Proceedings of 13th IOE Graduate Conference
Peer Reviewed
ISSN: 2350-8914 (Online), 2350-8906 (Print)
Year: 2023 Month: April Volume: 13

Performance Enhancement of Support Vector Machine Using Autoencoder For Network Intrusion Detection System

Raushan Kumar Pandit [a], Sharad Kumar Ghimire [b]

[a, b] *Department of Electronics & Computer Engineering, Pulchowk Campus, IOE, TU, Nepal*
[a] 076msdsa013.raushan@pcampus.edu.np, [b] skghimire@ioe.edu.np

Abstract
Network Intrusion Detection System (NIDS) is a promising area of research topic in the field of Cyber Security. Behavioral-based NIDS also called anomaly-based intrusion detection system provides a more effective solution to Network Security than conventional or signature-based intrusion detection systems such as Firewalls and Anti-viruses. NIDS can be designed by adopting any classifier algorithm which can analyze network traffic as normal or abnormal (attacks). But the performance of good NIDS depends heavily on the selection of the best classifier algorithm for a given dataset, improvement strategies used to increase the accuracy of the classifier, and decreasing the training & testing time of the algorithm. In this paper, we propose an unsupervised learning approach for feature learning and dimensionality reduction without loss of information and a multi-class support vector machine algorithm for classifying attacks. The purpose to apply unsupervised feature learning (UFL) is to reduce training and testing time considerably and it improves the prediction accuracy of Support Vector Machine (SVM). The proposed model has been built by using Sparse Autoencoder and SVM Classifier. Feature learning is done by the Sparse Autoencoder algorithm and it reconstructs a new feature representation with minimal loss. Once pretraining has been done the new feature is input into the SVM algorithm to enhance its capacity to identify intrusion and classification precision. The result shows the performance of SVM improved & accelerated by the use of the Sparse autoencoder.

Keywords
Support Vector Machine (SVM), Sparse-Autoencoder (SAE), CIC-IDS2017 Dataset, Kullback-Leibler (KL) Divergence, Hyperparameter Tunning, STL(Self Taught Learning), Unsupervised Feature Training (UFL)

1. Introduction

Anomaly-based NIDS is defined as a classifier model which can take network traffic data as input, analyze those data, and based on the hypothesis that abnormal traffic (attacks) pattern is different from the normal traffic pattern detects attacks from traffic data. The idea of the Intrusion detection system was first proposed by James P. Andersion in 1980. Since then, many matured model has been designed with machine learning algorithms such as J48, Random Forest, Nave Bayesian, NB-Tree, SVM, ensemble method, and deep learning algorithms such as Autoencoder, CNN, LSTM which are capable to satisfy the need of network security. However, these model still faces challenges in improving detection accuracy, reducing the time complexity of algorithms, and the algorithm's capability to detect new types of novel attacks. Also, during the last two decades network size and its nodes as well as data volume handled by network nodes increased significantly. Due to this also day by day vulnerability to network nodes and mutated forms of old attacks as well as new novel attacks increasing considerably.

In this paper, we have described our NIDS model which used Sparse Autoencoder based on STL (Self Taught Learning) framework for effective feature representation as well as dimensionality reduction, in combination with a shallow machine learning algorithm SVM. We have tried a genuine approach to address some serious challenges faced by NIDS such as Detection Accuracy, Class imbalance problems, and high training and testing time of Machine learning algorithms. We have compared the performance of SVM alone with SVM backed by the Sparse Autoencoder model with the help of the CIC-IDS2017 evaluation dataset. Models have been evaluated on performance metrics parameter like Accuracy, Precision, and Recall.

2. Related Works

Arnaud Rosay et al. explain the noises, and drawbacks of CIC-IDS2017 dataset, and suggest methods to clean and process that dataset so that this could be meaningful to feed into the proposed model to design NIDS [1]. The proposed algorithms such as RNN, CNN, and BLSTM described by Yin et al. and Isra Al-Turaiki et al. in [2, 3] respectively have given potential approaches to building NIDS with fantastic accuracy of the model with low false alarm rate than traditional algorithms like HNB(Hidden Naive Bayes), Decision Tree as described in paper [4] and [5] respectively.

