราง 5 ตัวอย่าง Hexadecimal n-gram, n = 3

n-gram	ffffffff	ffffffa3	ffffffa375	ffffa375
จำนวน	1	1	1	1

ตาราง 6 ตัวอย่าง Hexadecimal n-gram, n = 4

3.2.2 Basic block ของ Operation code (Opcode)

ใช้คำสั่ง objdump -d filename โดยแทน filename ด้วย ชื่อไฟล์ที่ต้องการ แล้วนำเอา Opcode มาเรียง ต่อกัน เช่น addl-cmp-jz-subl-subf-call-mov-push แล้วตัดด้วย Opcode ที่เปลี่ยนแปลงเส้นทางการทำงานของโปรแกรม เช่น jump, branch, return จะแยกได้ว่า พบ addl-cmp, subl-subf, mov-push อย่างละ 1 ชุด หรือสร้างเป็นตารางได้ดังนี้

```
40e2d8:       05 0a d8 5b a1          addl    $0xa15bd80a,%eax
40e2dd:       be 3c 00 56 38          cmp     $0x3856003c,%esi
40e2e2:       ac                      jz      %ds:(%esi),%al
40e2e3:       bf 5f fa 9f 12          subl    $0x129ffa5f,%edi
40e2e8:       1e                      subf    %ds
40e2e9:       2d b7 e5 f6 6c          call    $0x6cf6e5b7,%eax
40e2ee:       bc a3 0f 93 49          mov     $0x49930fa3,%esp
40e2f3:       55                      push    %rbp
```

รูป 4 ตัวอย่าง objdump -d

basic block	addl-cmp	subl-subf	mov-push
จำนวน	1	1	1

ตาราง 7 ตัวอย่าง Opcode basic block

3.2.3 n-gram ของ Operation code (Opcode)

ทำแบบเดียวกันกับ 3.2.2 แต่เปลี่ยนวิธีการตัด โดยให้ตัดเป็นแบบ n-gram เช่น ถ้าได้ลำดับของ Opcode เป็น addl-cmp-jz-subl-subf-call-mov-push จะสามารถตัดได้ดังนี้

n-gram	addl-cmp	ffa3	a375
จำนวน	3	1	1

ตาราง 8 ตัวอย่าง Opcode n-gram, n = 2

n-gram	addl-cmp-jz	cmp-jz-subl	jz-subl-subf	subl-subf-call
จำนวน	1	1	1	1

ตาราง 9 ตัวอย่าง Opcode n-gram, n = 3

3.2.4 จำนวนบรรทัดของแต่ละ Section

ใช้คำสั่ง objdump -d filename โดยแทน filename ด้วย ชื่อไฟล์ที่ต้องการ แล้วนับจำนวนบรรทัดของ section ที่สนใจจำนวน 16 section ดังนี้ .bss, .buildid, .data, .edata, .eh_frame, .gnu_debuglink, .idata, .rdata, .reloc, .rsrc, .text, .tls, CODE, DATA, UPX0, UPX1 เช่น

```
Disassembly of section CODE:
0000000100401910 <_ZNSt14numeric_limitsIlE3maxEv>:
   100401910:   55                      push   %rbp
   100401911:   48 89 e5                mov    %rsp,%rbp
   100401914:   48 b8 ff ff ff ff ff    movabs $0x7fffffffffffffff,%rax
   10040191b:   ff ff 7f
   10040191e:   5d                      pop    %rbp
   10040191f:   c3                      retq
Disassembly of section UPX0:
0000000100401920 <_ZNSt14numeric_limitsIlE3minEv>:
   100401920:   55                      push   %rbp
   100401921:   48 89 e5                mov    %rsp,%rbp
   100401924:   48 b8 00 00 00 00 00    movabs $0x8000000000000000,%rax
   10040192b:   00 00 80
   10040192e:   5d                      pop    %rbp
```

รูป 5 ตัวอย่าง objdump -d

section	CODE	UPX0
จำนวน	6	5

ตาราง 10 ตัวอย่าง section line count

3.3 Feature selection

คำนวณค่า TF-IDF ของ attribute ทุกค่าตามสูตรในหัวข้อ 2.4.2.4 แล้วเรียงตามลำดับจากมากไปน้อย จากนั้นเลือก attribute ที่มีค่า TF-IDF มากสุดจำนวน 8192 attribute มาใช้ในขั้นตอนถัดไป

3.4 การทดสอบแยกประเภท

3.4.1 การทดสอบการแยกประเภทเบื้องต้นด้วย 8192 attribute

การแยกประเภทจะทดสอบกับ classifier 3 แบบ คือ random forest, extreme gradient boosting, multilayer perceptron ร่วมกับกระบวนการทดสอบแบบ 10-fold cross validation คือ แบ่งชุดข้อมูลออกเป็น 10 ส่วน แล้วนำ 9 ส่วนเข้าสู่กระบวนการเรียนรู้ อีก 1 ส่วนที่เหลือใช้ในการทดสอบเพื่อประเมินประสิทธิภาพของ model ที่ได้รับการเรียนรู้มา และวนซ้ำเช่นนี้ 10 รอบ โดยสลับชุดข้อมูลในส่วนที่ใช้ในการทดสอบและการเรียนรู้ทุกครั้ง ซึ่งวิธีการเช่นนี้ มีข้อดี คือ สามารถใช้ได้กับข้อมูลที่มีจำนวนน้อยจนไม่สามารถแบ่งชุดข้อมูลออกเป็น training, testing และ validating ได้อย่างชัดเจน และยังสามารถป้องกันการเกิดความลำเอียง (bias) จนเกิดเหตุการณ์ overfitting ได้

นำข้อมูลข้างต้นจาก feature selection จำนวน 8192 attributes ยกเว้น Section line count ซึ่งมี 16 attributes ผ่านการปรับค่า hyperparameter ด้วยค่าดังต่อไปนี้

random forest	n_estimators	10	50	100	200	500
	max_features	sqrt	log2	0.1	0.25	0.5
extreme gradient boosting	n_estimators	100	500	1000	2500	
	learning_rate	0.1	0.06	0.02	0.01	
multilayer perceptron	hidden_layer_sizes	20	50	100	200	500
	max_iter	50	100	200	400	

ตาราง 11 Grid search hyperparameter tuning

จากนั้นเข้าสู่การทดสอบแบบ 10-fold cross validation และเพิ่มการทดสอบนำเอา attribute ต่างประเภทกันมารวมกัน โดยคาดหวังความแม่นยำที่มากขึ้นจากการที่ใช้ feature หลายประเภท

3.4.2 Soft voting

นำเอาผลลัพธ์จาก 3.4.1 มาพิจารณา แล้วใช้ classifier ที่ต่างกันสองชนิดที่ได้ผลลัพธ์ดีที่สุดร่วมกัน vote ด้วยอัตราส่วนต่าง ๆ กัน ร่วมกับ feature ชนิดที่ได้ผลลัพธ์ดีที่สุด

3.4.3 Recursive feature elimination

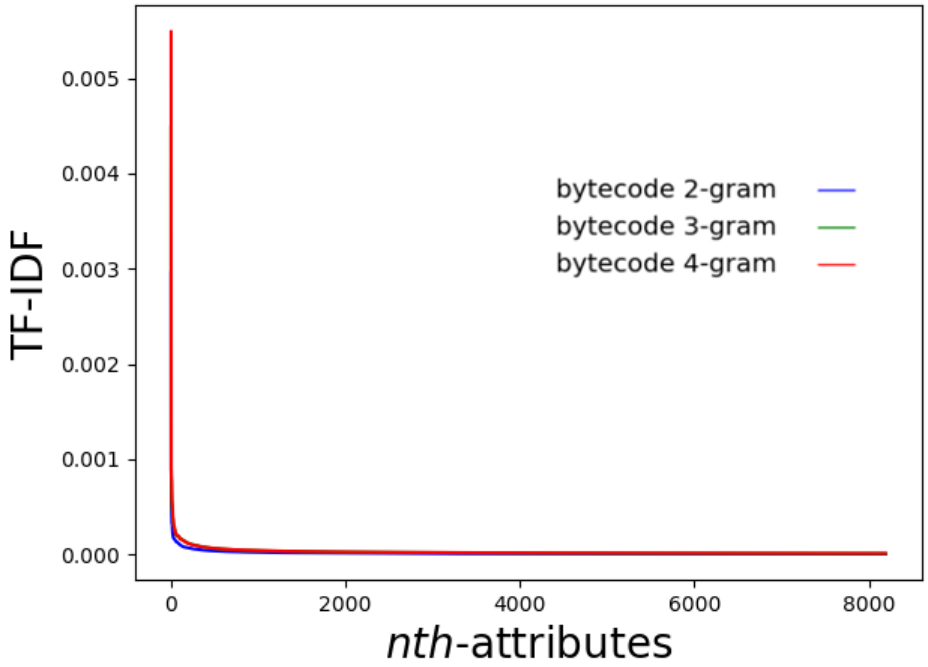
ขั้นตอนนี้จะนำเอา feature ชนิดที่ได้ค่าผลลัพธ์สูงสุดจาก 3.4.1 มาลด จำนวน attribute ขั้นตอนนี้จะทำการลดจำนวน feature ลงทีละครึ่งหนึ่งโดยดูจาก feature_importance_ ของ feature แต่ละตัว ซึ่งจะได้จากการ fit ตัว model เข้ากับ train data โดยค่า feature_importance_ ของ random forest จะขึ้นอยู่กับ impurity ที่เลือกไว้ ในที่นี้ใช้เป็น Gini และ ใน extreme gradient boosting จะขึ้นอยู่กับจำนวนครั้งที่ feature ถูกเลือกไปใช้ในต้นไม้

