Classifying Real vs Fake Instagram Accounts

Capstone Project

Carissa Hicks

```
library(ggplot2)
library(scales)
library(dplyr)
```

Data Exploration

We have 2 separate files of real and fake account data. We will combine these for the analysis.

```
real = read.csv("realAccountData.csv")
fake = read.csv("fakeAccountData.csv")
df = rbind(real, fake)
explore = df
```

```
summary(explore)
```

```
##      isFake       userBiographyLength userFollowerCount userFollowingCount
##  Min.   :0.0000   Min.   :  0.00      Min.   :   0.0    Min.   :   0.0
##  1st Qu.:0.0000   1st Qu.:  0.00      1st Qu.: 152.0    1st Qu.: 267.0
##  Median :0.0000   Median :  7.00      Median : 304.0    Median : 449.0
##  Mean   :0.1675   Mean   : 22.85      Mean   : 369.1    Mean   : 744.3
##  3rd Qu.:0.0000   3rd Qu.: 33.00      3rd Qu.: 481.0    3rd Qu.: 711.0
##  Max.   :1.0000   Max.   :150.00      Max.   :4492.0    Max.   :7497.0
##  userHasProfilPic userIsPrivate    userMediaCount   usernameDigitCount
##  Min.   :0.0000   Min.   :0.0000   Min.   :   0.0   Min.   : 0.0000
##  1st Qu.:1.0000   1st Qu.:0.0000   1st Qu.:   3.0   1st Qu.: 0.0000
##  Median :1.0000   Median :1.0000   Median :  20.0   Median : 0.0000
##  Mean   :0.9229   Mean   :0.6575   Mean   :  57.6   Mean   : 0.4958
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:  67.0   3rd Qu.: 0.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1058.0   Max.   :10.0000
##  usernameLength
##  Min.   : 5.00
##  1st Qu.: 9.00
##  Median :11.00
##  Mean   :11.12
##  3rd Qu.:13.00
##  Max.   :30.00
```

```
str(explore)
```

```
## 'data.frame':    1194 obs. of  9 variables:
##  $ isFake             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ userBiographyLength: int  0 30 9 22 0 26 0 0 51 14 ...
##  $ userFollowerCount  : int  258 263 51 297 113 545 52 1038 172 101 ...
##  $ userFollowingCount : int  238 482 78 480 242 995 121 6640 227 114 ...
##  $ userHasProfilPic   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ userIsPrivate      : int  0 1 1 1 1 1 1 0 0 1 ...
##  $ userMediaCount     : int  0 29 0 25 95 340 30 156 10 4 ...
##  $ usernameDigitCount : int  0 0 0 2 0 0 0 0 0 0 ...
##  $ usernameLength     : int  10 8 10 9 10 8 9 12 10 15 ...
```

