**Intrusion Detection System**

An Intrusion detection system model was built based on the data supplied to us. The following are the steps that were carried out respectively to do so: -

**Loading and Summarizing the dataset**

Initially, the dataset is loaded and evaluated by having a look at the columns, type of values the columns hold and also having a look at the number of unique values each column holds to get an initial idea on the data.

From the initial look, we can observe that there are different features that hold discrete values and some that hold continuous values. Also, a few of them are categorical variables which would need to be dealt with accordingly further down.

The final column/feature is the output variable which depicts if a particular row of data leads to be detected as an anomaly or not (normal).

**Data Pre-processing, Cleaning & Transformation**

Before building the model, it is important that we pre-process, clean and transform the data to a usable one such that we avoid any kind of redundancies, noise, missing data etc. that might actually disrupt us from building a good model.

The following were done by us: -

1. String to categorical variables - Converting features/columns with strings to categorial variables/numerical values – We mapped 3 features namely, protocol type, service and flag to numbers so that we could build a model accordingly.

This is an important step. We also mapped the output variable/feature, class to 0 and 1 where 1 represents an anomaly and 0 represents normal behaviour.

1. Remove duplicate rows – Another important step to be performed is the removal of duplicate rows since duplicate data might mess up the model that we are going to build. So, we remove any duplicate rows present in the dataset.
2. Remove missing data – When you are provided with a dataset, you might have missing or null values present in certain columns or rows. There are a few ways to deal with it based on the type of data.

If the number of missing or NaN values present in the dataset is considerably small compared to the size of the whole dataset, it wouldn’t hurt to remove such data. Hence, we remove the rows where any of the column value is missing/NaN.

**Feature Selection**

Once we are done with pre-processing and cleaning our data, the next step is to remove unwanted features or columns from our dataset. The lesser the features, the better or easier the classification and the performance also improves.

In addition, there is no use of features that have a single unique value throughout or almost no different unique values spread across the whole dataset. Also, we could remove highly correlated features so as to avoid redundancy.

We will perform these two steps in our solution as well: -

1. Removing quasi-constant features – Quasi-constant here means that a feature or column mostly has a single unique value spread over. Such features won’t add any value to our classification or model since its more of a constant. To remove such features, we make use of variance. If variance is 0, it means that a column or feature has only one unique value throughout.

In our solution, we’ll remove columns/features that has a variance of 0.01 or less so that we remove columns that have mostly a constant value. For doing so, we make use of Variance Threshold from the scikit-learn package.

In our case, we had 40 columns/features initially. After removing quasi-constant features, we were left with 32 features. 8 features were found to be quasi-constant. They are – *land, urgent, root\_shell, num\_shells, num\_access\_files, num\_outbound\_cmds, is\_host\_login, dst\_host\_srv\_diff\_host\_rate*

1. Remove highly correlated/redundant features – The second step we’ll perform in feature selection is finding correlated features and removing them from our final dataset (with which we’ll build our model). For this, we will compute the correlation matrix with the dataset. And if the correlation value is greater than a certain threshold, we will remove that feature.

In our scenario, we’ll take the threshold as 0.9 and remove those features which are highly correlated. There were 7 features that were removed, namely – *srv\_serror\_rate, srv\_rerror\_rate, num\_root, dst\_host\_srv\_serror\_rate, dst\_host\_srv\_rerror\_rate, dst\_host\_serror\_rate, dst\_host\_same\_srv\_rate*

**Modeling**

The final step is to build our model as the data is full prepared and ready. We’ll one again leverage scikit-learn to split our data into train and test sets.

We split our data into train and test in a 4:1 ratio i.e. 67% train data and 33% test data. We could opt 80-20 or 50-50 splits as well (train-test)

Post that, we use the Decision Tree Classifier with entropy as our criterion and a max depth of 4. And then we train the model with our training just taking less than 1 second. Post-training, we do the prediction with the test data and that also takes less than a second.

We can observe that the train and test accuracies are 91.27% and 90.66% respectively, which is pretty good.

To see if the accuracy can be improved, we try to train and test using the XG Boost algorithm as well leveraging the XGBClassifier. After train and test, we can observe that the accuracy improved by quite a good margin.

Training accuracy jumped to 98.72% and the test accuracy was 98.51%. The XG boost algorithm basically makes use of gradient boosting decision tree algorithm.