# Intrusion Detection using LDA and Random Forests on KDD CUP 99 Dataset

Girish Palya

## Contents

## Overview

Machine learning algorithms that utilize Euclidean distance or scalar products (like covariance) for regression or classification perform better when variables are continuous and may do poorly when variables are nominal (categorical). On the other hand, decision tree based algorithms are naturally suited for categorical variables but may do poorly for continuous variables (unless aggregation methods like bagging, random forests, and boosting are used).

Two wildly different learning algorithms are applied on a multi-class classification problem involving a dataset with a number of categorical and numeric variables, and performance is compared. *Linear Discriminant Analysis (LDA)* is used after selecting features through dimensional reduction (employing Principal Component Analysis), and *Random Forests* is used after selecting variables based on mean decrease of Gini coefficient. Random Forests outperformed LDA by a significant margin, even with sub-optimal number of trees.

## Dataset Description

KDD CUP 99 is a particularly challenging dataset with massive size, redundancy, a large number of variables (including both numeric and categorical), and a skewed target variable. It is a popular dataset used for intrusion detection in academic literature. The dataset was originally created in 1999 at MIT's Lincoln Laboratory during a DARPA sponsored event, where a number of attack scenarios were simulated and features were extracted.

Training dataset has 4,898,431 observations. However, due to high redundancy (78%) only 1,074,992 are unique data points. Test dataset has 311,029 examples.

Problem involves *classifying test examples into one of 4 network attack categories and a non-attack ("normal") category.*

```r
library(data.table)
library(caret)
library(MASS)
library(randomForest)
set.seed(1)

# train <- fread('kddcup.data', showProgress = FALSE)
train <- fread('kddcup.data_10_percent', showProgress = FALSE) # testing only
test <- fread('kddcup.test')
col_names <- fread('kddcup.names', skip = 1, header = FALSE, sep = ':')
setnames(train, new = c(col_names[, V1], 'target'))
setnames(test, new = c(col_names[, V1], 'target'))
categorical_vars <- c("target", col_names[grepl("symbolic", V2), V1])
```

### Features

There are 41 variables, 34 numeric and 7 categorical (nominal). One of the nominal variables ("service") has high cardinality (66 classes). Distribution of numeric variables is closer to normal. However, some of the variables are highly correlated.

```
##                    Var1                     Var2 correlation
## 1:          serror_rate          srv_serror_rate           1
## 2:          serror_rate     dst_host_serror_rate           1
## 3:          serror_rate dst_host_srv_serror_rate           1
## 4:      srv_serror_rate     dst_host_serror_rate           1
## 5:      srv_serror_rate dst_host_srv_serror_rate           1
## 6: dst_host_serror_rate dst_host_srv_serror_rate           1
```

### Target Classes

Dataset has five classes for target variable: Four types of attacks and a type for normal connection. Attacks are classified as follows:

- DOS: denial-of-service, e.g., SYN flood attack
- R2L: unauthorized access from a remote machine, e.g. guessing password
- U2R: unauthorized access to local superuser (root) privileges, e.g., various buffer overflow attacks
- Probing: surveillance and other types of probing, e.g., port scanning

Each intrusion is classified into many sub-types of attack: 24 attack types are present in the training data, and an additional 14 types in the test data only. Further, test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This makes the task more realistic.

```
## Only the sub-classes are present in the dataset. Main attack classes
##   are derived and encoded into the data frame.
DoS <- c("back", "land", "neptune", "pod", "smurf", "teardrop",
         "apache2", "udpstorm", "processtable", "worm", "mailbomb")
Probe <- c("satan", "ipsweep", "nmap", "portsweep", "mscan", "saint")
R2L <- c("guess_passwd", "ftp_write", "imap", "phf", "multihop",
         "warezmaster", "xsnoop", "xlock", "snmpguess", "snmpgetattack",
         "httptunnel", "sendmail", "named", "warezclient", "spy")
U2R <- c("buffer_overflow", "loadmodule", "rootkit", "perl", "xterm",
         "sqlattack", "ps")
attack_map <- function(atype) {
  atype_ <- sub("\\.", "", atype)
  if (atype_ %in% DoS) { return("DoS") }
  if (atype_ %in% Probe) { return("Probe") }
  if (atype_ %in% R2L) { return("R2L") }
  if (atype_ %in% U2R) { return("U2R") }
  if (atype_ == "normal") { return("normal") }
  stop(paste("Unknow attack type", atype))
}
train[, target := sapply(target, attack_map)]
test[, target := sapply(target, attack_map)]
```

