Machine Learning Engineer Nanodegree

Capstone Project

Udacity
May 15st, 2019

I. Definition

Project Overview

KDD Cup 1999 Data is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining.
The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between ``bad'' connections, called intrusions or attacks, and ``good'' normal connections.
This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.
Here is a paper made an Intensive Preprocessing of KDD Cup 99 for Network Intrusion Classification Using Machine Learning Techniques.

Problem Statement

The problem is to build software to detect network intrusions protects a computer network from unauthorized users, including perhaps insiders.

The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between ``bad'' connections, called intrusions or attacks, and ``good'' normal connections.

It’s important to note that our problem is basically a research problem and it’s about trying to use new technique and algorithms combinations to get competitor results than to find practically solution of the problem.

Our focus in this problem is to classify bad and good connections (binary classification) and not to find the type of each bad connection.
The solution approach is divided into 7 main steps:

1) Data exploration and processing.
2) Build a Naïve Model to get minimum Performance Threshold
3) Build Benchmark model and Tune parameters with a grid search.
4) Build 3 classifiers and compare (F-score, accuracy and training time) of each with visualization then choose the best one to be the solution model.
5) Build CNN classification model to use it as a feature extraction layer
6) Train chosen model (KNN) on the extracted features from CNN
7) Build confusion Matrix and ROC curve to visualize solution model performance.

Metrics

Our Data set is unbalanced as the number of each class is:

- Normal connections=157871, (19.6%).
- Intrusion connection= 647179, (80.4%).

So, I will use F-beta-score as the basic metric to measure the performance of the model.
We also will use accuracy to measure overall performance, but It will not be enough without F-score as it doesn't care about False positive or False Negative.
Accuracy= (True positive + True negative) / dataset size
F-beta= (1+B^2) * Precision * Recall / ((B^2) * Precision) +Recall

We prefer to use beta=1 as we want equal weights of precision and recall this is because: to not miss classify normal connections as Intrusion connection is in the same importance as to not miss classify Intrusion connection as a normal connection.

We used Roc curve and confusion matrix to visualize performance which is suitable for unbalanced data.

II. Analysis

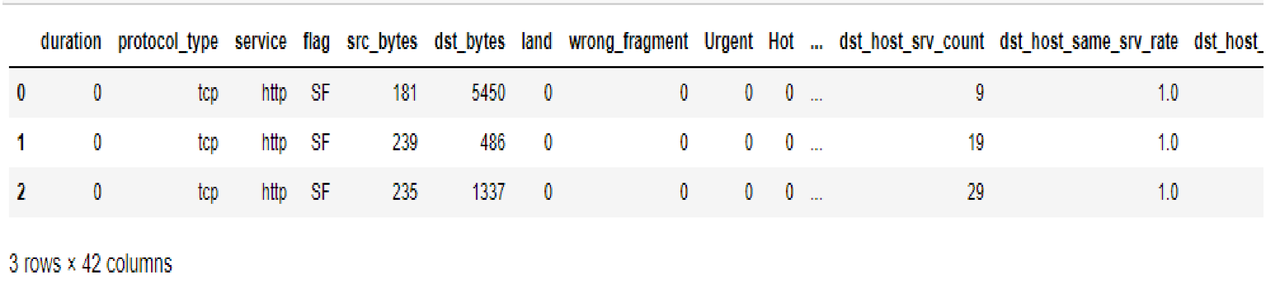
Data Exploration

Data will be used is (NSL-KDD) dataset set it's a data set suggested solving some of the inherent problems of the KDD'99 data set, The final files used after downloading and extracting data from source link are found here.
It's (805050 rows of TCP connections) with 42 feature describing connection details.
Dataset consists of 2 CSV files (kddtrain.csv, kddtest.csv).
Our used data solve some problems in KDD'99 original data as said in the source link and some important enhancements are:

1-It does not include redundant records in the train set.

2-There are no duplicate records in the proposed test sets.