บทที่ 4. ผลการทดลอง

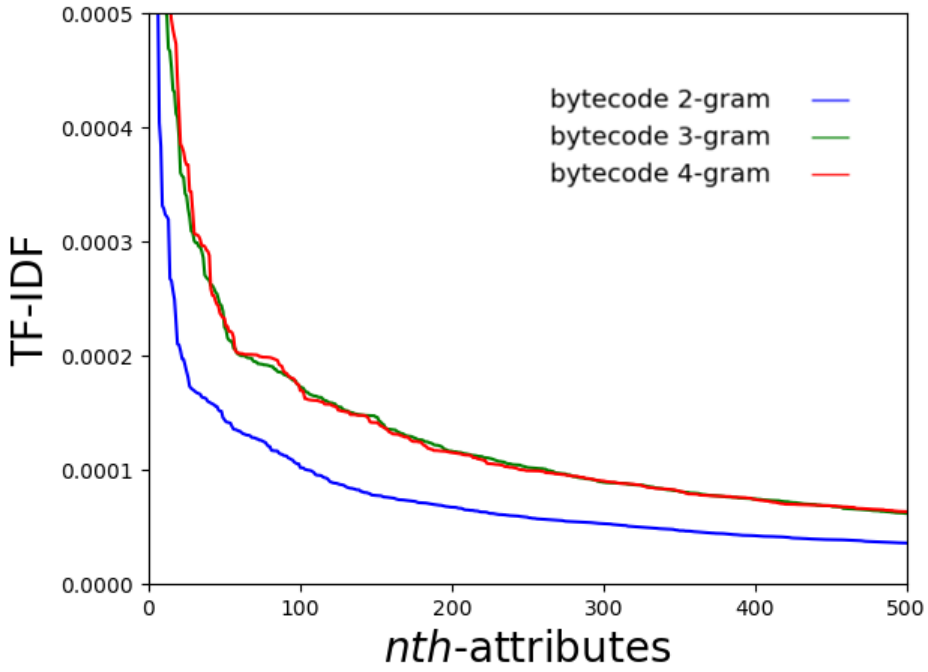
4.1 ผล feature selection

กราฟผลลัพธ์ของค่า TF-IDF จากการคำนวณทุก attribute แล้วเลือก 8192 attribute ที่ได้ค่า TF-IDF มากสุด จากนั้นเรียงค่าจากมากไปน้อย แล้ววาดกราฟแยกกัน แบ่งออกเป็นสองชนิด คือ bytecode รูป 6 ขยายในรูป 7 และ opcode รูป 8 ขยายในรูป 9 โดยมีแกน x แทน attribute และ แกน y แทนค่าที่ได้จากการคำนวณตามสูตร TF-IDF

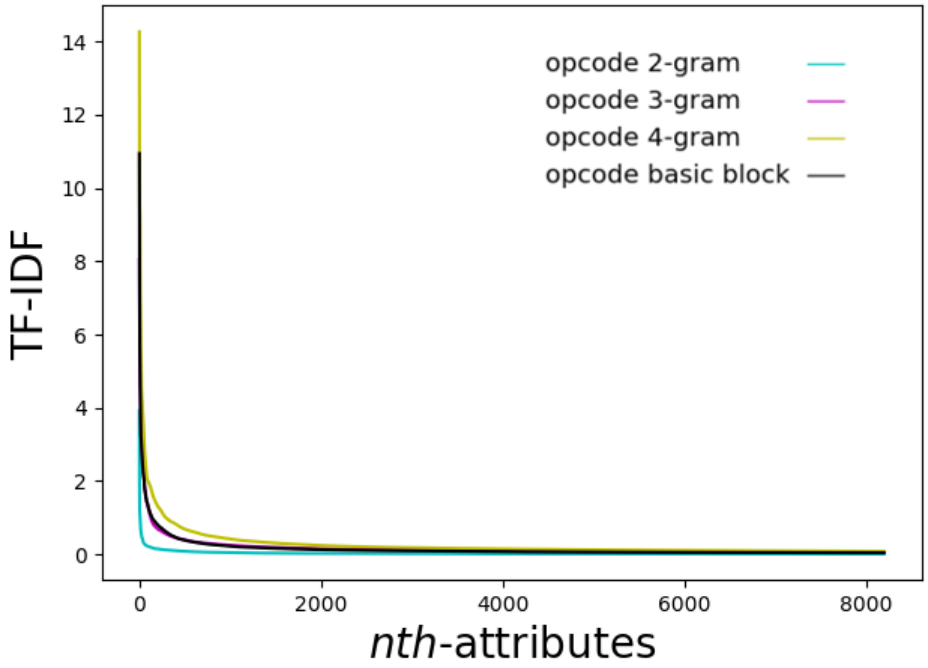
เปรียบเทียบ รูป 6 กับ รูป 8 แสดงถึงค่า TF-IDF ที่แตกต่างกันมากเนื่องจากชนิดของ feature ที่แตกต่างกัน และ รูป 7 กราฟขยาย TF-IDF ของ bytecode แสดงถึงค่า TF-IDF ที่ได้จาก bytecode 3-gram และ 4-gram ที่ใกล้เคียงกันเกือบจะเป็นเส้นทับกันสนิท เช่นเดียวกับ รูป 9 กราฟขยาย TF-IDF ของ opcode ที่มี opcode 3-gram ใกล้เคียงกับ opcode basic block



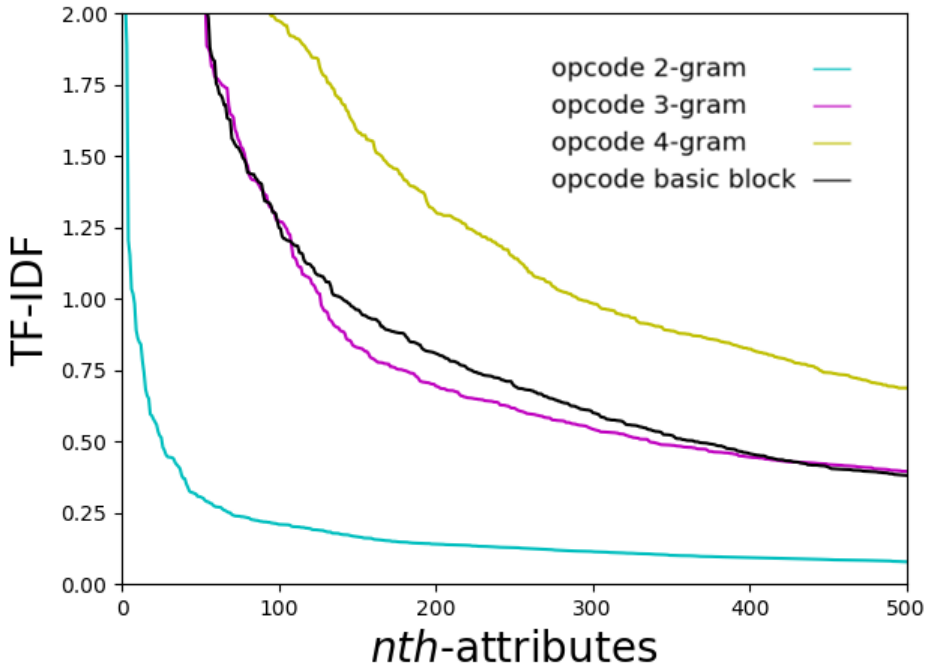
รูป 6 กราฟ TF-IDF ของ bytecode



รูป 7 กราฟขยาย TF-IDF ของ bytecode



รูป 8 กราฟ TF-IDF ของ opcode



รูป 9 กราฟขยาย TF-IDF ของ opcode

4.2 ผลการทดสอบการแยกประเภทเบื้องต้นด้วย 8192 attribute

ผลลัพธ์จากขั้นตอน 4.2 ผลการทดสอบการแยกประเภทเบื้องต้นด้วย 8192 attribute ด้วย 3 classifier คือ random forest, extreme gradient boosting, multilayer perceptron แสดงถึงค่า F_1-score มีการทดสอบนำเอา feature ต่างชนิดกันมาร่วมกัน ในชื่อของ Hybrid type 1 เกิดจาก bytecode 4-gram, opcode basic block, section line count และ Hybrid type 2 เกิดจาก bytecode 4-gram, 3-gram opcode, section line count

	F_1 score				
	0.91874	0.921824	0.917763	0.898839	0.909688
Section line count	0.905844	0.901503	0.907916	0.909688	0.92437
	0.936378	0.938111	0.925325	0.927869	0.940032
Bytecode 2-gram	0.938111	0.938111	0.926829	0.928453	0.943144
	0.960784	0.956098	0.939245	0.949919	0.958065
Bytecode 3-gram	0.950081	0.955272	0.946688	0.934426	0.965289
	0.965964	0.957237	0.942904	0.946341	0.952998
Bytecode 4-gram	0.957929	0.959481	0.9504	0.931148	0.965058
	0.938111	0.943089	0.931818	0.920065	0.932258
Opcode 2-gram	0.928571	0.948553	0.924559	0.92916	0.951747
	0.947883	0.943639	0.930308	0.930533	0.936306
Opcode 3-gram	0.940032	0.931818	0.933764	0.932897	0.93823
	0.946688	0.935065	0.939245	0.926045	0.94061
Opcode 4-gram	0.938907	0.935065	0.930757	0.922824	0.950166
	0.938511	0.930757	0.935484	0.9312	0.935691
Opcode basic block	0.9374	0.9408	0.925806	0.926108	0.933333
	0.967532	0.956098	0.944625	0.943639	0.951299
Hybrid type 1	0.962963	0.96129	0.945338	0.92459	0.968491
	0.964052	0.960912	0.949097	0.946341	0.951299
Hybrid type 2	0.961165	0.956522	0.943639	0.922824	0.966887