Target variable is skewed. 99% of the observations belongs to either the Normal or DoS categories.

```
##     Class Training Set Percentage (Training) Test Set Percentage (Test)
## 1:    DoS       391458                79.239   229855            73.901
## 2: normal        97278                19.691    60593            19.481
## 3:  Probe         4107                 0.831     4166             1.339
## 4:    R2L         1126                 0.228    16345             5.255
## 5:    U2R           52                 0.011       70             0.023
```

## Data Preparation

Following transformations are applied on training and test data:

- Remove redundant rows
- Remove variables that have all 0 values
- Convert strings to factors (for nominal variables)

```
# Remove redundant rows
train <- unique(train)

# Remove variables that have all 0's
stopifnot(any(train[, is_host_login == 0]))
stopifnot(any(train[, num_outbound_cmds == 0]))
train[, `:=`(is_host_login = NULL, num_outbound_cmds = NULL)]
test[, `:=`(is_host_login = NULL, num_outbound_cmds = NULL)]
```

```
# Convert strings to factors
categorical_vars <- categorical_vars[!(categorical_vars %in%
                                 c("is_host_login", "num_outbound_cmds"))]
train[, (categorical_vars) := lapply(.SD, as.factor), .SDcols = categorical_vars]
test[, (categorical_vars) := lapply(.SD, as.factor), .SDcols = categorical_vars]
```

## Evaluation Metric

Since this is a multi-class classification problem, weighted F1 score is chosen as the analysis metric.

F1 score is the harmonic mean of precision and recall.

$$F1 = \frac{2}{recall^{-1} + precision^{-1}} = \frac{2.precision.recall}{precision + recall}$$

Weighted F1 score is the average of F1 scores over all classes of target variable, weighed proportional to the frequency of occurrence of target class (aka “support”).

$$Weighted\ F1 = \frac{1}{n} \sum_{i=1}^{m} Support_i.F1_i$$

where $n$ is the total number of examples in the test dataset, and $m$ is the number of classes in the target variable. $Support_i$ is the number of times class $i$ occurs in the test dataset.

For convenience, weighted F1 score is expressed as a percentage in this study.

```
weighted_F1_score <- function(predicted) {
  cm <- confusionMatrix(predicted, test$target)
  F1 <- cm$byClass[, "F1"]
  weights <- sapply(sub("Class: ", "", names(F1)), function(x) {
    test[, .N, by=target][as.character(target) == x, N]
  })
  weights <- weights / nrow(test)
  round((F1 %*% weights) * 100, digits = 1)
}
```

## Linear Discriminant Analysis (LDA)

LDA is a natural choice for multi-class classification. Unlike logistic regression which does binary classification by directly modelling conditional distribution of response variable, LDA takes an alternate approach based on Bayes’ theorem. Here the distribution of the predictors is modeled (instead of distribution of response variable) for each response class, and then flipped around using Bayes’ theorem (to get conditional probability of response variable). This means predictions for multiple classes of response variable is obtained in one step.

Before applying LDA,

- redundant continuous variables are eliminated using dimension reduction
- nominal variables are converted into dummy variables and redundancy is also eliminated among dummy variables.

**Feature Selection using PCA**

*Dimensionality reduction* is a process of reducing the dimensions of a matrix while still preserving a good amount of information. *Principal component analysis (PCA)* is a popular technique used in dimensionality reduction. The idea is as follows: We think of rows of a matrix as vectors representing points in Euclidean space. So, a (m x n) matrix will have m points in a n-dimensional space. We can "rotate" the axes of this space in a such a way that the first axis ("x" axis) is oriented along the direction that yields the maximum variance of values of the coordinates of original points. Similarly, second axis (chosen to be orthogonal to the first) is in a plane that yields second highest variance, and so on. If this process is repeated, we will likely hit a plateau where subsequent axes capture only a small amount of variance ("information"). We can drop these less significant axes, thereby reducing the dimensions of our matrix (and size of our dataset).

PCA is ideally suited for removing redundancy from large datasets like KDD CUP 99. Data is normalized (mean of 0 and variance of 1) before principal components are computed. PCA can only be applied to numeric variables.

