Taarak Shah
STAT 4893

Predicting Severity of Microsoft Office Exploits

Introduction and Background

For my individual research, I am working with a real life cybersecurity vulnerability dataset obtained from http://www.cvedetails.com. Cybersecurity vulnerabilities are defined as any exploit or security loophole in which a hacker can infiltrate a machine and gain access to a user's personal files or information (Arora et. al, 2006). The website is sectioned by product vendor and their individual products. From there, information exists on each vulnerability and severity. I want to focus specifically on vulnerabilities for Microsoft products (n=6746 vulnerabilties). From there, I want to compare various machine learning models with R that will predict the severity of future vulnerabilities that are disclosed to the public. The goal of this analysis is to account for some of the missing data, apply different learning algorithms after ensuring assumptions are met, and compare classification performance among the methods.

Methods and Materials

For this project, I have a dataset consisting of all vulnerabilities for all Microsoft products from 1999 to 2019. This dataset consists of 8 features and 1 target variable. The 8 features are all concerning the nature of the vulnerability, such as the level of access gained, the complexity of the exploit, etc. The target variable is the severity of the vulnerability, a score assigned by the National Vulnerability Database at http://nvd.nist.gov. The features with which we will predict `severity` are as follows:

- `gained_access:` A variable indicating the level of access gained by the hacker; a factor with 3 levels, "none", "user", "admin"

- `access:` A variable indicating the form in which the hacker has access; a factor with 3 levels, “remote”, “local”, “local network”
- `complexity:` A variable indicating how difficult the exploit is; a factor with 3 levels, “low”, “medium”, “high”
- `confidentiality:` A variable indicating how much information the hacker has access to; a factor with 3 levels, “low”, “medium”, “high”
- `integrity:` A variable indicating how much the machine being hacked was compromised; a factor with 3 levels, “low”, “medium”, “high”
- `availability:` A variable indicating how easy or hard the exploit is to perform; af factor with 3 levels, “low”, “medium”, “high”

The methods I will use in this project include a variety of classification techniques. The classification techniques I have chosen to implement are logistic regression, classification trees with random forest, and classification trees without random forest. We will remove the missing values of each column. This is because our predictors are categorical and thus using mean imputation or regression imputation would not be useful in any way. I will then run the various classification techniques to predict the target variable. I will compare the error rates for these methods after the analysis for each classifier (Demšar, 2006). These methods will be used for a response with 10 classes, which forces a slightly different means of analysis than 2-class responses. In the next paragraphs, I will illustrate the assumptions for each classification technique and the general process of the analysis.

We split the datasets into a training and test set using the validation set method with a 70% training and 30% test set. Then, we conducted our analysis using three methods: multinomial logistic regression, classification trees, and then classification trees again with

random forests. Multinomial logistic regression was used since it predicts a discrete variable given other independent covariates with a logit link function on our linear predictor. We also have strong assumptions for this method, such as no multicollinearity, no outliers, and independence in errors. Decision trees are a nonparametric regression and classification method that creates regions corresponding to specific values of the predictors, then assigns the maximum class probability of the response. We also implement a random forest algorithm, which constructs *B* bootstrapped training datasets given a subset *m* predictors. We follow the general rule and choose $m = \sqrt{p}$ predictors, where *p* is the total number of predictors. This method aids in providing decorrelated trees and reduces the variance of our prediction. Note that each of these classifiers do not assume linearity, which we cannot do since all our covariates are categorical.

Analysis and Results

For each method, we first constructed our model on the training data. We then predicted our model with the test data and classified our posterior probabilities using Bayes rule (choosing the highest posterior probability of the classes, and classifying the respective observation into that class), and then calculated our test error rate by comparing our predicted classifications to our test set observations.

Multinomial Logistic Regression

For logistic regression, we first built a model with all available predictors, then used best subset selection to determine which were most important. The model obtained with best subset selection is below. π_i is the probability of severity belonging to class *i*.

$$\log\left(\frac{\hat{\pi}_i}{1-\hat{\pi}_i}\right) = \beta_0 + \beta_1 * \texttt{gained\_access}_i - \beta_2 * \texttt{access}_i - \beta_3 * \texttt{complexity}_i - \beta_4 * \texttt{confidentiality}_i - \beta_5 * \texttt{integrity}_i - \beta_6 * \texttt{availability}_i + \epsilon$$

After building this model, we conducted a deviance test on it to see if it performed as good as the saturated model with all predictors. Our hypothesis test is as follows:

$$H_0 : l_{\text{saturated}} = l_{\text{fitted}}$$
$$H_a : l_{\text{saturated}} \neq l_{\text{fitted}}$$

We obtained a very high p-value of 1, which may be a result of overfitting the data. A p-value greater than $\alpha = 0.05$ tells us that we fail to reject the null hypothesis H_0, indicating that our fitted model is as good as the saturated model. As peculiar as the high p-value is, we have no better alternative to fit the data, so we continue our analysis with the model above.

We then used the model to predict on our test dataset. We obtained our posterior probabilities π_i and implemented Bayes classification rule for each observation I in the test set. This means the observation is classified with severity with its highest class probability. With this model, we obtained a test error rate of 0.0257, which is an indication of a fairly good classifier. We will now assess whether various implementations of classification trees perform better in this regard.

Classification Trees

For this section, we first construct a classification tree with all available predictors, then prune it back and select the subtree that leads to the largest reduction in the Gini index. The Gini index is a measure of the "impurity" of a node in a decision tree. Purity of a node is the ability to label an observation as a specific class. If a node has multiple class labels, it is considered impure, and if it only has one class label, it is considered pure. Pruning the tree leads to a reduction in the impurity of a node, which should give us a better classification performance. After pruning the tree, we calculate the test error rate of the unpruned tree and compare it to the pruned tree. The results are in Table 1 below.

Tree Type	Test Error Rate
Unpruned	0.101
Pruned	0.259

Table 1: Comparison of Decision Tree Errors

Surprisingly, the unpruned tree consists of better predictive performance. This may be because we are losing too much information with the pruned tree, and if we were to include more information to lower the error rate, then we would hardly be better off than versus using the unpruned tree. Both trees are displayed below in Figure 1 and 2.

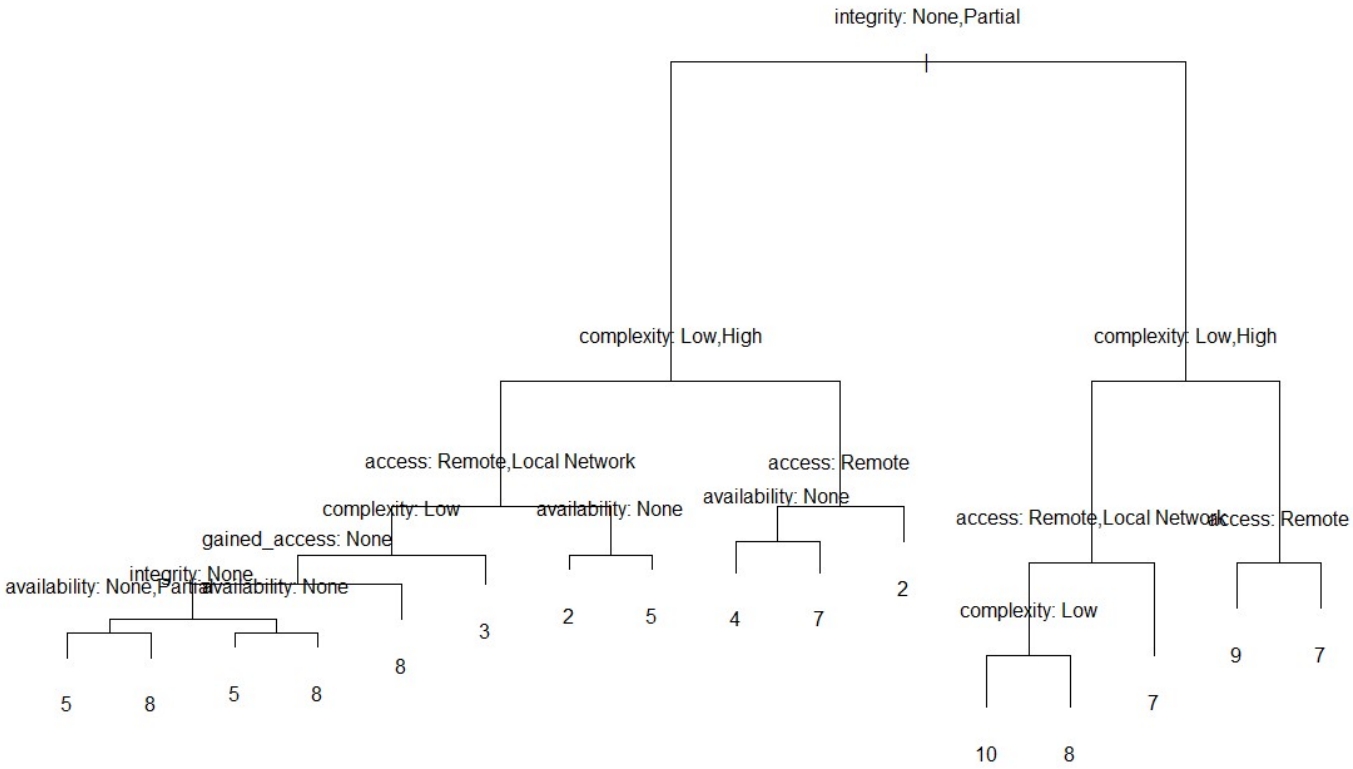


Figure 1: Unpruned Tree

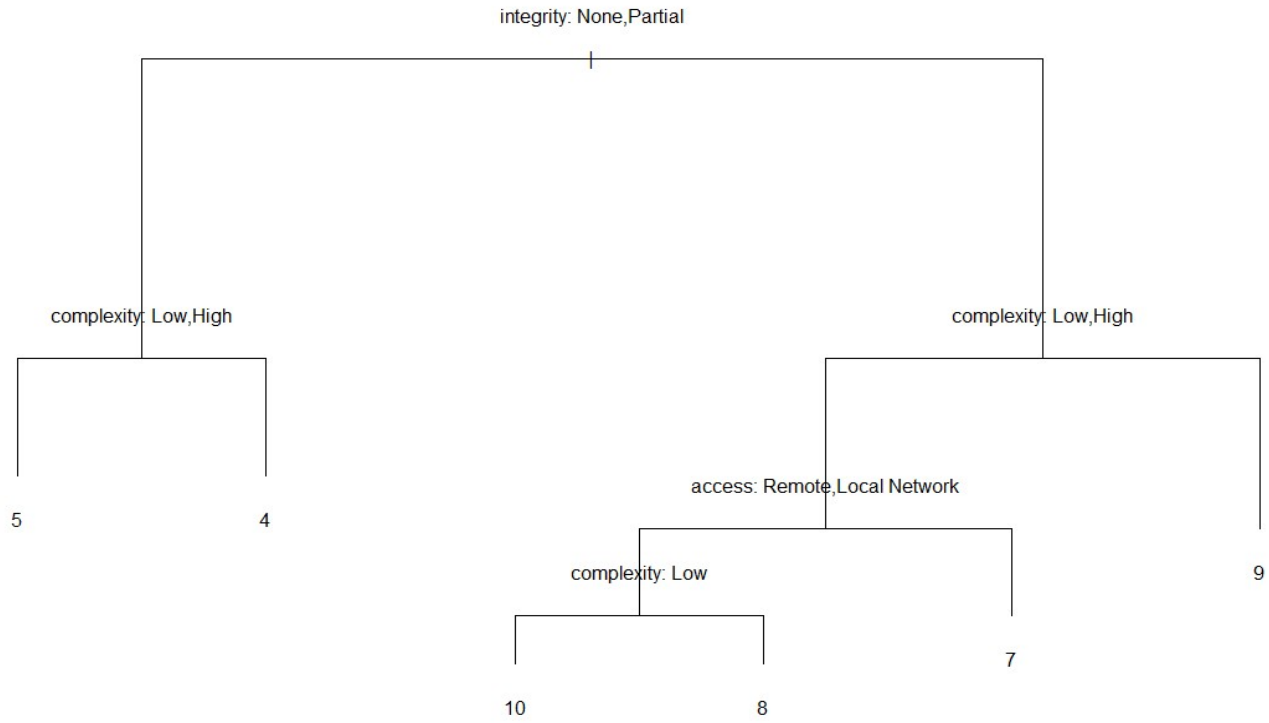


Figure 2: Pruned Tree

It is evident that pruning the tree when using strictly categorical predictors is not optimal. With the pruned tree, we see above that all classes are not even accounted for in classification of the response, so of course there will be misclassification. Using a method like this leads to high variance and high error rates.

Thankfully, there is a better way to use classification trees for prediction, and that is with the implementation of a random forest algorithm. With random forest, we construct $B = 500$ bootstrapped trees with a subset $m = 3$ of our p predictors. We chose $B = 500$ since this led to the lowest error rate among the values of 250, 500, 1000, and 2000. This also significantly reduced computation time along with providing the lowest error rate. We chose $m = 3$ since we had $p = 7$ predictors total, and a good rule of thumb is to use $m = \sqrt{p}$ predictors as we indicated earlier. This will help lower variance in our classification and improve predictive power. For the random

forest implementation, we obtained a test error rate of 0.0262, very low and similar to multinomial logistic regression.

Another advantage of bootstrapping over many trees is that we can construct a plot of variable importance to show which predictors are the most important in modeling. This is measured by the decrease of the Gini index induced by each split over a given predictor, and is averaged over the B trees. The variable importance plot is given below.

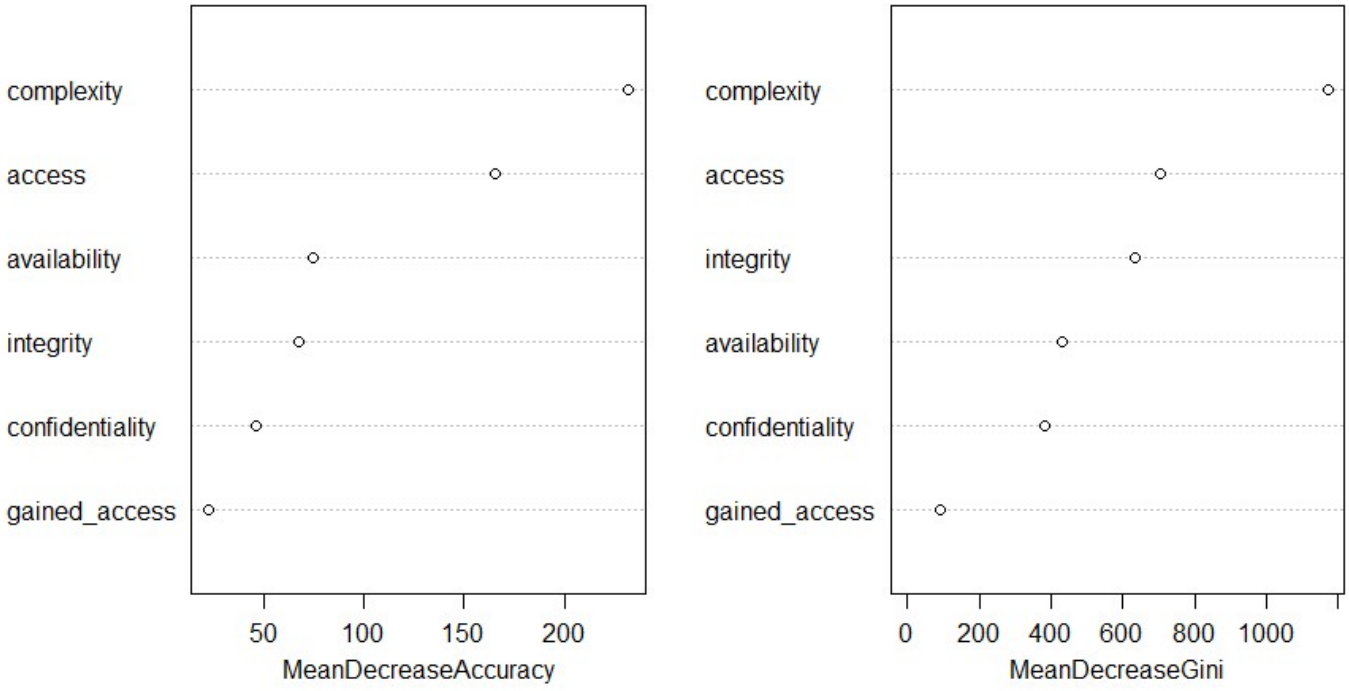


Figure 3: Important Variables Determined by Random Forest

The left plot details the variables that contribute the most to a lower error rate. The right plot details the variables that are most important in constructing the tree. The same variables are important in both. Knowing this is helpful and gives an at-a-glance look at which variables contribute most to predicting our response successfully.

Discussion and Summary

Our results are summarized in the table below.

Method	Test Error Rate
Multinomial Logistic Regression	**0.0257**
Unpruned Tree	0.1007
Pruned Tree	0.2593
Random Forest	0.0262

Table 2: Comparison of All Error Rates

When conducting this analysis, the main goal was to build a model with low error rates to provide good predictive performance for the severity score. We used three methods: logistic regression, classification trees with unpruned and pruned trees, and random forest. Overall, our results show that logistic regression and random forest perform best with this data. We obtain similarly low test error rates for these methods on both datasets.

The next goal was determining which variables are most important in predicting the severity score. Through best subset selection for logistic regression and constructing a variable importance plot for random forest, we determined that our most important predictor was `complexity`, a factor indicating how difficult it is to perform the exploit. We also observed that this is one of the important splits that leads to a large reduction in the test error rate and Gini index in our random forest analysis.

Future analysis that can be conducted to further increase predictive performance could be to consider different classification methods, like support vector machines with various kernels, a more versatile classifier. We would expect it to perform similarly to logistic regression (since the

theory behind both methods are very intertwined), but it may lead to better predictive performance if the classes are well separated. This could lead to better estimates of our classes and would help predict the correct classification. Overall, our results show that logistic regression and random forest perform best in predictive performance. We determined the most important predictors using the results of these methods and were able to adequately predict the severity with a relatively low error rate.

References

Arora, A., Nandkumar, A. & Telang, R. Does information security attack frequency increase with vulnerability disclosure? An empirical analysis. *Inf Syst Front* **8,** 350–362 (2006). https://doi.org/10.1007/s10796-006-9012-5

Demšar, J. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, **7**, 1–30, (2006). https://dl.acm.org/doi/10.5555/1248547.1248548

Horton, N. J. & Kleinman, K. P. A Comparison of Missing Data Methods and Software to Fit Incomplete Data Regression Models, *The American Statistician*, **61:1**, 79-90, (2007). https://amstat.tandfonline.com/doi/abs/10.1198/000313007X172556

Appendix (R Code)

Taarak Shah

4/3/2020

```
#misc
library(class)
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(MASS)
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 3.6.3

## -- Attaching packages ---------------------------------------------------
---------- tidyverse 1.3.0 --

## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## v purrr   0.3.3

## -- Conflicts ------------------------------------------------------------
---- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## x dplyr::select() masks MASS::select()

library(nnet)

#model selection
library(leaps)
#ROC / AUC
library(ROCR)

## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

#trees
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli

library(DAAG)

##
## Attaching package: 'DAAG'

## The following object is masked from 'package:MASS':
##
##     hills

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin

library(gbm)

## Loaded gbm 2.1.5
```

Data cleaning

```
# import data
cyber = read.csv("data_postmerge.csv", header=TRUE, fileEncoding="UTF-8-BOM")
# convert target variable to categorical
cyber = cyber %>%
  mutate(score = case_when(
    score < 0.5                ~ "0",
    score >= 0.5 & score < 1.5 ~ "1",
    score >= 1.5 & score < 2.5 ~ "2",
    score >= 2.5 & score < 3.5 ~ "3",
    score >= 3.5 & score < 4.5 ~ "4",
    score >= 4.5 & score < 5.5 ~ "5",
    score >= 5.5 & score < 6.5 ~ "6",
    score >= 6.5 & score < 7.5 ~ "7",
    score >= 7.5 & score < 8.5 ~ "8",
    score >= 8.5 & score < 9.5 ~ "9",
    score >= 9.5               ~ "10",
```

```
))

# remove cve_id, pubdate, upd_date, auth (only has one level)
drop = c("cve_id","pubdate","upd_date", "auth")
cyber = cyber[,!(names(cyber) %in% drop)]
head(cyber)

##   score gained_access access complexity confidentiality integrity
## 1     8         Admin Remote       High        Complete  Complete
## 2     5          User Remote       High         Partial   Partial
## 3     5          None Remote        Low         Partial      None
## 4     8         Admin Remote       High        Complete  Complete
## 5     7         Admin  Local        Low        Complete  Complete
## 6     8         Admin Remote       High        Complete  Complete
##   availability
## 1     Complete
## 2      Partial
## 3         None
## 4     Complete
## 5     Complete
## 6     Complete

# fix data types, reorder factors

# gained_access: none, user, admin
# access = remote, local, localnet
# complexity: low, med, high
# confidentiality: none, partial, complete
# integrity: none, partial, complete
# availability: none, partial, complete
cyber$score = as.factor(cyber$score)
cyber$gained_access = factor(cyber$gained_access, levels(cyber$gained_access)
[c(2,3,1)])
cyber$access = factor(cyber$access, levels(cyber$access)[c(3,1,2)])
cyber$complexity = factor(cyber$complexity, levels(cyber$complexity)[c(2,3,1)
])
cyber$confidentiality = factor(cyber$confidentiality, levels(cyber$confidenti
ality)[c(2,3,1)])
cyber$integrity = factor(cyber$integrity, levels(cyber$integrity)[c(2,3,1)])
cyber$availability = factor(cyber$availability, levels(cyber$availability)[c(
2,3,1)])

# remove rows with missing categorical vars
cyber = cyber[complete.cases(cyber), ]
head(cyber)

##   score gained_access access complexity confidentiality integrity
## 1     8         Admin Remote       High        Complete  Complete
## 2     5          User Remote       High         Partial   Partial
## 3     5          None Remote        Low         Partial      None
```

```
## 4     8         Admin Remote       High        Complete  Complete
## 5     7         Admin  Local        Low        Complete  Complete
## 6     8         Admin Remote       High        Complete  Complete
##   availability
## 1     Complete
## 2      Partial
## 3         None
## 4     Complete
## 5     Complete
## 6     Complete
```

Split test and train

```
set.seed(4893)
n = length(cyber[,1])
index = sample(seq(1:n), 0.7*n)
train = cyber[index,]
test = cyber[-index,]
```

Logistic regression

```
mult.glm = multinom(score ~ gained_access + access + complexity + confidentia
lity + integrity + availability, data=train, family="binomial")

## # weights:  140 (117 variable)
## initial  value 10872.806809
## iter  10 value 3248.674952
## iter  20 value 2630.013262
## iter  30 value 1524.199335
## iter  40 value 995.753887
## iter  50 value 815.464879
## iter  60 value 682.808187
## iter  70 value 588.611714
## iter  80 value 527.149406
## iter  90 value 459.701423
## iter 100 value 418.469115
## final  value 418.469115
## stopped after 100 iterations

summary(mult.glm)

## Warning in sqrt(diag(vc)): NaNs produced

## Call:
## multinom(formula = score ~ gained_access + access + complexity +
##     confidentiality + integrity + availability, data = train,
##     family = "binomial")
##
## Coefficients:
##    (Intercept) gained_accessUser gained_accessAdmin  accessLocal
## 10   -40.38833         -31.46761           67.96707 -244.0814830
## 2    107.69860         -85.19440           87.69532    0.2226187
```

```
## 3     63.04913         -66.40251           27.37301  -12.9671398
## 4    102.65703          27.08158           23.59290  -12.8423698
## 5     97.35077          29.23569          -65.94085  -26.7233638
## 6     84.30426          22.42036           19.35131  -44.6231889
## 7     57.95515          20.73722           31.26741  -72.6494168
## 8   -149.57236          24.35372           29.33548 -296.3383336
## 9   -125.09575           3.95082           67.15445 -350.4657421
##    accessLocal Network complexityMedium complexityHigh
## 10          -167.86528       -131.52357    -387.422096
## 2             70.00194         23.40077      -5.332377
## 3            110.99235         21.43250      38.662556
## 4             60.81988         24.14984     -10.605658
## 5             51.77469         16.14063     -24.408123
## 6             40.73572         15.92838     -55.435065
## 7             19.21141         15.74840    -106.733793
## 8             13.15296        -44.01745    -166.056838
## 9           -245.40047        -71.29662    -279.549604
##    confidentialityPartial confidentialityComplete integrityPartial
## 10               26.48499               97.546088         5.299824
## 2                18.63391              -70.974977      -103.231085
## 3                23.48621              -52.775953       -98.723666
## 4                29.20328              -41.259347       -92.429058
## 5                36.62271              -13.658466       -85.305382
## 6                44.24640                1.336491       -76.233552
## 7                55.36745               30.049768       -69.449797
## 8                69.69643              245.080675       -38.977811
## 9                71.37994               48.535469        15.925781
##    integrityComplete availabilityPartial availabilityComplete
## 10         162.39725          64.3551789            101.49752
## 2          -77.30960          -0.4889323            -45.14913
## 3          -50.79093           5.3712164            -21.09623
## 4          -74.70170           9.3414572            -10.61642
## 5          -28.65418          18.5986272             18.59447
## 6           -1.04338          25.2858023             45.05417
## 7           32.03011          37.9108522             73.94109
## 8         -119.49850         201.7544828            283.46975
## 9          299.81500         106.9595676             96.18774
##
## Std. Errors:
##    (Intercept) gained_accessUser gained_accessAdmin  accessLocal
## 10   24.592604               NaN       4.430858e+01 1.756685e+02
## 2     6.743738               NaN                NaN 2.761451e+00
## 3     4.780940      5.388545e-09       3.023791e-03 2.393204e+01
## 4     6.312208      2.668575e+00                NaN 3.967353e+00
## 5     6.322285      2.104070e+00                NaN 4.657587e+00
## 6     6.420639      1.523677e+00       1.154276e+02 5.362405e+00
## 7    15.749201      1.511615e+00       1.696972e+01 1.728226e+01
## 8    19.781339      1.599706e+00       1.700628e+01 2.776795e+01
## 9    22.622486      1.065630e-07       4.430859e+01 4.049343e-42
##    accessLocal Network complexityMedium complexityHigh
```

```
## 10        3.291512e-08              NaN           NaN
## 2         5.812967e+00         3.066919      3.249953
## 3         4.920538e+00         3.245607      3.026234
## 4         5.667832e+00         2.800674      4.032731
## 5         5.554646e+00         2.798606      4.899164
## 6         5.548579e+00         2.881433     17.916223
## 7         1.269983e+01         3.001760    129.636309
## 8         1.296507e+01        45.994268     83.644036
## 9         2.656915e-78        37.752694    260.919765
##    confidentialityPartial confidentialityComplete integrityPartial
## 10           1.589082e-05                36.29068        13.168562
## 2            6.003605e+00                20.51878         6.714083
## 3            5.392443e+00                24.09012         6.311950
## 4            5.205299e+00                12.97918         6.170581
## 5            5.181372e+00                12.92748         6.200252
## 6            5.211551e+00                13.06024         6.237020
## 7            1.010842e+01                17.68715         6.860304
## 8            7.415812e+00                28.37150         7.414576
## 9            2.868058e+01                53.34531        28.680576
##    integrityComplete availabilityPartial availabilityComplete
## 10      36.290682296           13.168562             36.29068
## 2       12.436760560           14.097838            232.68211
## 3        9.789913386           14.468492             74.60844
## 4        0.002588842           14.346666             70.69552
## 5        9.275972983           14.411817             70.71065
## 6        9.015250243           14.487309             70.74907
## 7       15.146800361           18.428524             72.53205
## 8       42.634037970            7.414576             58.22768
## 9       18.372478916           28.680576             52.44715
##
## Residual Deviance: 836.9382
## AIC: 1070.938
```

```
# best subset
#(bss.reg.sum = summary(regsubsets(score ~ gained_access + access + complexit
y + confidentiality + integrity + availability, data=train)))
#which.max(bss.reg.sum$adjr2)
# best includes all covariates

# deviance test
1-pchisq(mult.glm$deviance, mult.glm$edf, lower.tail = FALSE)
```

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

$$H_0: l_{\text{saturated}} = l_{\text{fitted}}$$

$$H_a: l_{\text{saturated}} \neq l_{\text{fitted}}$$

Fail to reject H_0, so then the fitted model is as good as the saturated model.

```
pred.glm = predict(mult.glm, test, type="class")
glm.ER = sum(pred.glm != test$score) / length(test$score)
glm.ER
```

```
## [1] 0.0256917
```

```
table.glm = table(test$score, pred.glm)
table.glm
```

```
##     pred.glm
##        1  10   2   3   4   5   6   7   8   9
##   1    0   0   0   0   0   0   0   0   0   0
##   10   0  75   0   0   0   0   0   0   0   0
##   2    0   0 108   1   0   0   0   0   0   0
##   3    0   0   0  42   0   0   0   0   0   0
##   4    0   0   0   0 232  14   0   0   0   0
##   5    0   0   0   2   0 289   1   0   0   0
##   6    0   0   0   0   0   1  37   7   0   0
##   7    0   0   0   0   0   0   0 264  17   0
##   8    0   0   0   0   0   0   0   0 347   1
##   9    0   8   0   0   0   0   0   0   0 578
```

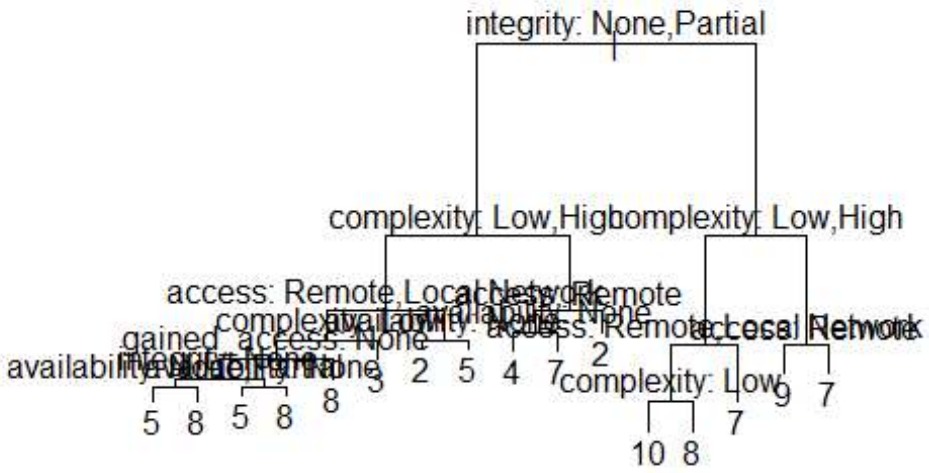
Decision trees

Unpruned tree

```
# decision tree, unpruned
groot1 = tree(score ~ gained_access + access + complexity + confidentiality +
integrity + availability, data=train)
summary(groot1)
```

```
##
## Classification tree:
## tree(formula = score ~ gained_access + access + complexity +
##     confidentiality + integrity + availability, data = train)
## Variables actually used in tree construction:
## [1] "integrity"     "complexity"    "access"        "gained_access"
## [5] "availability"
## Number of terminal nodes:  16
## Residual mean deviance:  0.565 = 2659 / 4706
## Misclassification error rate: 0.09297 = 439 / 4722
```

```
plot(groot1)
text(groot1, pretty = 0)
```




```
# test error rate for unpruned tree
pred.tree = predict(groot1, newdata = test, type = "class")
table(test$score, pred.tree)
```

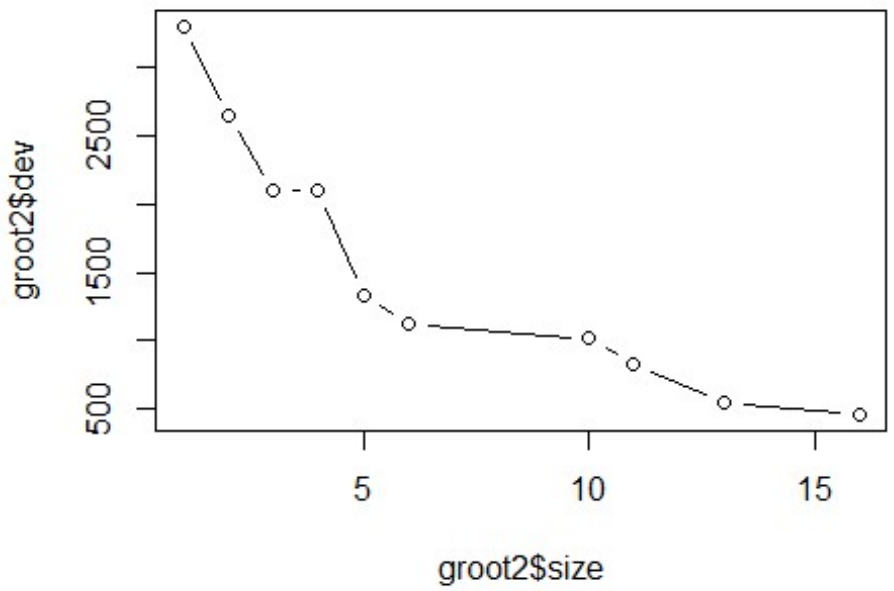
```
##     pred.tree
##        1  10   2   3   4   5   6   7   8   9
##   1    0   0   0   0   0   0   0   0   0   0
##   10   0  75   0   0   0   0   0   0   0   0
##   2    0   0  92   1   0  16   0   0   0   0
##   3    0   0   0  41   0   1   0   0   0   0
##   4    0   0  11  10 184  18   0  23   0   0
##   5    0   0  20  23   0 247   0   0   2   0
##   6    0   0   0   0  13  11   0  14   7   0
##   7    0   0   0   0   2   0   0 262  17   0
##   8    0   3   0   0   0   4   0   0 341   0
##   9    0   8   0   0   0   0   0   0   0 578
```

```
tree.ER1 = sum(pred.tree != test$score) / length(test$score)
tree.ER1
```

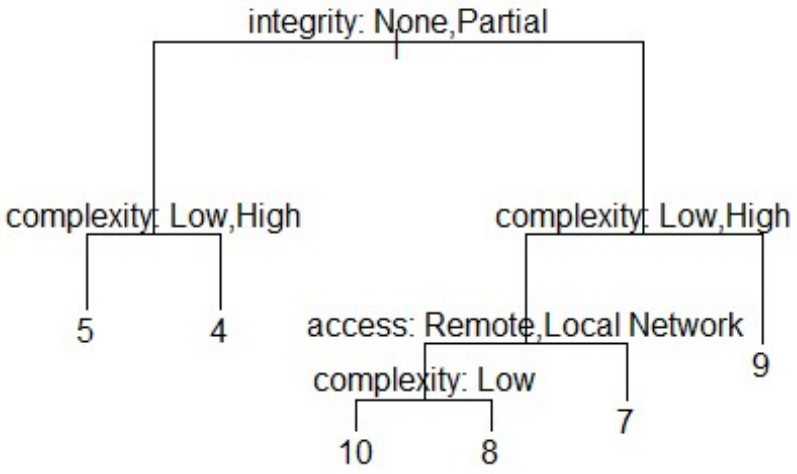
```
## [1] 0.1007905
```

Prune tree

```
set.seed(4893)
groot2 <- cv.tree(groot1, FUN = prune.misclass)
plot(groot2$size, groot2$dev, type = "b")
```




```
groot3 = prune.misclass(groot1, best = 6)
plot(groot3)
text(groot3, pretty = 0)
```



```
# test error rate for pruned tree
pred.tree2 = predict(groot3, newdata = test, type = "class")
table(test$score, pred.tree2)
```

```
##     pred.tree2
##        1  10   2   3   4   5   6   7   8   9
##   1    0   0   0   0   0   0   0   0   0   0
##   10   0  75   0   0   0   0   0   0   0   0
##   2    0   0   0   0  25  84   0   0   0   0
##   3    0   0   0   0   0  42   0   0   0   0
##   4    0   0   0   0 213  33   0   0   0   0
##   5    0   0   0   0   6 284   0   0   2   0
##   6    0   0   0   0  26  18   0   1   0   0
##   7    0   0   0   0  61  16   0 165   1  38
##   8    0   3   0   0   0 161   0   0 184   0
##   9    0   8   0   0   0   0   0   0   0 578
```

```
tree.ER2 = sum(pred.tree2 != test$score) / length(test$score)
tree.ER2
```

```
## [1] 0.2593874
```

Then use RF algorithm. Bagging will not be that useful since it will lead to correlated trees. Good RF default for classification tree is $m = \sqrt{p}$.

```
set.seed(4893)
#random forest
rf = randomForest(score ~ gained_access + access + complexity + confidentiali
ty + integrity + availability, data = train, mtry = 3, importance = TRUE)
pred.rf = predict(rf, newdata = test, type = "class")
table(test$score, pred.rf)
```

```
##     pred.rf
##        1  10   2   3   4   5   6   7   8   9
##   1    0   0   0   0   0   0   0   0   0   0
##   10   0  75   0   0   0   0   0   0   0   0
##   2    0   0 108   1   0   0   0   0   0   0
##   3    0   0   0  42   0   0   0   0   0   0
##   4    0   0   0   0 231  15   0   0   0   0
##   5    0   0   0   2   0 290   0   0   0   0
##   6    0   0   0   0   0   2  36   7   0   0
##   7    0   0   0   0   0   0   0 264  17   0
##   8    0   0   0   0   0   1   0   0 347   0
##   9    0   8   0   0   0   0   0   0   0 578
```

```
rf.ER = sum(pred.rf != test$score) / length(test$score)
rf.ER
```

```
## [1] 0.02618577
```

```
importance(rf)
```

```
##                 1        10         2        3         4         5
## gained_access   0  5.733397  16.30574 15.01260  12.97624  11.99814
## access          0 51.854853 123.79322 41.67151  77.01343 106.33018
## complexity      0 68.007002  18.92748 77.36292 186.80744 199.48269
## confidentiality 0 20.563223  58.94821 25.01669  53.69949  46.73367
## integrity       0 29.929166  63.24770 43.91000  65.52387  78.08661
## availability    0 17.504721  66.16032 41.05658  87.87584  82.09573
##                        6          7          8         9
## gained_access   12.46650  -2.890692   9.484457  -2.28595
## access          43.80140 135.445601  84.235769  58.74445
## complexity      34.06054  51.817505 160.750655 146.80181
## confidentiality 30.01398  23.367412  21.036135  17.08437
## integrity       33.61535  38.391059  35.991842  37.65700
## availability    34.91518  25.644446  25.313160  16.82640
##                 MeanDecreaseAccuracy MeanDecreaseGini
## gained_access                22.54789         92.94951
## access                      165.40346        703.44604
## complexity                  231.80518       1171.56132
## confidentiality              46.43269        384.09777
## integrity                    67.40184        635.57915
## availability                 74.47835        430.59264
```

```
varImpPlot(rf)
```