Sumaiya Thaseen et al. demonstrate dimensionality reduction by Chi-Square feature extraction and using multi SVM delivered good detection accuracy of the model [6]. The model proposed in this paper struggles to detect U2R attacks on NSL-KDD dataset. For new attacks training of the model will be difficult. Imene Zenbout et al. and Kemker et al. describe the classification of cancer & hyperspectral images based on a Self-taught learning framework using Stacked Sparse & Convolutional autoencoder in [7] and [8] but taken into consideration to understand different criteria for use of autoencoders. Utilizing Deep Learning Techniques for Effective

Zero-Day Attack Detection by Hindy et al. in [9] might be one of the latest papers which evaluates their model performance on NSL-KDD as well as CIC-IDS2017 dataset. The approach applied in this paper is One-Class SVM and Deep Autoencoder. Did not talk about the training & testing time of the model.

This paper is based on paper [10] by Al-Qatf et al., which focuses on training and testing time as well as the accuracy of the Model. Also, it has implemented the proposed methodology based on the Self-taught learning (STL) framework. The dataset used in this NSL-KDD benchmark. The performance of the proposed model received high by tuning hyper-parameter terms like p, λ, β came in sparse autoencoder loss function & SVM kernel function. Also, the model was trained & tested with 10-cross validation with best values up to 1000 epochs. After studying the paper, still there is room to play with these tuning parameters, validation & epochs to find the best result with the CIC-IDS2017 benchmark dataset.

3. Methodology

3.1 Evaluation Dataset Description

The CIC-IDS2017 dataset is provided by the Canadian Institute of Cybersecurity publicly for researchers[11]. According to paper [11], the data was recorded by CICFlowMeter(formally ISCX FlowMeter) for 5 days, inside their local lab infrastructure is divided into two classes one contains 4 machines launching attacks toward the second class containing 10 victim machines.

The network traffic is also labeled into 15 classes. Among them, there are 14 types of attacks along with one Benign (Normal) traffic. There is a total of 2,830,743 instances with 78 Features.

In this research work, for a classification task support vector machine algorithm has been used. Even after employing Sparse Autoencoder (Deep Learning Approach) for unsupervised feature learning but still, at last, a support vector machine has been used for classification tasks. With the help of Numpy & Pandas library redundant records & constant values features have been removed. Also, Infinity values and NaN values have been taken care of critically.

3.1.1 Small Dataset (CIC-IDS2017)

Due to time complexity & resource limitations, SVM time computation increases with the increase in data size. For the whole dataset, it's almost impossible to train the model with efficient timing. This is the reason, a small dataset has been made from the original one. Small set data has been made by taking records randomly. There are no specific rules but out of a total of fifteen classes, only four classes ("Normal Traffic", "DosHulk", "Port Scan" and "DDoS") are reduced randomly because these classes have a count of instances more than one hundred twenty-five thousand at least. Remaining all instances from the original whole data set for the remaining eleven classes have been taken in the small data set because these are minority classes of attacks that are much crucial for our Model and strictly can't be ignored.

The small data set contains 354603 instances with 31 features after dropping highly correlated features (above 90 %) & Chi-square feature selection [9] & [6]. More details about the number of instances with types of attack are shown in Table 1. As explained above only four classes of traffic have been reduced randomly.

Table 1: Small Dataset Instance Count(CIC-IDS2017)

Traffic (Description)	CIC-IDS2017 (Instances Count)
Benign	100,000
DoS Hulk	80,000
PortScan	75,000
DDoS	60,000
DOS GoldenEye	10,293
FTP-Patator	7,935
SSH-Patator	5,897
DOS Slowloris	5,796
DoS Slowhttptest	5,499
Bot	1,956
Web Attack Brute Force	1,507
Web Attack XSS	652
Infiltration	36
Web Attack SQL Injection	21
Heartbleed	11
Total	354,603

3.1.2 SMOTE applied Small Dataset (CIC-IDS2017)

Imbalanced classification involves developing predictive models on classification datasets that have a severe class imbalance. Here, the condition arises to deal with a Severe Class imbalance problem. Because there are some attacks like "Heartbleed", "SQL Injection", "Infiltration", "Web attack Brute Force", and "Web attack XSS" which fall under a high minority class.

Oversampling minority classes is one strategy to solve the issue of class imbalance. The Synthetic Minority Oversampling Method, or SMOTE for short, is a particular approach for the minority class. By oversampling minority groups, SMOTE will aid in the creation of new synthetic data. If SMOTE applied for minority classes and an under-sample for the majority class, the model would perform better.

Actually, to some extent, hyper-parameter tunning has been done in SVM as well as in SAE-SVM Model to tackle the class imbalance problem to get better results even with the unbalanced dataset. But still, some attacks are undetected by models so SMOTE has been applied strategically in such a way only to those undetected classes. The dataset with instances counts achieved after applying SMOTE to a small data set (Table 1) is shown in Table 2.

