Automatic Vulnerability Classification Using Machine Learning

Marian Gawron(✉), Feng Cheng, and Christoph Meinel

Hasso Plattner Institute (HPI), University of Potsdam, 14482 Potsdam, Germany
{marian.gawron,feng.cheng,christoph.meinel}@hpi.de

Abstract. The classification of vulnerabilities is a fundamental step to derive formal attributes that allow a deeper analysis. Therefore, it is required that this classification has to be performed timely and accurate. Since the current situation demands a manual interaction in the classification process, the timely processing becomes a serious issue. Thus, we propose an automated alternative to the manual classification, because the amount of identified vulnerabilities per day cannot be processed manually anymore. We implemented two different approaches that are able to automatically classify vulnerabilities based on the vulnerability description. We evaluated our approaches, which use Neural Networks and the Naive Bayes methods respectively, on the base of publicly known vulnerabilities.

Keywords: Vulnerability analysis · Security analytics · Data mining Machine learning · Neural Networks

1 Introduction

Nowadays, the overall number of possible combinations of applications and operating systems and the complexity of each piece of software results in an inability to manually survey the configuration of modern systems. Thus, the maintenance and recognition of all components and their reported vulnerabilities requires a tremendous effort. In the current situation, the huge amount of vulnerabilities complicates the administration and protection of modern IT infrastructures. Therefore, it is desirable to automatically process vulnerability information. Common Vulnerability Scoring System (CVSS) parameters [7] of the vulnerability are usually used to enrich vulnerability information with additional metrics that allow automatic processing. Some of the parameters can be used to derive an estimation of the severity and the effect of a vulnerability. In particular, the attack range and the impact on the basic security principles, namely `availability`, `confidentiality`, and `integrity`, are crucial to analyze the vulnerability.

The identification and assignment for the CVSS [7] base metrics is a time-consuming action that requires expert knowledge. Usually, professional analysts from the National Institute of Standards and Technology (NIST) have to perform this scoring manually [4]. They try to match the new vulnerability to a predefined template. But if the description of the vulnerability is ambiguous, an even more time consuming manual analysis has to be performed. Then, already analyzed vulnerabilities are identified that have similar information or keywords in their descriptions. The scores and attributes of these vulnerabilities are used as a guidance to perform the final evaluation and scoring of the new vulnerability. This process [4] reveals that also the human experts start their investigation with the textual description of the vulnerability. Commonly used services to report newly identified vulnerabilities, e.g. the vulnerability report form of Carnegie Mellon [2], limit the information to a textual description. So, the human experts do not initially receive more comprehensive information. Although they might benefit from additional background information from the common knowledge, we can identify a direct dependency between textual description and CVSS attributes within the evaluation procedure of a new vulnerability.

Since this scoring method requires advanced knowledge and a manual investigation, the scoring procedure sometimes leads to a delay in the attribution. Consequently, there are vulnerabilities, which have been released without CVSS metrics. The missing metrics are usually integrated later. This duration could be crucial for analytic systems that rely on these metrics to evaluate or even detect the vulnerability. The delay of the scoring could amount to several days. For example, the vulnerability in OpenSSH with the identifier CVE-2016-0777 was published on the 14th of January in 2016, whereas the CVSS attributes were released on the 19th of January 2016. Thus, there have been 5 days without the possibility of a classification or an automatic processing of the vulnerability. This delay could also not be explained with the low usage and minor distribution of the software, since OpenSSH is a commonly used program, which is also pre-installed in many Linux distributions. We believe that this delay and the manual workload of the vulnerability analysts could be dramatically reduced with an automatic classification of the vulnerability. Therefore, we will propose machine learning approaches to perform the classification based on the textual description.

The paper will begin with an introduction to illustrate the problem and its importance. Afterwards, some related ideas and approaches will be introduced and described. However, the problem is currently not widely explored, which results in the lack of numerous approaches that tackle the problem. Then, our own approaches will be explained in Sect. 3. We will describe a Neural Network approach and the Naive Bayes approach that both learn to classify the vulnerabilities based on extracted features of the description. In Sect. 4 our evaluation results are presented and illustrated. Thereafter, we will mention additional steps, which we plan to pursue in the future. Finally, we will conclude our work and summarize the contributions.

