Network Intrusion Detection using Decision Trees

Edward Nataniel C. Apostol
2010-27627
CS 176 THX
edward.nataniel@gmail.com

1. Introduction

A network intrusion is defined as any form of unauthorized activity on a computer network. The activity can be in a form of scans, attacks, or misuses of the resources of the digital network. Today, there is an increase of network intrusion attacks due to digitalization. The difficulty of detecting and countering these attacks are also increasing.

This analysis will demonstrate how a machine learning technique can be used to gain insights on how to detect and counter these attacks.

2. Objectives

The objective of this analysis is to gain useful insights from the different features that might determine whether a connection is normal or anomalous. Using the obtained information, it will be easier to detect and counter these attacks.

3. Methodology

The dataset in this study came from the Defense Advanced Research Projects Agency (DARPA), an agency in the United States that is responsible for the development of military technologies.

The dataset was obtained from several gigabytes of binary TCP dump data coming from more than a month of network traffic.

The file *kddcup_data_10_percent.csv* contains a subset of millions of connection records from a military environment. The dataset has a total of 494021 records and 42 attributes. The 41 features can be categorized into three:

1. Basic features of TCP connections
2. Content features within a connection as suggested by domain knowledge
3. Traffic features computed within a two-second time window

The data in this file will be used to build the classifier.

The file *kddcup_testdata_unlabeled_10_percent.csv* is the test data set. The generated decision tree model will decide on the labels of the unlabeled records.

The file *corrected.csv* is a file downloaded from the UCI Knowledge Discovery in Databases Archive. It contains the corrected labels for the test data. This labels are necessary to be able to compute the test accuracy.

The file *pa2.r* is a script written to train the C5.0 decision tree. The tree will contain the rules generated from the training data that will be used to predict the labels of the unlabeled records.

3.1 Package Installation

The package *c50* will be used to create the decision tree and the collection of rules. The package also contains functions that plots the tree, predict new samples, etc.

```
> install.packages("C50")
> library(C50)
```

3.2 Reading and Storing the Training Data

```
> read.csv("kddcup_data_10_percent.csv", header
    = FALSE)
> colnames(trainData) <- colLabels

# x is the dataframe of predictors
# y is the factor vector
> x <- trainData[, 1:41]
> y <- trainData[, 42]
```

The training data will come from the file named *kddcup_data_10_percent.csv*. The variable x will store the predictors or the first 41 columns of the data while the variable y will store the labels.

3.3 Training the Decision Tree

```
> treeModel <- C50::C5.0(x, y)
> summary(treeModel
```

The decision tree will be trained by calling the *c5.0* function from the *c50* package. It will take some time for the training to be completed because of the huge amount of training data. The *summary* function prints out the detailed summary of the c5.0 model. It prints the structure of the tree, evaluation on the training data, and the attribute usage in percent. This function will be useful in determining the most predictive attributes.

3.4 Computing for the Classification Accuracy for the Training Set

The classification accuracy for the training set can be computed by predicting the labels of the training set and comparing the predicted labels to the actual labels from the original dataset.

```
> predictedLabels <- predict(treeModel, x,
type = "class")
> trainingAccuracy <- sum(as.character(
    predictedLabels) == as.character(y)) /
    length (y)
> trainingAccuracy
```

Predicting the labels can be done by using the predict function from the c50 package. The second line in the code snippet above counts the number of matches between the predicted labels and the actual labels.

3.5 Generation of Rules

```
> treeModel2 <- C50::C5.0(x, y, rules = TRUE)
> summary(treeModel2)
```

The set of rules can be generated from the training data by just calling the c50 function and adding the argument rules=TRUE. By calling the summary function again, the list of rules will be printed.

3.6 Predicting the Labels for the Test Data

The *predict* function from the c50 package was used in predicting the labels for the test data. The test data include type of attacks that are not in the training data.

```
> testData = read.csv("
    kddcup_testdata_unlabeled_10_percent.csv",
    header = FALSE)
> colnames(testData) <- colLabels
> predictedLabels2 <- predict(treeModel,
     testData, type = "class")
```

The confidence values of the prediction can also be displayed by changing the type argument from *class* to *prob*. Almost all of the confidence values of the prediction are above 99.9%.

3.7 Computing for the classifaction accuracy for the test data

Since the test data is unlabeled, the labels can be downloaded from the UCI Knowledge Discovery in Databases Archive (http://kdd.ics.uci.edu) to calculate for the classification accuracy of the model to the test data set.

```
> testDataCorrected <- read.csv("
    corrected.csv", header = FALSE)
> colnames(testDataCorrected) <- colLabels
> testDataCorrected <- testDataCorrected[,42]
> testAccuracy <- sum(
    as.character(predictedLabels2)
     == as.character(testDataCorrected))/
     length(testDataCorrected)
> testAccuracy
```

Predicting the labels can be done by using the *predict* function from the *c50* package. The 4th line of the code snippet above counts the number of matches between the predicted labels and the actual labels.

4. Experimental Results

The classification accuracy for the given training set is 99.97% while the training accuracy for the given testing dataset is 91.64%. Also, the confidence values of the prediction are above 99.90%.

The *summary* function prints out a detailed summary of the c5.0 model. It includes the attribute usage or the importance of the attribute. The importance is represented by the percentage of records that fall under that fall into terminal nodes after the split.

```
Attribute usage:
     100.00% wrong_fragment
     99.68% service
     76.85% src_bytes
     18.60% same_srv_rate
     18.38% flag
     18.24% dst_host_serror_rate
     16.85% num_compromised
     15.59% dst_host_same_srv_rate
     11.68% dst_host_srv_count
     11.64% dst_host_count
     10.37% dst_host_diff_srv_rate
     10.25% dst_host_same_src_port_rate
     10.21% dst_host_srv_diff_host_rate
      9.91% srv_serror_rate
      9.76% hot
      9.70% num_failed_logins
      2.11% srv_count
      1.91% logged_in
      1.78% duration
      0.48% diff_srv_rate
      0.46% protocol_type
      0.24% rerror_rate
      0.18% serror_rate
      0.15% root_shell
      0.14% dst_bytes
      0.14% num_root
      0.13% count
      0.02% num_file_creations
      0.00% land
      0.00% num_shells
```

5. Analysis and Discussion of Results

The accuracy of the classifier is quite high for both the training and testing set.

The number of wrong fragments, network service on the destination service, and the number of data bytes from the source to the destination src_bytes, are the most important attributes or the major predictors. They greatly affect the classification of records.

The number of file creations operation, land (whether the connection comes from the same host/port), and number of shell prompts, are the least important attributes.

Some interesting rules with very high confidence are:

1. If the number of source bytes is greater than 10073 and there exist a compromised condition, then the connection is automatically a back attack (a type of denial of service attack).

2. If the service is in any of the following: co_i, ftp, gopher, link, mtp, name, private, remote_job, rje,

ssh, time, and the dst_host_srv_diff_host_rate is greater than 0.48, then the connection is an ipsweep (a type of probing attack).

3. If the flag/error status of the connection is SH and the number of connections to the same service as the current connection in the past two seconds is less than or equal to 80, then the connection is an nmap (a type of probing attack).

6. Conclusion

Machine learning techniques like the decision tree learning can be used to detect network attack intrusions. The model can learn what patterns are normal and what patterns are unusual with a high degree of accuracy. Decision tree learning can generate a model that classifies whether a connection is normal connection or an attack. Taking note of the rules with high confidence can also be very useful.