3.2 Proposed Model

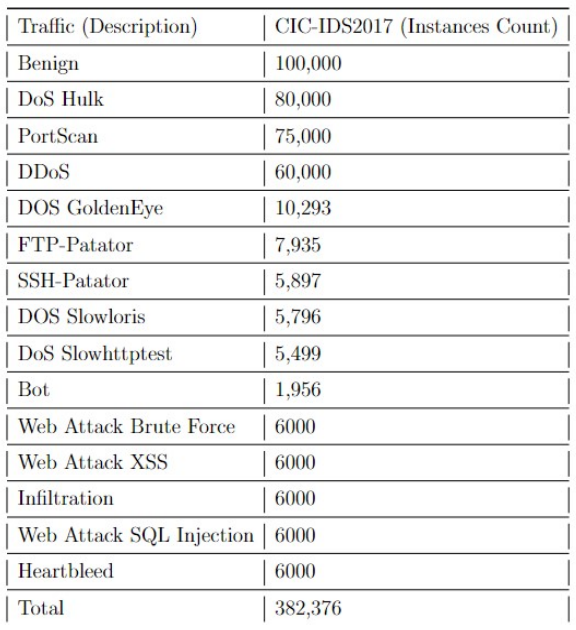
3.2.1 Problem Formulation & Overview

As seen in the figure 1, after feature selection the processed labeled data set is divided into training and test set. For the explanation purpose, suppose the training labeled dataset contains m records which are represented by (X_l^m, Y^m).

where, $X_l^m, Y^m = \{(x_l^1, y^1), (x_l^2, y^2), ..., (x_l^m, y^m)\}$,

input feature vector $X_l^i \varepsilon R^n$ (The subscript "l" indicates that it is

Table 2: Small Dataset Instance Count,after SMOTE applied(CIC-IDS2017)



Traffic (Description)	CIC-IDS2017 (Instances Count)
Benign	100,000
DoS Hulk	80,000
PortScan	75,000
DDoS	60,000
DOS GoldenEye	10,293
FTP-Patator	7,935
SSH-Patator	5,897
DOS Slowloris	5,796
DoS Slowhttptest	5,499
Bot	1,956
Web Attack Brute Force	6000
Web Attack XSS	6000
Infiltration	6000
Web Attack SQL Injection	6000
Heartbleed	6000
Total	382,376

labeled record) and $y^i \varepsilon$ {1,2,3,... C} are corresponding labels for multi-class classification (The "i" subscript denotes the range of the record i.e. 1 to m).

If we remove the corresponding label for input x_l^i from the training dataset, the remaining record will be called an unlabeled training set which is denoted by,

$X_u^i = (x_u^1, x_u^2, ..., x_u^m) \varepsilon R^n$

To get better representation & low dimensionality of input $x_l^1, x_l^2, ..., x_l^m$ εR^n, we can't initialize weighted parameter $w_i j^l$ (Subscripts & Superscript meaning will be given into Sparse Autoencoder section) equal to zero. Otherwise, all hidden nodes of the Sparse Autoencoder will compute exactly the same functions of the input. In order to serve symmetry breaking, we fed unlabeled training samples $x_u^1, x_u^2, ..., x_u^m$ εR^n to the Sparse Autoencoder algorithm (the first step in Figure 4). After learning parameters W & b1 (Bias parameter) by Sparse Autoencoder due to previously fed unlabeled sample, now we feed $x_l^1, x_l^2, ..., x_l^m$ (both training as well as the test set at the second step in Figure 3) to Sparse Autoencoder which tries to reconstruct and learn its output values $(\hat{x}_l^1, \hat{x}_l^2, ..., \hat{x}_l^m)$ to be equal to its input $(x_l^1, X_l^2, ..., x_l^m)$.

In this process from the hidden layer, we can get the low-dimensional representation that is $\{(a_l^1, y1), (a_l^2, y^2), ..., (a_l^m, y^m)\}$ where original input data is replaced with activation function a. Thus, we will have new training set of data (a_l^i, y^i). We train our SVM model with this low-dimensional data. Exactly like the training set, we follow the same process through Sparse Autoencoder for initial test data & we will get low-dimensional a_{test}. This a_{test} will be used to evaluate the SVM model.

