MU3D Lie Detection Report

Kun Bu

2022-07-29

```
library(reshape2)
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.1.2
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(FactoMineR)
library(devtools)
```

```
## Loading required package: usethis
```

```
## Warning: package 'usethis' was built under R version 4.1.2
```

```
install_github('sinhrks/ggfortify')
```

```
## Skipping install of 'ggfortify' from a github remote, the SHA1 (8e3e7df6) has not changed since last
##   Use `force = TRUE` to force installation
```

```
library(ggfortify)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.2
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.2
```

```
## randomForest 4.7-1.1
```

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:psych':
##
##     outlier

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

```
#import video level dataset
MU3D_Video_Level_Data0 <- read.csv("MU3D_Video_Level_Data.csv")
head(MU3D_Video_Level_Data0)
```

```
##     VideoID Valence Veracity Sex Race VidLength_ms VidLength_sec WordCount
## 1 BF001_1PT       1        1   0    0        38783         38.78       110
## 2 BF001_2NL       0        0   0    0        37120         37.12        88
## 3 BF001_3NT       0        1   0    0        38484         38.48       120
## 4 BF001_4PL       1        0   0    0        38026         38.03       124
## 5 BF002_1PT       1        1   0    0        36351         36.35        91
## 6 BF002_2NL       0        0   0    0        36650         36.65        73
##   Accuracy TruthProp Attractive Trustworthy Anxious
## 1     0.77      0.77       4.55        4.32    3.18
## 2     0.40      0.60       3.55        3.75    3.05
## 3     0.77      0.77       3.27        3.95    2.82
## 4     0.58      0.42       4.05        4.05    3.11
## 5     0.59      0.59       4.86        4.36    3.32
## 6     0.33      0.67       5.05        4.62    2.33
##
## 1                                                                      My best friend
## 2
## 3                                     So this specific person is actually just a really mean and neg
## 4 This person is actually a really kind person. She has so many friends. She’s very popular. Everyone
## 5
## 6
```

Boxplot for variables

```
#remove veractiy first
MU3D_Video_Level_Data <- MU3D_Video_Level_Data0[,-3]
str(MU3D_Video_Level_Data)
```

```
## 'data.frame':    320 obs. of  13 variables:
##  $ VideoID      : chr  "BF001_1PT" "BF001_2NL" "BF001_3NT" "BF001_4PL" ...
##  $ Valence      : int  1 0 0 1 1 0 0 1 1 0 ...
##  $ Sex          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Race         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ VidLength_ms : int  38783 37120 38484 38026 36351 36650 29141 36480 36960 29610 ...
##  $ VidLength_sec: num  38.8 37.1 38.5 38 36.4 ...
##  $ WordCount    : int  110 88 120 124 91 73 95 104 91 114 ...
##  $ Accuracy     : num  0.77 0.4 0.77 0.58 0.59 0.33 0.6 0.64 0.64 0.27 ...
##  $ TruthProp    : num  0.77 0.6 0.77 0.42 0.59 0.67 0.6 0.36 0.64 0.73 ...
```

```
##  $ Attractive   : num  4.55 3.55 3.27 4.05 4.86 5.05 4.4 4.27 3.09 3 ...
##  $ Trustworthy  : num  4.32 3.75 3.95 4.05 4.36 4.62 4.5 3.73 4.27 4.55 ...
##  $ Anxious      : num  3.18 3.05 2.82 3.11 3.32 2.33 3.15 2.91 2.64 2.73 ...
##  $ Transcription: chr  "My best friend is a really nice person. Um. She’s always kind to everyone. Sh
```

