DATA MINING AND DATA WAREHOUSING PROJECT

DATA DRIVEN INTRUSION DETECTION SYSTEM BASED ON CLASSIFICATION ALGORITHMS

SUBMITTED BY:

AAYUSH KALIA (18103004)

ABHISHEK SINGH (18103007)

AMAN KUMAR RAHI (18103015)

ARCHIT JINDAL (18103019)

AVIRAL GUPTA (18103025)

SUBMITTED TO:

DR. NONITA SHARMA

ASSISTANT PROFESSOR

CSE DEPARTMENT

NIT JALANDHAR

1. Title:

Data Driven Intrusion Detection System based on Classification Algorithms

2. Abstract:

Intrusion is an act of any unauthorized activity on a network. Network intrusion can lead to stealing of valuable and information along with hurting privacy of users on the network. Due to all this, intrusion detection is an important aspect needed to provide security to internet users and organisations. To implement the required model, we need a dataset which has a large amount of accurate and reliable data. In this assignment we have used NSL-KDD Dataset. KDD'99 had a problem of redundancy, which sometimes lead our model to be biased towards a specific property. We have used NSL-KDD dataset to compare the various classification algorithms and find the one with highest accuracy. The classification algorithms compared are :

- Random Forest
- SVM
- K Neighbours

All the work has been done on Google Collab using Python as the programming language.

3. Keywords:

- IntrusionDetection System
- Protocol
- Random Forest
- NSL-KDD dataset
- SVM
- K Neighbours
- Anomaly

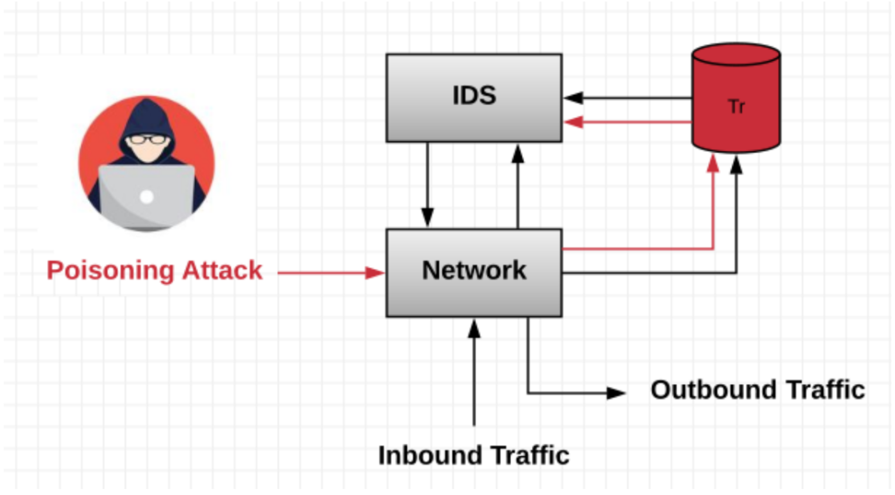
4. Introduction

4.1 Real Life Application of IDS

The main aim of IDS is to raise an alert when it discovers any malicious activity, which has been classified as abnormal behaviour.

So, here in this project, we'll be building a data-driven Intrusion Detection System (IDS), and analyse it based on different classification algorithms like Random Forest, KNN, SVM.

to process the voluminous database, various machine learning techniques can be used



4.2 Dataset Analysis

KDD’99 Dataset – This dataset contains a large amount of network traffic records.

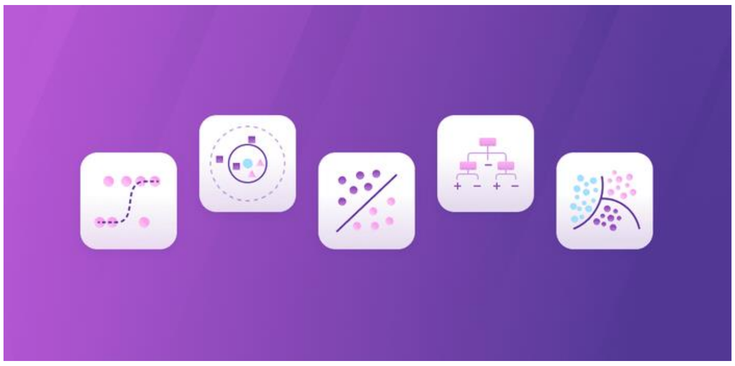
When was KDD’99 Dataset collected?

In the year 1999 in a competition.

NSL-KDD Dataset – Successor and better version of KDD-99Dataset.

Classification (What is classification, its advantages, its scope, limitations, and methods)

Classification is a supervised learning method, which categorizes a provided dataset into classes.



Various Classification Methods:

1. Logistic Regression: Although named regression, it is a classification algo.
   Advantages: It gives us information according to the probabilistic approach.
   Disadvantages: It assumes linearity between dependent and independent variables

2. Naïve Bayes: It is derived from Bayes’ theorem. It assumes that features are independent.
   Advantages: It is efficient.

Disadvantages: It assumes features are independent.

3. K-Nearest Neighbours: It is a lazy learning method. Returns the majority classification in k closest points.
   Advantages: Easy and quick.
   Disadvantages: We have to decide the value of k.

4. Decision Tree: It is a greedy algorithm which selects best available feature.
   Advantages : There is no need of feature scaling.
   Disadvantages: Usually leads to overfitting

5. Random Forest: It makes many decision trees and select the answer with maximum frequency.
   Advantages: Less prone to overfitting, outputs the importance features.
   Disadvantages: Computations can become very complex

6. Support Vector Machine (SVM): Uses a plane to separate the features.

Advantages: No overfitting.

Disadvantages: Not a good option when many values are present.

4.3 Structure of project Report

The following table shows the structure of project report from 5th section onwards:

Section	**Name**	**Detail**
5	Classification Methods	Discussion about various classification algorithms
6	Experimentation	Complete demonstration of the Project
7	Results and Discussion	Comparison between intrusion detection models built on KNN, SVM and Random Forest.
8	Conclusion and Future Scope	Final project summary and future scope of improvement.

5. Methods

a) Decision Tree

A DT stands for a directed tree with roots.

Internal nodes in DTs are those with outgoing edges. The DT's other nodes are terminal nodes or leaves.

The attributes are classified by DTs using a series of hierarchical decisions. The split criteria is determined by judgments taken at internal nodes.

Each leaf in a DT is allocated to a class or chance.

Small changes in the training set cause various splits, which result in a different DT. As a result, for DTs, the error contribution due to variance is important. Ensemble learning, discussed in the next section, can help palliate the error due to variance.

How does a tree decide where to split ?

The dataset can be splitted using entropy and information gain.

1. **Information Gain**
   Entropy is a measure of how much difference there is in a set of data.

The weighted entropies of each branch are subtracted from the original entropy to quantify Information Gain for a break. By using these metrics to train a Decision Tree, the best split is determined by optimising Information Gain.

$$\boxed{Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)}$$

$$Gain(S, A) = Entropy(S) - \sum_{v \epsilon Values(A)} \frac{|S_v|}{|S|}.Entropy(S_v)$$

ii. Gini Index

• The GI is a metric that determines how often a randomly selected variable is incorrectly detected.

$$GiniIndex = 1 - \sum_j p_j^2$$

Advantages

- It is easy to grasp because it follows a constant method that somebody follows whereas creating any call-in real-life.
- It is terribly helpful for the resolution of decision-related issues.
- It helps to place confidence in all the attainable outcomes for a haul.
- There is less demand for knowledge cleansing compared to alternative algorithms.

Disadvantages

- The decision tree contains legion layers, which makes it advanced.
- It may have an associate overfitting issue, which might be resolved exploitation the Random Forest formula.
- For a lot of category labels, the process quality of the choice tree could increase.

b) Naïve Bayes algorithm

It is derived from Bayes' theorem. It assumes that features are independent.

Main goal of Bayesian classification is to determine the posterior probabilities, or the likelihood of a mark provided certain observed characteristics, P(L | features). We can represent Bayes' theorem as follows:

wP(M|f)=P(M)P(f|M) / P(features)

Where, (M | f) is posterior probability.

(M) is prior probability.

(f|M) is likelihood.

(f) is the priorprobability.

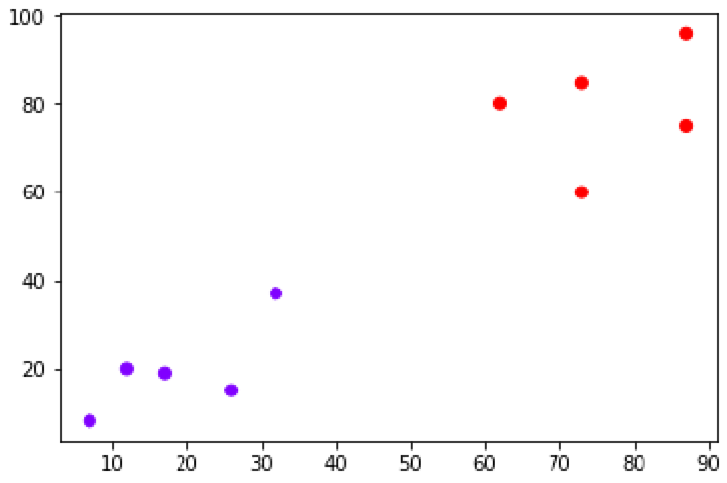
c) K Nearest Neighbors Algorithm

It is a lazy learning method. Returns the majority classification in k closest points.

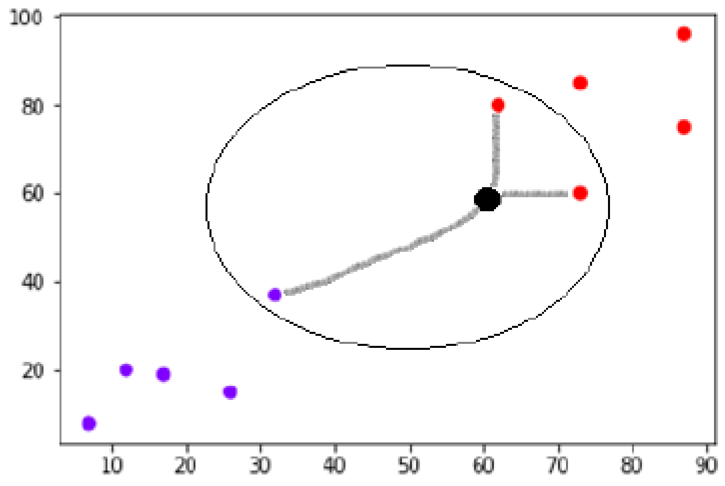
Working of KNN

The two diagrams below help in understanding knn.

Let's say a dataset that is plotted in the following way:



The newest black mark (at 60,60) must now be sorted into the either of 2 classes. We will assume k=3, . The following diagram depicts this -

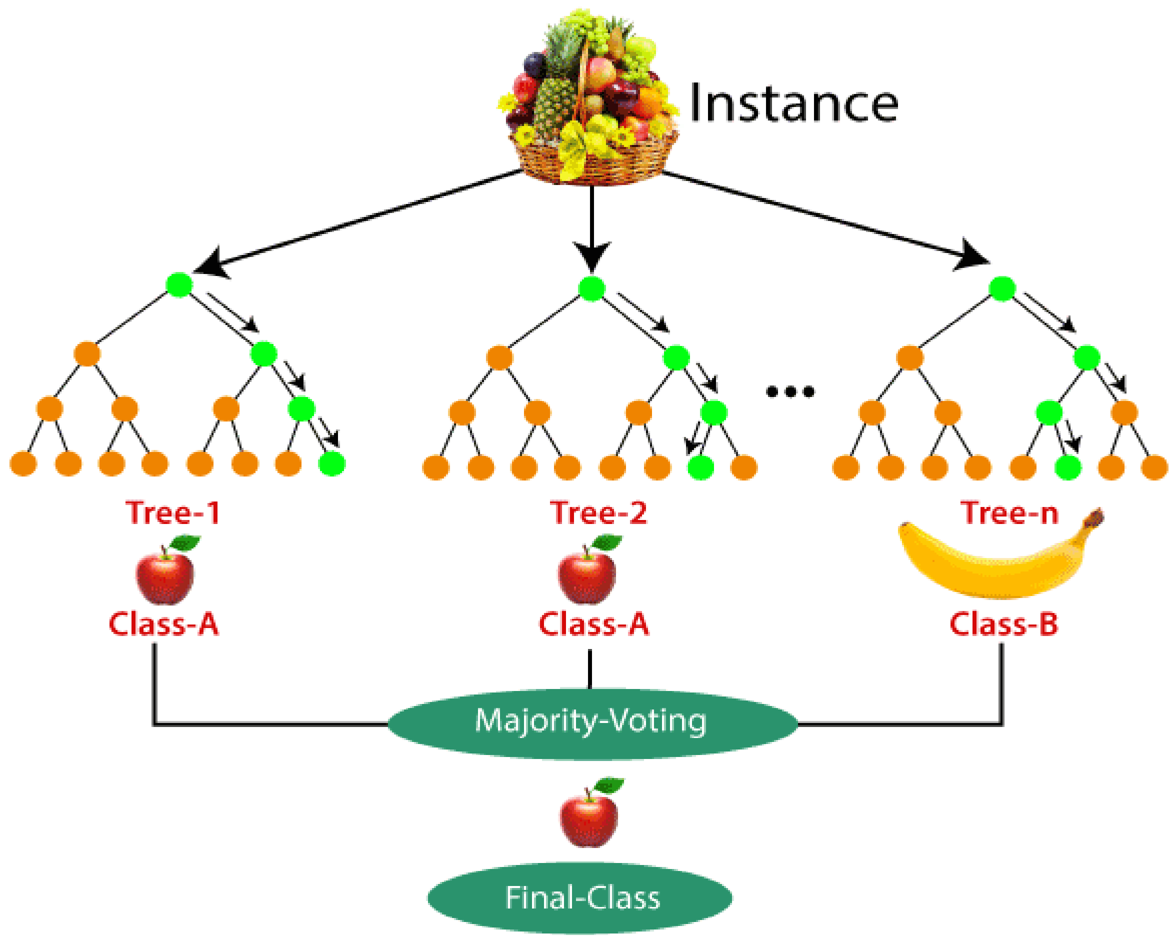


The newest point will be in red class.

d) Random Forest Algorithm

In this we combine various decision trees to get an answer.

Consider a scenario, where is a dataset containing several fruit images. As a result, the Random forest classifier is given this dataset. Each decision tree is assigned a subset of the dataset to work with. During the training process, each decision tree generates a prediction result, and when a new data point appears, the Random Forest classifier forecasts the final decision based on the majority of outcomes. Get the following illustration:



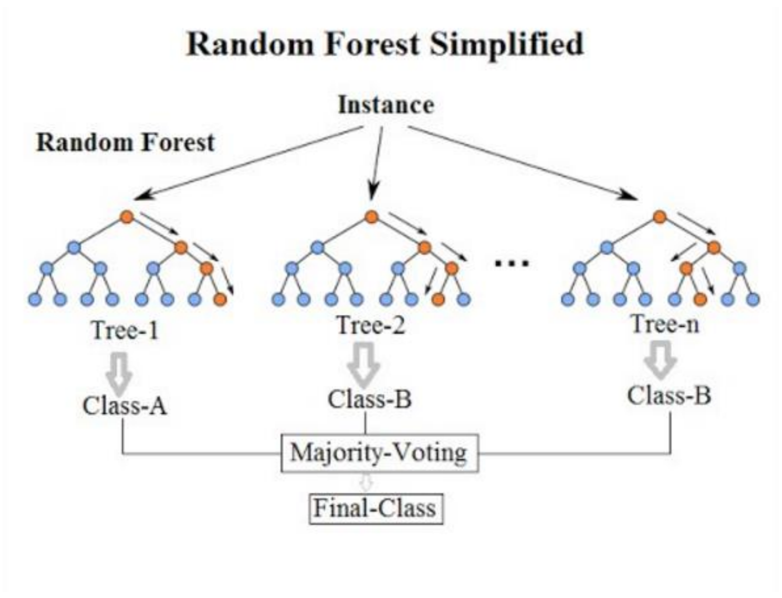
6. Experimentation:

6.1 Random Forest

We have various decision trees and they produce their outputs. Then, majority voting is done to find the final classification.

Prerequisite:

Individual tree predictions must have low correlations with one another.



Pros:

- Robust against outliers.
- On a huge dataset, it performs well.
- Better than other classification algorithms in terms of accuracy.

Cons:

- When dealing with categorical variables, random forests are considered to be biased.
- It is not suitable for linear methods with a lot of sparse features

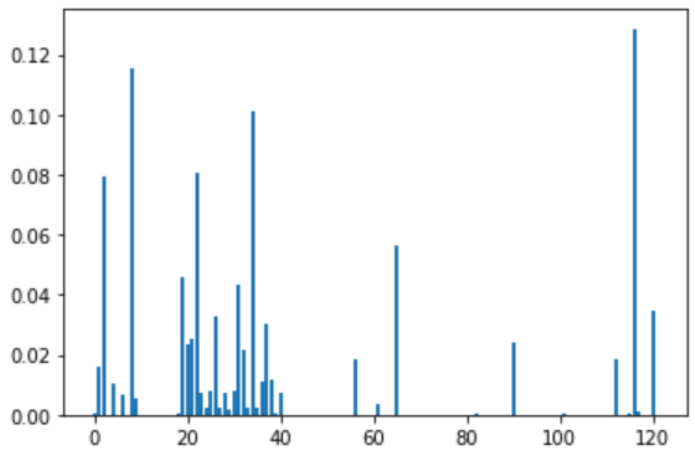
sklearn.ensemble.RandomForestClassifier

class `sklearn.ensemble.`**`RandomForestClassifier`**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

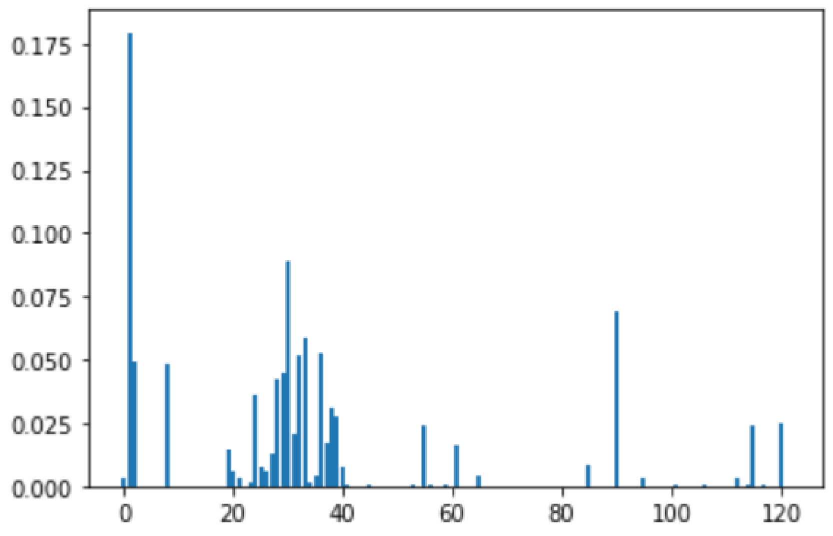
```python
from sklearn.ensemble import RandomForestClassifier

# Build the model
clf_DOS = RandomForestClassifier(n_estimators=10, n_jobs=2)
clf_Probe = RandomForestClassifier(n_estimators=10, n_jobs=2)
clf_R2L = RandomForestClassifier(n_estimators=10, n_jobs=2)
clf_U2R = RandomForestClassifier(n_estimators=10, n_jobs=2)

clf_DOS.fit(X_DOS_train, Y_DOS_train.astype(int))
clf_Probe.fit(X_Probe_train, Y_Probe_train.astype(int))
clf_R2L.fit(X_R2L_train, Y_R2L_train.astype(int))
clf_U2R.fit(X_U2R_train, Y_U2R_train.astype(int))
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=2,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

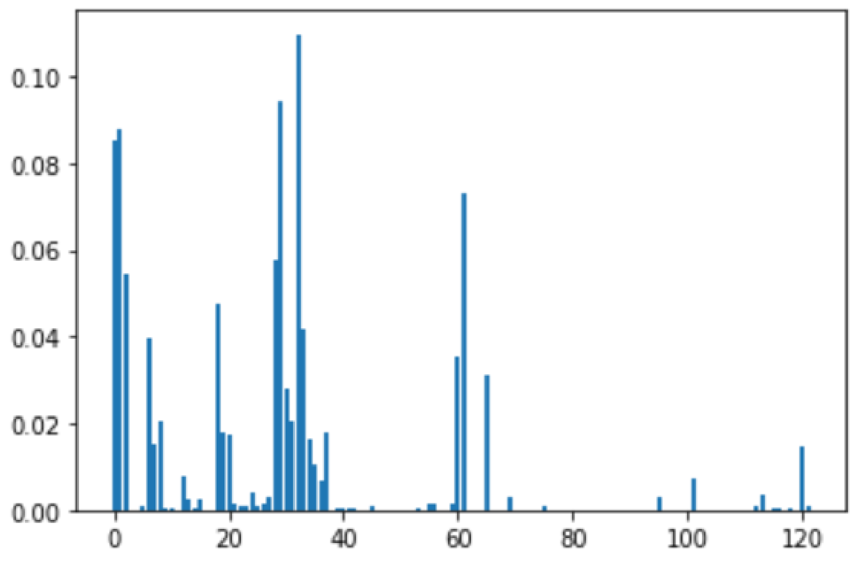
```python
importance_DOS = clf_DOS.feature_importances_
pyplot.bar([x for x in range(len(importance_DOS))], importance_DOS)
pyplot.show()
```



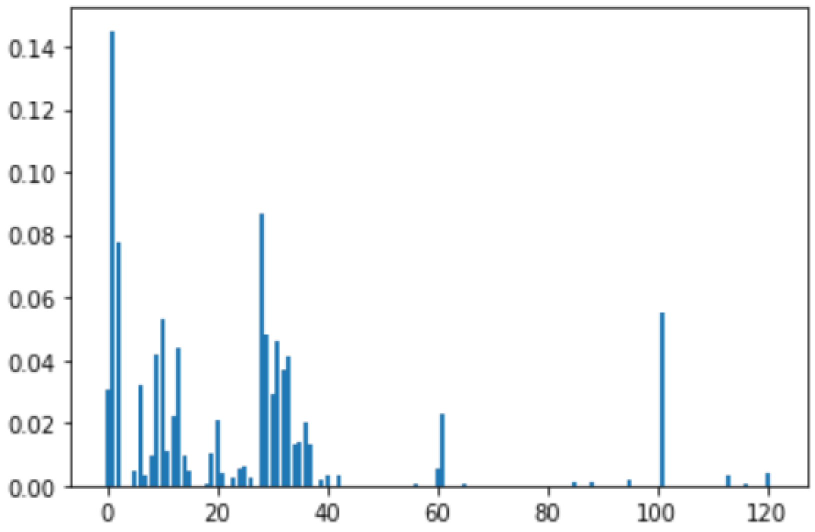
```
importance_Probe = clf_Probe.feature_importances_
pyplot.bar([x for x in range(len(importance_Probe))], importance_Probe)
pyplot.show()
```



```
importance_R2L = clf_R2L.feature_importances_
pyplot.bar([x for x in range(len(importance_R2L))], importance_R2L)
pyplot.show()
```



```
importance_U2R = clf_U2R.feature_importances_
pyplot.bar([x for x in range(len(importance_U2R))], importance_U2R)
pyplot.show()
```



Prediction and Evaluation

```
# DOS
# Apply the classifier to the test data
Y_DOS_pred = clf_DOS.predict(X_DOS_test)
pd.crosstab(Y_DOS_test, Y_DOS_pred, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

Predicted Attacks	0	1
Actual Attacks		
0	9682	29
1	6035	1425

```
# Probe
Y_Probe_pred = clf_Probe.predict(X_Probe_test)
pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

Predicted Attacks	0	2
Actual Attacks		
0	9453	258
2	978	1443

```
# R2L
Y_R2L_pred = clf_R2L.predict(X_R2L_test)
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

Predicted Attacks	0
Actual Attacks	
0	9711
3	2885

```
# U2R
Y_U2R_pred = clf_U2R.predict(X_U2R_test)
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

Predicted Attacks	0	4
Actual Attacks		
0	9711	0
4	65	2

Cross Validation: Accuracy, Precision, Recall, F-measure

```
# DOS
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99796 (+/- 0.00197)
Precision: 0.99906 (+/- 0.00172)
Recall: 0.99678 (+/- 0.00401)
F-measure: 0.99772 (+/- 0.00257)
```

```
Accuracy_RF_DOS = accuracy.mean()
```

```
# Probe
accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99662 (+/- 0.00238)
Precision: 0.99603 (+/- 0.00470)
Recall: 0.99370 (+/- 0.00516)
F-measure: 0.99508 (+/- 0.00491)
```

```
Accuracy_RF_Probe = accuracy.mean()
```

```
# U2R
accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99693 (+/- 0.00224)
Precision: 0.96088 (+/- 0.10669)
Recall: 0.84985 (+/- 0.13520)
F-measure: 0.92658 (+/- 0.11544)
```

```
[ ]  Accuracy_RF_U2R = accuracy.mean()
```

```
[ ]  # R2L
     accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
     print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
     print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
     Accuracy: 0.98015 (+/- 0.00598)
     Precision: 0.97534 (+/- 0.00980)
     Recall: 0.96846 (+/- 0.01388)
     F-measure: 0.97082 (+/- 0.00915)
```

```
Accuracy_RF_R2L = accuracy.mean()
```

6.2 KNeighbours

It is a lazy learning method. Returns the majority classification in k closest points.

Working

Distance is found from anyone of the common methods and class with maximum occurrences out of nearest is assigned to new point. The Euclidean distance method is a common one.

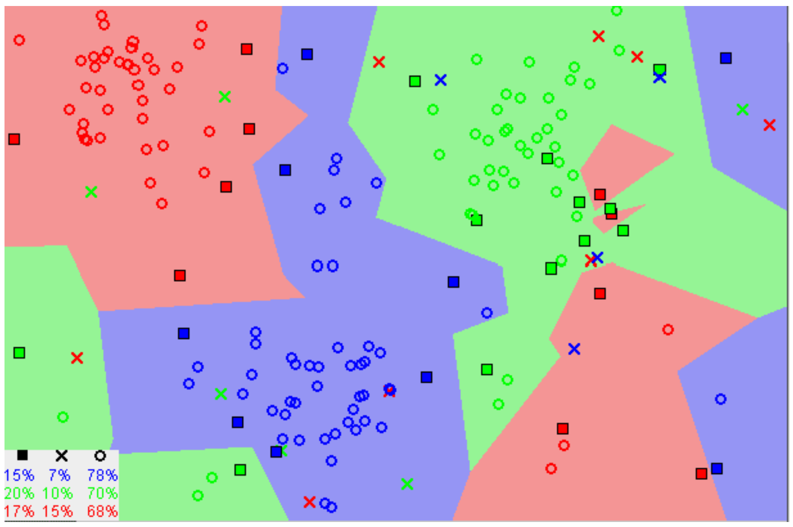
Manhattan, Minkowski, and Hamming distance methods are examples of other distance methods. The hamming distance must be used for categorical variables.

Pros of KNN

- With enough representative data, it is possible to perform well in practice.

Cons of KNN

- Data storage
- It's important to know that we have a useful distance feature.



```python
from sklearn.neighbors import KNeighborsClassifier

clf_KNN_DOS = KNeighborsClassifier()
clf_KNN_Probe = KNeighborsClassifier()
clf_KNN_R2L = KNeighborsClassifier()
clf_KNN_U2R = KNeighborsClassifier()

clf_KNN_DOS.fit(X_DOS_train, Y_DOS_train.astype(int))
clf_KNN_Probe.fit(X_Probe_train, Y_Probe_train.astype(int))
clf_KNN_R2L.fit(X_R2L_train, Y_R2L_train.astype(int))
clf_KNN_U2R.fit(X_U2R_train, Y_U2R_train.astype(int))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```python
# DOS
Y_DOS_pred_KNN = clf_KNN_DOS.predict(X_DOS_test)
pd.crosstab(Y_DOS_test, Y_DOS_pred_KNN, rownames=['Actual attacks'], colnames=['Predicted Attacks'])
```

```
Predicted Attacks     0     1
   Actual attacks
               0   9422   289
               1   1573  5887
```

```python
# Probe
Y_Probe_pred_KNN = clf_KNN_Probe.predict(X_Probe_test)
pd.crosstab(Y_Probe_test, Y_Probe_pred_KNN, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

```python
# R2L
Y_R2L_pred_KNN = clf_KNN_R2L.predict(X_R2L_test)
pd.crosstab(Y_R2L_test, Y_R2L_pred_KNN, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

```python
# U2R
Y_U2R_pred_KNN = clf_KNN_U2R.predict(X_U2R_test)
pd.crosstab(Y_U2R_test, Y_U2R_pred_KNN, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

Cross Validation: Accuracy, Precision, Recall, F-measure

```
# DOS
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_KNN_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99715 (+/- 0.00278)
Precision: 0.99678 (+/- 0.00383)
Recall: 0.99665 (+/- 0.00344)
F-measure: 0.99672 (+/- 0.00320)
```

```
Accuracy_KNN_DOS = accuracy.mean()
```

```
# Probe
accuracy = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99077 (+/- 0.00403)
Precision: 0.98606 (+/- 0.00675)
Recall: 0.98508 (+/- 0.01137)
F-measure: 0.98553 (+/- 0.00645)
```

```
Accuracy_KNN_Probe = accuracy.mean()
```

```
# R2L
accuracy = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

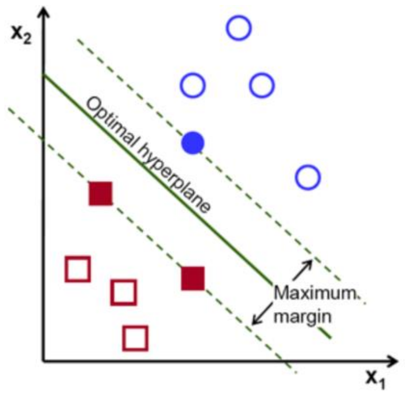
```
Accuracy: 0.96737 (+/- 0.00730)
Precision: 0.95311 (+/- 0.01274)
Recall: 0.95484 (+/- 0.01326)
F-measure: 0.95389 (+/- 0.01030)
```

6.3 Support Vector Machine (SVM)

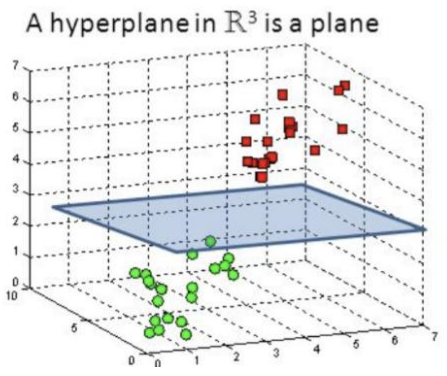
Using SVM (Support Vector Machine) classifier on the model-

A brief intro to SVM-

Support vector basically aims at finding a plane which is also known by the name hyperplane in N-dimensions which can distinctly classify the data points with great accuracy. For the separation of the points of dataset, we need to draw some boundaries and those boundaries are also know as hyperplanes. So basically, the main motive is to find such kind of plain that has the maximum margin. Below are the visualizations of the Hyperplanes in 2D and 3D.



(2D hyperplane)



(3D hyperplane)

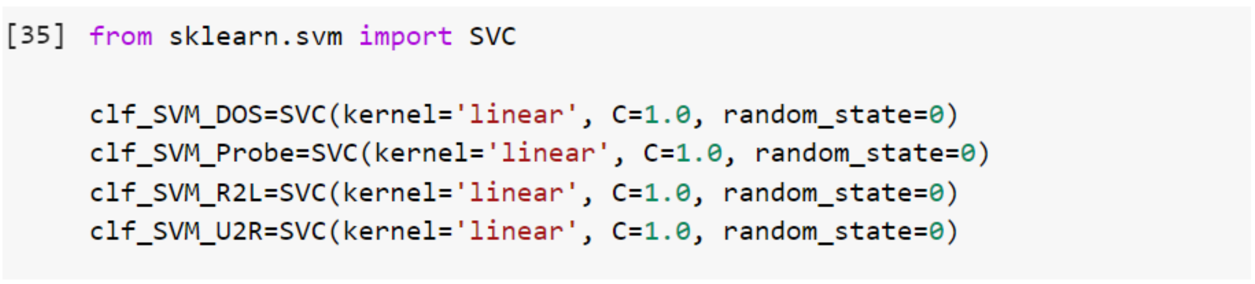
Now applying the SVM to our NSL-KDD training dataset-

For the implementation sake, we have used the sklearn library in order to perform the computations most optimally-

Now, next is step by step explanation regarding the implementation of SVC and fitting to the training-data.

Step by Step Explanation of implementation using SVC (Sklearn) -

1)

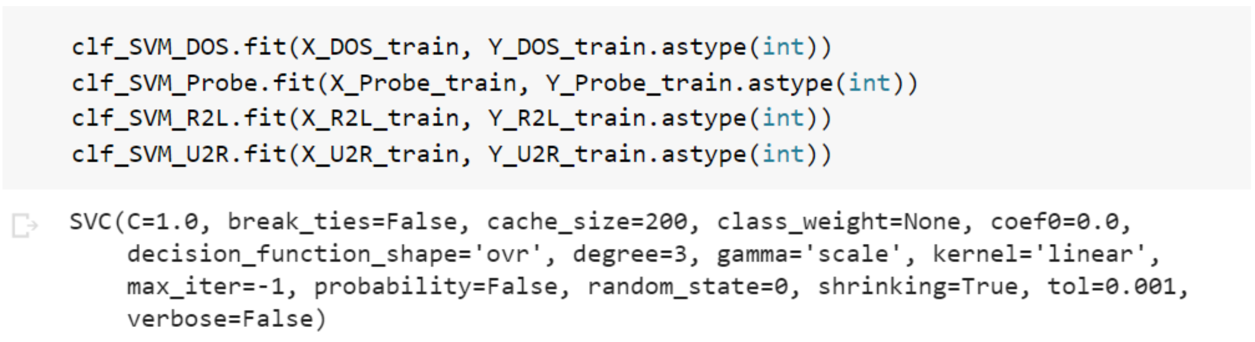


```
[35] from sklearn.svm import SVC

     clf_SVM_DOS=SVC(kernel='linear', C=1.0, random_state=0)
     clf_SVM_Probe=SVC(kernel='linear', C=1.0, random_state=0)
     clf_SVM_R2L=SVC(kernel='linear', C=1.0, random_state=0)
     clf_SVM_U2R=SVC(kernel='linear', C=1.0, random_state=0)
```

So, firstly SVC is imported from sklearn.svm, then for the different type attacks as listeb in the starting of the document (we have created four separate instances of the SVC).

Kernel- Kernel is basically a function that helps in more clear classification of the points of the dataset. So now how does a kernel works? It basically maps a lower dimension points to higher dimensions and helps in the better classification. SVM used in our classification model is basically the linear SVM which helps in construction or finding the equation of hyperplane to separate the datapoints so as to have a clear boundary of separation.

2)

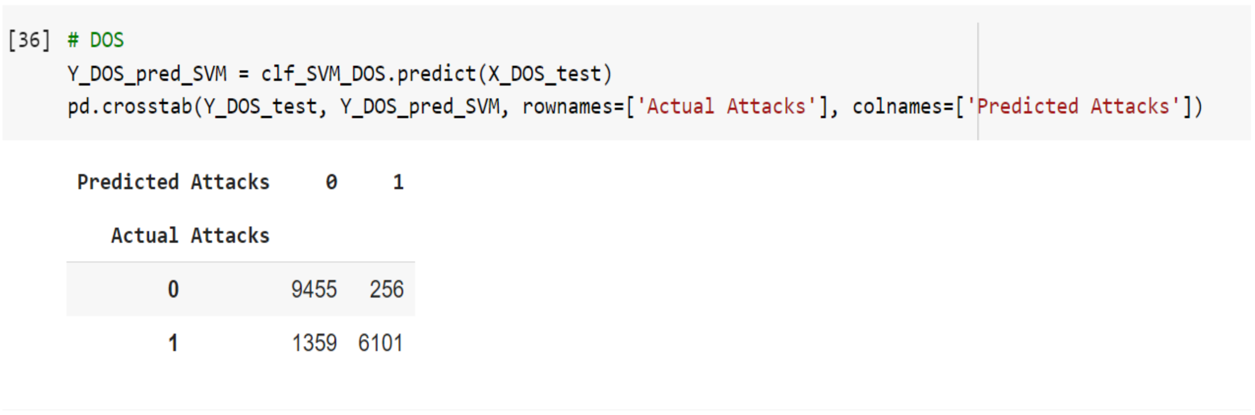


```
clf_SVM_DOS.fit(X_DOS_train, Y_DOS_train.astype(int))
clf_SVM_Probe.fit(X_Probe_train, Y_Probe_train.astype(int))
clf_SVM_R2L.fit(X_R2L_train, Y_R2L_train.astype(int))
clf_SVM_U2R.fit(X_U2R_train, Y_U2R_train.astype(int))
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```

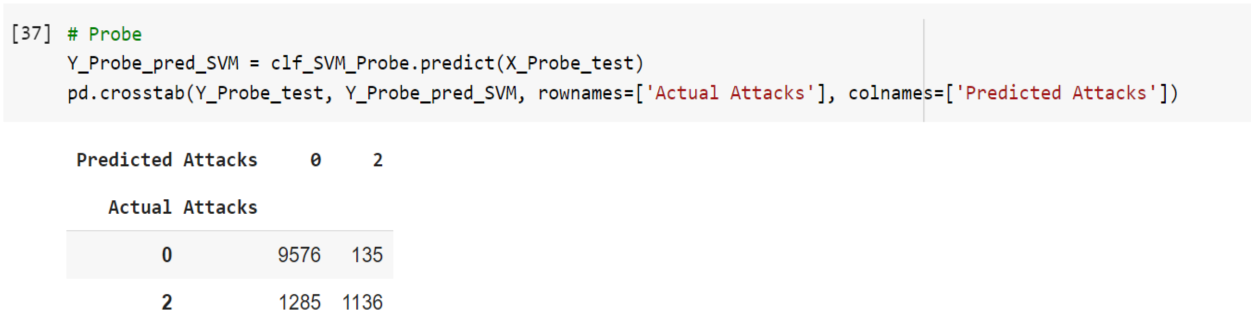
Now it’s the turn of the fitting step, we give our training and testing sets to SVM for all the four different possible attacks and get our four models ready.

3) Now the time for making predictions using our four models-



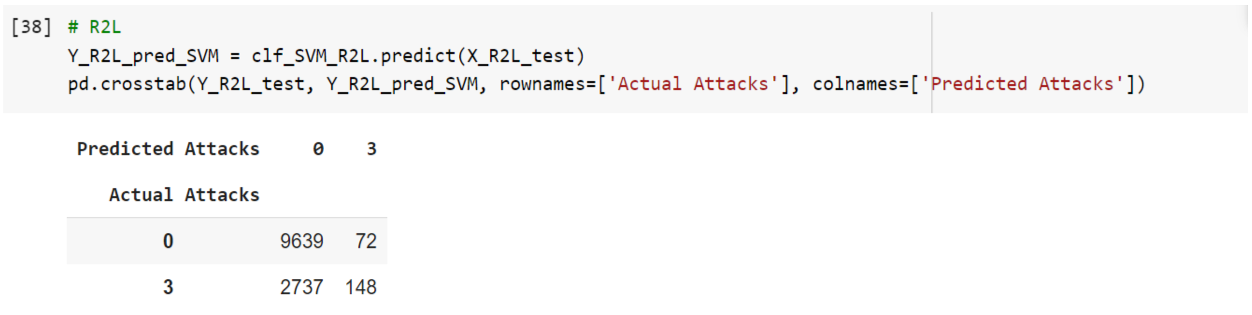
```
[36] # DOS
     Y_DOS_pred_SVM = clf_SVM_DOS.predict(X_DOS_test)
     pd.crosstab(Y_DOS_test, Y_DOS_pred_SVM, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

```
Predicted Attacks     0     1
   Actual Attacks
                0  9455   256
                1  1359  6101
```



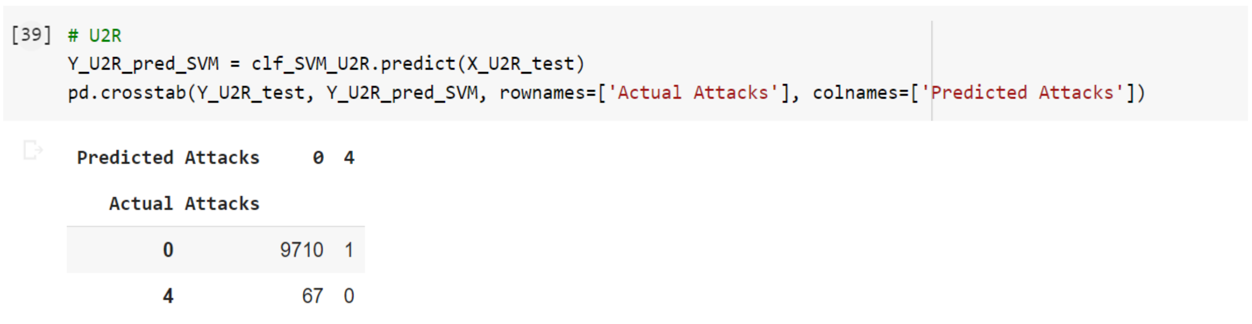
```
[37] # Probe
     Y_Probe_pred_SVM = clf_SVM_Probe.predict(X_Probe_test)
     pd.crosstab(Y_Probe_test, Y_Probe_pred_SVM, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

```
Predicted Attacks     0     2
   Actual Attacks
                0  9576   135
                2  1285  1136
```



```
[38] # R2L
     Y_R2L_pred_SVM = clf_SVM_R2L.predict(X_R2L_test)
     pd.crosstab(Y_R2L_test, Y_R2L_pred_SVM, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

```
Predicted Attacks     0    3
   Actual Attacks
                0  9639   72
                3  2737  148
```



```
[39] # U2R
     Y_U2R_pred_SVM = clf_SVM_U2R.predict(X_U2R_test)
     pd.crosstab(Y_U2R_test, Y_U2R_pred_SVM, rownames=['Actual Attacks'], colnames=['Predicted Attacks'])
```

```
Predicted Attacks     0  4
   Actual Attacks
                0  9710  1
                4    67  0
```

pd.crosstab( ) -> This basically helps in the generation of a confusion matrix so as to measure the different parameters for our classifier.

Now, Cross Validation: Accuracy, Precision, Recall, F-measure calculations-

```
[40] # DOS
     from sklearn.model_selection import cross_val_score
     from sklearn import metrics
     accuracy = cross_val_score(clf_SVM_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='accuracy')
     print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_SVM_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='precision')
     print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_score(clf_SVM_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='recall')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_SVM_DOS, X_DOS_test, Y_DOS_test, cv=10, scoring='f1')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

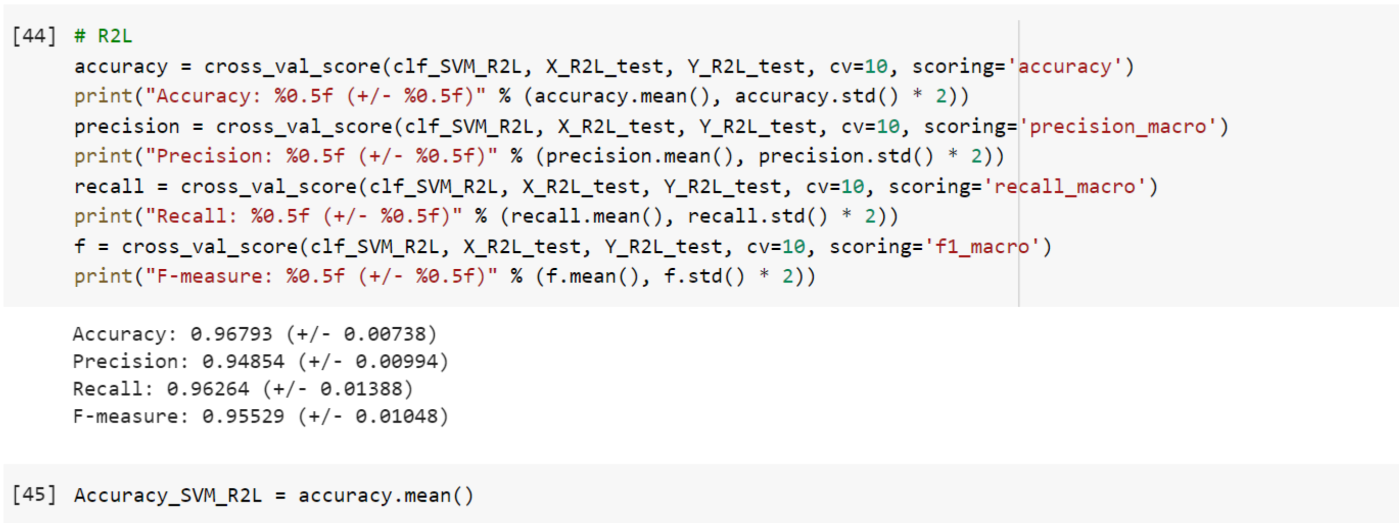
```
     Accuracy: 0.99371 (+/- 0.00375)
     Precision: 0.99107 (+/- 0.00785)
     Recall: 0.99450 (+/- 0.00388)
     F-measure: 0.99278 (+/- 0.00428)
```

```
[41] Accuracy_SVM_DOS = accuracy.mean()
```

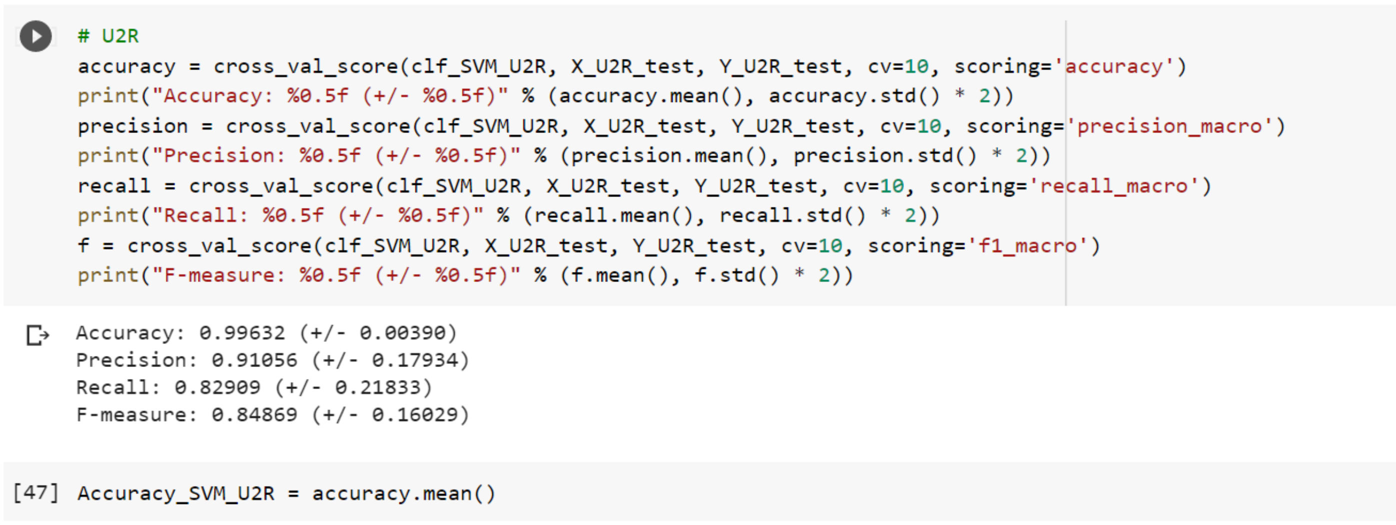
```
[42] # Probe
     accuracy = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
     print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
     print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
     Accuracy: 0.98450 (+/- 0.00526)
     Precision: 0.96907 (+/- 0.01031)
     Recall: 0.98365 (+/- 0.00686)
     F-measure: 0.97613 (+/- 0.00800)
```

```
[43] Accuracy_SVM_Probe = accuracy.mean()
```



```
[44] # R2L
     accuracy = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
     print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
     print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

     Accuracy: 0.96793 (+/- 0.00738)
     Precision: 0.94854 (+/- 0.00994)
     Recall: 0.96264 (+/- 0.01388)
     F-measure: 0.95529 (+/- 0.01048)

[45] Accuracy_SVM_R2L = accuracy.mean()
```

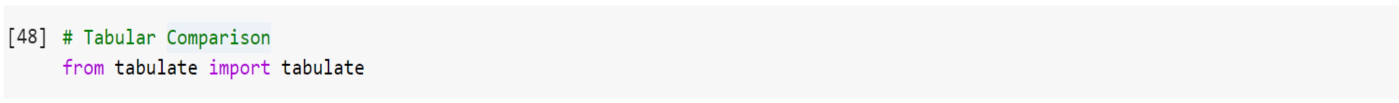


```
     # U2R
     accuracy = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
     print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
     precision = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
     print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
     recall = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
     print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
     f = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
     print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

     Accuracy: 0.99632 (+/- 0.00390)
     Precision: 0.91056 (+/- 0.17934)
     Recall: 0.82909 (+/- 0.21833)
     F-measure: 0.84869 (+/- 0.16029)

[47] Accuracy_SVM_U2R = accuracy.mean()
```

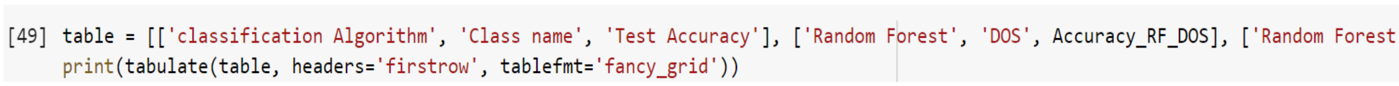
Hence, by using the cross_val_score library, we got our different scores for all the four types of attacks.

RESULTS & DISCUSSIONS-

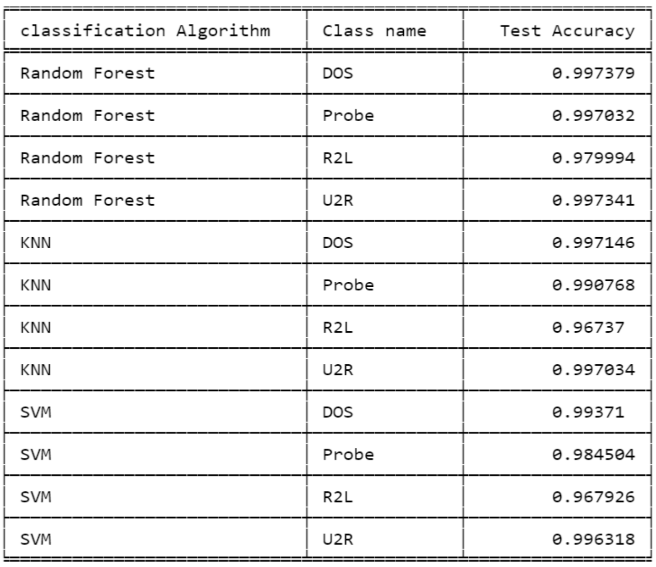
In this section, accuracies of various algorithms are being compared by using graphical analysis.



```
[48] # Tabular Comparison
     from tabulate import tabulate
```

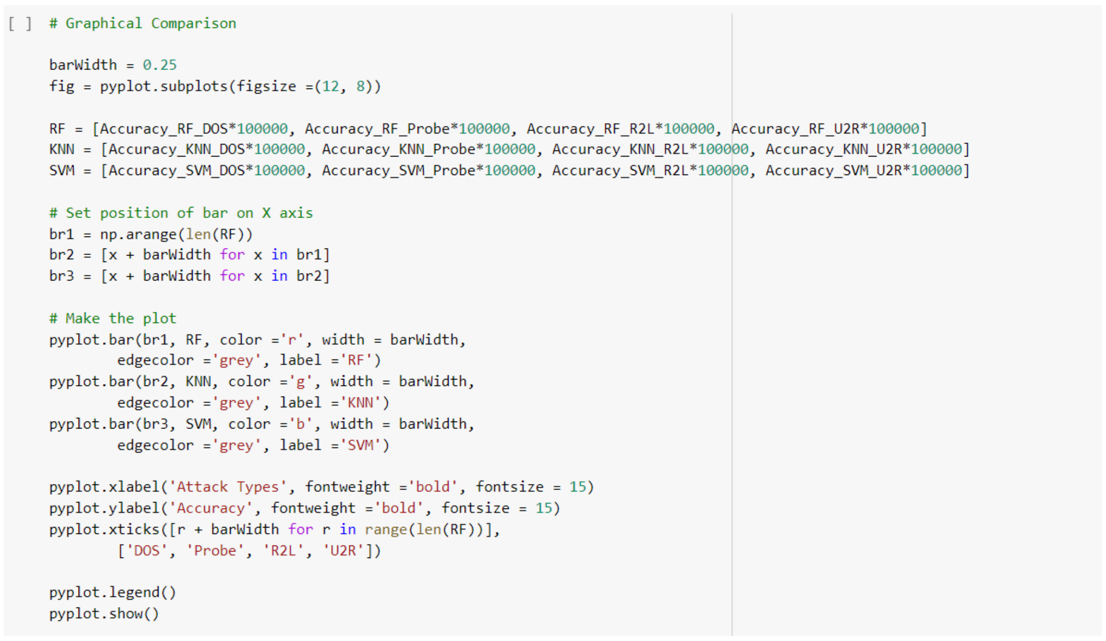


```
[49] table = [['classification Algorithm', 'Class name', 'Test Accuracy'], ['Random Forest', 'DOS', Accuracy_RF_DOS], ['Random Forest
     print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

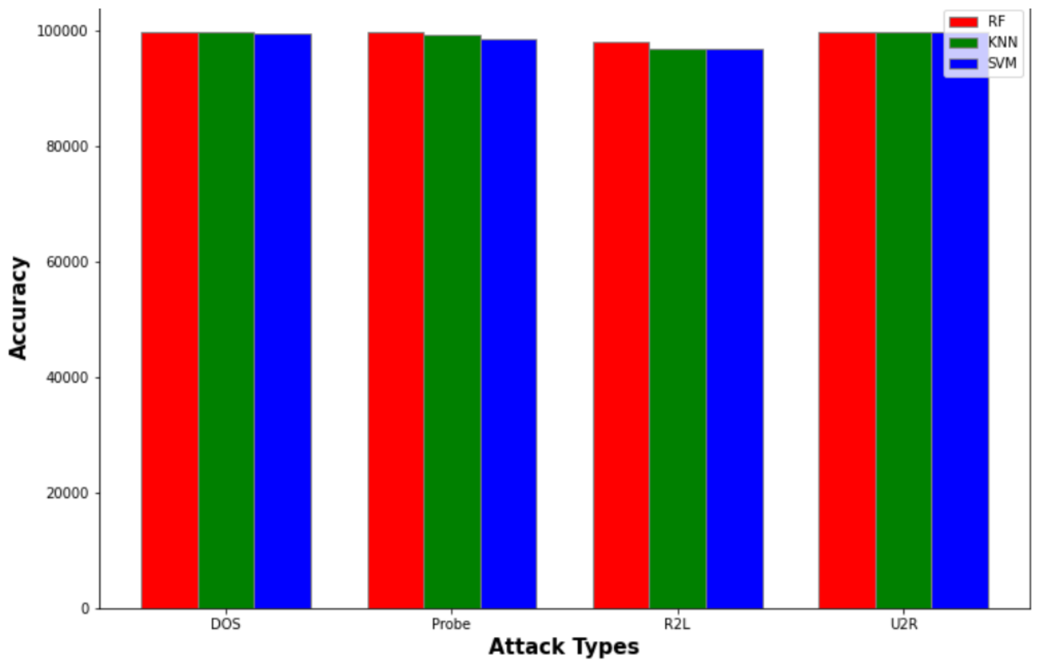


```
╒════════════════════════════╤══════════════╤═════════════════╕
│ classification Algorithm   │ Class name   │   Test Accuracy │
╞════════════════════════════╪══════════════╪═════════════════╡
│ Random Forest              │ DOS          │        0.997379 │
├────────────────────────────┼──────────────┼─────────────────┤
│ Random Forest              │ Probe        │        0.997032 │
├────────────────────────────┼──────────────┼─────────────────┤
│ Random Forest              │ R2L          │        0.979994 │
├────────────────────────────┼──────────────┼─────────────────┤
│ Random Forest              │ U2R          │        0.997341 │
├────────────────────────────┼──────────────┼─────────────────┤
│ KNN                        │ DOS          │        0.997146 │
├────────────────────────────┼──────────────┼─────────────────┤
│ KNN                        │ Probe        │        0.990768 │
├────────────────────────────┼──────────────┼─────────────────┤
│ KNN                        │ R2L          │        0.96737  │
├────────────────────────────┼──────────────┼─────────────────┤
│ KNN                        │ U2R          │        0.997034 │
├────────────────────────────┼──────────────┼─────────────────┤
│ SVM                        │ DOS          │        0.99371  │
├────────────────────────────┼──────────────┼─────────────────┤
│ SVM                        │ Probe        │        0.984504 │
├────────────────────────────┼──────────────┼─────────────────┤
│ SVM                        │ R2L          │        0.967926 │
├────────────────────────────┼──────────────┼─────────────────┤
│ SVM                        │ U2R          │        0.996318 │
╘════════════════════════════╧══════════════╧═════════════════╛
```

Here we can see that accuracy is almost similar using various algorithms, next we will visualize this using a graph for better visualizations.



```
[ ]  # Graphical Comparison

     barWidth = 0.25
     fig = pyplot.subplots(figsize =(12, 8))

     RF = [Accuracy_RF_DOS*100000, Accuracy_RF_Probe*100000, Accuracy_RF_R2L*100000, Accuracy_RF_U2R*100000]
     KNN = [Accuracy_KNN_DOS*100000, Accuracy_KNN_Probe*100000, Accuracy_KNN_R2L*100000, Accuracy_KNN_U2R*100000]
     SVM = [Accuracy_SVM_DOS*100000, Accuracy_SVM_Probe*100000, Accuracy_SVM_R2L*100000, Accuracy_SVM_U2R*100000]

     # Set position of bar on X axis
     br1 = np.arange(len(RF))
     br2 = [x + barWidth for x in br1]
     br3 = [x + barWidth for x in br2]

     # Make the plot
     pyplot.bar(br1, RF, color ='r', width = barWidth,
             edgecolor ='grey', label ='RF')
     pyplot.bar(br2, KNN, color ='g', width = barWidth,
             edgecolor ='grey', label ='KNN')
     pyplot.bar(br3, SVM, color ='b', width = barWidth,
             edgecolor ='grey', label ='SVM')

     pyplot.xlabel('Attack Types', fontweight ='bold', fontsize = 15)
     pyplot.ylabel('Accuracy', fontweight ='bold', fontsize = 15)
     pyplot.xticks([r + barWidth for r in range(len(RF))],
             ['DOS', 'Probe', 'R2L', 'U2R'])

     pyplot.legend()
     pyplot.show()
```



(Multiplied by 100000 because for less precision (decimal points) accuracy was almost sam

8. Conclusion

We observe finally RF (Random Forest) has the highest accuracy in comparison with other classifiers (KNN and SVM) for intrusion detection systems. Thus, an Intrusion Detection

System (IDS) driven by NSL-KDD dataset can be constructed by training with Random Forest Algorithm.

Future Scope of Improvement

In the future we can work on exploring new techniques having better accuracy rate than our current model. One such technique that can be applied is to use feature elimination, which will provide us with a better accuracy. Availability of a better dataset than NSL-KDD will also make our model more efficient.