3.2.2 Sparse AutoEncoder

An Autoencoder neural network is an unsupervised feature learning algorithm that applies Stochastic gradient descent

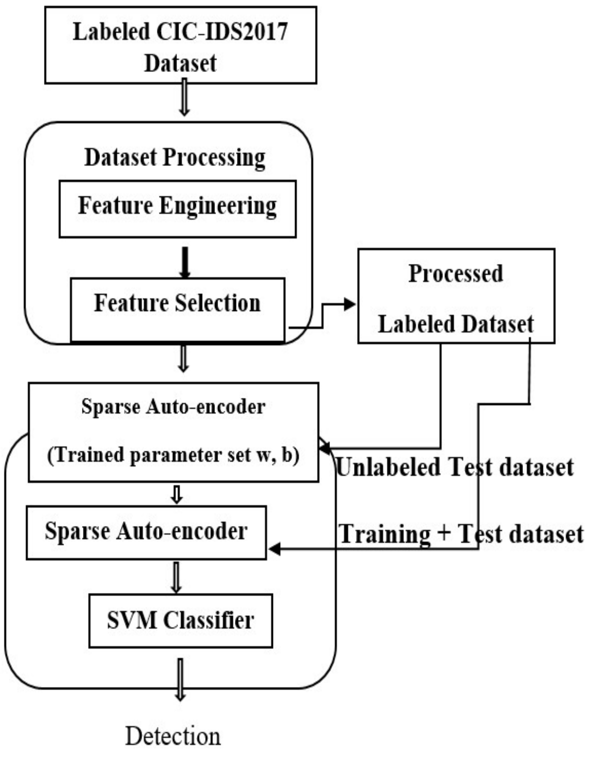


Figure 1: NIDS Model combing Sparse Autoencoder with SVM [10]

which is implemented by a backpropagation algorithm. An autoencoder tries to learn a function $Z_{w,b}(x) \approx \hat{x}$. That means, it trying to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x. The Sigmoid identity activation functions $\frac{1}{1+e^{-z}}$ for range [0, 1] has been used for SAE (Sparse Autoencoder). For explanation purposes, let's suppose there are

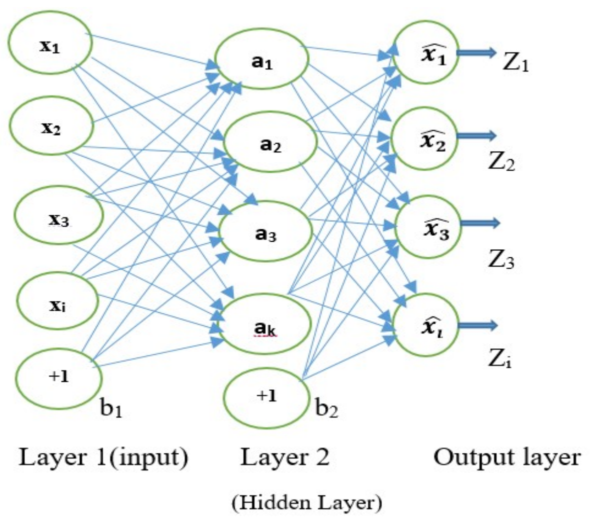


Figure 2: Sparse Autoencoder

an 'N' number of units in Layer 1 and the Output Layer of Autoencoder. Also, 'k' is the number of units in a hidden layer of Autoencoder. The weight & bias associated with input are expressed as

w_{ij}^l which is the Weight for connection between unit j in layer 'l' and the unit i in layer 'l+1',
b_i^l which is the bias for unit i in layer 'l+1' and

a_i^l which is activation for unit i in layer 'l'.

The generalized equation for the hidden layer (activation function) and output layer is written as

$$a = f(X_i^l) = g(wx + b_1) \tag{1}$$

$$Z = f(a_i^l) = g(Va + b_2) \tag{2}$$

As shown in the above figure, the equation for low-dimensional (compressed) output from the hidden layer could be written as
$a_1^2 = f(w_{11}^1 x_1 + w_{12}^1 x_2 + ... + w_{1j}^1 x_i + b_1^1)$
$a_2^2 = f(w_{21}^1 x_1 + w_{22}^1 x_2 + ... + w_{2j}^1 x_i + b_2^1)$
$a_k^2 = f(w_{i1}^1 x_1 + w_{i2}^1 x_2 + ... + w_{ik}^1 x_i + b_k^1)$
The weighted parameter (W, V), and bias parameter b in equation 1 and 2 updated like
$w_{ij}^l = w_{ij}^l - \alpha \frac{\partial J(w,b)}{\partial w_{ij^l}}$ and
$b_i^l = b_i^l - \alpha \frac{\partial J(W,b)}{\partial w_{ij}^l}$.

$$Where, J(w,b) = \frac{1}{2}\|Z_{w,b}(x_i) - \hat{x}_i\|^2 + \frac{\lambda}{2}\sum_l \sum_i \sum_j (w_{ij}^l)^2 \tag{3}$$

The equation 3 is the cost function of Autoencoder. In equation 3, the first term is squared-error which tries to minimize stochastic gradient descent and the second term is the regularization term, often referred to as a weight decay term. It tends to reduce the weight size and aids in preventing over-fitting. The weight decay parameter is λ controls the relative importance of the two terms. In Autoencoder, if hidden units are less than input or output units then input information is forced to compress at the hidden layer. But even when the number of hidden units is large or even equal to input units then still we can discover interesting structures, by imposing another constraint on the Network. If we impose a "Sparsity" constraint on the hidden units, then the Autoencoder will still discover interesting structures in the data, even if the number of hidden units is large. Let's think the neuron is active if its output value is close to 1 and is "inactive" if its output value is close to 0. We would like to constrain the neurons to be inactive most of the time. This can be done by adding the Sparsity Penalty term which is Kullback-Leibler divergence.

$$KL(p\|\hat{p}_j) = p \log \frac{p}{p_j} + (1-p)\log \frac{1-p}{1-p_j} \tag{4}$$

Where 'p' is the Sparsity constraint parameter whose value is 0 to 1 & β is the Penalty term which controls the weight of the sparsity penalty term. The penalty term equation 4 has zero value when $p = p_j$, which means activation must be near 0. And it increases monotonically as p_j diverges from p.

Now, the overall cost function for Sparse Autoencoder will be written as

$$J_s parse(w,b) = J(w,b) + \beta \sum_{j=1}^{k} KL(p\|p_j) \tag{5}$$

In equation 5, the terms λ, p, and β are considered for hyperparameter tunning to reduce the overall cost function of the sparse autoencoder. Ultimately these parameters help the algorithm to learn input features much more effectively.

3.2.3 SVM(Support Vector Machine)

Based on statistical learning theory (STL), the SVM is a supervised machine learning technique that creates a hyperplane to separate a class of positive examples from a class of negative instances using structural risk reduction criteria. It is one of the popular ML techniques for modeling classification as well as regression problems due to its good accuracy and performance. It splits data points with hyperplanes. The main aim to maximize the distance between support vectors separating all classes is necessary. The points that lie on the marginal plane line are called support vectors. The goal is to maximize margin by adjusting the cost function of SVM to get good performance of the model. It uses kernel functions such as Radial Basic Function (RBF), linear, Sigmoid, and polynomial. The RBF kernel has been used for the proposed model, which is also called the Gaussian kernel that tries to make the hyperplane in N-dimensional space that distinctly classifies the data points. Also, RBF comes with artificial parameters C & γ as shown in equation 6 & 3 so that parameter tuning can be done easily to select the best parameters which can give fine accuracy & prediction from the model. The equation for RBF Kernel can be defined as

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2) \tag{6}$$

Where $||x_i - x_j||^2$ is the squared Euclidean distance between two features and γ can be further defined by $\gamma = \frac{1}{2\sigma^2}$. The σ^2 is the variance associated with each attribute in the validation data set.

The objective function of SVM can be defined as

$$min||w||^2 + C\sum \xi \tag{7}$$

where "w" is the margin of hyperplanes, ξ is the error rate due to slack variables, and "C" is the Overfitting constant. If "C" is very large, the optimization algorithm reduces $||w||$ which leads to generalization loss. Also, if "C" is small, it gives a large training error. So it is crucial to find the optimal value of "C".

In a nutshell, by taking low-dimensional & the best features learned training dataset from Sparse Autoencoder by tuning these parameters of equation 5, feeding them into SVM where again tuning done to SVM parameters(RBF kernel) and getting enhanced prediction & accurate results from STL-framework based model (SAE-SVM). The "GridSearchCV" is an automatic model selection tool from the Sklearn Library of Python. It facilitates fitting SVM model at best γ & "C" values from the list of different γ & "C" values supplied artificially for RBF Kernel. Results and parameter tuning are described in detail in the Result Section.