2 Related Work

Early ideas to use textual data, which is accumulated in vulnerability databases was already presented in [15]. The authors refer to different data mining algorithms to benefit from the diverse knowledge inside various vulnerability databases. However, they state that immense effort should be invested into the normalization and compatibility to benefit from vulnerability information of different sources. Furthermore, they want to use data mining to be able to predict and avoid vulnerabilities in future software products. The authors of [15] claim that learned classifiers should be able to identify vulnerability patterns in software before the code is included in productive software. Therefore, formal vulnerability characteristics should be constructed to train the mining algorithms.

Another application of machine learning to the field of vulnerabilities could be observed in [13]. The authors applied text mining algorithms to the source code of different software components. Then, they predict the likelihood of each component to contain vulnerabilities. This method allows them to highlight components that should be reviewed by security analysts, since it is very likely that the components are vulnerable. In [6], another method to identify vulnerabilities in software is presented. Beside the source code evaluation, the authors of [6] also propose the usage of metrics of version control systems and the architecture of the software itself to improve the prediction of vulnerabilities. The authors of [10] also investigated the possibility to apply machine learning algorithms on vulnerabilities. They tried to predict vulnerabilities in software components based on similarities to other software components. They also integrated the approach into an existing tool, which is called Vulture. The major difference is that they used vulnerability databases as a source to find vulnerabilities in programs, which they identified by using the CPE-IDs (Common Platform Enumeration IDs) [8]. They did not try to predict characteristics of the vulnerability itself.

Another paper utilized machine learning approaches to predict the likelihood for the existence of an up-to-now undiscovered vulnerability in a piece of software using data from the National Vulnerability Database (NVD) [9]. The authors of [18] applied several machine learning algorithms, but concluded that the prediction capability of their models based on the data from NVD is poor. Their goal to build a prediction model for the time-to-next disclosed vulnerability per application was unfeasible as they encountered several problems. For instance, they found that new vulnerabilities affect all previous versions, which results in an unusable `versiondiff` feature. Finally they conclude that the data from NVD might not be designed to be used for machine learning approaches at all.

Furthermore, an interesting approach of vulnerability classification and identification is presented in [17]. The authors utilized text mining methods to identify characteristics in the description of vulnerabilities and bugs. Their idea is to classify bug reports in two categories, namely regular bugs and hidden impact bugs. The major difference is that the hidden impact bugs could violate security policies. Thus, these hidden impact bugs are usually related to vulnerabilities. So, the procedure can be used to identify vulnerabilities from bug reports.

The classification of vulnerabilities was also investigated in [1]. The authors tried classify vulnerabilities based on the fact if the vulnerability was exploited or not. In contrast to our approach they based their findings on a different subset of vulnerabilities and used the OSVDB (Open Source Vulnerability Database). They concentrated on the fact if an exploit is available or if it is rumored or unavailable, which dramatically decreases the amount of usable data. In addition, they also regarded textual features as binary representations in a bag-of-words style. So they will have a high-dimensional vector for textual attributes that represents the words that are present in the corresponding field. This approach is similar to our approach, which was used in the preprocessing step for Naive Bayes. For our Neural Network we did not simply chose a binary representation for the words but the GloVe [11] tool to also retain semantic meanings.

3 Classification Approach

The classification of vulnerability information is a fundamental step in the process of an automatic processing. As it was described, the information about newly identified vulnerabilities rarely contains sufficient details to automatically process the impact of an exploit or fully understand the prerequisites. Usually the reports contain a description of the vulnerability, information about the affected software, and a rough textual description about a possible impact or even less information. The descriptions are submitted in natural language and information about the affected product could be specified by CPE-IDs (Common Platform Enumeration Identifiers) [8] or by keywords. A well-known example is the submission form of the CERT [2]. This process requires a manual investigation and translation to formal vulnerability attributes, such as CVSS metrics [7]. The formal metrics could then be used in an automatic analysis of existing vulnerabilities, since the information can be interpreted by machines.

Our idea aims at an automation of this translation to formal attributes. Therefore, we wanted to investigate the possibilities of a machine-learning approach to derive the formal attributes from the description of the vulnerability. We wanted to solve two interesting questions. The first question was if it is possible to derive the previously mentioned characteristics. The other question was, if the accuracy of the automated approach is sufficient to classify the vulnerability autonomously. We implemented two different classification methods to evaluate the accuracy and discuss advantages between the approaches. We will begin with our Neural Network approach and introduce the Naive Bayes classification afterwards.

3.1 Neural Network

We chose a Neural Network as one of our approaches, since it is a common choice for multi-label classification and classification problems with a high complexity. We believe that the approach is well suited to work with descriptions in natural

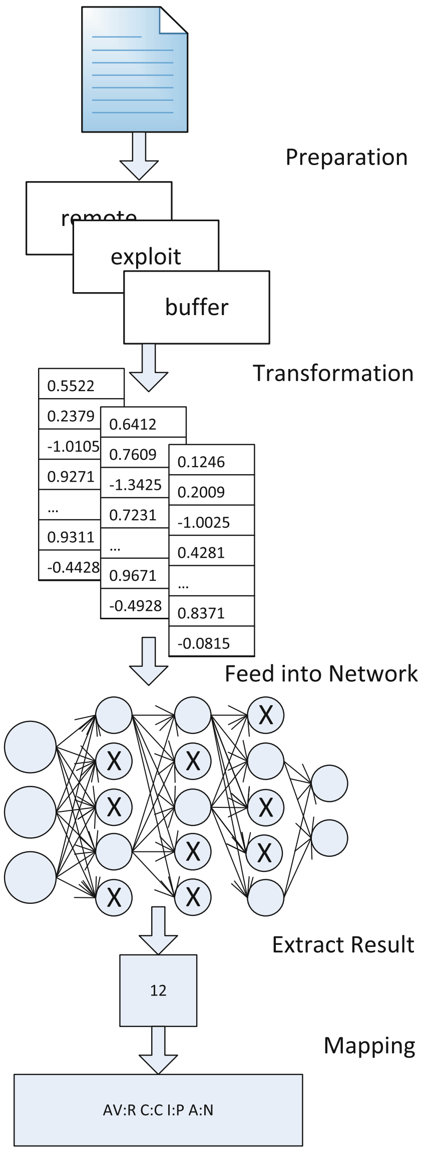

Fig. 1. Workflow of the classification

language if the domain is limited and a sufficient amount of training data is available. In our case both requirements can be met with already classified vulnerabilities. We can use the descriptions of 72,000 older vulnerabilities to train our network. Additionally, we found that the number of distinct words in all descriptions accumulate to 104,000 words. Thus, we could build our limited vocabulary and collected already classified data for our training set. The overall workflow to classify vulnerabilities based on their description using Neural Networks is illustrated in Fig. 1. We will describe the individual steps according to the illustration.

The first step, which was illustrated as the preparation phase in Fig. 1, was to remove unnecessary information from the descriptions. Since the description is created with natural language, it contains words that do not provide any additional benefit to the overall meaning. Thus, we removed stop words with the help of a predefined list of well-known stop words that we gathered from [16]. Furthermore, we used word stemming methods to overcome the differences of words that are produced by declination and conjugation. Additionally, every word with capital characters is transformed to lower case. This will resolve any differences that arise from word orders inside the sentences. So, every description is transformed into a list of substantial words.

In the next step the input data for our network has to be derived from the preprocessed descriptions. This step was integrated into our workflow as the transformation of input values. Since Neural Networks are usually not able to work on textual data, this additional transformation has to be performed. Therefore, we chose GloVe [11], which is an unsupervised learning algorithm to derive vector representation from given input words. The distinctive feature of this algorithm is the capability to identify words with similar meanings and to retain the relationship between multiple pairs of words. For example the difference of the vectors for "king" and "queen" and the difference of the vectors for "man" and "woman" have a high similarity. The algorithm assigns a 50 dimensional vector to each of the identified words. This dictionary can then be used to replace the vectors for each word in the description. In the final step, this algorithm creates a list of vectors out of our previously processed descriptions. Thus, each description consists of 50 floating point values multiplied by the number of words in the description. Even though the average number of words per description is 37, we have to consider the maximum number of words of all descriptions to create our list of word vectors. This requirement arises from the condition of uniform input values to the Neural Network. If we would work with the average number of words, we would have to omit several words for some descriptions, which could lead to a loss of knowledge about those. Thus, we decided to expand all descriptions to the maximum word count. Therefore, we enlarge the shorter descriptions with null vectors to equalize the different lengths of the individual descriptions. For our experiments we identified a maximum length of around 400 words and enlarged smaller descriptions accordingly. So we end up with around 2,000 input values per description that originate from 400 words and 50 floating point values for each word.

Finally, we only have to translate the formal values of the CVSS [7] attributes to numbers to completely work in the numerical space and feed the data into our Neural Network.

This final translation strongly depends on the attributes which should be predicted. First we need to identify all possible values for the selected attribute. Then we can simply enumerate the possible values and use this mapping to translate the textual information to numbers. For the single attribute predictions, such as integrity or attack-range, we identified the distinct values `None, Partial, Complete` and `Unknown, Local, Adjacent Network, Network` respectively.

For our combined predictions the possible combinations of the selected attributes had to be considered. So the combination of confidentiality, integrity, and availability (CIA) results in 27 distinct values. For our overall combination we added the attack-range which means that we would have to work with 108 different values. The mapping itself has to be persisted to allow the back translation, which we perform as our final step of the workflow 1.

At this point we can feed our data into the Neural Network. We decided to use 80% of the data to train the network and the remaining 20% for testing. The Neural Network was configured with a learning rate of 0.001 and it runs for 100 epochs. Additionally, we tested various configurations for the number of hidden layers as well as the number of units per layer. The most promising results have been achieved with a network of 3 layers and 1200, 700, and 50 units for the respective layers. During the evaluation, which will be described in the next chapter, we encountered the problem of overfitting of our network. We tackled the problem with the specification of a dropout rate. This rate describes a probability that an individual node is kept or dropped from propagation. Thus, not all nodes of each layer influence the output anymore. This method is also illustrated in the Neural Network of our workflow in Fig. 1. It is shown that the nodes marked with an “X” do not propagate their results to the following layers of the network.

Afterwards the results are extracted and the numerical values can be mapped back to human readable values of the CVSS attributes. Thus, we can create the single attribute value or the combination of attributes from a description. So we can use the approach to automatically classify vulnerabilities according to the specified attribute. This final step concludes the workflow that is illustrated in Fig. 1.

3.2 Naive Bayes

Beside Neural Networks, we implemented another method to derive CVSS [7] attributes of vulnerabilities automatically. We implemented a Naive Bayes [12] classifier that can predict the selected attribute based on the associated description. We chose the Naive Bayes classification method, since we believe that it is a widely known and applied algorithm for classification problems. It was already thoroughly studied and investigated and could therefore serve well as a comparison and reference value to the Neural Network approach.

First of all, we have to perform some preparation steps similar to the Neural Network approach. We determine the most meaningful words of the descriptions from all vulnerabilities depending on the individual feature. Therefore, we choose to create our features based on a bag-of-words model that uses the existence of words in descriptions. The overall dictionary was created out of 72,000 descriptions resulting in a list of around 104,000 words. In an initial training round the meaningful words are collected automatically. Hence, the corpus of all vulnerability descriptions has to be cleansed from stop-words and we also use word stemming to resolve differences that arise by declination and conjugation. Afterwards the remaining words are ranked by importance, which is measured by the impact that an existence of a word in a vulnerability description has to the classification result. Therefore, we separated the previously cleansed descriptions based on the value of the attribute that should be predicted. So we create one set of descriptions that is correlated with a remote attack range and a different set that is correlated with a local attack range for example. Afterwards, we compute the term frequency and the inverse document frequency to find the most promising candidates that could indicate a special value of the attribute. We believe that the term frequency is a well suited method to find these candidates, since it is more likely that a word has a higher impact on the characteristic if it appears more often in the respective set of descriptions. Another observation was based on the differentiation of descriptions based on the attribute that should be investigated. We found that words with a high frequency in one group of descriptions and a low frequency in the other group of descriptions have a strong impact on the classification. This observation can also be generalized since we know of the effect of domain specific vocabulary and keyphrase extraction [3]. In natural languages we usually use a specific vocabulary to describe scenarios in a certain domain. Thus, it is possible to evaluate the utilized vocabulary to derive the domain and try to predict the meaning of the description. So we compared the term frequencies group wise and selected the words with the highest discrepancies. For example, words like “remote”, “message”, or “connection” have a strong indication that the attack range of a vulnerability is remote. Whereas, words, such as “crash” or “denial” suggest a violation of availability.