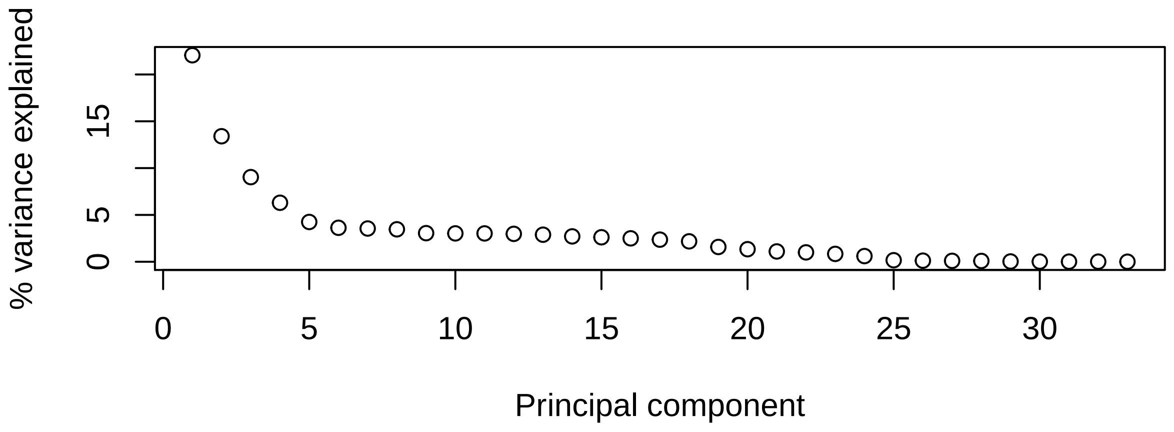
```r
# Select only numeric variables (PCA computes euclidean distances)
numeric_vars <- setdiff(names(train), categorical_vars)
train.numeric <- train[, .SD, .SDcols = numeric_vars]

# Scale variables to unit variance and center variables (mean = 0)
train.scaled <- scale(train.numeric)

# Compute principal components
pca <- prcomp(train.scaled)
```

Plot percentage variance explained by each principal component. Principle components represent transformation of original data, where coordinates of points refer to the rotated axes. Direction of new axes are along the eigenvectors and thus principle components (matrix product of original inputs with eigenvectors) represent new 'variables' in the rotated axes.

```r
pca.var <- pca$sdev^2
pca.var <- pca.var / sum(pca.var) * 100
plot(x = seq(length(pca.var)), y = pca.var, xlab = "Principal component",
     ylab = "% variance explained")
```



The bottom 13 principal components explain only 4.07 % of total variance. These can be dropped (dimensional reduction).

```
# Compute the scores (the "new" variables, i.e. the PCs' scores)
train.pca_scores = train.scaled %*% pca$rotation

# Drop the last 13 variables (PCs)
train.reduced <- data.table(train.pca_scores[, 1:20])

# Reduce dimensions of the test dataset
test.numeric <- test[, .SD, .SDcols = numeric_vars]
test.scaled <- scale(test.numeric)
test.pca_scores = test.scaled %*% pca$rotation
test.reduced <- data.table(test.pca_scores[, 1:20])
```

Before considering nominal variables, benchmark LDA's classification performance with only numeric variables included in the model.

```
lda.fit <- lda(target ~ ., data = cbind(train.reduced, target = train$target))
lda.predict <- predict(lda.fit, cbind(test.reduced, target = test$target))
lda_F1_numeric_only <- weighted_F1_score(lda.predict$class)
lda_F1_numeric_only
```

```
##      [,1]
## [1,] 39.4
```

Weighted F1 Score for model with only numeric variables is 39.4.

### Dummy variables

Nominal variable `service` has very high cardinality (66 classes). Encoding 65 dummy variables would result in a wide and sparce input matrix. For pragmatic reasons this variable is dropped.

Following table shows total number of classes in each nominal variable.

```
train.cat_vars <- train[, .SD, .SDcols = categorical_vars]
test.cat_vars <- test[, .SD, .SDcols = categorical_vars]
train.cat_vars[, lapply(.SD, function(x) length(unique(x))),
                       .SDcols = !c("target")]
```

```
##    protocol_type service flag land logged_in is_guest_login
## 1:             3      66   11    2         2              2
```

```
train.cat_vars[, service := NULL]
test.cat_vars[, service := NULL]
```