There are no missing values in the dataset

```
any(is.na(explore))
```

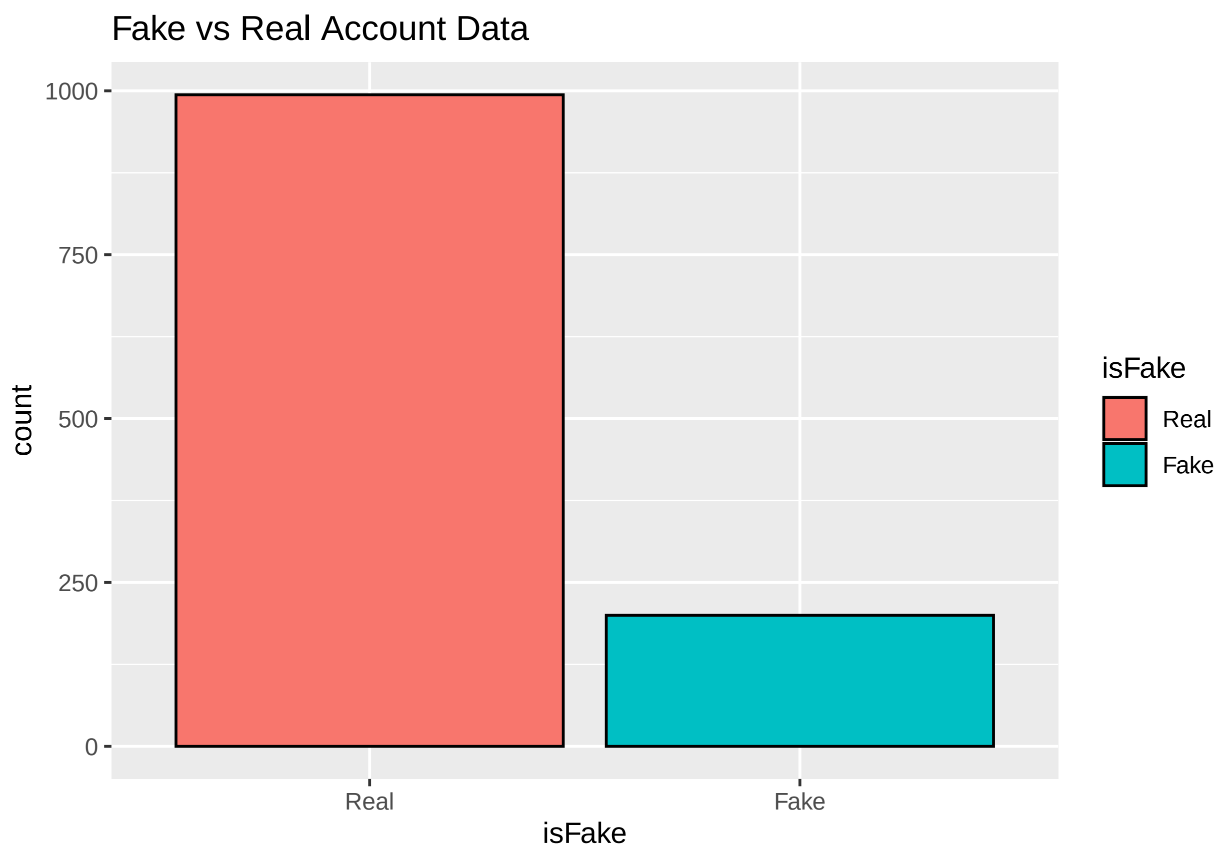
```
## [1] FALSE
```

factoring the categorical data for the visualization aspect

```
cols = c("isFake", "userHasProfilPic", "userIsPrivate")
explore[cols] = lapply(explore[cols], factor)
```

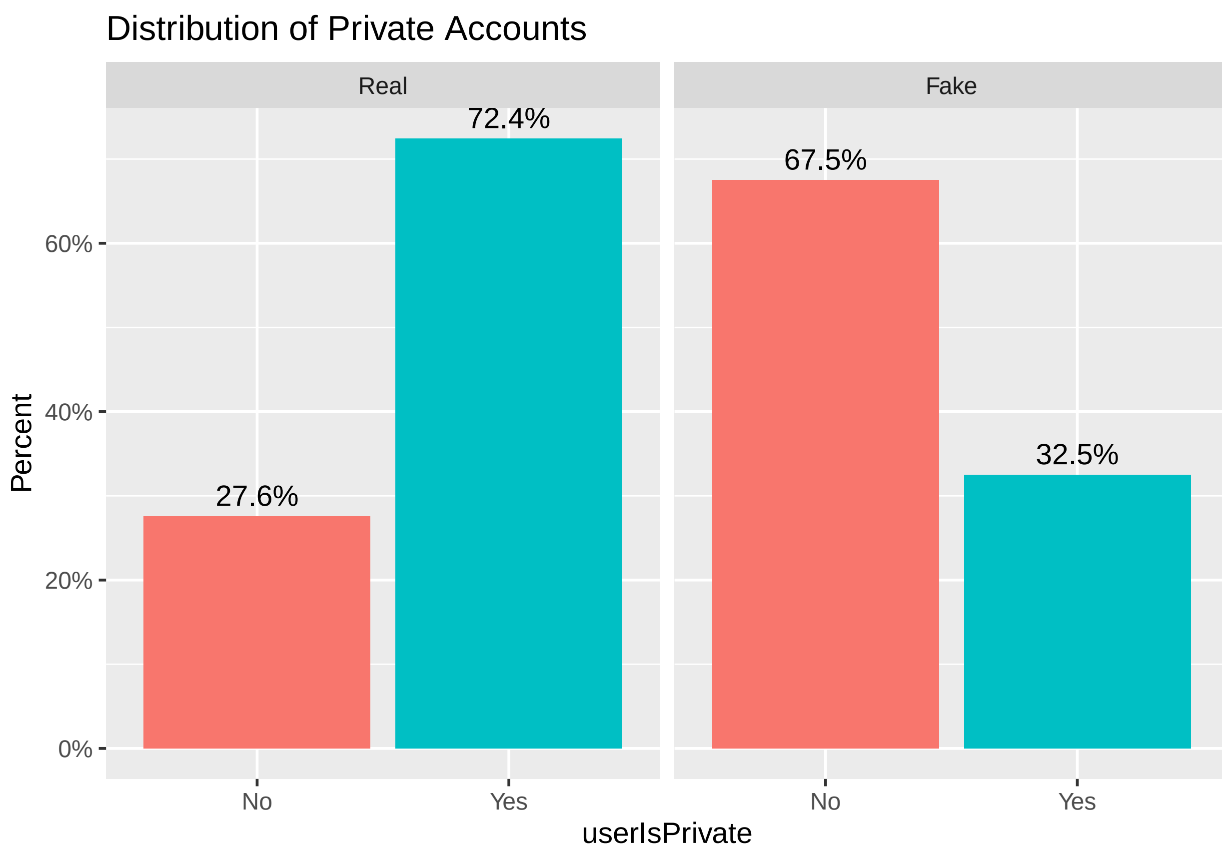
```
levels(explore$isFake) = c("Real", "Fake")
levels(explore$userHasProfilPic) = c("No", "Yes")
levels(explore$userIsPrivate) = c("No", "Yes")
```

```
ggplot(data=explore, aes(x=isFake, fill=isFake))+
  geom_bar(color="black")+
  ggtitle("Fake vs Real Account Data")
```

