Network Intrusion Detection System using an Improvised Convolution Neural Network : Comprehensive analysis and review

Abstract:

There has been a rapid growth in recent years in the number of devices which are interconnected to each other to share information, resources, etc. The more it has become essential to be a part of a network, more security issues are being introduced in this chain. Intrusion detection systems(IDSs) are a modern approach to deal with the security issues that any device or even data which is a part of any network system is vulnerable to. Network intrusion detection operates in such a way that the device can function as usual being in an "open" network while keeping a check on unauthorized access, misuse, and abuse of the computer system(or any device which is in the network) by both system insiders and external penetrators. In order to detect intrusions, intrusion detection systems often use statistical fact and rule-based exploration models based on different approaches. Several host-based and network-based IDSs are investigated in this article, and the features of the network topologies are discovered. A comparison of various IDS approaches with their main properties are provided in this paper to highlight the techniques applied in them as well as the area less focused in the literature surveys as we have identified. A detailed explanation of machine learning and deep learning IDS approaches are discussed here. The metrics, simulations, and environment have been analyzed and contrasted for machine learning and deep learning algorithms and a new approach for the convolution neural network for IDS has also been proposed as a token of contribution in the existing CNN models. Finally, future directions for NIDSs have been discussed for its subsequent advancement.

Keywords: Intrusion detection, Machine Learning, Accuracy, Deep learning, Regression, Classification, Principal component analysis, key-feature representation, CNN

Table 1: List of abbreviations used in the manuscripts along with their full form.

Acronym	Definition
AI	Artificial Intelligence
ML	Machine Learning
CNN	Convolution Neural Network
DL	Deep Learning
NIDS	Network Intrusion Detection System

1. Introduction:

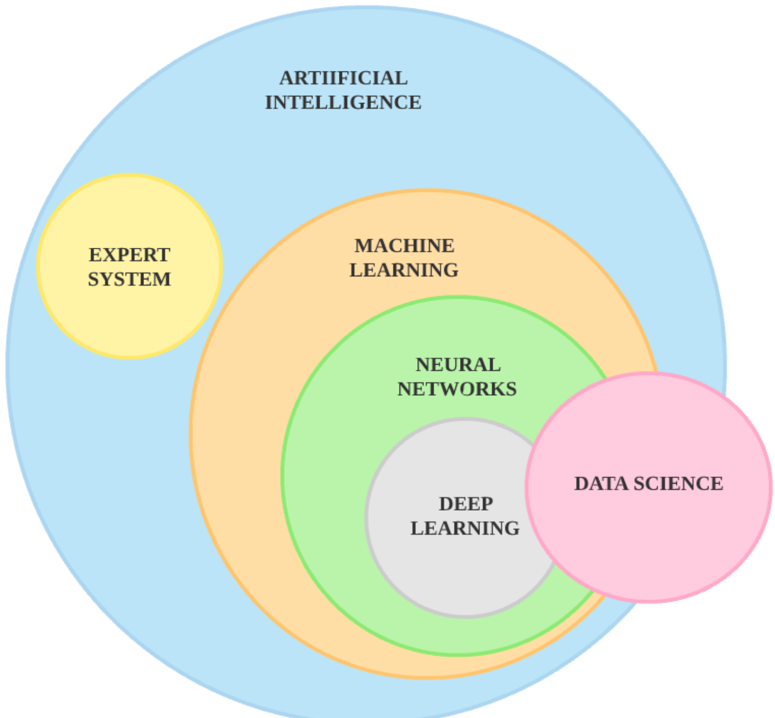


Figure 1. General Taxonomy – Artificial Intelligence Techniques

Cybersecurity, or information technology security (IT security) is the safety of PCs and networks from data disclosure, harm or theft to hardware or software. It is basically a utility of technologies, techniques and controls to shield systems, networks, programs, gadgets and information from cyber assaults. It ambitions to lessen the chance of vulnerabilities and guard against the unauthorized exploitation of system,resources,networks and data. Hence, Cyber security protects an individual or an organization against a loss which he or she can face while working on any system or digital device.

With the current development and boom withinside the improvement of internet and communication technologies over previous couple of years, network safety has emerged as a critical studies domain. Emerging novel attacks pose a massive challenge for community safety to accurately detect all kinds of intrusions. Computer networks are extensively utilized by diverse industries, enterprise and diverse fields that are present today. Therefore, constructing dependable networks is a completely crucial undertaking for an IT administrator. There are many forms of attacks threatening the availability, integrity and confidentiality of system networks which additionally impacts the corporation that trusts the network.
The Denial Of Service attack (DOS) is one of the most common dangerous attacks however there are numerous others like this which impacts the ethics of a networking environment e.g. SPY and PHF are some examples.
After the digital revolution, large quantities of data have been generated with time through various networks but this data should be accessible to only those who want it. Growing computer networks have made the process of data analysis very difficult and this makes the data or information more and more vulnerable to any sort of misuse. Network intrusion detection system provides a proper mechanism that is needed to prevent unauthorized access to data and data modification. Network Intrusion detection system

(NIDS) is one of the kind solutions against any sort of harmful attack and can be used to identify vulnerabilities. Effectiveness of NIDS is measured by the number of detected attacks and less false alarms. Out of all the emerging technologies, AI and ML are broadly utilized for building intrusion detection models, which adjust with the consistent changes in the organization network attack.

We have tried to present a comparative study between some of the best existing ML and DL models for intrusion detection and with the help of deep understanding of the algorithms, we've presented an improvised CNN model which we can use to predict whether there is any intrusion in the system or not. This can be done by using the data extracted from the request being sent which will be passed through the trained model after some preprocessing to make it fit for prediction. The Model will be able to predict whether the incoming request is a normal request or not.

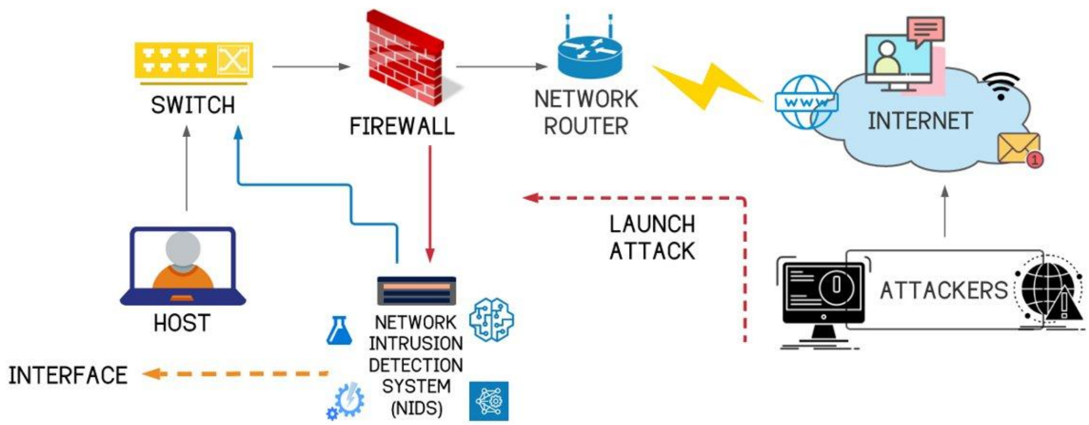


Figure 2. NIDS Representation

2. Literature Survey:

Table 2. Comparison with Previous surveys/ Comparison with other similar review articles

AUTHOR & YEAR	METHODOLOGY/ TECHNIQUES USED	ADVANTAGES	ISSUES	METRIC USED
Shone, N. Ngoc, T.N. Phai, V.D. Shi, Q. (2018)	Deep Belief Networks (DBNs), encoder-decoder paradigm, RNN model Non-Symmetric Deep Auto-Encoder, Stacked Non-Symmetric Deep Auto-Encoders	low levels of bias, robustness to outliers and overfitting correction	To assess and extend the capability of our model to handle zero-day attacks	5% improvement in accuracy and training time reduction of up to 98.81% compared to existing models.
Xu,Xin, Stanley Rocha, Álvaro (2018)	Genetic attribute reduction algorithm model based on rough set theory, Genetic attribute reduction algorithm and neural network combination	Great in dealing with redundant information	construction of the framework of intrusion detection system	calculation error of the system is reduced to 0.0001
Baig, Mirza M. Awais, Mian M. ,El-Alfy, El-Sayed M.(2017)	artificial neural network, cascading classifiers, ensemble learning, SVMs, ANNs, decision trees	multi-class intrusion detection (CANID) in computer network traffic, can efficiently detect various types of cyber attacks in computer networks	can be tested for more classification tasks involving a large number of classes, can be further refined using a filtering and example weighting strategy that favors the spare classes a little more than the remaining classes	accuracy : 99.36% Precision & recall: above 0.97 F1- score: > 0.96

Zha, Y. Li, J. (2018)	reconfigurable complex matching accelerator (CMA) enabled by the emerging nonvolatile memory technology (resistive random access memory)	demonstrates better performance and energy efficiency and the emerging RRAM-TCAM coprocessor, resolve the storage shortage by providing a high-density storage infrastructure	delay of the match line when performing search, ClassBench is designed for packet classification, which is the primary bottleneck in high-performance routers.	achieves 84.9% area reduction

Zhang, C. Ni, M. Yin, H. , et al. (2018)	support vector data description (SVDD)	overcome the unfitness of traditional SIL due to multi-noise samples and clustering instability	processing high-dimensional data with non-uniform density	Accuracy and f-score higher than other traditional/conventional existing methods.
Wang, Cheng-Ru Xu, Rong-Fang Lee, Shie-Jue, Lee, Chie-Hong (2017)	Supervised learning Extreme learning machine Adaptively incremental learning Support vector machine	effective in building models with good attack detection rates and fast learning speed	reduce the training time, or distributed computing algorithms can be applied to speed up the modeling process	Accuracy: CAI: 82.74% SVM: 85.87% MLP: 81.61% Recall: CAI: 98.41% SVM: 95.19% MLP: 99.48%
Santoro, Diego , Escudero-Andreu, Ginés Kyriakopoulos, Konstantinos G. Et al.	Hybrid-NIDS (H-NIDS) based on Dempster-Shafer (DS) Theory of Evidence	ability to detect novel attacks of anomaly-based NIDSs	development of a real-time hybrid NIDS able to detect a wider range of threats and cyber-attacks against wireless networks	high accuracy and precision rates.

(2016)				
Molina-Coronado, B. Mori, U. Mendiburu, A. Miguel-Alonso, J. (2020)	data mining and KDD	reduction in the number of tasks (focusing mostly on the data mining part), and the availability of a common evaluation framework for the comparison of NIDS proposals	lack of realism in the scenario, as both the normal and the malicious traffic is artificially generated	low rates of false alarms (false positives)
Ravi, N. Shalinie, S.M. (2020)	SDRK machine learning (ML) algorithm, supervised deep neural networks (DNNs) and unsupervised clustering techniques	ease of reconfiguration, flexible addition of features, reduced latency, improved service efficiency	when the controller fails, the IoT network also fails, to optimize the retraining time to fine-tune the performance of the model for the real-time network	Accuracy: 99.78%.
Xu, C. Shen, J. Du, X. Zhang, F (2018)	deep learning, neural network model	It showed that the GRU is more suitable as a memory unit for IDS than LSTM, and proved that it is an effective simplification and improvement of LSTM.	optimize the system so that it can be applied to real network environments and be implemented more efficiently.	The overall detection rate was 99.42% using KDD 99 and 99.31% using NSL-KDD with false positive rates as low as 0.05% and 0.84%, respectively.
Chapaneri, R., & Shah, S. (2019).	SVM, decision trees (DT), Naïve Bayes (NB), artificial neural networks (ANN); extreme learning machines (ELMs) [96% accuracy on NSL KDD with a significant reduction in testing time (2.6 s)- UNSW-NB15 dataset]	KSVCR is used for multi-class classification Ramp loss function implemented; Fast and can process network packets one or in a chunk.	Huge volume of data, High cost of false alarms in most IDSs, anomalous class distribution	presence of outlier samples in dataset and shows improved performance on both NSL KDD (98.6% accuracy) and UNSW NB15 (93.5% accuracy) datasets.

Mishra, P., Varadharajan, V., Tupakula, U., & Pilli, E. S. (2018).	Decision Tree Artificial Neural Network Naive Bayes Classifier Support Vector Machine Fuzzy Association rules	Feature Extraction method: SVM and Clustering or by using statistical methods such as Particle Swarm Optimization, Principal Component Analysis, Gradual Feature Removal, and Mutual Information based Feature Selection	deep algorithms generation - lot of data for training & classification algorithm, challenging to adopt for real-time classification because of the level of complexity involved in training huge amount of data, requirement of high-performance hardware to process the huge training data	User to Root Attacks: FNT, PSO & GA, 12 DARPA 98 (99.7%); Multi NN, KDD'99 (99.7%) Remote to Local Attacks: Ensemble of ANN, SVM, MARS, DARPA 98(100%); FNT, PSO & GA, 12 DARPA 98(99.09%)
Khan, F. A., & Gumaei, A. (2019, July).	adopted classifiers here were KNN, NB, NB-KE, SVM-POLY, SVM-RBF, SMO, DT, DS, HT, and RF. Classifier model: KNN Accuracy (%): 99.8393 Time taken to build a classifier model (in seconds): 0.06	good accuracy for detecting the denial of service (DoS) attack; accuracy of support vector machine (SVM) and decision tree (C4.5) methods; the accuracy of C4.5 was better than that of SVM.	loss of realism withinside the scenario, as each the everyday and the malicious visitors is artificially generated	DT, RF, HT and KNN classifiers are the highest with reasonable time cost of building.
Hu, Q., Yu, S. Y., & Asghar, M. R. (2020)	Packet capturing mechanisms, Packet detection mechanisms, Regular expression signatures	Improved memory and resource consumption. Multithreaded architecture improve IDS performance, reduce the packet drop rate	Large volume of multiple small flows can impact the CPU and memory usage as well as lead to the higher packet drop rate	Performance checking, Accuracy Checking, Checking both performance and accuracy

Gurung, S., Ghose, M. K., & Subedi, A. (2019).	Deep network to train classify network traffic between normal connections, intrusions. Reducing false alarm rate. Deep network system (sparse auto-encoder with logistic regression) NSL-KDD dataset. Logistic on NSL-KDD dataset NIDS, deep learning, Sparse auto-encoder, logistic classifier, NSL-KDD	Higher accuracy rate - Signature-Based Intrusion Detection approaches and also reduces False Positives and Negatives. Network learns & adjusts itself. Deep-net identify intrusion and adjust with the newer data to classify an intruder	Solid NIDS - inaccessibility of the real-time network data model, consisting of both intrusion and normal use, on a continuous basis evolving and changing attack patterns, long learning curve and insufficient knowledge in data sets.	The precision score obtained was 84.6% and recall score was 92.8% whereas specificity and negative predictive values were 80.7% and 90.7% respectively. The overall accuracy of the model was 87.2%.
Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019).	KDDCup 99 dataset; The DNN model which performed well on KDDCup 99 is applied on other datasets, such as NSL-KDD, UNSW-NB15, Kyoto, WSN-DS, and CICIDS 2017. DNNs performed well in comparison. Scale-hybrid-IDS-Alert Net used – realtime.	Performance of DNN is superior. With DNN network topologies, performance in terms of accuracy is closer to each other’s. Multi-class classification, accuracy, true positive rate (TPR), false positive rate (FPR). Repeated multi-level learning	Remote client address, TTL, TCP options and TCP are small and limited in KDDCup 99 data sets but actually exhibit to be of large.	KDDCup 99 and NSL KDD datasets accuracy range 95% to 99%. UNSW-NB15 and WSN-DStrain accuracy in range 65% to 75%
Jiang, K., Wang, W., Wang, A., & Wu, H. (2020).	OSS and SMOTE (dataset) for model training. Reduce training time. Two intrusion datasets (NSL-KDD and UNSW-NB15) .Random Forest, LeNet, AlexNet, CNN and BiLSTM.	Detection accuracy on Dos - 96.21%, detection accuracy on R2L - 61.32%..	Poor classification results, high false miss rate, high false detection rate, high false alarm rate and imbalanced data	NSL-KDD and UNSW-NB15 dataset, and classification accuracy achieved 83.58% and 77.16%.
Su, T., Sun, H., Zhu, J., Wang, S., & Li, Y. (2020).	Novel model BAT-MC and real NSL-KDD dataset. BLSTM layer connects forward LSTM and backward LSTM to extract features on traffic bytes. BAT-MC model achieves pretty high accuracy.	BAT-MC model - less. The RNN model improves faster and also has lower accuracy than BAT and BAT-MC model.	Original data too large, resulting in problems, data processing overflows, and inconsistent weights.	84.25% accuracy - BAT-MC network

Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020).	Deep discriminative models (Deep neural networks (DNNs), Recurrent neural networks (RNNs), Convolutional neural networks (CNNs)), Generative/unsupervised models (Restricted Boltzmann machine (RBMs), Deep belief networks (DBNs), Deep Boltzmann machines (DBMs))	Recurrent neural networks, deep neural networks, restricted Boltzmann machine, deep belief networks, convolutional neural networks, deep Boltzmann machines, and deep autoencoders	Deep discriminative models, mean false alarm rate of the deep autoencoders is better than three techniques, including, restricted Boltzmann machine, deep belief network, and deep Boltzmann machine	Convolution neural network gets a higher accuracy 97.376%, The deep autoencoders gets a higher accuracy 97.372%,

She, C., Ma, Y., Jia, L., Fei, L., & Kou, B. (2016).	space information network, intrusion-detection model, anomaly and misuse detection.	Has great properties; combines 2 technologies - ISA-IDS and MAIDS. Provides Misuse warning	New malicious code cannot be found. consumes numerous assets, few times discard illegal data which should be stored.	distributed model is designed for long distance conveyed design of a space data network.
Yin, C., Zhu, Y., Fei, J., & He, X. (2017).	LSTM, Bidirectional RNNs algorithm, NSL-KDD dataset. Naive Bayesian, Random Forest, Multi-layer Perceptron, Support Vector Machine.	GPU acceleration reduces training time; RNN-IDS model have strong modeling ability.higher accuracy rate, detection rate, low false positive rate	partially connected nodes of layers; decreased RNNs no longer display the capacity of deep learning knowledge to version high-dimensional features	RNN-IDS model 97.09% accuracy with NSL-KDD dataset.
Huang, K., Zhang, Q., Zhou, C., Xiong, N., & Qin, Y. (2017).	Active learning, visual sensor networks (VSNs), big data, self-organizing map (SOM),	An efficient intrusion detection approach for VSNs is proposed, traffic pattern learning. Hierarchical self-organizing map (HSOM),active learning strategy - accelerate the training process.	Large and dynamic video data is fabricated by visual sensors. Hence it becomes a tedious task to detect attacks quickly.	high detection accuracy and good real-time performance.AS S, TA, RUS, and ROS.95.3% accuracy.

Yang, H., & Wang, F. (2019)	Improved convolutional neural network (ICNN); KDDTest + dataset; advanced features by CNN	Recall rate is 4.24% and 1.16% higher than LeNet-5 and RNN; detection accuracy is 8.82% and 0.51% higher than LeNet-5 and DBN	The experiment parameters have the issue of large calculation tasks and low extraction efficiency. Model training process has poor generalization ability and low convergence speed.	KDDTest + dataset AUC values of KDDTest+ and KDDTest−21 : 0.9392 and 0.9020 [ROC CURVE]
Yang, H., Qin, G., & Ye, L. (2019).	DBN-SVM detection method; support vector machine (SVM); multi-restricted Boltzmann machine (RBM); Rbf kernel SVM; focus is to improve detection accuracy.	Effectively detects intrusion behavior. Deep learning model is proposed. Proposed method has higher accuracy,recall, precision rate when compared with SVM, DBN and PCA-SVM.	Sample size is small(wireless network intrusion type); accuracy rate is lower when compared with 5-layer DBN,	Accuracy rate: 97.45% Recall rate: 97.48% Precision rate: 97.78% with NSL-KDD dataset
Hindy, H., Brosset, D., Bayne, E., Seeam, A. K., Tachtatzis, C,et al. (2020).	ML/Deep Learning (DL) algorithms based Artificial Neural Networks (ANN), clustering, Genetic Algorithms (GA).	ML is used by 97.25% of the examined IDS - ANN, k-means and SVM gives better result.	limited number of available datasets	IDS – ANN 97.25%

Viegas, E., Santin, A. O., & Abreu Jr, V. (2020).	long-lasting intrusion detection architecture - building reliable ML model; Big Data, High-Speed Networks. Coupling batch techniques, hybrid detection architecture and stream learning techniques	Five years network traffic data is analyzed comprising 20 TB of data, daily updated model reaches higher accuracy.	modifications in network traffic behavior yields low accuracy rates very fastly.	MAWIFlow - dataset, 90% accuracy

Zhong, W., Yu, N., & Ai, C. (2020).	Using convolution neural networks a fully connected feedforward neural network is formed with a multi-level clustering algorithm.Big Data based Hierarchical Deep Learning System (BDHDLS) is proposed. BDHS understands both information stored in the payload and network traffic characteristics.	Understands unique data in a single cluster and time is reduced. Boosts the performance of IDS. Efficient technique generates multi-level cluster tree.	In the 1/3 phase, the hierarchical clustering procedure is finished for every one stage cluster with low best in parallel to provide the cluster subtree.	NSL-KDD, Kyoto2009, DARPA1998, ISCX2012 and CICIDS2017 dataset are studied along with the model.
Wu, K., Chen, Z., & Li, W. (2018).	Novel model utilizing convolutional neural networks (CNNs) is proposed. Imbalanced data set problem is solved.	Improves accuracy of class and reduced false alarm rate (FAR).Raw traffic vector format is converted into image format. Efficient and trustable solution.	Models sometimes has lower precision rate and has problems in data format.	NSL-KDD dataset 81% more efficient
Hu, Z., Wang, L., Qi, L., Li, Y., & Yang, W. (2020).	Novel ID model - adaptive synthetic sampling (ADASYN) algorithm and improved convolutional neural network (CNN).	The ADASYN method balances sample distribution. On model training, Split convolution module (SPCCNN) - increase diversity, eliminate interchannel information redundancy.	Low recognition accuracy (ACC) and high false alarm rate (FAR) is sometimes being unavoidable.	To test AS-CNN,NSL-KDD dataset is used.4.60% and 2.79% higher accuracy than CNN and RNN models
Wei, P., Li, Y., Zhang, Z., Hu, T., Li, Z., & Liu, D. (2019)	particle swarm optimization, deep belief network, genetic algorithm, artificial fish swarm algorithm	Average detection time is reduced by 24.69%. Common class accuracy is improved by 14.80%.	Fitness of the model is lowered to improve optimized network structure.	AFSA-GA-PSO applied - DBN-IDS and accuracy 82.36% obtained for KDDTest+ and 66.25% obtained for KDDTest-21.

Xiao, Y., Xing, C., Zhang, T., & Zhao, Z. (2019).	Data dimensionality reduction, Naive Bayes, Random Forest, Logistic Regression, SVM, Decision Tree - ML algorithms are used to implement network intrusion detection.	Has faster training speed (CNN–IDS), in spite of having massive data environment features can be easily selected.	Still not able to address low detection rate problem.	KDD-CUP99 dataset; Accuracy of model : 94.0%
Zeng, Y., Gu, H., Wei, W., & Guo, Y. (2019).	Encrypted traffic classification,Normal, Brute Force SSH, DDoS, HttpDoS, and Infiltrating are five set of dataset classes	Novel DL-Based framework provides robust and accurate results with less storage resource requirement.	Flow volume or flow duration - not acquired.	99.87% accuracy
Anthi, E., Williams, L., Słowińska, M., Theodorakopoulos, G., & Burnap, P. (2019).	Signature/Event/Rule Based IDSs Machine Learning IDSs Attack Type Classification IoT Smart Home Testbed	IoT devices are successfully distinguished in the network.higher values correspond to better classification performance.	The system is tested using four attacks and has a very low false positive rate.	F-measure performance: 1) 96.2%; 2) 90.0%; and 3) 98.0%.

3. Background:

We have used Google Collab's Free Tier plan to conduct all of our experiments. We use two experiments to study the performance of the IDS mode.

3.1 Machine learning approaches-based network intrusion detection systems

Any website visited by different people creates massive traffic, which thus can be utilized to contemplate and characterize ordinary, abnormal or malicious traffic. For this reason, research has been directed to discover optimized machine learning methods to build Network Intrusion Detection tools.

Here we have aimed to aid the research in this field. Majorly used data set for building this model is the NSL-KDD dataset. On the dataset various algorithms have been trained and tested. Considering the factors influencing effectiveness it has been observed that the dataset is tremendously skewed and not preprocessed properly. Thus, feature selection and handling of unknown network intrusions are major problem.

After the digital revolution, large quantities of data have been generated with time through various networks, but this data should be accessible to only those who want it. Growing computer networks have made the process of data analysis very difficult and this makes the data or information more and more vulnerable to any sort of misuse. A proper mechanism is needed to prevent unauthorized access to data and data modification. Thus, we have come up with a framework for network intrusion detection system which can help detect any harmful attack to the network. Models must be build distinguishing any change in network, in this manner alarming the network. This characteristic makes the intrusion prevention systems an enhance and upgrade to intrusion detection systems.

Initially, the dataset is pre-processed and classified whether an attack happens or not and then data is encoded and normalized. After data is pre-processed machine learning algorithms are performed and at each step flow of the program is revised. The algorithms used are TSNE of binary classification dataset, LGBM, SGB, Random Forest Naive Byes, Logistic Regression, KNN and Multiple ROC in Single Plot Tells the efficiency and accuracy. In the process flow NSL-KDD dataset is taken, and preprocessing is done. Then the dataset is encoded and feature reduction on the dataset is performed. Later the dataset is normalized, and machine learning algorithms have been performed. Finally testing algorithms on the test set and optimizing the network intrusion. Required libraries (python libraries or modules) are Pycaret, Shap, Numpy, Matplotlib, Sklearn, Pickle and tqdm. Once the model is built with a good accuracy, we can provide it with an interface where anyone can run the model in an abstract way and can use the system. It not only increases its real time application but also make it simpler and more feasible for use.

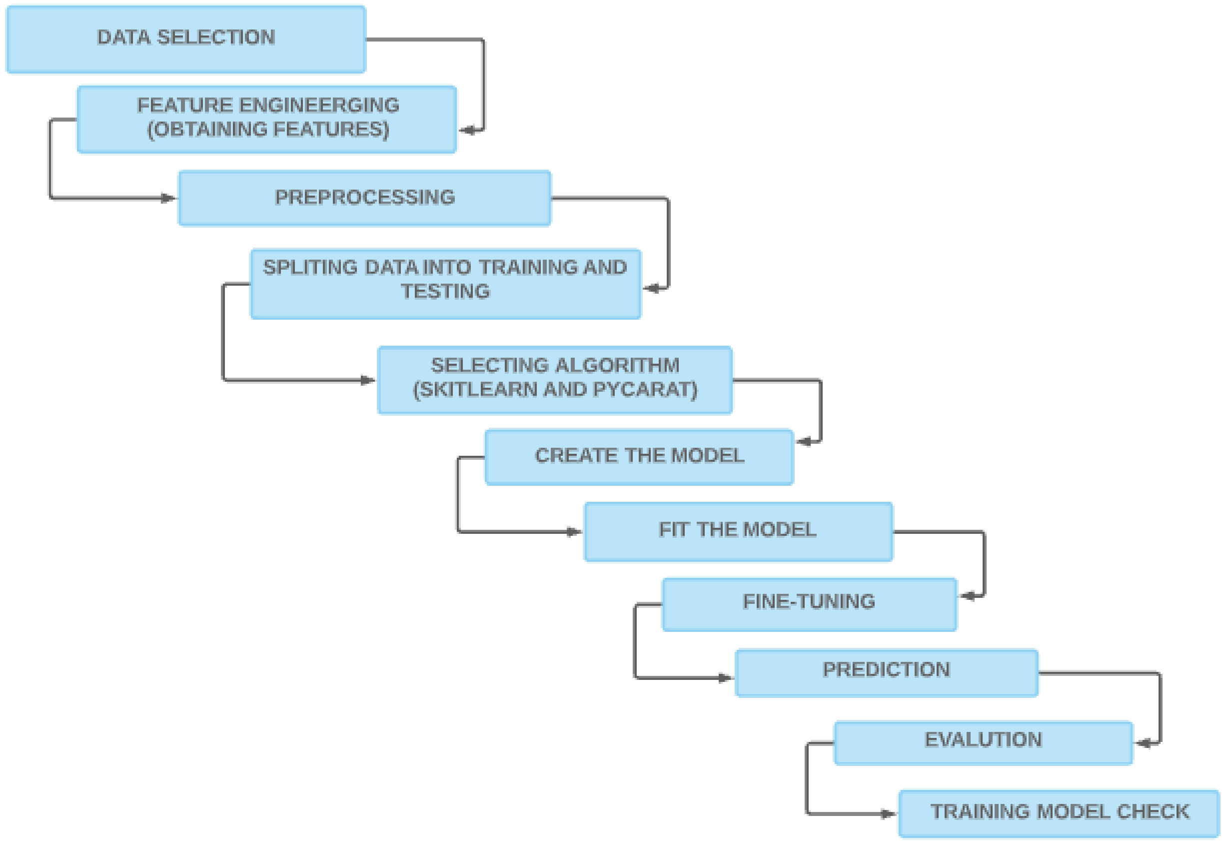


Figure 3. Process Flow of an algorithm

Environment Setup

```
[ ]  # import relevant modules
     %matplotlib inline
     import matplotlib
     import matplotlib.pyplot as plt
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import sklearn
     import imblearn
     import sys

     # Ignore warnings
     import warnings
     warnings.filterwarnings('ignore')

     # Settings
     pd.set_option('display.max_columns', None)
     np.set_printoptions(threshold=sys.maxsize)
     np.set_printoptions(threshold=np.inf, linewidth=np.nan)
     np.set_printoptions(precision=3)
     sns.set(style="darkgrid")
     plt.rcParams['axes.labelsize'] = 14
     plt.rcParams['xtick.labelsize'] = 12
     plt.rcParams['ytick.labelsize'] = 12

     print("pandas : {0}".format(pd.__version__))
     print("numpy : {0}".format(np.__version__))
     print("matplotlib : {0}".format(matplotlib.__version__))
     print("seaborn : {0}".format(sns.__version__))
     print("sklearn : {0}".format(sklearn.__version__))
     print("imblearn : {0}".format(imblearn.__version__))
```

```
pandas : 1.1.5
numpy : 1.19.5
matplotlib : 3.2.2
seaborn : 0.11.2
sklearn : 1.0.1
imblearn : 0.8.1
```

