IoT attacks detection

Yulia Zamyatins

April, 2021

Contents

Introduction

The **detection_of_IoT_botnet_attacks_N_BaIoT** data set[1] contains a traffic data from 9 commercial IoT devices authentically infected by Mirai and BASHLITE (gafgyt).

```
## 'data.frame':    1018298 obs. of  116 variables:
##  $ botnet               : chr  "benign" "benign" "benign" "benign" ...
##  $ MI_dir_L5_weight     : num  1 1 1.86 1 1.68 ...
##  $ MI_dir_L5_mean       : num  60 354 360 337 172 ...
##  $ MI_dir_L5_variance   : num  0 0 35.8 0 18487.4 ...
##  $ MI_dir_L3_weight     : num  1 1 1.91 1 1.79 ...
##  $ MI_dir_L3_mean       : num  60 354 360 337 183 ...
##  $ MI_dir_L3_variance   : num  0 0 35.9 0 18928.2 ...
##  $ MI_dir_L1_weight     : num  1 1 1.97 1 1.93 ...
##  $ MI_dir_L1_mean       : num  60 354 360 337 193 ...
##  $ MI_dir_L1_variance   : num  0 0 36 0 19154 ...
##  $ MI_dir_L0.1_weight   : num  1 1 2 1 1.99 ...
##  $ MI_dir_L0.1_mean     : num  60 354 360 337 198 ...
```

[1] detection_of_IoT_botnet_attacks_N_BaIoT Data Set

```
##  $ MI_dir_L0.1_variance : num  0 0 36 0 19182 ...
##  $ MI_dir_L0.01_weight  : num  1 1 2 1 2 ...
##  $ MI_dir_L0.01_mean    : num  60 354 360 337 198 ...
##  $ MI_dir_L0.01_variance: num  0 0 36 0 19182 ...
##  $ H_L5_weight          : num  1 1 1.86 1 1.68 ...
##  $ H_L5_mean            : num  60 354 360 337 172 ...
##  $ H_L5_variance        : num  0.00 4.59e-06 3.58e+01 0.00 1.85e+04 ...
##  $ H_L3_weight          : num  1 1 1.91 1 1.79 ...
##  $ H_L3_mean            : num  60 354 360 337 183 ...
##  $ H_L3_variance        : num  0.00 4.58e-03 3.59e+01 0.00 1.89e+04 ...
##  $ H_L1_weight          : num  1 1.03 2 1 1.93 ...
##  $ H_L1_mean            : num  60 354 360 337 193 ...
##  $ H_L1_variance        : num  0 4.3 40.4 0 19153.8 ...
##  $ H_L0.1_weight        : num  1 2.6 3.59 1 1.99 ...
##  $ H_L0.1_mean          : num  60 347 352 337 198 ...
##  $ H_L0.1_variance      : num  0 34.1 100.1 0 19182 ...
##  $ H_L0.01_weight       : num  1 5.32 6.32 1 2 ...
##  $ H_L0.01_mean         : num  60 344 348 337 198 ...
##  $ H_L0.01_variance     : num  0 22.2 81.6 0 19182.2 ...
##  $ HH_L5_weight         : num  1 1 1.86 1 1 ...
##  $ HH_L5_mean           : num  60 354 360 337 60 ...
##  $ HH_L5_std            : num  0 0.00214 5.98242 0 0 ...
##  $ HH_L5_magnitude      : num  60 354 360 337 524 ...
##  $ HH_L5_radius         : num  0.00 4.59e-06 3.58e+01 0.00 3.18e+04 ...
##  $ HH_L5_covariance     : num  0 0 0 0 0 ...
##  $ HH_L5_pcc            : num  0 0 0 0 0 ...
##  $ HH_L3_weight         : num  1 1 1.91 1 1 ...
##  $ HH_L3_mean           : num  60 354 360 337 60 ...
##  $ HH_L3_std            : num  0 0.0676 5.994 0 0 ...
##  $ HH_L3_magnitude      : num  60 354 360 337 483 ...
##  $ HH_L3_radius         : num  0.00 4.58e-03 3.59e+01 0.00 4.64e+04 ...
##  $ HH_L3_covariance     : num  0 0 0 0 0 ...
##  $ HH_L3_pcc            : num  0 0 0 0 0 ...
##  $ HH_L1_weight         : num  1 1.03 2 1 1 ...
##  $ HH_L1_mean           : num  60 354 360 337 60 ...
##  $ HH_L1_std            : num  0 2.07 6.36 0 0 ...
##  $ HH_L1_magnitude      : num  60 354 360 337 438 ...
##  $ HH_L1_radius         : num  0 4.3 40.4 0 58393.6 ...
##  $ HH_L1_covariance     : num  0 0 0 0 0 ...
##  $ HH_L1_pcc            : num  0 0 0 0 0 ...
##  $ HH_L0.1_weight       : num  1 2.6 3.59 1 1 ...
##  $ HH_L0.1_mean         : num  60 347 352 337 60 ...
##  $ HH_L0.1_std          : num  0 5.84 10 0 0 ...
##  $ HH_L0.1_magnitude    : num  60 347 352 337 420 ...
##  $ HH_L0.1_radius       : num  0 34.1 100.1 0 62080.6 ...
##  $ HH_L0.1_covariance   : num  0 0 0 0 0 ...
##  $ HH_L0.1_pcc          : num  0 0 0 0 0 ...
##  $ HH_L0.01_weight      : num  1 5.32 6.32 1 1 ...
##  $ HH_L0.01_mean        : num  60 344 348 337 60 ...
##  $ HH_L0.01_std         : num  0 4.71 9.03 0 0 ...
##  $ HH_L0.01_magnitude   : num  60 344 348 337 418 ...
##  $ HH_L0.01_radius      : num  0 22.2 81.6 0 62388.6 ...
##  $ HH_L0.01_covariance  : num  0 0 0 0 0 ...
##  $ HH_L0.01_pcc         : num  0 0 0 0 0 ...
```

```
##  $ HH_jit_L5_weight     : num  1 1 1.86 1 1 ...
##  $ HH_jit_L5_mean       : num  1.51e+09 4.98 2.32 1.51e+09 1.51e+09 ...
##  $ HH_jit_L5_variance   : num  0.00 4.23e-07 6.06 0.00 0.00 ...
##  $ HH_jit_L3_weight     : num  1 1 1.91 1 1 ...
##  $ HH_jit_L3_mean       : num  1.51e+09 4.98 2.40 1.51e+09 1.51e+09 ...
##  $ HH_jit_L3_variance   : num  0 0.000422 6.079503 0 0 ...
##  $ HH_jit_L1_weight     : num  1 1.03 2 1 1 ...
##  $ HH_jit_L1_mean       : num  1.51e+09 5.09 2.57 1.51e+09 1.51e+09 ...
##  $ HH_jit_L1_variance   : num  0 0.418 6.581 0 0 ...
##  $ HH_jit_L0.1_weight   : num  1 2.6 3.59 1 1 ...
##  $ HH_jit_L0.1_mean     : num  1.51e+09 3.12e+01 2.26e+01 1.51e+09 1.51e+09 ...
##  $ HH_jit_L0.1_variance : num  0 6976 5228 0 0 ...
##  $ HH_jit_L0.01_weight  : num  1 5.32 6.32 1 1 ...
##  $ HH_jit_L0.01_mean    : num  1.51e+09 5.80e+01 4.88e+01 1.51e+09 1.51e+09 ...
##  $ HH_jit_L0.01_variance: num  0 12741 11172 0 0 ...
##  $ HpHp_L5_weight       : num  1 1 1.86 1 1 ...
##  $ HpHp_L5_mean         : num  60 354 360 337 60 ...
##  $ HpHp_L5_std          : num  0 0.00214 5.98242 0 0 ...
##  $ HpHp_L5_magnitude    : num  60 354 360 337 60 ...
##  $ HpHp_L5_radius       : num  0.00 4.59e-06 3.58e+01 0.00 0.00 ...
##  $ HpHp_L5_covariance   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HpHp_L5_pcc          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HpHp_L3_weight       : num  1 1 1.91 1 1 ...
##  $ HpHp_L3_mean         : num  60 354 360 337 60 ...
##  $ HpHp_L3_std          : num  0 0.0676 5.994 0 0 ...
##  $ HpHp_L3_magnitude    : num  60 354 360 337 60 ...
##  $ HpHp_L3_radius       : num  0 0.00458 35.92849 0 0 ...
##  $ HpHp_L3_covariance   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HpHp_L3_pcc          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ HpHp_L1_weight       : num  1 1.03 2 1 1 ...
##  $ HpHp_L1_mean         : num  60 354 360 337 60 ...
##  $ HpHp_L1_std          : num  0 2.07 6.36 0 0 ...
##  $ HpHp_L1_magnitude    : num  60 354 360 337 60 ...
##   [list output truncated]
```

The data set has 115 attributes (parameters), below is the description of their headers:

1. It has 5 time-frames: L5, L3, L1, L0.1 and L0.01.
2. The statistics extracted from each stream for each time-frame:
   - *weight*: the weight of the stream (can be viewed as the number of items observed in recent history)
   - *mean*
   - *std (variance)*
   - *radius*: the root squared sum of the two streams' variances
   - *magnitude*: the root squared sum of the two streams' means
   - *covariance*: an approximated covariance between two streams
   - *pcc*: an approximated correlation coefficient between two streams
3. It has following stream aggregations:
   - *MI*: ("Source MAC-IP" in N-BaIoT paper) Stats summarizing the recent traffic from this packet's host (IP + MAC)
   - *H*: ("Source IP" in N-BaIoT paper) Stats summarizing the recent traffic from this packet's host (IP)
   - *HH*: ("Channel" in N-BaIoT paper) Stats summarizing the recent traffic going from this packet's host (IP) to the packet's destination host.

- *HH_jit*: ("Channel jitter" in N-BaIoT paper) Stats summarizing the jitter of the traffic going from this packet's host (IP) to the packet's destination host.
- *HpHp*: ("Socket" in N-BaIoT paper) Stats summarizing the recent traffic going from this packet's host+port (IP) to the packet's destination host+port. Example 192.168.4.2:1242 -> 192.168.4.12:80

Thus, the column *'MI_dir_L5_weight'* in the data set shows the weight of the recent traffic from the packet's host for L5 time-frame.

I've added extra *'botnet'* column, where I keep information about the attacks from the different botnets and benign traffic. I've used "ga_" prefix for gafgyt attacks, and "ma_" prefix for Mirai attacks.

The team that collected this data set used 2/3 of their benign traffic (their train set) to train their deep autoencoder. Then they used remaining 1/3 of benign traffic and all the malicious data (their test set) to detect anomalies with deep autoencoder. The detection of the cyberattacks launched from each of the above IoT devices concluded with 100% TPR.

In HarvardX PH125.9x Data Science course we learned several algorithms that can be used for the classification, such as **KNN**, **rpart** or **randomForest**.

My aim in this project is to check how well all these algorithms can detect anomalies in the data set, how accurate they can perform the classification.

The source data set consists of *.csv and *.rar files, each representing the benign traffic or the attack, that are divided into folders with devices names. Because R has no package that can unpack *.rar files for its own, so all *.rar archives have to be manually unpacked with command line or third-party applications, depending on OS. Therefor, I had to prepare the data set for this project that can be easily downloaded and unpacked.

But the whole data set size was more that 1TB. Nowadays it's not a problem to find a hosting to share this huge data set, but I thought that everyone who will check this project won't be happy to download it. Because the team, that collected this data set, trained and tested their autoencoder for each device separately, I decided that I can use the data only from one device for my project. I've used the data only from Danmini doorbell device, but the data for other devices can be downloaded from the source and prepared for the classification using download-data.R script.

The data set for Danmini doorbel has the size of 970MB in *.csv file and only 200MB in *.zip archive. Because the archive file size exceeds GitHub limits of file size that can be stored with it, I used the third-party file hosting to share it.

The archive file is downloading during the first time of using project's scripts and saving as *.csv file in local project data folder. For the next time, saved *.csv files is used.

Dataset analisys

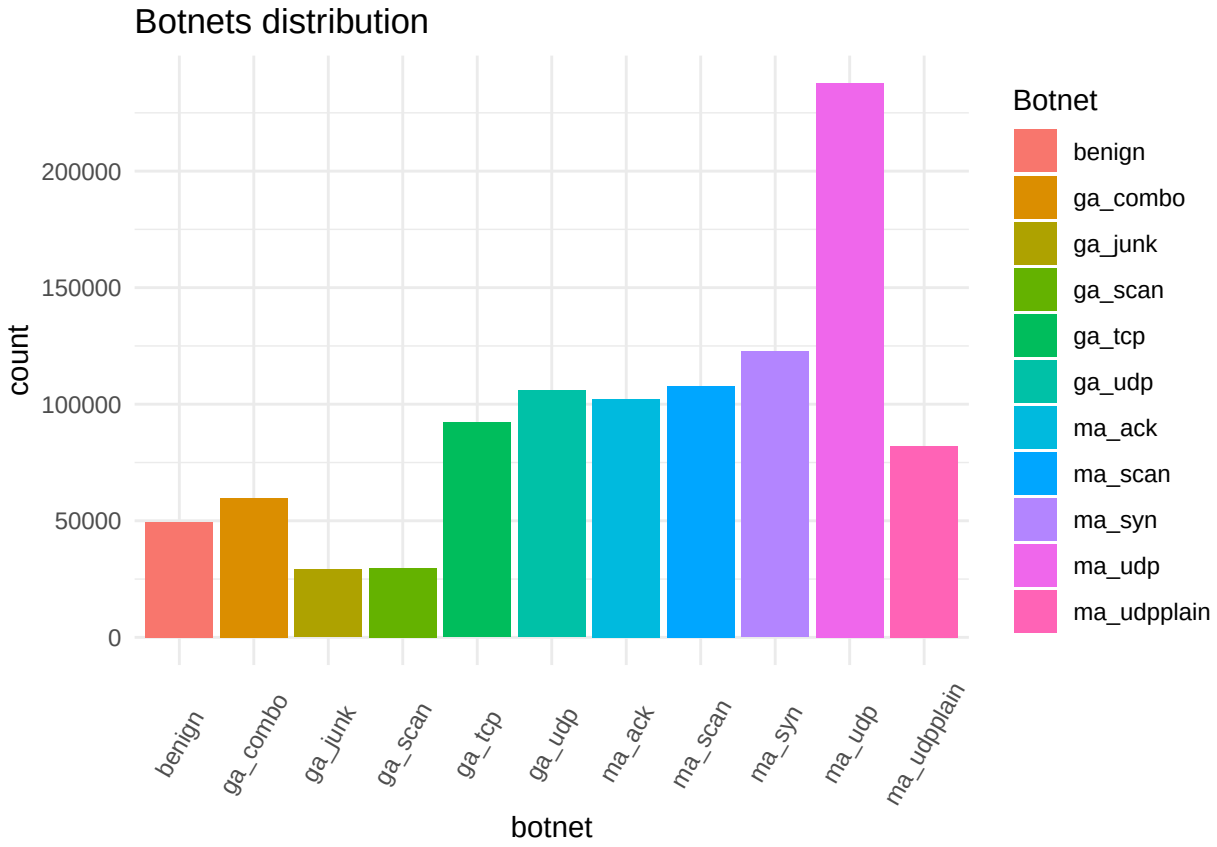
```
# Data frame dimension
dim( df )
```

```
## [1] 1018298     116
```

The data set for Danmini doorbell device consists of more than 1 million rows. It has 115 predictors, and the 'botnet' column contains the outcome for the classification.

```
## # A tibble: 11 x 3
##    botnet           n   prop
##    <fct>        <int>  <dbl>
##  1 ma_udp      237665 0.233
##  2 ma_syn      122573 0.120
##  3 ma_scan     107685 0.106
##  4 ga_udp      105874 0.104
##  5 ma_ack      102195 0.100
##  6 ga_tcp       92141 0.0905
##  7 ma_udpplain  81982 0.0805
##  8 ga_combo     59718 0.0586
##  9 benign       49548 0.0487
## 10 ga_scan      29849 0.0293
## 11 ga_junk      29068 0.0285
```

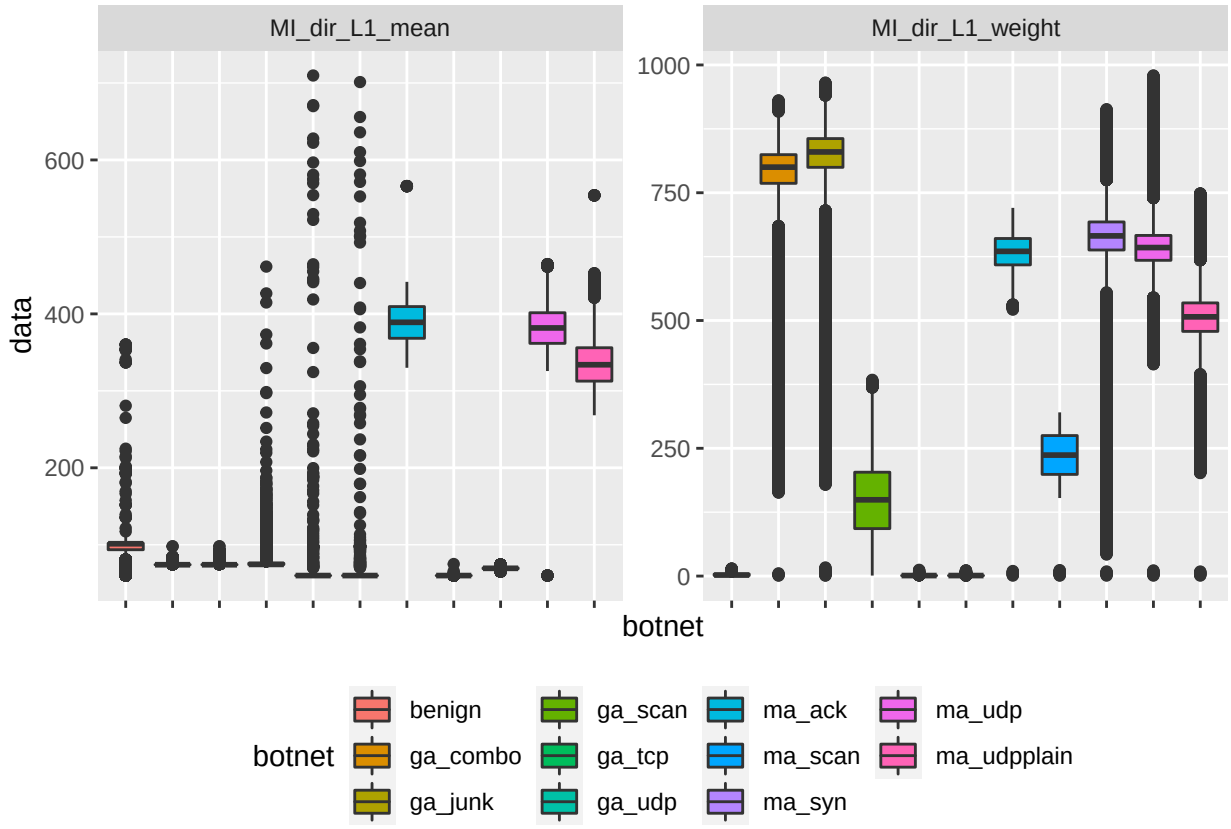
There are 11 botnets in the data set (10 attacks plus benign traffic) and benign traffic is only 4.87% of all traffic.



Because the whole data set has 115 predictors, the full data exploration will take a lot of pages. So I created data_exploration.pdf with full data exploration. The data_exploration.R script and data_exploration.Rmd Rmarkdown files also can be found on GitHub.

Here I just mention my conclusion from this exploration.

My research has shown that I can use only *weight*, *mean* and *covariance* columns for the classification if I need to reduce the data set dimension. This is clearly visible on small time frames such as L1.



The plot above for *MI_dir_L1_weight* column shows that almost all types of the attacks, excluding *ga_tcp* and *ga_udp*, can be separated from benign traffic, as their IQR and medians have higher values that benign traffic has. *ga_tcp* and *ga_udp* attacks camouflaging as benign traffic very well on this plot.

But the plot for *MI_dir_L1_mean* column shows that IQR and median is higher for benign traffic comparing with these parameters for *ga_tcp* and *ga_udp* attacks. So this column data will help to separate *ga_tcp* and *ga_udp* attacks from the benign traffic.

Thus, as I wrote above, *weight* and *mean* columns can be used for the attacks classification. Because all the attacks has outliers, the information these columns contain is not enough, so *covariance* column can be used also for the classification.

Once again, the full data exploration can be found on GitHub:

- data_exploration.pdf
- data_exploration.R script
- data_exploration.Rmd Rmarkdown

Methods

As I mention above, we have learned several methods for the data classification in HarvardX PH125.9x Data Science course, such as **KNN**, **rpart** and **randomForest**.

rpart and **randomForest** are easy to use algorithms because all need to do is only to separate the data frame into two parts (train and test set) and allow function to do its work. That's why I decided also to check how well **KNN** method can perform the attack classification. I used two approaches for **KNN**: PCA and *train()* function from *caret* package.

The team, that collected this data frame, used 2/3 of the benign traffic to train deep autoencoder. I can't use this approach, because all learned algorithms should know all possible outcome after training. Thus, I simply divide the data set into two equal parts (train and test sets). Anyway, my aim is only to check how accurate these algorithms are in the classification.

I have

```
## [1] '4.0.2'
```

R version, and it has

```
## [1] 8060
```

MB memory limit.

Instead of increasing memory limit, I decided to use only part of data set for some algorithms.

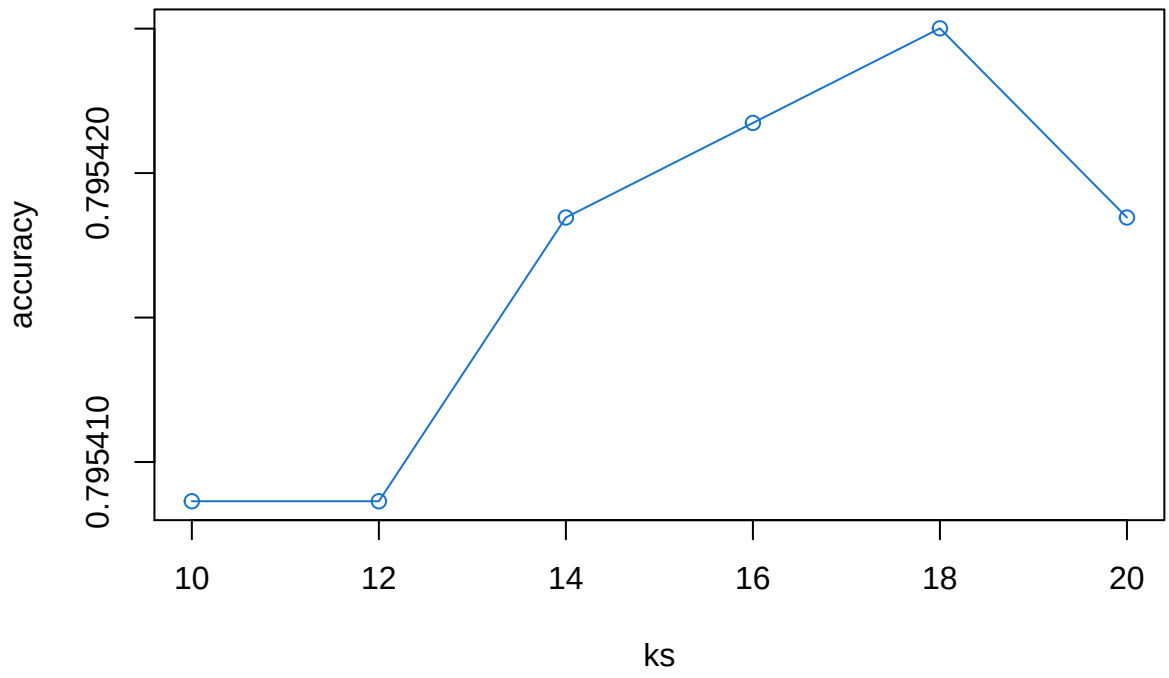
KNN (with PCA)

For this method, I select only columns that contain *weight* parameter, then separate data set into train and test sets, make them equal dimension and convert to matrices. After that I perform PCA using train set.

```
## Importance of components:
##                             PC1       PC2       PC3       PC4       PC5
## Standard deviation     3.391e+04 1.424e+04 3.616e+03 2672.4956 784.44166
## Proportion of Variance 8.371e-01 1.477e-01 9.520e-03    0.0052   0.00045
## Cumulative Proportion  8.371e-01 9.848e-01 9.943e-01    0.9995   0.99996
##                             PC6       PC7   PC8   PC9  PC10  PC11  PC12 PC13
## Standard deviation     170.66268 138.31731 54.05 23.82 12.31 5.136 3.939 1.92
## Proportion of Variance   0.00002   0.00001  0.00  0.00  0.00 0.000 0.000 0.00
## Cumulative Proportion    0.99998   1.00000  1.00  1.00  1.00 1.000 1.000 1.00
##                           PC14   PC15     PC16      PC17      PC18      PC19
## Standard deviation     1.74 0.2351 0.005893 8.494e-09 2.039e-10 7.127e-11
## Proportion of Variance 0.00 0.0000 0.000000 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.00 1.0000 1.000000 1.000e+00 1.000e+00 1.000e+00
##                             PC20      PC21      PC22      PC23      PC24
## Standard deviation     1.419e-11 1.356e-11 1.331e-11 1.107e-11 4.819e-12
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC25
## Standard deviation     8.447e-14
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

While PCA shows that first 6 principal components explain 99,998% of the variability, I decided to perform cross-validation and check how many PCA matrix dimensions give maximum accuracy in the classification.

Because the algorithm is slow and my research have shown that 6 dimensions is not enough, I use only dimensions from 10 to 20 only (with step = 2).




```
## [1] "K =  18"
```

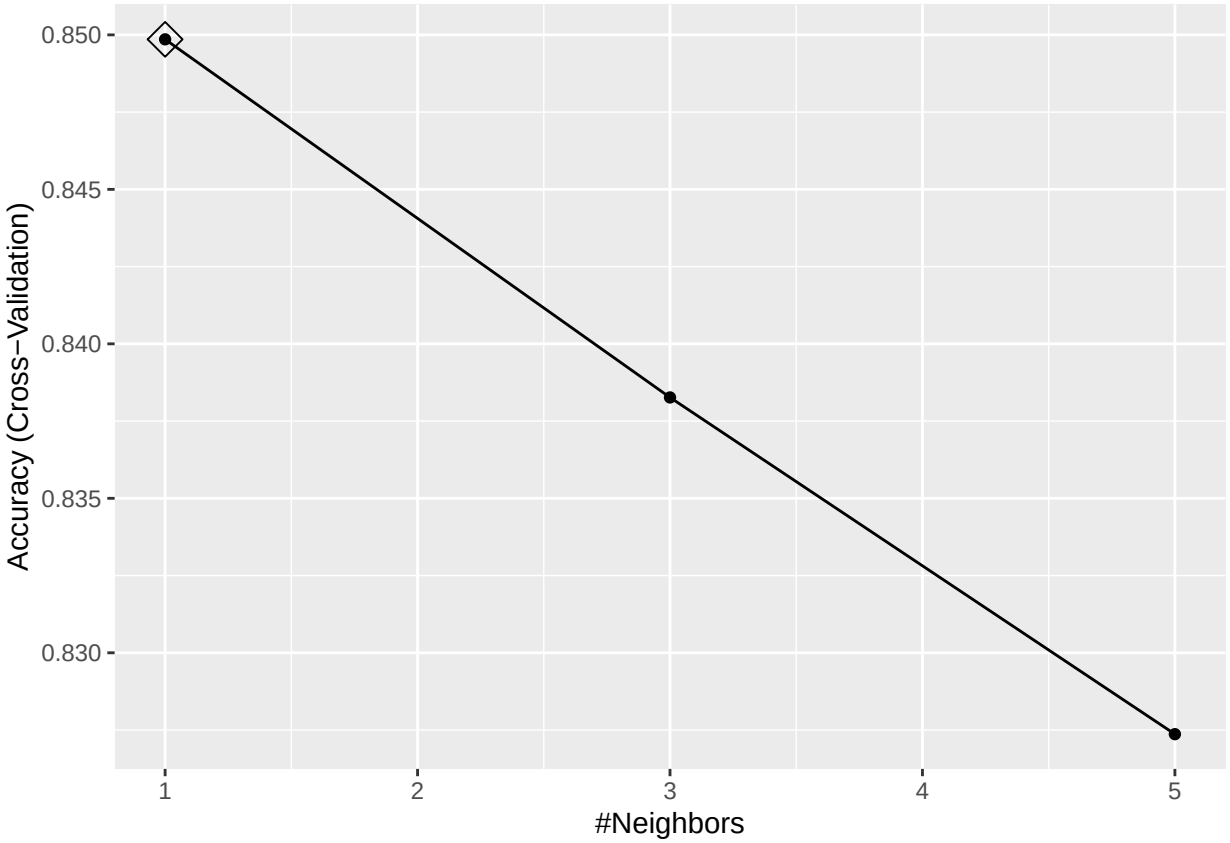
```
## [1] "Max accuracy =  0.795425009902222"
```

This method doesn't give significant result as its classification accuracy is only 79.54%. From other hand, I used only one *weight* column as a predictor.

I decided do not perform classification using *weight*, *mean* and *covariance* columns because of memory limit and because its really slow. Thus, I decided to use *KNN* method but using *train()* function from *caret* package.

KNN (with train() function)

For this method, at first, I decided to use all 115 columns as predictors, but because of memory limit, I can use only 2% of data set. I also use 10-fold cross-validation to find better parameters for the classification.



Perform classification and calculate its accuracy:

```
##  Accuracy
## 0.8578581
```

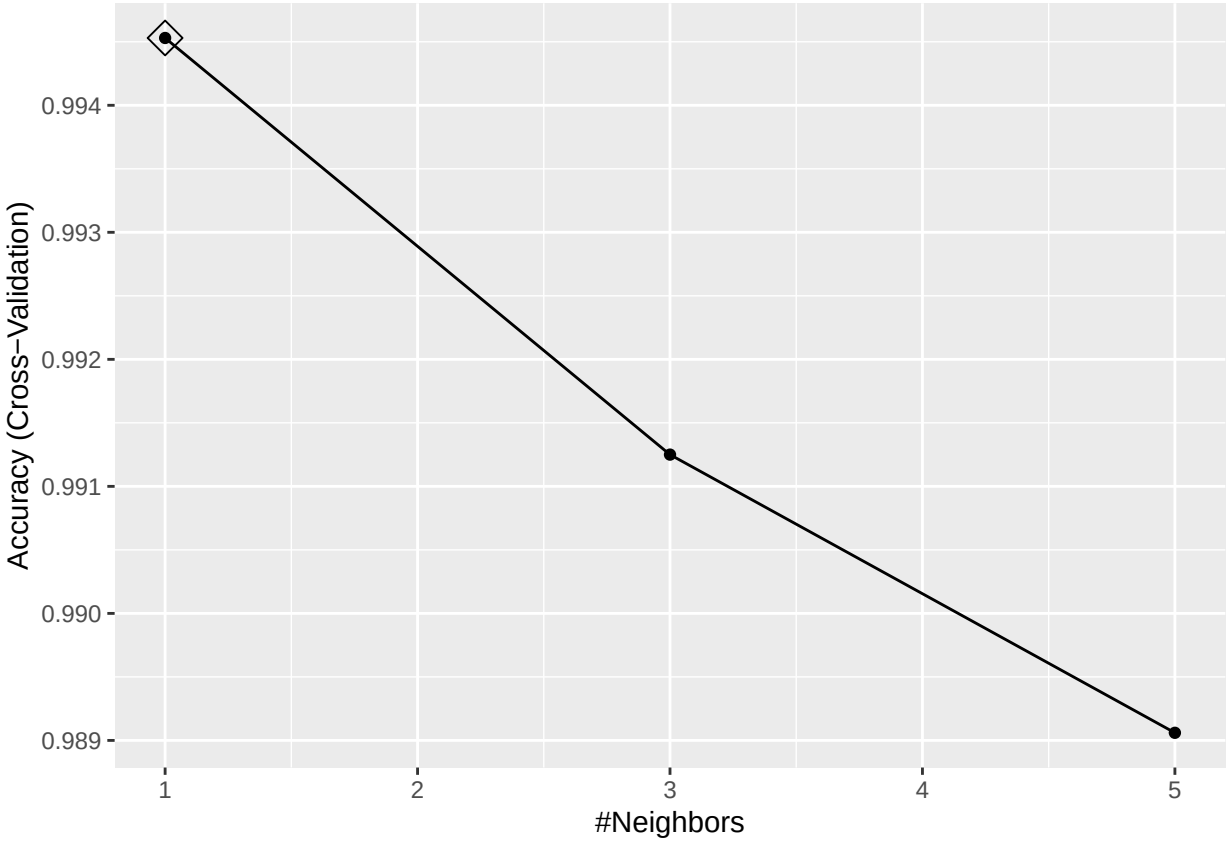
I suppose, because I used more predictors than in the first method, the final result is better.

```
##                     Sensitivity Specificity
## Class: benign         0.9697581   0.9981426
## Class: ga_combo       0.9431438   0.9970800
## Class: ga_junk        0.9037801   0.9971706
## Class: ga_scan        0.9832776   0.9996966
## Class: ga_tcp         0.9989154   1.0000000
## Class: ga_udp         1.0000000   0.9998904
## Class: ma_ack         0.6369863   0.9567921
## Class: ma_scan        1.0000000   0.9998902
## Class: ma_syn         0.6843393   0.9587100
## Class: ma_udp         0.8081615   0.9427657
## Class: ma_udpplain    0.8158537   0.9833458
```

We have learned in the course, that k=1 may lead to overtraining, because each row in the data set was used to predict itself. In case of cyber attacks classification, I think, its really reasonable to use k=1 for higher accuracy.

But I used only 2% of data.

So, my next step is to reduce data set dimension keeping only *weight*, *mean* and *covariance* columns, and use bigger data set. With my memory limit I can use 20% of the original data set.



Perform classification and calculate its accuracy:

```
##  Accuracy
## 0.9948446
```

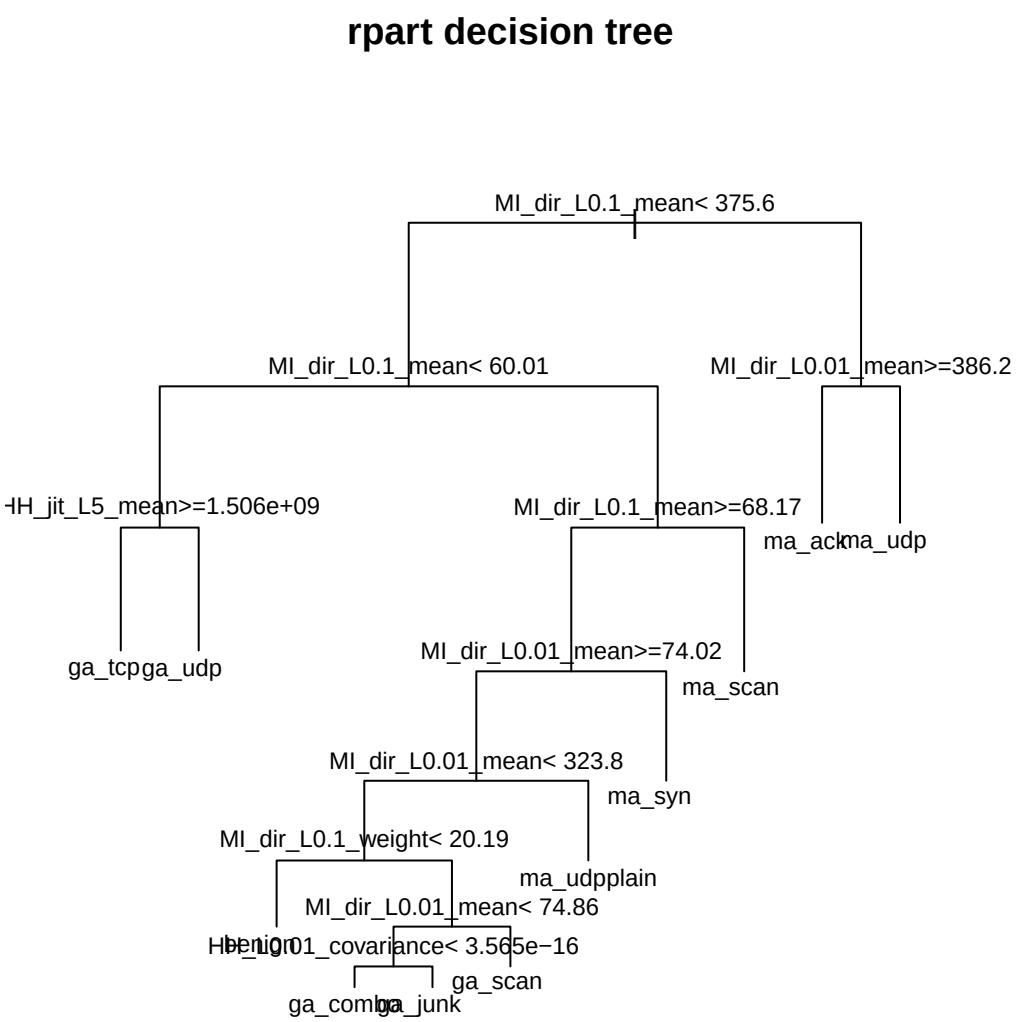
The result shows that using only *weight*, *mean* and *covariance* columns the accuracy of the classification is 99.48%!

```
##                    Sensitivity Specificity
## Class: benign        0.9917255   0.9993187
## Class: ga_combo      0.9783992   0.9987691
## Class: ga_junk       0.9580323   0.9988780
## Class: ga_scan       0.9845896   0.9996055
## Class: ga_tcp        0.9992404   0.9998596
## Class: ga_udp        0.9991500   0.9998794
## Class: ma_ack        0.9936399   0.9991814
## Class: ma_scan       0.9988857   0.9998243
## Class: ma_syn        0.9992658   0.9999107
## Class: ma_udp        0.9967181   0.9991418
## Class: ma_udpplain   0.9991462   0.9999893
```

rpart

This method allows to use all 115 predictors and all data set even with my memory limit.

It's also allowed to visualize its decision tree:



Even the algorithm uses all 115 predictors, it mostly uses *mean* column and sometimes makes its decision using *weight* and *covariance* columns. The data exploration also has shown that using only *weight*, *mean* and *covariance* columns will be enough to make attacks classification.

```
##                          Overall
## H_L0.01_mean          353405.477
## H_L0.01_variance       56341.434
## H_L0.01_weight         24000.599
## H_L0.1_mean           264537.670
## H_L0.1_variance        71452.686
```

```
## H_L0.1_weight          24001.840
## HH_jit_L0.01_mean      49212.699
## HH_jit_L0.1_mean       49212.699
## HH_jit_L1_mean         49212.699
## HH_jit_L1_weight       17184.633
## HH_jit_L3_mean         49212.699
## HH_jit_L5_mean         49212.699
## HH_L0.01_covariance     8484.052
## HH_L0.01_pcc            7101.011
## HH_L0.1_weight         17184.187
## HH_L1_weight           17184.633
## MI_dir_L0.01_mean     383099.472
## MI_dir_L0.01_variance 112251.735
## MI_dir_L0.01_weight    24000.599
## MI_dir_L0.1_mean      264537.670
## MI_dir_L0.1_variance   71452.686
## MI_dir_L0.1_weight     24001.840
## MI_dir_L1_mean        164362.573
## MI_dir_L5_weight           0.000
## MI_dir_L5_mean             0.000
## MI_dir_L5_variance         0.000
## MI_dir_L3_weight           0.000
## MI_dir_L3_mean             0.000
## MI_dir_L3_variance         0.000
## MI_dir_L1_weight           0.000
## MI_dir_L1_variance         0.000
## H_L5_weight                0.000
## H_L5_mean                  0.000
## H_L5_variance              0.000
## H_L3_weight                0.000
## H_L3_mean                  0.000
## H_L3_variance              0.000
## H_L1_weight                0.000
## H_L1_mean                  0.000
## H_L1_variance              0.000
## HH_L5_weight               0.000
## HH_L5_mean                 0.000
## HH_L5_std                  0.000
## HH_L5_magnitude            0.000
## HH_L5_radius               0.000
## HH_L5_covariance           0.000
## HH_L5_pcc                  0.000
## HH_L3_weight               0.000
## HH_L3_mean                 0.000
## HH_L3_std                  0.000
## HH_L3_magnitude            0.000
## HH_L3_radius               0.000
## HH_L3_covariance           0.000
## HH_L3_pcc                  0.000
## HH_L1_mean                 0.000
## HH_L1_std                  0.000
## HH_L1_magnitude            0.000
## HH_L1_radius               0.000
## HH_L1_covariance           0.000
```

```
## HH_L1_pcc                0.000
## HH_L0.1_mean             0.000
## HH_L0.1_std              0.000
## HH_L0.1_magnitude        0.000
## HH_L0.1_radius           0.000
## HH_L0.1_covariance       0.000
## HH_L0.1_pcc              0.000
## HH_L0.01_weight          0.000
## HH_L0.01_mean            0.000
## HH_L0.01_std             0.000
## HH_L0.01_magnitude       0.000
## HH_L0.01_radius          0.000
## HH_jit_L5_weight         0.000
## HH_jit_L5_variance       0.000
## HH_jit_L3_weight         0.000
## HH_jit_L3_variance       0.000
## HH_jit_L1_variance       0.000
## HH_jit_L0.1_weight       0.000
## HH_jit_L0.1_variance     0.000
## HH_jit_L0.01_weight      0.000
## HH_jit_L0.01_variance    0.000
## HpHp_L5_weight           0.000
## HpHp_L5_mean             0.000
## HpHp_L5_std              0.000
## HpHp_L5_magnitude        0.000
## HpHp_L5_radius           0.000
## HpHp_L5_covariance       0.000
## HpHp_L5_pcc              0.000
## HpHp_L3_weight           0.000
## HpHp_L3_mean             0.000
## HpHp_L3_std              0.000
## HpHp_L3_magnitude        0.000
## HpHp_L3_radius           0.000
## HpHp_L3_covariance       0.000
## HpHp_L3_pcc              0.000
## HpHp_L1_weight           0.000
## HpHp_L1_mean             0.000
## HpHp_L1_std              0.000
## HpHp_L1_magnitude        0.000
## HpHp_L1_radius           0.000
## HpHp_L1_covariance       0.000
## HpHp_L1_pcc              0.000
## HpHp_L0.1_weight         0.000
## HpHp_L0.1_mean           0.000
## HpHp_L0.1_std            0.000
## HpHp_L0.1_magnitude      0.000
## HpHp_L0.1_radius         0.000
## HpHp_L0.1_covariance     0.000
## HpHp_L0.1_pcc            0.000
## HpHp_L0.01_weight        0.000
## HpHp_L0.01_mean          0.000
## HpHp_L0.01_std           0.000
## HpHp_L0.01_magnitude     0.000
## HpHp_L0.01_radius        0.000
```

```
## HpHp_L0.01_covariance      0.000
## HpHp_L0.01_pcc             0.000
```

Predictors importance shows the same results.

```
##  Accuracy
## 0.9865639
```

Surprisingly, the accuracy of the classification is not so high. But the algorithm works faster than previous ones.

```
##                     Sensitivity Specificity
## Class: benign         0.9996771   0.9998926
## Class: ga_combo       0.9978566   0.9861337
## Class: ga_junk        0.5419017   0.9998767
## Class: ga_scan        0.9997990   0.9998685
## Class: ga_tcp         0.9988713   0.9999935
## Class: ga_udp         0.9990744   0.9999912
## Class: ma_ack         1.0000000   0.9999935
## Class: ma_scan        0.9999814   1.0000000
## Class: ma_syn         1.0000000   1.0000000
## Class: ma_udp         0.9999748   0.9999923
## Class: ma_udpplain    0.9999268   0.9999915
```

randomForest

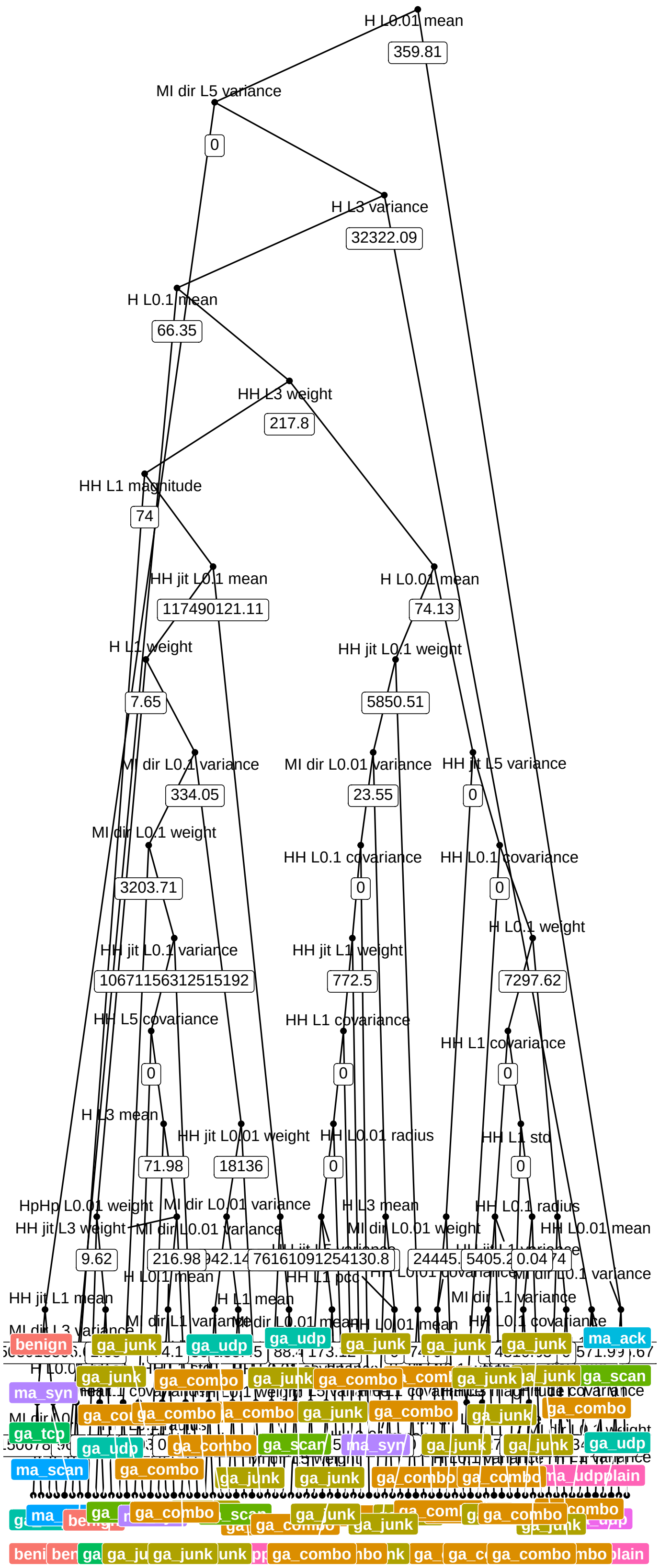
The algorithm builds a lot of trees for each forest, so the computation time will be a challenge. Because of memory limit I use only 10% of the original data set.

```
## Accuracy
## 0.999784
```

The accuracy of the classification is 99.98%! It's really significant result!

```
##                     Sensitivity Specificity
## Class: benign         1.0000000   0.9999174
## Class: ga_combo       1.0000000   0.9999374
## Class: ga_junk        0.9972490   1.0000000
## Class: ga_scan        1.0000000   1.0000000
## Class: ga_tcp         0.9995660   1.0000000
## Class: ga_udp         0.9994333   0.9999781
## Class: ma_ack         1.0000000   1.0000000
## Class: ma_scan        0.9998143   0.9999780
## Class: ma_syn         1.0000000   0.9999777
## Class: ma_udp         0.9999159   1.0000000
## Class: ma_udpplain    1.0000000   0.9999786
```

It will be interesting to visualize at least one decision tree from the 'forest'. The *randomForest* package has no tool for visualization, but the decision tree can be visualized using method described in "Plotting trees from Random Forest models with ggraph" article.



Let’s also check the importance of the predic-

tors:

```
##                          MeanDecreaseGini
## MI_dir_L5_weight             1.629290e+02
## MI_dir_L5_mean               6.067832e+02
## MI_dir_L5_variance           2.759348e+02
## MI_dir_L3_weight             3.471970e+02
## MI_dir_L3_mean               7.948098e+02
## MI_dir_L3_variance           3.364843e+02
## MI_dir_L1_weight             6.613762e+02
## MI_dir_L1_mean               1.481637e+03
## MI_dir_L1_variance           6.199132e+02
## MI_dir_L0.1_weight           8.220898e+02
## MI_dir_L0.1_mean             2.484490e+03
## MI_dir_L0.1_variance         1.574105e+03
## MI_dir_L0.01_weight          5.639033e+02
## MI_dir_L0.01_mean            3.316452e+03
## MI_dir_L0.01_variance        2.211612e+03
## H_L5_weight                  1.567768e+02
## H_L5_mean                    5.665939e+02
## H_L5_variance                2.151719e+02
## H_L3_weight                  2.527266e+02
## H_L3_mean                    8.745435e+02
## H_L3_variance                3.821558e+02
## H_L1_weight                  6.178779e+02
## H_L1_mean                    1.632974e+03
## H_L1_variance                6.676451e+02
## H_L0.1_weight                9.381563e+02
## H_L0.1_mean                  2.562858e+03
## H_L0.1_variance              1.515624e+03
## H_L0.01_weight               5.501501e+02
## H_L0.01_mean                 3.074607e+03
## H_L0.01_variance             1.986043e+03
## HH_L5_weight                 9.464418e+01
## HH_L5_mean                   1.181018e+02
## HH_L5_std                    2.282920e+00
## HH_L5_magnitude              3.917197e+02
## HH_L5_radius                 5.177709e+00
## HH_L5_covariance             2.326344e+00
## HH_L5_pcc                    1.725577e+00
## HH_L3_weight                 1.568513e+02
## HH_L3_mean                   1.489100e+02
## HH_L3_std                    4.155548e+00
## HH_L3_magnitude              2.707469e+02
## HH_L3_radius                 7.161284e+00
## HH_L3_covariance             8.074331e+00
## HH_L3_pcc                    5.694683e+00
## HH_L1_weight                 1.617010e+02
## HH_L1_mean                   1.404513e+02
## HH_L1_std                    2.098789e+01
## HH_L1_magnitude              4.248483e+02
## HH_L1_radius                 1.802139e+01
## HH_L1_covariance             1.542340e+01
## HH_L1_pcc                    1.494636e+01
```

```
## HH_L0.1_weight           1.428854e+02
## HH_L0.1_mean             1.605112e+02
## HH_L0.1_std              1.827604e+01
## HH_L0.1_magnitude        5.146708e+02
## HH_L0.1_radius           2.389628e+01
## HH_L0.1_covariance       6.097681e+01
## HH_L0.1_pcc              6.676787e+01
## HH_L0.01_weight          1.399914e+02
## HH_L0.01_mean            1.868315e+02
## HH_L0.01_std             2.827777e+01
## HH_L0.01_magnitude       4.002919e+02
## HH_L0.01_radius          4.141310e+01
## HH_L0.01_covariance      2.202690e+02
## HH_L0.01_pcc             1.733290e+02
## HH_jit_L5_weight         9.225907e+01
## HH_jit_L5_mean           6.698658e+02
## HH_jit_L5_variance       1.266254e+01
## HH_jit_L3_weight         1.506776e+02
## HH_jit_L3_mean           7.075282e+02
## HH_jit_L3_variance       2.860039e+01
## HH_jit_L1_weight         1.833773e+02
## HH_jit_L1_mean           6.893838e+02
## HH_jit_L1_variance       5.066983e+01
## HH_jit_L0.1_weight       9.019264e+01
## HH_jit_L0.1_mean         8.563301e+02
## HH_jit_L0.1_variance     6.745372e+01
## HH_jit_L0.01_weight      1.325344e+02
## HH_jit_L0.01_mean        9.284398e+02
## HH_jit_L0.01_variance    4.467145e+01
## HpHp_L5_weight           2.571133e+01
## HpHp_L5_mean             1.167354e+02
## HpHp_L5_std              4.687386e+00
## HpHp_L5_magnitude        8.902054e+01
## HpHp_L5_radius           3.367449e+00
## HpHp_L5_covariance       4.451310e-01
## HpHp_L5_pcc              7.108996e-02
## HpHp_L3_weight           4.826900e+01
## HpHp_L3_mean             1.324760e+02
## HpHp_L3_std              1.018287e+01
## HpHp_L3_magnitude        1.079529e+02
## HpHp_L3_radius           5.093146e+00
## HpHp_L3_covariance       4.739273e-01
## HpHp_L3_pcc              1.574406e-01
## HpHp_L1_weight           1.575448e+01
## HpHp_L1_mean             1.075175e+02
## HpHp_L1_std              1.865342e-01
## HpHp_L1_magnitude        2.088341e+02
## HpHp_L1_radius           1.312879e+00
## HpHp_L1_covariance       2.135188e-01
## HpHp_L1_pcc              1.409168e-01
## HpHp_L0.1_weight         2.938127e+01
## HpHp_L0.1_mean           1.055527e+02
## HpHp_L0.1_std            4.570772e+00
## HpHp_L0.1_magnitude      1.177564e+02
```

```
## HpHp_L0.1_radius          7.766010e+00
## HpHp_L0.1_covariance      3.358746e-01
## HpHp_L0.1_pcc             1.702916e-01
## HpHp_L0.01_weight         6.641604e+01
## HpHp_L0.01_mean           9.184377e+01
## HpHp_L0.01_std            1.852310e+01
## HpHp_L0.01_magnitude      1.417621e+02
## HpHp_L0.01_radius         7.859250e+00
## HpHp_L0.01_covariance     3.085601e-01
## HpHp_L0.01_pcc            1.624912e-01
```

As expected, *mean*, *weight* and *covariance* columns more important for making decision tree with *variance* and *radius* columns.

Result

```
## # A tibble: 5 x 5
##   Method                        Predictors    Data_proportion Parapeters Accuracy
##   <chr>                         <chr>                  <dbl> <chr>         <dbl>
## 1 KNN, using PCA                25, (weight)            0.6  "k = 18"      0.795
## 2 KNN, using train() function~ 115                      0.02 "k = 1"       0.858
## 3 KNN, using train() function~ 24, (weight|~            0.2  "k = 1"       0.995
## 4 rpart                         115                     1    ""            0.987
## 5 randomForest                  115                     0.1  ""            1.00
```

As seen from the result table, *randomForest* algorithm gives the best accuracy in the attacks classification. But it was performed only with 10% of the original data set.

The next algorithm that gives great accuracy is *KNN* that used train() function from *caret* package. But it used only *weight*, *mean* and *covariance* columns for the classification and 0nly 20% of the original data set.

rpart algorithm uses all data set and all 115 predictors, works fast, but doesn't give significant result.

Conclusion

The team that collected the original data set, got 100% TPR in detecting cyber attacks.

The algorithms that we learned in HarvardX PH125.9x Data Science course are in no way inferior in attacks detection accuracy. *randomForest* algorithm gives 99,98% accuracy in attacks detection on 10% of original data set.

The team that collected the data set, used only benign traffic to train their autodecoder. I can't use this approach with *randomForest* because it has to know all possible outcomes after training. That's why I simply divided the data set into two parts: train and test sets.

I also have memory limit on my laptop, that's why I can use only 10% of original data set to check with *randomForest* algorithm. I've checked this algorithm on a computer with 4.0.4 R version and 32GB of memory limit. It's allowed to use only 60% of original data set, but gave 99.99% of attack detection accuracy.

Thus, increasing memory limit will allow to use all the original data set, and therefore, increase the attack detection accuracy.