2019

Security Bug Report Classification using Feature Selection, Clustering, and Deep Learning

Tanner D. Gantzer
West Virginia University, tdgantzer@mix.wvu.edu

2019

Security Bug Report Classification using Feature Selection, Clustering, and Deep Learning

Tanner D. Gantzer

Security Bug Report Classification using Feature Selection, Clustering, and Deep Learning

Tanner D. Gantzer

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Katerina Goseva-Popstojanova, Ph.D., Chair
Roy Nutter, Ph.D.
Matthew Valenti, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2019

Keywords: Cybersecurity, Bug Reports, Text Classification, Feature Selection, Deep Learning

Abstract

Security Bug Report Classification using Feature Selection, Clustering, and Deep Learning

Tanner D. Gantzer

As the numbers of software vulnerabilities and cybersecurity threats increase, it is becoming more difficult and time consuming to classify bug reports manually. This thesis is focused on exploring techniques that have potential to improve the performance of automated classification of software bug reports as security or non-security related. Using supervised learning, feature selection was used to engineer new feature vectors to be used in machine learning. Feature selection changes the vocabulary used by selecting words with the greatest impact on classification. Feature selection was able to increase the F-Score across the datasets by increasing the precision. We also explored unsupervised classification based on clustering. A distribution of software issues was created using variational autoencoders, where the majority of security related issues were closely related. However, a portion of non-security issues also ended up in the distribution. Furthermore, we explored recent advances in text mining classification based on deep learning. Specifically, we used recurrent networks for supervised and semi-supervised classification. LSTM networks outperformed the Naive Bayes classifier in projects with a high ratio of security related issues. Sequence autoencoders were trained on unlabeled data and tuned with labeled data. The results showed that using unlabeled software issues different from the testing datasets degraded the results. Sequence autoencoders may be used on large datasets, where labeled data is scarce.

Acknowledgments

I would like to thank my research advisor Dr. Katerina Goseva-Popstajanova for her generous support and guidance of my research. I would also like to thank the professors at West Virginia University who influenced me to pursue a master's degree including my committee members Dr. Roy Nutter and Dr. Matthew Valenti. Access to the three NASA projects was granted by the NASA Katherine Johnson Independent Verification and Validation Facility in Fairmont, WV. I would also like to thank fellow peer, Mohammad Ahmad, for his aid on acquiring the Red Hat Linux dataset used in this research. Furthermore, I would like to express my sincere gratitude to my friends and family for their continuous support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cybersecurity of software has become more relevant through the recent years. There has been an increased threat of cyber attacks as online infrastructure and commerce grow. The development of software now requires a large focus in keeping the software secure. Since software may never be fully secure, security must be continuously managed. Just one vulnerability may compromise the software leading to costly and harmful damage. The validation and verification stages of the software development cycle are critical in securing the software. However, testers may not keep security in mind when discovering new software issues. Many software development teams may find it important to focus on the security related software issues over other software issues.

Software faults have been studied in open source projects and across NASA projects [12], [13], [14], [38]. Software vulnerabilities have been analyzed across the NASA projects used in this thesis [10], [32]. A successful automated identification of security bug reports has been done on NASA projects through machine learning approaches [11]. Automated classification of security bug reports has also been attempted on other datasets [30], [9], [34].

This work focuses on automated classification of security bug reports through machine learning methods. It builds on the previous work [11] by exploring the effectiveness of several state-of-the-art text mining appraches.

1.1 Key Terms

A **bug report** is a description of a software related issue.

A **vulnerability** is a security weakness in software which can be exploited by an attacker.

CWE refers to the Common Weakness Enumeration community-developed list. The list acts as a common measuring tool of analyzing software security weaknesses [26] [27]. Specifically CWE-888 list that relates to software fault patterns is used.

CVE stands for Common Vulnerabilities and Exposures. CVE system provides identification numbers and descriptions for known vulnerabilities.

A **NN** (Neural Network) is a framework of different machine learning practices that work together to map a complex input to an output.

A **RNN** (Recurrent Neural Network) is a type of Neural Network that uses data in the format of a series to take into account the history and structure of the data.

LSTM (Long Short Term Memory) Neural Networks are a type of recurrent networks that focus on solving long term dependencies with the addition of gated inputs.

VAE (Variational Autoencoder) Networks learn an efficient compression of the input data.

1.2 Problem Statement

Software issues are often categorized by type and risk. Developers may be more concerned with specific types of software issues, such as security related software issues because they may be weaknesses or vulnerabilities that can be exploited by malicious attackers. The manual classification software issues is an expensive task as it requires time and domain expertise. This thesis has a goal of exploring methods to automatically classify software issues to security related and non-security related. Given a dataset of software issues, the methods explored are capable of classifying security related issues. The performance metrics of the methods are explored and vary with different datasets. The developed machine learning implementation used to test the methods has the input of textual fields of software issues in a comma-separated values file and classifies each issue as security or non-security related.

Our machine learning implementation may be altered to accept different datasets or used to train other supervised learners.

1.3 Research Questions and Contributions

The research questions explore potential improvements in security bug report classification. Research Question 1 relates to improving the features used in text mining. Research Questions 2 and 3 focus on learning algorithms used in text mining.

RQ 1: What feature selection methods show the greatest improvement of the automated bug report classification?

RQ 2: Can unsupervised classification based on clustering outperform the anomaly detection based on cosine similarity?

(a) How do Term Frequency feature vectors perform in combination with K-Means clustering?

(b) Can Variational Autoencoders be used to represent security related software issues in a distribution of software bug reports?

RQ 3: Can deep learning outperform traditional supervised machine learning techniques?

(a) How do supervised deep neural networks compare to traditional supervised machine learning techniques?

(b) How do semi-supervised deep neural networks compare to supervised learning techniques?

The main contributions of this thesis are as follows:

We used the work done by Goseva and Tyo [11] as a baseline for software bug report classification. One goal was to improve the precision by lowering the false positive rate. Analyzing the projects showed that security related words were present in non-security bug reports, which may have caused a high false positive rate. Feature selection was used to attempt to remove these words, as well as to reduce the dimensionality of the feature vectors.

We also explored a clustering approach using variational autoencoders. Lastly, we used LSTM network classification and compared the results to the classification based on Naive Bayes algorithm.

An overview of the contributions of the thesis are shown in Figure 1.1. The darker blue blocks represent new contributions, while the yellow blocks represent a similar step used in the related work [11]. This thesis explores one way to feature engineer the feature vectors used in machine learning, and two methods to apply deep learning solutions to unsupervised and supervised classification.

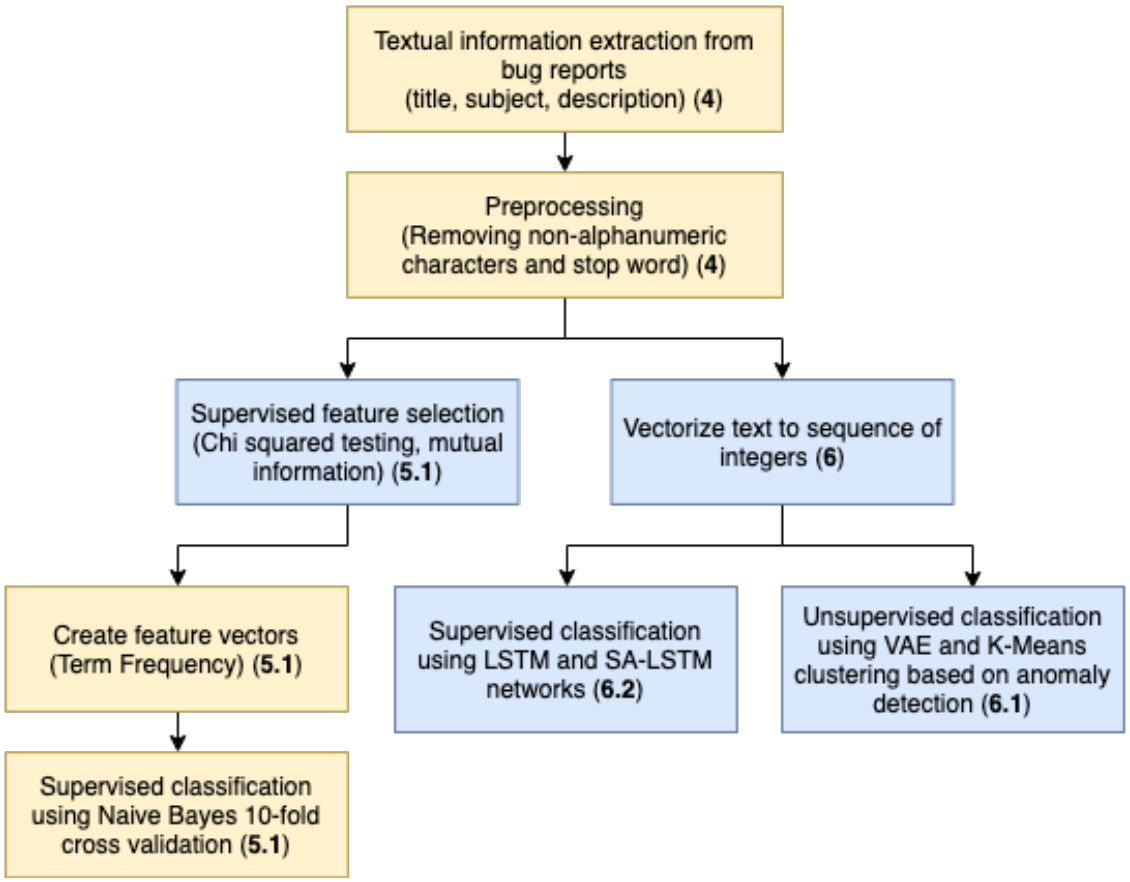

Figure 1.1: Thesis Contribution Diagram

1. Feature selection to improve the vocabulary of the features used in text mining.

   We expanded the vocabulary of the feature vectors consisting of CWE-888 descriptions with the descriptions of software issues. Then, we used feature selection with chi-squared or mutual information dependency tests to remove words in which classifying

an issue did not depend heavily on. The added vocabulary of the dataset was used to improve the classification of non-security software issues. Feature selection weeded out words from the combined vocabulary. Some related works used feature selection to improve bug report classification. Chi-squared and info gain feature selection were used to rank features for use in classifying configuration bug reports [36]. Feature selection was also used for dimensionality reduction to categorize bug reports based on CWE standards [34]. Unlike related works, this work focuses on improving the vocabulary of the feature vectors to classify software bug reports to security related and non-security related.

2. Unsupervised classification through clustering.

   We explored the use of clustering for unsupervised learning and compared the performance with similar work done on images and textual reviews [6], [37]. PCA analysis and VAE networks were used to visualize our data in a two dimensional space, while K-means clustering was used to predict the data class.

3. Security bug report classification using deep learning.

   We experimented with LSTM neural networks to explore if the use of deep learning methods can benefit the supervised classification problem. The LSTM network does not use document vectors such as Term Frequency, but instead use tokenized sequences that are embedded into word vectors. The word vectors attempt to relate together words that are similar in natural language. LSTM networks also account for the word order in software issues.

4. Semi-supervised classification with LSTM sequence autoencoders.

   We attempt a semi-supervised learning technique using sequence autoencoders to take advantage of the addition of unlabeled data. Similar work was done by Dai et. al. for classification of movie reviews [5]. A LSTM autoencoder network was used to pre-train the weights of a similar LSTM neural network using unlabeled data. The pre-trained weights were sent to a LSTM network. The pre-trained LSTM network became more stable and less volatile.

1.4 Organization of the Thesis

The organization of the thesis is as follows. Chapter 2 surveys the related work to classifying bug reports. Chapter 2 also reviews methods in text mining used in text representation and text classification. In Chapter 3 we delve deeper into the motivation behind the work. We also discuss the work upon which this thesis builds on. We discuss the baseline used to compare the results. We also discuss the performance metrics used to evaluate our findings. Chapter 4 is an overview of the three NASA datasets and the one Red Hat Enterprise Linux 4 dataset used. We also discuss the data preprocessing steps used on the datasets. In Chapter 5 we explore feature selection using chi-squared and mutual information tests. Feature selection is a supervised method to improve the vocabulary of feature vectors. Naive Bayes was used in Chapter 5 to classify security bug reports. In Chapter 6 we explore clustering to be used in unsupervised classification. Data is represented as a term frequency vector and a variational autoencoder representation. We classify security bug reports with K-means clustering. In Chapter 7 we explore both supervised and semi-supervised learning with LSTM networks. Sequence autoencoders are firstly trained on unlabeled data and tuned with labeled data. Chapter 8 gives a summary of this thesis including the methods used and overall results.

Chapter 2

Related Work

Related works were surveyed on bug report classification findings and text classification methods. The bug report section surveys works relevant to classifying security bug reports. The text classification section surveys state-of-the-art methods through text classification.

2.1 Related Work on Bug Report Classification

Not all works were focused on security and non-security bug reports. The work done by Xia et al. focused on classifying configuration bug reports. The remaining works related to security bug reports. The works used different approaches to classify bug reports both through improving the feature vectors and the algorithms of machine learning.

Gegick et al. developed an approach to label bug reports as security related or non security related [9]. Firstly, they developed three lists: start, stop, and synonym lists. A start list was created from the vocabulary of the bug issues in which individual words are considered to be related to security issues. A stop list was created from words that are not believed to aid in to the classification process such as articles, prepositions, and conjunctions. The third list was a synonym list that relates a word used less frequently to a word used more frequently, giving it a higher frequency score. Singular value decomposition was used to reduce the size of the feature vector matrix. The approach used SAS Text Miner to construct a term-by-document matrix and for training the predictive model. The model identified 77% of security related reports in a Cisco project that were mislabeled as non security related.

However, the model resulted in high false positive scores.

Zaman et al. analyzed security the differences between security bug reports and performance bug reports in a Firefox dataset [38]. Security bugs should be fixed faster because of the critical aspect of the vulnerabilities they may cause. However, security bugs require more expertise and knowledge of the overall software to fix. More individuals are often involved to fix security software issues, further complicating the problem.

Xia et al. classified bug reports as configuration bugs using supervised learning methods such as Naive Bayes [36]. Feature selection methods such as chi-squared testing and information gain were used on the feature vectors. The results showed that Naive Bayes Multinomial with information gain feature selection performed the best on average throughout the datasets.

Wen et al. explored improving vulnerability classification through feature extraction [34]. Feature extraction was done using Chi-squared testing. Information gain was not suited for the application as the vulnerability numbers were different for each category.

Peters et al. proposed an approach to classify security bug reports through filtering and ranking [30]. The proposed framework *FARSEC* is based on a combination of filtering and ranking methods with a goal to reduce mislabelling of security bug reports. Security related keywords are scored based on the keyword's frequency in both security and non-security reports. Non-security bug reports with scores similar to security bug reports are removed based on the keyword scores. Using the prediction model, FARSEC returns ranked lists of bug reports where most of the actual security bug reports are at the top of the list. FARSEC attempts to solve the class inbalance issue. The proposed method reports a reduction in the number of mislabeled issues by 38%. The precision metric varied project to project. The highest precision was 50% with a recall of 16.7%. The highest recall throughout the tested project was 50% with a precision of 5%.

Goseva et al. explored the automated identification of security bug reports [11]. This thesis uses the work done as a baseline to build on. The related work uses machine learning methods such as Naive Bayes, SVM, and Random Forest with term frequency feature vectors. We improved the feature vectors with feature selection on an expanded vocabulary. The unsupervised approach used anomaly detection with distance based similarity

scores. We explored clustering based on unsupervised learning. We also used supervised and unsupervised deep neural networks for classification of security bug reports.

Poddar et al. designed a loss function to be used in a neural network architecture to detect duplicate bug reports and clustered reports into latent topics [31]. The classification approach was based on supervised learning. The clustering approach learned meaningful latent clusters without additional supervision. The purpose was to annotate user submitted reports.

2.2 Related Work on Text Classification

This section surveys work relating to general text based classification. Many previous machine learning approaches used document term matrices or feature vectors to represent the text data. The sentence representation section reviews new ways to represent text data to capture semantic information. Related works focus on classification through deep neural networks such as LSTM networks and variational autoencoders.

2.2.1 Sentence Representations

Variational Autoencoders (VAEs) have been used to perform semi-supervised classification problems. A VAE network is similar to an autoencoder network where the output is the same as the input. However, in a VAE, a constraint is added that forces it to generate latent vectors that follow a unit Gaussian distribution. Kingma et al. first proposed Variational Autoencoders in 2013 [19].

Mikolov et al. proposed two architectures for computing continuous vector representations based on simple neural network architectures [24]. The first model proposed was the Continuous Bag-of-Words Model (**CBOW**) which was not dependent on word order. The model was trained to predict a word, while the inputs are the words before and after the target word. A projection layer is shared for all words in order to average all the word vectors. The second model proposed was the **Continuous Skip-gram** model that was dependent on word order. The model took one word as input and was trained to predict the neighboring words. A window size was given and as the window size was increased, the quality of the

word vectors increased, but also increased computational complexity. The models were able to be trained on very large datasets with one trillion words for a practically unlimited size of vocabulary. The vector was set up in a way that similar words are similar in distance. For instance a well trained vector *X = vector("biggest") - vector("big") + vector("small")* will return *"smallest"*. The proposed methods are now commonly referred to as *word2vec*. The word vectors are commonly used as a building block in many natural language processing applications.

Le and Mikolov proposed an unsupervised framework that learns continuous distributed vector representations of text called **Paragraph Vectors** [21]. The representations are dependent to specific phrases as opposed to individual words. Two versions were experimented with: Distributed Bag of Words version of Paragraph Vectors (PV-DBOW) and Distributed Memory version of Paragraph Vectors (PV-DM). PV-DM has a similar implementation as learning word vectors from Mikolov [24]. The model takes in the order of words to predict a future word; however, the model also includes a paragraph ID token. The paragraph token may be thought of as another word that acts as memory, which remembers what is missing from the current context. In PV-CBOW, the model does not care about the order of the words. The PV-CBOW model was given a paragraph ID token and trained on randomly selected neighboring words at each iteration of stochastic gradient descent. The features from the vectors can be fed directly to conventional classifiers such as logistic regression, support vector machines, or K-means. The proposed method outperformed baseline methods such as bag-of-words models and Restricted Boltzmann Machines models.

Bahdanau et al. proposed that the use of a fixed-length vector is a bottleneck in improving performance in encoder-decoder architectures [2]. The proposed technique allowed a model to automatically (soft-)search for parts of sentences that are relevant in predicting the target. The proposed method expanded on recurrent neural network Encoder-Decoder frameworks, by instead using a bidirectional recurrent neural network. The reasoning was that in a conditional RNN the annotation of each word summarizes the word and the preceding words, and in a bidirectional recurrent neural network the annotation will summarize the preceding words and the future words. The researches tested their architecture on English-French sentence translation, and the searching recurrent neural network outperformed the

traditional encoder-decoder recurrent neural network.