- *Features Information*:
  - Here is training data sample:



	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_
0	0	tcp	http	SF	181	5450	0	0	0	0	...	9	1.0	
1	0	tcp	http	SF	239	486	0	0	0	0	...	19	1.0	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	29	1.0	

3 rows × 42 columns

  - There are 42 features their types are: float (15), int (23), object (4).
    - Object values will be encoded (‘class’, ‵protocol_type’, ‘service’, ‘flag’).
  - From train data describe there are several features that have high standard deviation which indicates that the data points are spread out over a wide range of values and needs scaling like features:
    - ‘src_bytes’ std=> 988,218, ‘dst_bytes’ std=> 33,040
  - Training data has two columns that their values are unique and equal to zero.
    - Columns are ['num_outbound_cmds', 'is_host_login'].
  - Testing data has one column that all its values are unique and equal to zero ['num_outbound_cmds'] and this column also unique in the training set which means it’s can be removed.
  - Testing data it includes specific attack types not in the training data as described in KDD’99 data set source, and this will badly affect our result so we will need to recombine and shuffle dataset to make training data generalized.
- *Label Information*:

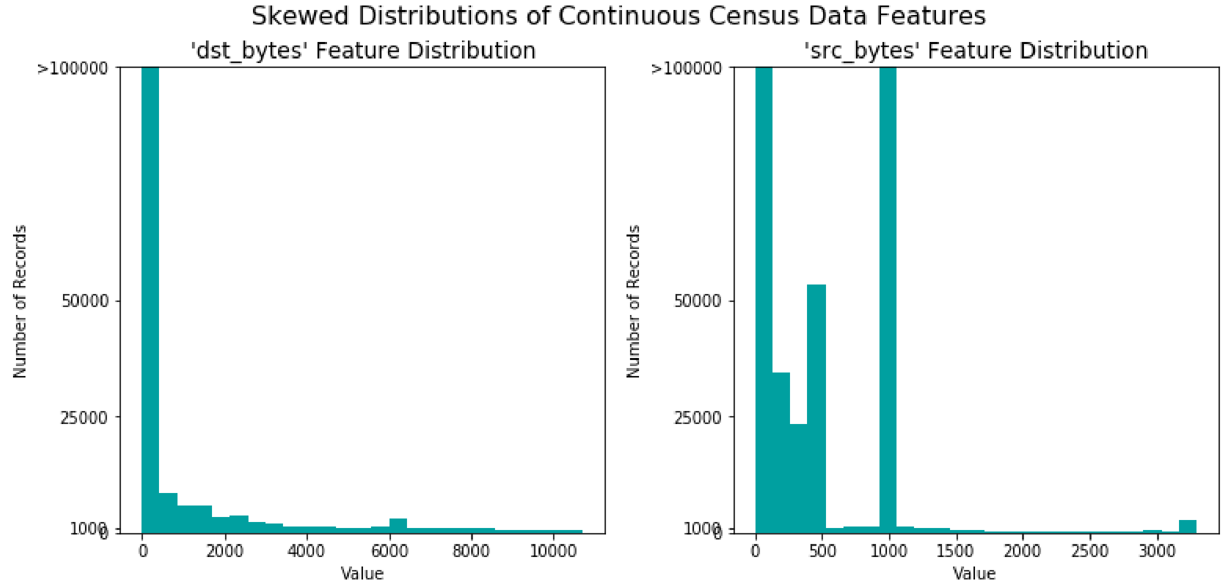
Each connection is labeled as either normal or as an attack, with exactly one specific attack type. The Label consists of 5 unique classes (Normal, DOS, PROBE, R2L, U2R)

Intrusion labels will be reassigned with the same value to have only two labels (normal and intrusion) to fit our problem scope. The number of each class in the data set is:

Class	Train	Test
Dos	391458	229853
Normal	97278	60593
PROBE	4107	4166
R2l	1126	16347
U2R	52	70

Exploratory Visualization

Here is a visualization of two features built with function implemented in visuals file Distribution values which indicates a high range of values that need to be rescaled:



It's clear that data distributed in high range and the number if distribution records not balanced along values, it's very high in small range (low values) and low in high values.

Algorithms and Techniques

Algorithms will be used divided into three sections:

1) Benchmark model
2) Solution models
3) Feature Extraction model

-Benchmark Model:

Ensemble Methods (Gradient Boosting):

- It will be used as the Benchmark model
- It's good for this problem as It's suitable for large data and Good at reducing variance and provide higher stability, can optimize on different loss functions and provides several hyperparameter tuning options that make the function

fit very flexible and get high Threshold score which will be a good challenge when we will try to exceed it.