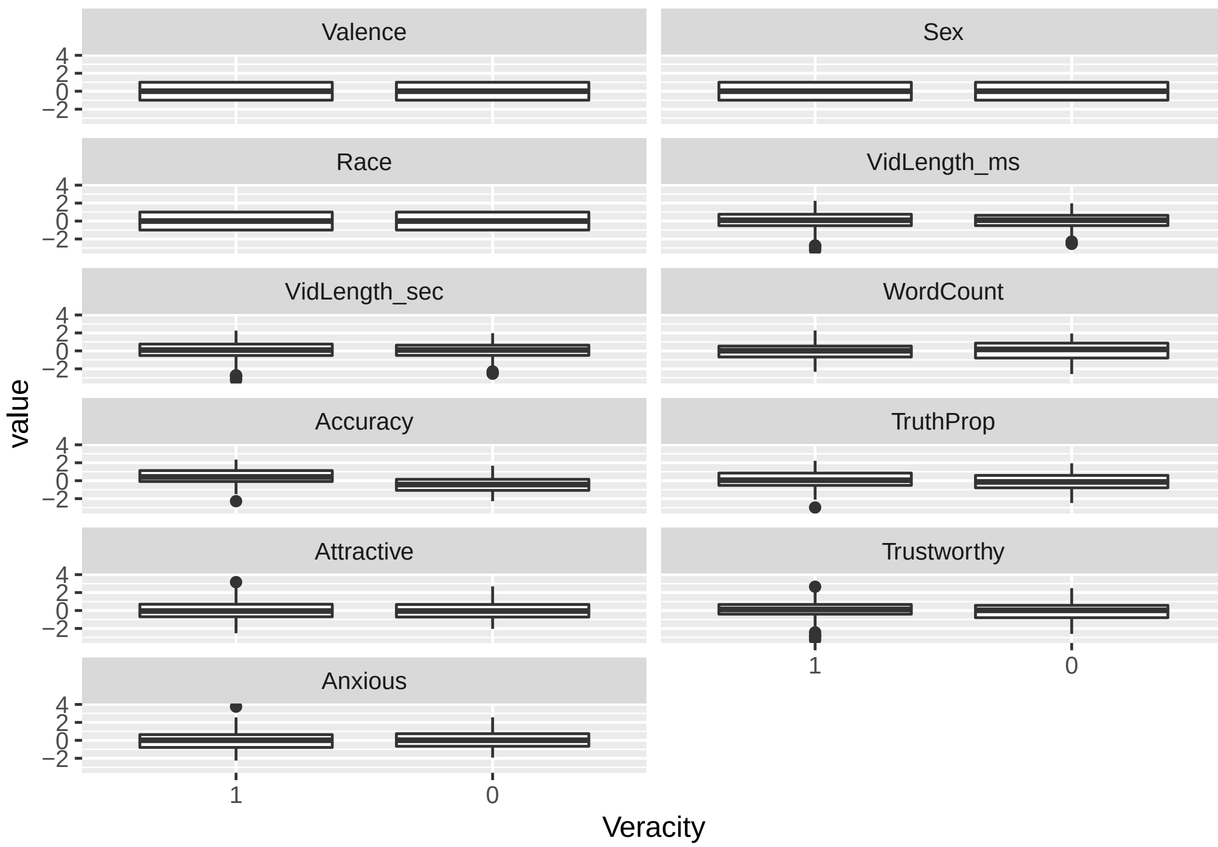
```
#scaled
colnames(MU3D_Video_Level_Data)
```

```
##  [1] "VideoID"       "Valence"       "Sex"           "Race"
##  [5] "VidLength_ms"  "VidLength_sec" "WordCount"     "Accuracy"
##  [9] "TruthProp"     "Attractive"    "Trustworthy"   "Anxious"
## [13] "Transcription"
```

```
MU3D_Video_Level_Data.scaled <- data.frame(scale(MU3D_Video_Level_Data[,-c(1,13)]))

#level veractiy
levels0 <- unique(c(MU3D_Video_Level_Data0$Veracity, MU3D_Video_Level_Data0$Veracity))
#add veracity back
MU3D_Video_Level_Data.scaled$Veracity <- factor(MU3D_Video_Level_Data0$Veracity,levels = levels0)
#melt data for boxplot
MU3D_Video_Level_Data.scaled.melt <- melt(MU3D_Video_Level_Data.scaled, id.var = "Veracity")

#plot boxplot
ggplot(data = MU3D_Video_Level_Data.scaled.melt, aes(x=Veracity, y = value))+
  geom_boxplot() +
  facet_wrap(~variable, ncol=2)
```



SVM split train and test 80/20

```
smp_size_raw <- floor(0.80 * nrow(MU3D_Video_Level_Data.scaled))
train_ind_raw <- sample(nrow(MU3D_Video_Level_Data.scaled), size = smp_size_raw)
train_raw.df <- as.data.frame(MU3D_Video_Level_Data.scaled[train_ind_raw, ])
test_raw.df <- as.data.frame(MU3D_Video_Level_Data.scaled[-train_ind_raw, ])
levels <- unique(c(train_raw.df$Veracity, test_raw.df$Veracity))
test_raw.df$Veracity  <- factor(test_raw.df$Veracity, levels=levels)
train_raw.df$Veracity <- factor(train_raw.df$Veracity, levels=levels)

# tuning best svm model for linear kernel
linear.tune <- tune.svm(Veracity ~ ., data = train_raw.df,
                        kernel = "linear",
                        cost = c(0.001, 0.01, 0.1, 1, 5, 10))
summary(linear.tune) #best cost is 1, misclassification rate no larger than 25%
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.2813846
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.5624615 0.07290786
## 2 1e-02 0.3281538 0.12316033
## 3 1e-01 0.2932308 0.11506668
## 4 1e+00 0.2813846 0.11368538
## 5 5e+00 0.2813846 0.11368538
## 6 1e+01 0.2813846 0.11368538
```

```
best.linear <- linear.tune$best.model
linear.test <- predict(best.linear, newdata = test_raw.df)
table(linear.test, test_raw.df$Veracity)
```

```
##
## linear.test  1  0
##           1 20  0
##           0 12 32
```

```
confusionMatrix(linear.test, test_raw.df$Veracity, dnn = c("Prediction", "Reference")) # 76.6% accuracy
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 20  0
##          0 12 32
##
##                Accuracy : 0.8125
```

```
##                  95% CI : (0.6954, 0.8992)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.283e-07
##
##                   Kappa : 0.625
##
##  Mcnemar's Test P-Value : 0.001496
##
##             Sensitivity : 0.6250
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7273
##              Prevalence : 0.5000
##          Detection Rate : 0.3125
##    Detection Prevalence : 0.3125
##       Balanced Accuracy : 0.8125
##
##        'Positive' Class : 1
##
```

```
#
# # tuning best svm model for sigmoid kernel
# sigmoid.tune <- tune.svm(Veracity ~. ,data = train_raw.df,
#                          kernel = "sigmoid",
#                          gamma = c(0.1,0.5,1,2,3,4),
#                          coef0 = c(0.1,0.5,1,2,3,4))
# summary(sigmoid.tune)
# best.sigmoid <- sigmoid.tune$best.model
# sigmoid.test <- predict(best.sigmoid, test_raw.df)
# table(sigmoid.test,test_raw.df$Veracity)
# confusionMatrix(sigmoid.test, test_raw.df$Veracity, dnn = c("Prediction", "Reference")) # poor 64%, k



# tuning best svm model for polynomial kernel
poly.tune <- tune.svm(Veracity ~ ., data = train_raw.df,
                       kernel = "polynomial",
                       degree = c(2, 3, 4, 5, 6),
                       coef0 = c(0.1, 0.5, 1, 2, 3, 4))
summary(poly.tune) #best degree is 3,coef0 = 3,  misclassification rate no larger than 17%( better than
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree coef0
##       4     2
##
## - best performance: 0.2030769
##
## - Detailed performance results:
##    degree coef0     error dispersion
## 1       2   0.1 0.2658462 0.09309296
```

```
## 2       3   0.1 0.2698462 0.07707063
## 3       4   0.1 0.3116923 0.07152403
## 4       5   0.1 0.3006154 0.08630695
## 5       6   0.1 0.3316923 0.09455999
## 6       2   0.5 0.2541538 0.09372081
## 7       3   0.5 0.2541538 0.07188566
## 8       4   0.5 0.2381538 0.05530288
## 9       5   0.5 0.2461538 0.05239309
## 10      6   0.5 0.2580000 0.03906905
## 11      2   1.0 0.2458462 0.08269237
## 12      3   1.0 0.2264615 0.07206469
## 13      4   1.0 0.2303077 0.03241780
## 14      5   1.0 0.2343077 0.03111811
## 15      6   1.0 0.2344615 0.03259700
## 16      2   2.0 0.2460000 0.07269563
## 17      3   2.0 0.2149231 0.05289788
## 18      4   2.0 0.2030769 0.03017372
## 19      5   2.0 0.2264615 0.03456730
## 20      6   2.0 0.2147692 0.03681751
## 21      2   3.0 0.2421538 0.07234660
## 22      3   3.0 0.2187692 0.04518528
## 23      4   3.0 0.2109231 0.02663163
## 24      5   3.0 0.2149231 0.04193260
## 25      6   3.0 0.2226154 0.03613875
## 26      2   4.0 0.2421538 0.07234660
## 27      3   4.0 0.2227692 0.05810870
## 28      4   4.0 0.2033846 0.02666469
## 29      5   4.0 0.2190769 0.05704169
## 30      6   4.0 0.2227692 0.04889200
```

```r
best.poly <- poly.tune$best.model
poly.test <- predict(best.poly, newdata = test_raw.df)
table(poly.test, test_raw.df$Veracity)
```

```
##
## poly.test  1  0
##         1 29  8
##         0  3 24
```

```r
confusionMatrix(poly.test, test_raw.df$Veracity, dnn = c("Prediction", "Reference")) # 78.12% accuracy,
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  1  0
##          1 29  8
##          0  3 24
##
##                Accuracy : 0.8281
##                  95% CI : (0.7132, 0.911)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 5.029e-08
##
##                   Kappa : 0.6562
##
```

```
##  Mcnemar's Test P-Value : 0.2278
##
##             Sensitivity : 0.9062
##             Specificity : 0.7500
##          Pos Pred Value : 0.7838
##          Neg Pred Value : 0.8889
##              Prevalence : 0.5000
##          Detection Rate : 0.4531
##    Detection Prevalence : 0.5781
##       Balanced Accuracy : 0.8281
##
##        'Positive' Class : 1
##
```

```
# tuning best svm model for radial kernel
rad.tune <- tune.svm(Veracity ~ ., data = train_raw.df,
                     kernel = "radial",
                     gamma = c(0.1, 0.5, 1, 2, 3, 4))
summary(rad.tune) #best gamma = 0.1,  misclassification rate no larger than 19%( better than linear ker
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma
##    0.5
##
## - best performance: 0.2729231
##
## - Detailed performance results:
##   gamma     error dispersion
## 1   0.1 0.2775385 0.07154609
## 2   0.5 0.2729231 0.06830030
## 3   1.0 0.3872308 0.09990673
## 4   2.0 0.5041538 0.09545366
## 5   3.0 0.5507692 0.07152697
## 6   4.0 0.5469231 0.06864232
```

```
best.rad <- rad.tune$best.model
rad.test <- predict(best.rad, newdata = test_raw.df)
table(rad.test, test_raw.df$Veracity)
```

```
##
## rad.test  1  0
##        1 24  5
##        0  8 27
```

```
confusionMatrix(rad.test, test_raw.df$Veracity, dnn = c("Prediction", "Reference")) # 79% accuracy, kap
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 24  5
```

```
##          0  8 27
##
##                Accuracy : 0.7969
##                  95% CI : (0.6777, 0.8872)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 9.405e-07
##
##                   Kappa : 0.5938
##
##  Mcnemar's Test P-Value : 0.5791
##
##             Sensitivity : 0.7500
##             Specificity : 0.8438
##          Pos Pred Value : 0.8276
##          Neg Pred Value : 0.7714
##              Prevalence : 0.5000
##          Detection Rate : 0.3750
##    Detection Prevalence : 0.4531
##       Balanced Accuracy : 0.7969
##
##        'Positive' Class : 1
##
```

```
# feature extraction
set.seed(3117)
rfeCNTL <- rfeControl(functions = lrFuncs, method = "cv", number = 11)
svm.features <- rfe(train_raw.df[,1:11], train_raw.df[,12],
                    sizes = c(11, 10, 9, 8, 7, 6, 5),
                    rfeControl = rfeCNTL,
                    method = "svmLinear")
svm.features
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (11 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          5   0.7147 0.4291    0.06559  0.1301        *
##          6   0.7145 0.4284    0.08798  0.1744
##          7   0.6910 0.3810    0.09778  0.1966
##          8   0.6911 0.3817    0.09114  0.1821
##          9   0.6991 0.3970    0.08475  0.1699
##         10   0.6953 0.3899    0.08124  0.1627
##         11   0.6953 0.3899    0.08124  0.1627
##
## The top 5 variables (out of 5):
##    Accuracy, TruthProp, Valence, WordCount, VidLength_sec
```

```
svm.features$fit$coefficients # Accuracy, TruthProp, VidLength_ms, VidLength_sec, Valence
```

```
##   (Intercept)      Accuracy     TruthProp       Valence     WordCount
##  -0.055746912  -1.133245482  -0.345753063   0.149608408   0.150722455
## VidLength_sec
```

```
##   0.006819527
```

```
# use above 8 features to train polynomial svm

#SVM split train and test 80/20

train_raw.df <- as.data.frame(MU3D_Video_Level_Data.scaled[train_ind_raw, c(1,4,5,7,8,12)])
test_raw.df <- as.data.frame(MU3D_Video_Level_Data.scaled[-train_ind_raw, c(1,4,5,7,8,12)])
levels <- unique(c(train_raw.df$Veracity, test_raw.df$Veracity))
test_raw.df$Veracity  <- factor(test_raw.df$Veracity, levels=levels)
train_raw.df$Veracity <- factor(train_raw.df$Veracity, levels=levels)
#write.csv(test_raw.df, "test_raw.df.csv")
#write.csv(train_raw.df,"train_raw.df.csv")

# tuning best svm model for polynomial kernel
poly.tune <- tune.svm(Veracity ~ ., data = train_raw.df,
                      kernel = "polynomial",
                      degree = c(2, 3, 4, 5, 6),
                      coef0 = c(0.1, 0.5, 1, 2, 3, 4))
summary(poly.tune) #best degree is 4,coef0 = 4,  misclassification rate no larger than 19%( better than
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree coef0
##       6     3
##
## - best performance: 0.07046154
##
## - Detailed performance results:
##    degree coef0      error dispersion
## 1       2   0.1 0.19138462 0.08547720
## 2       3   0.1 0.15261538 0.06301560
## 3       4   0.1 0.16461538 0.08953382
## 4       5   0.1 0.17615385 0.09364628
## 5       6   0.1 0.19953846 0.06877934
## 6       2   0.5 0.18000000 0.05332347
## 7       3   0.5 0.14061538 0.05550414
## 8       4   0.5 0.14846154 0.06899390
## 9       5   0.5 0.14846154 0.06821188
## 10      6   0.5 0.15646154 0.08318780
## 11      2   1.0 0.16461538 0.06959260
## 12      3   1.0 0.12523077 0.06586181
## 13      4   1.0 0.12492308 0.07052280
## 14      5   1.0 0.13646154 0.09524956
## 15      6   1.0 0.15600000 0.07993488
## 16      2   2.0 0.18815385 0.06284113
## 17      3   2.0 0.14446154 0.07796710
## 18      4   2.0 0.12876923 0.07327073
## 19      5   2.0 0.15200000 0.10038139
## 20      6   2.0 0.09338462 0.07807699
## 21      2   3.0 0.18430769 0.06109584
```

```
## 22      3   3.0 0.13307692 0.07928096
## 23      4   3.0 0.14076923 0.06906628
## 24      5   3.0 0.09338462 0.08250531
## 25      6   3.0 0.07046154 0.06437399
## 26      2   4.0 0.17646154 0.05844827
## 27      3   4.0 0.11723077 0.08663723
## 28      4   4.0 0.12892308 0.06880515
## 29      5   4.0 0.07415385 0.06767987
## 30      6   4.0 0.07446154 0.06887926
```

```
best.poly <- poly.tune$best.model
poly.test <- predict(best.poly, newdata = test_raw.df)
table(poly.test, test_raw.df$Veracity)
```

```
##
## poly.test  1  0
##         1 30  3
##         0  2 29
```

```
confusionMatrix(poly.test, test_raw.df$Veracity, dnn = c("Prediction", "Reference")) # 95.31% accuracy,
```

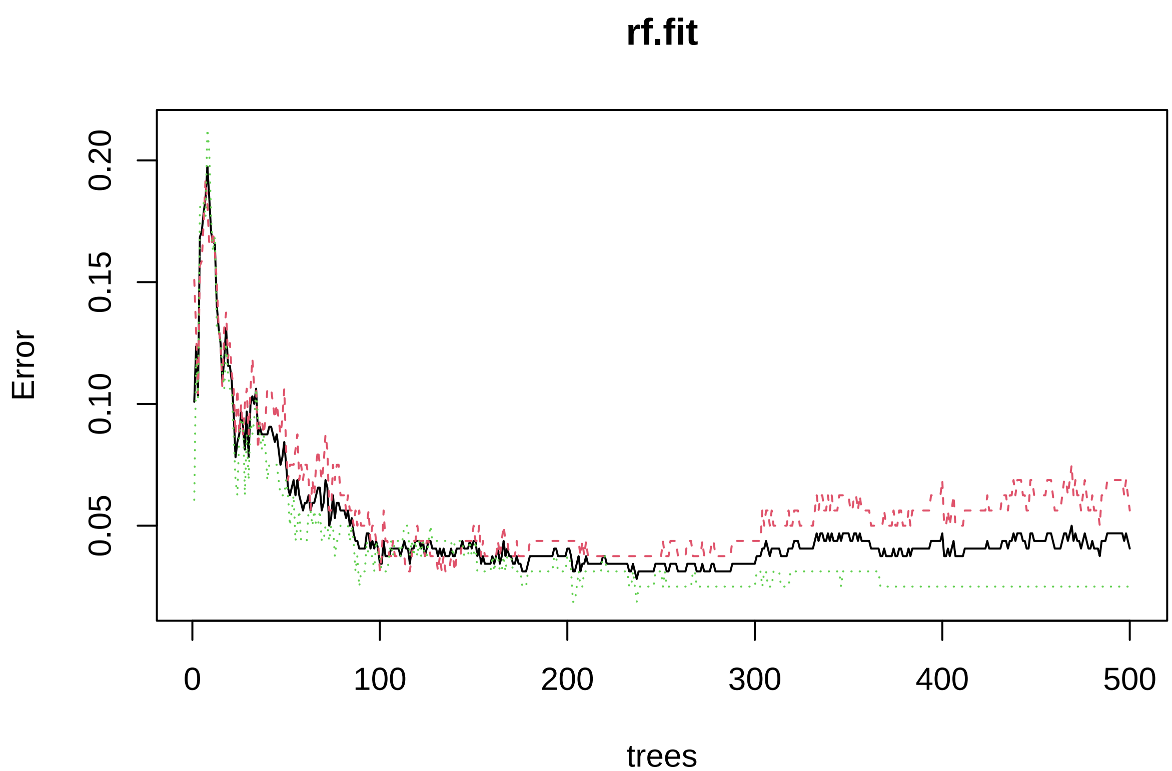
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 30  3
##          0  2 29
##
##                Accuracy : 0.9219
##                  95% CI : (0.827, 0.9741)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 4.501e-13
##
##                   Kappa : 0.8438
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9375
##             Specificity : 0.9062
##          Pos Pred Value : 0.9091
##          Neg Pred Value : 0.9355
##              Prevalence : 0.5000
##          Detection Rate : 0.4688
##    Detection Prevalence : 0.5156
##       Balanced Accuracy : 0.9219
##
##        'Positive' Class : 1
##
```

Random Forest

```
# fit random forest
rf.fit <- randomForest(Veracity~., data= MU3D_Video_Level_Data.scaled)
rf.fit
```

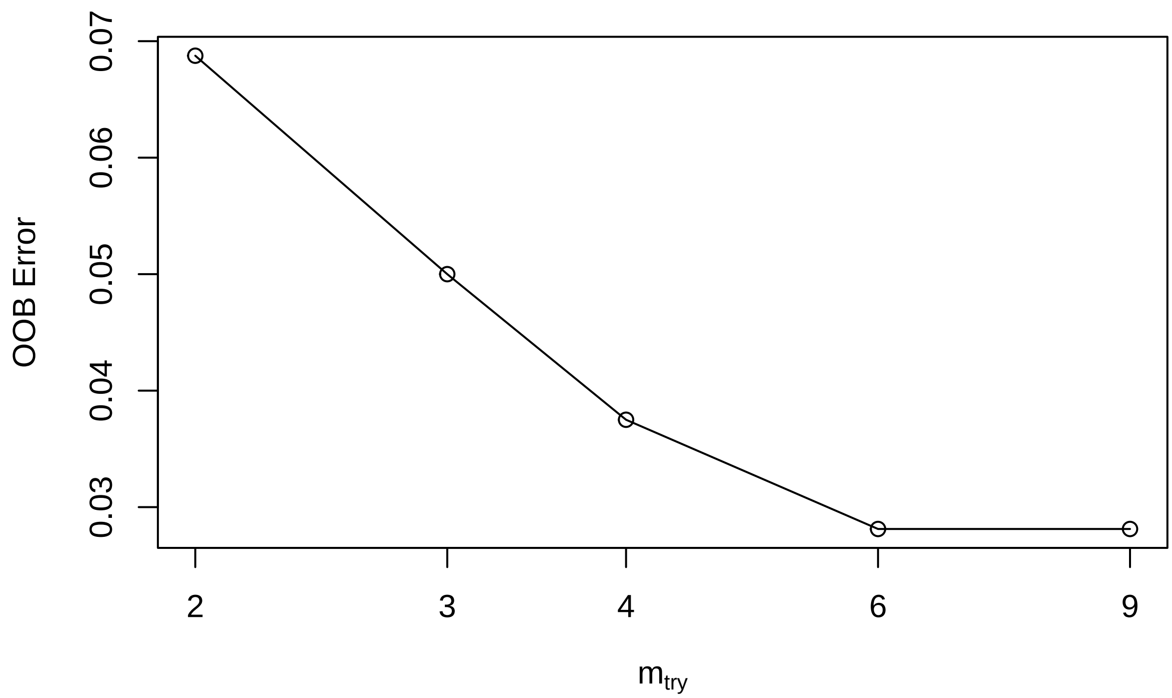
```
##
## Call:
##  randomForest(formula = Veracity ~ ., data = MU3D_Video_Level_Data.scaled)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.06%
## Confusion matrix:
##     1   0 class.error
## 1 151   9     0.05625
## 0   4 156     0.02500
```

```
plot(rf.fit)
```




```
mtry <- tuneRF(MU3D_Video_Level_Data.scaled[,-12],MU3D_Video_Level_Data.scaled$Veracity, ntreeTry=1000,
               stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
```

```
## mtry = 3  OOB error = 5%
## Searching left ...
## mtry = 2     OOB error = 6.88%
## -0.375 0.01
## Searching right ...
## mtry = 4     OOB error = 3.75%
## 0.25 0.01
## mtry = 6     OOB error = 2.81%
## 0.25 0.01
## mtry = 9     OOB error = 2.81%
## 0 0.01
```




```
best.m <- mtry[mtry[,2] == min(mtry[,2]), 1]
print(mtry)
```

```
##       mtry OOBError
## 2.OOB    2 0.068750
## 3.OOB    3 0.050000
## 4.OOB    4 0.037500
## 6.OOB    6 0.028125
## 9.OOB    9 0.028125
```

```
print(best.m)
```

```
## 6.OOB 9.OOB
##     6     9
```

Predict and plot AUC

```
set.seed(123)
rf.fit1 <-randomForest(Veracity~.,data=train_raw.df, mtry=best.m, importance=TRUE,ntree=500)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
```

```
print(rf.fit1)
```

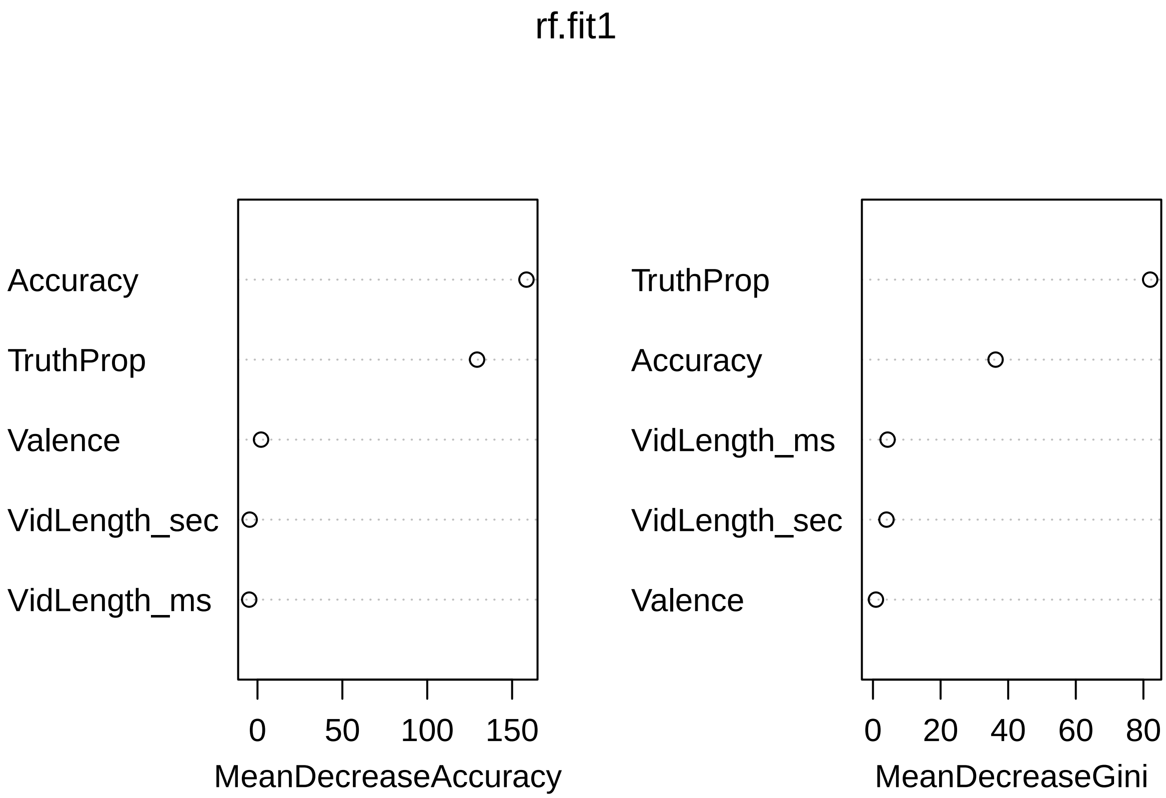
```
##
## Call:
##  randomForest(formula = Veracity ~ ., data = train_raw.df, mtry = best.m,      importance = TRUE, nt
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
```

```
##         OOB estimate of  error rate: 4.3%
## Confusion matrix:
##     1   0 class.error
## 1 123   5   0.0390625
## 0   6 122   0.0468750
```

```
#Evaluate variable importance
importance(rf.fit1)
```

```
##                          1          0 MeanDecreaseAccuracy MeanDecreaseGini
## Valence        -0.06650118   2.936162             2.091551        0.8722462
## VidLength_ms   -4.54595678  -1.724702            -4.867315        4.3295138
## VidLength_sec  -2.92403510  -3.402833            -4.611382        3.9966848
## Accuracy      117.33061436 113.151504           158.442587       36.2682527
## TruthProp      95.13237372  97.564414           129.347039       81.9994120
```

```
varImpPlot(rf.fit1)
```

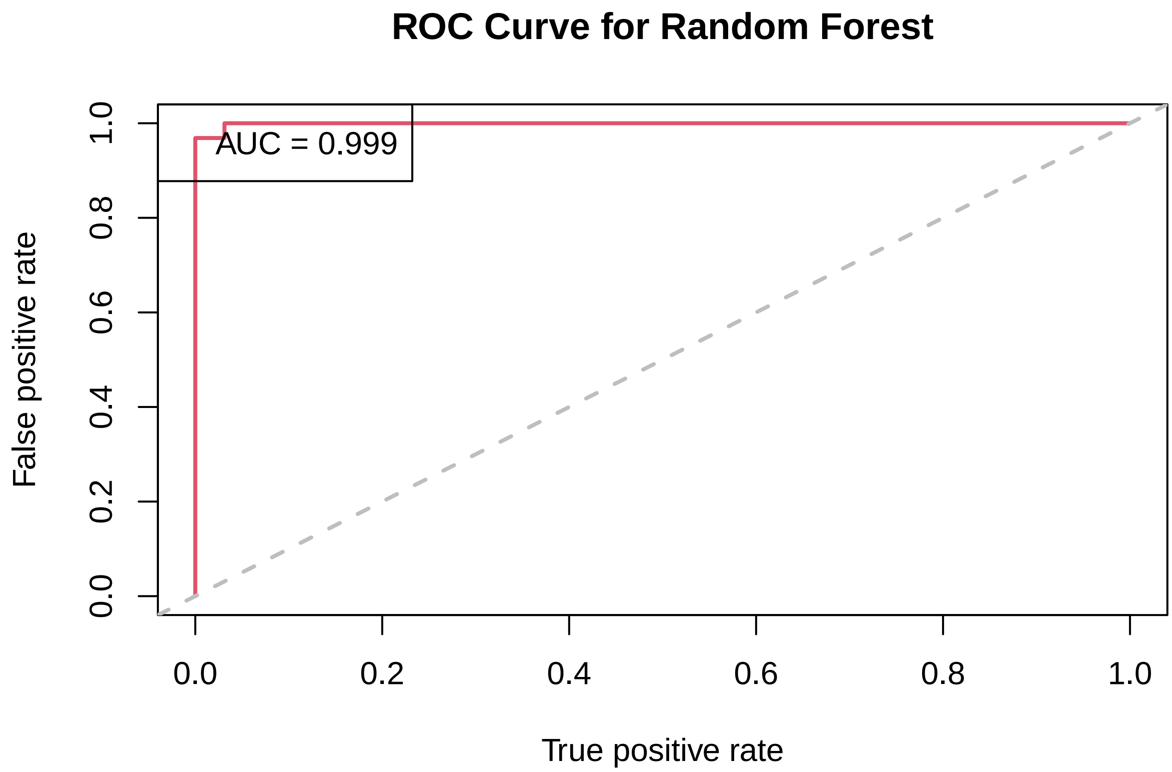



```
pred1=predict(rf.fit1, test_raw.df, type = "prob")
library(ROCR)
perf = prediction(pred1[,2], test_raw.df$Veracity)
# 1. Area under curve
auc = performance(perf, "auc")
1-auc@y.values[[1]]
```

```
## [1] 0.9990234
```

```
# 2. True Positive and Negative Rate
pred3 = performance(perf, "fpr","tpr")
# 3. Plot the ROC curve
plot(pred3,main="ROC Curve for Random Forest",col=2,lwd=2)
```

```
abline(a=0,b=1,lwd=2,lty=2,col="gray")
legend("topleft", c(paste0("AUC = ", round(1-auc@y.values[[1]],4))))
```



Summary confusion matrix

```
rf.pred <- predict(rf.fit1, test_raw.df)
confusionMatrix(rf.pred, test_raw.df$Veracity)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   1  0
##           1 31  0
##           0  1 32
##
##                Accuracy : 0.9844
##                  95% CI : (0.916, 0.9996)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9688
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9688
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9697
```

```
##               Prevalence : 0.5000
##           Detection Rate : 0.4844
##     Detection Prevalence : 0.4844
##        Balanced Accuracy : 0.9844
##
##         'Positive' Class : 1
##
```