Kiros et al. proposed **Skip-Thought Vectors** for unsupervised learning of distributed sentence representations [20]. Skip-Thought Vectors are trained to predict sentences S_{i-1} and S_{i+1}, given the source sentence S_i. The sequence to sequence model is represented as an autoencoder. The encoder architecture encodes the sequence. Then two decoder architectures decode the two sentences. The model uses Recurrent Neural Network architecture, specifically with Gated Recurrent Units. The sentence representations from the encoder may be used as a replacement to word vectors. The encoder's vocabulary may be expanded to words it has never seen before. Using a very large word representation, such as word2vec pretrained model, the larger word representation can be mapped into the word representation of the encoder using un-regularized L2 linear regression loss. In the experiment, the Skip-Thought models were trained with a vocabulary size of 20,000 words; after vocabulary expansion the models can successfully encode 930,911 words. The proposed method differs from other methods as it takes context from outside of just one sentence.

Hill et al. compared models that learn distributed sentence representations from unlabeled data [15]. Hill noted that deeper models are preferable for representations used in supervised systems, but shallow log-linear models work best in systems with spatial distance metrics. Hill proposed two new approaches based on previous models, **Sequential (Denoising) Autoencoders** and **FastSent**.

2.2.2 Text Classification

Dai et al. explored at two approaches for pretraining unlabeled data [5]. The main idea was to use a sequence autoencoder with unlabeled data. Sequence autoencoder LSTM networks (**SA-LSTM**) consisted of a recurrent network to decode the input sequence into a hidden state, then use a recurrent network with the same weights to decode the output. The network may be fed large amounts of unlabeled data to improve generalizability. The weights obtained from the sequence autoencoder are able to be used as an initialization of a supervised network. Dai et al. also experiment with using language models (**LM-LSTM**), which had the same approach as sequence models but cuts off the encoder portion of the model. This

allows a deep representation of input data. The method was able to match or surpass reported results for multiple datasets. One of the experiments tested sentiment analysis on movie reviews. The SA-LSTM networks trained on unlabled data from Amazon or IMDB reviews outperformed the LSTM trained on only labeled Rotten Tomatoes reviews. Another experiment done to categorize Wikipedia articles showed SA-LSTM networks outperformed normal LSTM networks as well as the state of the art methods such as Naive Bayes, SVM, and ConvNet.

Dilokthanakul et al. use Gaussian mixture variational autoencoders for unsupervised clustering [6]. The experiment was done on the MNIST dataset (0-9 handwritten digits) and showed same digits clustered together. Although the handwritten digits were images and not text data, a similar approach may be used on textual data.

Yang et al. used improved variational autoencoders with LSTM and convolutional layers for semi-supervised and unsupervised learning [37]. Experiments were done using improved variational autoencoders as language models. The models competed with LSTM language models. Unsupervised classification was performed on a dataset of food reviews from Yelp. The dimensionality was first reduced by running PCA on the features.

Chapter 3

Background

Computer security may be managed throughout the software development cycle. During the software development cycle developers focus on what the software needs to do and what the software's resource needs are such as memory and computational power. Software security may fall out of focus during development; however, this should not be the case. Many new technologies are influencing the world around us. This is great for the quality of life, although it may open new opportunities for cybercriminals to take advantage.

Software security should be an important step in the software development cycle. Software bug reports may be used as a tool for detecting software issues and vulnerabilities. Depending on the how new software issues are discovered, it may be difficult to determine which software issues are security related. Fixing security related issues may be a priority across products to protect the users.

This work builds upon previous work done at West Virginia University using NASA mission data [11]. Specifically, the work used both supervised and unsupervised approaches to classify security bug reports in three NASA data sets. The three NASA datasets (Ground Mission IV&V Issues, Flight Mission IV&V Issues, Flight Mission Developer Issues) are further detailed in Section 4.1. The datasets included descriptions of software issues, but did not include the security status. The CWE-888 list of software weaknesses was used to categorize software issues [27]. The Common Weakness Enumeration (CWE) lists contain common software security weaknesses to be used as a baseline for evaluating software issues [26]. Hand labeling decided if an issue was related to CWE-888 descriptions, and then it

was considered to be a security risk.

The classification was based on using three different feature vectors: Binary Bag of Words Frequency (BF), Term Frequency (TF), and Term Frequency-Inverse Document Frequency (TF-IDF). Text preprocessing was performed prior extracting feature vectors. The text preprocessing included forcing all text to lower case, stemming words, and removing stop words. Both unsupervised learning and supervised classification were performed on the same datasets, using the same feature vectors.

Text mining has the increased difficulty of capturing natural language information when compared to other data mining works. The standard prior to neural networks has been to use feature vectors based on word occurrence, such as binary bag of words models and term frequency models. However, these models do not capture the semantic importance of word ordering and natural language information. Recently, deep learning approaches have attempted to extract intricate semantic representations of text [24], [21], [20].

The unsupervised learning method was based on anomaly detection. Anomaly detection has the benefit where only one class needs to be known. The details of the experiment are given in Section 3.1. Supervised learning has the advantage of learning from issues given a label. Supervised learning will generally have higher classification performance metrics compared to unsupervised learning. This requires the training data to be prelabeled which costs many resources one may not afford. The supervised approach method is detailed in Section 3.2.

3.1 Unsupervised Learning

Unsupervised learning was used with anomaly detection to differentiate issues from what appears to be normal. Anomaly detection works with security classification as the security class will be much smaller than the non-security class. Therefore, a known group of security issues can be used where any issue similar to that list will be classified as security related. In prior work [11], the CWE-888 list relating to software fault patterns was used as the normal data. Cosine similarity distance measures scored software issues based on the similarity to CWE-888 descriptions. The issues with similarity scores above a calculated threshold were

classified as security related. The threshold was calculated from an average of scores on a validation test set. Euclidean distance measures were also explored; however, the calculated thresholds were not as stable as in cosine similarity.

Prior unsupervised classification results are shown in Table 3.1. The methods with the highest G-Score value are in bold.

Table 3.1: Classification Performance of Unsupervised Learning Baseline using Cosine Similarity [11]

Dataset	*Ground Mission IV&V Issues*			*Flight Mission IV&V Issues*			*Flight Mission Developer Issues*		
Feature vector	*BF*	*TF*	***TFIDF***	*BF*	***TF***	*TFIDF*	*BF*	***TF***	*TFIDF*
Threshold	0.305	0.286	**0.263**	0.283	**0.216**	0.235	0.321	**0.26**	0.22
Accuracy	62.5%	64.3%	**73.0%**	64.9%	**67.8%**	49.2%	50.4%	**55.4%**	51.7%
Precision	12.6%	15.0%	**17.7%**	57.5%	**58.1%**	41.2%	70.2%	**69.3%**	65.9%
Recall	66.2%	78.7%	**69.9%**	56.1%	**77.7%**	55.4%	42.7%	**57.9%**	55.1%
PFA	37.8%	36.9%	**26.7%**	28.9%	**39.1%**	55.1%	34.8%	**49.4%**	54.9%
F-Score	21.2%	25.2%	**28.3%**	56.8%	**66.5%**	47.3%	51.6%	**63.1%**	60.0%
G-Score	64.1%	70.0%	**71.5%**	62.7%	**68.3%**	49.6%	53.1%	**54.0%**	49.6%

3.2 Supervised Learning

Using a supervised approach, several learners were tested. Naive Bayes, a probabilistic learner, performed consistently well across the projects. Other supervised learners included Bayesian Network (BN), k-Nearest-Neighbors (kNN), Random Forest (RF), and Support Vector Machines (SVM). The results varied depending on the feature vectors used and the specific dataset. Overall Term Frequency showed the best performance with respect to the G-Score. However, some learners performed better with specific feature vectors. For instance, Random Forest had better results using a TF-IDF vector compared to Term Frequency vector. SVM had considerably worse results with a TF-IDF vector compared to using Term Frequency. The results from Term Frequency will be used as the baseline for supervised learning.

Baseline supervised classification, with Term Frequency feature vectors, results are shown in Table 3.2. The methods with the highest G-Score are in bold.

Table 3.2: Classification Performance of Supervised Learning Baseline [11]

	Supervised System	*TF_BN*	*TF_kNN*	*TF_NB*	*TF_NBM*	*TF_RF*	*TF_SVM*
Ground	*Accuracy*	87.4%	93.5%	85.2%	**87.9%**	94.9%	94.1%
Mission	*Precision*	37.1%	57.3%	32.0%	**38.0%**	82.6%	66.0%
IV&V	*Recall*	93.4%	60.3%	83.1%	**93.4%**	41.9%	47.1%
Issues	*PFA*	13.1%	3.7%	14.6%	**12.6%**	0.7%	2.0%
	F-Score	53.1%	58.8%	46.2%	**54.0%**	55.6%	54.9%
	G-Score	90.0%	74.2%	84.2%	**90.3%**	58.9%	63.6%
	Supervised System	*TF_BN*	*TF_kNN*	*TF_NB*	*TF_NBM*	*TF_RF*	*TF_SVM*
Flight	*Accuracy*	69.6%	70.9%	75.1%	78.3%	80.4%	**83.8%**
Mission	*Precision*	57.9%	60.3%	67.8%	67.8%	75.9%	**78.8%**
IV&V	*Recall*	95.5%	86.0%	75.2%	89.8%	76.4%	**82.8%**
Issues	*PFA*	48.4%	39.6%	24.9%	29.8%	16.9%	**15.6%**
	F-Score	72.1%	70.9%	71.3%	77.3%	76.2%	**80.7%**
	G-Score	67.0%	71.0%	75.1%	78.8%	79.6%	**83.6%**
	Supervised System	*TF_BN*	*TF_kNN*	*TF_NB*	*TF_NBM*	*TF_RF*	*TF_SVM*
Flight	*Accuracy*	65.8%	61.0%	66.9%	70.6%	70.4%	**72.3%**
Mission	*Precision*	65.8%	71.7%	75.0%	73.6%	71.0%	**76.8%**
Developer	*Recall*	100.0%	67.2%	74.6%	86.4%	93.2%	**83.1%**
Issues	*PFA*	100.0%	51.1%	47.8%	59.8%	73.4%	**48.4%**
	F-Score	79.4%	69.4%	74.8%	79.5%	80.6%	**79.8%**
	G-Score	0.0%	56.6%	61.4%	54.9%	41.4%	**63.7%**

In general results show good recall scores, meaning fewer security related bug reports are missed. However, the precision values were rather low. A low precision means non-security related bug reports are classified as security related. We believe there is room for improvement in the overall performance of the automated classification. Therefore, this thesis explores several approaches aimed at improving classification performance.

We develop new implementations of the prior work to compare as a baseline. However, we do not use word stemming on the the feature vectors. We also use a vocabulary size of 450 words based on the document frequency. The new baseline using Naive Bayes is shown in Table 3.3. The differences are small; however, the recall degraded in the Flight Developer Mission project. Results are compared to the prior work labeled as "*Previous Baseline*" [11].

Table 3.3: Baseline Supervised Classification with Term Frequency Performance (Naive Bayes)

Project	*Method*	*Accuracy*	*Precision*	*Recall*	*PFA*	*F-Score*	*G-Score*
Ground Mission IV&V Issues	*Previous Baseline*	85.2%	32.0%	83.1%	14.6%	46.2%	84.2%
	New Baseline	84.7%	31.8%	84.2%	14.3%	46.2%	84.4%
Flight Mission IV&V Issues	*Previous Baseline*	75.1%	67.8%	75.2%	24.9%	71.3%	75.1%
	New Baseline	78.1%	71.8%	77.2%	21.3%	74.4%	77.9%
Flight Mission Developer Issues	*Previous Baseline*	66.9%	75.0%	74.6%	47.8%	74.8%	61.4%
	New Baseline	58.7%	74.2%	57.0%	38.9%	64.4%	59.4%

3.3 Metrics used for Performance Evaluation

During classification, performance is measured by true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). True positives are security related issues correctly classified as security related. True negatives are non-security related issues correctly classified as non-security. False positives are non-security related issues wrongly classified as security related. False negatives are the missed security related issues classified as non-security issues.

Accuracy describes the total amount of correctly classified issues with respect to the total number of issues. Accuracy may give an unrealistic representation when used on an unbalanced data set. For example, in a case where only 10% of the data set is security related, a learner may declare all issues as non security related and show an accuracy of 90%, when in fact it has a poor performance.

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Precision accounts for the fraction of relevant issues among the issues retrieved as positive. As seen in Equation 3.2, precision is the fraction of correctly classified security issues over the total number of issues classified as security related. The metric tells us the percentage of issues classified as security related that are actually security related.

$$\textbf{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

Recall shows the sensitivity to classify an issue as security related. It represents the percentage of security related issues that were correctly classified as security related. A high recall score is important in critical scenarios. Recall and precision are usually contending with one another, where if one goes up the other typically goes down.

$$\textbf{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

The probability of false alarm is the probability of a non-security related issue being labeled as security related. In security critical scenarios, the probability of false alarm will tend to be high. Using a weaker threshold on declaring issues as security related allows fewer security issues to be missed.

$$\textbf{PFA} = \frac{FP}{TN + FP} \tag{3.4}$$

F-Score is the harmonic mean of the precision and recall. The F-Score represents a balance in the performance between precision and recall. Harmonic means have high values when both values have high values. Harmonic mean is typically used when comparing values where as one value increases, the other value may decrease.

$$\textbf{F-Score} = 2 \cdot \frac{\textbf{Precision} \cdot \textbf{Recall}}{\textbf{Precision} + \textbf{Recall}} \tag{3.5}$$

G-Score is the harmonic mean of recall and (1-PFA). A high G-Score represents a better performing classifier. The experiment is focused on detecting security related issues without over classifying all issues as security related, therefore G-Score will be an important representation of a classifier's performance.

$$\textbf{G-Score} = 2 \cdot \frac{\textbf{Recall} \cdot (1 - \textbf{PFA})}{\textbf{Recall} + (1 - \textbf{PFA})} \tag{3.6}$$

Chapter 4

Data-set collection and Feature extraction

The datasets used throughout this thesis include three NASA projects and one open source Red Hat Linux Enterprise 4 Bugzilla dataset. Two datasets are from NASA IV&V analysis of a ground mission and a flight mission. The third dataset comes from the developers of the flight mission. The three NASA datasets were used for our baseline. We were given access to the NASA datasets by the NASA Katherine Johnson Independent Verification and Validation Facility in Fairmont, WV. The fourth Red Hat Linux Enterprise 4 dataset was added to explore the generalizability to the proposed solutions. Access to the Red Hat Enterprise Linux 4 dataset was made possible by fellow peer Mohammad Ahmad.

4.1 Datasets

A summary of the datasets is shown in Table 4.1.

Table 4.1: Information About the Four Datasets

Mission	Total # closed bug reports	Security bug reports	Dataset
Ground	1,779	133	Ground mission IV&V
Flight	383	157	Flight mission IV&V
	513	374	Flight mission Developers
Red Hat	11,145	445	Red Hat Enterprise Linux 4

The Ground Mission IV&V Issues dataset includes 1,779 issues. After labeling, 133 (7.5%) were determined to be security related. However, 277 more issues were found to be related to a related security issue of the testing systems. Since the CWE's do not cover testing systems, these 277 issues were not considered to be security related. During the design of the mission, 127 out of the 133 security related issues come from the code phase of the mission. In the mission, 53% of the labeled CWE's were primary classes of "Memory Access". The other dominant classes include "Unused Entities", "Exception Management", "Risky Values", and "Resource Management".

The Flight Mission IV&V Issues dataset includes 157 (41%) security related issues out of the 383 software issues.The dominant CWE classes were "Other", "Risky Values", "Memory Access", and "Unused Entities".

The Flight Mission Developer Issues dataset has the highest ratio of security software issues over total software issues. Out of the 513 software issues, 374 (73%) issues are security related. The dominant CWE classes in the dataset are "Risky Values", "Exception Management", "Memory Access", "Channel", and "Unused Entities". The Developer Flight dataset differentiates from the others as most issues come from the developers during development compared to during testing.

The Red Hat Enterprise Linux 4 dataset includes 11,145 software issues. A software

issue was labeled as security related if the description included CVE tag. The CVE list is a dictionary containing identifiers and descriptions for specific vulnerabilities [25]. While each CVE describes one specific vulnerabilty, it may be mapped to a CWE identifier. A total of 445 (4%) issues were labeled as security related. The dataset was collected from the Bugzilla website and is open source.

4.2 Preprocessing and Feature Extraction

The following fields were extracted from the software issues: title, subject, description. The fields were then concatenated together to form a complete description. Non-alphanumeric characters were removed. The words were converted to lowercase to lower the amount of varying words. Stop words were removed using python's Natural Language Toolkit English stop word list [28]. Word stemming was not used so word embeddings may be used in latter sections. Each issue has a corresponding label stored independently, 1 for security related and -1 for non security related.

Stop word lists were used to remove common words that are believed to not affect the learning process. Word stemming may be used in text classification processes; however, we do not use it. Word stemming forces words to their most reduced root. This relates the versions of past, present, future, plural, and others all to one meaning. Word stemming is useful for forcing all the tenses of a word to be the same, so the learner does not differentiate the words. Word stemming may be useful using the document term feature vectors; however, it may confuse the word embeddings used in Section 7. Word embeddings keep similar words close in a vector representation. All tenses of a word will be closely related in a well trained word embedding.

Binary Bag-of-Words Frequency (BF), Term-Frequency (TF), and Term Frequency-Inverse Document Frequency (TF-IDF) were used as the feature vectors in the previous work [11]. We limit the experiment to using Term Frequency feature vectors. The feature vectors contains an index for every word in the vocabulary. Each software issue is represented by the occurrences of vocabulary words. The values inside the feature vectors are based on the overall frequency of each word. We used term frequency feature vectors in Section 5 and

Section 6.

Software issues may also be tokenized and represented by a sequence of integers. Each integer represents the index of a vocabulary word. A dictionary maps all words to their indexes. Sequences of integers are a preferred input in neural networks to keep the word order and reduce dimensionality. Sequences of integers are used in Section 6 for the input of variational autoencoders and in Section 7.

Chapter 5

Feature Selection

An important step in text mining is the feature extraction process. Feature extraction transforms the text into features that are used by the learning algorithms. The standard examples include document-term matrices. Feature vectors use a vocabulary, usually the vocabulary of the dataset, where each feature is a word in the vocabulary. Depending on the type of document-term matrix, the documents are represented by the presence of features, frequency of features, or some type of manipulation of features. The more words in a vocabulary may lead to more patterns to be learned, leading to higher performance. However, a bigger vocabulary complicates the problem and can make using different learners more difficult.

In this chapter we address Research Question 1.

RQ 1: What feature selection methods show the greatest improvement of the automated bug report classification?

5.1 Background on Feature Selection

The previous work used the vocabulary of CWE-888 to capture security related features [11]. In some projects this leads to a low precision values, meaning a high number of false positives because much of the vocabulary was security related. In this thesis the new approach used the vocabulary of both the CWE-888 weaknesses as well as the vocabulary of a bug report dataset.

The words in the vocabulary were then shrunk down for two reasons. One reason was that there are words that do not contribute to the classification as they are independent to relating issues as security or non security. A smaller vocabulary also allows for more efficient learning across different algorithms. Some algorithms such as Random Forest require large amounts of memory, which grows as the vocabulary size grows. Another reason was that within certain datasets, some words may confuse the learner. As the vocabulary is shrunk, there is a potential to remove the confusing words.

Typically seven feature selection models used in text mining; Document Frequency, Chi-Square, Information Gain, Consistency, Filter, Gain Ratio, and Cfs [23]. In this thesis we focus on Chi-Square and Information Gain feature selection. Document frequency may be used when generating feature vectors. Document frequency is dependent on the feature vector where words can be given a cutoff point if they are not frequent in the documents. Chi-Square is a statistical measure of dependence between stochastic variables. For every word, one can measure the dependence that word has on the two classes using Equation 5.1 [23]. In the equation, O is the observed frequency and E is the expected frequency. The words with high independence are removed.

$$\mathcal{X}^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{5.1}$$

Information Gain measures the dependency between to variables through entropy. Entropy is given by Equation 5.2 [23]. The probability is associated with the frequency of category j at node t. INFO in Equation 5.3 splits t into k partitions where n_i is the number of records in the partition [23].

$$INFO_{Entropy(t)} = -\sum_{i=1} p(\frac{j}{t}) log_2 p(\frac{j}{t}) \tag{5.2}$$

$$Gain = Entropy(t) - \left(\sum_{i=1}^{k} (\frac{n_i}{n}) Entropy(i) \right) \tag{5.3}$$

In related works, Xia et al used chi-squared and information gain feature selection on feature vectors to classifiy configuration bug reports [36]. Wen et al. used Chi-squared

testing for dimensionality reduction on a bug report classification approach to classify bug reports based on CWE descriptions [34].

5.2 Feature Selection Approach

The process of feature selection was done using Sci-kit Lean's implementation of *SelectKBest*, *chi2*, and *mutual_info_classif* [29]. Mutual information is an implementation of info gain that compares two variables. A supervised process was necessary to select the features used in the vocabulary. A validation test set of the dataset was created. We used a validation set of 20% of the starting dataset. 10-fold cross validation was used on the remaining dataset for training and testing. We first expanded the vocabulary from the CWE-888 vocabulary with the addition of the vocabulary of the validation test set.

The size of the vocabulary of machine learning will affect the overall performance across machine learning. A smaller sized vocabulary may not have enough information to recognize patterns across software issues. A large vocabulary may not be necessary as many words are only found in few software issues and do not contribute to connecting software issues as security related. The most effective vocabulary size will depend on the dataset and the size of the dataset. A larger dataset will have more words that are common enough across the issues to affect the machine learning. Through several test experiments, a vocabulary of 250 words was chosen to attempt to balance between a vocabulary that was too small, and a vocabulary that did not differentiate from the complete vocabulary.

After a vocabulary was created from feature selection, the feature vectors used in the previous work may be created [11]. Term Frequency (TF) feature vectors are created using the new vocabulary. Supervised learning was then applied to the feature vectors. Different supervised learning classifiers may be used on the feature vectors. Naive Bayes was chosen because it performed well across different feature vectors and datasets. The open source machine learning tool Weka was used for the Naive Bayes implementation [35]. No special parameters differed from the default implementation. Preprocessing and feature selection code was written using Python [33] and Sci-Kit Learn [29] [3].

A new baseline was calculated using the top 450 words in the CWE-888 vocabulary based

on Document Frequency. A vocabulary of size 450 words was selected to be large enough to be similar to the full vocabulary, but small enough so it was computationally feasible. The baseline was slightly different from the other experiments as a validation set was necessary. Chi-Squared and Mutual Information Feature Selection was performed to keep only 250 words from the CWE-888 vocabulary and the vocabulary of the bug reports. These words have the highest dependence on if an issue is security related or not.

5.3 Results

The results are compared to the baseline "450 CWE", where the feature vector's vocabulary is from just the CWE-888 list. The results for "Chi 250" represent feature selection using Chi-squared testing and clipping till 250 of the most influencing words remained. The results for "MI 250" represent feature selection using mutual information. Results for each dataset are in tables containing the baseline, chi-squared feature selection, and mutual information feature selection. Results with the highest G-Score are in bold.

It was observed feature selection increased the precision value across all projects. Feature selection also increased the F-Score and G-Score across all projects besides the Flight Mission Developer Issues dataset. The dataset is relatively small and the baseline precision value was relatively high to begin with.

5.3.1 Red Hat Enterprise 4 Results

The results for feature selection on the Red Hat Linux 4 Dataset are discussed next. The experimental results are shown in Table 5.1. The baseline classification performance on the Red Hat resulted in a very low precision metric.

The only metric that degraded from feature selection was recall. However, precision was still low and may be due to the dataset itself. Both chi-squared and mutual information feature selection improved the G-Score and precision. Precision improved by 79% while only degrading recall by 8% using chi-squared feature selection. A box plot comparing the results is shown in Figure 5.1.

Table 5.1: Red Hat Enterprise Linux 4 Term Frequency Performance (Naive Bayes)

Feature Vector	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
CWE 450	69.5%	10.6%	90.3%	68.7%	18.9%	78.0%
Chi 250	**85.3%**	**18.9%**	**83.2%**	**85.4%**	**30.8%**	**84.3%**
MI 250	84.5%	17.2%	76.9%	84.8%	28.1%	80.7%

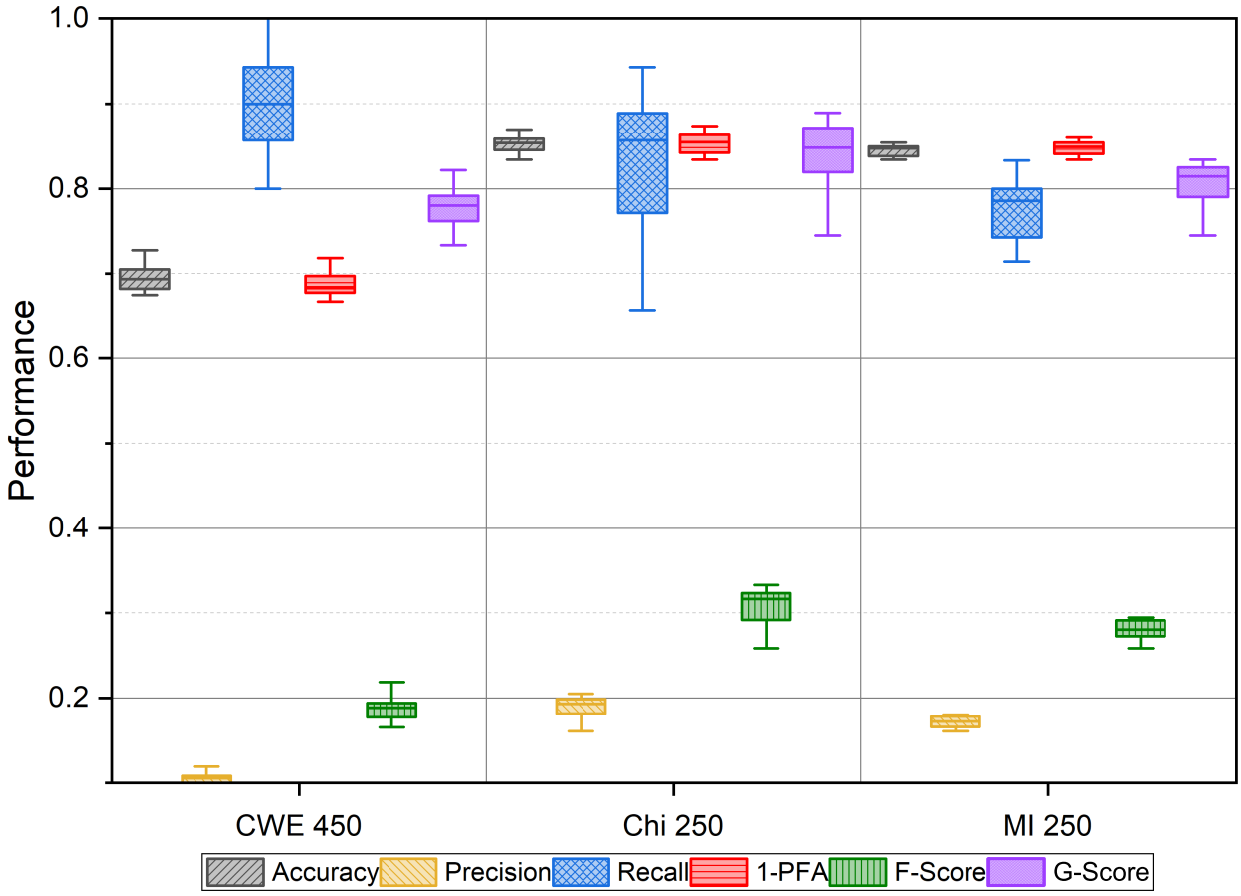

Figure 5.1: Red Hat Enterprise Linux 4 Term Frequency Feature Selection Box Plot (Naive Bayes)

5.3.2 Ground Mission IV&V Issues Results

The Ground Mission IV&V Issues dataset baseline results showed a low precision value. Feature selection results are shown in Table 5.2. Feature selection results were similar to

the baseline while chi-squared feature selection slightly improved the F-Score and G-Score values. However, mutual information feature selection improved the precision by 7.1% while degrading recall by 5.2%. A box plot comparing the results is shown in Figure 5.2.

Table 5.2: Ground Mission IV&V Issues Term Frequency Feature Selection Performance (Naive Bayes)

Feature Vector	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
CWE 450	85.4%	33.3%	85.7%	85.4%	48.0%	85.5%
Chi 250	**85.5%**	**33.6%**	**85.7%**	**85.5%**	**48.2%**	**85.6%**
MI 250	87.0%	35.7%	81.3%	87.5%	49.6%	84.3%

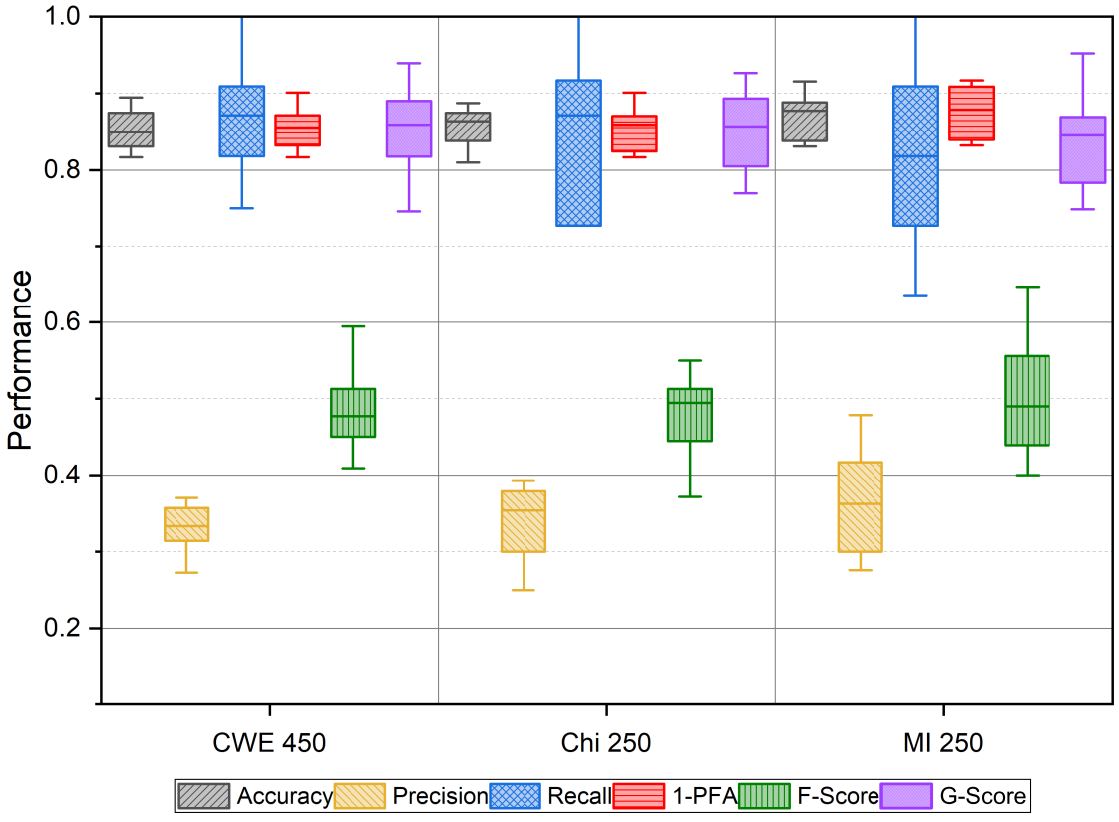

Figure 5.2: Ground Mission IV&V Issues Term Frequency Feautre Selection Box Plot (Naive Bayes)

5.3.3 Flight Mission IV&V Issues Results

The Flight Mission IV&V Issues dataset baseline resulted in a relatively high precision value. Feature selection results are shown in Table 5.3. Mutual information feature selection improved slightly across all metrics. Chi-squared feature selection degraded the results. A box plot comparing the results is shown in Figure 5.2.

Table 5.3: Flight Mission IV&V Issues Term Frequency Feature Selection Performance (Naive Bayes)

Feature Vector	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
CWE 450	78.8%	71.1%	80.8%	77.3%	75.7%	79.0%
Chi 250	75.2%	66.9%	77.6%	73.5%	71.9%	75.5%
MI 450	**80.4%**	**72.4%**	**84.0%**	**77.9%**	**77.8%**	**80.8%**

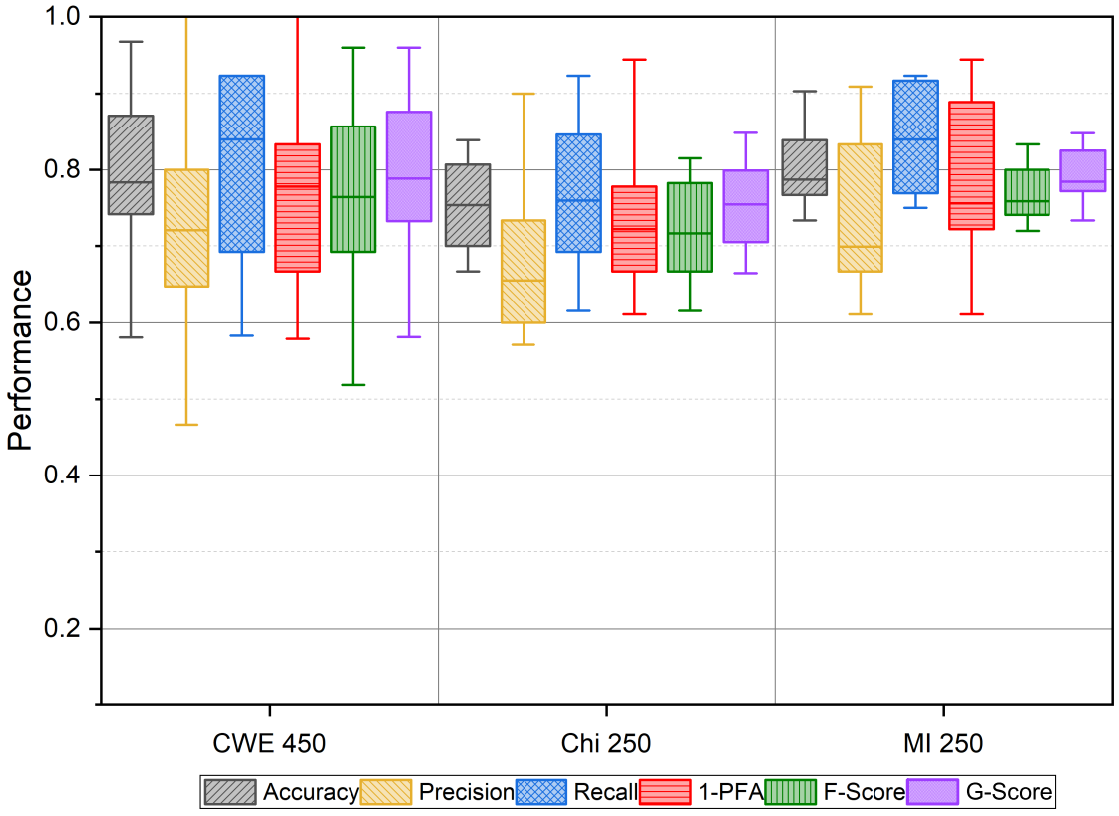

Figure 5.3: Flight Mission IV&V Issues Term Frequency Feature Selection Box Plot (Naive Bayes)

5.3.4 Flight Mission Developer Issues Results

The Flight Mission Developer Issues dataset baseline resulted in a relatively high precision value. Feature selection results are shown in Table 5.4. Feature selection degraded the performance on the dataset. The dataset was relatively small so the validation set used to train feature selection was small. A box plot comparing the results is shown in Figure 5.2.

Table 5.4: Flight Mission Developer Issues Term Frequency Selection Performance (Naive Bayes)

Feature Vector	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
CWE 450	**55.2%**	**69.7%**	**54.6%**	**56.3%**	**61.2%**	**55.4%**
Chi 250	59.6%	69.2%	67.8%	44.4%	68.5%	53.6%
MI 250	61.1%	72.2%	65.1%	53.8%	68.4%	58.9%

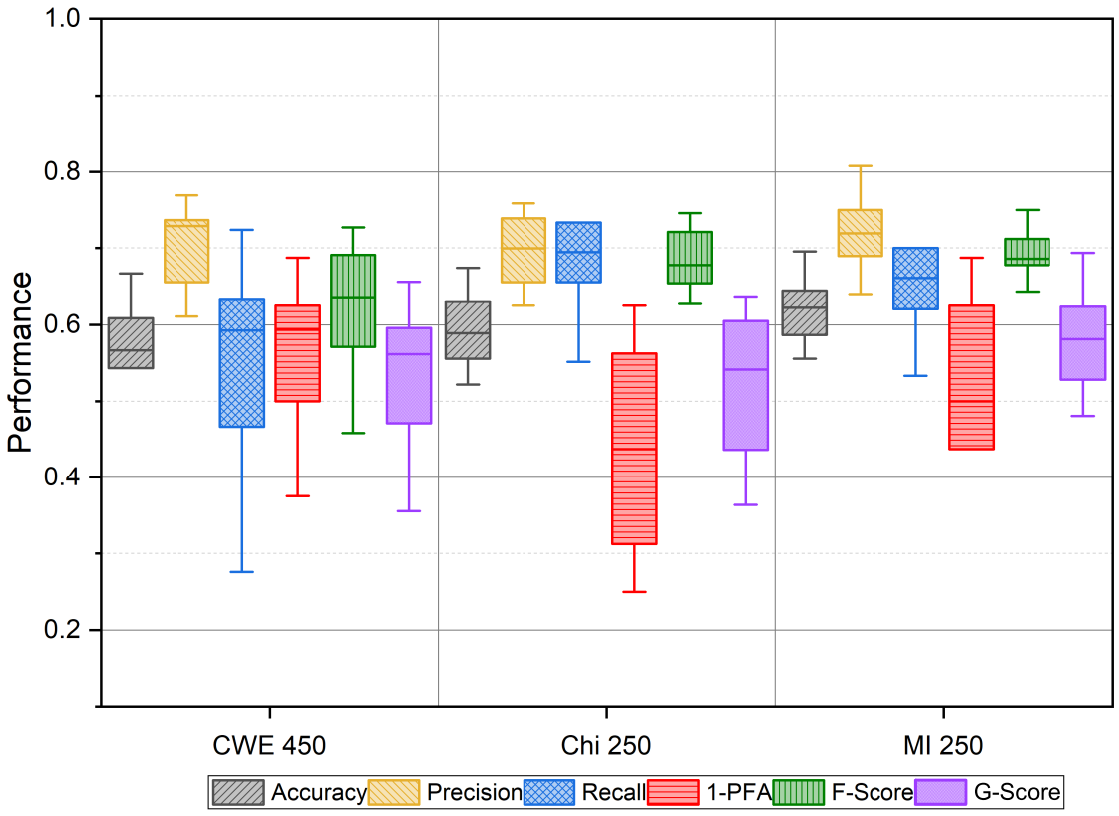

Figure 5.4: Flight Mission Developer Issues TF Chi-squared Box Plot (Naive Bayes)

5.4 Feature Selection Conclusion

Feature selection may be used as to reduce feature vectors of large vocabularies. The amount of words in feature vectors and type of feature selection may be tuned to suit the needs of the particular problem.

Through the use of feature selection, the baseline results were improved in certain areas as seen by Table 5.5.

Table 5.5: Feature Selection Improvement of Classification Performance (Naive Bayes)

Dataset	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Red Hat	*Chi 250*	85.3%	18.9%	83.2%	85.4%	30.8%	84.3%
Enterpise Linux 4	*Improvement*	+22.7%	+78.9%	-7.9%	+24.3%	+62.8%	+8.0%
Ground Mission	*MI 250*	87.0%	35.7%	81.3%	87.5%	49.6%	84.3%
IV&V Issues	*Improvement*	+1.9%	+7.1%	-5.2%	+2.5%	+3.3%	-1.5%
Flight Mission	*MI 250*	80.4%	72.4%	84.0%	77.9%	77.8%	80.8%
IV&V Issues	*Improvement*	+2.1%	+1.8%	+4.0%	+0.7%	+2.8%	+2.3%
Flight Mission	*MI 250*	61.1%	72.2%	65.1%	53.8%	68.4%	58.9%
Developer Issues	*Improvement*	+10.8%	+3.6%	+19.3%	-4.4%	+11.8%	+6.3%

Research Question 1 was explored.

RQ 1: What feature selection methods show the greatest improvement of the automated bug report classification?

1. Both Chi-squared feature selection and mutual information (info gain) feature selection may be used and the results be depend on the datasets.

   The results showed that Chi-squared had the greatest affect on improving the precision metric at the cost of recall metric on the Red Hat Enterprise Linux 4 dataset. However, mutual information feature selection improved the classification performance on the Ground Mission IV&V Issues, the Flight Mission IV&V Issues, and the Flight Mission Developer Issues datasets.

2. Feature selection may be used to reduce the size of the vocabulary.

Feature selection may be desired if one needs to build a vocabulary to be effective at a limited size. It may be necessary to develop an approach to select the best vocabulary size based on a validation test set. Feature selection improved the results in the two large datasets. However, the vocabulary size was not changed for the two smaller datasets.

3. Feature selection may improve a low precision value.

   The two datasets with low precision values were improved by feature selection. The datasets that started with relatively high precision values did not improve as much from feature selection.

5.5 Threats to Validity

Construct validity threats concerned with measuring what we intend to measure arose from dealing with different datasets. The size of the dataset affects the size of the vocabulary. When all vocabularies of the datasets are shrunk down to the same size, there is a chance for a greater change on the larger datasets. Feature selection takes a supervised approach which is generated from a validation set. Different validation sets will impact the feature selection.

Internal validity threats concerned with unknown influences arose from the input data. An assumption was made that the datasets of appropriate labeling for the software issues. There is also no guarantee that an issue is or is not a security issue without exploring the project's data itself which is often unavailable. The issues were labeled as security related based on the similarity to CWE descriptions. There is also a threat to the data quality. There is some assurance of data quality due to the standard of record keeping in NASA records.

Conclusion validity threats affect the ability to draw the correct conclusions. The conclusions are based on performance metrics which are all reported. The thesis focuses on the G-Score value as it is the harmonic mean between precision and recall. We care most about recall and precision values.

External validity threats are concerned with the ability to generalize results. The experiments are based on four large datasets that were created by different developers over multiple years. We also do not claim the findings in this thesis are valid across other software systems.

Chapter 6

Clustering

Unsupervised learning has the advantage of not requiring labeled data. Anomaly detection requires only one class and classifies all outliers away from the normal as the different class. Distance measures such as cosine similarity and euclidean distance may be used to compute the similarity from a data point to the normal points. A threshold must be computed when using anomaly detection for classification with cosine similarity.

Clustering may be used as an unsupervised classification. Clustering groups datapoints together based on the distance between points. Each cluster attempts to only have datapoints of the same class. This section explores Research Questions 2, 2(a), and 2(b).

RQ 2: Can unsupervised classification based on clustering outperform the anomaly detection based on cosine similarity?

(a) How do Term Frequency feature vectors perform in combination with K-Means clustering?

(b) Can Variational Autoencoders be used to represent security related software issues in a distribution of software bug reports?

K-means algorithm uses a local search to partition data points into k clusters. K-means is initialized with k-clusters chosen arbitrarily. Each point is then assigned a cluster closest to it. The centers are recomputed and the cycle continues till the clusters stabilize [1].

Autoencoders may be used for unsupervised training. In an autoencoder network, the output is the same as the input, therefore there is no need for labeled data. The autoencoder

network may be split into two parts: encoder and decoder. The encoder portion typically creates a smaller representation of the input data. This latent representation may represent the input data in a meaningful way. The decoder portion is used to map the latent representation to target data, in this case, the input data. After training, the decoder portion is no longer necessary.

6.1 Background on Variational Autoencoders

Variational Autoencoders (VAE) proposed by Kingma et al. capture the latent representation of a date set through the use of a distribution, often a Gaussian distribution [19]. Variational Autoencoders may be used for training a generative model [7]. Generative modeling is an area of machine learning which represents models of distributions $\mathrm{P}(X)$ defined over datapoints X. Generative models attempt to capture the dependencies between features in order to reorganize new features into similar objects. A generative model takes in known examples X which are distributed to some unknown distribution $P_{gt}(X)$.

Training a model to represent the unknown distribution has been proven difficult prior to VAEs. There have been three drawbacks to the problem. First, a model might require strong assumptions about the structure in the data. Second, the model might be required to make approximations of the data leading to sub optimal models. Thirdly, the model would rely on computationally expensive inference procedures such as Markov chain Monte Carlo. VAEs allow the the training to be fast via backpropagation and do not require specific structures in the data sets. Although VAEs do make approximations, the error introduced is generally small with large data sets [7].

The goal for a VAE is to capture the dependencies between dimensions. A generative model with many dimensions and complicated dependencies will be more difficult to train. A VAE first decides which variable to generate. This decision is represented as a *latent variable* z. In order for the model to represent the dataset, for every data point X, there needs to be one (or more) settings of the latent variable that generates an output similar to X. The model has a vector of latent variables z in a high dimensional space Z. The latent variables are sampled over some probability density function (PDF) $P(z)$ defined over Z. Now there

is a family of deterministic functions *f(z; θ)* parameterized by a vector θ in some space Θ, where f: $Z \times \Theta \rightarrow \mathcal{X}$, and $\mathcal{X}$ is the high-dimensional space of X. Since f is deterministic, z is random, and θ is fixed, then *f(z; θ)* is a random variable in the space $\mathcal{X}$. The task is then to optimize θ so that one can sample z from *P(z)*. Now *f(z; θ)* will be similar to the X datapoints in the dataset as seen in Equation 6.1 below [7].

$$P(X) = \int P(X|z;\, \theta)P(z)dz. \tag{6.1}$$

The model is based on "maximum likelihood" where if a model is likely to produce the dataset samples then it is also likely to produce similar dataset samples. The output follows a Gaussian distribution where $P(X|z;\, \theta) = \eta(X|\ f(z;\, \theta),\, \sigma^2 \cdot I)$. The distribution then has a mean of *f(z; θ)* and covariance equal to the identity matrix I times some scalar σ. The goal of the VAE is to maximize Equation 6.1.

There are two problems that need to be solved before the model can represent the dataset. The model must choose how to represent the latent variables z and how to deal with the integral over z. VAEs assert that samples z can be drawn from a distribution $\eta(0,\, I)$. These simple latent variables z are then mapped using complicated functions. In an autoencoder like structure, the first set of layers map z to the latent values. The later layers map the latent values to a represented data point.

Now the second problem, to maximize *P(X)*, can be completed by finding a computable formula for *P(X)* and taking the gradient of that formula. The formula is found by sampling over a large number of z values and computing $P(X) \approx \frac{1}{n} \sum_i P(X|z_i)$. However, with high dimensions, n may need to be very large to find an appropriate approximation. Given that for most z, *P(X|z)* will be close to zero. Therefore, variational autoencoder's key concept is to sample from a new function $\mathcal{Q}(z|X)$ which takes a sample X and returns a distribution of z likely to produce something similar to X. By applying Kullback-Leiber divergence and Bayes rule, the core of variational autoencoders is Equation 6.2.

$$logP(X) - \mathcal{D}[\mathcal{Q}(z|X)||P(z|X)] = E_{z\sim Q}[logP(X|z)] - \mathcal{D}[\mathcal{Q}(z|X)||P(z)] \tag{6.2}$$

The left side of the equation has the value to maximize: log *P(X)*, plus an error term that makes $\mathcal{Q}$ produce z's that can reproduce X. The right side of the equation can be optimized using gradient descent.

6.2 Variational Autoencoders Proposed Approach

An implementation of a variational autoencoder was created from a provided architecture from Keras [18]. The provided architecture had a hidden units dense layer of 512 units; however, the number of units was adjusted to 256 units to compensate for the large datasets running on a CPU. The latent representation dimension was only 2. For larger datasets the latent representation size may need to be increased; however, the size did not affect results in a meaningful way in the datasets tested. A batch size of 128 was used and was trained for 35 epochs. The loss function "binary_crossentropy" was used. Code was written using Python [33], Sci-Kit Learn [29] [3], Matplotlib [17], and Keras [4]. The architecture is shown in Figure 6.1 and is an adaption of a Variational Autoencoder proposed by Kingma et al [19] and provided by Keras [18].

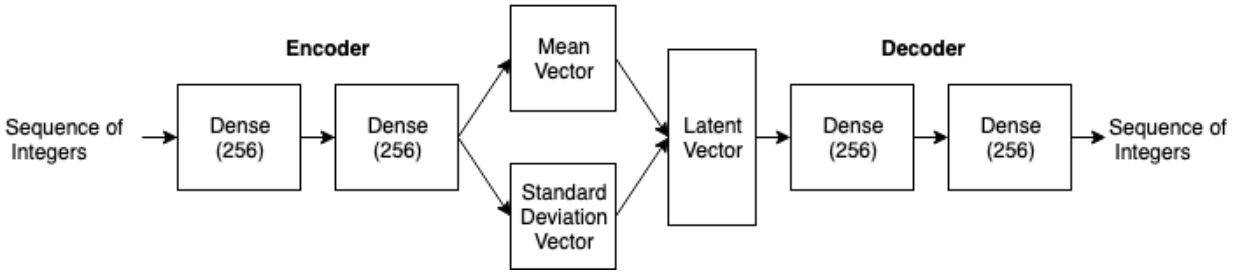

Figure 6.1: VAE Network Architecture Adaption [19] [18]

The weights of the variational autoencoder were used in an encoder network which outputs a latent representation of the data. The latent representation is used in K-Means clustering for classification. The K-Means implementation used the expected-maximization algorithm from Elkan [8]. The security clusters were determined from the CWE-888 issues. The amount of security clusters was determined by a set number of clusters. Any software issue that was in a cluster labeled as security related was classified as security related.

K-Means clustering is used with k=100 clusters. The large k was selected as the datapoints cluster closely together, and different clusters may form within a large amount of datapoints. The Euclidean distance measure was used for determining the clusters. CWE-888 descriptions determine if a cluster was security related or non security related. Firstly

all clusters that do not include a CWE issue are classified as non-security. Next, the 25 clusters with the smallest amount of CWE issues are also classified as non-security. The remaining clusters are classified as security related. The value of 25 clusters was selected through tuning a validation test set for the best classification performance.

A Term Frequency representation of the data is also used. The Term Frequency feature vector is used in the same K-Means algorithm. To visualize the Term Frequency data in a two dimensional space, Principal Component Analysis (PCA). PCA does not affect the K-Means algorithm.

6.3 Clustering Results

The figures in this section represent latent representations of the software issues. The first figure for a project shows all software issues and CWE-888 descriptions labeled by the ground truth. The labels are used for visualization purposes. The ideal results would be security related issues clustering on or near the CWE descriptions while the non-security related issues would vary away from the CWE clusters. The second figure shows the same plot with K-means clustering and classification. The two plots repeat for variational autoencoders and term frequency representation visualized through PCA. Tables of results were included for the projects in which security related software issues clustered with CWE-888 descriptions. The classifier with the highest G-Score value is in bold in the provided tables. The method with data represented by variational autoencoders is labeled as "VAE". The method with data represented by term frequency feature vectors is labeled as "TF".

We used K-means clustering to classify software issues as security related or not. However, the data was not represented well enough to form clusters of security related issues differentiating from non-security related issues. Therefore, cosine similarity anomaly detection is our preferred method for unsupervised classification on security bug reports.

The representation of data did show the CWE-888 issues are similar to each other when represented by variational autoencoders. Portions of security software issues were similar to one another; however, non-security software issues were also closely similar.

6.3.1 Red Hat Enterprise Linux 4 Results

The Red Hat Enterprise Linux 4 dataset was represented by the VAE embeddings in Figure 6.2. The VAE network was trained on all software issues including the CWE-888 list. The network training was unsupervised and the output is the same as the input. The network was cut to the latent representation and K-Means was used to predict security related bug reports.

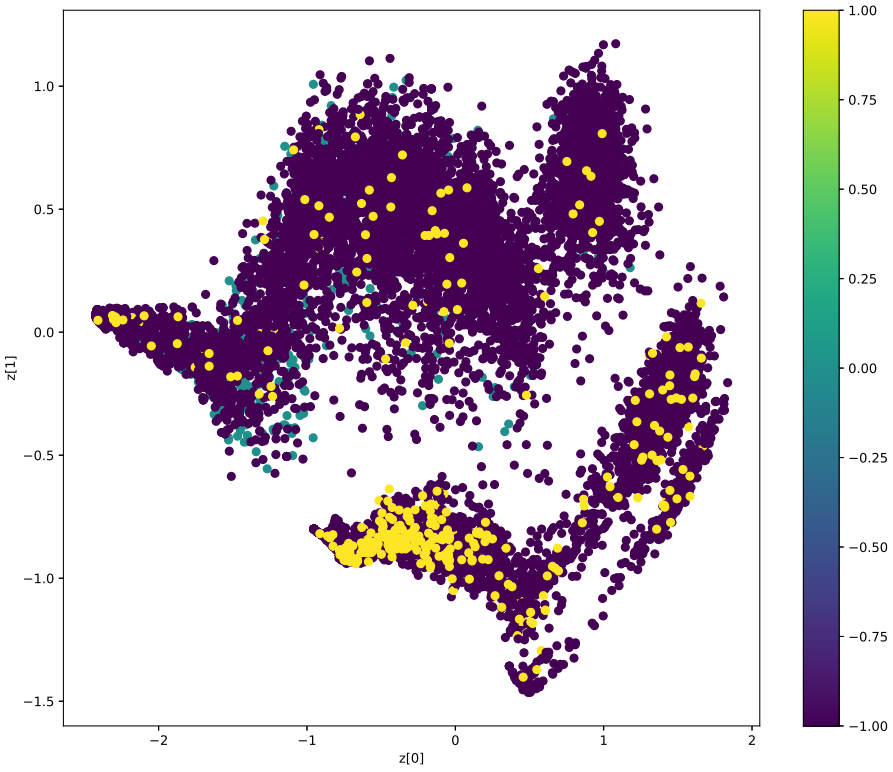

Figure 6.2: Redhat Enterprise Linux 4 VAE Representation: Security[1] CWE[0] Non-security[-1]

After multiple trials, the VAE network was not able to represent the Red Hat dataset in a way that distributed CWE-888 descriptions along with security related issues. Since the distribution does not cluster security issues with CWE descriptions, the classification is unable to perform, therefore metric performance results are not included for this dataset.

However, it can be seen by the representation of data that the majority of security related issues do cluster together. Although, they are clustered with a portion of non-security related issues, comparing the issues to a different baseline may be beneficial.

Similar findings were found using PCA analysis on the dataset as shown by Figure 6.3. The CWE-888 lists were very different than all issues in the dataset and are clustered far apart. The data was not represented in a way in which K-Means clustering could differentiate between security related and non-security related software issues. PCA analysis visualization also showed security related issues were similar to one another.

