ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

PREDICTING SOFTWARE VULNERABILITIES USING TOPIC MODELING WITH ISSUES

M.Sc. THESIS

Fatma Gül BULUT

Department of Communication Systems

Satellite Communication and Remote Sensing Programme

Thesis Advisor: Dr. Ayşe TOSUN

JUNE 2019

ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

PREDICTING SOFTWARE VULNERABILITIES USING TOPIC MODELING WITH ISSUES

M.Sc. THESIS

Fatma Gül BULUT
(705151016)

Department of Communication Systems

Satellite Communication and Remote Sensing Programme

Thesis Advisor: Dr. Ayşe TOSUN

JUNE 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

KONU MODELLEME YÖNTEMİ İLE YAZILIM GÜVENLİK AÇIKLARINI TAHMİN ETME

YÜKSEK LİSANS TEZİ

Fatma Gül BULUT
(705151016)

İletişim Sistemleri Anabilim Dalı

Uydu Haberleşmesi ve Uzaktan Algılama Programı

Tez Danışmanı: Dr. Ayşe TOSUN

HAZİRAN 2019

Fatma Gül Bulut, a **M.Sc.** student of ITU Informatics Institute student ID **705151016**, successfully defended the **thesis** entitled "**PREDICTING SOFTWARE VULNERABILITIES USING TOPIC MODELING WITH ISSUES**", which she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Dr. Ayşe Tosun** İstanbul Technical University	
Co-advisor :	**Dr. Haluk Altunel** Hacettepe University	....
Jury Members :	**Dr. Şerif Bahtiyar** İstanbul Technical University	....
	Dr. M. Tahir Sandıkkaya İstanbul Technical University	....
	Assoc. Prof. Dr. Cemal Yılmaz Sabancı University	....

Date of Submission : 03 May 2019
Date of Defense : 11 June 2019

To my family,

FOREWORD

I would like to thank my thesis supervisor Dr. Ayşe TOSUN for her knowledge and encouragement during the completion of this thesis. It would have been impossible to produce without her guidance and patience.

I would like to thank my co-advisor Dr.Haluk ALTUNEL for sharing his valuable ideas and guidance whenever I needed.

I would like to give my special thanks to my family, especially my lovely husband Ömer BULUT, for his gentle support and love when my mood was up and down throughout my research.

May 2019 Fatma Gül BULUT

TABLE OF CONTENTS

ABBREVIATIONS

ANN	: Artificial Neural Network
PCI-DSS	: Payment Card Industry Data Security Standard
NLP	: Natural Language Processing
HIPAA	: Health Insurance Portability and Accountability Act
OWASP	: Open Web Application Security Project
FISMA	: Federal Information Security Management Act
CVE	: Common Vulnerabilities and Exposures
NVD	: The National Vulnerability Database
TF-IDF	: The Term Frequency - Inverse Document Frequency
LDA	: Latent Dirichlet Allocation
SVR	: Support vector regression
CART	: Classification and Regression Trees
SVM	: Support Vector Machines
OS	: Open Source
NB	: Naïve Bayes
FLDA	: Fisher's Linear Discriminant Algorithm Act
RT	: Random Tree
AE	: Absolute Error
RE	: Relative Error
MMRE	: Mean magnitude of relative error
MdMRE	: Median magnitude of relative error
Pred(k)	: Prediction Indicator
CC	: Correlation Coefficient
AUC	: Area Under The Curve

LIST OF TABLES

LIST OF FIGURES

Page

PREDICTING SOFTWARE VULNERABILITIES USING TOPIC MODELING WITH ISSUES

SUMMARY

Developing secure software is one of the most important topics of the 21st century and the future. Today, this important topic has become one of the main themes in software development life cycle. Omissions in software development life cycle processes can cause severe software vulnerabilities.

During software development, developers work with the goal of both writing secure software and creating a flawless job under the time pressure. Developing secure software is part of the software lifecycle but requires training and experience. Developers tend to put secure software development to the second place under challenging situations within limited time targets. It is important to be able to automatically predict software vulnerabilities under such stressful situations for saving time and effort of security analysts and testers.

A number of studies have been carried out both in academia and industry in the context of predicting software vulnerabilities. Predictor models are proposed using metrics, statistical methods and machine learning algorithms to highlight residual vulnerabilities in software systems. The common purpose of all of the studies is to prevent the vulnerabilities that may occur by predicting the software vulnerabilities at an early stage.

This thesis aims to predict software vulnerabilities by using machine learning algorithms, text mining techniques and natural language processing methods.

Open datasets are mostly used in a built software vulnerability prediction model. Records of software vulnerabilities are highly confidential records for institutions and organizations. For this reason, vulnerability data recorded in open source systems are mostly used in the studies. In this thesis, security scansconducted in an information technology company is collected and masked in order to build vulnerability prediction models.

Many machine learning algorithms have been used to predict software vulnerabilities. In the literature, this prediction problem is considered as a classification problem. Classification is located under the supervised learning heading from machine learning algorithms. It is ideal to use classification algorithms when the purpose is to distinguish whether a module is safe or unsafe. In this thesis, the problem is considered as a regression problem. Regression is also under the supervised learning heading from machine learning algorithms. Our goal is to reveal the formula that predicts the number of vulnerabilities on the basis of priority class and to contribute to the early detection of software vulnerabilities.

In the thesis study, decision trees (support and regression), support vector regression and artificial neural network algorithms were selected from machine learning algorithms for a regression problem.

Many different metrics have been used as input in the models developed to predict software vulnerabilities. These entries are mostly software code metrics. Software code metrics, as well as error records metrics, developer based statistical measurements, are experienced as inputs in models. In this thesis, post-release issue records are used as model input.

Software vulnerability records used as model output in the study were obtained from CxSAST reports, a static code analysis tool. In the report, software vulnerabilities were divided into 3 groups as low, medium and high according to their severity. These three significance levels were used separately as outputs, and their totals were also used as outputs.

Within the scope of the thesis study, 2 different prediction models were established. Models A and model B are given to these models. In this thesis, 10-fold cross-validation and leave-one-out cross-validation methods were used and the results were compared.

When the results are compared, there is a low relationship between issue records and software vulnerabilities. The decision tree model has been found to have the most accurate estimation results compared to other algorithms. Support vector regression and artificial neural network decision trees follow the changing algorithm parameters.

Extending the dataset to improve the results is thought to be a step. The dataset used in the thesis study includes a period of 6 months and data of an institution.

KONU MODELLEME YÖNTEMİ İLE YAZILIM GÜVENLİK AÇIKLARINI TAHMİN ETME

ÖZET

Güvenli yazılım geliştirmek, 21. yüzyılın ve geleceğin önemli konularından bir tanesidir. Günümüzde bu önemli konu yazılım geliştirme yaşam döngüsü süreçlerinin ana temalarından biri haline gelmiştir. Güvenli yazılım geliştirme yaşam döngüsü adımlarında yapılan ihmaller yazılım güvenlik açıklarına neden olabilmektedir.

Yazılım geliştirme aşamasında, geliştiriciler hem güvenli yazılım yazmak hem de hatasız bir iş çıkarmak hedefi ile çalışmaktadır. Bu hedef yanında zaman baskısı unutulmamalıdır. Güvenli yazılım geliştirmek yazılım yaşam döngüsünün bir parçasıdır ancak eğitim ve deneyim gerektirmektedir. Geliştiriciler kısıtlı zaman hedefleri içerisinde güvenli yazılım geliştirmeyi bazı zorlu durumlarda ikinci plana atabilmektedir. Bu zorlu durum karşısında yazılım güvenlik açıklarını ortaya çıkmadan tahmin edebilmek önemli hale gelmiştir.

Güvenlik açıklarının tahmin edilmesi üzerine akademi ve endüstri tarafında bir çok çalışma gerçekleştirilmektedir. Yazılım güvenlik açıkları ile ilişki kurulabilecek var olan metrikler makine öğrenmesi algoritmaları ve istatistiksel yöntemler ile modeller kurularak tahminler yapılmaktadır. Var olan metrikler dışında ampirik yaklaşımlar ile yeni metrikler ortaya çıkarılmaktadır. Yapılan tüm çalışmalarının ortak amacı güvenlik açıklarını erken evrede tahmin ederek oluşabilecek zafiyetlerin önüne geçebilmektir.

Bu tez çalışması ile makine öğrenmesi algoritmaları, metin madenciliği teknikleri ve doğal dil işleme yöntemleri kullanılarak yazılım güvenlik açıklarını tahmin etmek amaçlanmıştır. Makine öğrenmesi algoritmaları sayısal öğrenme ve model tanıma çalışmalarından geliştirilmiş bilgisayar biliminin ve yapay zekanın bir alt dalıdır. Günümüze kadar gerçekleştirilmiş güvenlik açıkları tahminleme çalışmalarında öncelikli tercih edilen yöntemlerdendir. Metin madenciliği metin verisi üzerinden yapısallaştırılmış veri elde etmeye olanak sağlar. Doğal dil işleme, doğal dildeki metinlerden ve/veya seslerden anlamlı ve istenen bilgilerinin bilgisayarın anlayabileceği anlamların çıkarılmasını sağlayan yöntemler bütünüdür, yapay zekanın ve dilbimin bir alt dalıdır.

Yazılım güvenlik açıklıkları ile ilgili yapılan çalışmalarda çoğunlukla açık veri setleri kullanılmıştır. Güvenlik açıklarına dair kayıtlar kurum ve kuruluşlar için gizliliği yüksek seviyeli kayıtlardır. Bu sebeple yapılan çalışmalarda çoğunlukla açık kaynak sistemler üzerine gerçekleştirilmektedir. Bu tez çalışmasında iki farklı veri seti kullanılmaktadır. Bilgi teknoloji şirketine ait güvenlik tarama verileri ve açık kaynak olan WireShark projesinin verileri çıktı olarak kullanılmaktadır. Bilgi teknoloji şirketine ait veriler maskelenerek paylaşılmaktadır. Bu çalışma öncesi tez çalışmasında yer alan kişiler ile kurum arasında gizlilik anlaşması oluşturulmuştur.