3.2.4 Algorithm of Proposed Methodology

3.3 Experiment Environment

Table 3: Environment Specifications

SN	Google Colab Pro	Specifications
1	CPU	Xeon(R) Single Core
2	GPU	NVIDIA Tesla T4
3	GPU Storage	15 GB
4	CPU Storage	12.7 GB
5	DISK Storage	80 GB

Algorithm 1 Algorithm for SAE Training

Input: Step1-Unlabled Training Dataset, Step2- Training + Test dataset (all), Sparse autoencoder architecture, optimizer & parameters ('SGD', batch-size & no. of epochs), and regularization parameters)
Output: Trained Sparse Autoencoder & low dimensional datasets

```
1. Sparse Autoencoder ← build autoencoder (ANN_Architecture, Kullback-Leibler
   "p" value and regularizer parameters )
2. Optimizer ← SGD-minibatch (Stochastic Gradient Descent ) batch-size=256,
   epochs=500
   ← SGD-minibatch (Stochastic Gradient Descent ) batch-size=256, epochs=256
3. Step 1: Training with an unlabeled training data set to set weight & bias parameters
4. Step 2: Training with all data (Training + Test set)
5. return sparse autoencoder model & save the trained model
6. Generate low dimensional data $\{(a_l^1, y1), (a_l^2, y^2), ...., (a_l^m, y^m)\}$ (Sigmoid activation
   function applied to input data)
7. Save data for classification purposes by SVM
```

The specifications of the environment while doing experimentation have been tabulated in table 3. In order to implement code for Sparse Autoencoder we have used the TensorFlow Keras model. Also, the SciKit-learn library is taken into consideration while building the SVM model.

4. Result & Discussion

The performance of all the models (SVM and STL-SAE-SVM) has been depicted in this section. The comparison has been made with results for the given dataset & model. The performance metrics like Precision, Recall, Training time, and Testing time have been shown in the bar diagram or tabulated form to evaluate each model.

4.1 Performance of SVM Based NIDS

4.1.1 Precision & Recall Metrics for Small Dataset (Table 1)

The small dataset (Table 1) has been split into an 80:20 ratio. Eighty percent of instances were taken as training of the SVM model. The rest instances were for predicting the model.

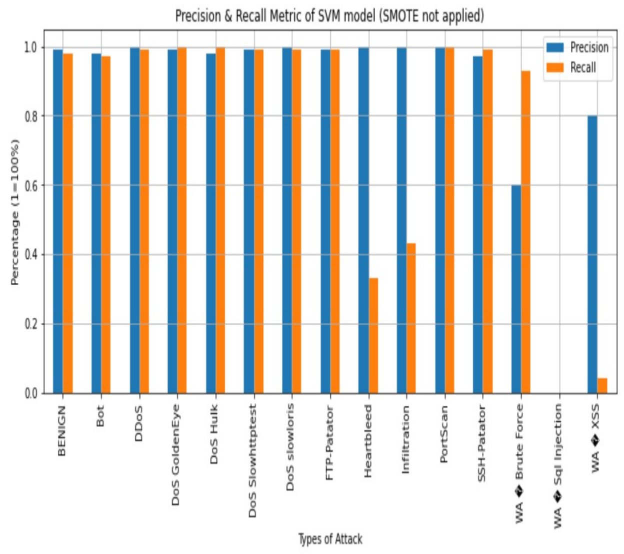


Figure 3: Precision & Recall at 'C'=100,'gamma'=1 for SVM model only

Table 4: Parameters, Train & Test Time for SVM

Parameters	Training Time	Testing Time
C=0.1,1,10,10^2,10^3 γ=1,0.1,10^{-2},10^{-3},10^{-4}	8 Hour, 33 minutes	6 minutes
Best ={'C'=100,γ=1 }		

After the training, testing, and prediction, the overall accuracy of the SVM model only is found 99%. Precision & Recall result for each class (attack) has been shown in Figure 3 in bar graph form at best parameters for RBF kernel. The time taken by the model and parameters used for the model is tabulated in Table 4.

For the evaluation of NIDS "Recall" would be the best parameter. The SVM model only is performing worst when it comes to detecting or classifying attacks like "Heartbleed", "Infiltration", "Web Attack SQL Injection", and, "Web Attack Cross-site scripting " because, in Figure 3, the recall values have been 32 %, 42 %, 0.02 %, and, 2 % respectively. It has been performing better for the rest of the attacks or classes.