For the final classification we found out that using the 500 most significant words per attribute is sufficient to achieve satisfying results. This limitation also reduces the computation time dramatically. The approach of an automatically generated list of meaningful words has the benefit that the features are comprehensible and a manual verification is possible. The final features that are used within the Naive Bayes classifier are binary representations of the existence of the previously identified meaningful words. So we iterate through the list of our previously created set of meaningful words and flag the existence or the non-existence. These features are then passed to the Naive Bayes classifier to train the model.

The workflow for each description is similar to our Neural Network approach. First we prepare the descriptions by removing stop-words and applying word stemming. Afterwards we transform the individual words to numerical features, which are binary in the Naive Bayes approach. The transformation will produce a binary vector that represents the existence of each meaningful term. Finally we feed the vector into our classifier and receive the classification result in a textual format.

4 Classification Results

We used real descriptions of already published vulnerabilities for our experiment. So we gathered all available vulnerabilities with their descriptions and CVSS [7] attributes that have been published before the 1st of January 2016. Thus, the collection consists of 72,490 vulnerabilities with known CVSS attributes. We used 80% to train the algorithm and 20% of the descriptions as a testing set. The division between training and testing data is performed with a shuffling before the partition to let the algorithm train independent of the chosen training data. Because of this random arrangement in the data the test results may vary for each classifier creation. Thus we executed our classifier creation several times and used the average measurements for our evaluation. Additionally, we used 2,400 vulnerabilities that have been published in 2016 as a validation set to evaluate the trained algorithm. This allows us to test our classifiers more thoroughly on data that was definitely not used in any of the training steps. It is even more important in the Neural Network approach since the testing data will be used to adjust and fine-tune the parameters in each iteration of the network. So the descriptions of the validation set did not have any impact on the training of the Neural Network. When the network has to predict the selected attributes the result represents the accuracy that can also be achieved on newly discovered vulnerabilities.

Our CVSS attribute prediction focused on the most important attributes `attack range`, `availability`, `confidentiality`, and `integrity`. We built separate classifiers for each of these attributes. Furthermore, we implemented a classifier for a combination of confidentiality, integrity, and availability, which we refer to as `CIA` classifier. Finally, a classifier for the combination of all attributes was constructed as well. The combined classifiers were created with separate Neural Networks or Naive Bayes classifiers respectively. We combined the primary values of each attribute that is part of the combination and derived a list of possible value combinations for each attribute. Then, the combinations and the vulnerability descriptions are fed into our algorithms to build the combined classifiers.

Because of the different distributions of the attribute values in our data set, we had to adjust the training data to end up with a balanced set of descriptions for the selected attributes. For example the availability attribute divides the overall amount of around 72,000 vulnerabilities into 17,700 vulnerabilities with a `complete`, 31,900 vulnerabilities with a `partial`, and 22,800 vulnerabilities with a `unaffected` specification. The difference is even more significant in the attack range. More than 80% of the vulnerabilities can be exploited remotely. This imbalance results in a major problem of machine learning capabilities. Usually, the application of machine learning algorithms requires the precondition of a well-balanced set of training data. So, that the algorithm is not influenced by the pure majority of attribute characteristics during the classification process. Thus, we adjust our training set to be more balanced and reduce the number of considered vulnerabilities. For example, our availability classifier will not use all 72,000 descriptions, but it will use 17,700 descriptions from each of the characteristics, namely `complete`, `partial`, and `unaffected`, because of the before mentioned balancing method. This leads to the final training set of 53,100 vulnerabilities, which is well-balanced. Hence, the classification should only depend on the descriptions themselves and not on the difference in the amount.