ตาราง 12 Random forest F_1-score

	F_1 score				
	0.907591	0.90939	0.906096	0.880399	0.893688
Section line count	0.906149	0.88	0.896104	0.902801	0.912752
	0.955519	0.959083	0.947712	0.957377	0.949757
Bytecode 2-gram	0.95624	0.963696	0.942904	0.938436	0.961857
	0.970492	0.965404	0.955224	0.965854	0.960912
Bytecode 3-gram	0.96563	0.959612	0.954397	0.952381	0.970297
	0.966997	0.960656	0.957237	0.970492	0.954397
Bytecode 4-gram	0.972358	0.954545	0.958065	0.952381	0.978441
	0.941368	0.944625	0.939044	0.944625	0.938111
Opcode 2-gram	0.936995	0.951299	0.943639	0.947712	0.966777
	0.95581	0.949429	0.947712	0.950081	0.946341
Opcode 3-gram	0.955954	0.945161	0.938511	0.930921	0.961857
	0.952537	0.944444	0.941368	0.938312	0.936102
Opcode 4-gram	0.94382	0.934641	0.934189	0.919804	0.953488
	0.941176	0.944984	0.94822	0.946166	0.940032
Opcode basic block	0.94382	0.946515	0.930757	0.927393	0.953488
	0.975369	0.96563	0.960656	0.962357	0.95315
Hybrid type 1	0.959742	0.951299	0.956522	0.954098	0.971808
	0.970492	0.960526	0.960656	0.96563	0.94822
Hybrid type 2	0.958199	0.959481	0.962723	0.954545	0.975124

ตาราง 13 Extreme gradient boosting F_1-score

	F_1 score				
	0.8125	0.785841	0.819466	0.785586	0.761905
Section line count	0.818653	0.792727	0.816807	0.83304	0.810811
	0.931323	0.931148	0.907618	0.894155	0.904836
Bytecode 2-gram	0.92233	0.91354	0.88586	0.893688	0.912903
	0.944079	0.936791	0.92562	0.918138	0.921095
Bytecode 3-gram	0.943457	0.923328	0.91354	0.910596	0.948247
	0.946166	0.944262	0.931373	0.925566	0.931419
Bytecode 4-gram	0.949919	0.932476	0.910891	0.892063	0.951424
	0.916667	0.9216	0.899351	0.89404	0.919176
Opcode 2-gram	0.918239	0.9184	0.901099	0.910891	0.902479
	0.933764	0.939245	0.92953	0.91653	0.916272
Opcode 3-gram	0.932258	0.927769	0.921348	0.930693	0.941957
	0.938907	0.93575	0.918033	0.919614	0.916129
Opcode 4-gram	0.941748	0.943639	0.928803	0.918301	0.947195
	0.932258	0.939297	0.936102	0.922581	0.92283
Opcode basic block	0.928339	0.9374	0.926108	0.930921	0.938843
	0.916399	0.915309	0.898089	0.906752	0.91866
Hybrid type 1	0.921136	0.904065	0.900322	0.89211	0.906667
	0.911765	0.912338	0.89533	0.894231	0.897893
Hybrid type 2	0.912837	0.889256	0.907348	0.888889	0.890728

ตาราง 14 Multilayer perceptron F_1-score

ตาราง ค่า hyperparameter จาก grid search ด้วยค่าจาก ตาราง 11 Grid search hyperparameter tuning ซึ่งผลที่ได้ มีแนวโน้มจะใช้ค่า n_estimators, max_features, max_iter, hidden_layer_sizes ที่มาก และ ใช้เวลาในการคำนวณมากขึ้น เพื่อให้ได้ค่า F_1-score ที่สูงขึ้น

	Random forest		XGBoost		Multilayer perceptron	
	n_estim	max_feat	n_estim	learn_rate	max_iter	hid_layer
Section line count	200	sqrt	500	0.1	200	500
Bytecode 2-gram	500	0.25	1000	0.1	200	100
Bytecode 3-gram	500	0.5	2500	0.02	400	100
Bytecode 4-gram	500	0.5	1000	0.06	50	20
Opcode 2-gram	100	0.5	2500	0.1	100	500
Opcode 3-gram	100	0.25	2500	0.06	50	100
Opcode 4-gram	200	0.25	1000	0.1	100	500
Opcode basic block	200	0.1	2500	0.06	400	200
Hybrid type 1	500	0.25	1000	0.06	400	500
Hybrid type 2	500	0.5	500	0.06	100	100

ตาราง 15 Hyperparameter ที่ใช้

4.3 ผลการทดสอบ soft voting

ตารางแสดงผลการใช้สอง classifier ร่วมกันโหวตในอัตราส่วนต่าง ๆ โดยเลือก classifier ที่ได้ผลลัพธ์ดีที่สุดจากขั้นตอนก่อนหน้า คือ random forest และ extreme gradient boosting ร่วมกับ bytecode 3-gram และ bytecode 4-gram โดยช่องสองช่องแรกของแต่ละตารางแสดงถึงค่าอัตราส่วนการโหวต ตาราง 18 และ ตาราง 20 แสดงค่า F_1-score ส่วน ตาราง 19 และ ตาราง 21 แสดงเวลาที่ใช้ในการเรียนรู้ หน่วยเป็นวินาที และแต่ละช่องแสดงผลลัพธ์ของแต่ละ fold จากการทดสอบแบบ 10 fold cross validation

เปรียบเทียบการใช้ classifier สองชนิดร่วมกัน สังเกตได้ว่าจะต้องใช้เวลาในการเรียนรู้เพิ่มขึ้นเป็นผลรวมของทั้งสองวิธี แต่ผลของ F_1 score ที่ได้ เปลี่ยนแปลงเพียงเล็กน้อย ทั้งนี้ขึ้นอยู่กับ Hyperparameter ที่ใช้แตกต่างกันใน bytecode 3-gram และ bytecode 4-gram

Random forest	F_1 score	0.960784	0.956098	0.939245	0.949919	0.958065
		0.950081	0.955272	0.946688	0.934426	0.965289
	training time(s)	411.116	416.4896	400.3873	407.2481	402.392
		412.506	405.1465	403.7491	398.9403	405.2453
Extreme gradient boosting	F_1 score	0.970492	0.965404	0.955224	0.965854	0.960912
		0.96563	0.959612	0.954397	0.952381	0.970297
	training time(s)	186.6477	185.6745	185.5037	188.239	188.3628
		187.0251	185.876	186.9899	186.5846	186.882

ตาราง 16 สำหรับเปรียบเทียบ bytecode 3-gram

Random forest	F_1 score	0.965964	0.957237	0.942904	0.946341	0.952998
		0.957929	0.959481	0.9504	0.931148	0.965058
	training time(s)	279.6148	283.139	280.496	282.1405	283.0631
		286.517	286.457	283.7403	281.1444	285.1342
Extreme gradient boosting	F_1 score	0.966997	0.960656	0.957237	0.970492	0.954397
		0.972358	0.954545	0.958065	0.952381	0.978441
	training time(s)	57.67496	57.75847	57.6718	58.02094	58.3006
		58.54206	57.73437	58.14617	57.82056	58.55787

ตาราง 17 สำหรับเปรียบเทียบ bytecode 4-gram

RF	XGB	F_1 score				
		0.949757	0.951299	0.953748	0.96732	0.944625
95	5	0.942904	0.965289	0.944984	0.955954	0.960784
		0.951456	0.949593	0.955272	0.96563	0.944625
90	10	0.942904	0.966887	0.943457	0.957655	0.960912
		0.952998	0.951299	0.959872	0.967105	0.943089
85	15	0.944444	0.965289	0.94686	0.955954	0.964286
		0.952998	0.951299	0.956661	0.963696	0.943089
80	20	0.944444	0.966997	0.950081	0.96248	0.962723
		0.952846	0.95114	0.956661	0.960396	0.946166
75	25	0.944444	0.970297	0.948387	0.96248	0.961039
		0.952846	0.952846	0.956661	0.960396	0.947712
70	30	0.94599	0.966997	0.948553	0.96248	0.964169
		0.952692	0.954397	0.954984	0.960396	0.949429
65	35	0.949429	0.966997	0.951768	0.967427	0.965742
		0.954397	0.954397	0.954984	0.960396	0.947712
60	40	0.954248	0.966997	0.954984	0.969005	0.964169
		0.956098	0.954397	0.954984	0.960396	0.947541
55	45	0.954248	0.966997	0.954984	0.969005	0.965854
		0.957655	0.954397	0.956522	0.960396	0.947541
50	50	0.954248	0.968699	0.954984	0.969005	0.965854
		0.956098	0.955954	0.958199	0.960396	0.947541
45	55	0.95581	0.970395	0.953451	0.969005	0.965854
		0.957655	0.954248	0.959742	0.958678	0.947541
40	60	0.95581	0.970395	0.951923	0.969005	0.964169
		0.957655	0.955954	0.959742	0.958678	0.95082
35	65	0.957377	0.970395	0.953451	0.970588	0.965742
		0.959217	0.955954	0.96129	0.958541	0.954098
30	70	0.957377	0.970297	0.953451	0.970588	0.965742
		0.959217	0.957516	0.96129	0.960133	0.954098
25	75	0.957377	0.970297	0.956522	0.97377	0.96732
		0.959217	0.957516	0.96129	0.960133	0.954098
20	80	0.958949	0.970297	0.958065	0.975369	0.96732
		0.960784	0.957516	0.96129	0.96173	0.954098
15	85	0.958949	0.970297	0.958065	0.975369	0.96732
		0.959083	0.955954	0.96129	0.96173	0.955665
10	90	0.962357	0.967105	0.958065	0.975369	0.96732
		0.955665	0.955954	0.96129	0.96173	0.958949
5	95	0.964052	0.967105	0.959612	0.975369	0.96732

ตาราง 18 soft voting bytecode 3-gram F_1-score

RF	XGB	Training time (seconds)				
		548.1228	547.1712	546.235	544.852	541.9255
95	5	538.9531	548.3148	547.9313	543.0637	539.2657
		548.7124	546.5205	538.2004	549.0445	540.2275
90	10	535.322	553.2894	545.859	535.9654	538.1889
		549.8479	542.5294	542.9338	549.4916	544.9455
85	15	534.1304	546.0784	554.3988	543.933	541.4058
		550.7633	549.3439	541.0327	551.5405	545.4662
80	20	544.3503	558.0095	551.9327	544.0027	543.6986
		549.133	549.9144	541.2956	596.8556	578.93
75	25	591.6644	601.9724	599.0254	602.3081	582.7764
		600.8999	686.8293	666.3142	644.0637	664.1239
70	30	678.6719	692.6188	687.716	682.3225	672.2794
		619.1229	554.0416	550.164	572.244	547.3033
65	35	549.015	552.4025	553.278	545.1317	545.7128
		564.9137	551.7041	546.0957	552.4265	544.4094
60	40	549.2594	558.7407	558.3028	551.7932	546.3686
		553.5483	550.6734	546.9911	569.9094	549.5022
55	45	541.7601	559.1075	555.035	545.6564	541.9817
		550.4249	550.8421	553.3095	552.7928	548.8304
50	50	537.3755	558.0272	559.8085	542.4649	552.7396
		549.425	548.781	545.8811	564.0914	549.9785
45	55	542.5917	569.4152	563.8339	543.8495	556.6543
		553.5364	560.6469	555.7177	552.5549	545.9402
40	60	554.6628	557.0952	562.4716	538.9749	544.9276
		565.8433	557.486	549.6655	545.9628	547.8516
35	65	552.2016	569.047	552.467	555.3149	555.7934
		552.988	553.3716	555.7021	561.3473	547.7827
30	70	535.9352	555.447	561.5705	544.7022	545.0765
		552.717	559.5806	545.6719	551.9544	545.9686
25	75	541.2624	566.2188	558.1752	546.8944	547.1375
		558.5384	552.8532	547.8138	551.8878	561.6123
20	80	542.565	564.2747	557.8378	544.6115	551.8171
		550.8037	560.2612	547.055	553.5616	548.8027
15	85	539.503	558.995	552.1098	543.8963	556.7009
		555.6015	555.733	549.2049	553.524	549.6476
10	90	543.0316	558.7429	550.8259	546.0286	559.1076
		559.0336	561.6198	547.2308	553.1661	556.9582
5	95	542.5779	552.5453	550.2883	555.4006	546.0493

ตาราง 19 soft voting bytecode 3-gram training time

RF	XGB	F_1 score				
		0.941558	0.951299	0.9504	0.95581	0.94599
95	5	0.946166	0.956954	0.950081	0.960656	0.964052
		0.943457	0.954397	0.9504	0.957516	0.944444
90	10	0.952846	0.956954	0.948553	0.963934	0.965742
		0.951456	0.956098	0.955128	0.95581	0.946166
85	15	0.954397	0.956954	0.950081	0.963934	0.965742
		0.951456	0.956098	0.955128	0.957377	0.947883
80	20	0.954397	0.962109	0.950081	0.962233	0.967427
		0.951456	0.957792	0.955128	0.959083	0.949593
75	25	0.956098	0.963576	0.950081	0.963934	0.965854
		0.952998	0.95935	0.9568	0.960656	0.95114
70	30	0.956098	0.966887	0.951768	0.963934	0.965854
		0.954545	0.95935	0.956661	0.960656	0.952846
65	35	0.957655	0.968595	0.951768	0.96563	0.967427
		0.957792	0.95935	0.956661	0.960656	0.952846
60	40	0.95935	0.968595	0.955128	0.96563	0.967427
		0.957792	0.957655	0.956661	0.958949	0.952846
55	45	0.95935	0.970297	0.956661	0.96563	0.970684
		0.957792	0.957655	0.958333	0.957237	0.952692
50	50	0.960912	0.971993	0.956661	0.96563	0.972268
		0.95624	0.956098	0.958333	0.957237	0.954248
45	55	0.960912	0.971993	0.956661	0.968801	0.970588
		0.95624	0.952692	0.958333	0.961983	0.957516
40	60	0.962602	0.973597	0.958199	0.968699	0.972268
		0.957792	0.954248	0.958333	0.961983	0.959083
35	65	0.964169	0.971901	0.953451	0.968699	0.970684
		0.957792	0.954248	0.958333	0.961983	0.957516
30	70	0.964169	0.971901	0.951923	0.970297	0.970684
		0.95935	0.954248	0.96	0.961983	0.959217
25	75	0.964169	0.971901	0.953451	0.968595	0.970684
		0.962602	0.954248	0.961538	0.963696	0.959217
20	80	0.964169	0.971901	0.953451	0.968595	0.970684
		0.962602	0.954248	0.961538	0.963696	0.959217
15	85	0.964169	0.971901	0.951768	0.968595	0.969005
		0.962602	0.954248	0.961538	0.963696	0.96248
10	90	0.96248	0.970199	0.951768	0.968595	0.96732
		0.960912	0.954248	0.961538	0.963696	0.96248
5	95	0.960912	0.970199	0.948553	0.968595	0.96732

ตาราง 20 soft voting bytecode 4-gram F_1-score

RF	XGB	Training time (seconds)				
		380.3069	383.6277	373.334	390.3916	382.3845
95	5	378.7735	383.1581	383.3226	385.4598	382.8347
		382.442	387.0219	326.8564	331.9251	330.2997
90	10	328.6427	333.3251	334.5166	327.2999	326.2511
		329.2093	334.4295	324.6114	334.429	327.0246
85	15	325.9377	329.9675	332.9894	328.8671	330.6008
		332.3136	335.5472	328.8137	331.4049	331.3218
80	20	325.3432	329.5813	336.8897	331.1849	328.3695
		333.5349	338.187	327.7354	330.9571	326.5914
75	25	327.7346	330.5233	330.0249	332.4897	326.6258
		335.5395	341.6868	325.5346	331.7397	332.6239
70	30	330.0509	329.9961	334.922	332.0929	326.7613
		329.7146	334.6228	326.4623	332.4488	327.512
65	35	325.0254	336.6644	330.6926	330.304	326.6495
		330.361	334.3798	324.6177	336.833	327.8682
60	40	324.696	333.9014	330.1797	335.3048	326.1149
		329.7419	334.8629	327.999	336.4243	326.1477
55	45	325.4301	337.3929	329.5309	333.5271	331.7382
		330.9194	337.5508	325.1618	331.4073	326.0817
50	50	327.9638	335.2235	330.7496	333.4549	329.8881
		330.0647	335.1045	324.8587	331.7442	329.5858
45	55	325.1436	333.6624	330.2218	328.1725	326.0338
		328.2131	333.9862	324.8108	333.6149	326.8198
40	60	324.4588	329.3599	336.0914	328.5544	327.2231
		329.7634	333.7751	325.116	331.0656	331.6017
35	65	331.9101	331.5619	329.5369	327.1656	329.4656
		329.7908	336.8351	327.2185	333.9605	329.8065
30	70	327.3959	333.3597	330.4111	327.7356	326.3766
		330.1053	337.2891	324.9854	336.0383	329.9376
25	75	324.6728	330.1358	330.4044	331.0061	327.1103
		332.7134	335.5637	328.5276	335.5376	331.4691
20	80	326.0243	335.7048	334.7486	333.0397	330.3283
		332.75	337.384	328.6903	336.7639	327.3938
15	85	325.256	330.1895	335.8067	334.35	330.8291
		331.7974	336.5876	329.3725	332.2533	330.0808
10	90	328.8355	332.0034	330.0615	330.4889	329.1902
		329.9475	336.8433	329.459	330.927	330.2236
5	95	331.9019	329.3942	330.6581	328.152	330.7542

ตาราง 21 soft voting bytecode 4-gram training time

4.4 ผลการทดสอบ recursive feature elimination

ตารางแสดงผลจากขั้นตอน 3.4.3 Recursive feature elimination ใช้ classifier เป็น random forest และ extreme gradient boosting ร่วมกับชุดข้อมูลที่ผลลัพธ์ดีที่สุดคือ bytecode 4-gram โดยแต่ละตารางแสดงจำนวน feature ที่ลดลงทีละครึ่งหนึ่ง ตาราง 22 และ ตาราง 24 แสดงค่า F_1-score ส่วน ตาราง 23 และ ตาราง 25ตารางที่สองแสดงเวลาที่ใช้ในการเรียนรู้ หน่วยเป็นวินาที แต่ละช่องแสดงผลลัพธ์ของแต่ละ fold จากการทดสอบแบบ 10 fold cross validation แสดงให้เห็นถึงค่าของ F_1 score ที่จะค่อย ๆ สูงขึ้นเล็กน้อยขณะลดจำนวน feature ด้วย recursive feature elimination จากนั้น ค่า F_1 score จะลดลงอย่างมาก และเมื่อ feature ลดน้อยลง เวลาที่ใช้ในการเรียนรู้ก็ลดลงเช่นกัน

n-features	F_1 score				
	0.953947	0.949264	0.965517	0.954397	0.949429
8192	0.946688	0.932039	0.959481	0.941368	0.962602
	0.954098	0.95098	0.965517	0.955954	0.947883
4096	0.946688	0.939837	0.957516	0.941368	0.961039
	0.954098	0.949264	0.962233	0.954248	0.95098
2048	0.952998	0.936791	0.960656	0.943274	0.962602
	0.955665	0.949264	0.963934	0.954248	0.954248
1024	0.951456	0.941748	0.964052	0.94822	0.964286
	0.959083	0.952692	0.962357	0.955954	0.95098
512	0.954693	0.943274	0.960656	0.946166	0.969005
	0.960656	0.954397	0.964052	0.95581	0.959083
256	0.957929	0.943089	0.956098	0.944625	0.972268
	0.959217	0.959217	0.962357	0.959083	0.962357
128	0.951299	0.951299	0.954693	0.957929	0.965742
	0.952692	0.952224	0.957655	0.954397	0.965404
64	0.946515	0.943274	0.95114	0.954839	0.96248
	0.943274	0.933993	0.955954	0.947368	0.95624
32	0.926829	0.933333	0.952537	0.951613	0.960912
	0.936585	0.927731	0.947541	0.939044	0.95114
16	0.915033	0.92233	0.942904	0.947883	0.95315
	0.933977	0.911519	0.931148	0.924837	0.935065
8	0.906149	0.903654	0.921311	0.9374	0.9184
	0.872375	0.86532	0.86711	0.85761	0.872611
4	0.856688	0.856672	0.863787	0.846405	0.873147
	0.825083	0.812903	0.8	0.805873	0.801272
2	0.808241	0.799353	0.807947	0.78374	0.795417