- It’s sensitive to noisy data and outliers, which is Not the case in our data as our data precleared.
- We will use Grid Search to tune its parameters.

-Solution Models:

Here we will compare and choose from these Algorithms with keeping their default parameters and choose the best one then tunes parameters of chosen one and increase Its performance with a feature extraction model.

Naïve Bayes (GaussianNB)

- It's very fast, perform well for large data, no need for tuning parameters, rarely overfit and insensitive to irrelevant features.
- If features are dependent.
- It's good for this problem as data is large and may have some irrelevant features which make it a good choice.

Support Vector Machines (SVM):

- Work well in a complicated domain can separate data efficiently with kernel trick.
- Tends to overfit, have many parameters to tune.
- It's good for this problem as we need high F1-score and SVM will help as it has several hyperparameters we can tune with grid search which gives me the ability to separate data in the way that maximize the F1-score.

K nearest neighbors Classifier (KNN):

- It’s good at recognizing the hidden pattern in complicated data like ours.
- It is more widely used in classification problems
- Easy to interpret output, low Calculation time and high Predictive Power
- No need for tuning parameters only K will set to 2

-Feature extraction Model:

Convolutional neural network (CNN):

- This will use this model to extract features from data set to use them with the chosen solution model

- It can recognize complex patterns in the data set which will help to overcome the high threshold score of the benchmark model
- CNN Model Architecture:
    - Layer1: Convolution1D (57)
    - Layer2: MaxPooling1D (2)
    - Layer3: Flatten
    - Layer4: Dense (128)
    - Layer5: Dense (2)

Benchmark

The Benchmark model will be Gradient Boosting Classifier.
Our Data set is widely used in the research area, and there is no specific solution to the problem so the challenge is trying to increase performance as much as we can, so we need a powerful benchmark model to compare our results with trying to challenge ourselves to beat it and make a strong argument that you've adequately solved the problem.

-Gradient Boosting Classifier is a powerful classifier which is a big deal for our case.
-We tuned its parameters with grid search by searching between:
Estimators (10,100,300).
learning rate (0.01,0.1,1).
-Then calculate performance Threshold of the best estimator.

-Performance Threshold we get is:

- F1-score= 0.9747
- Accuracy-score= 0.9901

III. Methodology

Data Preprocessing

In preprocessing We Start with preprocessing each of our data files (training and testing files), but after preprocessing we found In building models section that training data from training data file we've processed wasn't generalized the whole data set well, and that causes limitation of performance in testing data results in about 92% of F1-score and accuracy .

So, we've combined the two files and shuffle and reprocess data with the same steps and these steps are:

1. Drop the training label from training and testing sets
2. Assign dropped column to y_train and y_test
3. One hot encoding for each object class in training and testing data which clarified in the exploration section
4. Scaling training and testing data using MinMaxScaler which scale data to be from 0 to 1 to treat the distribution of data in a high range of values, and we pick MinMaxScaler which provide only positive values scaling as we want to use data with naïve Bayes Algorithm and don't need negative values.
5. Visualize 2 columns of training data after scaling.
6. Change scaled data (X_train/X_test) from (2D array) to (3D array) to be able to be used as an input of CNN
7. Change y_train test from array to data frame then to a matrix to be used with CNN.

Then we recombine train and test data to shuffle them using train test split with these steps:

1. Combine train and test files to one file and save it in a result file.
2. Split result file to features and labels.
3. Reprocess data with previous steps.
4. Shuffle and split features and labels using the train test split with test size 0.2

Implementation

In this section, we trained a Benchmark model then train several solutions classifiers, compare between them to choose the best one to be our solution model. Steps are:

1. Build a Benchmark model using Gradient Boosting Classifier

    - Define classifier and train it with train data coming from train test split
    - Use Grid search to tune parameters from
        - Number of estimators: [10,100,300]
        - Learning rate: [0.01,0.1,1]
    - Pick the best estimator which is: 0.1 & 300
    - Calculate Accuracy and F1-score and we get:
        - F1-score= 0.9747
        - Accuracy-score= 0.9901

2. Build a naive model to use it as a minimum Threshold of any model we going to build, which predicts that all connections are intrusions and then calculate its accuracy and F1-score and considering intrusions as the positive label.