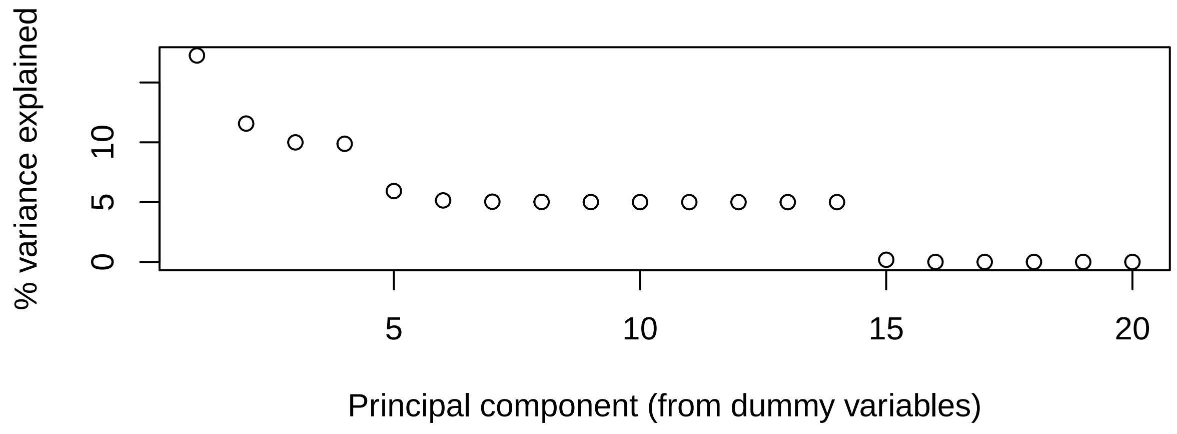
Convert nominal variables to dummy variables.

```
# Set contrasts.arg to keep all levels
contrasts_arg <- list(
  protocol_type = contrasts(train.cat_vars$protocol_type, contrasts = F),
  flag = contrasts(train.cat_vars$flag, contrasts = F),
  land = contrasts(train.cat_vars$land, contrasts = F),
  logged_in = contrasts(train.cat_vars$logged_in, contrasts = F),
  is_guest_login = contrasts(train.cat_vars$is_guest_login, contrasts = F))

# Encode dummy variables
dummies_tr <- model.matrix(target ~ ., train.cat_vars,
                        contrasts.arg = contrasts_arg)
train.dummy_matrix <- dummies_tr[, c(2:ncol(dummies_tr))]
```

Introducing a large number of dummy variables results in *collinearlity* since many of them are correlated. To improve model performance, we extract features (again) using dimensional reduction (PCA) on dummy variables.

```
# Compute principal components on dummy variables after scaling
train.dm_scaled <- scale(train.dummy_matrix)
# Calculate principal components
pca_dummy <- prcomp(train.dm_scaled)
# Plot % of variance explained by each principal component
pca_dummy.var <- pca_dummy$sdev^2
pca_dummy.var <- pca_dummy.var / sum(pca_dummy.var) * 100
plot(x = seq(length(pca_dummy.var)), y = pca_dummy.var,
     xlab = "Principal component (from dummy variables)",
     ylab = "% variance explained")
```



Drop the last 6 principal components since they together explain only 0.1794959 % of total variance.

```
# Compute the scores (the "new" variables, i.e. the PCs' scores)
train.pca_dummy_scores = train.dm_scaled %*% pca_dummy$rotation
colnames(train.pca_dummy_scores) <-
  sapply(colnames(train.pca_dummy_scores), function(x) sub("$", "_dummy", x))
```

```r
# Drop the last 6 variables (PCs)
train.reduced <- cbind(train.reduced, train.pca_dummy_scores[, 1:14])
train.reduced[, target := train$target]

# repeat for test dataset (encode dummy vars, drop least significant PCs)
dummies_tt <- model.matrix(target ~ ., test.cat_vars,
                           contrasts.arg = contrasts_arg)
test.dummy_matrix <- dummies_tt[, c(2:ncol(dummies_tt))]
test.dm_scaled <- scale(test.dummy_matrix)

# Compute the scores (the "new" variables), i.e. the PCs' scores using
#    rotation axes from training set)
test.pca_dummy_scores = test.dm_scaled %*% pca_dummy$rotation
colnames(test.pca_dummy_scores) <-
  sapply(colnames(test.pca_dummy_scores), function(x) sub("$", "_dummy", x))
# Drop the last 6 variables (PCs)
test.reduced <- cbind(test.reduced, test.pca_dummy_scores[, 1:14])
test.reduced[, target := test$target]
```