4.1.2 Precision & Recall Metrics for SMOTE applied small dataset (Table 2)

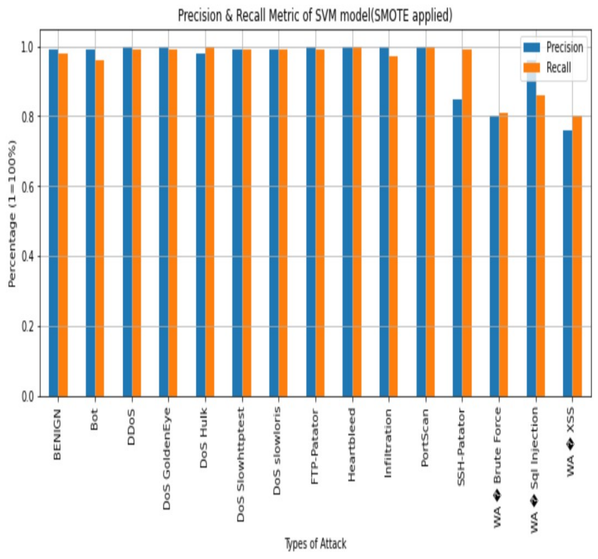


Figure 4: Precision & Recall at 'C'=100,'gamma'=1 for SVM model only

After the training, testing, and prediction, the overall accuracy of the SVM model only is found 98%.

In Bar Diagram (Figure 4), the recall values for "Heartbleed", "Infiltration", "Web Attack SQL Injection", and "Web Attack Cross-site scripting " have been found 100 %, 99 %, 83 %, and, 80 % respectively. These are improved due to applying SMOTE in comparison to Subsection 4.1.1. These are quite a descent values for extreme minority class of attacks. For the rest of the majority class of attack it has been above 97 %.

4.2 Perfomance of STL-SAE-SVM based NIDS

4.2.1 Precision & Recall Metrics for Small Dataset (Table 2)

The table 6 shows a list of parameters we have chosen for training unlabeled as well as labeled data with SAE.

Table 5: Parameters, Train & Test Time for SVM

Parameters	Training Time	Testing Time
C=0.1,1,10,10^2,10^3 γ=1,0.1,10^{-2},10^{-3},10^{-4}	8 Hour, 45 minutes	5 minutes
Best={'C'=100,γ=1 }		

Table 6: Parameters for Sparse Autoencoder

Parameters (λ,p, β)	Values	Best Values
'p'	0.01,0.02, 0.03,0.05	0.01
β	1,2,3,4	3
λ	$10^{-3}, 10^{-4}$, $10^{-5}, 10^{-6}$	0.000001

Table 7: Total Train & Test time of STL-SVM Model at best parameters for SAE and SVM

Best Values	TrainTime	TestTime	Epoch
λ = 0.000001, β=3, p=0.01, C=100, γ=1	29 min (X_u^i) 21min(X_l^i) 49min(SVM)	4 min	512 256 NA
Total	1 Hr 52 min	4 min	

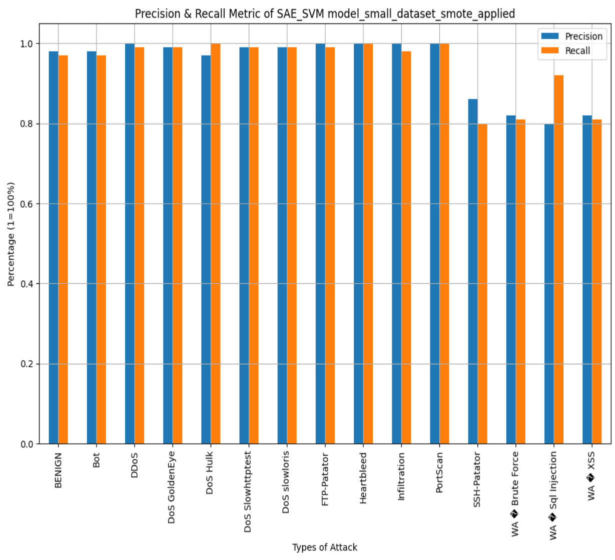


Figure 5: Precision & Recall of STL-SAE-SVM model at best parameters described in Table 7

We can train our STL-SVM model with more instances than SVM alone model but for comparison purposes, we have taken the same instances of a dataset (which was used for the SVM Model, Table 2). For the experimentation, as explained in Section 3.2.1 we have trained Sparse Autoencoder with unlabeled data first up to 512 epochs and with labeled data up to 256 epochs. Time consumed by SAE-SVM (Sparse autoencoder-Support Vector Machine) in training & testing as mentioned in table 7. If we compare SVM model only (Table 5) with the STL-based-SAE-SVM model (Table 7) then we can conclude applying the Self-taught learning-based (STL) unsupervised learning approach(Sparse autoencoder) to SVM model significantly reduced training time of model for NIDS.