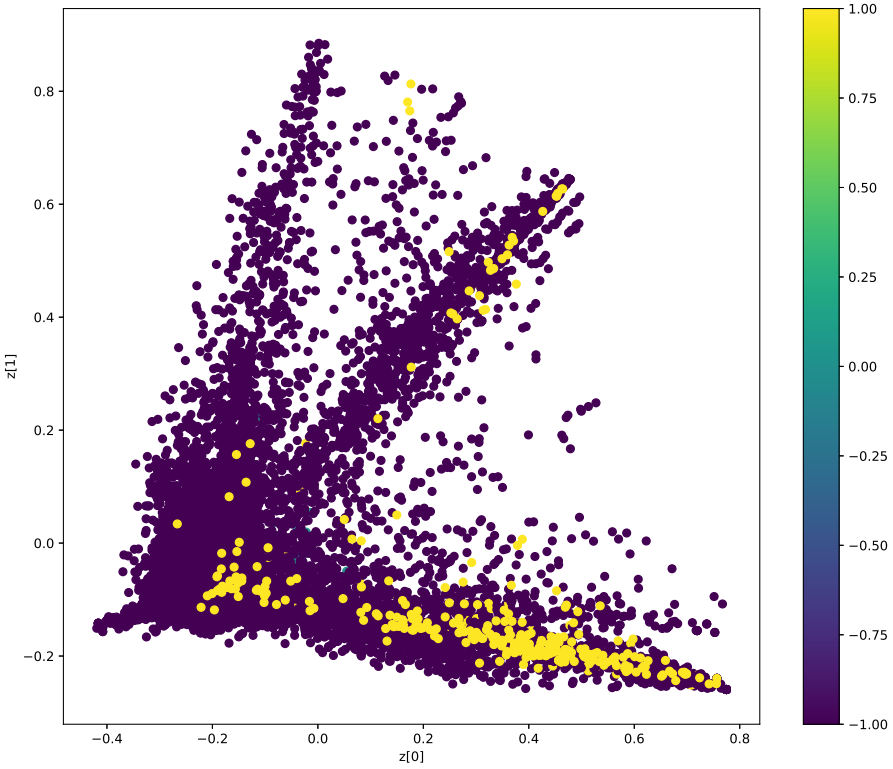

Figure 6.3: Red Hat Enterprise Linux 4 PCA Representation: Security[1] CWE[0] Non-security[-1]

The software issues and CWE-888 descriptions were not similar in the Red Hat dataset under the experimented representations.

6.3.2 Ground Mission IV&V Issues Results

Unsupervised classification through clustering was applied to the NASA IV&V Ground project as well. The results in Table 6.1 show the baseline cosine similarity results compared to the clustering results. K-means clustering on Term Frequency vector was not able to detect a majority of security related issues resulting in a low recall. K-Means clustering through VAE representation performed similarly to the baseline, with a higher recall value.

Table 6.1: Ground Mission IV&V Issues Classification from Clustering Results

Dataset	*Classifier*	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Ground	*Cosine Sim*	*TF*	64.3%	15.0%	78.7%	63.1%	25.2%	70.0%
Project	*K-Means*	*TF*	78.8%	15.0%	36.7%	82.4%	21.3%	50.8%
IV&V Issues	***K-Means***	***VAE***	**63.7%**	**16.3%**	**88.5%**	**61.6%**	**27.6%**	**72.6%**

The VAE representation is shown in Figure 6.4. All issues in the datasets along with the CWE-888 list were used to train the network to represent the input data through a distribution.

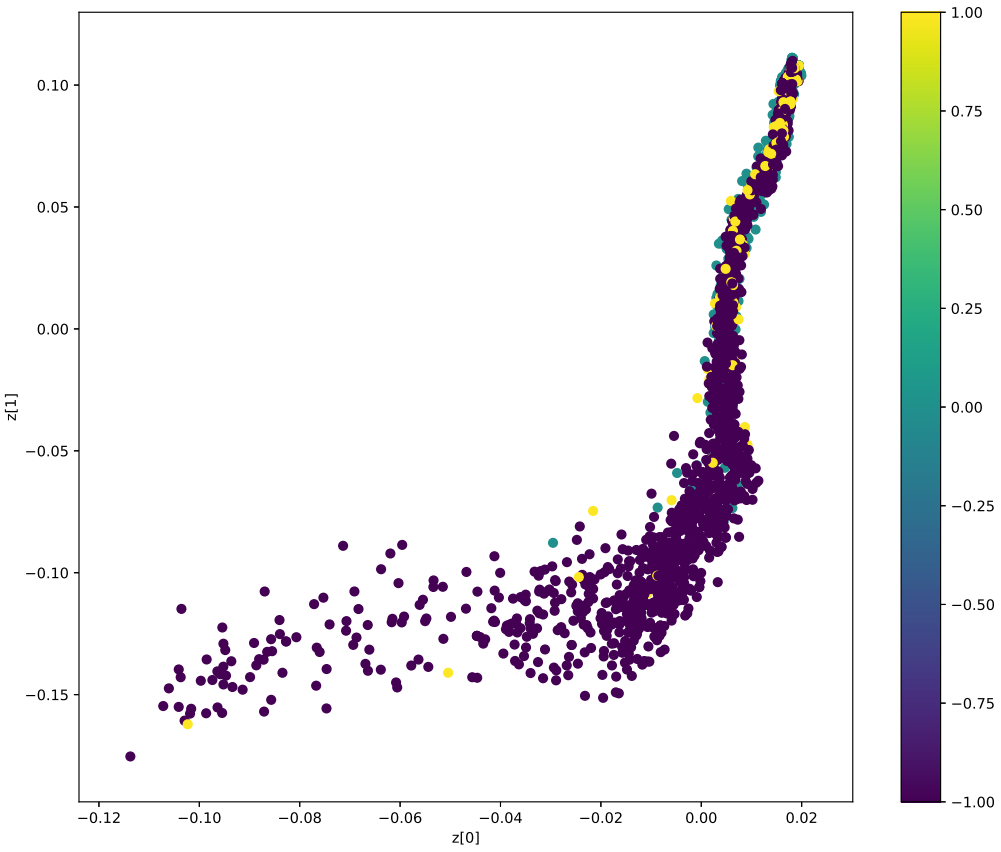

Figure 6.4: Ground Mission IV&V Issues Ground Truth VAE Representation: Security[1] CWE[0] Non-security[-1]

The ground truth graph did show an overlap of security related software issues and CWE-888 issues. However, it was difficult to distinguish the security related issues from the non-security related issues. K-Means clustering is performed on the data representation seen in Figure 6.5. The clustering appears to label the majority of security related issues; however, there is an overlap of non-security related issues.

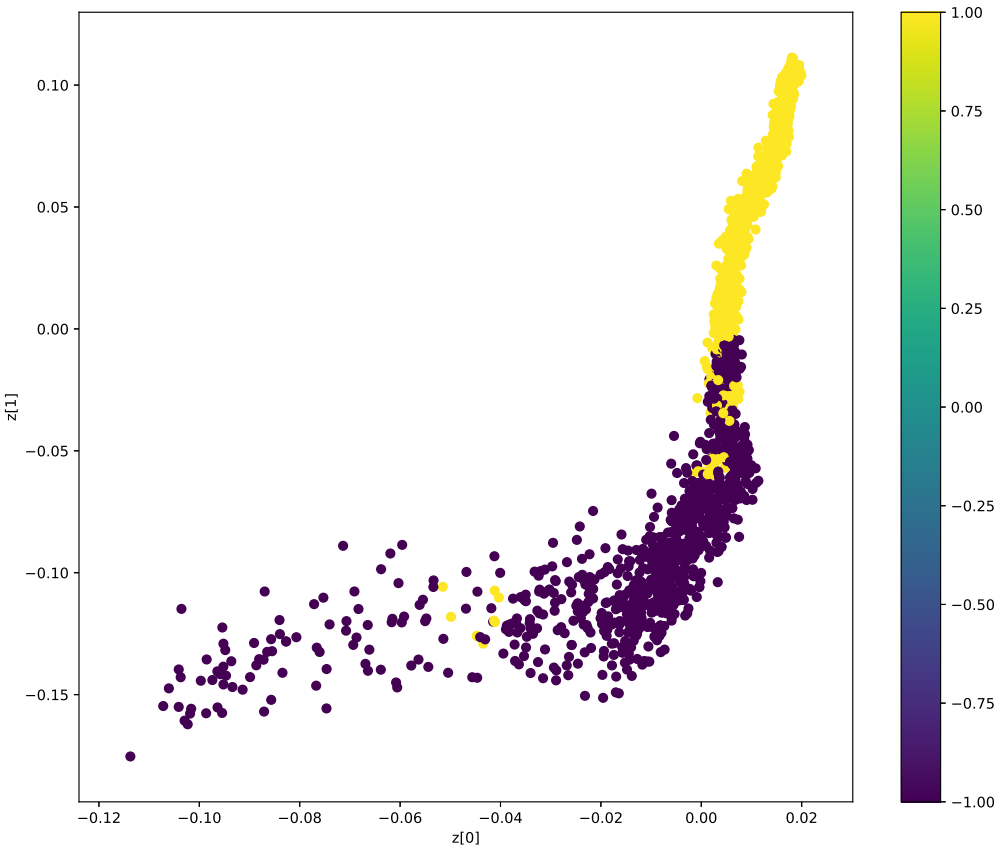

Figure 6.5: Ground Mission IV&V Issues K-Means VAE Representation: Security[1] Non-security[-1]

PCA analysis of the dataset showed the CWE-888 descriptions were not similar to the security related issues. The CWE-888 descriptions appeared to be represented away from all issues. Security related issues were represented similarly.

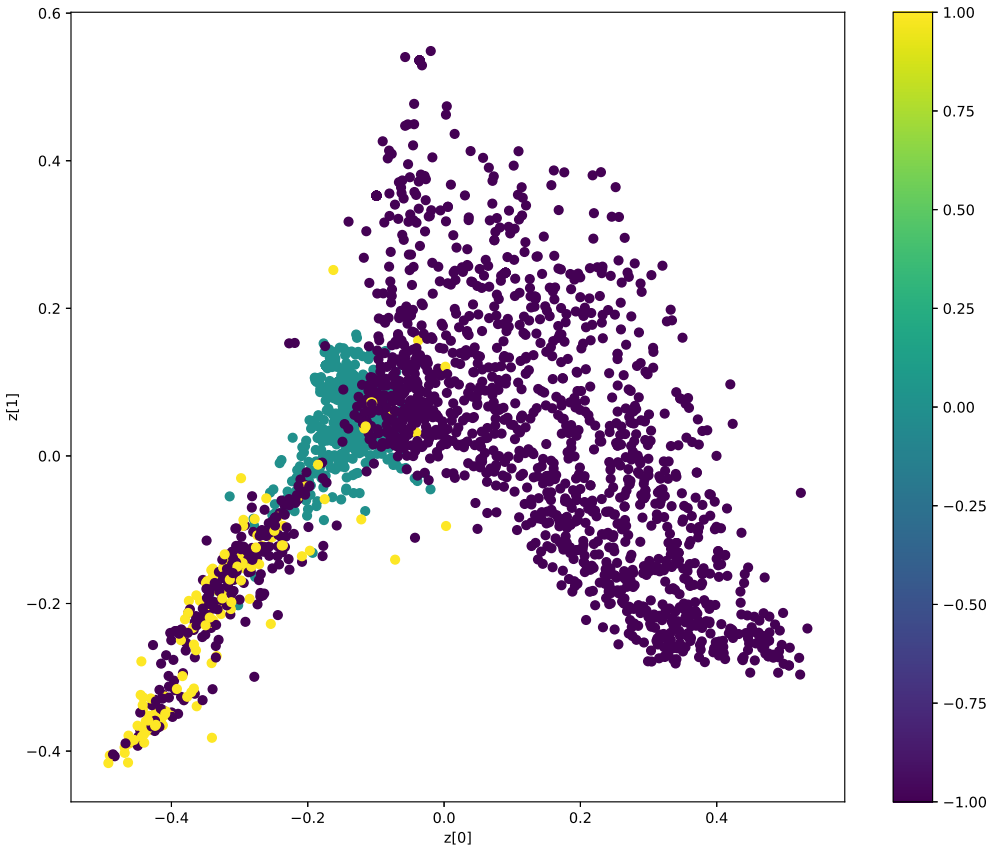

Figure 6.6: Ground Mission IV&V Issues Ground Truth TF PCA Representation: Security[1] CWE[0] Non-security[-1]

K-Means clustering as shown by Figure 6.7 did not cluster security software issues with CWE-888 descriptions.

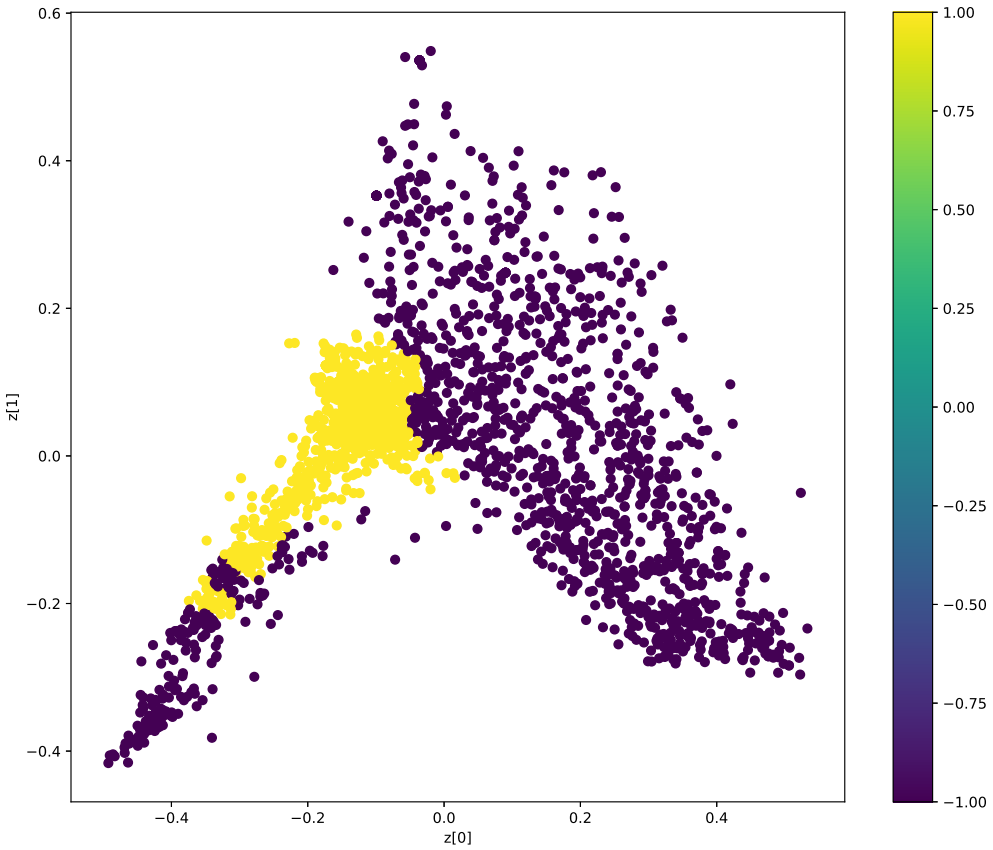

Figure 6.7: Ground Mission IV&V Issues K-Means TF PCA Representation: Security[1] Non-security[-1]

6.3.3 Flight Mission IV&V Issues Results

Flight Mission IV&V Issues unsupervised classification results are shown in Table 6.10. Both clustering methods did not perform well compared to the baseline. The main reason was that the majority of security related issues did not cluster within the CWE-888 clusters. Cosine similarity was still able to perform effectively for unsupervised classification. The reason may be seen in Figure 6.10, although the security issues did not cluster with the CWE-888 clusters, the issues were closer and therefore more similar than the non-security

related issues.

Table 6.2: Flight Mission IV&V Issues Classification from Clustering Results

Dataset	*Classifier*	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Flight	***Cosine Sim***	***TF***	**67.8%**	**58.1%**	**77.7%**	**60.9%**	**66.5%**	**68.3%**
Project	*K-Means*	*TF*	62.4%	69.4%	15.8%	95.1%	25.8%	27.1%
IV&V Issues	*K-Means*	*VAE*	66.1%	68.0%	33.5%	88.9%	44.9%	48.7%

Flight Mission IV&V Issues dataset representation shown in Figure 6.8 shows clustering of CWE-888 descriptions along with security related issues. Many of the other security related issues were scattered throughout the representation away from the CWE clustering.

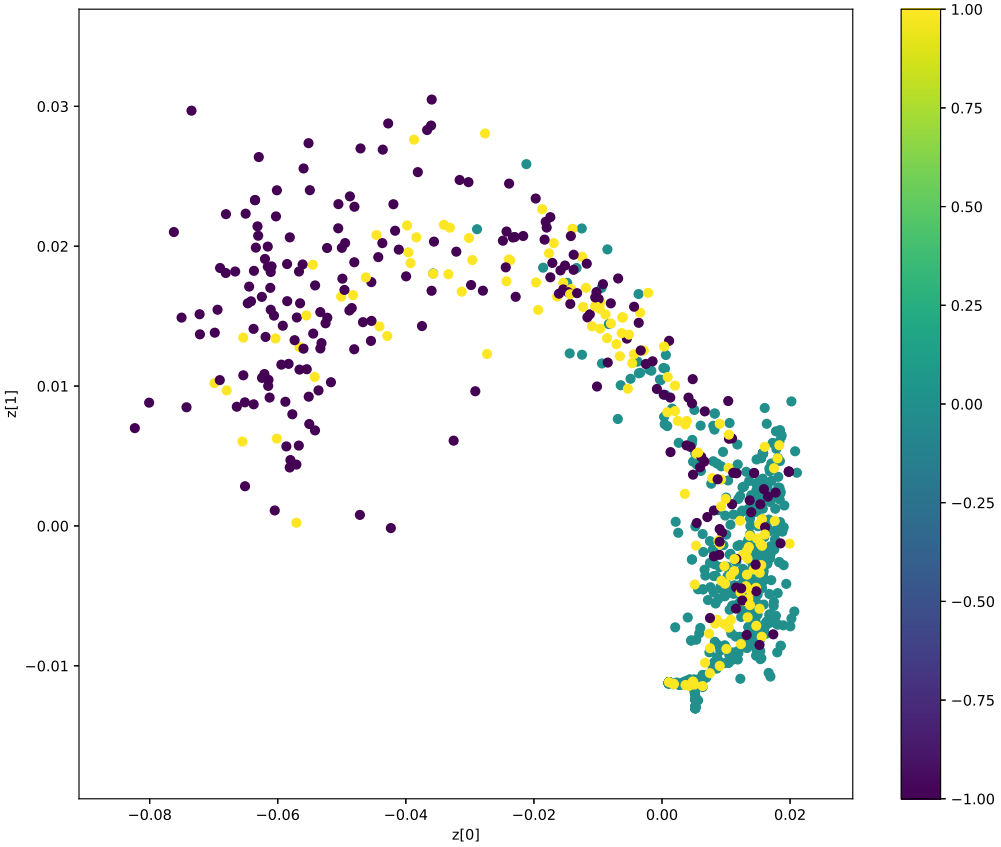

Figure 6.8: Flight Mission IV&V Issues Ground Truth VAE Representation: Security[1] CWE[0] Non-security[-1]

K-Means clustering was applied to the representation of data to detect the security related issues. Comparing the two graphs, the algorithm was able to form non-security related clusters within gaps of the CWE-888 clusters to reduce the false positives.

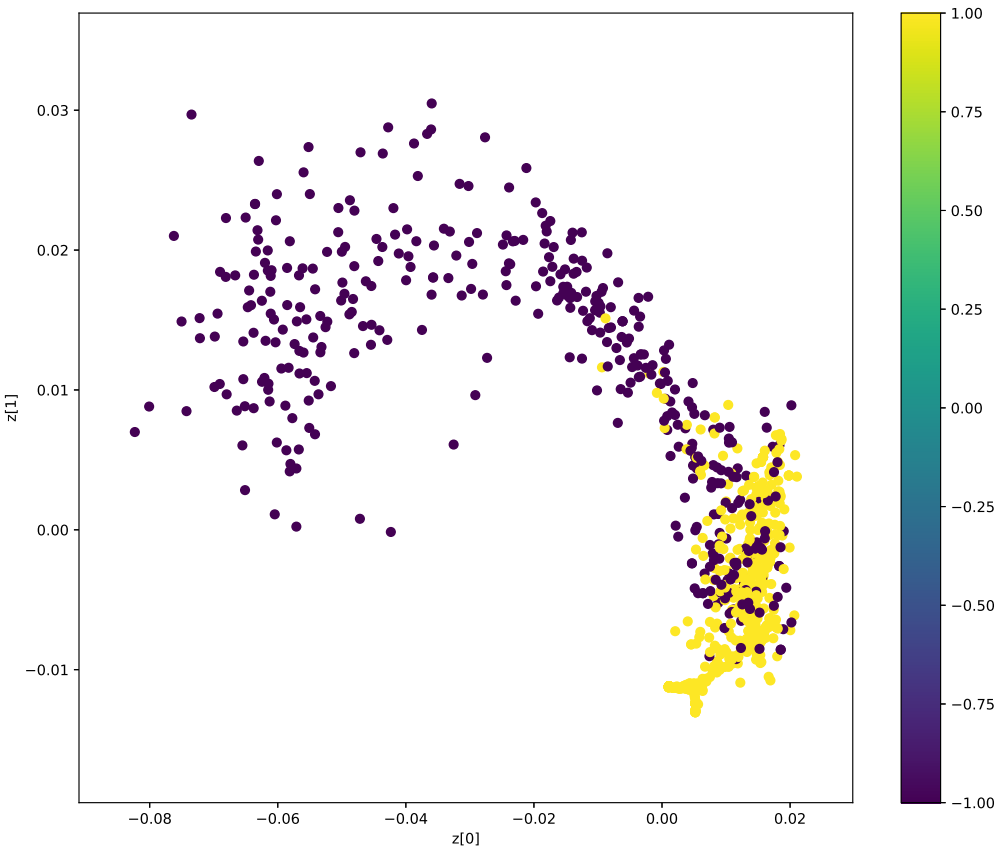

Figure 6.9: Flight Mission IV&V Issues K-Means VAE Representation: Security[1] Non-security[-1]

PCA analysis on the Flight Mission IV&V Issues dataset clusters the security related issues closer together. Even though the CWE-888 descriptions are aligned with the security issues, The CWE-888 descriptions were spread out which made it difficult to cluster.

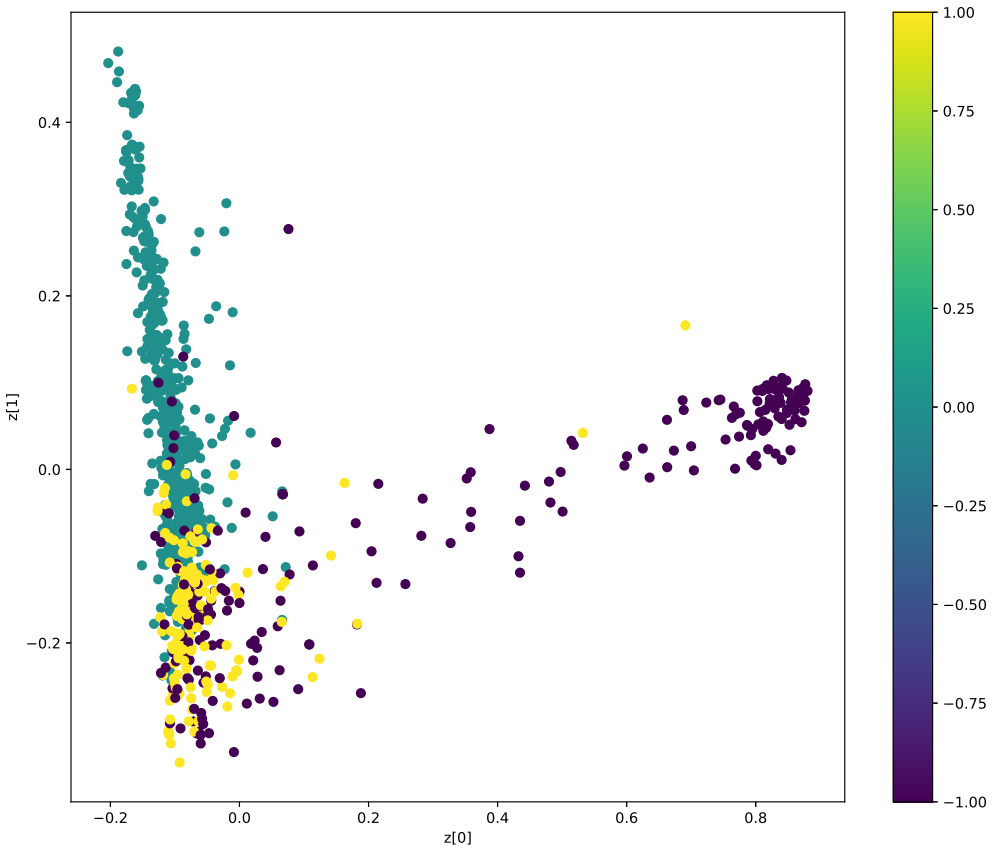

Figure 6.10: Flight Mission IV&V Issues Ground Truth TF PCA Representation: Security[1] CWE[0] Non-security[-1]

K-Means clustering mislabels a large portion of security related issues because of the spread of the CWE-888 descriptions. The non-security related issues did cluster away from the CWE-888s, which created a very lower false positive rate.

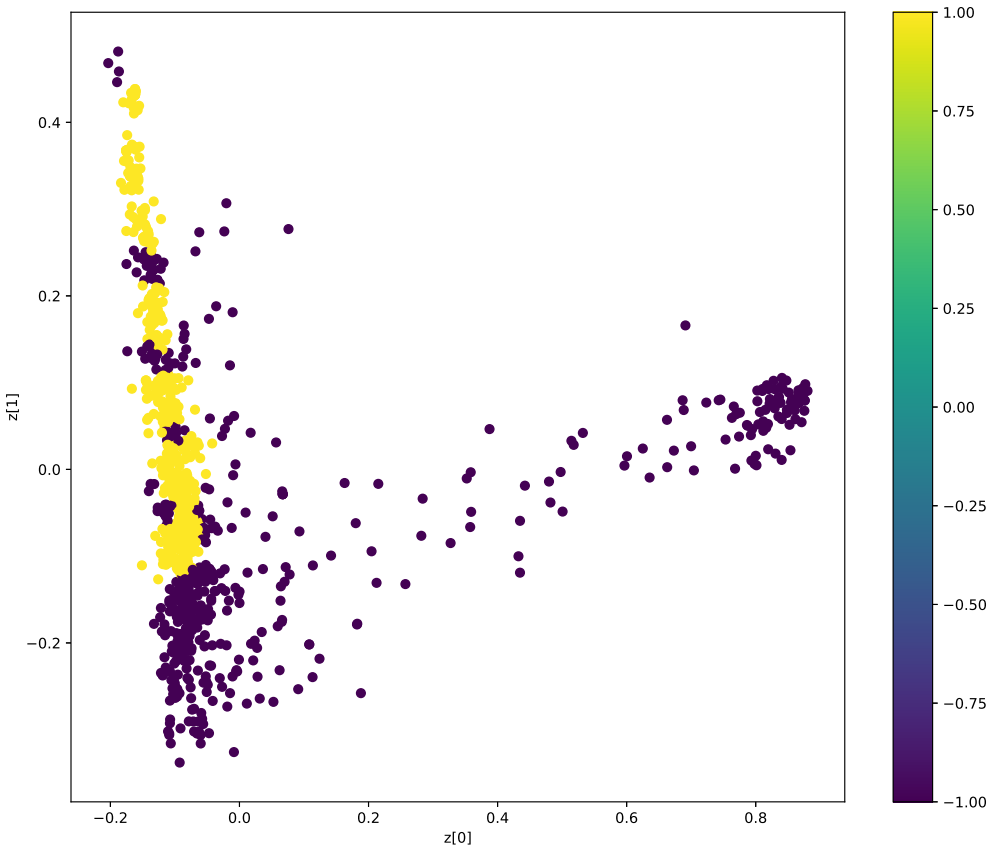

Figure 6.11: Flight Mission IV&V Issues K-Means TF PCA Representation: Security[1] Non-security[-1]

6.3.4 Flight Mission Developer Issues Results

Flight Mission Developer Issues unsupervised classification results in Table 6.3 show that clustering with the TF feature vector performs very poorly. However, clustering using a VAE representation performed similarly to cosine similarity. The VAE managed to represent the data so that a portion of the security related issues were similar to the CWE-888 descriptions.

Table 6.3: Flight Mission Developer Issues Classification from Clustering Results

Dataset	*Classifier*	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Flight	***Cosine Sim***	***TF***	**55.4%**	**69.3%**	**57.9%**	**50.6%**	**63.1%**	**54.0%**
Project	*K-Means*	*TF*	39.2%	68.4%	13.9%	62.4%	32.2%	24.0%
Developer Issues	*K-Means*	*VAE*	52.9%	67.6%	54.6%	49.7%	60.4%	52.0%

Flight Mission Developer Issues VAE representation in Figure 6.12 did not appear to distinguish between security and non security related issues. The majority of issues aligned along with the CWE-888 descriptions. The remaining issues were dispersed away from the majority clusters.

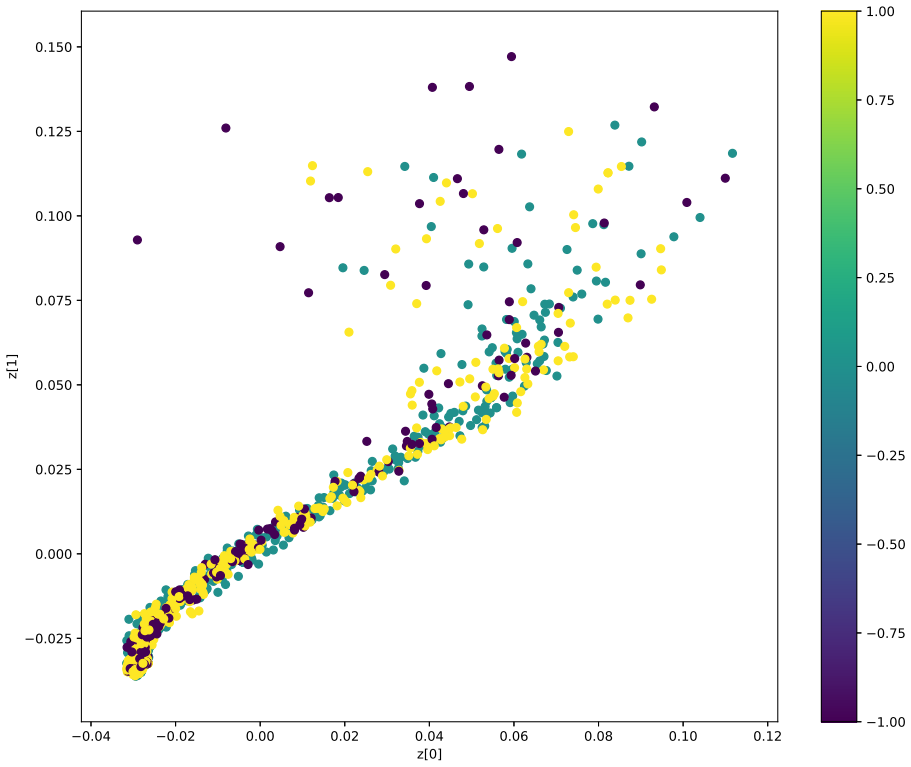

Figure 6.12: Flight Mission Developer Issues Ground Truth VAE Representation: Security[1] CWE[0] Non-security[-1]

K-Means clustering attempted to pick out the clusters with the most CWE-888 issues. The clusters attempted to only cluster the issues that are closely related to the CWE-888

descriptions, this allows for non-security related clusters to formed tightly around.

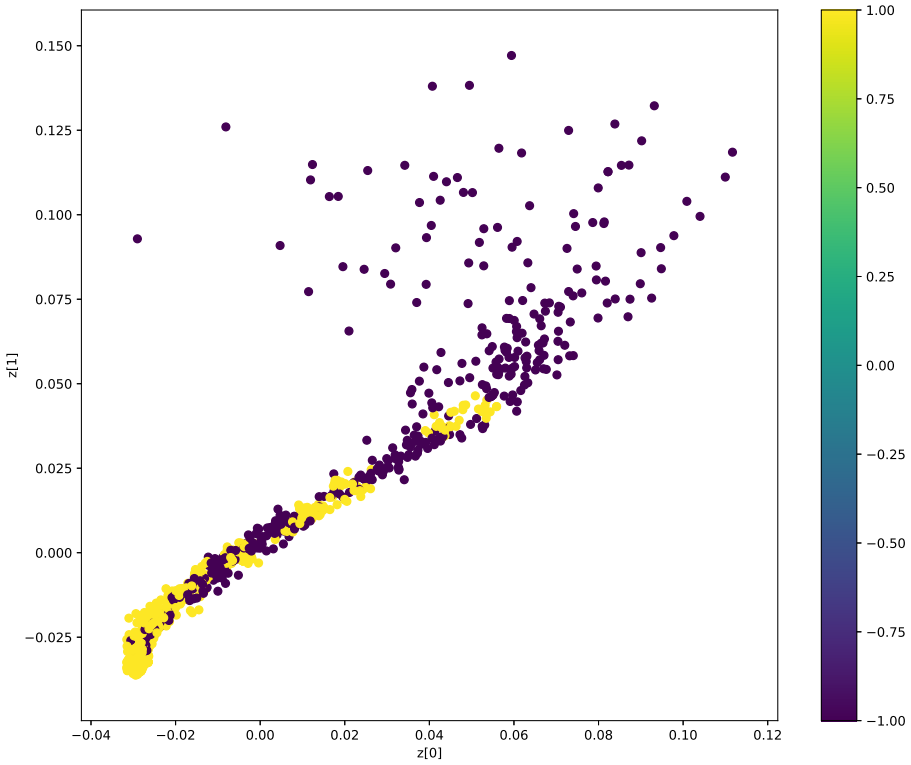

Figure 6.13: Flight Mission Developer Issues K-Means VAE Representation: Security[1] Non-security[-1]

PCA analysis representation shows that the CWE-888 issues were not similar to the dataset's software issues. The representation was therefore not beneficial in a clustering scenario.

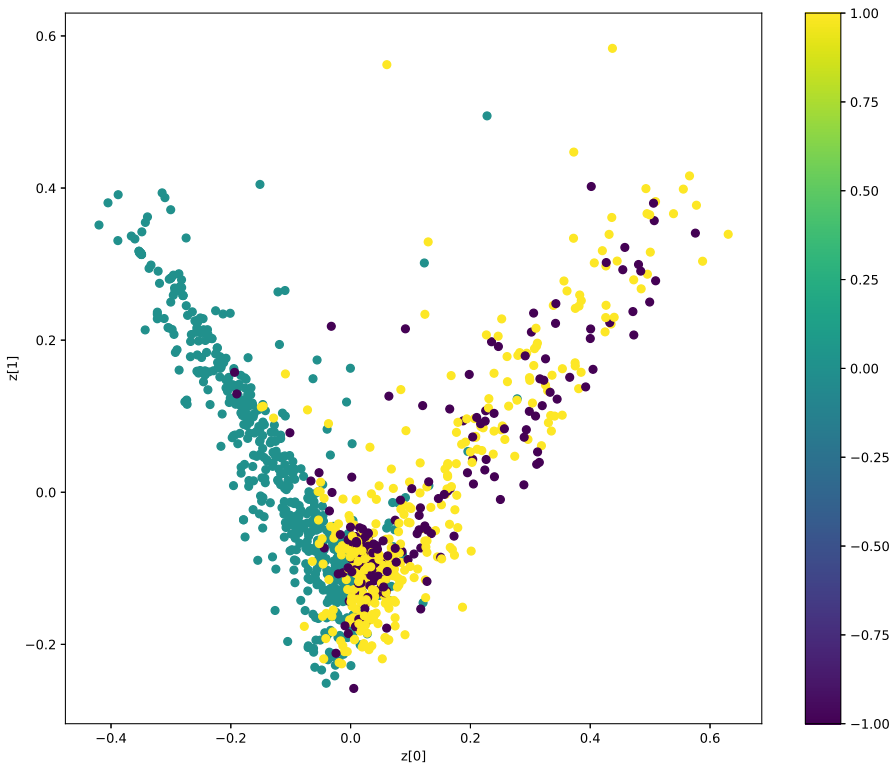

Figure 6.14: Flight Mission Developer Issues Ground Truth TF PCA Representation: Security[1] CWE[0] Non-security[-1]

K-Means clustering was attempted on the TF feature vector; however, many of the security related issues were mislabeled. A distance measure based classification method would perform better for the TF feature vector of this dataset because the datapoints were not represented in the desired way.

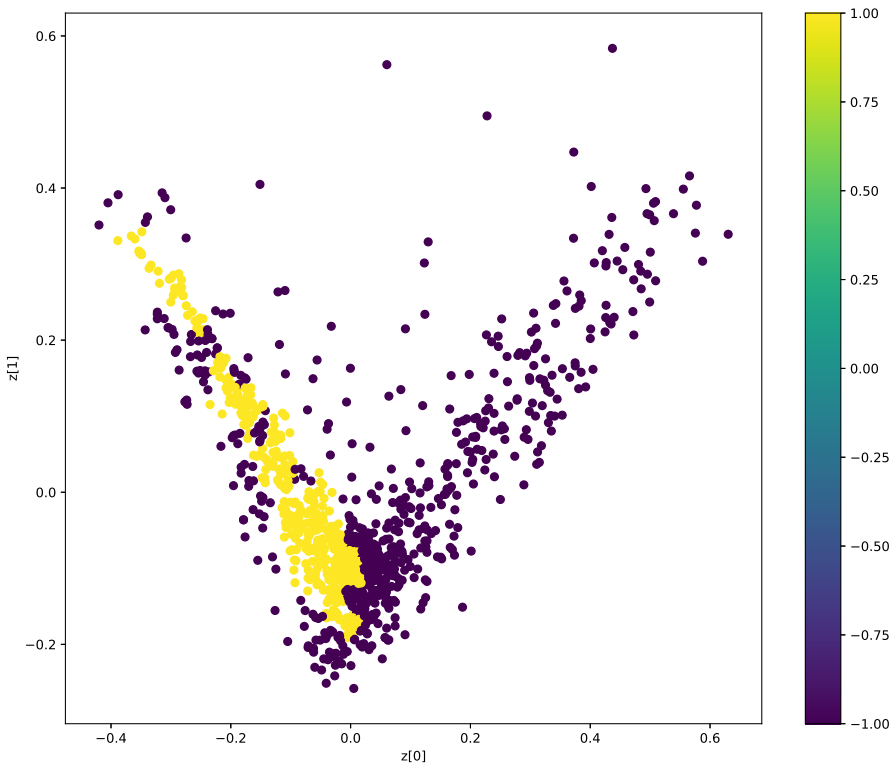

Figure 6.15: Flight Mission Developer Issues K-Means TF PCA Representation: Security[1] Non-security[-1]

6.3.5 Clustering Conclusion

Security bug report classification was explored through clustering. Term frequency feature vectors and variational autoencoder representations were used with K-means clustering.