Yazılım güvenlik açıklarını tahmin etmede bir çok makine öğrenmesi algoritması kullanılmıştır. Literatürde bu tahmin problemi daha çok sınıflandırma problemi olarak ele alınmıştır. Sınıflandırma makine öğrenmesi algoritmalarından denetimli öğrenme başlığı altında yer alır. Amaç bir modülün güvenli ya da güvensiz olduğunu ayırt etmek olduğunda, sınıflandırma algoritmalarının kullanılması idealdir. Bu tez çalışmasında elde edilen veri problemin tipine karar vermede etkili olmuştur. Şirkete ait veriler için problem regresyon problemi olarak ele alınmıştır. Wireshark projesi verileri ile sınıflandırma algoritmaları kullanılmıştır. Regresyon da makine öğrenmesi algoritmalarından denetimli öğrenme başlığı altında yer alır. Regresyon probleminde amacımız güvenlik açığı sayısını, öncelik sınıfı bazında doğru tahmin eden formülü ortaya çıkarmak ve yazılım güvenlik açıklarının erken evrede tespitinin sağlanmasına katkıda bulunmaktır. Sınıflandırma probleminde ise açılan hata kayıtlarının yazılım güvenlik açığı ile ilişkisinin olup olmayacağı tahmin edilecektir.

Tez çalışmasında makine öğrenme algoritmalarından karar ağaçları (classification and regression tree), destek vektör regresyonu (support vector regression) ve yapay sinir ağları (artificial neural network) algoritmaları regresyon problemi için seçilmiştir. Karar ağaçları sınıflandırma ve regresyon problemlerinde kullanılabilirler. Çok sayıda kayıt içeren bir veri kümesini, karar kuralları uygulayarak küçük kümelere bölmek için kullanılan bir algoritmadır. Destek vektör regresyonu, destek vektör makinalarından geliştirilmiştir. Önceki çalışmalarda, verilerin doğrusal olarak ayrılamadığı durumlarda destek vektörleri algoritmalarının problem çözmedeki performansı ve yeteneğinin daha iyi olduğu paylaşılmaktadır. Yapay sinir ağları, biyolojik nöronlardan esinlenilen bir hesaplama modelidir. Literatürde pek çok çalışmada tahmin algoritması olarak seçilmiştir. Sınıflandırma problemi için ise Naive Bayes, RandomTree ve Fisher's Linear Discriminant algortimaları kullanılmıştır.

Yazılım güvenlik açıklarını tahmin etmede geliştirilen modellerde girdi olarak bir çok farklı metrik kullanılmıştır. Bu girdiler çoğunlukla yazılım kod metrikleri olmuştur. Yazılım kod metriklerinin yanı sıra hata kayıtları metrikleri, geliştirici bazlı istatistiksel ölçümler modellerde girdi olarak deneyimlenmiştir. Bu tez çalışmasında regresyon problemi model girdisi olarak sürüm sonrası olay kayıtları kullanılmaktadır. Tez çalışmasında kullanılan olay kayıtları aylık olarak raporlanmaktadır, çalışma 6 aylık veri üzerinde gerçekleştirilmiştir. Elde edilen 6 aylık veriden olay kayıtlarının açıklamalarına metin madenciliği ve doğal dil işleme adımları işletilmiştir. Doğal dil işleme alt başlıklarından biri olan konu modelleme olay kayıtlarının açıklama verileri üzerine uygulanmış, 5 ana konu 3516 adet olay kaydı veri kümesi kullanılarak belirlenmiştir. Konu modelleme yöntemi olarak Gizli Dirichlet Tahsisi (Latent Dirichlet Allocation) kullanılmıştır. Aynı doğal dil işleme adımları WireShark projesi verileri üzerinde de işletilmiştir. Wireshark projesinde veri seti 2017-2018 yıllarına ait verilerden oluşmaktadır. Model girdisi için 2017-2018 yıllarında açılan hata kayıtları kullanılmıştır.

Literatürde incelenen çalışmalarda yazılım metrikleri üzerinde yapılan toplama ve kümeleme işlemlerinin model tahminlerini etkilediği paylaşılmaktadır. Bu sav üzerine çıkarılan 5 konu modeline toplama ve matematiksel işlemler geçrekleştirilerek yeni toplama metrikler oluşturulmuştur. Bu toplama metrikler minimum değer, maksimum değer, hoveer indeksi ve medyandır.

Çalışmada model çıktısı olarak kullanılan yazılım güvenlik açığı kayıtları bir statik kod analizi aracı olan CxSAST raporlarından elde edilmiştir. Elde edilen raporda yazılım güvenlik açıkları önem derecesine göre düşük, orta ve yüksek olmak üzere 3 gruba ayrılmıştır. Bu üç önem derecesi ayrı ayrı çıktı olarak kullanılmış, toplamları da ayrıca çıktı olarak kullanılmıştır.

Literatürde yapılan çalışmalarda çoğunlukla yazılım sınıfı boyutunda tahminleme gerçekleştirilmektedir. Bu tez çalışmasında elde edilen veri seti değerlendirildiğinde yazılım ürünü boyutunda tahminleme modeli kurulması kararlaştırılmıştır. Kurum verisi ile kurulan modeller yazılım ürününün yazılım güvenlik açığı sayısını tahminlemektedir.

Tez çalışması kapsamında kurum verileri ile 2 farklı tahmin modeli kurulmuştur. Bu modellere model A ve model B isimleri verilmiştir. Model A için yazılım ürünü bazında açılan olay kayıtları açıklamaları birleştirilerek tekrar konu modellemesi algoritmasından geçirilmiştir. 3516 adet olay kaydı açıklaması kullanılarak oluşturulan modelden yazılım ürünü bazlı konu modeli skorları elde edilmiştir. Model A için ayrıca yazılım ürünü bazında açılan olay kaydı sayıları da girdi olarak kullanılmıştır. Bu olay kaydı sayıları kendi içlerinde 3 gruba ayrılmış olarak raporlanmaktadır, model A'da da bu sayırlar ayrı ayrı metrik olarak değerlendirilmiştir. Bu yazılım ürünü bazındaki 5 konu modeli skoru model A'nın girdilerini oluşturmaktadır. Model B için yazılım metrikleri üzerinde kullanılan toplama ve kümeleme işlemleri olay kaydı bazında elde edilen konu modeli skorlarına uygulanmıştır. Olay kayıtları yazılım ürünü bazında filtrelenmiş ve bahsi geçen 4 toplama metriği her konu modeli için hesaplanmıştır. Hesaplanan toplama metrikler Model B'nin girdilerine eklenmiştir. Her iki modelde de yazılım ürünü önem derecesi girdi olarak kullanılmıştır.

Bir makine öğrenmesi modelinin doğruluğunun test edilmesi için veri kümesinin ne şekilde ayrılacağına çapraz doğrulama yöntemi ile karar verilmektedir. Bu tez çalışmasında 10 katmanlı çapraz doğrulama ve tek çıkışlı çapraz doğrulama yöntemleri kullanılmış ve sonuçları karşılaştırılmıştır.

Elde edilen sonuçlar karşılaştırıldığında olay kayıtları ve yazılım güvenlik açıkları arasında düşük de olsa bir ilişki bulunmaktadır. Karar ağaçları modelinin diğer algoritmalara nazaran en doğru tahminleme sonucunu elde ettiği görülmüştür. Değişen algoritma parametlererine göre destek vektör regresyonu ve yapay sinir ağları karar ağaçları algoritmasının ardından gelmektedir.

Sonuçların iyileştirilmesi için veri setinin genişletilmesi bir adım olabileceği düşünülmektedir. Tez çalışmasında kullanılan veri seti 6 aylık bir dönemi içermekte ve bir kuruma ait verilerdir.

1. INTRODUCTION

Software security is one of the sub-branches of information security. Software security is the case that the software is resilience against possible harm caused by others. McGraw explains software security as an idea of engineering software[1]. The need to address software security as a software requirement at every step of the software life cycle was written by McGraw [1]. According to the guide prepared by the Information and Information Security Research Center, software architecture and design can prevent the attack even if it can continue towards the working process and is able to recognize the situations of abuse [2].

Developing secure software is one of the important topic and this important topics has become one of the critical main themes of software development life cycle processes [40]. Security concern must inform every phase of software development life cycle, from requirements engineering to design, implementation testing and deployment [42]. The security omissions in software development life cycle steps can cause software vulnerabilities [41]. Time and budget pressures on software developers can cause omission in secure software development steps. Software vulnerabilities can occur as a result of these omissions.

Quality is an important issue for large-scale companies, small businesses, and all developers interested in software. Security is one of the features that affect software quality. For this reason, security is an important topic [43]. If a software is described as high quality, it is expected that the maintenance costs of the software will be low. [44]. At this point, considering the future budget and time costs of the security vulnerabilities, the maintenance costs of the secure software are expected to be low. This results in a relationship between security and maintenance costs. Different methods are used to detect, enhance and monitor software security. In penetration testing method [36], test scenarios are applied on source code and vulnerable-prone result of the software is revealed. As a result of these tests, findings are revealed. Another method is the use of static code analysis tools [37]. The vulnerable-prone of the software can be revealed with static code analysis tools [37]. In addition to these methods, machine learning algorithms are also studied on security. Prediction models based on machine learning are built with records of past software vulnerabilities of software system with software metrics, developer-based metrics etc. A number of studies are carried out on the academy and industry side on the prediction of software vulnerabilities [45]. Predictions are built by using software metrics, machine learning algorithms and statistical methods and models that can be associated with software vulnerabilities. The common purpose of all of the studies is to prevent the vulnerabilities that may occur by predicting the software vulnerabilities at an early stage.

With this thesis, we have been informed about the software vulnerabilities ecosystem and the studies. The importance of early detection of software vulnerabilities was observed and prediction models were studied.

1.1. Purpose of Thesis

The purpose of this thesis is to built a model and prediction of software vulnerabilities using topic models derived from textual descriptions of issue records and machine learning techniques.

Today, software security is one of the important facts of the software life cycle from the design and development steps and affects the process. However, time and cost pressures make it difficult to develop secure software. This is one of the reasons why software vulnerabilities have occurred.

Corporate companies are assisted by static code analysis tools for detection software vulnerabilities. Apart from these tools, they perform periodic penetration tests. In addition to these detection methods, software developers are provided with secure software development training.

In this thesis, a 6-month software vulnerability and issue records data belonging to a corporate company was used. Software security checks in this company are followed by Checkmarx CxSAST tool. Software vulnerabilities findings have been obtained from the Checkmarx CxSAST scanning results. The issue records are the bugs opened by the help-desk staff to the software products in the production. The issue records are recorded on the JIRA and the data is obtained from the monthly report. Empirical studies have been carried out on the correlation between the results of the software security scanning and the issue record metrics. The results obtained with the corporate dataset were compared with the open source project data. Wireshark dataset was used for this comparison.

2. LITERATURE REVIEW

2.1 Purpose

Software vulnerabilities are one of the major research topics in the software ecosystem. In this section, previous research and studies on these topics will be examined. In this sections will demonstrate the fundamental aspects of vulnerabilities prediction such as software metrics, text mining and prediction models as well as the algorithms used in the literature.