ตาราง 22 10cv random forest after recursive feature elimination F_1-score

n-features	Training time (seconds)				
	334.3028	324.2879	330.5853	331.6354	330.3613
8192	332.4574	331.9918	337.3824	327.982	322.7423
	244.4149	241.6352	243.304	244.9172	243.5293
4096	246.7081	247.9344	248.3303	246.9794	241.4414
	152.1916	150.3986	151.9791	152.4223	152.0746
2048	153.0679	153.7704	153.6813	154.5249	151.4609
	80.451	80.19829	82.14212	82.18493	81.65282
1024	81.79154	82.05381	82.95515	82.07695	81.15597
	42.91126	42.7475	43.11598	43.40692	43.46199
512	43.29943	43.60506	44.44384	43.32167	42.87063
	23.7193	23.39341	23.64238	23.52665	23.44468
256	23.7831	23.82678	23.63727	23.60822	23.62563
	12.71962	12.74325	12.72422	12.758	12.78536
128	12.7295	12.62059	12.79659	12.75634	12.60007
	7.292307	7.375827	7.523619	7.349585	7.321288
64	7.378535	7.281708	7.284372	7.401494	7.322462
	4.717587	4.770323	4.652002	4.687623	4.652907
32	4.788276	4.687985	4.681046	4.797083	4.69311
	3.072632	3.179775	3.174773	3.157466	3.175209
16	3.035422	3.074319	3.072541	3.171642	3.173739
	2.67344	2.666714	2.548093	2.671429	2.643714
8	2.537924	2.666053	2.611054	2.662234	2.669755
	2.565	2.569542	2.570369	2.665812	2.574248
4	2.562139	2.565806	2.543171	2.545693	2.555265
	2.773582	2.649969	2.66821	2.804163	2.814741
2	2.654922	2.662941	2.687482	2.681493	2.738063

ตาราง 23 10cv random forest after recursive feature elimination training time

n-features	F_1 score				
	0.961165	0.958814	0.958541	0.960912	0.958814
8192	0.962357	0.967427	0.973941	0.971061	0.97377
	0.962602	0.958949	0.956811	0.95935	0.957237
4096	0.96563	0.969106	0.972268	0.971061	0.969005
	0.962723	0.962233	0.956811	0.964169	0.960526
2048	0.959217	0.96732	0.972268	0.972625	0.973856
	0.964286	0.963816	0.958541	0.960912	0.960526
1024	0.965742	0.974026	0.97561	0.971061	0.976974
	0.965964	0.962233	0.961857	0.96248	0.962233
512	0.962357	0.978862	0.974026	0.974277	0.980328
	0.962602	0.963816	0.970199	0.962602	0.967105
256	0.96732	0.972268	0.980519	0.977346	0.981878
	0.967427	0.960396	0.968595	0.967427	0.962233
128	0.965742	0.970588	0.980519	0.980519	0.981878
	0.957792	0.950495	0.960133	0.958949	0.965517
64	0.962602	0.969305	0.970684	0.960656	0.963934
	0.946515	0.955519	0.956811	0.951299	0.948929
32	0.958678	0.9504	0.962602	0.951456	0.95114
	0.938511	0.921273	0.925125	0.922314	0.937705
16	0.933764	0.918831	0.94599	0.936791	0.92635
	0.897436	0.891486	0.872727	0.906149	0.90671
8	0.900813	0.885993	0.91256	0.909968	0.898361
	0.837748	0.841216	0.850847	0.844371	0.872727
4	0.859038	0.826087	0.893964	0.858553	0.837359
	0.739726	0.727273	0.762821	0.755906	0.745283
2	0.744479	0.720859	0.772586	0.71406	0.725581

ตาราง 24 10cv extreme gradient boosting after recursive feature elimination F_1-score

n-features	Training time (seconds)				
	59.4456	59.95857	58.58048	58.95883	58.22911
8192	58.28445	59.49568	59.69072	58.51636	58.17337
	34.03303	34.07373	34.14031	34.16567	34.11697
4096	34.13087	34.05107	34.59342	34.18687	34.47485
	23.02132	22.9236	22.863	22.89243	22.86084
2048	22.90339	22.87262	23.00514	22.97485	22.93025
	16.09553	16.08206	16.15152	16.05579	16.15797
1024	16.15266	16.07178	16.26156	16.16933	16.12267
	8.950951	8.888672	8.865063	8.869333	8.903448
512	8.84118	8.894274	8.981919	8.926711	8.919433
	5.097747	5.240868	5.098294	5.221982	5.222028
256	5.215514	5.238912	5.276337	5.221573	5.242783
	3.204455	3.215703	3.213676	3.200609	3.197729
128	3.198352	3.207939	3.208846	3.206023	3.224841
	2.367793	2.327777	2.358153	2.329773	2.329243
64	2.349115	2.355445	2.354427	2.355806	2.382621
	1.758152	1.792748	1.80002	1.82012	1.769325
32	1.76527	1.776224	1.765819	1.753735	1.759976
	1.612942	1.606234	1.579082	1.586821	1.666362
16	1.664251	1.65401	1.630197	1.542736	1.535221
	1.409796	1.421107	1.410854	1.400417	1.392649
8	1.396396	1.396727	1.392233	1.394638	1.405561
	1.386383	1.334137	1.296646	1.306085	1.298842
4	1.319989	1.318659	1.305331	1.325563	1.315311
	1.223465	1.289644	1.24785	1.226521	1.22476
2	1.242899	1.232939	1.229603	1.263252	1.269223

ตาราง 25 10cv extreme gradient boosting after recursive feature elimination training time

บทที่ 5. วิเคราะห์ผลการทดลอง

5.1 วิเคราะห์ผลการทดสอบการแยกประเภทเบื้องต้นด้วย 8192 attribute

จาก 4.2 ผลการทดสอบการแยกประเภทเบื้องต้นด้วย 8192 attribute ตาราง 26 แสดงค่า F_1-score, train time แสดงค่าเฉลี่ยจากทั้ง 10 fold โดย Hybrid type 1 เกิดจาก bytecode 4-gram, opcode basic block, section line count ต่อกัน และ Hybrid type 2 เกิดจาก bytecode 4-gram, 3-gram opcode, section line count ต่อกัน ผลลัพธ์จากการใช้ bytecode 4-gram และ bytecode 3-gram ร่วมกับ extreme gradient boosting ได้ผลดีที่สุด รองลงมาคือ random forest ส่วนการนำ attribute ต่างชนิดมาใช้ร่วมกันแสดงให้เห็นว่าผลลัพธ์ไม่ได้ดีขึ้นกว่าเดิม โดยเวลาที่ใช้ในการเรียนรู้แต่ละ model แตกต่างกันไปตามจำนวน attributes และ hyperparameter ใน ตาราง 15

	Random forest	Extreme gradient boosting	Multilayer perceptron
Section line count	0.911617446 0.960673928	0.899496823 0.601303816	0.803733538 4.423258471
Bytecode 2-gram	0.934236232 418.0477082	0.953258208 146.6882283	0.909740064 108.6704027
Bytecode 3-gram	0.951586605 406.3220317	0.962020279 186.7785318	0.928489091 60.04135408
Bytecode 4-gram	0.952946072 283.1446204	0.962556825 58.02277942	0.931556052 15.73045666
Opcode 2-gram	0.934793164 100.7122748	0.94541971 200.3577195	0.910194129 320.4567896
Opcode 3-gram	0.936541083 45.56688328	0.948177895 184.1514444	0.928936619 81.29305699
Opcode 4-gram	0.936537172 67.06175089	0.93987054 59.42534688	0.930811713 390.6534435
Opcode basic block	0.93350912 12.80160551	0.942255274 62.89452643	0.93146785 108.9697778
Hybrid type 1	0.952586538 293.9095358	0.961063113 117.7235218	0.907950843 577.3713723
Hybrid type 2	0.952273918 397.7922876	0.961559744 46.88780761	0.900061485 78.90536661