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Load Data

```
[ ]  # Dataset field names
     datacols = ["duration","protocol_type","service","flag","src_bytes",
         "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
         "logged_in","num_compromised","root_shell","su_attempted","num_root",
         "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
         "is_host_login","is_guest_login","count","srv_count","serror_rate",
         "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
         "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
         "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
         "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
         "dst_host_rerror_rate","dst_host_srv_rerror_rate","attack", "last_flag"]

     # Load NSL_KDD train dataset
     dfkdd_train = pd.read_table("/content/KDDTrain.txt", sep=",", names=datacols) # change path to where the dataset is located.
     dfkdd_train = dfkdd_train.iloc[:,:-1] # removes an unwanted extra field

     # Load NSL_KDD test dataset
     dfkdd_test = pd.read_table("/content/KDDTest.txt", sep=",", names=datacols)
     dfkdd_test = dfkdd_test.iloc[:,:-1]
```

Train dataset

```
[ ]  # View train data
     dfkdd_train.head(3)

     # train set dimension
     print('Train set dimension: {} rows, {} columns'.format(dfkdd_train.shape[0], dfkdd_train.shape[1]))
```

```
Train set dimension: 125973 rows, 42 columns
```

Test dataset

```
[ ]  # View test data
     dfkdd_test.head(3)

     # test set dimension
     print('Test set dimension: {} rows, {} columns'.format(dfkdd_test.shape[0], dfkdd_test.shape[1]))
```

```
Test set dimension: 22544 rows, 42 columns
```

Data Preprocessing

Map attack field to attack class

NSL-KDD dataset has 42 attributes for each connection record including class label containing attack types. The attack types are categorized into four attack classes as described by Mahbod Tavallaee et al. in [A Detailed analysis of the KDD CUP 99 Data Set] as:

1. **Denial of Service (DoS)**: is an attack in which an adversary directed a deluge of traffic requests to a system in order to make the computing or memory resource too busy or too full to handle legitimate requests and in the process, denies legitimate users access to a machine.
2. **Probing Attack (Probe)**: probing network of computers to gather information to be used to compromise its security controls.
3. **User to Root Attack (U2R)**: a class of exploit in which the adversary starts out with access to a normal user account on the system (gained either by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.
4. **Remote to Local Attack (R2L)**: occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine.

```
[ ]  mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint': 'Probe','mscan': 'Probe',
            'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf': 'DoS','mailbomb': 'DoS',
            'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
            'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow': 'U2R','xterm': 'U2R','ps': 'U2R',
            'sqlattack': 'U2R','httptunnel': 'U2R',
            'ftp_write': 'R2L','phf': 'R2L','guess_passwd': 'R2L','warezmaster': 'R2L','warezclient': 'R2L','imap': 'R2L',
            'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','worm': 'R2L','snmpgetattack': 'R2L',
            'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',
            'normal': 'Normal'
            }
```

```
[ ]  # Apply attack class mappings to the dataset
     dfkdd_train['attack_class'] = dfkdd_train['attack'].apply(lambda v: mapping[v])
     dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(lambda v: mapping[v])
```

```
[ ]  # Drop attack field from both train and test data
     dfkdd_train.drop(['attack'], axis=1, inplace=True)
     dfkdd_test.drop(['attack'], axis=1, inplace=True)
```

```
[ ]  # View top 3 train data
     dfkdd_train.head(3)
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root	num_file_creations	num_shells	num
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Exploratory Data Analysis

```
[ ]  # Descriptive statistics
     dfkdd_train.describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root	num_file_creations	num
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	12597
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111	0.204409	0.001222	0.395736	0.279250	0.001342	0.001103	0.302192	0.012669	
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366	2.149968	0.045239	0.489010	23.942042	0.036603	0.045154	24.399618	0.483935	
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	77.000000	5.000000	1.000000	7479.000000	1.000000	2.000000	7468.000000	43.000000	

```
[ ]  dfkdd_train['num_outbound_cmds'].value_counts()
     dfkdd_test['num_outbound_cmds'].value_counts()
```

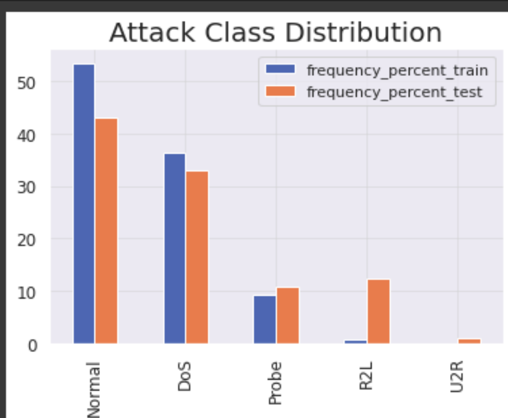
```
0    22544
Name: num_outbound_cmds, dtype: int64
```

```
[ ]  # 'num_outbound_cmds' field has all 0 values. Hence, it will be removed from both train and test dataset since it is a redundant field.
     dfkdd_train.drop(['num_outbound_cmds'], axis=1, inplace=True)
     dfkdd_test.drop(['num_outbound_cmds'], axis=1, inplace=True)
```

```
# Attack Class Distribution
attack_class_freq_train = dfkdd_train[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_test = dfkdd_test[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_train['frequency_percent_train'] = round((100 * attack_class_freq_train / attack_class_freq_train.sum()),2)
attack_class_freq_test['frequency_percent_test'] = round((100 * attack_class_freq_test / attack_class_freq_test.sum()),2)

attack_class_dist = pd.concat([attack_class_freq_train,attack_class_freq_test], axis=1)
attack_class_dist
```

	attack_class	frequency_percent_train	attack_class	frequency_percent_test
Normal	67343	53.46	9711	43.08
DoS	45927	36.46	7458	33.08
Probe	11656	9.25	2421	10.74
R2L	995	0.79	2754	12.22
U2R	52	0.04	200	0.89

```
# Attack class bar plot
plot = attack_class_dist[['frequency_percent_train', 'frequency_percent_test']].plot(kind="bar");
plot.set_title("Attack Class Distribution", fontsize=20);
plot.grid(color='lightgray', alpha=0.5);
```



Scaling Numerical Attributes

```
[ ]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()

     # extract numerical attributes and scale it to have zero mean and unit variance
     cols = dfkdd_train.select_dtypes(include=['float64','int64']).columns
     sc_train = scaler.fit_transform(dfkdd_train.select_dtypes(include=['float64','int64']))
     sc_test = scaler.fit_transform(dfkdd_test.select_dtypes(include=['float64','int64']))

     # turn the result back to a dataframe
     sc_traindf = pd.DataFrame(sc_train, columns = cols)
     sc_testdf = pd.DataFrame(sc_test, columns = cols)
```

Encoding of Categorical Attributes

```
[ ]  from sklearn.preprocessing import LabelEncoder
     encoder = LabelEncoder()

     # extract categorical attributes from both training and test sets
     cattrain = dfkdd_train.select_dtypes(include=['object']).copy()
     cattest = dfkdd_test.select_dtypes(include=['object']).copy()

     # encode the categorical attributes
     traincat = cattrain.apply(encoder.fit_transform)
     testcat = cattest.apply(encoder.fit_transform)

     # separate target column from encoded data
     enctrain = traincat.drop(['attack_class'], axis=1)
     enctest = testcat.drop(['attack_class'], axis=1)

     cat_Ytrain = traincat[['attack_class']].copy()
     cat_Ytest = testcat[['attack_class']].copy()
```

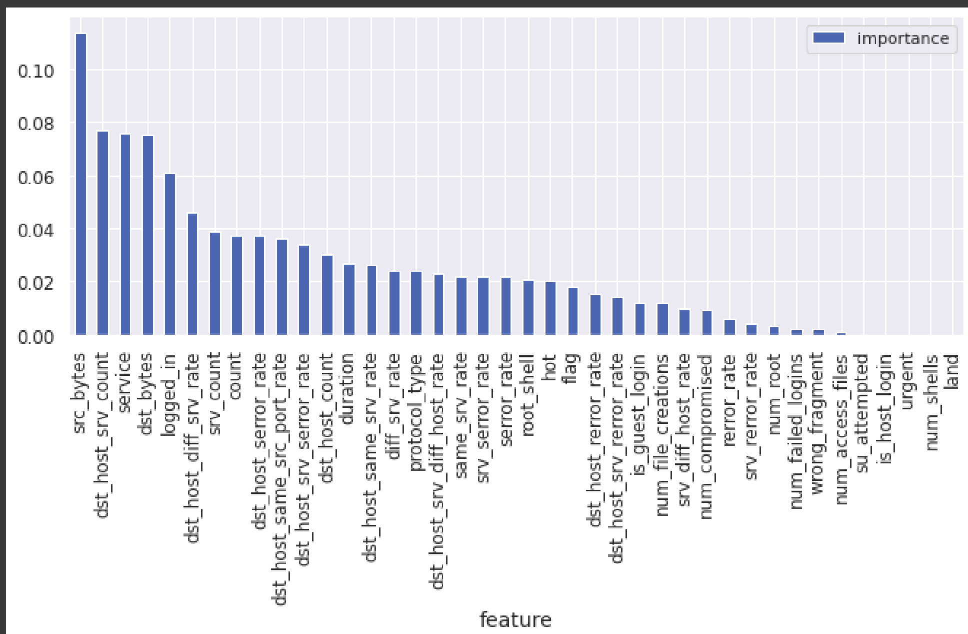
Data Sampling

```
[ ]  from imblearn.over_sampling import RandomOverSampler
     from collections import Counter

     # define columns and extract encoded train set for sampling
     sc_traindf = dfkdd_train.select_dtypes(include=['float64','int64'])
     refclasscol = pd.concat([sc_traindf, enctrain], axis=1).columns
     refclass = np.concatenate((sc_train, enctrain.values), axis=1)
     X = refclass

     # reshape target column to 1D array shape
     c, r = cat_Ytest.values.shape
     y_test = cat_Ytest.values.reshape(c,)

     c, r = cat_Ytrain.values.shape
     y = cat_Ytrain.values.reshape(c,)

     # apply the random over-sampling
     ros = RandomOverSampler(random_state=42)
     X_res, y_res = ros.fit_resample(X, y)
     print('Original dataset shape {}'.format(Counter(y)))
     print('Resampled dataset shape {}'.format(Counter(y_res)))
```

```
Original dataset shape Counter({1: 67343, 0: 45927, 2: 11656, 3: 995, 4: 52})
Resampled dataset shape Counter({1: 67343, 0: 67343, 3: 67343, 2: 67343, 4: 67343})
```

Feature Selection

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     rfc = RandomForestClassifier();

     # fit random forest classifier on the training set
     rfc.fit(X_res, y_res);
     # extract important features
     score = np.round(rfc.feature_importances_,3)
     importances = pd.DataFrame({'feature':refclasscol,'importance':score})
     importances = importances.sort_values('importance',ascending=False).set_index('feature')
     # plot importances
     plt.rcParams['figure.figsize'] = (11, 4)
     importances.plot.bar();
```



```
[ ]  from sklearn.feature_selection import RFE
     import itertools
     rfc = RandomForestClassifier()

     # create the RFE model and select 10 attributes
     rfe = RFE(rfc, n_features_to_select=10)
     rfe = rfe.fit(X_res, y_res)

     # summarize the selection of the attributes
     feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), refclasscol)]
     selected_features = [v for i, v in feature_map if i==True]
```

```
[ ]  selected_features
```

```
['src_bytes',
 'dst_bytes',
 'logged_in',
 'count',
 'srv_count',
 'dst_host_srv_count',
 'dst_host_diff_srv_rate',
 'dst_host_same_src_port_rate',
 'dst_host_serror_rate',
 'service']
```

Dataset Partition

```
[ ]  # define columns to new dataframe
     newcol = list(refclasscol)
     newcol.append('attack_class')

     # add a dimension to target
     new_y_res = y_res[:, np.newaxis]

     # create a dataframe from sampled data
     res_arr = np.concatenate((X_res, new_y_res), axis=1)
     res_df = pd.DataFrame(res_arr, columns = newcol)

     # create test dataframe
     reftest = pd.concat([sc_testdf, testcat], axis=1)
     reftest['attack_class'] = reftest['attack_class'].astype(np.float64)
     reftest['protocol_type'] = reftest['protocol_type'].astype(np.float64)
     reftest['flag'] = reftest['flag'].astype(np.float64)
     reftest['service'] = reftest['service'].astype(np.float64)

     res_df.shape
     reftest.shape
```

```
(22544, 41)
```

```
[ ]  from collections import defaultdict
     classdict = defaultdict(list)

     # create two-target classes (normal class and an attack class)
     attacklist = [('DoS', 0.0), ('Probe', 2.0), ('R2L', 3.0), ('U2R', 4.0)]
     normalclass = [('Normal', 1.0)]

     def create_classdict():
         '''This function subdivides train and test dataset into two-class attack labels'''
         for j, k in normalclass:
             for i, v in attacklist:
                 restrain_set = res_df.loc[(res_df['attack_class'] == k) | (res_df['attack_class'] == v)]
                 classdict[j +'_' + i].append(restrain_set)
                 # test labels
                 reftest_set = reftest.loc[(reftest['attack_class'] == k) | (reftest['attack_class'] == v)]
                 classdict[j +'_' + i].append(reftest_set)

     create_classdict()
```

```
[ ]  for k, v in classdict.items():
         k
```

```
[ ]  pretrain = classdict['Normal_DoS'][0]
     pretest = classdict['Normal_DoS'][1]
     grpclass = 'Normal_DoS'
```

Finalize data preprocessing for training

```
[ ]  from sklearn.preprocessing import OneHotEncoder
     enc = OneHotEncoder(handle_unknown='ignore')

     Xresdf = pretrain
     newtest = pretest

     Xresdfnew = Xresdf[selected_features]
     Xresdfnum = Xresdfnew.drop(['service'], axis=1)
     Xresdfcat = Xresdfnew[['service']].copy()

     Xtest_features = newtest[selected_features]
     Xtestdfnum = Xtest_features.drop(['service'], axis=1)
     Xtestcat = Xtest_features[['service']].copy()


     # Fit train data
     enc.fit(Xresdfcat)

     # Transform train data
     X_train_1hotenc = enc.transform(Xresdfcat).toarray()

     # Transform test data
     X_test_1hotenc = enc.transform(Xtestcat).toarray()

     X_train = np.concatenate((Xresdfnum.values, X_train_1hotenc), axis=1)
     X_test = np.concatenate((Xtestdfnum.values, X_test_1hotenc), axis=1)

     y_train = Xresdf[['attack_class']].copy()
     c, r = y_train.values.shape
     Y_train = y_train.values.reshape(c,)

     y_test = newtest[['attack_class']].copy()
     c, r = y_test.values.shape
     Y_test = y_test.values.reshape(c,)
```

Train Models

```
[ ]  from sklearn.svm import SVC
     from sklearn.naive_bayes import BernoulliNB
     from sklearn import tree
     from sklearn.model_selection import cross_val_score
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import VotingClassifier

     # Train KNeighborsClassifier Model
     KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
     KNN_Classifier.fit(X_train, Y_train);

     # Train LogisticRegression Model
     LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
     LGR_Classifier.fit(X_train, Y_train);

     # Train Gaussian Naive Baye Model
     BNB_Classifier = BernoulliNB()
     BNB_Classifier.fit(X_train, Y_train)

     # Train Decision Tree Model
     DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
     DTC_Classifier.fit(X_train, Y_train);