Research Questions 2, 2(a), and 2(b) were explored.

RQ 2: Can unsupervised classification based on clustering outperform the anomaly detection based on cosine similarity?

(a) How do Term Frequency feature vectors perform in combination with K-Means clustering?

(b) Can Variational Autoencoders be used to represent security related software issues

in a distribution of software bug reports?

1. Clustering through the tested representations and algorithms was not able to outperform the anomaly detection based on cosine similarity.

2. Clustering with Term Frequency feature vectors performs poorly.

   The representation made it difficult to distinguish between security and non-security related issues.

3. Variational Autoencoders do not distribute software issues well enough for K-means clustering.

   Variational autoencoders do represent portions of security related software issues similarly. CWE-888 descriptions are closely related in the variational autoencoder representations.

For the datasets, cosine similarity was able to perform consistently well. One reason is because the CWE-888 list did not start as a distribution. A software issue needs to be similar to just one CWE-888 description to be considered security related. In clustering, the security software issues need to form clusters where CWE-888 descriptions would also be represented. The second reason is that, in cosine similarity, the security related issues do not need to be very similar. Security issues need to be more similar to CWE-888 descriptions. In order for a software issue to fall into a security cluster it needs to be considered very similar. However, in a different application this property may aid in the problem of selecting a threshold. Unsupervised cosine similarity works by selecting a threshold based on the total scores. If text documents that were not even considered software issues were tested, many issues will still be considered security related under cosine similarity. However, with clustering, if none of the security issues are security related, there is lower chance of labeling false positives.

6.4 Threats to Validity

The same Internal, External, and Conclusion validity threats discussed in Section 5.5 apply to this section.

Construct validity threats concerned with measuring what we intend to measure arose during the experiment. In order to classify software issues as security related, a k value is needed. The same k value was used across all datasets for consistency. Also, clustering is sensitive to noisy datapoints. Outliers may push clusters away from centers. Variational autoencoders are random and a new representation will be generated each experiment. The autoencoders where trained on the complete dataset, so overfitting may affect the distribution; however, it was a unsupervised approach with no software issue labels used.

Chapter 7

Deep Learning Classification

Neural networks have the advantage of mapping complex functions of an input to a targeted output. For instance, we can build a machine that takes in documents (bug reports) and outputs a score vector consisting of non-security and security related scores. The bug reports that are security related should have a higher security related score. The network must be trained using bug reports for this outcome to occur. Neural networks use an objective function that measures the error between the output scores and the desired pattern. The network adjusts its parameters, called weights, after each data input to reduce the error. The learning algorithm computes a gradient vector which indicates how much the error will increase or decrease based on the change of a weight. The weight is adjusted in the opposite direction of the gradient vector. The objective function averaged over the training data shows slopes of the minimums [22].

Although neural networks may be designed around the structure of the input data, document-term matrices such as Term Frequency feature vectors are generally not used. Instead, the sentences are broken up by spaces, separating each word or phrase using a tokenizer [4]. Each word was represented by an integer relating to the index of that word in the vocabulary. Using a sequence of word indexes reduces the dimensionality of the input. With a feature vector, the length of the input will be the size of the vocabulary. Using a sequence of word indexes, the length was the size of the largest sentence or a fixed sized sentence. More advanced word vectors use a sequence of integers as a starting point. The sequence of indices will allow word order to affect the learning. It may also be used to create word

vectors and word embedding that attempt to structure the vectors to represent information from the natural language.

Research Questions 3, 3(a), and 3(b) are addressed in this section.

RQ 3: Can deep learning outperform traditional supervised machine learning techniques?

(a) How do supervised deep neural networks compare to traditional supervised machine learning techniques?

(b) How do semi-supervised deep neural networks compare to supervised learning techniques?

7.1 Background on LSTM Networks

Recurrent neural networks (RNN) add the benefit of taking into account history of an input. This can be very beneficial in scenarios where the data is a sequence or time series. There is a benefit in using recurrent neural networks with text data because the semantics of a sentence comes from the words, order or words, and structure. However, whenever there is a large gap between inputs, RNNs have trouble connecting the information.

Long Short Term Memory (LSTM) networks are designed to solve the problem of long term dependencies [16]. The LSTM networks allow the cells to store states where the network decides what gets added and what gets forgotten. Recurrent network modules have one single neural network layer. LSTM cells are composed of four different neural network layers. Three of these layers act as gates to decide what to forget, what to pass on to the input, and what to output. Finally there is a *tanh* layer to act as an activation layer.

Sequence autoencoders may be used as a pretraining stage to supervised learning as shown by the work done by Dai [5]. This method is believed to stabilize the parameters of the supervised model as well as the added benefit from the generalization from unlabeled data.

Language Models follow the same structure; however, the encoder portion of the network is removed after training to return a feature representation. A language model could be used as a feature vector to different machine learning algorithms such as Naive Bayes.

7.2 LSTM Network Proposed Approach

The LSTM network architecture shown in Figure 7.1 was adapted from Dai et al [5]. An implementation is made using python and Keras [4]. The word embedding layer is trained so that the input features are related based on term similarity. The goal of the embedding layer is to map the semantic meaning into a geometric space. The software issues are converted to a sequence of integers and used as the input of the network. The network is trained to classify the issues as security related or non-security related. The network was trained on 10 epochs. The loss function "binary_crossentropy" and the optimizer "Adam" were used.

For a LSTM network, the new weights are trained on 90% of the data in the fold. The remaining 10% of data is used for testing. During training, 10% of the training data is used for validation testing to prevent over fitting. The training data was forced to be balanced between security and nonsecurity issues by undersampling non-security related issues. Undersampling did not affect the testing data.

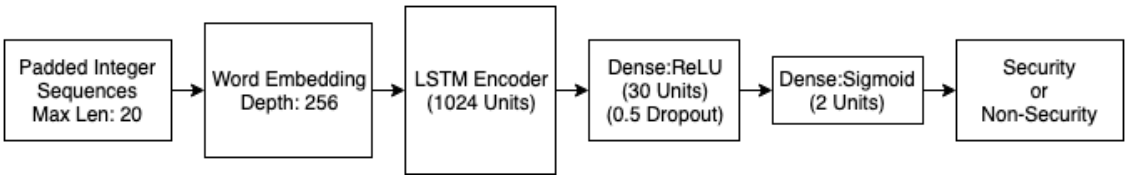

Figure 7.1: LSTM Network Architecture [5]

A similar sequence autoencoder network to Dai et al [5], uses an autoencoder network consisting of LSTM layers where the output is the same as the input. The network is trained to reconstruct the original inputs. The network can be trained as an unsupervised approach, making the approach semi-supervised. Weights of the network can be used as the weights of a pretrained supervised model such as the architecture in Figure 7.1. The architecture was adapted from Dai et el. [5].

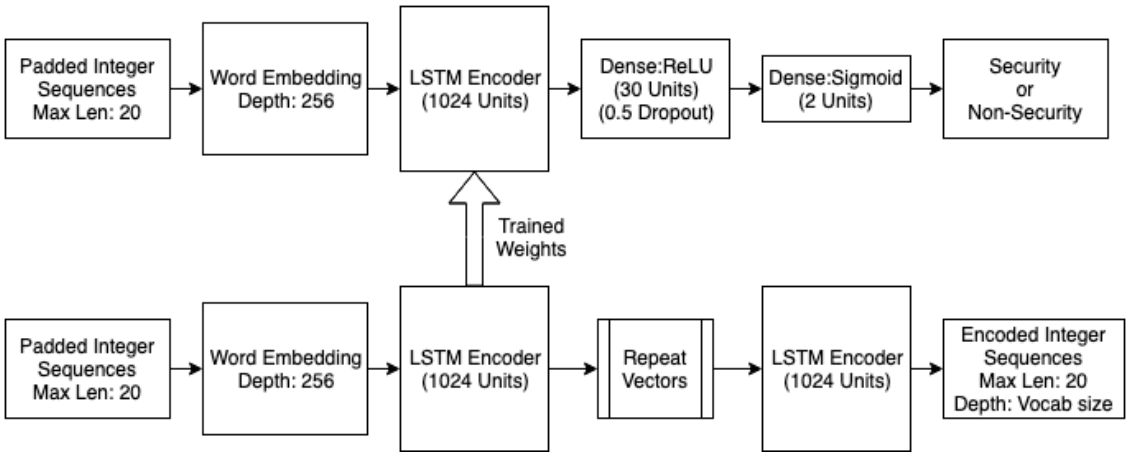

Figure 7.2: Sequence Autoencoder LSTM Network Architecture [5]

One sequence autoencoder used on the Red Hat Enterprise Linux 4 dataset was trained on a unlabeled subset of 30% of the Red Hat dataset. The NASA projects were smaller, so one sequence autoencoder was trained on 3,000 unlabeled software issues from other NASA IV&V datasets. After the sequence autoencoder was trained, the weights were transferred to a LSTM Network which was then trained on labeled data creating a semi-supervised approach. All NASA projects use the weights from the same sequence autoencoder which are then tuned in different LSTM networks.

7.3 LSTM Networks Results

Supervised and semi-supervised classification techniques were explored. Naive Bayes classification was set as the baseline. Support Vector Machines were also competitive in the classification performance; however, the results varied more and must be tuned specifically to a dataset.

The results are compared in tables with “Naive Bayes”, “LSTM”, and “SA-LSTM”. “SA-LSTM” represents a LSTM network that was pretrained by a sequence autoencoder. Classification results with the highest G-Score value are in bold. Results are also compared in box plots.

7.3.1 Red Hat Enterprise Linux 4 Results

The Red Hat Enterprise Linux 4 classification performance is shown in Table 7.1. The precision metric in the baseline was very low. The classification performance slightly improved across all metrics using a LSTM network. The classification improved even more with a SA-LSTM network. The Red Hat Enterprise Linux 4 project was the only project which trained a sequence autoencoder on a separate subset of the dataset. The precision metric improved by 19.3% compared to the baseline, while recall degraded by only 2.8%. A box plot comparing the results is shown in Figure 7.3

Table 7.1: Red Hat Enterprise Linux 4 Project Supervised and Semi-supervised Classification Results

Classifier	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Naive Bayes	*TF*	73.5%	11.8%	84.3%	73.0%	20.7%	78.2%
LSTM	*Sequence*	75.8%	13.0%	85.8%	75.4%	22.6%	80.3%
SA-LSTM	***Sequence***	**78.7%**	**14.1%**	**81.9%**	**78.6%**	**24.0%**	**80.2%**

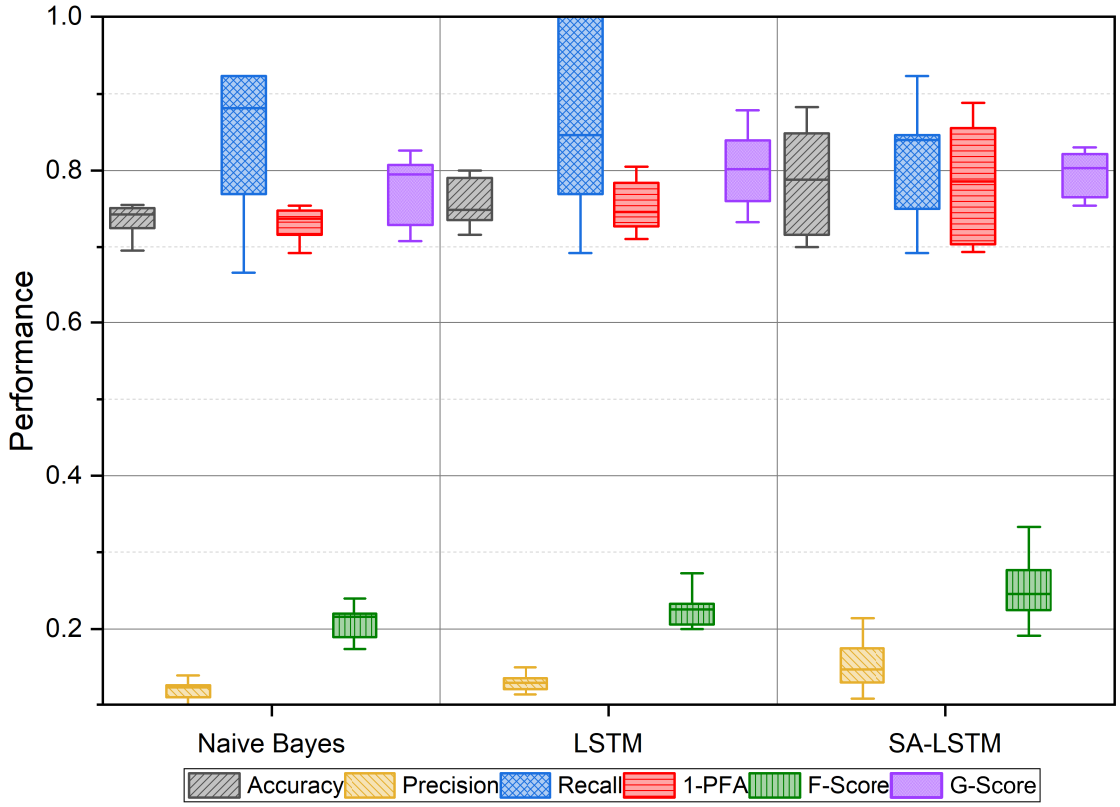

Figure 7.3: Red Hat Linux 4 (Half) Supervised Learning Box Plot

7.3.2 Ground Mission IV&V Issues Results

In the Ground Mission IV&V Issues project the LSTM network performance was worse than the Naive Bayes performance shown in Table 7.2. Naive Bayes outperformed all of the tested classifiers in the baseline including Support Vector Machines. The Ground Mission IV&V Issues project had a small amount of security related issues to learn from. The sequence autoencoder was not trained on unlabeled data from the Ground Mission IV&V Issues project. A box plot comparing the results is shown in Figure 7.4

Table 7.2: Ground Mission IV&V Issues Supervised and Semi-supervised Classification Results

Classifier	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Naive Bayes	***TF***	**84.7%**	**31.8%**	**84.2%**	**84.7%**	**46.2%**	**84.4%**
LSTM	*Sequence*	65.9%	17.0%	87.1%	64.2%	28.5%	73.9%
SA-LSTM	*Sequence*	71.5%	17.4%	70.5%	71.6%	27.9%	71.1%

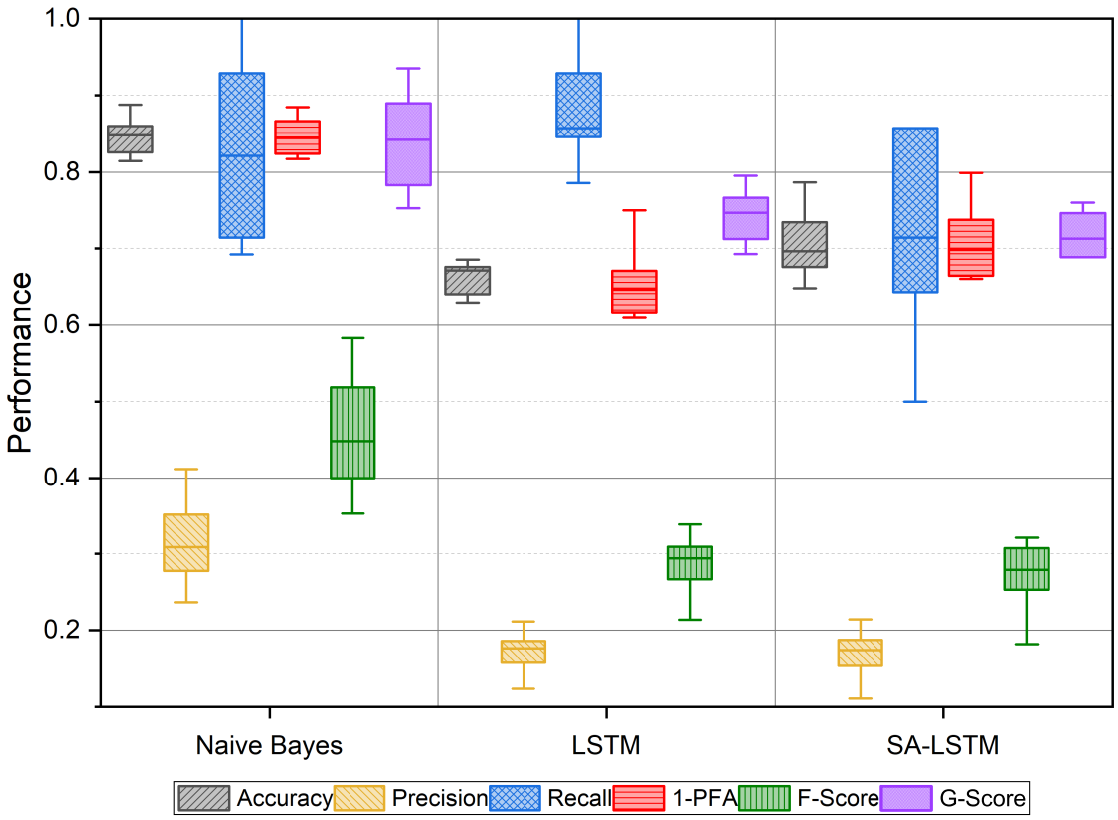

Figure 7.4: Ground Mission IV&V Issues Supervised Learning Box Plot

7.3.3 Flight Mission IV&V Issues Results

The LSTM network outperformed Naive Bayes in all performance metrics as shown in Table 7.3. The SA-LSTM network which was trained on unlabeled data from other IV&V missions performed the worst. A box plot comparing the results is shown in Figure 7.3.