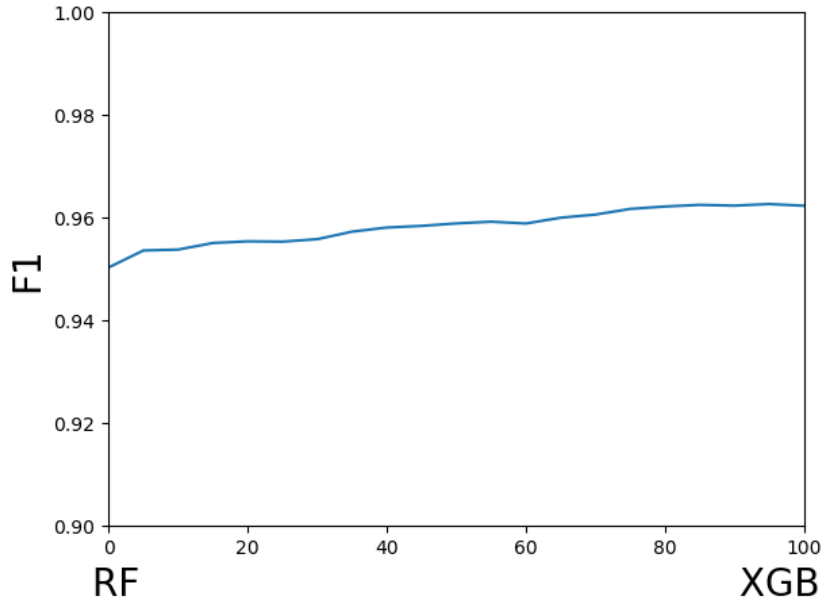
ตาราง 26 แสดงค่า F_1-score, train time

5.2 วิเคราะห์ผลการทดสอบ soft voting

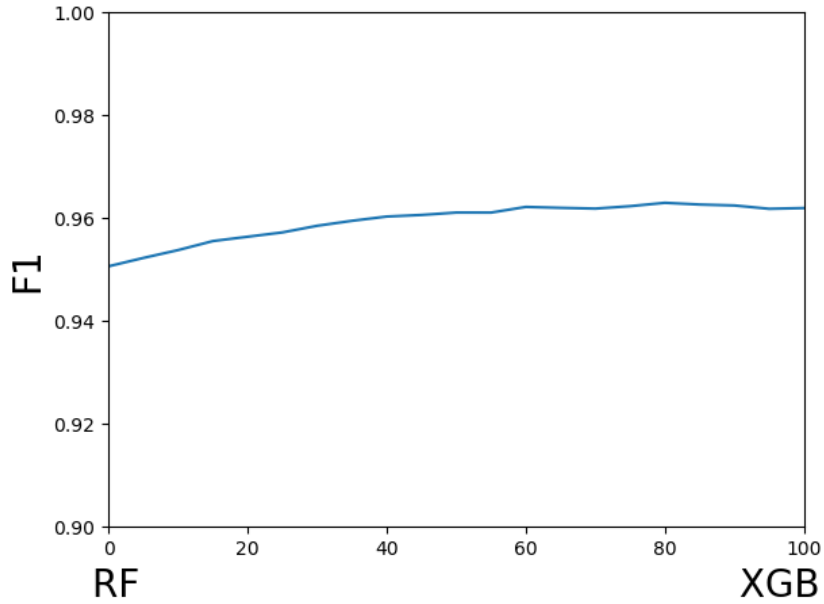
ข้อมูลจาก 4.3 ผลการทดสอบ soft voting สรุปผลเป็นตารางค่าเฉลี่ยจากทั้ง 10 fold ได้ดังตาราง 27 และสามารถสร้างกราฟ รูป 10 กราฟ soft voting ด้วย bytecode 3-gram และ รูป 11 กราฟ soft voting ด้วย bytecode 4-gram ได้ดังนี้

RF	XGB	bytecode 3-gram	bytecode 4-gram
100	0	0.950360396	0.950673082
95	5	0.953666473	0.952296605
90	10	0.953839203	0.953824427
85	15	0.955119701	0.955576685
80	20	0.955446832	0.956418838
75	25	0.955385648	0.957259601
70	30	0.955864975	0.958548533
65	35	0.957326126	0.959513271
60	40	0.958129043	0.960343405
55	45	0.958450344	0.960652497
50	50	0.958930024	0.961117322
45	55	0.959270274	0.96111123
40	60	0.958916635	0.962212909
35	65	0.960040207	0.962034466
30	70	0.960655621	0.961884807
25	75	0.961754145	0.962359873
20	80	0.962225524	0.963010212
15	85	0.962542015	0.962674042
10	90	0.962393988	0.962492651
5	95	0.962704876	0.961845394
0	100	0.962373123	0.961991755

ตาราง 27 สรุปF_1-score soft voting



รูป 10 กราฟ soft voting ด้วย bytecode 3-gram

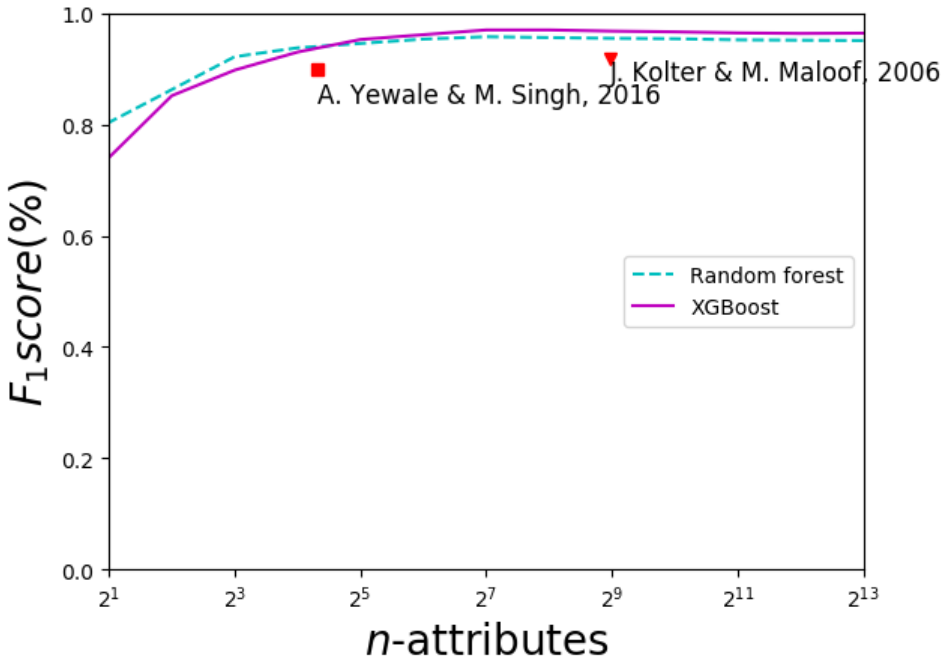


รูป 11 กราฟ soft voting ด้วย bytecode 4-gram

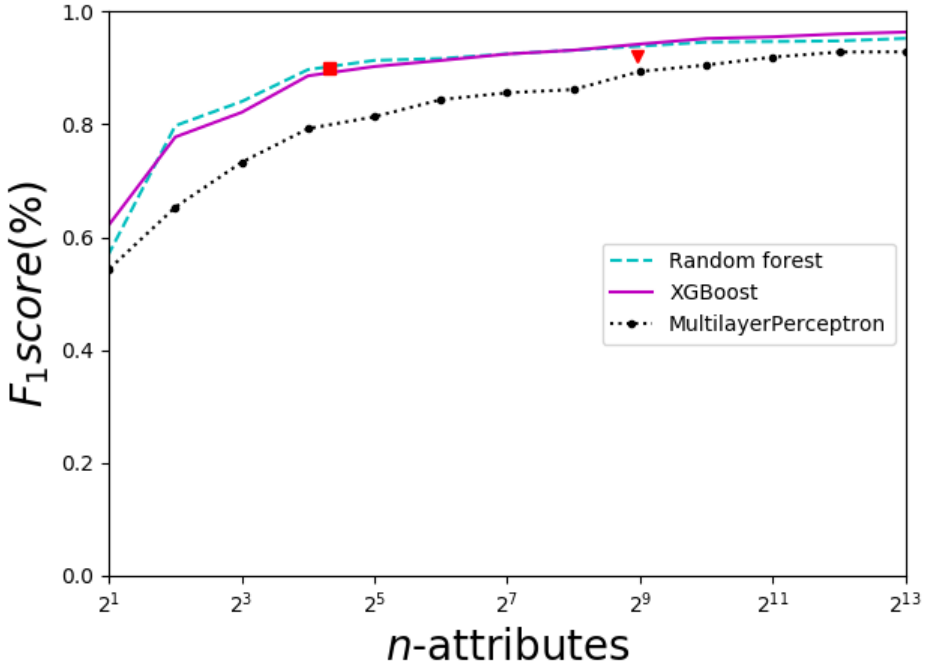
กราฟและตารางข้างต้น แสดงให้เห็นว่า การร่วม vote กันระหว่าง random forest และ extreme gradient boosting ด้วยอัตราส่วนต่าง ๆ สามารถทำให้ได้ค่า F_1-score เพิ่มขึ้นเล็กน้อย แต่อย่างไรก็ตาม แนวโน้มของค่า F_1-score จะสูงกว่าในฝั่งของ extreme gradient boosting นอกจากนี้ การ vote จะต้องสร้าง model จำนวนเพิ่มขึ้น ซึ่งส่งผลต่อทรัพยากรที่ต้องใช้มากขึ้น ทั้งการคำนวณและเวลา

5.3 วิเคราะห์ผลการทดสอบ recursive feature elimination

จาก รูป 12 กราฟ recursive feature elimination with 4 fold cross-validation จะเห็นได้ว่า model ที่สร้างขึ้น มี F_1-score สูงกว่า baseline ทั้งสองmodelที่เลือกมาคือ baseline 1 เป็นงานของ J.Kolter & M. Maloof, ปี ค.ศ.2006 [18] ซึ่งใช้ feature เป็น bytecode 4-gram จำนวน 500 ตัว ใช้ feature selection ด้วย binary information gain และ ใช้ boosted decision tree เป็น classifier และ baseline 2 เป็นงานของ A. Yewale & M. Singh ในปี ค.ศ. 2016 [33] ซึ่งใช้ feature เป็น opcode 1-gram จำนวน 20 ตัว feature selection ด้วย principal component analysis และ ใช้ random forest เป็น classifier และนอกจากนี้ จะสังเกตุได้ว่าจากกราฟเริ่มต้นที่ 8192 attribute extreme gradient boosting จะได้ F_1-score สูงกว่า random forest แต่เมื่อลด attribute ลงถึงจำนวนหนึ่ง extreme gradient boosting จะกลับมีค่า F_1-score ต่ำกว่า random forest และเมื่อพิจารณา รูป 13 กราฟ ลดจำนวน feature ด้วย TF-IDF จะพบว่ามีค่า F_1-score ลดลง และเข้าใกล้ baseline มากขึ้น