     # Train RandomForestClassifier Model
     #RF_Classifier = RandomForestClassifier(criterion='entropy', n_jobs=-1, random_state=0)
     #RF_Classifier.fit(X_train, Y_train);

     # Train SVM Model
     #SVC_Classifier = SVC(random_state=0)
     #SVC_Classifier.fit(X_train, Y_train)

     ## Train Ensemble Model (This method combines all the individual models above except RandomForest)
     #combined_model = [('Naive Baye Classifier', BNB_Classifier),
     #                  ('Decision Tree Classifier', DTC_Classifier),
     #                  ('KNeighborsClassifier', KNN_Classifier),
     #                  ('LogisticRegression', LGR_Classifier)
     #                 ]
     #VotingClassifier =  VotingClassifier(estimators = combined_model,voting = 'soft', n_jobs=-1)
     #VotingClassifier.fit(X_train, Y_train);
```

Evaluate Models

```
[ ]  from sklearn import metrics

     models = []
     #models.append(('SVM Classifier', SVC_Classifier))
     models.append(('Naive Baye Classifier', BNB_Classifier))
     models.append(('Decision Tree Classifier', DTC_Classifier))
     #models.append(('RandomForest Classifier', RF_Classifier))
     models.append(('KNeighborsClassifier', KNN_Classifier))
     models.append(('LogisticRegression', LGR_Classifier))
     #models.append(('VotingClassifier', VotingClassifier))

     for i, v in models:
         scores = cross_val_score(v, X_train, Y_train, cv=10)
         accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
         confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
         classification = metrics.classification_report(Y_train, v.predict(X_train))
         print()
         print('============================== {} {} Model Evaluation =============================='.format(grpclass, i))
         print()
         print ("Cross Validation Mean Score:" "\n", scores.mean())
         print()
         print ("Model Accuracy:" "\n", accuracy)
         print()
         print("Confusion matrix:" "\n", confusion_matrix)
         print()
         print("Classification report:" "\n", classification)
         print()
```

```
============================== Normal_DoS Naive Baye Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9737760413571536

Model Accuracy:
 0.9737686173767133

Confusion matrix:
 [[65346  1997]
 [ 1536 65807]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.98      0.97      0.97     67343
         1.0       0.97      0.98      0.97     67343

    accuracy                           0.97    134686
   macro avg       0.97      0.97      0.97    134686
weighted avg       0.97      0.97      0.97    134686



============================== Normal_DoS Decision Tree Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9997698368976862

Model Accuracy:
 0.9999480272634127

Confusion matrix:
 [[67343     0]
 [    7 67336]]

Classification report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     67343
         1.0       1.00      1.00      1.00     67343

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686
```

```
============================== Normal_DoS KNeighborsClassifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.99656981084704

Model Accuracy:
 0.9977577476500898

Confusion matrix:
 [[67287    56]
 [  246 67097]]

Classification report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     67343
         1.0       1.00      1.00      1.00     67343

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686



============================== Normal_DoS LogisticRegression Model Evaluation ==============================

Cross Validation Mean Score:
 0.9808072130256406

Model Accuracy:
 0.980836909552589

Confusion matrix:
 [[65532  1811]
 [  770 66573]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.99      0.97      0.98     67343
         1.0       0.97      0.99      0.98     67343

    accuracy                           0.98    134686
   macro avg       0.98      0.98      0.98    134686
weighted avg       0.98      0.98      0.98    134686
```

Test Models

```
[ ]   for i, v in models:
          accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
          confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
          classification = metrics.classification_report(Y_test, v.predict(X_test))
          print()
          print('============================== {} {} Model Test Results =============================='.format(grpclass, i))
          print()
          print ("Model Accuracy:" "\n", accuracy)
          print()
          print("Confusion matrix:" "\n", confusion_matrix)
          print()
          print("Classification report:" "\n", classification)
          print()
```

```
============================== Normal_DoS Naive Baye Classifier Model Test Results ==============================

Model Accuracy:
 0.8336536781408352

Confusion matrix:
 [[5487 1971]
 [ 885 8826]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.86      0.74      0.79      7458
         1.0       0.82      0.91      0.86      9711

    accuracy                           0.83     17169
   macro avg       0.84      0.82      0.83     17169
weighted avg       0.84      0.83      0.83     17169



============================== Normal_DoS Decision Tree Classifier Model Test Results ==============================

Model Accuracy:
 0.8165880365775525

Confusion matrix:
 [[5591 1867]
 [1282 8429]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.81      0.75      0.78      7458
         1.0       0.82      0.87      0.84      9711

    accuracy                           0.82     17169
   macro avg       0.82      0.81      0.81     17169
weighted avg       0.82      0.82      0.82     17169
```

```
============================== Normal_DoS KNeighborsClassifier Model Test Results ==============================

Model Accuracy:
 0.8666200710583027

Confusion matrix:
 [[5787 1671]
 [ 619 9092]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.90      0.78      0.83      7458
         1.0       0.84      0.94      0.89      9711

    accuracy                           0.87     17169
   macro avg       0.87      0.86      0.86     17169
weighted avg       0.87      0.87      0.86     17169



============================== Normal_DoS LogisticRegression Model Test Results ==============================

Model Accuracy:
 0.8418661541149747

Confusion matrix:
 [[5963 1495]
 [1220 8491]]

Classification report:
               precision    recall  f1-score   support

         0.0       0.83      0.80      0.81      7458
         1.0       0.85      0.87      0.86      9711

    accuracy                           0.84     17169
   macro avg       0.84      0.84      0.84     17169
weighted avg       0.84      0.84      0.84     17169
```

3.2 Deep learning approaches-based network intrusion detection systems

NIDS implementation using DL algorithms for NSL-KDD dataset

```
[ ]  import numpy as np
     import tensorflow as tf
     from sklearn.preprocessing import Normalizer
     from sklearn.preprocessing import LabelEncoder
     from sklearn.preprocessing import MinMaxScaler
     from tensorflow.keras.models import Sequential, load_model
     from tensorflow.keras.layers import Dense,Dropout,LSTM,Embedding,SimpleRNN, GRU,Activation,Flatten
     from tensorflow.keras.layers import Conv1D,MaxPool1D, Flatten
     from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,CSVLogger
     from tensorflow.keras.utils import plot_model
     from tensorflow.keras.preprocessing import sequence
     from sklearn.metrics import (precision_score, recall_score,confusion_matrix,f1_score, accuracy_score,mean_squared_error,mean_absolute_error)
     import pandas as pd
     from tensorflow.keras import callbacks
     from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
     import matplotlib.pyplot as plt
     import scikitplot as skplt
     #import scikitplot as skplt
```

```
[ ]  import tensorflow.keras
     print('keras: %s' % tensorflow.keras.__version__)

     keras: 2.2.4-tf
```

```
[ ]
     from pycaret.classification import *
```

```
[ ]  train_url = 'NSL_KDD_Train.csv'
     test_url = 'NSL_KDD_Test.csv'
```

```
[ ]  col_names = ["duration","protocol_type","service","flag","src_bytes",
         "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
         "logged_in","num_compromised","root_shell","su_attempted","num_root",
         "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
         "is_host_login","is_guest_login","count","srv_count","serror_rate",
         "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
         "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
         "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
         "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
         "dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]
```

```
[ ]  ds_train=pd.read_csv(train_url,header=None, names = col_names)
     ds_test=pd.read_csv(test_url, header=None, names = col_names)
```

```
[ ]  ds=pd.concat([ds_train,ds_test], axis=0)
```

```
ds.head(5)
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rat
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	0.03	0.17	0.00	0.00	0.0
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	0.60	0.88	0.00	0.00	0.0
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	0.05	0.00	0.00	1.00	1.0
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	0.00	0.03	0.04	0.03	0.0
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	0.00	0.00	0.00	0.00	0.0

5 rows × 42 columns

```
ds.loc[(ds.label != 'normal'),'label']='attack'
```

```
ds.shape
```

```
(148517, 42)
```

Feature engineering by pycaret

```
### dataset processed
dsp_tr=setup(data = ds, sampling=False,
             target = 'label',train_size=0.85,
             numeric_imputation = 'mean',
             categorical_features = ['protocol_type','service','flag'],
             normalize=True,normalize_method='minmax',
             feature_selection=True,feature_selection_threshold=0.2,
             silent = True)
```

Setup Succesfully Completed!

	Description	Value
0	session_id	5307
1	Target Type	Binary
2	Label Encoded	attack: 0, normal: 1
3	Original Data	(148517, 42)
4	Missing Values	False
5	Numeric Features	26
6	Categorical Features	15
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(148517, 42)
11	Transformed Train Set	(126239, 40)
12	Transformed Test Set	(22278, 40)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	True
16	Normalize Method	minmax
17	Transformation	False
18	Transformation Method	None
19	PCA	False
20	PCA Method	None
21	PCA Components	None
22	Ignore Low Variance	False
23	Combine Rare Levels	False
24	Rare Level Threshold	None
25	Numeric Binning	False
26	Remove Outliers	False
27	Outliers Threshold	None
28	Remove Multicollinearity	False
29	Multicollinearity Threshold	None
30	Clustering	False
31	Clustering Iteration	None
32	Polynomial Features	False
33	Polynomial Degree	None
34	Trignometry Features	False
35	Polynomial Threshold	None
36	Group Features	False
37	Feature Selection	True
38	Features Selection Threshold	0.200000
39	Feature Interaction	False
40	Feature Ratio	False
41	Interaction Threshold	None

```
[ ]  X_train=dsp_tr[2]
     X_test=dsp_tr[3]
     y_train=dsp_tr[4]
     y_test=dsp_tr[5]
```

```
[ ]
```

```
[ ]  trainX=np.array(X_train)
     testT=np.array(X_test)
     y_train = np.array(y_train)
     y_test = np.array(y_test)
```

```
[ ]   X_train = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
     X_test = np.reshape(testT, (testT.shape[0], 1, testT.shape[1]))
```

RNN(Recuurent Neural Networks)

Simple RNN

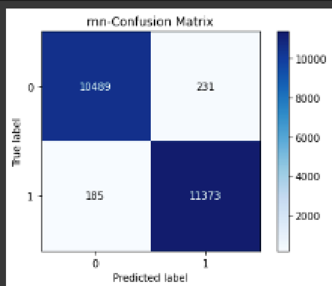
```
[ ]  batch_size = 32
     id=dsp_tr[2].shape[1]
     # 1. define the network
     model_rnn = Sequential()
     model_rnn.add(SimpleRNN(8,input_dim=id, return_sequences=True))
     model_rnn.add(Dropout(0.1))
     model_rnn.add(SimpleRNN(8, return_sequences=False))
     model_rnn.add(Dropout(0.1))
     model_rnn.add(Dense(1))
     model_rnn.add(Activation('sigmoid'))
```

```
[ ]
     model_rnn.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
     es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.0001,patience=5) ## early stoppoing
     Echeckpointer = callbacks.ModelCheckpoint(filepath="RNN-checkpoint-{epoch:02d}.hdf5", verbose=1, save_best_only=True, monitor='val_acc',mode='max')
     #csv_logger = CSVLogger('training_set_iranalysis1.csv',separator=',', append=False)
     model_rnn.fit(X_train, y_train, batch_size=batch_size, nb_epoch=1000, validation_data=(X_test, y_test),callbacks=[es])
     model_rnn.save("best_model_rnn.hdf5")
```

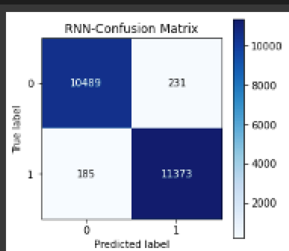
```
WARNING:tensorflow:The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 126239 samples, validate on 22278 samples
Epoch 1/1000
126239/126239 [==============================] - 20s 155us/sample - loss: 0.1125 - accuracy: 0.9561 - val_loss: 0.0981 - val_accuracy: 0.9591
Epoch 2/1000
126239/126239 [==============================] - 16s 126us/sample - loss: 0.1063 - accuracy: 0.9585 - val_loss: 0.0935 - val_accuracy: 0.9653
Epoch 3/1000
126239/126239 [==============================] - 16s 126us/sample - loss: 0.0998 - accuracy: 0.9607 - val_loss: 0.0875 - val_accuracy: 0.9649
Epoch 4/1000
126239/126239 [==============================] - 17s 131us/sample - loss: 0.0953 - accuracy: 0.9623 - val_loss: 0.0806 - val_accuracy: 0.9647
Epoch 5/1000
126239/126239 [==============================] - 18s 139us/sample - loss: 0.0920 - accuracy: 0.9634 - val_loss: 0.0793 - val_accuracy: 0.9688
Epoch 6/1000
126239/126239 [==============================] - 16s 130us/sample - loss: 0.0877 - accuracy: 0.9647 - val_loss: 0.0746 - val_accuracy: 0.9687
Epoch 7/1000
126239/126239 [==============================] - 17s 131us/sample - loss: 0.0854 - accuracy: 0.9659 - val_loss: 0.0715 - val_accuracy: 0.9690
Epoch 8/1000
126239/126239 [==============================] - 17s 137us/sample - loss: 0.0819 - accuracy: 0.9673 - val_loss: 0.0687 - val_accuracy: 0.9717
Epoch 9/1000
126239/126239 [==============================] - 17s 132us/sample - loss: 0.0802 - accuracy: 0.9675 - val_loss: 0.0666 - val_accuracy: 0.9722
Epoch 10/1000
126239/126239 [==============================] - 17s 131us/sample - loss: 0.0781 - accuracy: 0.9691 - val_loss: 0.0647 - val_accuracy: 0.9737
Epoch 11/1000
126239/126239 [==============================] - 17s 135us/sample - loss: 0.0763 - accuracy: 0.9697 - val_loss: 0.0623 - val_accuracy: 0.9752
Epoch 12/1000
126239/126239 [==============================] - 17s 134us/sample - loss: 0.0740 - accuracy: 0.9710 - val_loss: 0.0602 - val_accuracy: 0.9761
Epoch 13/1000
126239/126239 [==============================] - 18s 141us/sample - loss: 0.0723 - accuracy: 0.9712 - val_loss: 0.0602 - val_accuracy: 0.9750
Epoch 14/1000
126239/126239 [==============================] - 17s 137us/sample - loss: 0.0723 - accuracy: 0.9715 - val_loss: 0.0595 - val_accuracy: 0.9768
Epoch 15/1000
126239/126239 [==============================] - 17s 134us/sample - loss: 0.0708 - accuracy: 0.9721 - val_loss: 0.0579 - val_accuracy: 0.9789
Epoch 16/1000
126239/126239 [==============================] - 18s 143us/sample - loss: 0.0704 - accuracy: 0.9731 - val_loss: 0.0566 - val_accuracy: 0.9793
Epoch 17/1000
126239/126239 [==============================] - 17s 136us/sample - loss: 0.0691 - accuracy: 0.9735 - val_loss: 0.0572 - val_accuracy: 0.9792
Epoch 18/1000
126239/126239 [==============================] - 18s 141us/sample - loss: 0.0686 - accuracy: 0.9738 - val_loss: 0.0558 - val_accuracy: 0.9784
Epoch 19/1000
126239/126239 [==============================] - 18s 143us/sample - loss: 0.0679 - accuracy: 0.9737 - val_loss: 0.0548 - val_accuracy: 0.9794
Epoch 20/1000
126239/126239 [==============================] - 18s 146us/sample - loss: 0.0665 - accuracy: 0.9747 - val_loss: 0.0541 - val_accuracy: 0.9796
Epoch 21/1000
126239/126239 [==============================] - 18s 139us/sample - loss: 0.0667 - accuracy: 0.9747 - val_loss: 0.0543 - val_accuracy: 0.9798
Epoch 22/1000
```

```
[ ]  126239/126239 [==============================] - 19s 152us/sample - loss: 0.0620 - accuracy: 0.9771 - val_loss: 0.0506 - val_accuracy: 0.9818
     Epoch 37/1000
     126239/126239 [==============================] - 19s 148us/sample - loss: 0.0603 - accuracy: 0.9775 - val_loss: 0.0509 - val_accuracy: 0.9811
     Epoch 38/1000
     126239/126239 [==============================] - 20s 157us/sample - loss: 0.0624 - accuracy: 0.9768 - val_loss: 0.0519 - val_accuracy: 0.9813
```

```
[ ]
  y_pred_rnn = model_rnn.predict_classes(X_test)
  y_probs_rnn=model_rnn.predict_proba(X_test)
  np.savetxt('rnn_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_rnn), axis=1)), fmt='%01d')
  np.savetxt('rnn_prob_predictions.txt', np.around(np.transpose(y_probs_rnn),decimals=5), fmt='%.5f')



  ######Plot confusion matrix
```




```
[ ]  skplt.metrics.plot_confusion_matrix(y_test, y_pred_rnn)
     plt.title("RNN-Confusion Matrix")
     plt.rcParams['figure.figsize']=(5,4)
     plt.show()
```



```
[ ]  accuracy = accuracy_score(y_test, y_pred_rnn)
     print("accuracy:",accuracy)
     f1score=f1_score(y_test, y_pred_rnn)
     print("f1-acore:",f1score)
     cm=confusion_matrix(y_test, y_pred_rnn)
     print("confusion matrix:\n",cm)
     pr=precision_score(y_test,y_pred_rnn)
     print("Precision:",pr)
     rs=recall_score(y_test,y_pred_rnn)
     print("Recall_score:",rs)
```

```
accuracy: 0.9813268695574109
f1-acore: 0.9820395475347551
confusion matrix:
 [[10489   231]
 [  185 11373]]
Precision: 0.9800930713547052
Recall_score: 0.9839937705485378
```

```
[ ]  ### evaluate/calculate metrics  from saved model

     # load a saved model
     from tensorflow.keras.models import load_model
     saved_model = load_model('lstm_model.hdf5')
     loss,accur = saved_model.evaluate(X_test, y_test)
     print("\n Loss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
```

```
22278/22278 [==============================] - 2s 87us/sample - loss: 0.0198 - accuracy: 0.9926

 Loss: 0.02, Accuracy: 98.13%
```

LSTM

```
[ ]  id=dsp_tr[2].shape[1]
     batch_size=32
```

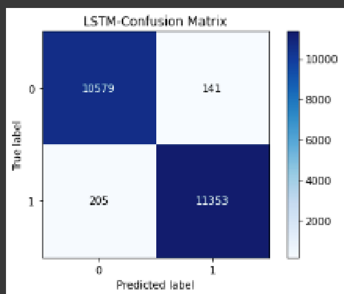
```
[ ]  model_lstm = Sequential()
     #model_dnn3.add(Flatten())
     model_lstm.add(LSTM(8,input_dim=id, return_sequences=True))
     model_lstm.add(Dropout(0.1))
     model_lstm.add(LSTM(8, return_sequences=False))
     model_lstm.add(Dropout(0.1))
     model_lstm.add(Dense(1))
     model_lstm.add(Activation('sigmoid'))
```

```
[ ]
```

```
[ ]  model_lstm.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
     es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.0001,patience=5) ## early stoppoing


     model_lstm.fit(X_train, y_train, batch_size=batch_size, nb_epoch=1000, validation_data=(X_test, y_test),callbacks=[es])
     model_lstm.save("lstm_model_lstm.hdf5")
```

```
WARNING:tensorflow:The `nb_epoch` argument in `fit` has been renamed `epochs`.
Train on 126239 samples, validate on 22278 samples
Epoch 1/1000
126239/126239 [==============================] - 32s 252us/sample - loss: 0.1512 - accuracy: 0.9500 - val_loss: 0.0964 - val_accuracy: 0.9637
Epoch 2/1000
126239/126239 [==============================] - 24s 189us/sample - loss: 0.0916 - accuracy: 0.9643 - val_loss: 0.0787 - val_accuracy: 0.9688
Epoch 3/1000
126239/126239 [==============================] - 25s 195us/sample - loss: 0.0779 - accuracy: 0.9689 - val_loss: 0.0689 - val_accuracy: 0.9725
Epoch 4/1000
126239/126239 [==============================] - 26s 207us/sample - loss: 0.0693 - accuracy: 0.9722 - val_loss: 0.0631 - val_accuracy: 0.9759
Epoch 5/1000
126239/126239 [==============================] - 24s 194us/sample - loss: 0.0633 - accuracy: 0.9751 - val_loss: 0.0566 - val_accuracy: 0.9777
Epoch 6/1000
```

```
[ ]
y_pred_lstm = model_lstm.predict_classes(X_test)
y_probs_lstm=model_lstm.predict_proba(X_test)
np.savetxt('lstm_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_lstm), axis=1)), fmt='%01d')
np.savetxt('lstm_prob_predictions.txt', np.around(np.transpose(y_probs_lstm),decimals=5), fmt='%.5f')


######Plot confusion matrix

skplt.metrics.plot_confusion_matrix(y_test, y_pred_lstm)
plt.title("LSTM-Confusion Matrix")
plt.show()
```




```
[ ] ### evaluate/calculate metrics  from saved model

# load a saved model
from tensorflow.keras.models import load_model
saved_model = load_model('lstm_model.hdf5')
loss,accur = saved_model.evaluate(X_test, y_test)
print("\n Loss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
```

```
22278/22278 [==============================] - 2s 85us/sample - loss: 0.0198 - accuracy: 0.9926

 Loss: 0.02, Accuracy: 98.13%
```

```
[ ] accuracy = accuracy_score(y_test, y_pred_lstm)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred_lstm)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred_lstm)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred_lstm)
print("Precision:",pr)
rs=recall_score(y_test,y_pred_lstm)
print("Recall_score:",rs)
#misclassified_samples = X_test[y_test != y_pred_lstm]
#mc=misclassified_samples.shape[0]
#print("Misclassified :",mc)
```

```
accuracy: 0.9844689828530389
f1-acore: 0.984990456359535
confusion matrix:
 [[10579   141]
 [  205 11353]]
Precision: 0.9877327301200627
Recall_score: 0.982263367364596
```

GRU

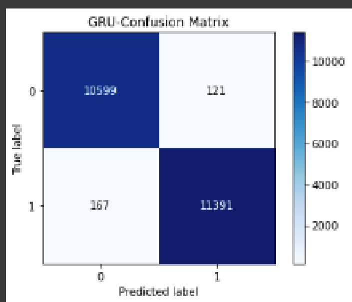
```
[ ]  batch_size = 64
     id=dsp_tr[2].shape[1]
```

```
[ ]  model_gru = Sequential()
     #model_gru.add(Flatten())
     model_gru.add(GRU(32,input_dim=id, return_sequences=True))
     model_gru.add(Dropout(0.1))
     model_gru.add(GRU(32, return_sequences=True))
     model_gru.add(Dropout(0.1))
     model_gru.add(GRU(32, return_sequences=True))
     model_gru.add(Dropout(0.1))
     model_gru.add(GRU(32, return_sequences=False))
     model_gru.add(Dropout(0.1))
     #model_gru.add(Flatten())
     model_gru.add(Dense(1))
     model_gru.add(Activation('sigmoid'))
```

```
[ ]  # try using different optimizers and different optimizer configs

     model_gru.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
     es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.0001,patience=5) ## early stoppoing

     model_gru.fit(X_train, y_train, validation_data=(X_test, y_test),batch_size=batch_size, epochs=1000, callbacks=[es])
     model_gru.save("model_gru.hdf5")
```

```
Train on 126239 samples, validate on 22278 samples
Epoch 1/1000
126239/126239 [==============================] - 35s 279us/sample - loss: 0.1303 - accuracy: 0.9505 - val_loss: 0.0873 - val_accuracy: 0.9631
Epoch 2/1000
126239/126239 [==============================] - 22s 173us/sample - loss: 0.0820 - accuracy: 0.9666 - val_loss: 0.0727 - val_accuracy: 0.9707
Epoch 3/1000
126239/126239 [==============================] - 23s 181us/sample - loss: 0.0674 - accuracy: 0.9728 - val_loss: 0.0587 - val_accuracy: 0.9787
Epoch 4/1000
126239/126239 [==============================] - 23s 181us/sample - loss: 0.0584 - accuracy: 0.9769 - val_loss: 0.0531 - val_accuracy: 0.9804
Epoch 5/1000
```

```
[ ]
y_pred_gru = model_gru.predict_classes(X_test)
y_probs_gru=model_gru.predict_proba(X_test)
np.savetxt('gru_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_gru), axis=1)), fmt='%01d')
np.savetxt('gru_prob_predictions.txt', np.around(np.transpose(y_probs_gru),decimals=5), fmt='%.5f')




######Plot confusion matrix

skplt.metrics.plot_confusion_matrix(y_test, y_pred_gru)
plt.title("GRU-Confusion Matrix")
plt.show()
```




```
[ ]
accuracy = accuracy_score(y_test, y_pred_gru)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred_gru)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred_gru)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred_gru)
print("Precision:",pr)
rs=recall_score(y_test,y_pred_gru)
print("Recall_score:",rs)
```

```
accuracy: 0.9870724481551306
f1-acore: 0.9875162548764 63
confusion matrix:
 [[10599   121]
 [  167 11391]]
Precision: 0.9894892286309938
Recall_score: 0.9855511334140855
```

Deep Neural networks

3 layers

```
[ ]  batch_size = 64
     id=dsp_tr[2].shape[1]
     # define network
     model_dnn3 = Sequential()
     model_dnn3.add(Flatten())
     model_dnn3.add(Dense(1024,input_dim=id,activation='relu'))
     model_dnn3.add(Dropout(0.01))
     model_dnn3.add(Dense(768,activation='relu'))
     model_dnn3.add(Dropout(0.01))

     model_dnn3.add(Dense(1))
     model_dnn3.add(Activation('sigmoid'))
```

```
[ ]  # try using different optimizers and different optimizer configs

     model_dnn3.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
     es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.0001,patience=5) ## early stoppoing

     model_dnn3.fit(X_train, y_train, validation_data=(X_test, y_test),batch_size=batch_size, epochs=1000, callbacks=[es])
     model_dnn3.save("dnn_two_layer_model_dnn3.hdf5")
```

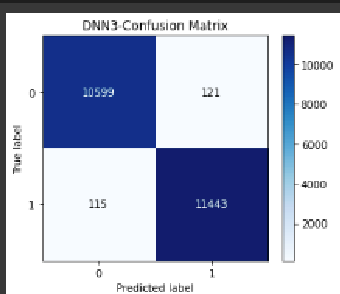
```
Train on 126239 samples, validate on 22278 samples
Epoch 1/1000
126239/126239 [==============================] - 37s 291us/sample - loss: 0.0678 - accuracy: 0.9736 - val_loss: 0.0470 - val_accuracy: 0.9832
Epoch 2/1000
126239/126239 [==============================] - 32s 255us/sample - loss: 0.0421 - accuracy: 0.9836 - val_loss: 0.0418 - val_accuracy: 0.9847
Epoch 3/1000
126239/126239 [==============================] - 31s 242us/sample - loss: 0.0371 - accuracy: 0.9856 - val_loss: 0.0347 - val_accuracy: 0.9880
Epoch 4/1000
126239/126239 [==============================] - 33s 261us/sample - loss: 0.0335 - accuracy: 0.9868 - val_loss: 0.0354 - val_accuracy: 0.9862
Epoch 5/1000
126239/126239 [==============================] - 35s 277us/sample - loss: 0.0313 - accuracy: 0.9881 - val_loss: 0.0376 - val_accuracy: 0.9863
Epoch 6/1000
126239/126239 [==============================] - 32s 252us/sample - loss: 0.0292 - accuracy: 0.9885 - val_loss: 0.0339 - val_accuracy: 0.9865
Epoch 7/1000
126239/126239 [==============================] - 31s 247us/sample - loss: 0.0275 - accuracy: 0.9893 - val_loss: 0.0493 - val_accuracy: 0.9831
Epoch 8/1000
126239/126239 [==============================] - 31s 245us/sample - loss: 0.0271 - accuracy: 0.9894 - val_loss: 0.0321 - val_accuracy: 0.9884
Epoch 9/1000
126239/126239 [==============================] - 31s 245us/sample - loss: 0.0258 - accuracy: 0.9896 - val_loss: 0.0345 - val_accuracy: 0.9879
Epoch 10/1000
126239/126239 [==============================] - 30s 238us/sample - loss: 0.0248 - accuracy: 0.9901 - val_loss: 0.0374 - val_accuracy: 0.9879
Epoch 11/1000
126239/126239 [==============================] - 30s 241us/sample - loss: 0.0248 - accuracy: 0.9901 - val_loss: 0.0329 - val_accuracy: 0.9880
Epoch 12/1000
126239/126239 [==============================] - 32s 252us/sample - loss: 0.0236 - accuracy: 0.9907 - val_loss: 0.0308 - val_accuracy: 0.9892
Epoch 13/1000
126239/126239 [==============================] - 38s 298us/sample - loss: 0.0233 - accuracy: 0.9910 - val_loss: 0.0293 - val_accuracy: 0.9903
Epoch 14/1000
126239/126239 [==============================] - 33s 258us/sample - loss: 0.0224 - accuracy: 0.9911 - val_loss: 0.0326 - val_accuracy: 0.9889
Epoch 15/1000
126239/126239 [==============================] - 32s 251us/sample - loss: 0.0219 - accuracy: 0.9913 - val_loss: 0.0335 - val_accuracy: 0.9886
Epoch 16/1000
126239/126239 [==============================] - 30s 241us/sample - loss: 0.0217 - accuracy: 0.9915 - val_loss: 0.0291 - val_accuracy: 0.9902
Epoch 17/1000
126239/126239 [==============================] - 35s 277us/sample - loss: 0.0209 - accuracy: 0.9914 - val_loss: 0.0300 - val_accuracy: 0.9901
Epoch 18/1000
126239/126239 [==============================] - 34s 271us/sample - loss: 0.0210 - accuracy: 0.9916 - val_loss: 0.0327 - val_accuracy: 0.9894
```

```
[ ]  y_pred_dnn3 = model_dnn3.predict_classes(X_test)
     y_probs_dnn3=model_dnn3.predict_proba(X_test)
     np.savetxt('dnn3_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_dnn3), axis=1)), fmt='%01d')
     np.savetxt('dnn3_prob_predictions.txt', np.around(np.transpose(y_probs_dnn3),decimals=5), fmt='%.5f')
```

```
[ ]  accuracy = accuracy_score(y_test, y_pred=y_pred_dnn3)
     print("accuracy:",accuracy)
     f1score=f1_score(y_test, y_pred=y_pred_dnn3)
     print("f1-acore:",f1score)
     cm=confusion_matrix(y_test, y_pred=y_pred_dnn3)
     print("confusion matrix:\n",cm)
     pr=precision_score(y_test,y_pred=y_pred_dnn3)
     print("Precision:",pr)
     rs=recall_score(y_test,y_pred=y_pred_dnn3)
     print("Recall_score:",rs)
```

```
accuracy: 0.9894065894604542
f1-acore: 0.9897932704783323
confusion matrix:
 [[10599   121]
 [  115 11443]]
Precision: 0.9895364925631269
Recall_score: 0.9900501816923343
```

```
[ ]  skplt.metrics.plot_confusion_matrix(y_test, y_pred_dnn3)
     plt.title("DNN3-Confusion Matrix")
     plt.rcParams['figure.figsize']=(5,4)
     plt.show()
```