Also, the overall accuracy of the STL-based-SAE-SVM model is 99%. So, in terms of Accuracy, it has improved a bit than SVM model only for SMOTE applied small dataset.

As in Figure 6, the training loss occurred due to the training dataset by sparse autoencoder being about to converge with the validation loss curve with a very minimal gap. Mini-batch SGD (Stochastic gradient descent) as an optimizer has been used for sparse autoencoder. There are 30 nodes in the input, hidden & output layers of the sparse autoencoder.

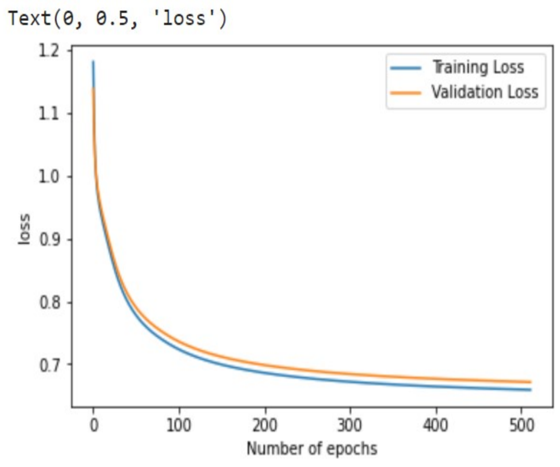


Figure 6: Training & Validation loss graph of SAE

5. Conclusion

For learning low-dimensional features from unprocessed data, single-layer sparse autoencoding is more effective. Automatically obtaining effective and suitable low-dimensional characteristics for the classification was made simple by SAE. Due to this SAE has drawn huge attention to its study in recent years. As seen in 4.2 section, the detection accuracy of SVM and training time of SVM enhanced due to the Feature learning approach backed by Sparse Autoencoder.

6. Future Enhancements

The proposed model shows decent results in terms of enhanced prediction accuracy as well as reduced training & testing time. Scikit-learn SVM is slow and it is unable to use a GPU accelerator. Thundersvm is an open-source library that leverages GPU and multi-core CPUs in applying SVM to solve problems in a much faster way with high efficiency. Thundersvm could enhance the efficiency of the model to some extent. Also, NIDS could be built using an Autoencoder alone on the basis of reconstruction loss comparison.

References

[1] Arnaud Rosay, Eloïse Cheval, Florent Carlier, and Pascal Leroux. Network intrusion detection: A comprehensive analysis of cic-ids2017. 2022.

[2] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5:21954–21961, 2017.

[3] Isra Al-Turaiki, Najwa Altwaijry, Abeer Agil, Haya Aljodhi, Sara Alharbi, and Lina Alqassem. Anomaly-based network intrusion detection using bidirectional long short-term memory and convolutional neural network. *The ISC International Journal of Information Security*, 2020.

[4] Levent Koc, Thomas A. Mazzuchi, and Shahram Sarkani. A network intrusion detection system based on a hidden naïve bayes multiclass classifier. *ELSEVIER*, 2021.

[5] Shih-Wei Lin, Kuo-Ching Ying, Chou-Yuan Lee, and Zne-Jung Lee. An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection. *Applied Soft Computing*, 12(10):3285–3290, 2012.

[6] Ikram Sumaiya Thaseen and Cherukuri Aswani Kumar. Intrusion detection model using fusion of chi-square selection and multi-class svm. *Journal of King Saud University-Computer and Infromation Sciences*, 2017.

[7] Imene Zenbout, Abdelkrim Bouramoul, and Souham Meshoul. Stacked sparse autoencoder for unsupervised features learning in pancancer mirna cancer classification. 2020.

[8] Ronald Kemker and Christopher Kanan. Self-taught feature learning for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(5):2693–2705, 2017.

[9] Hanan Hindy, Robert Atkinson, Christos Tachtatzis, Jean-Noël Colin, Ethan Bayne, and Xavier Bellekens. Utilising deep learning techniques for effective zero-day attack detection. *Electronics*, 9(10):1684, Oct 2020.

[10] Majjed Al-Qatf, Yu Lasheng, Mohammed Al-Habib, and Kamal Al-Sabahi. Deep learning approach combining sparse autoencoder with svm for network intrusion detection. *IEEE Access*, 6:52843–52856, 2018.

[11] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*, 2018.