รูป 12 กราฟ recursive feature elimination with 4 fold cross-validation



รูป 13 กราฟ ลดจำนวน feature ด้วย TF-IDF

บทที่ 6. สรุปผล

6.1 สิ่งที่ได้จากการวิจัย (contribution)

จาก บทที่ 5. วิเคราะห์ผลการทดลอง แสดงให้เห็นว่า

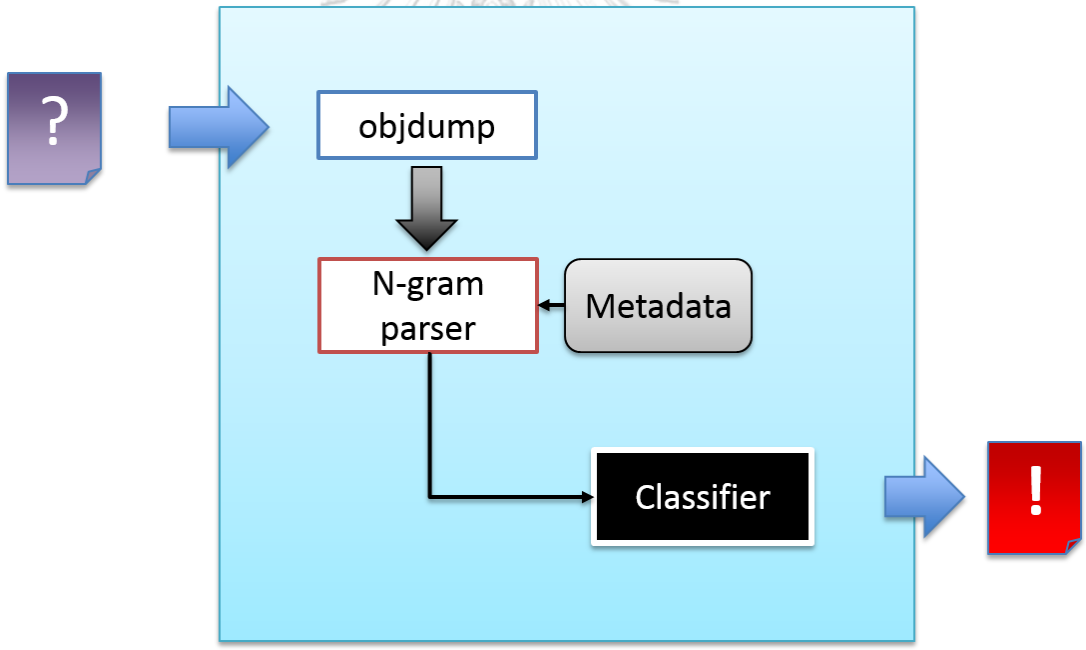
1. การนำ feature หลายชนิดมาใช้ร่วมกัน ไม่ได้ส่งผลดีอย่างมีนัยสำคัญ

2. การร่วมกันโหวตด้วย classifier ในอัตราส่วนต่าง ๆ สามารถเพิ่มความแม่นยำขึ้นได้เล็กน้อย แต่ต้องแลกมาด้วยทรัพยากรในการสร้าง model ขึ้นมาเรียนรู้และทำนายร่วมกัน

3. การลดจำนวน feature ด้วย recursive feature elimination with cross validation สามารถให้ผลลัพธ์ที่ดีกว่าการลดจำนวน feature ด้วย TF-IDF โดยตรง

6.2 การนำไปใช้

จากผลการวิจัย สามารถสร้างเป็นโปรแกรมได้ดังต่อไปนี้



รูป 14 แผนภาพระบบ

จาก รูป 14 แสดงแผนภาพอธิบายโปรแกรม ซึ่งจะทำงานโดยนำเข้าไฟล์เข้าไป ผ่านการสกัดด้วย objdump แล้วนำผลลัพธ์ที่ได้ไปทำการหาความถี่ของ n-gram ที่พบด้วย n-gram parser ซึ่งจะระบุประเภทของ n-gram และรายชื่อ n-gram ที่สนใจไว้ในไฟล์ metadata ผลลัพธ์ที่ได้จาก n-gram parser จะเป็น array ซึ่งส่งต่อไปยัง classifier เพื่อทำนายประเภทของไฟล์ และได้ผลลัพธ์สุดท้ายเป็นค่า 0 คือทำนายว่าเป็นไฟล์ benign หรือ 1 คือทำนายว่าเป็นไฟล์ virus

โดยโปรแกรมนี้ สามารถวางไว้ที่เครื่องบริการ(server) แล้วให้ผู้ใช้หลายคนร่วมกันส่งไฟล์ขึ้นไปสอนให้โปรแกรมเรียนรู้ และทำนายไฟล์ร่วมกันได้ หรือวางไว้ที่เครื่องลูกค้า(client) แล้วให้ทำงานด้วยตัวเองเมื่อไฟล์ถูกสั่งให้ทำงานก็ได้ โดยมีข้อจำกัดของโปรแกรมคือ ต้องได้รับการเรียนรู้อยู่เสมอ เพื่อให้ทันต่อแนวโน้มของไฟล์ที่เปลี่ยนไป

6.3 แนวทางการวิจัยในอนาคต

สามารถประยุกต์ใช้ deep learning ร่วมกับ feature ทุกชนิดที่มีอยู่และทรัพยากรในการประมวลผลที่มากเพียงพอ เพื่อเพิ่มความแม่นยำได้ และถ้าหากได้ผลดี ก็สามารถทำเป็นระบบให้ผู้ใช้จำนวนมากร่วมกันส่งไฟล์ให้เรียนรู้และทำนายได้

รายการอ้างอิง

[1] M. Garnaeva, V. Chebyshev, D. Makrushin, R. Unuchek, and A. Ivanov, "Kaspersky Security Bulletin 2014. Overall statistics for 2014," *Secur. List*, p. 31, 2014.

[2] V. Harrison and J. Pagliery, "Nearly 1 million new malware threats released every day," 2015. [Online]. Available: http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/. [Accessed: 01-Sep-2016].

[3] CloudTweaks, "Cloud Infographic – Computer Virus Facts And Stats," 2014. [Online]. Available: http://cloudtweaks.com/2014/04/cloud-infographic-computer-virus-facts-stats/. [Accessed: 01-Sep-2016].

[4] I. H. Witten, E. Frank, and M. A. All, *Data Mining. Practical Machine Learning Tools and Techniques*. 2011.

[5] S. Marsland, *Machine Learning: An Algorithmic Perspective*. 2009.

[6] R. Quinlan, "C4.5 : programs for machine learning," *Morgan Kaufmann Ser. Mach. Learn.*, p. x, 302 , 1993.

[7] C. Chang and C. Lin, "LIBSVM : A Library for Support Vector Machines," *Tist*, 2001. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[8] W. Cohen, "Fast Effective Rule Induction," *Mach. Learn. Proc. Twelfth Int. Conf.*, 1995.

[9] G. H. G. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," *Proc. Elev. Conf. Uncertain. Artif. Intell. Montr. Quebec, Canada*, vol. 1, pp. 338--345, 1995.

[10] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[11] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016, pp. 785–794.

[12] M. A. Hall, "Correlation-based feature subset selection for machine learning,"

1999, p. 178.

[13] L. Yu and H. Liu, “Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution,” *Int. Conf. Mach. Learn.*, pp. 1–8, 2003.

[14] D. A. Grossman and O. Frieder, *Information Retrieval*, vol. 53, no. 9. 2004.

[15] C. Zhang and Y. Ma, *Ensemble Machine Learning Methods and Applications*. 2012.

[16] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, “Data mining methods for detection of new malicious executables,” in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, 2001, pp. 38–49.

[17] Matthew G.Schultz Eleazar Eskin and E. Zadok, “MEF: Malicious Email Filter, A UNIX mail Filter that Detects Malicious Windows Executables,” 2001.

[18] J. Z. Kolter and M. A. Maloof, “Learning to detect malicious executables in the wild,” *J. Mach. Learn. Res.*, vol. 7, no. 11, pp. 2721–2744, Mar. 2006.

[19] M. Siddiqui, M. C. Wang, and J. Lee, “A survey of data mining techniques for malware detection using file features,” in *Proceedings of the 46th Annual Southeast Regional Conference on XX - ACM-SE 46*, 2008, vol. 6, p. 509.

[20] R. K. Shahzad and N. Lavesson, “Detecting scareware by mining variable length instruction sequences,” *2011 Inf. Secur. South Africa - Proc. ISSA 2011 Conf.*, 2011.

[21] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang, “Using Support Vector Machine to Detect Unknown Computer Viruses,” *Int. J. Comput. Intell. Res.*, vol. 2, no. 1, pp. 100–104, 2006.

[22] Tzu-Yen Wang, Chin-Hsiung Wu, and Chu-Cheng Hsieh, “A Virus Prevention Model Based on Static Analysis and Data Mining Methods,” in *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, 2008, no. October, pp. 288–293.

[23] D. Gavrilut, M. Cimpoesu, D. Anton, and L. Ciortuz, “Malware detection using machine learning,” in *2009 International Multiconference on Computer Science and Information Technology*, 2009, pp. 735–741.