2.2 Software Vulnerabilities

Software vulnerabilities are one of the significant subjects in the field of computer security [45]. Today, software security is one of the important facts of the software life cycle from the design to testing, and it affects the process. However, time and cost pressures make it difficult to develop secure software. This is one of the reasons why software vulnerabilities have occurred [3]. Ivan Krsul defines software vulnerability as “an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy” [46]. Software vulnerability is defined as a mistake in the technical specifications, development, or configuration of a software that, if it occurs, [implicit or explicit] violates the security policy of the software in the Ozment study [3]. Ozment started from Ivan Kursul's definition, but the term *mistake* was used instead of the term *error* and cites the IEEE Standard Glossary of Software Engineering Terminology (IEEE Standards 1990) to support this usage for definition of software vulnerability. Based on these definitions, a software vulnerability is defined as a ***software-related security mistake*** in this thesis.

Software vulnerabilities can be grouped into five categories according to Apple documentation [47]. These are buffer overflows, unvalidated input, race conditions, access-control problems, and authorization[47]. A buffer overflow occurs when an application attempts to write data past the end (or, occasionally, past the beginning) of a buffer [4]. Buffer overflows can cause applications to crash, can compromise data, and can contribute an attack vector for further privilege escalation to compromise the system on which the application is running. An unvalidated input attack occurs due to unsafe data. All data sources must be checked to prevent this attack. When working with shared data, files, databases etc. there are a number of simply made mistakes that can compromise security. This type of error causes race conditions attacks. Many security vulnerabilities are created by the incorrect use of access controls, or by the omission to use them at all. These uses also cause access control problems.

2.2.1 Software vulnerability identification methods

There are many techniques like penetration testing[36], software reviews[35], static analysis[37] and runtime anomaly detection [38] for identification of software vulnerabilities. However, applying these techniques manually is expensive in terms of the time and budget spent on specialized resources. Some of the existing techniques are available in the form of automated tools, but techniques like software inspections and reviews are intrinsically manual processes that depend on human experts to perform the analysis of the code [4]. In the Software Company, developers use penetration testing and static code analysis tools for detection software vulnerabilities. Penetration testing evaluates the security of a system by simulating attacks by malicious users and assessing whether the attacks are successful [5]. According to Jovanović and Irena's study[39], penetration tests are divided into three groups:

- Black-Box Penetration Testing: The black box penetration test is a type of attack without any knowledge of the target system to be attacked. Penetration tester or anyone who tries to reach the target system from outside without any knowledge is allowed to perceive the extent of the damage.
- Gray-Box Penetration Testing: The gray box penetration test provides an analysis of the damage that an unauthorized user of the internal network can give to the target systems. Data-stealing, authorization upgrade, and network weaknesses are monitored against network packet loggers. It is the most important penetration test type.
- White-Box Penetration Testing: The white box penetration test is a type of penetration test which is made available to all systems in the network. One of the employees is the attack simulation that attempts to break into and out of the network from outside or inside.

The static code analysis tool in OWASP is described as designed to analyze source code and/or compiled versions of code to help find software vulnerabilities[6]. Some tools are starting to move into the IDE. For the types of problems that can be detected during the software development phase itself, this is a powerful phase within the development life cycle to employ such tools, as it provides immediate feedback to the developer on issues they might be introducing into the code during code development itself. This immediate feedback is very useful, especially when compared to finding vulnerabilities much later in the development cycle [6]. There are many free or commercial type static code analysis tools in use. SonarQube, Flawfinder, Bandit, Brakeman, FindSecBugs, Google CodeSearchDiggity are examples of those that are free. SonarQube is used to measure, report and improve code quality. It works with the continuous inspection philosophy. CxSAST, CodeSonar, Fortify are examples of commercial type static code analysis tools. CxSAST scans an uncomplied code and does not require a completed build [7]. It even works from the developer's IDE. This allows organizations to use CxSAST earlier in the software development life cycle. Regulatory standards as Payment Card Industry Data Security Standard (PCI-DSS), Health Insurance Portability and Accountability Act (HIPAA), Federal Information Security Management Act (FISMA) require organizations to test for common vulnerabilities like those found in the Open Web Application Security Project (OWASP) Top 10 and the SANS top 25. CxSAST finds these all vulnerabilities. In this study, CxSAST report parameters are used for output in the prediction model.

Until 1999, identified software vulnerabilities were kept in every company's own database. In 1999, MITRE established the Common Vulnerabilities and Exposures (CVE). MITRE is a non-profit organization founded in 1958 and collaborating with US government agencies on national critical issues [8]. CVE is a free reference dictionary for public security vulnerabilities. Confirmation that a software flaw has a real security vulnerability that will qualify for a CVE number is subject to a series of review and approval processes. Since 1999, CVE has been recognized as an internationally recognized international standard in academia, government institutions and the business world [9].

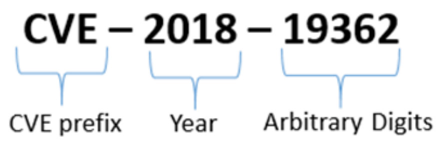


Figure 2.1 : CVE example.

The National Vulnerability Database (NVD) is a database that has been created since 1999 under the responsibility of the Cyber Security Unit within The National Institute of Standards and Technology (NIST) [10] [11]. The NIST is a state institution that has set standards in the field of technology since 1901 and operates under the US Department of Commerce. Software vulnerabilities data is automated in NVD published in management-appropriate formats. CVE database content is available on the NVD web page.

2.2.2 Software vulnerability prediction models

The number of software vulnerabilities reported has been growing every year. According to the NVD [11] in 2018, 16.515 vulnerabilities have been discovered – more than twice as many as were reported in 2016. These vulnerabilities, if exploited, can cause damages to educational institutions, corporations, government systems, software vendors and customers.

Software vulnerability prediction models evaluate security risks and predict future software vulnerabilities [48]. There are many studies on software vulnerability prediction in the literature. In this thesis we present a machine learning based approach to predict vulnerabilities. In the literature, these machine learning based approaches use software code metrics[12], text mining methods [13], data on error records[14], and developer-based metrics [14]. This section summarizes the studies according to these input features.

Meneely et al. [12] have signed one of the first studies to predict software vulnerabilities with software metrics. Meneely et al. [12] proposed a software metrics based prediction model, using the metrics related to code complexity, code churn, and developer’s activity. They form a hypothesis based on code complexity, code complexity and developer base measurements. They did experiments on Mozilla Firefox and the Red Hat Enterprise Linux kernel. They collected 28 metrics related to these three software metrics groups. They used logistic regression machine learning algorithm. With this experiment, they achieved a 75% recall score.

Meneely et al. continued their studies with empirical approaches. Meneely et al. [14] evaluated the correlation between pre-release bugs, reviewer experience and post-release vulnerabilities on the Chromium project. They got the result that, while an empirical connection between bugs and vulnerabilities exist, the connection is considerably weak. This empirical work is one of the inspiration for the selection of the thesis topic. The results of this study were replicated in the scope of the thesis studies and the results were obtained according to the comments of the authors.

Scandariato and Walden [13] recommend a prediction model using text mining on the source code. Software vulnerability prediction model using text mining, use machine learning algorithm on these tokens and its frequency. They conducted an experiment with Android applications, Java source code is transformed into tokens. They reached 80%–85% precision for several android application.

Zhang et al. [15] provide a method to predict vulnerabilities by combining both software metrics and text mining methods. It is a two stage method. First stage six basic classifiers produce outputs, and through the second stage consolidate certain outputs via a composer. In this work, they use three classifiers for each method, and a composer is used to consolidate the outputs of these classifiers. Random forest is used as the composer. The composer is trained with confidence values produced by classifiers and the corresponding status of each component such as vulnerable or not. The input of the composer is the confidence value generated by each classifier. They got an F1 score of 0.75 experimented with web applications Moodle, Phpmyadmin, Drupal.

Li et al. [16] present a new method for predicting vulnerabilities. They have created a framework using deep learning. They called this framework SySeVR. With this framework, they detected 15 vulnerabilities that were not reported to NVD.

Chowdhury and Zulkernine [17] evaluated code complexity, coupling, and compliance measures to predict software vulnerabilities. The researchers examined the mozilla firefox project for 4 years. they used the empirical approach proposed in this project in their research. The researchers examined the 52 release of the mozilla firefox project. Their approach is based on decision trees and achieves a mean accuracy of 72.85%, mean recall of 74.22%, mean fall-out of 28.51%, and mean F1 score of 73.00%.

Hovsepyan et al. [18] recommend a prediction model using text mining on the source code. They conducted experiments on three web applications by transforming the Java source code into tokens and their frequencies of occurences. They reached 80%–85% precision using support vector machine (SVM) algorithm.

Zimmermann et al. [19] found a weak correlation between vulnerabilities and different metrics, including code churn, code complexity, dependencies, and organizational measures. In the context of Windows Vista, they built two different predictors. The first was based on conventional metrics (i.e., code churn measures, code complexity metrics, dependency measures, code coverage measures, and organizational measures) and resulted in a median precision of 66.7% and median recall of 20%. The second prediction model was based on dependencies between binaries and has resulted in slightly lower precision (60%), but higher recall (40%).

Aversano et al. [20] investigated the source code of the changes as text in order to build a predictor to determine whether the introduced changes are buggy. The authors determined that the use of the K nearest neighbors technique results in a significant trade-off in terms of precision and recall. The approach was confirmed using two open source Java applications, yielding precision and recall values of 59%-69% and 59%-23% respectively.

Gegick et al. [21] designed an approach that uses text-mining techniques to train a model to identify which bug reports are security related. The approach was applied to a large Cisco software system and identified 78% of the security-related bug fixes.

Models in literature review have studied the relationship between software metrics, text mining methods(code-base) and vulnerabilities and also bugs and vulnerabilities. This thesis take a different approach to those predictions with corporate organization dataset.

3. PROBLEM STATEMENT

There have been several studies in the area of software vulnerabilities prediction for open-source systems and databases. In those works, vulnerabilities collected from the public database were used. This thesis study include corporate data. Unfortunately, security-related records are kept confidential in organizations. A non-disclosure agreement prior to this thesis work was signed .

At the software development stage, developers work with the pressure to write both secure software and perform flawless work and also time pressure should be noted. Developing secure software is part of the software life cycle but requires training and experience. With this thesis, we want to set a criterion where software developers and companies can follow vulnerabilities. We aim to facilitate the work of developers and to contribute to the measurement of software quality.

In this thesis, it is aimed to predict future software vulnerabilities from textual descriptions of issue/bug records. For this purpose, two different dataset were examined. The first dataset belongs to a corporate company. The company data includes the scan data of the static code analysis tool instead of the software vulnerability. We set out to predict these screening data at an earlier stage by taking the empirical approach of Camilo *et al.* [14] and using issue records. The issue records are the bugs opened by the help-desk or tester staff to the software products in the production. The other dataset belongs to the Wireshark project. In this dataset, software vulnerabilities and bug reports are published periodically. With this dataset will be used to model the predictions that bug records may cause software vulnerabilities in the future.

In summary, we have combined a list of research questions for vulnerabilitiy prediction studies with specific problems of organizations.

i. Can we propose a vulnerability prediction model using issue records?

ii. How succesfully do topic models extracted from the descriptions of issue records explain vulnerabilities?

4. PROPOSED METHOD

4.1 Data Used in The Method

In the thesis study, two different datasets were used. One of the dataset belongs to an information technology company the other dataset belongs to the Wireshark project. As security-related data are critical data for companies, company information is kept confidential.

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues [22].

For Wireshark dataset; collect the data to be used based on the bug records. Software vulnerabilities in the project are published by security advisory. In these reports, the bug record and the CVE id to which the vulnerabilities depend are shared. Bug records are recorded on the Bugzilla. Bugzilla is a bug and version tracking system developed by the Mozilla team and distributed with free software licenses [23]. The first release was released by Netscape in 1998 and was used by many companies to track bugs in open source and proprietary software. For the model to be created from Wireshark dataset, the bug and vulnerability records opened and closed in 2017 and 2018 were used. 106 software vulnerabilities were identified for the Wireshark project. Of the 106 software vulnerabilities, 105 were mapped to a bug record. Including these 105 vulnerability records, a total of 1112 bug records resolved in 2017 and 2018 were taken over Bugzilla.

For company dataset; firstly, processes were examined and data records were analyzed. As a result of the dataset analysis, it was decided to collect the data to be used based on the software product. All software products could not be included in the study because the infrastructure was not ready for all products. The sample software product group was used with the company dataset. In the 6-month dataset, an average of 26 software product data were obtained each month : table 4.1.2.

The company performs software security checks with CxSAST tool. For the thesiss study, 6-month screening data for the last half of 2018 were used. The data obtained from the survey includes software product based monthly security data. The report presents the findings for a software product grouped by severity. Severity levels are high, medium and low respectively. The report also shares the number of lines of code scanned on a software product basis. The properties that come with the report are shared in the table 4.1.1.

Table 4.1 : CxSAST report feature.

Feature No	Features
1	ID
2	ProjectId
3	LOC
4	HighSeverity
5	MediumSeverity
6	LowSeverity
7	RiskLevelScore
8	FailedLOC
9	StatisticsCalculationDate

Table 4.2 : CxSAST Scan Data.

Date	Product Count	LOC	Low	Medium	High
2018.07	25	16.110.186	31062	10681	1132
2018.08	27	13.047.525	31497	9248	945
2018.09	24	15.343.368	31152	9697	3519
2018.10	24	22.336.829	41814	17218	4628
2018.11	29	27.025.372	50999	17655	5608
2018.12	26	26.786.588	63659	17540	5674

Another dataset used next to the vulnerabilities data is the issue records data. The issue records are the bugs opened by the help-desk or tester staff to the software products in the production. There are several systems for monitoring issue records. In the Chromium project uses Google Code as bug tracking system. Chromium is an open-source browser project that aims to build a safer, faster, and more stable way for all users to experience the web [14].

In company dataset JIRA is used for the management of the issue records and a monthly report is generated on the tool. The life cycle of the issue record is followed by JIRA, consists of four status. These status are Pending, In progress, Pre-release, Done respectively. Each record does not go through pre-release status because no software changes are required in each record. The 6-month issue records report was analyzed for this study. 39.015 different issue records were examined and 39 features were discovered for each issue records. Table 4.1.3 shows the record-based features that came with the issue record report. When the issue records report was examined, it appeared that there were records opened for reasons that did not affect the software. Issue record data were analyzed and features that could be associated with software changes were chosen. Issue records that caused the software change were filtered according to the root cause feature in the report. As a result of filtering and selecting product group records, we have 3516 records out of 39100 records.

4.2 Algorithms and Techniques Used in The Method

In the studies conducted to predict software vulnerabilities, machine learning algorithms, text mining techniques, natural language processing algorithms were used in the literature research. In this thesis, these algorithms and techniques are included. In the data processing steps, text mining and natural language processing algorithms were used.

The most common methods used in vulnerabilities prediction studies are machine learning algorithms. Prediction models usually combine software metrics, textual metrics and vulnerabilities information to learn which modules seem to be more vulnerable-prone. In this thesis, the problem is discussed in two different ways for two different dataset. The classification problem for the Wireshark project and the regression problem for the company data are considered. Three different machine learning algorithms have been applied for this regression problem and in the data processing steps, text mining and natural language processing algorithms were used. Also three different machine learning algorithms have been applied for this classification problem and in the data processing steps, text mining and natural

Table 4.3: Issue Records Features.

Features	Uniq	Missing Value	Code Base
ID	+	-	-
RecordNo	+	-	-
Solution	-	-	-
Summary	+	-	-
Assignee	-	-	-
UnitName	-	-	+
RecordType	-	-	-
Headship	-	-	+
Status	-	-	-
ReportDate	-	-	-
Product	-	-	+
Platform	-	-	+
CreatedDate	-	-	-
RecordUpdate	-	-	-
IssueCategory	-	-	+
SolutionDate	-	-	-
SolutionMonth	-	-	-
SolutionYear	-	-	-
CompetionPeriod	-	-	-
CreatedMonth	-	-	-
CreatedYear	-	-	-
GeneratedSeverity	-	-	-
Outsourcer	-	+	-
Deadline	-	-	-
ProductCode	-	-	+
PlatformCode	-	-	+
NumberofReturn	-	-	-
Description	+	-	+
SolutionDescription	-	+	+
ReturnDescription	-	+	-
RootCauseCategory	-	+	+
SolutionCategory	-	+	+
AuthorizedGroup	-	-	+

language processing algorithms were used. In this section, these algorithms and techniques will be explained briefly.

First of all, we processed the data with text mining. The data we apply to text mining techniques are the descriptions of the issue records and bug records. Text mining, is the process of deriving high-quality information from text [24]. Mining and analyzing text helps organizations find potentially valuable business insights in corporate documents and other sources of text-based data. In this thesis, firstly basic text mining steps were operated step by step in the issue record and bug records descriptions.

Issue records from corporate dataset and bug records from OS dataset descriptions are parsed with text mining operations. Descriptions from the issue records of six -months and two years of bug records are passed through the following text mining stages [49];

- Split the description into sentences and into words.
- Lowercase the words and remove punctuation.
- Words that have fewer than 2 characters are removed.
- Stopwords are determined and removed.
- Words are stemmed and reduced to their root form.

The two datasets contain data in different languages. The issue records contain textual descriptions in Turkish, and bug records contain textual descriptions in English. Therefore, different stop words libraries were used.

As a result of these steps, each issue and bug record description text is ready for the feature selection step. In order to measure the repeat frequency of the words in the text, the TF-IDF (The Term Frequency - Inverse Document Frequency) statistical method is applied to the issue record description text passing through the text mining steps [25]. As a result of this operation, the weight factor of each word is calculated with the formula;

TF(x)= How many times have passed in x / Total number of terms in the record

IDF(x) = $\log_e$(Total number of terms / Number of records in which x is the term)

$$\text{TF-IDF}(x) = \text{TF}(x) * \text{IDF}(x) \qquad (4.1)$$

Topic modeling is a machine learning and natural language processing research area for determining the basic semantic structure of the text document [26], [27], [28], [29]. A large amount of non-structural text documents can be automatically organized, searched, and summarized using topic modeling methods. Latent Dirichlet Allocation

(LDA) is implemented as topic modeling method [50]. Within the scope of this study, the effectiveness of the data representation of the LDA is evaluated on the issue record description texts so that the text documents can be represented effectively.

Table 4.4: Example of Topic Model (Topic 0).

Topic No	Words
0	0.021*"word1" + 0.018*"word2" + 0.017*"word3" + 0.016*"word4" + 0.016*"word5" + 0.014*"word6" + 0.012*"word7" + 0.011*"word8" + 0.011*"word9" + 0.010*"word10"

Example topic model for issue records, topic 0, created in Table 4.2.1 were shared. Words have been masked because of the confidentiality agreement. Five main topics were determined using 3516 issue record datasets. With this step, topic-based scores were obtained for each issue record.

Topic models and text mining studies were developed by using *python* language in *pycharm* community IDE. *Gensim, sklearn* and *pandas et al.* libraries were used when creating topic models. DB browser for *SQlite* is used to store the dataset.

It has been observed that these algorithms are used in prediction models examined in the literature and successful results are obtained [13], [14], [15]. Within the scope of the thesis, models with different algorithms have been established. The three most successful algorithms are explained in the thesis.

Machine learning and statistical methods were investigated for regression problem; *Support vector regression (SVR), Artificial Neural Network (ANN)* and *Classification and Regression Trees (CART)* were decided to be used.

Machine learning and statistical methods were investigated for regression problem; *Naïve Baye s(NB), Fisher's Linear Discriminant Algorithm (FLDA)* and *Random Tree (RT)* were decided to be used.

First, the support vector regression was investigated. SVR was developed for regression problems from support vector machines (SVM) algorithm. SVR is an application of SVM to time-series forecasting [30].

Given a training dataset, $(x_1, y_1),\ldots,(x_N , y_N)$ where $x_i \in X$, $y_i \in R$, N is the size of training data, and X denotes the space of the input samples–for instance, R^n. The aim is to find a function which can estimate all these data well. SVR is one of the methods to perform the regression task [30]. In general, the estimation function in SVR takes the following form,

$$f(x)=(w \cdot \varphi(x)) + b \tag{4.2}$$

In this thesis, SVR will be applied with WEKA tool. SMOreg function implements the support vector machine for regression on WEKA [31].

ANN is a computational model based on the structure and functions of biological neural networks. Neural Network is a structure built in layers. The first layer input is called the last layer output. The middle layers are called Hidden Layers. Each layer contains a certain number of Neurons. These neurons are connected to each other by Synapses. Synapses contains a factor. These coefficients tell us how important the knowledge in the neurons they are connected to.