Table 7.3: Flight Mission IV&V Issues Supervised and Semi-supervised Classification Results

Classifier	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Naive Bayes	*TF*	78.1%	71.8%	77.2%	78.7%	74.4%	77.9%
LSTM	***Sequence***	**82.5%**	**74.3%**	**88.0%**	**78.7%**	**80.6%**	**83.1%**
SA-LSTM	*Sequence*	71.8%	63.0%	76.6%	68.4%	69.1%	72.3%

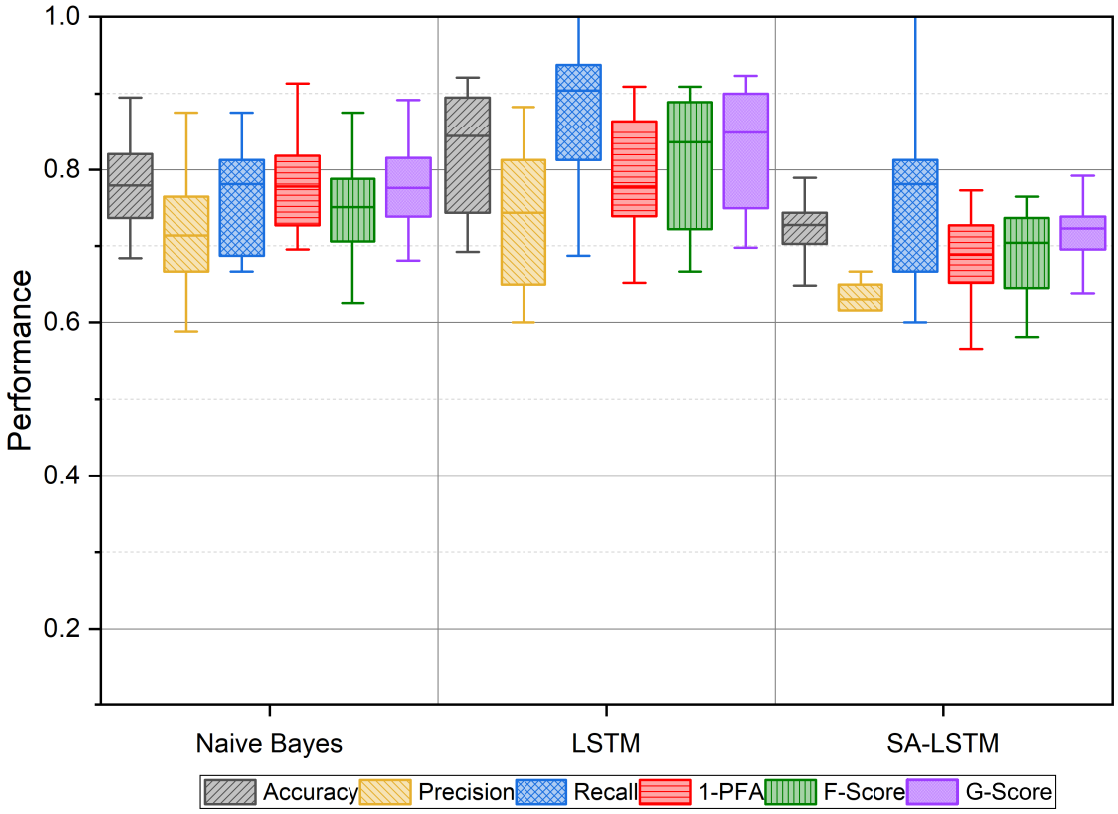

Figure 7.5: Flight Mission IV&V Issues Supervised Learning Box Plot

7.3.4 Flight Mission Developer Issues Results

The Flight Mission Developer Issues project results are shown in Table 7.4. Classification through Naive Bayes resulted in the highest G-Score value. However, recall improved to 92% from 57% at the expense of a high probability of false alarm. We believe the sequence autoencoder network did poorly because it was not trained on unlabeled data from the same dataset. A box plot comparing the results is shown in Figure 7.6.

Table 7.4: Flight Mission Developer Issues Supervised and Semi-supervised Classification Results

Classifier	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Naive Bayes	***TF***	**58.7%**	**74.2%**	**57.0%**	**62.1%**	**64.4%**	**59.4%**
LSTM	*Sequence*	73.5%	74.0%	92.0%	37.9%	82.0%	53.7%
SA-LSTM	*Sequence*	65.9%	67.3%	93.6%	12.8%	78.3%	22.6%

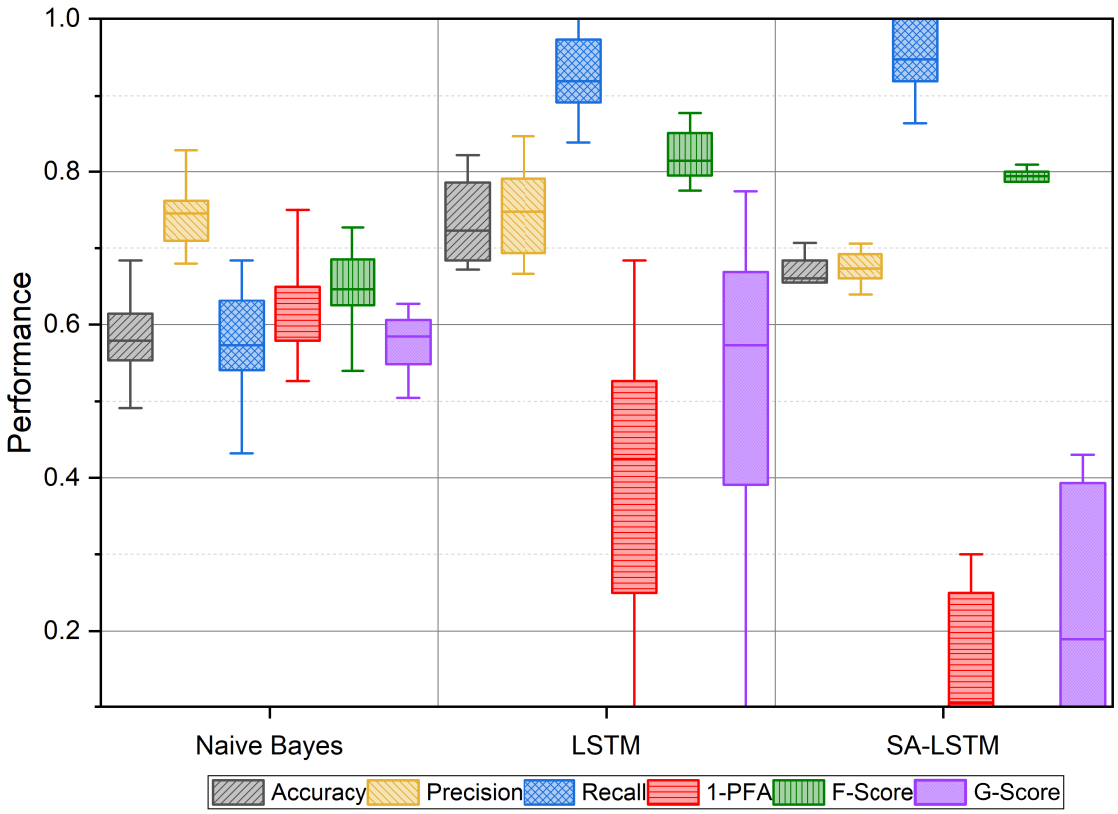

Figure 7.6: Flight Mission Developer Issues Supervised Learning Box Plot

7.4 LSTM Networks Conclusion

Supervised and semi-supervised classification methods were explored with LSTM networks. The semi-supervised approach used sequence autoencoders which may be trained on unlabeled data and tuned on labeled data. The overall improvements compared to Naive Bayes with Term Frequency feature vectors are shown in Table 7.5.

Table 7.5: LSTM Improvement of Classification Performance Compared to Naive Bayes

Dataset	*Classifier*	*Feature Vector*	*Accuracy*	*Precision*	*Recall*	*1-PFA*	*F-Score*	*G-Score*
Red Hat Enterprise Linux 4	*SA-LSTM*	*Sequence*	78.7%	14.1%	81.9%	78.6%	24.0%	80.2%
		Improvement	+7.1%	+19.3%	-2.8%	+7.6%	+16.1%	+2.5%
Ground Mission IV&V Issues	*LSTM*	*Sequence*	65.9%	17.0%	87.1%	64.2%	28.5%	73.9%
		Degradation	-22.1%	-46.4%	+3.4%	-24.3%	-38.2%	-12.5%
Flight Mission IV&V Issues	*LSTM*	*Sequence*	82.5%	74.3%	88.0%	78.7%	80.6%	83.1%
		Improvement	+5.7%	+3.6%	+13.9%	0.0%	+8.3%	+6.6%
Flight Mission Developer Issues	*LSTM*	*Sequence*	73.5%	74.0%	92.0%	37.9%	82.0%	53.7%
		Improvement	+25.1%	-0.3%	+61.5%	-38.8%	+27.2%	-9.5%

Research Questions 3, 3(a), 3(b) were explored.

RQ 3: Can deep learning outperform traditional supervised machine learning techniques?

(a) How do supervised deep neural networks compare to traditional supervised machine learning techniques?

(b) How do semi-supervised deep neural networks compare to supervised learning techniques?

1. Deep learning, specifically LSTM networks may outperform traditional supervised machine learning techniques.

   In three out of the four datasets, LSTM networks outperformed Naive Bayes classification. However, the classification performance degraded in the Ground Mission IV&V Issues dataset

2. Deep neural networks show higher recall scores compared to the Naive Bayes baseline.

   The recall value in the Flight Mission Developer Issues dataset improved by 61.5% compared to the baseline. The recall value in the Flight Mission IV&V Issues dataset improved by 13.9% without any other metrics degrading.

3. Semi-supervised deep neural networks map improved classification performance if trained on similar data to the dataset.

An LSTM sequence autoencoder was trained on a subset of the Red Hat Enterprise Linux 4 dataset and showed an improvement in precision by 19.3% compared to baseline. However, when the sequence autoencoder was not trained on the similar software issues dataset the results degraded compared to both supervised machine learning methods and supervised deep learning methods.

7.5 Threats to Validity

The same Internal, External, and Conclusion validity threats discussed in Section 5.5 apply to this section.

Construct validity threats concerned with measuring what we intend to measure arose during the experiment. Under sampling is performed on the training dataset for the LSTM networks; however, it was not used for Naive Bayes classification. Sequence autoencoders are trained on labeled data. We train sequence autoencoders on unlabeled data from similar datasets and unrelated datasets and show both classification performances.

Chapter 8

Conclusion

Discovering and fixing bug reports is an important step in the development and upkeep of software. Not all software issues are caused by vulnerabilities and therefore are not security related. Security related software issues may have higher priority and should be fixed first. This thesis had the goal to explore new solutions to detecting security related bug reports.

One area to improve is the feature vectors, specifically the vocabulary of a feature vector which influences the whole learning process. We applied feature selection on feature vectors to combine the vocabulary of CWE-888 descriptions and the software issues. Feature selection scores words based on the dependence to the classes and selects the words with the highest scores. If words are common across both classes, they are not likely to be selected as they may confuse the learner. We trained feature selection on a validation set of 20% of the dataset. The Red Hat Enterprise Linux 4 dataset showed the greatest improvements with Chi-squared feature selection. The precision values increased by 78% and the F-Score improved by 62.8%. The IV&V Ground and Flight projects' precision metrics improved by 7.1% and 1.8% respectively using mutual information feature selection. The Developer Flight project showed an improvement in precision of 3.6%, and F-Score improved by 11.8%. We believe feature selection can improve precision metrics that are low. Feature selection works best when given a large enough training set.

We used clustering for unsupervised classification of security related bug reports. We used term frequency feature vectors and variational autoencoder embeddings to represent the datasets. K-Means algorithm clustered the representations with CWE-888 descriptions

to be labeled as security related. CWE-888 descriptions were similar to one another in the representations. In the Red Hat Linux 4 dataset and the IV&V Ground and Flight datasets, the majority of security issues clustered together with each other. However, many non-security software issues were present throughout the clusters. The Developer Flight dataset representation showed the software issues were different from the CWE-888 descriptions. Anomaly detection using cosine similarity. which was used in the prior work, outperformed the classification through clustering using the representations and K-means. Future work could improve on the variational autoencoder by using LSTM and convolutional networks. Also, K-Means clustering may be improved by using different approaches to selecting appropriate k values for each dataset.

LSTM networks, a type of recurrent networks, were used for supervised classification. LSTM networks performed well in all datasets besides the IV&V Ground project, which had a high false positive rate. The IV&V Flight project's classification performance increased with LSTM networks throughout all metrics tested. Although the Developer Flight project's G-Score value decreased due to a higher false positive rate with LSTM networks, the recall value increased by 61.5% and the F-Score increased by 27.2%.

Sequence autoencoders benefit from using unlabeled data with semi-supervised classification. Sequence autoencoders were trained using just the input data where no labels are required. The trained weights were transferred to a supervised network which was tuned to the classification using labeled data. The sequence autoencoders stabilized the network, which led to more consistent results. We trained a sequence autoencoder using 30% of unlabeled issues from the Redhat Linux 4 dataset. The sequence autoencoder improved the precision and F-Score values by 19.3% and 16.1% respectively compared to Naive Bayes classifier. The NASA datasets have fewer issues, so the sequence autoencoder was trained on unlabeled data from different NASA IV&V missions. The sequence autoencoder did not improve the results of the NASA projects, that is, degraded the performance metrics. Sequence autoencoders worked the best when trained on unlabeled data from a validation set or very similar projects. Sequence autoencoders may be resourceful in large unlabeled datasets. The architecture may be improved by using bidirectional networks for generative deep learning.

References

[1] D. Arthur and S. Vassilvitskii, "How slow is the k -means method?" in *Symposium on Computational Geometry*, 2006, pp. 1–10. [Online]. Available: https://www2.cs.duke.edu/courses/spring07/cps296.2/papers/kMeans-socg.pdf

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *CoRR*, vol. abs/1409.0, 9 2014. [Online]. Available: http://arxiv.org/abs/1409.0473

[3] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[4] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[5] A. M. Dai and Q. V. Le, "Semi-supervised Sequence Learning," pp. 3079–3087, 2015. [Online]. Available: http://papers.nips.cc/paper/5949-semi-supervised-sequence-learning

[6] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders," *arXiv preprint arXiv:1611.02648*, 11 2016. [Online]. Available: http://arxiv.org/abs/1611.02648

[7] C. Doersch, "Tutorial on Variational Autoencoders," *arXiv preprint arXiv:1606.05908*, 6 2016. [Online]. Available: http://arxiv.org/abs/1606.05908

[8] C. Elkan, "Using the Triangle Inequality to Accelerate-Means," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003. [Online]. Available: https://www.aaai.org/Papers/ICML/2003/ICML03-022.pdf

[9] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 5 2010, pp. 11–20. [Online]. Available: http://ieeexplore.ieee.org/document/5463340/

[10] K. Goseva-Popstojanova and J. Tyo, "Experience Report: Security Vulnerability Profiles of Mission Critical Software: Empirical Analysis of Security Related Bug Reports," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 10 2017, pp. 152–163. [Online]. Available: http://ieeexplore.ieee.org/document/8109082/

[11] ——, "Identification of Security Related Bug Reports via Text Mining Using Supervised and Unsupervised Classification," *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 344–355, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8424985/

[12] M. Hamill and K. Goseva-Popstojanova, "Common Trends in Software Fault and Failure Data," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 484–496, 7 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4760152/

[13] ——, "Exploring the missing link: an empirical study of software fixes," *Software Testing, Verification and Reliability*, vol. 24, no. 8, pp. 684–705, 9 2014. [Online]. Available: http://doi.wiley.com/10.1002/stvr.1518

[14] ——, "Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system," *Software Quality Journal*, vol. 23, no. 2, pp. 229–265, 6 2015. [Online]. Available: http://link.springer.com/10.1007/s11219-014-9235-5

[15] F. Hill, K. Cho, and A. Korhonen, "Learning Distributed Representations of Sentences from Unlabelled Data," *arXiv preprint arXiv:1602.03483*, 2 2016. [Online]. Available: http://arxiv.org/abs/1602.03483

[16] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735

[17] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[18] Keras, "Variational autoencoder - Keras Documentation." [Online]. Available: https://keras.io/examples/variational_autoencoder/

[19] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv preprint arXiv:1312.6114*, 12 2013. [Online]. Available: http://arxiv.org/abs/1312.6114

[20] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-Thought Vectors," pp. 3294–3302, 2015. [Online]. Available: http://papers.nips.cc/paper/5950-skip-thought-vectors

[21] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," *International conference on machine learning*, 5 2014. [Online]. Available: http://arxiv.org/abs/1405.4053

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015. [Online]. Available: http://www.nature.com/articles/nature14539

[23] P. Meesad, P. Boonrawd, and V. Nuipian, "A chi-square-test for word importance differentiation in text classification," in *Proceedings of International Conference on Information and Electronics Engineering*, 2011, pp. 110–114.

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint arXiv:1301.3781*, 1 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[25] MITRE, "CVE - Home." [Online]. Available: https://cve.mitre.org/about/index.html

[26] ——, "CWE - About CWE." [Online]. Available: https://cwe.mitre.org/about/index.html

[27] ——, "CWE - CWE-888: Software Fault Pattern (SFP) Clusters (3.2)." [Online]. Available: https://cwe.mitre.org/data/definitions/888.html

[28] NLTK Project, "Natural Language Toolkit — NLTK 3.4 documentation." [Online]. Available: http://www.nltk.org/

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] F. Peters, T. Tun, Y. Yu, and B. Nuseibeh, "Text Filtering and Ranking for Security Bug Report Prediction," *IEEE Transactions on Software Engineering*, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8240740/

[31] L. Poddar, L. Neves, W. Brendel, L. Marujo, S. Tulyakov, and P. Karuturi, "Train One Get One Free: Partially Supervised Neural Network for Bug Report Duplicate Detection and Clustering," *arXiv preprint arXiv:1903.12431*, 3 2019. [Online]. Available: http://arxiv.org/abs/1903.12431

[32] J. P. Tyo, B. M. Statler, K. Goseva-Popstojanova, C. S. Roy Nutter, and M. C. Valenti, "Empirical Analysis and Automated Classification of Security Bug Reports," Master's thesis, West Virginia University, 2016. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=20160014477

[33] G. van Rossum and F. L. Drake, "Python Library Reference," *Cheat Sheets*, no. March, 2006. [Online]. Available: https://www.python.org/

[34] T. Wen, Y. Zhang, Q. Wu, and G. Yang, "ASVC: An Automatic Security Vulnerability Categorization Framework Based on Novel Features of Vulnerability Data," *Journal of Communications*, vol. 10, no. 2, pp. 107–116, 2015. [Online]. Available: https://pdfs.semanticscholar.org/504d/32ebbdba999c5737d0d75cf13b6268925e67.pdf

[35] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[36] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou, "Automated Configuration Bug Report Prediction Using Text Mining," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 107–116. [Online]. Available: https://issues.apache.org/jira/browse/ACCUMULO

[37] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick, "Improved Variational Autoencoders for Text Modeling using Dilated Convolutions," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 3881–3890. [Online]. Available: https://arxiv.org/pdf/1702.08139.pdf

[38] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs," in *Proceeding of the 8th working conference on Mining software repositories - MSR '11*. New York, New York, USA: ACM Press, 2011, p. 93. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1985441.1985457