[24] R. K. Shazhad, S. I. Haider, and N. Lavesson, “Detection of spyware by mining executable files,” *ARES 2010 - 5th Int. Conf. Availability, Reliab. Secur.*, pp. 295–302, 2010.

[25] R. K. Shahzad, N. Lavesson, and H. Johnson, “Accurate adware detection using opcode sequence extraction,” *Proc. 2011 6th Int. Conf. Availability, Reliab. Secur. ARES 2011*, pp. 189–195, 2011.

[26] R. K. Shahzad and N. Lavesson, “Veto-based malware detection,” *Proc. - 2012 7th Int. Conf. Availability, Reliab. Secur. ARES 2012*, pp. 47–54, 2012.

[27] G. Tahan, L. Rokach, and Y. Shahar, “Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-features,” *J. Mach. Learn. Res.*, vol. 13, pp. 949–979, 2012.

[28] D. Komashinskiy and I. Kotenko, “Using Low-Level Dynamic Attributes for Malware Detection Based on Data Mining Methods,” *Comput. Netw. Secur.*, pp. 254–269, 2012.

[29] R. K. (Blekinge I. of T. Shahzad and N. (Blekinge I. of T. Lavesson, “Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection,” no. August, pp. 98–117, 2012.

[30] G. Shanmugam, R. M. Low, and M. Stamp, “Simple substitution distance and metamorphic detection,” *J. Comput. Virol.*, vol. 9, no. 3, pp. 159–170, 2013.

[31] T. Singh, “Support Vector Machines and Metamorphic Malware Detection,” 2015.

[32] C. Vatamanu, D. Cosovan, D. Gavrilu, and H. Luchian, “A Comparative Study of Malware Detection Techniques Using Machine Learning Methods,” vol. 9, no. 5, pp. 1115–1122, 2015.

[33] A. Yewale and M. Singh, “Malware detection based on opcode frequency,” in *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2016, no. 978, pp. 646–649.

[34] “Welcome to VX Heaven!” [Online]. Available: http://vxheaven.org/. [Accessed: 01-Mar-2016].

ภาคผนวก

ภาคผนวก ก Code recursive feature elimination with cross validation

ดัดแปลงจาก class RFECV ต้นฉบับใน scikit-learn เพื่อให้สามารถคืนค่าจำนวน feature ที่เจาะจงไว้เท่านั้นคือ parameter ชื่อ n_features_to_select

```
class RFECVX(RFE, MetaEstimatorMixin):
  def __init__(self, estimator, step=1, cv=None, scoring=None, verbose=0,
    n_jobs=1, n_features_to_select=1):
    self.estimator = estimator
    self.step = step
    self.cv = cv
    self.scoring = scoring
    self.verbose = verbose
    self.n_jobs = n_jobs
    self.n_features_to_select = n_features_to_select

  def fit(self, X, y):
    """Fit the RFE model and automatically tune the number of selected
       features.
    Parameters
    ----------
    X : {array-like, sparse matrix}, shape = [n_samples, n_features]
        Training vector, where `n_samples` is the number of samples and
        `n_features` is the total number of features.
    y : array-like, shape = [n_samples]
        Target values (integers for classification, real numbers for
        regression).
    """
    X, y = check_X_y(X, y, "csr")

    # Initialization
```

```
cv = check_cv(self.cv, y, is_classifier(self.estimator))
scorer = check_scoring(self.estimator, scoring=self.scoring)
n_features = X.shape[1]
rfe = RFE(estimator=self.estimator,
          n_features_to_select=self.n_features_to_select,
          step=self.step, verbose=self.verbose - 1)


# Determine the number of subsets of features by fitting across
# the train folds and choosing the "features_to_select" parameter
# that gives the least averaged error across all folds.

# Note that joblib raises a non-picklable error for bound methods
# even if n_jobs is set to 1 with the default multiprocessing
# backend.
# This branching is done so that to
# make sure that user code that sets n_jobs to 1
# and provides bound methods as scorers is not broken with the
# addition of n_jobs parameter in version 0.18.

if self.n_jobs == 1:
  parallel, func = list, _rfe_single_fit
else:
  parallel, func, = Parallel(n_jobs=self.n_jobs), delayed(_rfe_single_fit)

scores = parallel(
  func(rfe, self.estimator, X, y, train, test, scorer)
  for train, test in cv.split(X, y))


scores = np.sum(scores, axis=0)
```

```
n_features_to_select = max(
  n_features - (np.argmax(scores) * self.step),
  self.n_features_to_select)
n_features_to_select=self.n_features_to_select

# Re-execute an elimination with best_k over the whole set
rfe = RFE(estimator=self.estimator,
          n_features_to_select=n_features_to_select, step=self.step)

rfe.fit(X, y)

# Set final attributes
self.support_ = rfe.support_
self.n_features_ = rfe.n_features_
self.ranking_ = rfe.ranking_
self.estimator_ = clone(self.estimator)
self.estimator_.fit(self.transform(X), y)

# Fixing a normalization error, n is equal to get_n_splits(X, y) - 1
# here, the scores are normalized by get_n_splits(X, y)
self.grid_scores_ = scores[::-1] / cv.get_n_splits(X, y)
return self
```

ภาคผนวก ข Code โปรแกรมผลลัพธ์จากงานวิจัย

โปรแกรมที่เขียนขึ้นเพื่อใช้งานผลลัพธ์จากงานวิจัยนี้ ต้องการ metadata ที่สอดคล้องกับ model ของ classifier และใช้ pickle ในการ load model classifier โดยในที่นี้ทำงานกับ bytecode 4-gram

ฟังก์ชั่น ParseFileNameToLongBytecode จะเรียกคำสั่ง objdump -s แล้วเขียนลงไฟล์ชื่อ tmpFile จากนั้นอ่านขึ้นมาและนำเอาส่วนที่เป็น bytecode มาต่อกันให้หมด return ออกเป็นสายอักขระของ bytecode

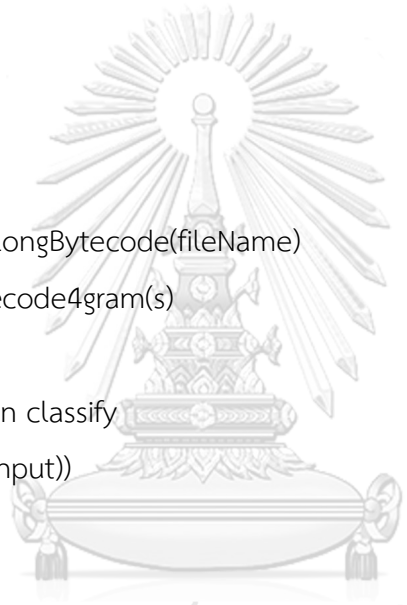
ฟังก์ชั่น parseBytecode4gram นำเอาผลลัพธ์ จาก ParseFileNameToLongBytecode มานับจำนวนใส่ลงใน array ซึ่งมีการเรียงตามลำดับของ bytecode ที่พบตาม metadata ที่กำหนดไว้ ในที่นี้ชื่อ meta8192

ฟังก์ชั่น classifyData จะ load model ของ classifier ขึ้นมาด้วย pickle ถ้าผู้ใช้ต้องการใช้งาน XGBoost ต้อง download library ของ XGBoost เสียก่อน มิเช่นนั้นจะพบข้อผิดพลาดในขั้นตอนนี้ หลังจาก load model ขึ้นมาแล้ว จะนำเอา input ชื่อ dataIn ไปใช้เป็น input สำหรับการทำนายด้วย model ดังกล่าว และ return ผลลัพธ์ออกไป

```
def ParseFileNameToLongBytecode(inFileName):
  commandStr = "objdump -s " + inFileName + " > tmpFile"
  print (commandStr)
  os.system(commandStr)
  f=open("tmpFile","r")
  longLine = ""
  for line1 in f:
    if (line1[0]==' ') :
      line2=line1[line1.index(' ',2):]
      longLine = longLine+line2[0:36].replace(" ","")
  f.close()
  return longLine

def parseBytecode4gram(longLine):
  #read meta
```

```
  f=open("metadata/meta8192","r")
  metaList = []
  countLine = 0
  for line1 in f:
    if (countLine < (8192*2)) :
      if(countLine%2==0):
        metaList += [line1.rstrip()]
    else:
      break
    countLine += 1
  f.close()

  idx = 0
  dataArr = [0]*8192
  lenLongLine = len(longLine)

  while idx < lenLongLine-6:
    theWord = longLine[idx:idx+8]
    if(theWord in metaList):
      metaIdx = metaList.index(theWord)
      dataArr[metaIdx] += 1
    idx += 2

  return dataArr

def classifyData (dataIn):
  model_rf = pickle.load(open("metadata/rf_d3_8192.dat", 'rb'))
  model_xg = pickle.load(open("metadata/xg_d3_8192.dat", 'rb'))

  re_dataIn = [dataIn]
```

```
  out_rf = model_rf.predict(re_dataIn)
  out_xg = model_xg.predict(re_dataIn)

  return [out_rf[0],out_xg[0]]

if __name__ == '__main__':
  fileName = sys.argv[1]
  if(len(fileName)==0):
    quit()

  #1.objdump
  #2.parse to array
  s = ParseFileNameToLongBytecode(fileName)
  dataInput = parseBytecode4gram(s)

  #3.load model and run classify
  print(classifyd01(dataInput))

  quit()
```



ประวัติผู้เขียนวิทยานิพนธ์

นายประสิทธิ์ อุษาฟ้าพนัส เกิดเมื่อวันที่ 7 มีนาคม พ.ศ.2530 ภูมิลำเนาอยู่ในจังหวัดชลบุรี สำเร็จการศึกษาวิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2552