### Training LDA

We fit LDA model on the full training set (with PCs from both continuous and dummy variables). Confusion matrix is as shown below.

```r
lda.fit <- lda(target ~ ., data = train.reduced)
lda.pred <- predict(lda.fit, test.reduced)
cm <- confusionMatrix(lda.pred$class, test$target)
cm$table
```

```
##             Reference
## Prediction     DoS normal  Probe    R2L    U2R
##     DoS      58629    113    242    110      1
##     normal  169462  59792    367  15440     22
##     Probe      721    496   3557    123      2
##     R2L       1043    169      0    663     17
##     U2R          0     23      0      9     28
```

Weighted F1 score (below) shows no improvement over a LDA model fitted on only PCAs from numeric variables. This may be expected. Discriminant analysis assumes a normal distribution of dependent variables. When categorical variables are encoded as integer values of 0 and 1 (values of dummy variables) the resulting distribution is neither normal nor multivariate, and therefore LDA handles them poorly.

```r
lda_wF1_full <- weighted_F1_score(lda.pred$class)
data.table(Model = c("LDA: Only numeric variables", "LDA: All variables"),
           "Weighted F1 Score"
           = c(lda_F1_numeric_only[1, 1], lda_wF1_full[1, 1]))
```

```
##                          Model Weighted F1 Score
## 1: LDA: Only numeric variables              39.4
## 2:          LDA: All variables              39.1
```

## Random Forests

Decision trees can be constructed by recursively dividing (binary splitting) the predictor space into distinct and non-overlapping regions until a termination criteria is reached. For every observation that falls into a region (leaf), we make the same prediction, which (for classification problems) is simply the *most commonly occurring class* of training observations in the region. For classification trees, the criteria for binary splits is the *Gini index* given by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

a measure of total variance across the $K$ classes. Here $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class ($0 \leq \hat{p}_{mk} \leq 1$). It is not hard to see that the *Gini index* takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the *Gini index* is referred to as a measure of node purity -— a small value indicates that a node contains predominantly observations from a single class.

Decision trees can be constructed both for continuous and qualitative variables. For splitting using a continuous variable, all observations below a cut-point is assigned to one branch of the split and the rest to another branch. Splitting on qualitative variable amounts to assigning some of the qualitative values to one branch and assigning the remaining to the other branch. Therefore, trees can easily handle qualitative predictors *without the need to create dummy variables.*

Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches. However, by aggregating many decision trees, using methods like *bagging*, *random forests*, and *boosting*, the predictive performance of trees can be substantially improved.

*Random Forests* builds a number of decision trees on bootstrapped training samples. The trees are kept *decorrelated* by choosing a suitable predictor to perform the split. Each time a split in a tree is considered, a random sample of $m$ predictors is chosen as split candidates from the full set of $p$ predictors. The split is allowed to use only one of those m predictors (typically $m \approx \sqrt{p}$). If we did not use random sampling we would be splitting using the very strong predictor in the data set, followed by a number of other moderately strong predictors. Then all of the bagged trees will look quite similar to each other. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance. We avoid this problem by forcing each split to consider only a subset of the randomly chosen predictors.

One interesting aspect is that the number of trees is not a critical parameter with *Random Forests.* Using large number of trees will not lead to overfitting.

### Feature Selection using Gini index

In a collection of bootstrapped classification trees we can obtain the importance of a variable using *Gini index.* We can add the total amount that the *Gini index* is decreased by splits over a given predictor, averaged over all trees. Mean decrease in *Gini index* for each variable (relative to the largest) represents its *importance.* We can remove redundant variables after ranking all variables in the order of their *importance.*

After ranking variables based on *mean reduction Gini*, *11 variables are removed.*

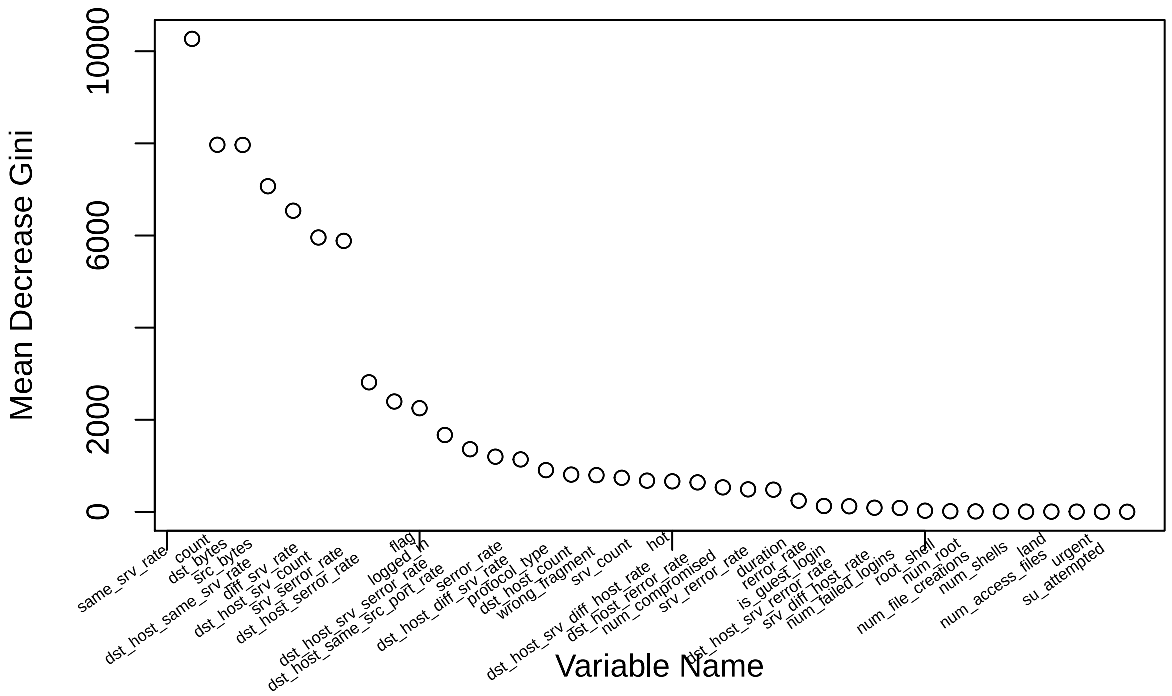
Aside: R's Random Forests implementation does not handle more than 53 levels (classes) in a nominal variable. One of the categorical variable (*"service"*) has 66 classes. We get an error as follows.

Can not handle categorical predictors with more than 53 categories.

The variable *"service"* is removed from training and test datasets.

```r
rf.fit <- randomForest(target ~ ., data = train[, .SD, .SDcols = !c("service")],
                       ntree = 50, importance = TRUE)
```

```r
gini <- sort(rf.fit$importance[, "MeanDecreaseGini"], decreasing = TRUE)
plot(x = seq(length(gini)), y = gini, xlab = "Variable Name",
     ylab = "Mean Decrease Gini", xaxt = "n")
axis(side = 1, labels = FALSE)
text(x = 1:length(gini),
     y = par("usr")[3] - 0.35,
     labels = names(gini),
     xpd = NA,
     srt = 35,
     adj = 1.3,
     cex = 0.5)
```



```r
removed_vars <- c(names(gini)[(length(gini)-10):length(gini)], "service")
```

### Training Random Forests

Number of trees is chosen to be 100, and it gives a reasonably good result. More trees will increase the weighted F1 score at the expense of time needed to compute.

Both confusion matrix and weighted F1 score shows marked improvement over LDA.

```r
rf.fit <- randomForest(target ~ .,
                       data = train[, .SD, .SDcols = !removed_vars],
                       ntree = 100)
rf.predicted = predict(rf.fit,
                       newdata = test[, .SD, .SDcols = !removed_vars])
cm <- confusionMatrix(rf.predicted, test$target)
rf_wf1 <- weighted_F1_score(rf.predicted)
cm$table
```

```
##             Reference
## Prediction     DoS normal  Probe    R2L    U2R
##     DoS     223768     67    207      0      0
##     normal    6051  60298    664  15954     60
##     Probe       36    226   3295     21      0
##     R2L          0      1      0    369      4
##     U2R          0      1      0      1      6
```

## Results

*Random Forests* outperforms *LDA* by a wide margin. Presence of high cardinality categorical variables does not adversely affect the performance of *Random Forests*. However, it takes longer to train compared to *LDA*. *LDA* fails to take advantage of categorical variables. It suffers from being unable to deal with the predominance of outliers in the distribution of dummy variables.

```r
data.table("Training Algorithm" = c("LDA", "Random Forests"),
           "Weighted F1 Score" = c(lda_wF1_full, rf_wf1))
```

```
##    Training Algorithm Weighted F1 Score
## 1:                LDA              39.1
## 2:     Random Forests              90.6
```

## References

- KDD CUP 99: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
- *An Introduction to Statistical Learning*, G. James, R. Tibshirani, et. al.