    - Calculate Intrusion percent
    - Calculate accuracy = intrusion percent/100
    - Calculate precision and recall from These equations

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

    - Recall =1 --> as False positive =0
    - precision = intrusion percent/100
    - The calculate F1-score from equation in metric section above.
      We Get:
        - Accuracy = 0.8043
        - F1score = 0.8915

3. Build a train-test pipeline function to use it with several classifiers to be able to train classifiers with input samples size and calculate several performance metrics.

    - The Input of the pipeline function is (learner, sample size, x train/ test, y train, test).

    - Function return: (predicted time, accuracy train, accuracy test, F-score train, F-score test).

4. Define 3 classifiers and set simples sizes (1%,10%,100%) then use pipeline function to train them.
    - Classifier A: GaussianNB
    - Classifier B: K-nearest-neighbors (KNN)
    - Classifier C: Support Vector Machine (SVC)
5. Build a visualization file having a function to visualize model performance output coming from train test pipeline function.

6. Train The 3 Defined classifiers with the pipeline function then visualize Their output Then pick best one from visualization results.

- KNeighborsClassifier is the best one but still less than the benchmark and need more improvement:
    - F1score = 0.9754
    - Accuracy = 0.9902

7. Define CNN Model to be used as a Feature Extraction Layer to extract features from the train and test data then pass them to The KNN model to improve results
    - Define model architecture
        - Layer0: Convolution1D (57)
        - Layer1: MaxPooling1D (2)
        - Layer2: Flatten
        - Layer3: Dense (128)
        - Layer4: Dropout (0.4)
        - Layer5: Dense (2)
    - Compile model:
        - loss= Categorical Cross entropy
        - optimizer=Adadelta
        - metrics=accuracy
    - Feature Extraction from Layer 0&3
    - Classification Layer 4&5
    - Convert y_train and y_test from 3d matrix to np array using the Argmax function to use them with KNN
8. Define and train KNN classifier on extracted features from CNN
    - Define KNN and Train the get accuracy:
        - F1-Score: 0.9760
        - Accuracy: 0.9904

Refinement

As mentioned, we 've started our solution with picking KNN Model and it gets initial results of:

- F1-score = 0.9754
- Accuracy = 0.9902

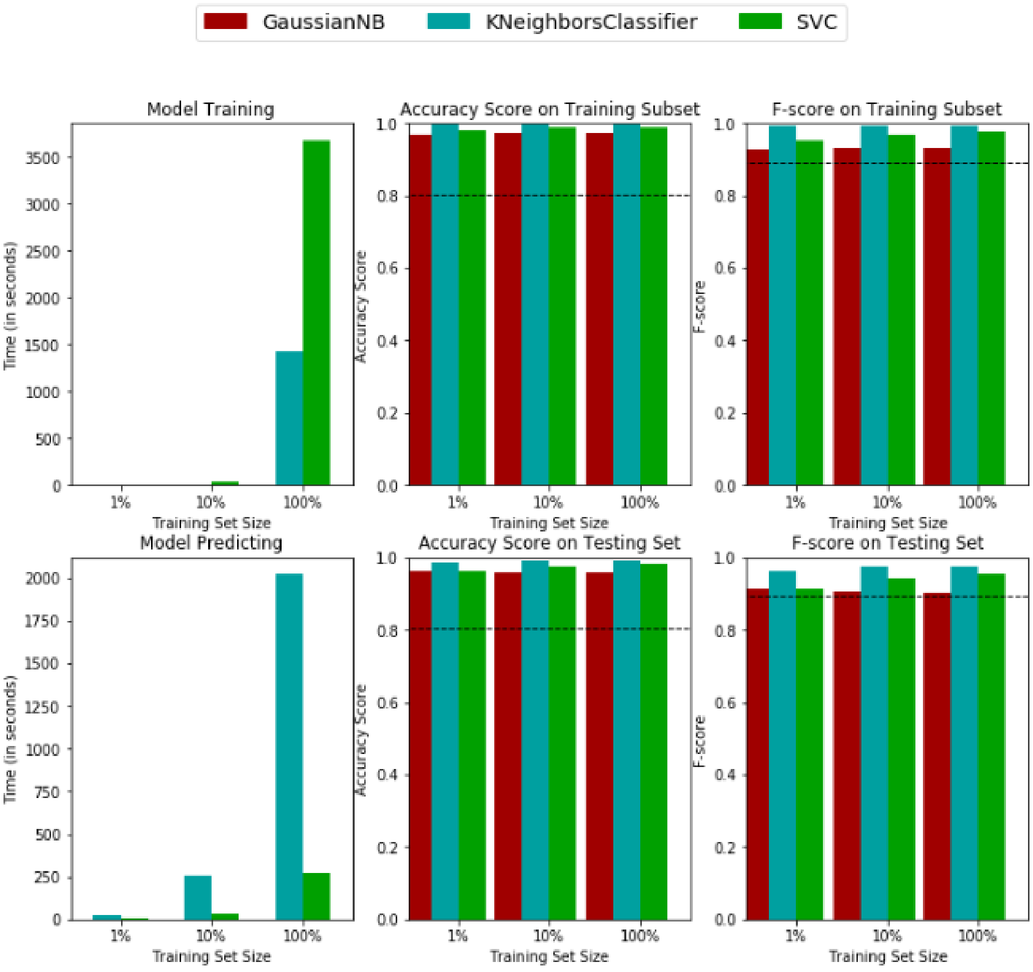
Which is close to Benchmark model but still need more improvement
We used CNN for feature extraction to extract important features from CNN layers and use these data to train on with KNN and the results are:

- F1-Score: 0.9760
- Accuracy: 0.9904

IV. Results

Model Evaluation and Validation

Choosing KNN Classifier was depending on a comparison between it and SVM, GaussianNB in performance and time to train as the visualization show:



KNN beats the two classifiers in performance, it gives the best F-1 and accuracy score in training and testing data.

But It takes too large time about (33 min) on predicting.

The model trained in various input samples and it gave almost the same performance on testing for both accuracy and f-score, so it’s trusted and generalized well.

Justification

Benchmark model established earlier gives results of:

- F1-score = 0.9747 ---> 97.47%
- Accuracy = 0.9901 ---> 99.01%

Our Solution model gives result of:

- F1-score = 0.9760 ---> 97.60%
- Accuracy = 0.9904 ---> 99.04%

This Indicates 0.13% increasing in F1-score.

And 0.03% increasing in Accuracy.

And This is a Justification for beating the Benchmark model.

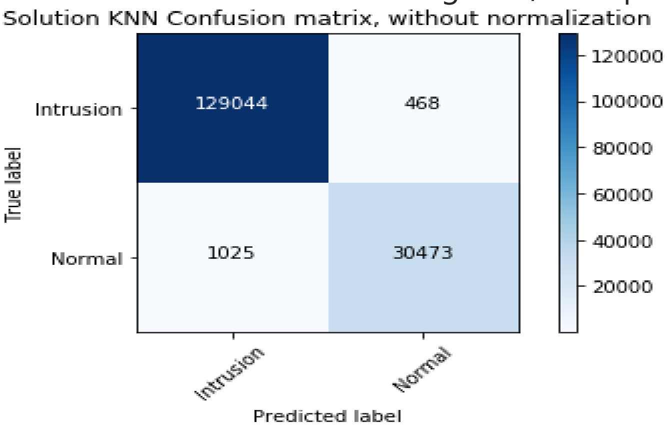
So now our model able to classify between good and Bad connections with High performance.

V. Conclusion

Free-Form Visualization

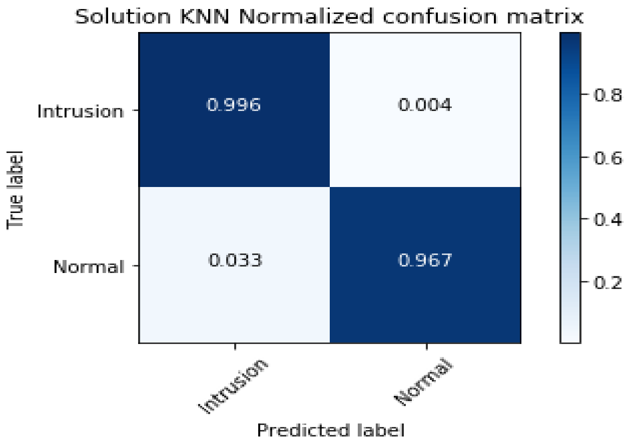
We have visualized the 3 models performance comparison in the model evaluation and validation section.