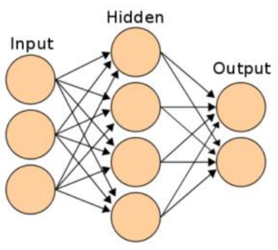


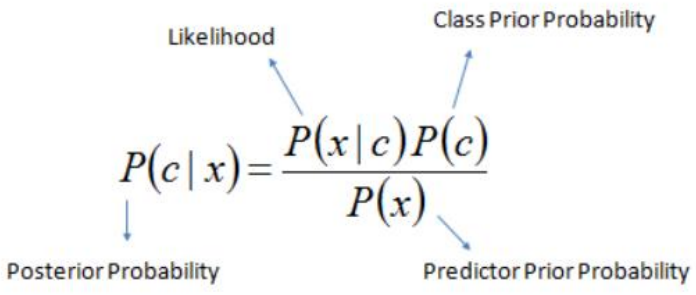
Figure 4.1 : Neural Network layer.

In this thesis, ANN will be applied with WEKA tool. MultilayerPerceptron function implements on WEKA [31].

Decision Trees are an important type of algorithm for predictive modeling machine learning. In the decision tree learning, a tree structure is formed and the class labels on the leaf level of the tree and the handles that go to these leaves and with the arms coming from the beginning are expressed. Requires fast data preprocessing. According to most alternative techniques, data can be used with very little processing. The pretreatment stage is shorter and simpler than the other alternatives. It can be used for processing both numeric and class data. Most machine learning algorithms are either useful in numerical applications or useful for classification problems. Decision tree learning can be used in both areas [32].

Decision Trees uses the white box model. In the white box model, which is an approach to software engineering, each step is viewable and interpretable. Again, black box approach, which is a software engineering application, is mostly covered by artificial neural network in machine learning. While the input and output can be interpreted in this method, it is not possible to observe and interpret the internal dynamics of the system at every step.

Naive Bayes is an algorithm that performs operations according to the probability calculation. It handles the train data according to its formula and produces a percentage ratio for each case and performs the classification according to the probabilities [32].



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 4.2 : Mathematical Representation of Naive Bayes.

Fisher's linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more [33].

5. PROPOSED MODEL

We aim to try a new association for our vulnerability prediction model. This study investigates the prediction of software vulnerabilities using topic models derived from textual descriptions of issue records and machine learning techniques. Within the scope of the thesis study, we use two different datasets. The first dataset belongs to the information technology company and the second dataset is the open source dataset of the wireshark project. The models to be created from the first datas will be called company models and the models created with the wireshark project dataset will be called open source (OS) models.

The proposed company model is based on the software product. The target is to predict how vulnerable the software product is. Our research questions will be answered with the proposed model. Therefore, we aim to propose solutions for them in the later sections throughout the experiments.

We consider our problem as a regression problem for company dataset. Artifical neural networks, support vector regression and classification and regression trees methods were used in the creation of the model. We consider our second problem as a classification problem for Wireshark dataset. Artifical neural networks, support vector regression and classification and regression trees methods were used in the creation of the model.

In the development of the system, we were used: issue records and CxSAST scan reports. Two datasets are matched and modeled on the basis of software products.

The proposed OS model is based on the bug records. This problem is to correctly predict whether the intended bug record has caused the vulnerability. Our research questions will be answered with the proposed model. Therefore, we aim to propose solutions for them in the later sections throughout the experiments.

We consider our problem as a classification problem for OS dataset. Artifical neural networks, support vector regression and classification and regression trees methods were used in the creation of the model. We consider our second problem as a classification problem for Wireshark dataset. Naïve Bayes, Fisher's Linear Discriminant and Random Tree methods were used in the creation of the model.

The built models were trained with two different cross-validation techniques. One of them is 10-fold cross validation and the other one is leave-one-out technique. The reason for using the leave-one-out technique is the lack of data.

Vulnerabilities prediction models basically take a set of independent attributes and retrieve a class label. It consists of four steps, which will be explained in this section:

- “What are the inputs to the vulnerability prediction model?”
- “What are the outputs of the vulnerability prediction model?”
- “How does the model operate between the inputs and output?”
- “How can we assess the performance of the model?”

5.1 Inputs of the Company Model

As inputs of this study, we need list of software products. From these software products, we collect issue records which are product-based records. Issue records features can be easily collected from JIRA. Moreover, they provide useful information about the characteristics of software product in terms of maintenance costs. Selected issue record features shared in table 5.1.1.

Table 5.1: Selected Issue Record Features.

Feature No	Features
1	ID
2	RecordNo
3	Solution
27	ProductCode
28	PlatformCode
30	Description
35	SolutionCategory

The LDA topic modeling method was run with the corpus generated from the 3516 issue record descripton. As a result of this stage, five topics were obtained.

Topic model scores were obtained on the basis of the issue record. In this thesis, two different models were created by using topic model scores of the issue records as input. These models will then be referred to as model A and model B for company model.

For model A, to obtain these issue record topic scores on the basis of the software product, the descriptions of the issue records that were opened to the software product were combined and tested in the model and the scores were obtained based on the product. Model A inputs shared in table 5.1.2.

Table 5.2 : Model A Inputs.

Feature No	Features
1	ProductClass
2	ProductID
3	TOPIC_0
4	TOPIC_1
5	TOPIC_2
6	TOPIC_3
7	TOPIC_4
8	IssueRecord_Level_1
9	IssueRecord_Level_2
10	IssueRecord_Level_3
11	LineofCodes

For model B, we moved topic model scores from issue record level to software product level through the aggregation schemes [34]. The aggregation schemes which we use in model B are shown in table 5.1.3. Model B inputs shared in table 5.1.4.

Table 5.3: List of the aggregation schemes.

Category	Aggregation schemes	Formula
Central Tendency	Minimum	Min(m_i)
Central Tendency	Maximum	Max(m_i)
Central Tendency	Median	$M_m = \begin{cases} m \\ \frac{1}{2}(m \end{cases}$
Inequality Index	Hoover Index	$H_m = \frac{1}{2} \sum_{i=1}^{N} \mid \frac{m_i}{\sum m} - \frac{1}{N} \mid$

To evaluate the prediction performance of our vulnerability prediction models, we use topic model scores, issue records count and software product severity.

Table 5.4: Model B Inputs.

Feature No	Features
1	ProductClass
2	ProductID
3	TOPIC_0_min
4	TOPIC_0_max
5	TOPIC_0_hoover
6	TOPIC_0_median
7	TOPIC_1_min
8	TOPIC_1_max
9	TOPIC_1_hoover
10	TOPIC_1_median
11	TOPIC_2_min
12	TOPIC_2_max
13	TOPIC_2_hoover
14	TOPIC_2_median
15	TOPIC_3_min
16	TOPIC_3_max
17	TOPIC_3_hoover
18	TOPIC_3_median
19	TOPIC_4_min
20	TOPIC_4_max
21	TOPIC_4_hoover
22	TOPIC_4_median

5.2 Inputs of the OS Model

As inputs of wireshark project model, we need description of bug records. Bug records features can be easily collected from Bugzilla. Moreover, they provide useful information about the characteristics of software product in terms of maintenance costs. The features found in the bug report taken from Bugzilla are shown in the table 5.2.1.

The LDA topic modeling method was run with the corpus generated from the 1122 bug record summary. As a result of this stage, five topics were obtained and used for inputs. Model OS inputs shared in table 5.2.2.

Table 5.5: Bug record features.

Feature No	Features
1	Bug ID
2	Product
3	Component
4	Assignee
5	Status
6	Resolution
7	Summary
8	Year
9	Month

Table 5.6: Model OS Inputs.

No	Inputs
1	Topic 0
2	Topic 1
3	Topic 2
4	Topic 3
5	Topic 4

5.3 Outputs of the OS Model

Software vulnerability prediction can be described as a classification algorithm for model OS. It will be more accurate to specify the classification data in the property that will generate the class data instead of the output. In order to process the class data, advisory reports published in the wireshark project were used. Each of the vulnerabilities published in these reports was mapped to a bug. Vulnerable class of bugs that are paired with a vulnerability is marked yes.

5.4 Outputs of the Company Model

Software vulnerability prediction can be described as a regression algorithm for company model. We make predictions about the vulnerability count of the software product with our model. Therefore, the expected outputs are from the CxSAST report. The number of software vulnerabilities in the report is divided into three severity levels which are low, medium, high. Each severity level was evaluated singularly. Company outputs shared in table 5.4.1. For example, low severity measurement was used as a single output in models. The sum of the measurements of all severity levels is also used as an output in the models which name is total. In the table 5.4.2 shared the model details where model output is low.

Table 5.7: Outputs (Company Model).

Feature No	Features
1	Low
2	Medium
3	High
4	Total

Table 5.8: Models (output: low).

Output	Model	Technique	Concept	Cross-validation
Low	Model A	SVR	Kernel: Poly C:1	10-folds
Low	Model A	SVR	Kernel: Poly C:1	Leave-one-out
Low	Model B	SVR	Kernel: Poly C:1	10-folds
Low	Model B	SVR	Kernel: Poly C:1	Leave-one-out
Low	Model A	SVR	Kernel: Puk C:1	10-folds
Low	Model A	SVR	Kernel: Puk C:1	Leave-one-out
Low	Model B	SVR	Kernel: Puk C:1	10-folds
Low	Model B	SVR	Kernel: Puk C:1	Leave-one-out
Low	Model A	SVR	Kernel: RBF C:1	10-folds
Low	Model A	SVR	Kernel: RBF C:1	Leave-one-out
Low	Model B	SVR	Kernel: RBF C:1	10-folds
Low	Model B	SVR	Kernel: RBF C:1	Leave-one-out
Low	Model A	CART	NoPuring : True MaxDepht : -1	10-folds
Low	Model A	CART	NoPuring : True MaxDepht : -1	Leave-one-out
Low	Model B	CART	NoPuring : True MaxDepht : -1	10-folds
Low	Model B	CART	NoPuring : True MaxDepht : -1	Leave-one-out
Low	Model A	ANN	-	10-folds
Low	Model A	ANN	-	Leave-one-out
Low	Model B	ANN	-	10-folds
Low	Model B	ANN	-	Leave-one-out

5.5 Assessing the Performance of the Model

The prediction results obtained by testing the models were compared with the results of real software vulnerabilities. Absolute Error (AE), Relative Error (RE), Mean magnitude of relative error (MMRE), Median magnitude of relative error (MdMRE), Prediction Indicator (Pred(k)) and Correlation Coefficient (CC) used for evaluating regression tasks. Precision, Recall, F-measure and Area Under The Curve (AUC) used for evaluating classification tasks.

The formulas used to obtain the results are shown in the tables 5.5.1, 5.5.2, 5.5.3.

Table 5.9 : Absolute Error and Relative Error Equations.

Equation	Formula
AE	$\lvert actual - predicted \rvert$
RE	$\frac{AE}{actual}$

Table 5.10 : Regression Evaluation Metrics.

Metric	Definition
MRE	$Abs(RE_i)$
MMRE	$\frac{1}{n}\sum_i RE_i$
MdMRE	$Median(MRE)$
Pred(k)	*percentage of estimates that are within n% of the actual value*
CC	*relationship is between two variables*

Table 5.11: Classification Evaluation Metrics.

Metric	Definition
Precision	$\frac{TP}{TP + FP}$
Recall	$\frac{TP}{TP + FN}$
F-Measure	$\frac{2 * precision * recall}{precision + recall}$
AUC	*Chosen positive example is actually positive than that a randomly chosen negative example is positive*

6. RESULTS

There are four different output classes used in this thesis for regression problem. For each output, two models were tested separately and a total of 80 tests were performed. The results of the test were compared with the actual values. Predictive errors of estimation values are calculated. For error calculation, MRE, MMRE, PRED (25), MdMRE values which are commonly used in regression problems are used. The error values obtained are shared in the tables.

When the values were examined, it was observed that the best result was obtained in the models with the total value of the output. Box plot graphics of the models are available in the annex. Model A performs slightly better than Model B in terms of MMRE, MdMRE, and Pred(25). The aggregation processes used had a negative effect on the results obtained in Model B. Observations were also in this direction in the literature research. In order to be able to select a model, MMRE and MdMRE values should be close to zero. When we evaluate the models via the MdMRE error metric, the CART model created under Model A gives the best results. However, the results were not successful enough to make a recommendation. The obtained results are aimed to be improved in the future studies by expanding the dataset and evaluating the outiers.

The model established for the classification problem has been tested with selected machine learning algorithms. The model results were analyzed with the classifications evaluation metrics. The results are more successful compared to the results obtained with the company dataset. In this model study, bug record based classification did not narrow down the dataset and the results were more successful than company dataset.

Although Model OS is more successful than Company Model, two problems are considered in different types. Datasets are not suitable for installing the same model.

The data of corparate companies are not suitable for establishing every model recommend in the literature. In tables with start 6, shared the result details.

Table 6.1: Regression result for Model A (Output : low).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
Low	Model A	1	SVR	Kernel: Poly, C:1	10-folds	1,62	0,62	0,25	0.42
Low	Model A	2	SVR	Kernel: Poly, C:1	Leave-one-out	1,79	0,60	0,25	0.44
Low	Model A	3	SVR	Kernel: Puk, C:1	10-folds	10,04	0,86	0,14	0.33
Low	Model A	4	SVR	Kernel: Puk, C:1	Leave-one-out	9,60	0,96	0,14	0.36
Low	Model A	5	SVR	Kernel: RBF, C:1	10-folds	3,12	0,75	0,23	0.40
Low	Model A	6	SVR	Kernel: RBF, C:1	Leave-one-out	2,66	0,75	0,21	0.40
Low	Model A	7	CART	NoPuring : True	10-folds	2,86	0,44	0,35	0.69
Low	Model A	8	CART	NoPuring : True	Leave-one-out	1,47	0,32	0,45	0.80
Low	Model A	9	ANN	-	10-folds	24,47	2,13	0,06	0.14
Low	Model A	10	ANN	-	Leave-one-out	19,48	2,39	0,10	0.47

Table 6.2: Regression result for Model A (Output : medium).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
Medium	Model A	1	SVR	Kernel: Poly, C:1	10-folds	1,83	0,79	0,17	0.30
Medium	Model A	2	SVR	Kernel: Poly, C:1	Leave-one-out	1,52	0,75	0,17	0.31
Medium	Model A	3	SVR	Kernel: Puk, C:1	10-folds	18,07	1,53	0,14	0.16
Medium	Model A	4	SVR	Kernel: Puk, C:1	Leave-one-out	17,15	1,75	0,14	0.23
Medium	Model A	5	SVR	Kernel: RBF, C:1	10-folds	2,29	0,75	0,23	0.32
Medium	Model A	6	SVR	Kernel: RBF, C:1	Leave-one-out	2,42	0,72	0,18	0.32
Medium	Model A	7	CART	NoPuring : True	10-folds	21,79	0,88	0,22	0.63
Medium	Model A	8	CART	NoPuring : True	Leave-one-out	13,76	0,71	0,23	0.45
Medium	Model A	9	ANN	-	10-folds	147,07	4,90	0,07	0.11
Medium	Model A	10	ANN	-	Leave-one-out	91,65	6,33	0,08	0.14

Table 6.3: Regression result for Model A (Output : High).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
High	Model A	1	SVR	Kernel: Poly, C:1	10-folds	9,41	1,00	0,07	0.15
High	Model A	2	SVR	Kernel: Poly, C:1	Leave-one-out	9,58	1,16	0,04	0.25
High	Model A	3	SVR	Kernel: Puk, C:1	10-folds	28,81	3,13	0,07	0.68
High	Model A	4	SVR	Kernel: Puk, C:1	Leave-one-out	27,84	2,94	0,06	0.82
High	Model A	5	SVR	Kernel: RBF, C:1	10-folds	7,62	0,98	0,05	0.17
High	Model A	6	SVR	Kernel: RBF, C:1	Leave-one-out	7,70	0,99	0,06	0.25
High	Model A	7	CART	NoPuring : True	10-folds	25,26	0,96	0,12	0.44
High	Model A	8	CART	NoPuring : True	Leave-one-out	21,60	0,90	0,15	0.61
High	Model A	9	ANN	-	10-folds	121,48	8,00	0,05	0.20
High	Model A	10	ANN	-	Leave-one-out	96,98	6,90	0,04	0.38

Table 6.4: Regression result for Model A (Output : Total).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
Total	Model A	1	SVR	Kernel: Poly, C:1	10-folds	1,17	0,65	0,22	0.42
Total	Model A	2	SVR	Kernel: Poly, C:1	Leave-one-out	1,17	0,64	0,21	0.48
Total	Model A	3	SVR	Kernel: Puk, C:1	10-folds	11,65	1,14	0,12	0.44
Total	Model A	4	SVR	Kernel: Puk, C:1	Leave-one-out	11,14	1,10	0,14	0.49
Total	Model A	5	SVR	Kernel: RBF, C:1	10-folds	2,03	0,68	0,29	0.48
Total	Model A	6	SVR	Kernel: RBF, C:1	Leave-one-out	1,81	0,68	0,32	0.47
Total	Model A	7	CART	NoPuring : True	10-folds	10,31	0,30	0,44	0.62
Total	**Model A**	**8**	**CART**	**NoPuring : True**	**Leave-one-out**	**1,06**	**0,23**	**0,52**	**0.74**
Total	Model A	9	ANN	-	10-folds	21,87	1,84	0,07	0.45
Total	Model A	10	ANN	-	Leave-one-out	22,72	1,93	0,12	0.44

Table 6.5 : Regression result for Model B (Output : Low).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
Low	Model B	1	SVR	Kernel: Poly, C:1	10-folds	6,63	0,83	0,17	0.02
Low	Model B	2	SVR	Kernel: Poly, C:1	Leave-one-out	4,73	0,88	0,17	0.05
Low	Model B	3	SVR	Kernel: Puk, C:1	10-folds	17,94	1,83	0,10	0.23
Low	Model B	4	SVR	Kernel: Puk, C:1	Leave-one-out	17,14	1,74	0,11	0.26
Low	Model B	5	SVR	Kernel: RBF, C:1	10-folds	5,43	0,87	0,19	0.04
Low	Model B	6	SVR	Kernel: RBF, C:1	Leave-one-out	4,77	0,84	0,20	0.04
Low	Model B	7	CART	NoPuring : True	10-folds	23,75	0,67	0,21	0.44
Low	Model B	8	CART	NoPuring : True	Leave-one-out	20,12	0,78	0,19	0.33
Low	Model B	9	ANN	-	10-folds	47,46	2,66	0,10	0.25
Low	Model B	10	ANN	-	Leave-one-out	23,47	2,80	0,07	0.17

Table 6.6: Regression result for Model B (Output : Medium).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
Medium	Model B	1	SVR	Kernel: Poly, C:1	10-folds	5,84	0,79	0,16	0.09
Medium	Model B	2	SVR	Kernel: Poly, C:1	Leave-one-out	5,69	0,87	0,14	0.04
Medium	Model B	3	SVR	Kernel: Puk, C:1	10-folds	25,63	2,56	0,10	0.17
Medium	Model B	4	SVR	Kernel: Puk, C:1	Leave-one-out	25,13	2,80	0,10	0.23
Medium	Model B	5	SVR	Kernel: RBF, C:1	10-folds	4,79	0,77	0,14	0.00
Medium	Model B	6	SVR	Kernel: RBF, C:1	Leave-one-out	4,52	0,80	0,16	0.01
Medium	Model B	7	CART	NoPuring : True	10-folds	17,59	0,83	0,14	0.15
Medium	Model B	8	CART	NoPuring : True	Leave-one-out	13,24	0,67	0,23	0.20
Medium	Model B	9	ANN	-	10-folds	75,29	6,21	0,08	0.00
Medium	Model B	10	ANN	-	Leave-one-out	83,92	4,82	0,07	0.15

Table 6.7: Regression result for Model B (Output : High).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
High	Model B	1	SVR	Kernel: Poly, C:1	10-folds	10,91	0,98	0,09	0.19
High	Model B	2	SVR	Kernel: Poly, C:1	Leave-one-out	9,58	0,99	0,07	0.21
High	Model B	3	SVR	Kernel: Puk, C:1	10-folds	37,60	4,52	0,05	0.41
High	Model B	4	SVR	Kernel: Puk, C:1	Leave-one-out	36,19	3,35	0,04	0.71
High	Model B	5	SVR	Kernel: RBF, C:1	10-folds	8,716	1,50	0,02	0.09
High	Model B	6	SVR	Kernel: RBF, C:1	Leave-one-out	8,92	1,00	0,01	0.13
High	Model B	7	CART	NoPuring : True	10-folds	25,02	0,96	0,14	0.38
High	Model B	8	CART	NoPuring : True	Leave-one-out	19,83	0,72	0,25	0.49
High	Model B	9	ANN	-	10-folds	82,81	10,84	0,06	0.24
High	Model B	10	ANN	-	Leave-one-out	70,94	13,00	0,07	0.46

Table 6.8 : Regression result for Model B (Output : Total).

Output	Model	No	Technique	Concept	Cross-validation	MMRE	MdMRE	Pred(0,25)	CC
Total	Model B	1	SVR	Kernel: Poly, C:1	10-folds	8,36	0,93	0,13	0.08
Total	Model B	2	SVR	Kernel: Poly, C:1	Leave-one-out	6,37	0,93	0,08	0.02
Total	Model B	3	SVR	Kernel: Puk, C:1	10-folds	19,35	1,22	0,14	0.37
Total	Model B	4	SVR	Kernel: Puk, C:1	Leave-one-out	18,62	1,199	0,14	0.39
Total	Model B	5	SVR	Kernel: RBF, C:1	10-folds	4,17	0,84	0,22	0.08
Total	Model B	6	SVR	Kernel: RBF, C:1	Leave-one-out	3,79	0,79	0,22	0.08
Total	Model B	7	CART	NoPuring : True	10-folds	12,67	0,73	0,27	0.45
Total	**Model B**	**8**	**CART**	**NoPuring : True**	**Leave-one-out**	**14,41**	**0,44**	**0,32**	**0.50**
Total	Model B	9	ANN	-	10-folds	30,55	1,63	0,10	0.08
Total	Model B	10	ANN	-	Leave-one-out	34,22	1,91	0,13	0.37

Table 6.9: Classification result for Model OS.

Metric	Technique	Cross-validation	Yes	No	Weighted
Precison	NB	10-folds	0,26	0,95	0,89
	NB	One-leave-out	0,28	0,96	0,90
	FLDA	10-folds	0,19	**0,97**	**0,90**
	FLDA	One-leave-out	0,19	**0,97**	**0,90**
	RT	10-folds	**0,37**	0,95	0,89
	RT	One-leave-out	0,35	0,94	0,89
Recall	NB	10-folds	0,59	0,84	0,82
	NB	One-leave-out	0,65	0,86	0,82
	FLDA	10-folds	**0,74**	0,68	0,69
	FLDA	One-leave-out	**0,74**	0,68	0,60
	RT	10-folds	0,42	**0,93**	**0,88**
	RT	One-leave-out	0,36	**0,93**	**0,88**
F-Measure	NB	10-folds	0,37	0,89	0,85
	NB	One-leave-out	**0,39**	0,89	0,85
	FLDA	10-folds	0,30	0,80	0,75
	FLDA	One-leave-out	0,30	0,80	0,75
	RT	10-folds	**0,40**	**0,94**	**0,89**
	RT	One-leave-out	0,35	**0,94**	0,88
AUC	NB	10-folds	**0,75**	**0,75**	**0,75**
	NB	One-leave-out	0,74	0,74	0,74
	FLDA	10-folds	0,72	0,72	0,72
	FLDA	One-leave-out	0,71	0,71	0,71
	RT	10-folds	0,65	0,65	0,65
	RT	One-leave-out	0,63	0,63	0,63

7. THREATS TO VALIDITY

Possible threats to the validity of our findings are the dataset used. The company dataset has been studied for a period of 6-months, causing this dataset to shrink. It is planned to expand this dataset in the future studies and to repeat the study.

New features can be added by re-evaluating the feature selection step for the Model OS. In this study, only the topic scores of the bug records were classified. Only the 2017 and 2018 years from the Wireshark dataset have been used to build the model, while the training dataset range can be expanded to observe its effect on the prediction success.

The textual descriptions used from the company dataset and the Wireshark dataset were written in different languages. The text data entered is not written according to any standard. Pollution of text data might have led to separate time spent in text mining steps.

8. CONCLUSIONS AND FUTURE WORK

In this thesis, we have investigated whether topic models obtained from textual descriptions of bugs and issue records may be a helpful criterion in predicting software vulnerabilities.

This empirical analysis of predicting software vulnerabilities could be considered as a new prediction approach. In this thesis, we identify two research questions. The first research question is about the prediction of software vulnerabilities using issue records. The results show that there is a considerable relationship between bug/issue reports and vulnerabilities. In the second question, the success rates of the predictions made by using textual description and topic models obtained from textual descriptions were traced. The results show that software vulnerabilities can be avoided by tracking the scores of the topic models before software vulnerabilities occur. Especially when the Wireshark model results are examined, it is seen that the textual description of the bug records provide successful results in predicting software vulnerabilities. The recall score of the Wireshark model established by Fisher's Linear Discriminant algorithm is 74%. In the evaluation of the models for the classification problem, recall values have been prioritized as success criteria. Because the number of error records matched with the vulnerability in the dataset is about 10%. High recall score at this rate indicates that false positive values are high, which makes the model successful.

The aggregation procedures on topic model scores for the corporate data contributed negatively to the predictors' ability to learn. The same observation was obtained in the literature [34]. Standardizing the textual description of the bug and issue records would increase the success rate of the model.

The starting point of the thesis was to built the models developed in the literature with corporate firm data. However, the dataset used in the literature and the different datasets of corporate firms prevented replica studies. The investigation of corporate company processes and the data mining phase took longer than anticipated. The dataset obtained has a limited time interval. The thesis study will be developed in future studies with a larger dataset.

Wireshark project dataset was found to be appropriate for this study, which is intended to be compared with open source project data. Software vulnerabilities opened under the Wireshark project were paired with one-to-one bug records, which led to the problem of classification. As a result of this study, it is determined by the test results that the textual descriptions of the bug records contribute to the successful results in predicting the software vulnerabilities.
Outlier elements were observed in the company dataset. The dataset is under investigation. In future studies, it is planned to evaluate the outlier and categorize the dataset.

Naive Bayes and Fisher's Linear Discriminant algorithms achieve successful results for the classification problem. In future studies, it is planned to test and compare these machine learning algorithms with different datasets.

The sample software product group was used with the company dataset. In the future studies, it is planned to make prediction by being consistent with all products belonging to the company dataset. The reason for selecting the sample software product group is the lack of readiness of the measurement infrastructure.

REFERENCES

[1] **McGraw, G. (**2004). Software Security, IEEE Security & Privacy

[2] **TÜBİTAK BİLGEM, (**2018). Secure Software Development Guide

[3] **Ozment, A. (**2007). Vulnerability Discovery & Software Security**,** University of Cambridge, PhD Thesis

[4] **Antunes, N., and Vieira, M,.** (2015). On the Metrics for Benchmarking Vulnerability Detection Tools**,** *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Rio de Janeiro, Brazil doi: 10.1109/DSN.2015.30

[5] **Halfond, W. G. J., Choudhary, S., and Orso, A., (**2009): Penetration Testing with Improved Input Vector Identification. *International Conference on Software Testing Verification and Validation*, EEE Computer Society Washington, DC, USA

[6] **OWASP,** The Open Web Application Security Project, "About", date retrieved 02.05.2019 https://www.owasp.org/index.php/Source_Code_Analysis_Tools**,**

[7] **CxSAST**, Static Application Security Testing, **,** "About", date retrieved 02.05.2019 https://www.checkmarx.com/products/static-application-security-testing/

[8] **MITRE,** "Corporate Overwiev", date retrieved 18.04.2019, www.mitre.org.

[9] **CVE,** Common Vulnerabilities and Exposures**,** "About", date retrieved 18.04.2019, https://cve.mitre.org/about/

[10] **NIST,** The National Institute of Standards and Technology, date retrieved 18.04.2019, https://www.nist.gov/

[11] **NVD,** National Vulnerability Database, date retrieved 08.11.2016, https://nvd.nist.gov/

[12] **Shin, Y., Meneely, A., Williams, L., and Osborne, J. A., (**2010). Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities, *IEEE Transactions on Software Engineering.*, 772 – 787

[13] **Scandariato, R., and Walden, J. S., (**2014). Predicting vulnerable components: software metrics vs text mining, *IEEE 25th International Symposium on Software Reliability Engineering.*, Naples, Italy DOI: 10.1109/ISSRE.2014.32

[14] **Camilo, F., Meneely, M., and Nagappan, M., (**2015). Do Bugs Foreshadow Vulnerabilities? A Study of the Chromium Project, *IEEE/ACM 12th Working Conference on Mining Software Repositories*, Florence, Italy DOI: 10.1109/MSR.2015.32

[15] **Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., and Li, S., (**2015). Combining software metrics and text features for vulnerable file prediction, *20th International Conference on Engineering of Complex Computer Systems (ICECCS),* Gold Coast, QLD, Australia DOI: 10.1109/ICECCS.2015.15

[16] **Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., Chen, Z., Wang, S., and Wang, J., (**2018). SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities, *IEEE* https://arxiv.org/pdf/1807.06756

[17] **Chowdhury, I., and Zulkernine, M.,** (2010). Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities**,** *ACM Symposium on Applied Computing* Sierre, Switzerland

[18] **Hovsepyan, A., Scandariato, R., Joosen, W., and Walden, J., (**2014). Software Vulnerability Prediction using Text Analysis Techniques", IEEE 25th International Symposium on Software Reliability Engineering., Naples, Italy

[19] **Zimmermann, T., Nagappan, N., and Williams, L., (**2015). Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista, *International Conference on Software Testing, Verification and Validation (ICST),* Paris, France DOI: 10.1109/ICST.2010.32

[20] **Aversano, L., Cerulo, L., and Grosso, C. D., (**2007). Learning from bug introducing changes to prevent fault prone code, *International Workshop on Principles of software Evolution (IWPSE)* Pages 19-26

[21] **Gegick, M., Rotella,P., and Xie,T., (**2010). Identifying security bug reports via text mining: An industrial case study, *7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* Cape Town, South Africa DOI: 10.1109/MSR.2010.5463340

[22] **Wireshark Vulnerabilities,** Vulnerability Advisory, date retrieved 01.04.2019**,** https://www.wireshark.org/security/

[23] **Bugzilla,** date retrieved 01.04.2019**,** https://www.bugzilla.org/

[24] **Text Mining** https://en.wikipedia.org/wiki/Text_mining, date retrieved 19.04.2019

[25] **Qu, S., Wang, S., and Zou,Y., (**2008). Improvement of Text Feature Selection Method Based on TFIDF, in Future Information Technology and Management Engineering, FITME '08. International Seminar on, pp.79-81

[26] **Chen, T.H., Thomas, S. W. and Hassan, A. E.** (2016). "A Survey on the Use of Topic Models When Mining Software Repositories," Empirical Software Engineering, vol. 21, no. 5, pp. 1843–1919

[27] **Debortoli, S., Müller, O., Junglas, I., and Brocke,J. (**2016). Text Mining for Information Systems Researchers: An Annotated Topic Modeling Tutorial, Communications of the Association for Information Systems, vol. 39, p. Article 7

[28] **Blei, D. M. (**2012). “Probabilistic Topic Models,” Communications of the ACM, vol. 55, no. 4, pp. 77–84

[29] **Neuhaus S. and Zimmermann, T. (**2010). “Security Trend Analysis with CVE Topic Models,” in Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE 2010). San Jose: IEEE, 2010, pp. 111–120.

[30] **Smola, A. J. and Schölkopf, B. (**2004). A tutorial on support vector regression, *Statistics and Computing, Kluwer Academic Publishers* https://doi.org/10.1023/B:STCO.0000035301.49549.88

[31] **WEKA,** 2019 Waikato Environment for Knowledge Analysis, date retrieved 19.04.2019 http://weka.sourceforge.net/doc.stable-38/weka/classifiers/functions/SMOreg.html

[32] **Wang, T. and Li,W. (**2010). Naive Bayes Software Defect Prediction Model, *International Conference on Computational Intelligence and Software Engineering* DOI: 10.1109/CISE.2010.5677057

[33] **Kalsoom, A., Maqsood,M., Ghazanfar, M.A., Aadil, F., and Rho,S.** (2018). A dimensionality reduction-based efficient software fault prediction using Fisher linear discriminant analysis (FLDA), Springer US, *74: 4568.* https://doi.org/10.1007/s11227-018-2326-5

[34] **Zhang,F., Hassan, A.E., McIntosh,S., and Zou,Y. (**2016). The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING* https://doi.org/10.1109/TSE.2016.2599161

[35] **Perl, H.** (2015). “VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits”, CCS ’15, October 12-16, Denver, Colorado, USA

[36] **Arkin , B., Stender, S. and McGraw, G. (**2005). Software penetration testing, IEEE Security & Privacy, *84-87.* DOI: 10.1109/MSP.2005.23

[37] **Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A. and Gall**, **H. C.** (2018). "Context is king: The developer perspective on the usage of static analysis tools." IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). Campobasso, Italy.

[38] **Shahmehri, N. (**2012). “An advanced approach for modeling and detecting software vulnerabilities”, Information and Software Technology 54, pgs 997-1013, Elsevier,

[39] **Jovanović and Irena, (**2008). “Software Testing Methods and Techniques”, May 26,

[40] **Yoshioka, N., Washizaki, H., and Maruyma, K.** (2008). “A survey on security patterns,” Progress in Informatics, vol. 5, pp. 35–47, 2.

[41] **Rehman, S., and Mustafa, K.** (2009). "Research on Software Design LevelSecurity Vulnerabilities", ACM.

[42] **Devanbu, P. T. and Stubblebine,S.** (2000). "Software engineering for security: a roadmap", ICSE '00 Proceedings of the Conference on The Future of Software Engineering, Pages 227-239

[43**]** **Offutt, J.** (2002). "Quality Attributes of Web Software Applications", IEEE Software, Pages 25 – 32, DOI: 10.1109/52.991329

[44] **Zhi,J., Garousi,V., Sun,B., and Garousi, G.** (2014). "Cost, Benefits and Quality of Software Development Documentation: A Systematic Mapping", Journal of Systems and Software, DOI: 10.1016/j.jss.2014.09.042

[45] **Ghaffarian S.M., and Shahriari, H.R**. (2017). Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. ACM Comput. Surv. 50, 4, Article 56 (August 2017), 36 pages. https://doi.org/10.1145/3092566

[46] **Krsul**, **I. V.** "Software vulnerability analysis," (1998). Ph.D. dissertation, Purdue University

[47] **Apple**, (2019). "Types of Security Vulnerabilities", date retrieved 19.04.2019 https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Articles/TypesSecVuln.html

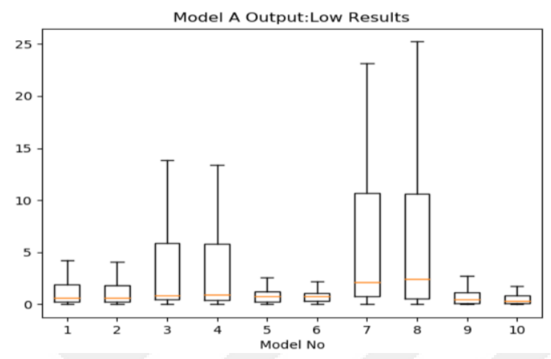
[48**] Alhazmi, O.H., Malaiya, Y.K. and Ray, I.,** (2007). Measuring, analyzing and predicting security vulnerabilities in software systems. Computers & Security 26, 219–228. https://doi.org/10.1016/j.cose.2006.10.002

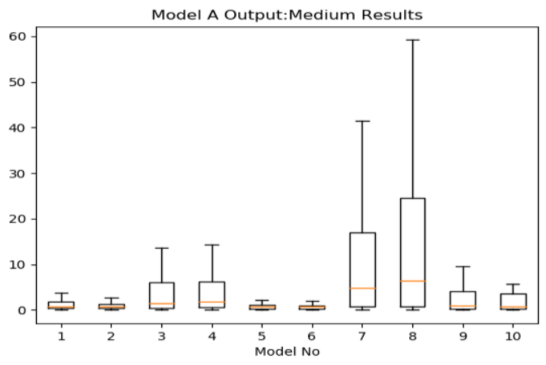
[49] **Solk**, **J.S**. (2008). "Text Data Mining: Theory and Methods" Vol. 2 (2008) 94–112, ISSN: 1935-7516, DOI: 10.1214/07-SS016

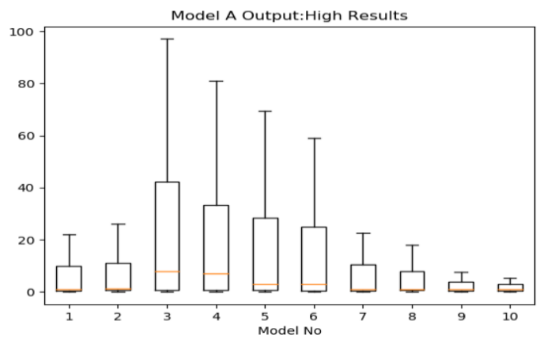
[50] **Blei, D.M., Ng, A.Y., and Jordan, M.I** (2003). "Latent Dirichlet Allocation" , Journal of Machine Learning Research 3 (2003) 993-1022

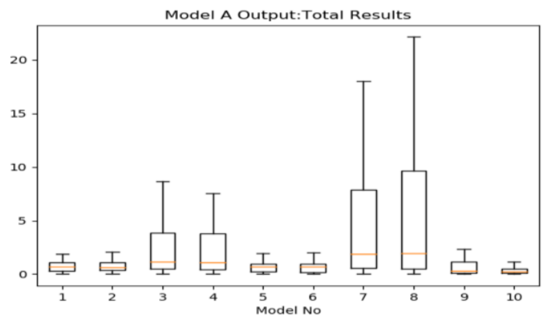
APPENDICES

APPENDIX A.1 : Graphical Represantation of Model A Results.

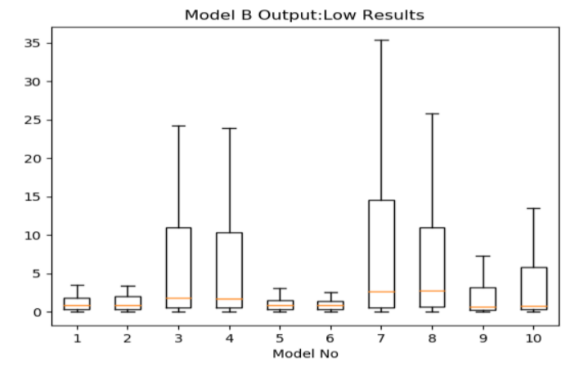


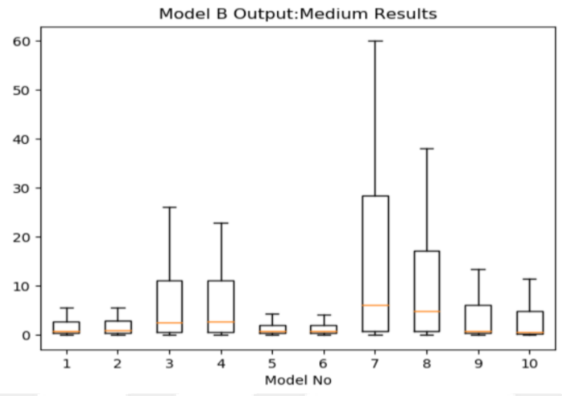


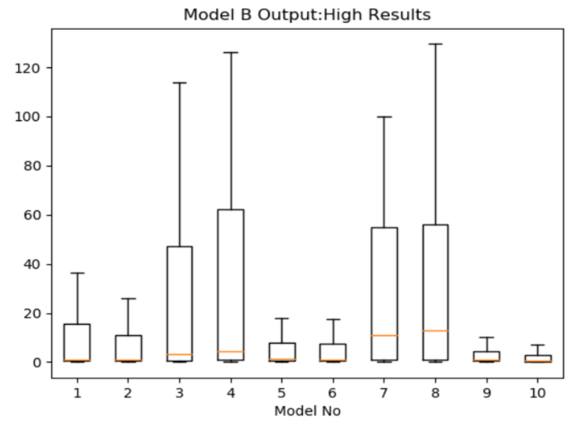


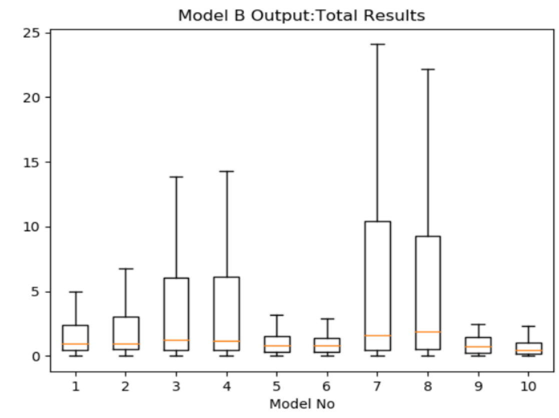


APPENDIX A.2 : Graphical Represantation of Model B Results.









CIRRICULUM VITAE

Name Surname : Fatma Gül BULUT

Place and Date of Birth : Bursa, 1991

E-Mail : fgulyavuz@gmail.com

EDUCATION:

- **B.Sc.:** Uludağ University, Electronic Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- May 2017 - *Present* - Software Engineer / Softtech
- November 2015 – May 2017 - Quality Management Engineer / Softtech
- March 2014 – October 2015 - Industrial Product Specialist / Tekno Tasarım

PUBLICATIONS/PRESENTATIONS ON THE THESIS

Bulut, F.G., Altunel, H., and Tosun, A. 2019: Predicting Software Vulnearabilities Using Topic Modeiling. *International Conference on Computer Science and Engineering - IEEE,* September 11-15, 2019 Samsun, Turkey – Summitted