Furthermore, our results mainly focus on the accuracy of the tested approaches, which slightly varies from the accuracy of a linear classifier that has to decide in a two-dimensional space. Therefore, our accuracy computation was derived from the Hamming score. The classifier accuracy is computed over all labels to correctly evaluate the effectiveness of our multi-label classification as also described in [14]. The formula to compute the accuracy is shown in Eq. 2.

$$CorrectlyClassified = \bigcap_{Label \in SelectedLabels} \{v \in Vulns | v.Label = result\} \tag{1}$$

$$ClassificationAccuracy = \frac{|CorrectlyClassified|}{|AllDescriptions|} \tag{2}$$

4.1 Naive Bayes

The first results and the initial impression, if it is possible to create an automatic classification approach for vulnerability descriptions was produced by a Naive Bayes classification. We chose the Naive Bayes approach because this method is commonly used for classification problems. Additionally, Naive Bayes could also be used to perform a multi-label classification, which was one of our requirements. We also believe that the Naive Bayes with its wide distribution is well suited to be a candidate to compare results from other approaches. We applied the Naive Bayes classification as it was described in Sect. 3.2.

So we created 4 separate classifiers that apply the Naive Bayes approach to each single attribute of our data set. The results are shown in Table 1. As we already expected the results of the training data are similar to the results of the test data. Because we randomly select the 20% test data at the beginning of each execution anew, there is no difference between test and training data. Furthermore, the validation set also produces similar results, which are also promising.

Table 1. Accuracy of the Naive Bayes approach

CVSS attribute	Train data	Test data	Validation data
Attack vector	89.9%	90.8%	92.3%
Availability	68.4%	68.0%	70.0%
Confidentiality	73.2%	72.4%	69.1%
Integrity	74.2%	73.6%	68.3%

Besides the mentioned results, the Naive Bayes classifier also offers an additional metric that describes the confidence of the classification result. We modified our algorithm to only perform the classification if the confidence value amounts to at least 0.75. The results that have been produced using the confidence do not differ significantly from the already listed findings in Table 1. We found that the accuracy of the attack vector classification drops by 2%, whereas the accuracy of the availability, confidentiality, and integrity classifiers increases by 3%. But the modified version omits 56, 419, 428, 321 vulnerabilities respectively. Nevertheless if the Naive Bayes approach would be used to automatically classify the descriptions in practice the confidence value should be included. If the confidence is not satisfying a manual classification should be considered to achieve accurate results.

4.2 Neural Networks

As it was already described in Sect. 3, we implemented a second approach that is based on Neural Networks. Our Neural Network was created with Tensorflow version 0.10[1] and Glove 1.2 [11]. Since we wanted to produce comparable results to our first approach we created different networks for the individual attributes. So, at first we also created one network for each of the four CVSS attributes. Each experiment was executed 10 times and we computed the accuracy according to the formula of Eq. 2. In the case of a single attribute the formula can be reduced to the fraction of correctly classified descriptions and overall number of descriptions. For the Neural Network we divided our input data set into training data and test data, similar to out Naive Bayes approach. We chose 80% of the data for the training set and the remaining 20% for the test data set. The difference is that the test data is already considered during the training of the model. We configured our network to run for 100 epochs. The network itself consists of 3 hidden layers with 1200, 700, and 50 units respectively. The classification results are illustrated in Table 2.

Table 2. Accuracy of the Neural Network approach

CVSS attribute	Train data	Test data	Validation data
Attack vector	99%	88.9%	80.3%
Availability	99%	80.7%	70.0%
Confidentiality	99%	81.1%	70.2%
Integrity	99%	81.9%	69.8%

The results also indicate the problem that arose with the Neural Network approach. As it was already mentioned in Sect. 3, we encountered the overfitting effect that often occurs with Neural Network approaches. The algorithm has

[1] https://www.tensorflow.org/.

been able to adjust too much to the training data. Thus, the high accuracy for the training data results in a poor accuracy of the test data. A commonly used method against overfitting is the utilization of a dropout rate. We described the working principle of the dropout rate in Sect. 3 and it is also illustrated in Fig. 1.

4.3 Combined Classifiers

We described the results for the two implemented classifiers in the previous sections and the comparison shows that both approaches are capable to classify vulnerabilities in consideration of single CVSS attributes. The two results for each of the attributes are comparable and therefore neither the Naive Bayes nor the Neural Network could achieve significantly better results than the other approach. But, our overall goal was to create an approach that is able to perform a detailed classification of multiple CVSS attributes. Therefore, we considered the attribute triple of confidentiality, integrity, and availability, which is commonly abbreviated as `CIA`. The results for the CIA-classification are illustrated in Table 3. We produced the results according to previously described methods. The major difference was to create the different value combinations of the attribute triples. Those combinations are considered to construct the possible labels for our data. Then, the workflow will be similar to the single-attribute based approach that was described earlier. So we trained our two classification methods on our training data set and applied the resulting models to the evaluation data set. The process was reiterated 10 times and the average accuracy was computed. Now the computation has to consider the individual number of Labels to compute the correct accuracy, which means we now use the MultiLabelAccuracy from Eq. 2.

Table 3. Accuracy of Naive Bayes and Neural Network on combined attributes

CVSS attribute combination	Naive Bayes	Neural Network
CIA test data	63.9%	71.2%
CIA validation data	51.6%	53.4%
Overall test data	61.4%	59.3%
Overall validation data	48.1%	49.1%

Moreover, the `Overall` classification includes the attack range in the process. Then we are able to produce predictions for `availability`, `integrity`, `confidentiality`, and `attack range`, which is usually sufficient for a deep analysis of the requirements and impacts of a vulnerability. The results are illustrated in Table 3. Finally, we can observe that it is possible to use machine learning techniques for an automated vulnerability classification to save processing time for vulnerability experts. In addition, we can see that the Neural Network performs slightly better in the combined classification.

5 Future Work

The presented approaches represent the first attempts to automatically classify vulnerabilities. This would allow a faster analysis of requirements and impacts of known vulnerabilities. Usually, the time period that is required to manually process the information from the description is crucial, since attackers could also use the information in this time frame. Therefore, we plan to continue this work to achieve still higher accuracy rates and enhance the classification process that it could be applied on public vulnerability databases, such as, National Vulnerability Database (NVD) [9], HPI-VDB [5]. The possibility of an automatic classification of incoming vulnerability descriptions should increase processing performance, because the experts still manually extract the attributes. Then, delays, as the delay of vulnerability CVE-2016-077, should not occur to often anymore.

Furthermore, we could use the trained models on public security forums to find discussions and insights about vulnerabilities. From time to time the community discusses information about vulnerabilities before the vulnerability itself becomes officially or publicly known. Thus, the attackers might have an advantage, since they could already interpret the content and benefit from the gained information. We believe that the trained models could be applied on natural language posts, discussions, or other texts. If a vulnerability classification was possible or if the used vocabulary is similar to the already known vocabulary from other vulnerability descriptions, it is very likely that the topic of the discussion or the text is a vulnerability. Finally, one could use the classification results to evaluate if the discussed vulnerability describes an already known vulnerability or if the description belongs to an unknown vulnerability, which increases the importance considerably.

6 Conclusion

This paper describes an approach to deal with delay in vulnerability classification that is caused by manual interaction. The problem arises because the report about a newly detected vulnerability contains human readable information about the vulnerability. This information will be converted to the vulnerability description in natural language. Nevertheless most of the analytic tools require formal attributes to determine the requirements and the impact of the vulnerability. We propose an automatic approach to classify vulnerabilities and their natural language descriptions into those formal attributes. Therefore, the first goal was to investigate the feasibility of an automated approach and secondly evaluate the accuracy. Thus, we implemented two different approaches that are capable of an automated classification. We used Naive Bayes [12] as one approach, since it is a widely distributed method to solve classification problems. For the other approach we rely on Neural Networks, as modern natural language processing systems utilize Neural Networks as well.

At first, we identified the most important characteristics of vulnerabilities, namely the CVSS attributes `availability`, `integrity`, `confidentiality`, and `attack range`. Then we created our training, testing, and validation data sets and trained our model on the vulnerability description. Both approaches required some additional preprocessing steps, as it was already described in Sect. 3. The application of the Naive Bayes classification has the advantage that it could directly work with natural language, whereas the Neural Network required one additional transformation step. After the training we evaluated the two approaches on the test data set. In addition to the test data set we also created one validation data set, since the Neural Network approach also uses the test data to train the model or adjust the parameters. So the most important evaluation metric is the accuracy of the validation data set, since it was not used during the training procedure. We found that the automated classification is possible and the accuracy for single attributes, which was around 70% to 90% depending on the attribute and the data set, is also satisfying. The important metric of the combined attributes results in lower accuracy, but it was still possible to achieve an accuracy of 60% to 70% on the test data and around 50% on the validation data. Since those attribute combinations accumulate to more than 100 possible combinations we believe that an accuracy of around 50% is an acceptable result. Furthermore it turned out that the result between test data and validation data differ for both methods. This fact could also indicate that the language of vulnerability descriptions in those sets differ as well. It could be possible that the vulnerabilities that have been published later use a different style to describe the impact of the vulnerability. Nevertheless the possibility to benefit from an automated classification approach has been shown.

References

1. Bozorgi, M., Saul, L.K., Savage, S., Voelker, G.M.: Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 105–114. ACM (2010)
2. Carnegie Mellon University: Cert/cc vulnerability report form (2017). Accessed 12 Mar 2017
3. Frank, E., Paynter, G.W., Witten, I.H., Gutwin, C., Nevill-Manning, C.G.: Domain-specific keyphrase extraction. In: 16th International Joint Conference on Artificial Intelligence (IJCAI 99), vol. 2, pp. 668–673. Morgan Kaufmann Publishers Inc., San Francisco (1999)
4. Franklin, J., Wergin, C., Booth, H.: CVSS implementation guidance. Nat. Inst. Stand. Technol. NISTIR-7946 (2014). http://nvlpubs.nist.gov/nistpubs/ir/2014/NIST.IR.7946.pdf
5. Hasso Plattner Institute: HPI vulnerability database (2017). Accessed 26 Mar 2017
6. Hein, D., Saiedian, H.: Predicting attack prone software components using repository mined change metrics. In: Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP, vol. 1, pp. 554–563 (2016)
7. Mell, P., Scarfone, K., Romanosky, S.: Common vulnerability scoring system. Secur. Priv. IEEE **4**(6), 85–89 (2006)
8. Mitre Corporation: CPE - Common Platform Enumeration (2017). Accessed 11 Mar 2017

9. National Institute of Standards and Technology: National vulnerability database (2017). Accessed 22 Feb 2017
10. Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A.: Predicting vulnerable software components. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 529–540. ACM (2007)
11. Pennington, J., Socher, R., Manning, C.D.: GloVe: global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543 (2014)
12. Russell, S.J., Norvig, P., Canny, J.F., Malik, J.M., Edwards, D.D.: Artificial Intelligence: A Modern Approach, vol. 2. Prentice Hall, Upper Saddle River (2003)
13. Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W.: Predicting vulnerable software components via text mining. IEEE Trans. Softw. Eng. **40**(10), 993–1006 (2014)
14. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Mach. Learn. **37**(3), 297–336 (1999)
15. Schumacher, M., Haul, C., Hurler, M., Buchmann, A.: Data mining in vulnerability databases. Comput. Sci., 12–24 (2000)
16. Text Fixer: Common English Words List (2017). Accessed 11 Mar 2017
17. Wijayasekara, D., Manic, M., McQueen, M.: Vulnerability identification and classification via text mining bug databases. In: IECON 2014–40th Annual Conference of the IEEE Industrial Electronics Society, pp. 3612–3618. IEEE (2014)
18. Zhang, S., Ou, X., Caragea, D.: Predicting cyber risks through national vulnerability database. Inf. Secur. J. Glob. Perspect. **24**(4–6), 194–206 (2015)