Now after choosing KNN as our classifier and we used CNN to enhance its result we got This Confusion matrix describing True/False positives and negatives of the final model:



Solution KNN Confusion matrix, without normalization

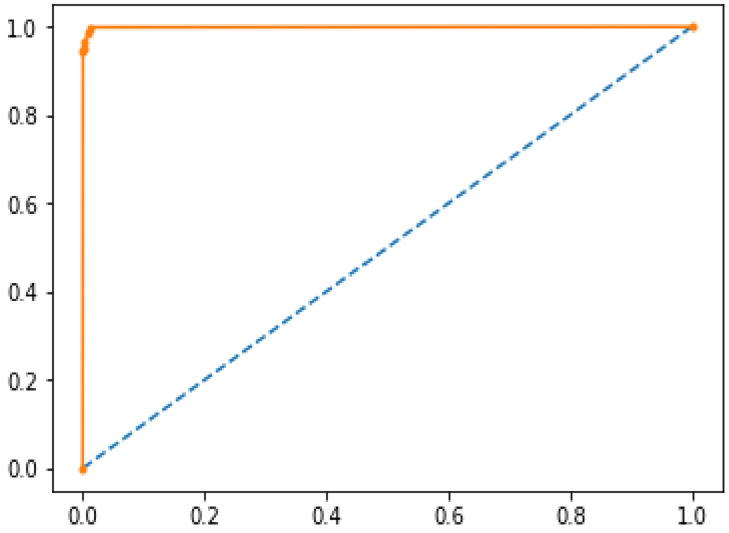
Here is a normalized Confusion matrix



From confusion matrix we notice that the percentage of False negative which indicates percentage of intrusions predicted as normal connections is very low = 0.004 which is good and also the false positives that indicates percentage of normal connections predicted as intrusion is low = 0.033

- Another visualization describes our model performance is the ROC-AUC curve
- We know That perfect split result square area when calculate True positives and false positives rates so the are under the curve (AUC) for the perfect split is equal to 1
- We Get AUC= 0.998751 which is very close to 1
- Doted line indicates random split and AUC of random split = 0.5

```
AUC: 0.998751
```



Reflection

To conclude, here is the Summarized steps of what we do to solve our problem:

1) First, we examine and explore data and we found that there is a need of some processing including scale, reshape, split, and shuffle data to prepare it to be used as input of our models
2) Then we build naïve classifier and a gradient boosting benchmark model and tune Its parameters with grid search to be a high threshold and compare our results with it.
3) We trained several classifiers types and compare their performance to choose best one which was KNN.
4) We start enhancement of KNN model trying to beat benchmark and we use CNN model to do this.
5) We build CNN model to work as feature extraction layer to pass its output features to the KNN model
6) Now we get our results and start visualizing results with ROC curve and Confusion matrix
7) Finally, we compare our results with Benchmark model and beat It

There is an interesting aspect of the project if we look to the problem as multiclass classification and try not only classify between good and bad connections but also classify the type of the intrusion which will be more challenging in both training and evaluating model performance as types of intrusions is completely unbalanced which will be.

Our problem basically a research problem and data available in this domain is rare which make it difficult to experiment our model with new data that model never seen.

Another Difficult aspect of the projecting is dealing with testing data without shuffle which it with testing data because it has types of connections not in training data which will limit our scores, but this makes the task more realistic.

The final model and solution fit our expectations for the problem as a research problem,

but I think it should not be used in a general setting to solve this types of problems because connections protocols and its details changes and upgraded permanently and may depends on routers or devices used also types of intrusions increases and changes permanently which make our model not Adequate to do this job.

Improvement

Improvement I think it can be made and enhance our solution results are:

- Improvement in CNN models this including trying different types of optimizers like adadelta or rmsprob I think may increase model performance
- Adding more nodes in input layer may lead to better feature selection but I wasn't able to experiment this due to GPU memory limitations as I can't assign a bath size with the K-function Keras backend.
- Using K-fold cross validation to get the best use of testing data I think will increase model performance.

I wasn't known how to implement feature selection from CNN layers, and I research for this to implement.

I think there is may be a better way of implementing CNN feature extraction that can get better performance.

I think There is a better solution exist specially if we try above improvements and trying to understand data well and types of connections that give wrong prediction definitely will help to pick the best appropriate algorithm and data processing.

References links:

https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

http://kdd.ics.uci.edu/databases/kddcup99/task.html

https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae