Network Intrusion Detection Using Data Mining Techniques

INSE 6180 Final Project Report

Oladimeji Salau (40018905)

Submitted to

Professor Nizar Bouguila

Concordia Institute for Information Systems Engineering

Concordia University, Montreal, Canada.

August 2017

Abstract

Network security is extremely important and mission-critical not just only for business continuity but also for thousands of other huge and increasing number of systems and applications running over network continously to deliver services. One of the ways network security is implemented and enforced is via intrusion detection or prevention systems. Traditional intrusion detection systems are usually rule-based and are not effective in detecting new and previously unknown intrusion events. Data mining techniques and machine algorithms have recently gained attention as an alternative approach to proactively detect network security breaches. In this project, six data mining algorithms: Support Vector Machine (SVM), Decision Tree and Random Forest, Naive Baye, K-Nearest Neighbor (KNN) and Logistic Regression classifiers were implemented to detect and classify network intrusion using NSL-KDD dataset. The results obtained generally indicates that models are biased towards classes with low distribution in the dataset.

Key Words — *Intrusion Detection*, *Support Vector Cachine*, *Decision Tree*, *Random Forests*.

Introduction

Intrusion generally refers to malicious activities directed at computer network system to compromise its integrity, availability and confidentiality. Network security is important because modern information technology relies on it to drive businesses and services. Security can be enforced on the network through intrusion detection systems (IDS). These are security devices or software usually implemented by large and medium organizations to enforce security policies and monitor network perimeter against security threats and malicious activities. Other associated systems include Firewall and Intrusion Prevention System (IPS). Essentially, intrusion detection device or application scrutinizes every incoming or outgoing network traffic and analyzes packets (both header and payload) for known and unknown events. Detected known events and violations are logged usually in a central security information and event management (SIEM) system. Malicious activities or unknown events may be set up to alert system administrator or the related packets dropped depending on the configurations enabled on the intrusion detection system.

Prevention of security breaches cannot be completely avoided. Hence, effective intrusion detection becomes important for organizations to proactively deal with security threats in their networks. However, many existing intrusion detection systems are rule-based and are not quite effective in detecting a new intrusion event that has not been encoded in the existing rules. Besides, intrusion detection rules development is time consuming and it is limited to knowledge of known intrusions only.

Data mining techniques, on the other hand, through supervised and unsupervised learning algorithms have been shown to be effective in identifying and differentiating known and new intrusions from network event records or data. It is therefore worthwhile to explore application of data mining techniques as an effective alternative approach to detect known and potential network intrusions.

The goal of this project is to implement data mining techniques in building models to classify network intrusions using algorithms such as Support Vector Machine (SVM), Decision Tree and Random Forest, Naive Baye, K-Nearest Neighbor (KNN) and Logistic Regression. Three of the algorithms: Support Vector Machine (SVM), Decision Tree and Random Forest classifier algorithms will be implemented in line with the following papers: Peddabachigari et al.(2007) [1], Zhang et al.(2008) [2], Panda et al.(2011) [3].

Related Works

Peddabachigari et al. [1] explored two hybrid techniques that involve combining Decision Trees (DT) and Support Vector Machines (SVM) in a hierarchical model and an ensemble approach combining the base classifiers in order to enhance intrusion detection accuracy and minimize computational cost. They concluded from their empirical results that the hybrid approach provides more accurate detection than a direct SVM or Decision Tree techniques.

Zhang et al. [2] applied Random Forests algorithm in misuse, anomaly and hybrid detection using Knowledge Discovery and Data Mining 1999 (KDD'99) dataset. They found that their hybrid system recorded higher overall detection performance with low false positive than other intrusion detection approaches.

Panda et al. [3] proposed a network intrusion detection framework that uses ten machine learning approaches including Decision Tree (J48), Bayesian Belief Network, Hybrid Naïve Bayes with Decision Tree, Rotation Forest, Hybrid J48 with lazy locally weighted learning, Discriminative multinomial Naïve Bayes, Combining Random Forests with Naïve Bayes and finally ensemble of classifiers using J48 and Naïve Bayes with AdaBoost (AB). The empirical results from their research suggest that the proposed approach of using various machine learning methodologies performed reasonably well in all categories of majority attacks investigated.

Project Scope

Intrusion detection using data mining techniques generally follows two main approaches; misuse detection and anomaly detection [4]. Misuse approach involves labeling each instance in a dataset as 'normal' or 'attack' and a learning algorithm trained on the data. These techniques are reusable in retraining intrusion detection models on different input dataset that include new types of attacks, as long as they have been labeled appropriately. A major advantage of misuse detection techniques is their ability to achieve high degree of accuracy in detecting known attacks and their variations. Their known disadvantage is the inability to detect attacks whose instances have not yet been observed [4].

Anomaly detection approach on the other hand, builds models of normal behavior and

detects any deviations as potential attacks. Anomaly detection techniques are thus able to detect new types of intrusions as deviations from normal usage. A potential disadvantage of anomaly detection techniques is the rate of false alarms that may have negative impact on intrusion detection effort [4].
The scope in this project is limited misuse detection approach.

Classification Algorithms

Support Vector Machines (SVM): The key objective of SVM is to draw a hyperplane that separates the two classes optimally such that the margin is maximum between the hyperplane and the observations. Although, it is possible to highlight different hyperplanes within a given space, the goal of SVM is to find the one that produces a high margin [4]. As all cases of classifications cannot be handled linearly, SVM employs some kernel functions like polynomial or radial basis function to transform the linear algorithms to nonlinear ones [3].
SVM belongs to the category of supervised learning algorithms that can analyze data and recognize patterns and are used for classification and regression analysis tasks. In addition to performing linear classification, SVMs can also efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces [5]. The effectiveness of SVM depends on the selection of kernel, the kernel's parameters, and soft margin parameter C. A common choice is a Gaussian kernel, which has a single parameter (gamma).
A potential drawback of the SVM is that the parameters of a solved model are difficult to interpret [5]. Despite this, SVM has found a wide applications especially in text classification tasks.

Decision trees (DT): Decision tree constructs a tree-like structure where each internal (non-leaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf) node denotes a class prediction. At each node, the algorithm chooses the best attribute to classify data into individual classes [5]. Decision tree is one of the most commonly used machine learning approaches in the field of intrusion detection [3].
Decision trees are used in data mining either for classification where the predicted outcome is the class to which the data belongs or regression analysis where the predicted

outcome can be considered a real number e.g. the price of a house [5]. Decision trees have also been used in developing certain techniques called ensemble methods to construct more than one decision trees. Such techniques include:
Bagging decision trees: an early ensemble method, which builds multiple decision trees by repeatedly resam- pling training data with replacement, and voting the trees for a consensus prediction [5].
Random Forest classifier: which uses a number of decision trees, in order to improve the classification rate.
Rotation Forest: in which every decision tree is first trained by applying principal component analysis (PCA) on a random subset of the input features [5].

Random Forests: Random Forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forests correct for decision trees' habit of overfitting to their training set [5].
One of the key advantages of Random Forests is that they can be used to rank the importance of variables in a regression or classification problem [5].

Project Methodology

The implementation methodology used in the project follows these functional steps: Environment Setup, Data Loading, Data Preprocessing, Train Models, Evaluate Models and Test Models. The project was implemented using Python programming language. Python, an open source general programming language has a well documented machine learning module called Scikit-learn with a wide range of machine learning algorithms for medium-scale supervised and unsupervised problems.

Project Implementation and Results

KDD Dataset

The dataset used in this project is the NSL-KDD dataset from the University of New Brunswick, Canada. The dataset is an improved version of the KDDCUP'99 dataset from DARPA Intrusion Detection Evaluation Program. The original KDD dataset is perhaps the most widely used dataset for machine learning intrusion detection tasks [7]. However, the results of statistical analysis conducted by Tavallaee et al. [7] revealed that the dataset is fraught with redundant records that can lead to poor evaluation of anomaly detection tasks. A new dataset, NSL-KDD, devoid of the deficiencies noted in KDDCUP'99 data has since been proposed by Tavallaee et al. [7]. The NSL-KDD dataset used in this project consists of 125,973 records training set and 22,544 records test set with 42 attributes/features for each connection sample including class label containing the attack types.

Data Load and Preprocessing

A. Mapping intrusion types to attack classes

After loading the dataset into Python (Jupyter Notebook) development environment, the first task performed was mapping various attack types in the dataset into four attack classes as described by Tavallaee et al. [7].

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

In [10]:
```
# Descriptive statistics
dfkdd_train.describe()
```

Out[10]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromis
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000(
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111	0.204409	0.001222	0.395736	0.279:
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366	2.149968	0.045239	0.489010	23.942(
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000(
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	77.000000	5.000000	1.000000	7479.000(

Figure 1

In [13]:
```
# 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.su
attack_class_freq_test['frequency_percent_test'] = round((100 * attack_class_freq_test / attack_class_freq_test.sum())

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

Out[13]:

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

Figure 2

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.

B. Exploratory Data Analysis (EDA)

Basic exploratory data analyses were carried out among other things to understand the descriptive statistics of the dataset, find instances of missing values and redundant features, explore the data type and structure and investigate the distribution of attack class in the dataset.

In [14]:
```
# 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);
```

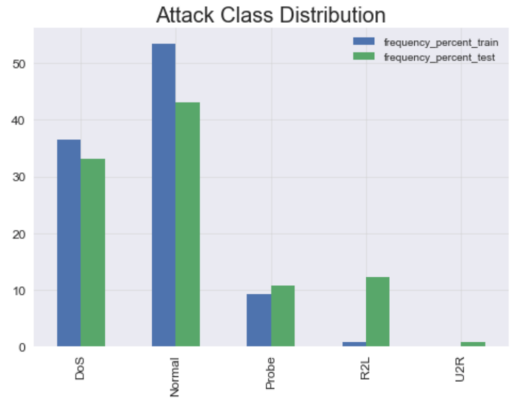

Figure 3

In [16]:
```
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)
```

Figure 4

Standardization of Numerical Attributes

The numerical features in the dataset were extracted and standardized to have zero mean and unit variance. This is a common requirement for many machine learning algorithms implemented in Scikit-learn python module.

Encoding Categorical Attributes

The categorical features in the dataset were encoded to integers. This is also a common requirement for many machine learning algorithms implemented in Scikit-learn.

```
In [17]: 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()
```

Figure 5

C. Data Sampling

The sparse distribution of certain attack classes such as *U2R* and *L2R* in the dataset while others such as *Normal*, *DoS* and *Probe* are significantly represented inherently leads to the situation of imbalance dataset. While this scenario is not unexpected in data mining tasks involving identification or classification of instances of deviations from normal patterns in a given dataset, research has shown that supervised learning algorithms are often biased against the target class that is weakly represented in a given dataset.

Certain approaches such as random data sampling and cost-sensitive learning method [8] have been suggested to address the problem of imbalance dataset. The use of sampling involves modification of an imbalanced data set by some mechanisms in order to provide a balanced distribution. While oversampling replicates and increases data in class label with low distribution, random undersampling removes data from the original dataset with high class frequency [8].

Cost-sensitive learning focuses on the imbalanced learning problem by using different cost matrices that describe the costs for misclassifying any particular data example rather than creating balanced data distributions through different sampling techniques [8].

Studies have shown that for several base classifiers, a balanced data set provides improved overall classification performance compared to an imbalanced data set [9]. Popular sampling techniques includes synthetic minority oversampling technique (SMOTE) and Random Oversampling and Undersampling.

In this project random oversampling technique was used to balance class distribution in the training dataset as shown below.

In [18]:

```
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_sample(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({0: 67343, 1: 67343, 2: 67343, 3: 67343, 4: 67343})
```

	Class Distribution	
	Before Sampling	After Sampling
Normal	67343	67343
DoS	45927	67343
Probe	11656	67343
L2R	995	67343
U2R	52	67343

D. Feature Selection

Feature selection is a key data preprocessing step in data mining task involving selection of important features as a subset of original features according to certain criteria to reduce dimension in order to improve the efficiency of data mining algorithms [8]. Most of the data includes irrelevant, redundant, or noisy features. Feature selection reduces the number of features,removes irrelevant, redundant, or noisy features, and brings about palpable effects on applications: speeding up a data mining algorithm, improving learning accuracy, and leading to better model comprehensibility [10]. There are two common approaches to select or reduce the features; a wrapper uses the intended learning algorithm itself to evaluate the usefulness of features, and a filter evaluates features according to heuristics based on general characteristics of the data. The

```
In [19]: 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();
```

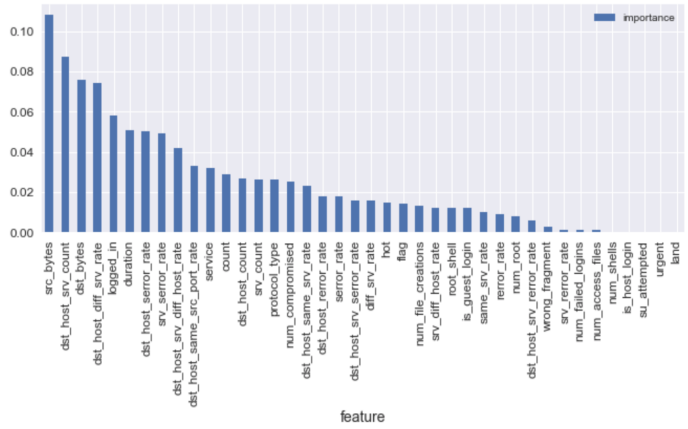

Figure 6

```
In [21]: selected_features

Out[21]: ['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']
```

Figure 7

wrapper approach is generally considered to produce better feature subsets but runs much more slowly than a filter [11].

In this project, a wrapper approach was used wherein a RandomForest classifier algorithm with a function to define feature importance was trained to extract feature importance from the training dataset. A second step implemented involved using a recursive feature extraction also based on RandomForest algorithm to extract top 10 features relevant to achieve accurate classification of classes in the training set.

```
In [23]: 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()

In [24]: for k, v in classdict.items():
             k

Out[24]: 'Normal_DoS'

Out[24]: 'Normal_U2R'

Out[24]: 'Normal_Probe'

Out[24]: 'Normal_R2L'
```

Figure 8

E. Data Partition

After selection of relevant features, The resampled dataset based on the selected features was partitioned into two-target classes (normal class and an attack class) for all the attack classes in the dataset to facilitate binary classification.

Train Models

The training dataset was used to train the following classifier algorithms: Support Vector Machine (SVM), Decision Tree, Random Forest, Naïve Baye, Logistic Regression and k-Nearest Neighbor (kNN). In addition, an ensemble VotingClassifier was trained to combine and average the prediction results from all the individually trained classifiers. The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses [12].

In [27]:

```
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 SVM Model
SVC_Classifier = SVC(kernel='poly', C=1, gamma=1, random_state=0)
SVC_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='gini', random_state=0)
DTC_Classifier.fit(X_train, Y_train);

# Train RandomForestClassifier Model
RF_Classifier = RandomForestClassifier()
RF_Classifier.fit(X_train, Y_train);

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

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression()
LGR_Classifier.fit(X_train, Y_train);

## Train Ensemble Model (This method combines NB + DTC + KNN + LGR)
combined_model = [('Naive Baye Classifier', BNB_Classifier),
                  ('Decision Tree Classifier', DTC_Classifier),
                  ('KNeighborsClassifier', KNN_Classifier),
                  ('LogisticRegression', LGR_Classifier)]
VotingClassifier =  VotingClassifier(estimators = combined_model)
VotingClassifier.fit(X_train, Y_train);
```

Figure 9

Evaluate Models

The trained models were evaluated using a 10-fold cross validation technique. The concept of k-fold cross validation involves generation of validation set out of training dataset to assess the model before it is exposed to test data.

In k-fold cross-validation, the training dataset is randomly divided into k equal sized subsamples. Out of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged or combined to generate a single value. The advantage of this method is that all samples are used for both training and validation [5].

In [ ]:

```
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()
```

Figure 10

Test Models Performance

The trained models were used to classify labels in a test dataset that has not been previously exposed to the algorithms. Performance metrics such as accuracy, confusion matrix and classification report were generated for each two-target attack class and for each model.

In [29]:

```
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		Normal_Probe		Normal_R2L		Normal_U2R	
	Trained Model Evaluation	Model Test Performance	Trained Model Evaluation	Model Test Performance	Trained Model Evaluation	Model Test Performance	Trained Model Evaluation	Model Test Performance
	Detection Accuracy (%)	Detection Accuracy (%)	Detection Accuracy (%)	Detection Accuracy (%)	Detection Accuracy (%)	Detection Accuracy (%)	Detection Accuracy (%)	Detection Accuracy (%)
SVM	98.33	84.55	97.47	87.66	97.38	80.26	99.31	97.10
Naive Baye	97.13	82.94	94.99	82.00	94.56	75.53	93.34	91.03
Decision Tree	99.99	79.04	99.97	52.46	99.97	79.88	99.99	97.98
RandomForest	99.99	76.12	99.99	84.60	99.99	77.91	99.99	97.98
KNN	99.71	85.99	99.69	88.30	99.76	78.66	99.94	97.94
Logistic Regression	98.23	85.05	96.68	84.64	96.45	77.97	95.81	97.88
Voting Classifier	99.78	84.55	99.73	84.30	99.70	80.25	99.92	97.99

CONFUSION MATRIX – MODEL TEST PERFORMANCE

Random Forest

Confusion Matrix		Predicted Label: DoS	Predicted Label: Normal
Actual Label	DoS	6678	780
	Normal	3320	6391

Confusion Matrix		Predicted Label: Normal	Predicted Label: Probe
Actual Label	Normal	8998	713
	Probe	1155	1266

Confusion Matrix		Predicted Label: Normal	Predicted Label: R2L
Actual Label	Normal	9711	0
	R2L	2754	0

Confusion Matrix		Predicted Label: Normal	Predicted Label: U2R
Actual Label	Normal	9711	0
	U2R	200	0

KNN

Confusion Matrix		Predicted Label: DoS	Predicted Label: Normal
Actual Label	DoS	5586	1872
	Normal	534	9177

Confusion Matrix		Predicted Label: Normal	Predicted Label: Probe
Actual Label	Normal	9039	672
	Probe	747	1674

Confusion Matrix		Predicted Label: Normal	Predicted Label: R2L
Actual Label	Normal	9602	109
	R2L	2551	203

Confusion Matrix		Predicted Label: Normal	Predicted Label: U2R
Actual Label	Normal	9707	4
	U2R	200	0

Logistic Regression

Confusion Matrix		Predicted Label: DoS	Predicted Label: Normal
Actual Label	DoS	5866	1592
	Normal	974	8737

Confusion Matrix		Predicted Label: Normal	Predicted Label: Probe
Actual Label	Normal	8358	1353
	Probe	511	1910

Confusion Matrix		Predicted Label: Normal	Predicted Label: R2L
Actual Label	Normal	9254	457
	R2L	2289	465

Confusion Matrix		Predicted Label: Normal	Predicted Label: U2R
Actual Label	Normal	9697	14
	U2R	196	4

Figure 11

CONFUSION MATRIX – MODEL TEST PERFORMANCE

SVM

Confusion Matrix		Predicted Label: DoS	Predicted Label: Normal
Actual Label	DoS	5399	2059
	Normal	562	9149

Confusion Matrix		Predicted Label: Normal	Predicted Label: Probe
Actual Label	Normal	9063	648
	Probe	849	1572

Confusion Matrix		Predicted Label: Normal	Predicted Label: R2L
Actual Label	Normal	9671	40
	R2L	2421	333

Confusion Matrix		Predicted Label: Normal	Predicted Label: U2R
Actual Label	Normal	9598	113
	U2R	175	25

Naive Baye

Confusion Matrix		Predicted Label: DoS	Predicted Label: Normal
Actual Label	DoS	5544	1914
	Normal	1015	8696

Confusion Matrix		Predicted Label: Normal	Predicted Label: Probe
Actual Label	Normal	8030	1681
	Probe	502	1919

Confusion Matrix		Predicted Label: Normal	Predicted Label: R2L
Actual Label	Normal	8915	796
	R2L	2254	500

Confusion Matrix		Predicted Label: Normal	Predicted Label: U2R
Actual Label	Normal	8982	729
	U2R	160	40

Decision Tree

Confusion Matrix		Predicted Label: DoS	Predicted Label: Normal
Actual Label	DoS	5322	2136
	Normal	1462	8249

Confusion Matrix		Predicted Label: Normal	Predicted Label: Probe
Actual Label	Normal	4206	5505
	Probe	262	2159

Confusion Matrix		Predicted Label: Normal	Predicted Label: R2L
Actual Label	Normal	9631	80
	R2L	2427	327

Confusion Matrix		Predicted Label: Normal	Predicted Label: U2R
Actual Label	Normal	9711	0
	U2R	200	0

Results and Discussion

From the results obtained, all the models evaluated achieved an average of 99% on training set while models performance on test set indicates an average of more than 80% across all the four attack groups investigated.
The test accuracy for 'Normal_U2R' across all the models showed a value of more than 90%. However, a review of performance as shown by the confusion matrix indicated that the 'U2R' detection rate was very poor across Decision Tree, RandomForest, KNN and Logistic Regression models. The attack class 'R2L' detection rate is also not far from being poor as 'Normal' and other classes with higher distribution got more attention of the models to the detriment of the low distribution classes.
The results generally showed that sampling may improve model training accuracy. However, a poor detection rate in detecting U2R and R2L minority attacks are inevitable because of their large bias available in the dataset [2].

Conclusion

A data mining project was carried out to build models for classifying network intrusions using NSL-KDD dataset. Classifier algorithms such as Support Vector Machine (SVM), Naive Baye, Decision Tree, K-Nearest Neighbor, Logistic Regression and Random Forest were trained, evaluated and tested to determine each model performance. Additionally, an ensemble VotingClassifier was also trained to combine and average the individual model performances.
A key lesson learned in this project is that, for classification task, accuracy is not a good measure of a model performance where there is an imbalance dataset. Accuracy value may be high for the model but the class with lower samples may not be effectively classified. Metrics such as Confusion Matrix is more realistic in evaluating classifier model performance than accuracy measure.

References

[1] S. Peddabachigiri, A. Abraham., C. Grosan and J. Thomas, "Modeling of Intrusion Detection System Using Hybrid Intelligent Systems" , Journals of Network Computer Application, 2007. Available from: https://www.researchgate.net/publication/222527188_Modeling_intrusion_detection_system_using_hybrid_intelligent_systems [accessed Jul 11, 2017].

[2] J. Zhang, M. Zulkernine, A. Haque "Random-Forests-Based Network Intrusion Detection Systems", IEEE Trans. Syst. Man Cybernet. Part C Appl. Rev. 8:649–659, 2008.

[3] M. Panda, A. Abraham, S. Das, M.R. Patra, "Network Intrusion Detection System: A Machine Learning Approach", Intelligent Decision Technologies 5(4), 347–356, 2011. Available from: https://www.researchgate.net/publication/220468036_Network_intrusion_detection_system_A_machine_learning_approach [accessed Jul 11, 2017].

[4] Paul, D, et al. "Data mining for network intrusion detection." Proc. NSF Workshop on Next Generation Data Mining. 2002.

[5] Data Science Association, "Introduction to Machine Learning", The Wikipedia Guide, 2016. Avaialable from: http://www.datascienceassn.org/content/introduction-machine-learning [accessed Aug 10 2017].

[6] M. Swamynathan "Mastering Machine Learning with Python in Six Steps", ISBN-13: 978-1-4842-2865-4, Apress, 2017.

[7] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani, "A Detailed analysis of the KDD CUP 99 Data Set". In the Proc. Of the IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2009), pp. 1-6, 2009.

[8] H. He and E.A. Garcia, "Learning from Imbalanced Data", IEEE Transactions On Knowledge And Data Engineering, Vol.21, No.9, 2009.

[9] G.M. Weiss and F. Provost, "The Effect of Class Distribution on Classifier Learning: An Empirical Study," Technical Report ML-TR-43, Dept. of Computer Science, Rutgers Univ., 2001.

[10] Y. Kim, W. N. Street, F. Menczer, and G. J. Russell, "Feature selection in data mining" in Data Mining: Opportunities and Challenges: J. Wang, Ed. Hershey, PA: Idea Group Publishing, 2003, pp. 80–105. [11] Liu H ,Setiono R, Motoda H, Zhao Z, Feature Selection: An Ever Evolving Frontier in Data Mining, JMLR: Workshop and

Conference Proceedings 10, pp. 4-13, 2010.
[12] Pedregosa et al., “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research 12, pp. 2825-2830, 2011.