DNN 4 layers

```
[ ]  batch_size = 64
     id=dsp_tr[2].shape[1]
     model_dnn4 = Sequential()
     model_dnn4.add(Flatten())
     model_dnn4.add(Dense(1024,input_dim=id,activation='relu'))
     model_dnn4.add(Dropout(0.01))
     model_dnn4.add(Dense(768,activation='relu'))
     model_dnn4.add(Dropout(0.01))
     model_dnn4.add(Dense(512,activation='relu'))
     model_dnn4.add(Dropout(0.01))
     model_dnn4.add(Dense(256,activation='relu'))
     model_dnn4.add(Dropout(0.01))
     model_dnn4.add(Dense(1))
     model_dnn4.add(Activation('sigmoid'))
```

```
[ ]  model_dnn4.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
     es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.0001,patience=5) ## early stoppoing

     model_dnn4.fit(X_train, y_train, validation_data=(X_test, y_test),batch_size=batch_size, epochs=1000, callbacks=[es])
     #model_dnn4.save("dnn4layer_model.hdf5")
```

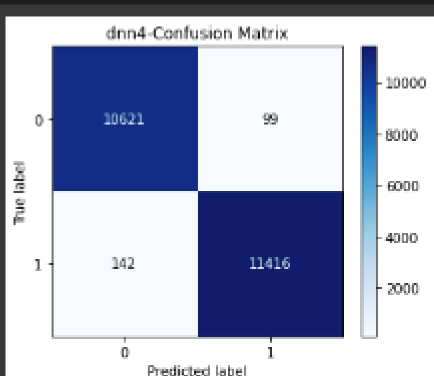
```
Train on 126239 samples, validate on 22278 samples
Epoch 1/1000
126239/126239 [==============================] - 64s 503us/sample - loss: 0.0213 - accuracy: 0.9920 - val_loss: 0.0360 - val_accuracy: 0.9883
Epoch 2/1000
115328/126239 [==========================>...] - ETA: 4s - loss: 0.0191 - accuracy: 0.9924
```

```
[ ]  y_pred_dnn4 = model_dnn4.predict_classes(X_test)
     y_probs_dnn4=model_dnn4.predict_proba(X_test)
     np.savetxt('dnn4_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_dnn4), axis=1)), fmt='%01d')
     np.savetxt('dnn4_prob_predictions.txt', np.around(np.transpose(y_probs_dnn4),decimals=5), fmt='%.5f')
```

```
[ ]  accuracy = accuracy_score(y_test, y_pred_dnn4)
     print("accuracy:",accuracy)
     f1score=f1_score(y_test, y_pred_dnn4)
     print("f1-acore:",f1score)
     cm=confusion_matrix(y_test, y_pred_dnn4)
     print("confusion matrix:\n",cm)
     pr=precision_score(y_test,y_pred_dnn4)
     print("Precision:",pr)
     rs=recall_score(y_test,y_pred_dnn4)
     print("Recall_score:",rs)
```

```
accuracy: 0.9891821527964808
f1-acore: 0.9895548909981363
confusion matrix:
 [[10621    99]
 [  142 11416]]
Precision: 0.9914025184541901
Recall_score: 0.9877141373940128
```

```
[ ]  skplt.metrics.plot_confusion_matrix(y_test, y_pred_dnn4)
     plt.title("dnn4-Confusion Matrix")
     plt.rcParams['figure.figsize']=(5,4)
     plt.show()
```



Convolutional Neural Networks

```
[ ]  import tensorflow.keras
     print('keras: %s' % tensorflow.keras.__version__)
```

```
keras: 2.2.4-tf
```

```
[ ]  X_train=dsp_tr[2]
     X_test=dsp_tr[3]
     y_train=dsp_tr[4]
     y_test=dsp_tr[5]
     trainX=np.array(X_train)
     testT=np.array(X_test)
     y_train = np.array(y_train)
     y_test = np.array(y_test)
```

```
[ ]  X_train = np.reshape(trainX, (trainX.shape[0],trainX.shape[1],1))
     X_test = np.reshape(testT, (testT.shape[0],testT.shape[1],1))
```

```
[ ]  #lstm_output_size = 128

     model_cnn = Sequential()
     model_cnn.add(Conv1D(64, 3, padding="same",activation="relu",input_shape=(id, 1)))
     model_cnn.add(Conv1D(64, 3, padding="same", activation="relu"))
     model_cnn.add(MaxPool1D(pool_size=(2)))
     model_cnn.add(Conv1D(128, 3, padding="same", activation="relu"))
     model_cnn.add(Conv1D(128, 3, padding="same", activation="relu"))
     model_cnn.add(MaxPool1D(pool_size=(2)))
     model_cnn.add(Flatten())
     model_cnn.add(Dense(128, activation="relu"))
     model_cnn.add(Dropout(0.5))
     model_cnn.add(Dense(1, activation="sigmoid"))
```

```
[ ]  model_cnn.compile(loss="binary_crossentropy", optimizer="adam",metrics=['accuracy'])
     es = EarlyStopping(monitor='val_accuracy', mode='max', min_delta=0.0001,patience=5) ## early stoppoing


     model_cnn.fit(X_train, y_train, epochs=1000,validation_data=(X_test, y_test),callbacks=[es])
     model_cnn.save("model_cnn_model.hdf5")
```

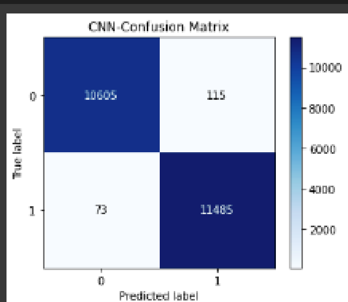
```
Train on 126239 samples, validate on 22278 samples
Epoch 1/1000
126239/126239 [==============================] - 104s 822us/sample - loss: 0.0747 - accuracy: 0.9711 - val_loss: 0.0453 - val_accuracy: 0.9843
Epoch 2/1000
126239/126239 [==============================] - 105s 833us/sample - loss: 0.0434 - accuracy: 0.9846 - val_loss: 0.0395 - val_accuracy: 0.9864
Epoch 3/1000
126239/126239 [==============================] - 95s 750us/sample - loss: 0.0378 - accuracy: 0.9866 - val_loss: 0.0338 - val_accuracy: 0.9877
Epoch 4/1000
126239/126239 [==============================] - 86s 683us/sample - loss: 0.0331 - accuracy: 0.9882 - val_loss: 0.0330 - val_accuracy: 0.9893
Epoch 5/1000
126239/126239 [==============================] - 92s 727us/sample - loss: 0.0309 - accuracy: 0.9892 - val_loss: 0.0323 - val_accuracy: 0.9890
Epoch 6/1000
```

[ ]
```python
y_pred_cnn = model_cnn.predict_classes(X_test)
y_probs_cnn=model_cnn.predict_proba(X_test)
np.savetxt('cnn_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_cnn), axis=1)), fmt='%01d')
np.savetxt('cnn_prob_predictions.txt', np.around(np.transpose(y_probs_cnn),decimals=5), fmt='%.5f')


######Plot confusion matrix

# skplt.metrics.plot_confusion_matrix(y_test, y_pred_cnn)
# plt.title("CNN-Confusion Matrix")
# plt.show()
```

[ ]
```python
accuracy = accuracy_score(y_test, y_pred_cnn)
print("accuracy:",accuracy)
f1score=f1_score(y_test, y_pred_cnn)
print("f1-acore:",f1score)
cm=confusion_matrix(y_test, y_pred_cnn)
print("confusion matrix:\n",cm)
pr=precision_score(y_test,y_pred_cnn)
print("Precision:",pr)
rs=recall_score(y_test,y_pred_cnn)
print("Recall_score:",rs)
```

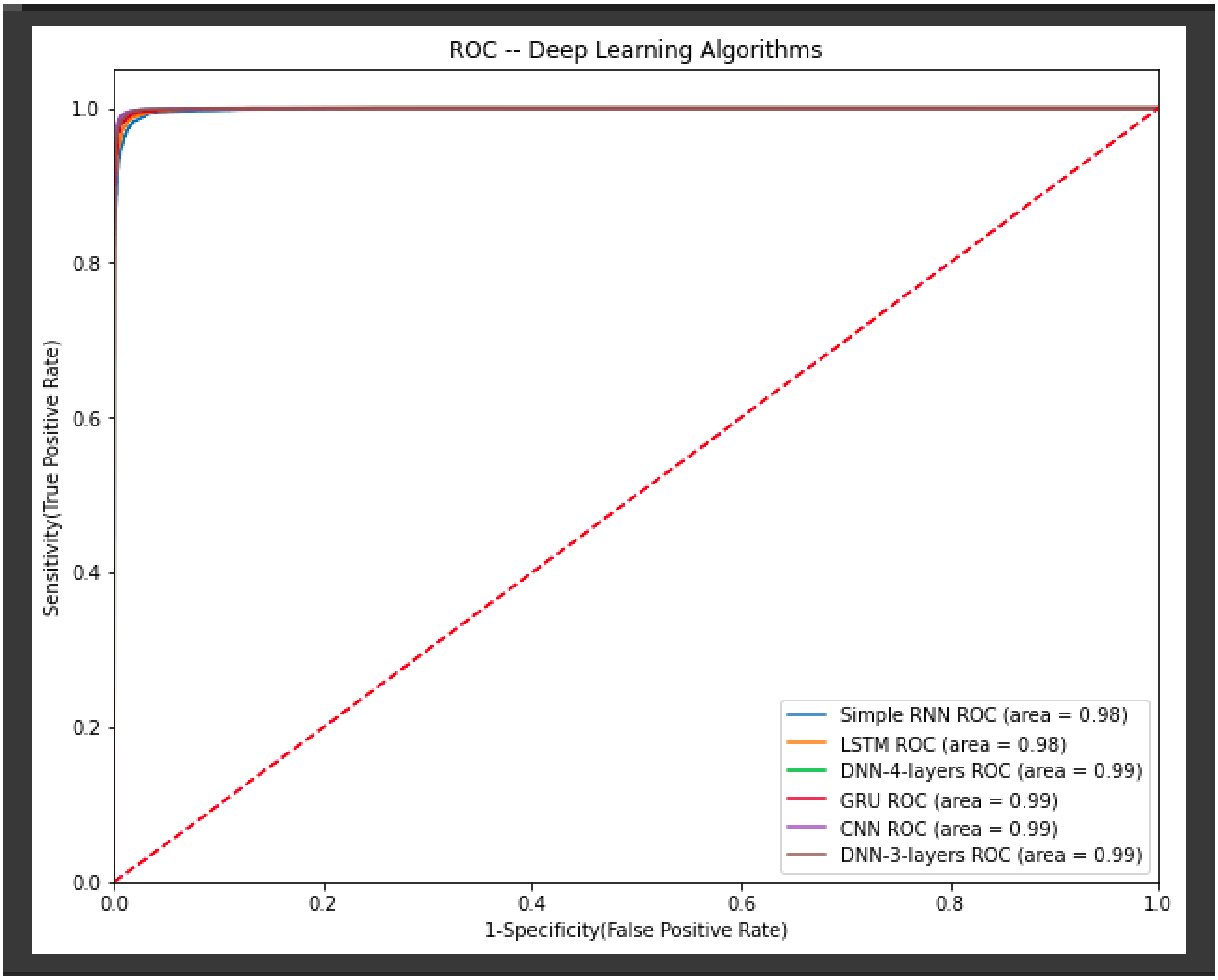
```
accuracy: 0.9915611814345991
f1-acore: 0.9918818550824768
confusion matrix:
 [[10605   115]
 [   73 11485]]
Precision: 0.9900862068965517
Recall_score: 0.9936840283786122
```

[ ]
```python
skplt.metrics.plot_confusion_matrix(y_test, y_pred_cnn)
plt.title("CNN-Confusion Matrix")
plt.rcParams['figure.figsize']=(5,4)
plt.show()
```



ROC Curves

```
from sklearn.metrics import roc_curve, auc,roc_auc_score
plt.figure()

# Add the models to the list that you want to view on the ROC plot
models = [
{
    'label': 'Simple RNN',
    'pred': y_pred_rnn,
    'model': model_rnn,
    'prob':y_probs_rnn
},
{
    'label': 'LSTM',
    'pred': y_pred_lstm,
    'model': model_lstm,
    'prob':y_probs_lstm

},
    {
    'label': 'DNN-4-layers',
    'pred': y_pred_dnn4,
    'model':model_dnn4,
    'prob' :y_probs_dnn4

},
    {
    'label': 'GRU',
    'pred': y_pred_gru,
    'model': model_gru,
    'prob':y_probs_gru
},
    {
    'label': 'CNN',
    'pred': y_pred_cnn,
    'model': model_cnn,
    'prob':y_probs_cnn
},
 {
    'label': 'DNN-3-layers',
    'pred': y_pred_dnn3,
    'model': model_dnn3,
     'prob':y_probs_dnn3
},
]

# Below for loop iterates through your models list
for m in models:
    model = m['model'] # select the model
    #model.fit(X_train, y_train) # train the model
#     y_pred=model.predict(X_test) # predict the test data
# Compute False postive rate, and True positive rate
    #fpr, tpr, thresholds =  roc_curve(y_test, model.predict_proba(X_test)[:,1])
    fpr, tpr, thresholds =  roc_curve(y_test, m['prob'])
# Calculate Area under the curve to display on the plot
    auc =  roc_auc_score(y_test,m['pred'])
# Now, plot the computed values
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))
# Custom settings for the plot
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('ROC -- Deep Learning Algorithms')
plt.legend(loc="lower right")
plt.rcParams['figure.figsize']=(16,8)
plt.show()   # Display
```


ROC -- Deep Learning Algorithms
Sensitivity(True Positive Rate)
1-Specificity(False Positive Rate)
0.0
0.2
0.4
0.6
0.8
1.0
Simple RNN ROC (area = 0.98)
LSTM ROC (area = 0.98)
DNN-4-layers ROC (area = 0.99)
GRU ROC (area = 0.99)
CNN ROC (area = 0.99)
DNN-3-layers ROC (area = 0.99)

4. Proposed Methodologies:

```python
from google.colab import drive
import pandas as pd
import numpy as np
from sklearn.preprocessing import (StandardScaler, OrdinalEncoder,LabelEncoder, MinMaxScaler, OneHotEncoder)
!pip install -q keras
from tensorflow.keras.utils import to_categorical
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer, MaxAbsScaler , RobustScaler, PowerTransformer
import matplotlib.pyplot as plt
import seaborn as sns



drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
#train+, test+ and test21- dataset loading
train='/content/drive/MyDrive/Colab Notebooks/KDDTrain.txt'
test='/content/drive/MyDrive/Colab Notebooks/KDDTest.txt'
test21='/content/drive/MyDrive/Colab Notebooks/KDDTest-21.txt'

featureV=["duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fragment","urgent","hot",
          "num_failed_logins","logged_in","num_compromised","root_shell","su_attempted","num_root","num_file_creations","num_shells",
          "num_access_files","num_outbound_cmds","is_host_login","is_guest_login","count","srv_count","serror_rate","srv_serror_rate",
          "rerror_rate","srv_rerror_rate","same_srv_rate","diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
          "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate","dst_host_srv_diff_host_rate","dst_host_serror_rate",
          "dst_host_srv_serror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate","label","difficulty"]

flagV=['OTH','RSTOS0','SF','SH','RSTO','S2','S1','REJ','S3','RSTR','S0']

protocol_typeV=['tcp','udp','icmp']

serviceV=['http','smtp','finger','domain_u','auth','telnet','ftp','eco_i','ntp_u','ecr_i','other','private','pop_3','ftp_data',
                 'rje','time','mtp','link','remote_job','gopher','ssh','name','whois','domain','login','imap4','daytime','ctf','nntp',
                 'shell','IRC','nnsp','http_443','exec','printer','efs','courier','uucp','klogin','kshell','echo','discard','systat',
                 'supdup','iso_tsap','hostnames','csnet_ns','pop_2','sunrpc','uucp_path','netbios_ns','netbios_ssn','netbios_dgm',
                 'sql_net','vmnet','bgp','Z39_50','ldap','netstat','urh_i','X11','urp_i','pm_dump','tftp_u','tim_i','red_i','icmp',
                 'http_2784','harvest','aol','http_8001']

binary_attack=['normal','ipsweep', 'nmap', 'portsweep','satan', 'saint', 'mscan','back', 'land', 'neptune', 'pod', 'smurf',
               'teardrop', 'apache2', 'udpstorm', 'processtable','mailbomb','buffer_overflow', 'loadmodule', 'perl', 'rootkit',
               'xterm', 'ps', 'sqlattack','ftp_write', 'guess_passwd', 'imap', 'multihop','phf', 'spy', 'warezclient',
               'warezmaster','snmpgetattack','named', 'xlock', 'xsnoop','sendmail', 'httptunnel', 'worm', 'snmpguess']

multiclass_attack={ 'normal': 'normal',
        'probe': ['ipsweep.', 'nmap.', 'portsweep.','satan.', 'saint.', 'mscan.'],
        'dos': ['back.', 'land.', 'neptune.', 'pod.', 'smurf.','teardrop.', 'apache2.', 'udpstorm.', 'processtable.','mailbomb.'],
        'u2r': ['buffer_overflow.', 'loadmodule.', 'perl.', 'rootkit.','xterm.', 'ps.', 'sqlattack.'],
        'r2l': ['ftp_write.', 'guess_passwd.', 'imap.', 'multihop.','phf.', 'spy.', 'warezclient.', 'warezmaster.','snmpgetattack.',
                'named.', 'xlock.', 'xsnoop.','sendmail.', 'httptunnel.', 'worm.', 'snmpguess.']}
```

```
[ ]   train_data=pd.read_csv(train,names=featureV)
      test_data=pd.read_csv(test,names=featureV)
```

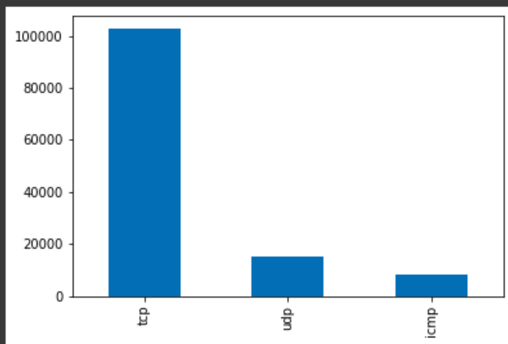
```
[ ]   test_21 = pd.read_csv(test21, names= featureV)
```

```
[ ]   train_data = train_data.query("service != 'aol'")
      train_data = train_data.query("service != 'harvest'")
      train_data = train_data.query("service != 'http_2784'")
      train_data = train_data.query("service != 'http_8001'")
      train_data = train_data.query("service != 'red_i'")
      train_data = train_data.query("service != 'urh_i'")
      train_data = train_data.query("service != 'printer'")
      train_data = train_data.query("service != 'rje'")


      #-------------------------------------------------------------------------------->>>>

      test_data = test_data.query("service != 'printer'")
      test_data = test_data.query("service != 'rje'")
```

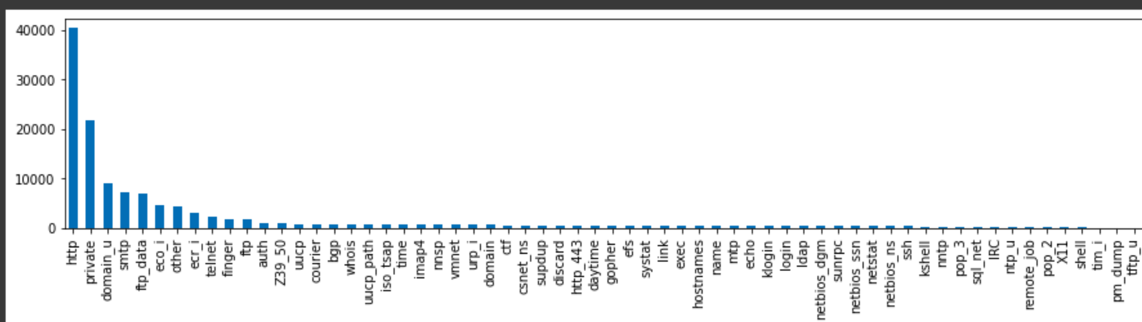
Working with **Training** Data

```
def bar_graph(feature):
    train_data[feature].value_counts().plot(kind="bar")
```

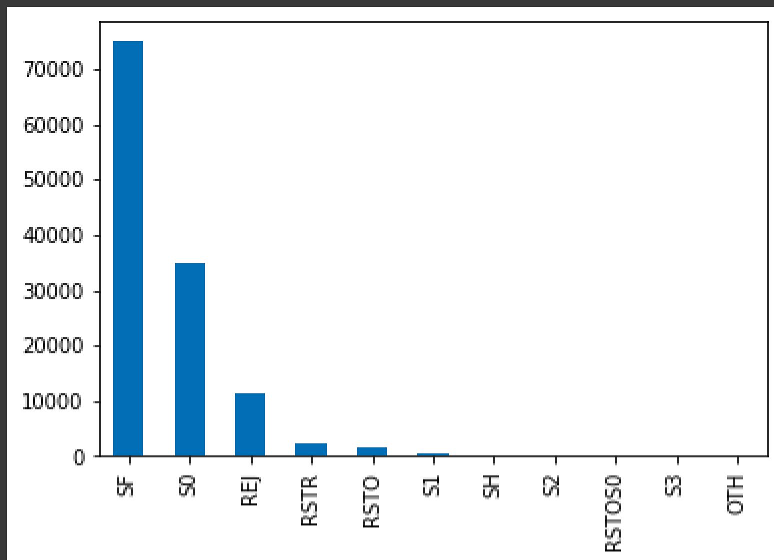
```
bar_graph('protocol_type')
```



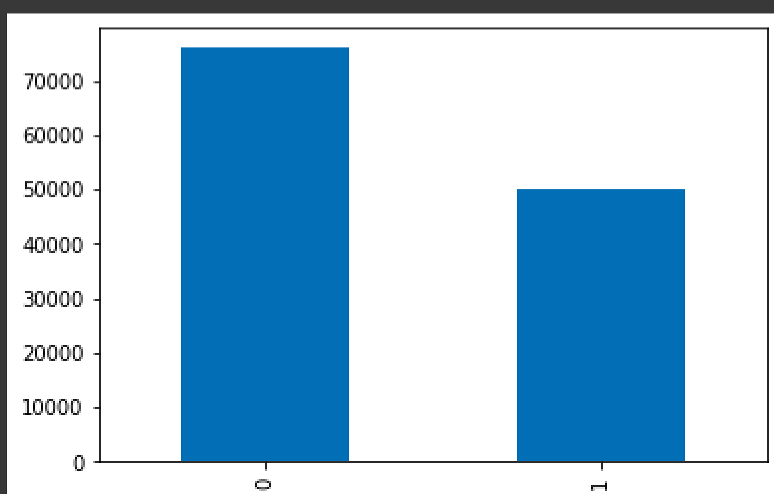

```
plt.figure(figsize=(15,3))
bar_graph('service')
```



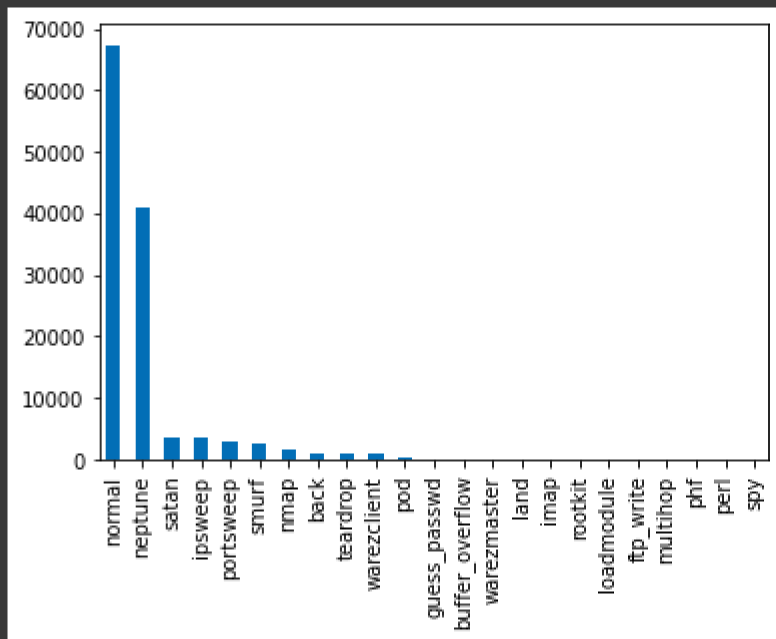
```
bar_graph('flag')
```




```
[ ]  bar_graph('logged_in')
```



```
bar_graph('label')
```



Working with **Validation** data

```
def preprocessing(data,cls,df):

#---------attack categorization--------------->
  data['label']=data['label'].replace(['normal.','normal'],0)

#--------------------binary classification------------------->
  if cls=='binary':
    for i in range(len(binary_attack)):
      data['label'] = data['label'].replace(binary_attack[i], 1)

#-------------multiclass classification------------------>
  elif cls=='multiclass':
    for i in range(len(multiclass_attack['probe'])):
      data['label']=data['label'].replace([multiclass_attack['probe'][i],multiclass_attack['probe'][i][:-1]],1)

    for i in range(len(multiclass_attack['dos'])):
      data['label']=data['label'].replace([multiclass_attack['dos'][i],multiclass_attack['dos'][i][:-1]],2)

    for i in range(len(multiclass_attack['u2r'])):
      data['label']=data['label'].replace([multiclass_attack['u2r'][i],multiclass_attack['u2r'][i][:-1]],3)

    for i in range(len(multiclass_attack['r2l'])):
      data['label'] = data['label'].replace([multiclass_attack['r2l'][i],multiclass_attack['r2l'][i][:-1]],4)
#--------------------------------------------->


#-----------------------splitting features and labels--------------->
  y=data['label']
  x=data.loc[:,'duration':'hot']
#------------------------------------------------------------------->
```

```python
#-----------------converting to binary feature vectors--------------------------------------->
  t=x.protocol_type.copy()
  t=pd.get_dummies(t)
  x=x.drop(columns='protocol_type',axis=1)
  x=x.join(t)

  t1=x.service.copy()
  t1=pd.get_dummies(t1)
  x=x.drop(columns='service',axis=1)
  x=x.join(t1)

  t2=x.flag.copy()
  t2=pd.get_dummies(t2)
  x=x.drop(columns='flag',axis=1)
  x=x.join(t2)
#------------------------------------------------------------------------------->

#-----------------converting to binary label vectors------------------------------------->
  yt=y.copy()
  yt=pd.get_dummies(yt)
#------------------------------------------------------------------------------->

  x = MinMaxScaler(feature_range=(0, 1)).fit_transform(x)
  # x = MaxAbsScaler().fit_transform(x)
  # x = StandardScaler().fit_transform(x)
  #print(x)
  #vectorizer = CountVectorizer()
  #p= vectorizer.fit_transform(x.protocol_type)

  #print(np.shape(p))
  #scaler = Normalizer(norm='l1').fit(x)
  #trainX = scaler.transform(x)
  if df=='train':
    return x,yt
  else:
    return x,y
```

```python
[ ]  x_train,Y_train=preprocessing(train_data,cls='binary',df='train')
     x_test,Y_test=preprocessing(test_data,cls='binary',df='test')
```

```
[ ]  x_21_test, y_21_test = preprocessing(test_21, cls = 'binary', df = 'test21')
```

```
[ ]  print(np.shape(x_train))
     print(np.shape(Y_train))
     print(np.shape(x_test))
     print(np.shape(Y_test))
     print(np.shape(x_21_test))
     print(np.shape(y_21_test))
```

```
(125793, 83)
(125793, 2)
(22525, 83)
(22525,)
(11850, 83)
(11850,)
```

```
[ ]  x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
     x_train.shape
```

```
(125793, 83, 1)
```

```
[ ]  x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
     x_test.shape
```

```
(22525, 83, 1)
```

```
[ ]  x_21_test = np.reshape(x_21_test, (x_21_test.shape[0], x_21_test.shape[1], 1))
     x_21_test.shape
```

```
(11850, 83, 1)
```

Model

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, SimpleRNN , GRU , Activation
from tensorflow.keras.layers import BatchNormalization
from keras import optimizers
import tensorflow as tf
from keras.layers import Convolution1D, Dense, Dropout, Flatten, MaxPooling1D , AveragePooling1D
```

```
model = Sequential()

model.add(Convolution1D(32, 3, padding="same",activation="relu",input_shape = (x_train.shape[1], 1)))
# model.add(Convolution1D(32, 3, activation="relu"))
model.add(MaxPooling1D(pool_size=(4)))
# model.add(tf.keras.layers.LayerNormalization())
# model.add(GRU(units = 128, return_sequences=True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.5))

model.add(Convolution1D(64, 3, padding="same",activation="relu"))
# model.add(Convolution1D(64, 3,activation="relu"))
model.add(MaxPooling1D(pool_size=(2)))
model.add(Dropout(0.5))
# model.add(tf.keras.layers.LayerNormalization())

# model.add(Convolution1D(64, 3, padding="same",activation="relu"))
# model.add(Convolution1D(64, 3,activation="relu"))
# model.add(AveragePooling1D(pool_size=(2)))
# model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(2, activation="softmax"))
```

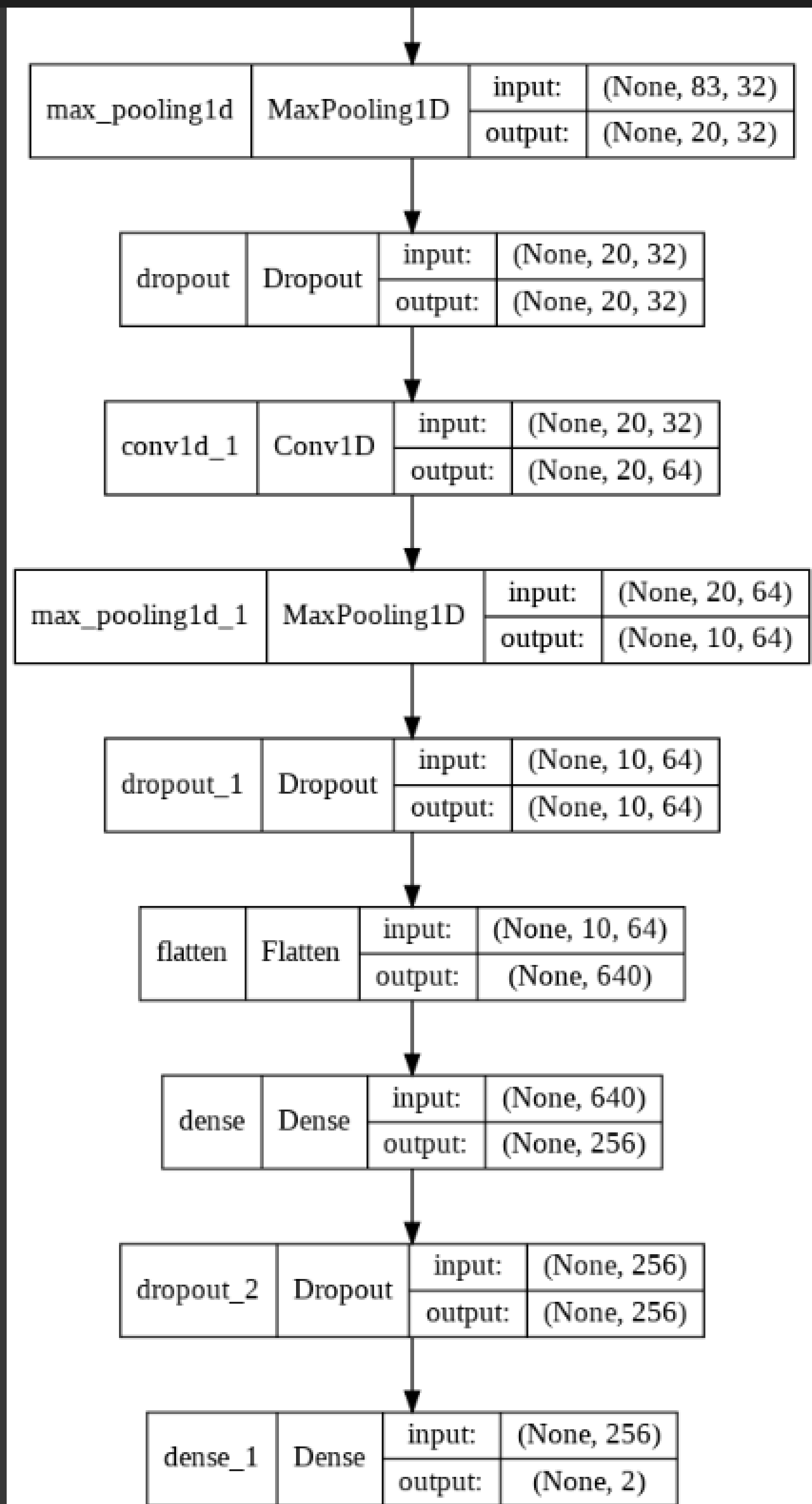
```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
```


```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
```


```
# tf.keras.optimizers.Adam(
#     learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
# train_op = tf.keras.optimizers.adam(amsgrad=True)
model.compile(optimizer ='adam',loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(x_train, Y_train, epochs = 100, batch_size = 128)
```

```
Epoch 1/100
983/983 [==============================] - 14s 13ms/step - loss: 0.1791 - accuracy: 0.9412
Epoch 2/100
983/983 [==============================] - 13s 13ms/step - loss: 0.1529 - accuracy: 0.9513
Epoch 3/100
983/983 [==============================] - 13s 13ms/step - loss: 0.1469 - accuracy: 0.9527
Epoch 4/100
983/983 [==============================] - 13s 13ms/step - loss: 0.1437 - accuracy: 0.9540
Epoch 5/100
```

```
tf.keras.utils.plot_model(model, 'my_first_model.png', show_shapes=True)
```



```
[ ]  pred = model.predict(x_test)
     y_pred= np.argmax(pred, axis = 1)
```

Evaluation of the model

```
[ ]  from sklearn.metrics import confusion_matrix,accuracy_score
     from sklearn.metrics import (precision_score, recall_score,
                                  f1_score, accuracy_score,mean_squared_error,mean_absolute_error)
```

```
[ ]  confusion_matrix(Y_test, y_pred)
```

```
array([[8728,  983],
       [3626, 9188]])
```

```
[ ]  accuracy =accuracy_score(Y_test, y_pred)*100
     print(accuracy)
```

```
79.53829078801333
```

```
[ ]  print(y_pred)
     # print(len(y_pred))
```

```
[1 1 0 ... 1 0 1]
```

```
[ ]  print(y_21_test)
```

```
0        1
1        1
2        1
3        0
4        1
        ..
11845    0
11846    0
11847    1
11848    1
11849    1
Name: label, Length: 11850, dtype: int64
```

```
[ ]  pred = model.predict(x_21_test)
     y_pred= np.argmax(pred, axis = 1)
```

```
[ ]  confusion_matrix(y_21_test, y_pred)
```

```
array([[1209,  943],
       [3626, 6072]])
```

```
[ ]  print(y_pred)
```

```
[0 1 0 ... 1 1 1]
```

```
[ ]  acc_21 = accuracy_score(y_21_test, y_pred)* 100
     print(acc_21)
```

```
61.44303797468355
```

```
[ ]  recall = recall_score(y_21_test, y_pred , average="binary")
     precision = precision_score(y_21_test, y_pred , average="binary")
     f1 = f1_score(y_21_test, y_pred, average="binary")
```

```
[ ]    print("accuracy")
       print("%.3f" %acc_21)
       print("racall")
       print("%.3f" %recall)
       print("precision")
       print("%.3f" %precision)
       print("f1score")
       print(f1)
```

```
accuracy
61.443
racall
0.626
precision
0.866
f1score
0.7266199964099803
```

```
[ ]  print("F-Score : ", f1*100)
     print("Precision : " , precision*100)
     print("Recall : ", recall*100)
     print("Accuracy : ",acc_21)
```

```
F-Score :  72.66199964099803
Precision :  86.55737704918033
Recall :  62.61084759744278
Accuracy :  61.44303797468355
```

5. Conclusion:

Our experiment results say that in both binary classification and multiclass classification, the intrusion detection model using neural networks achieve higher accuracy then traditional machine learning models using the same dataset.

Our suggested improvised cnn model can be a better alternative to the existing system. Though our models require more computation time, additional hardware can reduce that to a great extent.

6. Future Work:

Using the research done in paper , redundant and irrelevant features can be removed, which can significantly improve classifier performance. By identifying relevant features inside the dataset, accuracy increases. Furthermore, the authors suggested the use of UNSW-NB15, which removes the inherited issues of KDDCup 99 and NSL-KDD.

Finally, Principal component analysis (PCA) can be used, as shown in paper , to drastically reduce the number of features, training and testing time. Combining these three ideas, we believe that we can get further accuracy and performance improvements to our models.

References:

[1] A. Sahasrabuddhe, S. Naikade, A. Ramaswamy, B. Sadliwala and P. R. Futane, "Survey on intrusion detection system using data mining techniques," International Research Journal of Engineering and Technology, vol. 4, no. 5, pp. 1780–1784, 2017.

[2] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe and Y. A. Amal, "Intrusion detection model using machine learning algorithm on big data environment," Journal of Big Data, vol. 5, no. 1, pp. 521, 2018.

[3] L. Dali, A. Bentajer, E. Abdelmajid, K. Abou El Mehdi, H. Elsayed et al., "A survey of intrusion detection systems," in 2nd World Sym. on Web Applications and Networking, Tunisia, Piscataway: IEEE,pp. 1–6, 2015.

[4] K. Peng, V. C. M. Leung, L. Zheng, S. Wang, C. Huang et al., "Intrusion detection system based on decision tree over big data in fog environment," Wireless Communications and Mobile Computing, vol.2018, no. 5, pp. 1–10, 2018.

[5] T. Rupa Devi and S. Badugu, "A review on network intrusion detection system using machine learning," in Int. Conf. on Emerging Trends in Engineering 2019, LAIS, Switzerland: Springer Nature Switzerland, vol. 4, pp. 598–607, 2020.

[6] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for Internet of Things," Future Generation Computer Systems, vol. 82, pp. 761–768, 2017.

[7] H. Liu. and B. Lang, "Machine learning and deep learning methods for intrusion detection systems:A survey," Applied Sciences, vol. 9, no. 20, pp. 4396, 2019.

[8] F. Y. Yavuz, D. Ünal and E. Gul, "Deep learning for detection of routing attacks in the Internet ofThings," International Journal of Computational Intelligence Systems, vol. 12, no. 1, pp. 39–58, 2018.

[9] M. Panda and M. R. Patra, "A comparative study of data mining algorithms for network intrusion detection," in First Int. Conf. on Emerging Trends in Engineering and Technology, Nagpur,Maharashtra, 2008.

[10] K. Nalavade and B. B. Meshram, "Mining association rules to evade network intrusion in network audit data," International Journal of Advanced Computer Research, vol. 4, no. 2, pp. 560–567, 2014.

[11] Chapaneri, R., & Shah, S. (2019). A comprehensive survey of machine learning-based network intrusion detection. Smart Intelligent Computing and Applications, 345-356.

[11] Mishra, P., Varadharajan, V., Tupakula, U., & Pilli, E. S. (2018). A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Communications Surveys & Tutorials, 21(1), 686-728.

[12] Khan, F. A., & Gumaei, A. (2019, July). A comparative study of machine learning classifiers for network intrusion detection. In International Conference on Artificial Intelligence and Security (pp. 75-86). Springer, Cham.

[13] Gurung, S., Ghose, M. K., & Subedi, A. (2019). Deep learning approach on network intrusion detection system using NSL-KDD dataset. International Journal of Computer Network and Information Security, 11(3), 8-14.

[14] Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE Access, 7, 41525-41550.

[15] Jiang, K., Wang, W., Wang, A., & Wu, H. (2020). Network intrusion detection combined hybrid sampling with deep hierarchical network. IEEE Access, 8, 32464-32476.

[16] Su, T., Sun, H., Zhu, J., Wang, S., & Li, Y. (2020). BAT: Deep learning methods on network intrusion detection using NSL-KDD dataset. IEEE Access, 8, 29575-29585.

[17] A. Adebowale, S. A. Idowu and A. Amarachi, "Comparative study of selected data mining algorithms used for intrusion detection," International Journal of Soft Computing and Engineering, vol. 3, no. 3, pp. 237–241, 2013.

[18] S.Devaraju and S. Ramakrishnan, "Detection of attacks for ids using association rule mining algorithm," IETE Journal of Research, vol. 61, no. 6, pp. 624–633, 2015.

[19] Hu, Q., Yu, S. Y., & Asghar, M. R. (2020). Analysing performance issues of open-source intrusion detection systems in high-speed networks. Journal of Information Security and Applications, 51, 102426.

[20] Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications, 50, 102419.

[21] Hindy, H., Brosset, D., Bayne, E., Seeam, A. K., Tachtatzis, C., Atkinson, R., & Bellekens, X. (2020). A taxonomy of network threats and the effect of current datasets on intrusion detection systems. IEEE Access, 8, 104650-104675.

[22] She, C., Ma, Y., Jia, L., Fei, L., & Kou, B. (2016). Intrusion-detection model integrating anomaly with misuse for space information network. Journal of communications and information networks, 1(3), 90-96.

[23] Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. Ieee Access, 5, 21954- 21961.

[24] Huang, K., Zhang, Q., Zhou, C., Xiong, N., & Qin, Y. (2017). An efficient intrusion detection approach for visual sensor networks based on traffic pattern learning. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47(10), 2704-2713.

[25] Yang, H., & Wang, F. (2019). Wireless network intrusion detection based on improved convolutional neural network. Ieee Access, 7, 64366-64374.

[26] Hu, Z., Wang, L., Qi, L., Li, Y., & Yang, W. (2020). A novel wireless network intrusion detection method based on adaptive synthetic sampling and an improved convolutional neural network. IEEE Access, 8, 195741-195751.

[27] Wei, P., Li, Y., Zhang, Z., Hu, T., Li, Z., & Liu, D. (2019). An optimization method for intrusion detection classification model based on deep belief network. IEEE Access, 7, 87593-87605

[28] Xiao, Y., Xing, C., Zhang, T., & Zhao, Z. (2019). An intrusion detection model based on feature reduction and convolutional neural networks. IEEE Access, 7, 42210-42219.

[29] Zeng, Y., Gu, H., Wei, W., & Guo, Y. (2019). Deep-full-range: A deep learning based network encrypted traffic classification and intrusion detection framework. IEEE Access, 7, 45182-45190.

[29] Anthi, E., Williams, L., Słowińska, M., Theodorakopoulos, G., & Burnap, P. (2019). A supervised intrusion detection system for smart home IoT devices. IEEE Internet of Things Journal, 6(5), 9042-9053

[30] Yang, H., Qin, G., & Ye, L. (2019). Combined wireless network intrusion detection model based on deep learning. IEEE Access, 7, 82624-82632.

[31] Viegas, E., Santin, A. O., & Abreu Jr, V. (2020). Machine Learning Intrusion Detection in Big Data Era: A Multi-Objective Approach for Longer Model Lifespans. IEEE Transactions on Network Science and Engineering, 8(1), 366-376.

[32] Zhong, W., Yu, N., & Ai, C. (2020). Applying big data based deep learning system to intrusion detection. Big Data Mining and Analytics, 3(3), 181- 195.

[33] Wu, K., Chen, Z., & Li, W. (2018). A novel intrusion detection model for a massive network using convolutional neural networks. Ieee Access, 6, 50850-50859.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

[35] Wenjie Zhang, Dezhi Han, Kuan-Ching Li, Francisco Isidro Massetto. "Wireless sensor network intrusion detection system based on MK-ELM" , Soft Computing, 2020

[36] "Cloud Computing – CLOUD 2018" , Springer Science and Business Media LLC, 2018

[37] Saporito, Gerry. "A Deeper Dive into the NSL-KDD Data Set" Medium, https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set- 15c753364657. Accessed 17 September 2019.

[38] Y. Li, Y. Xu, Z. Liu, H. Hou, Y. Zheng, Y. Xin, et al., "Robust detection for network intrusion of industrial IoT based on multi-CNN fusion", Measurement, vol. 154, Mar. 2020.

[39] C. L. Yin, Y. F. Zhu, J. L. Fei and X. Z. He, "A deep learning approach for intrusion detection using recurrent neural networks", IEEE Access, vol. 5, pp. 21954-21961, 2017.

[40] Filiz Türkoğlu. "Author Attribution of Turkish Texts by Feature Mining" , Lecture Notes in Computer Science, 2007

[41] Md. Abdul Wahab. "Strain effect on single and double walled carbon nanotubes" , TENCON 2009 - 2009 IEEE Region 10 Conference, 2009

[42] Chidchanok Lursinsap. "Decision tree induction based on minority entropy for the class imbalance problem" , Pattern Analysis and Applications, 2016

[43] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In British Machine Vision Conference, 2014.

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei- Fei. Imagenet: A large-scale hierarchical image database. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 248255, 2009

[45] Md Zahangir Alom, VenkataRamesh Bontupalli, and Tarek M. Taha. ”Intrusion detection using Deep Belief Networks.” Aerospace and Electronics Conference (NAECON), 2015 National. IEEE, 2015