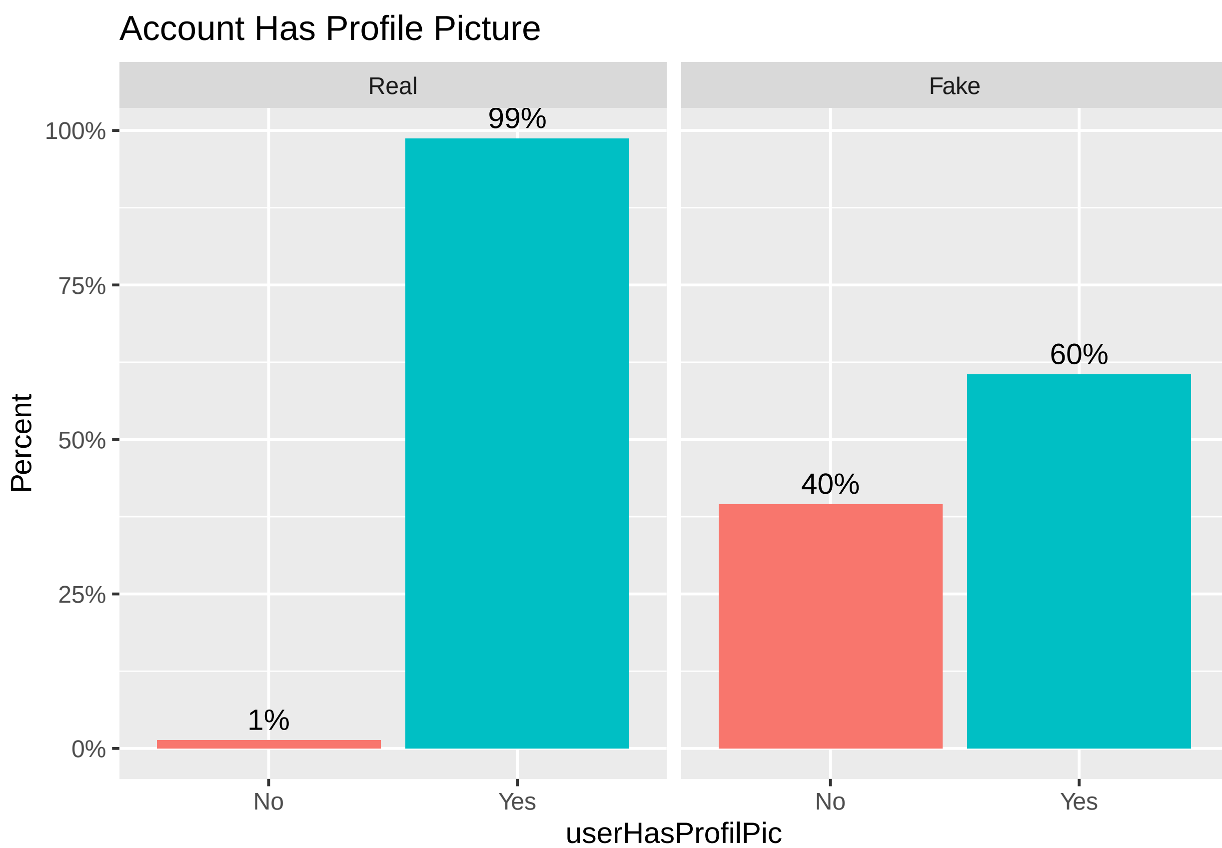



```
ggplot(data=explore, aes(x=userIsPrivate, group = isFake)) +
  geom_bar(aes(y=..prop.., fill = factor(..x..)),stat="count")+
  geom_text(aes(label=scales::percent(..prop..), y=..prop..), stat="count", vjust=-.5)+
  labs(y="Percent", fill="userIsPrivate")+
  facet_grid(~isFake)+scale_y_continuous(labels=scales::percent)+
  ggtitle("Distribution of Private Accounts")+
  theme(legend.position = "none")
```

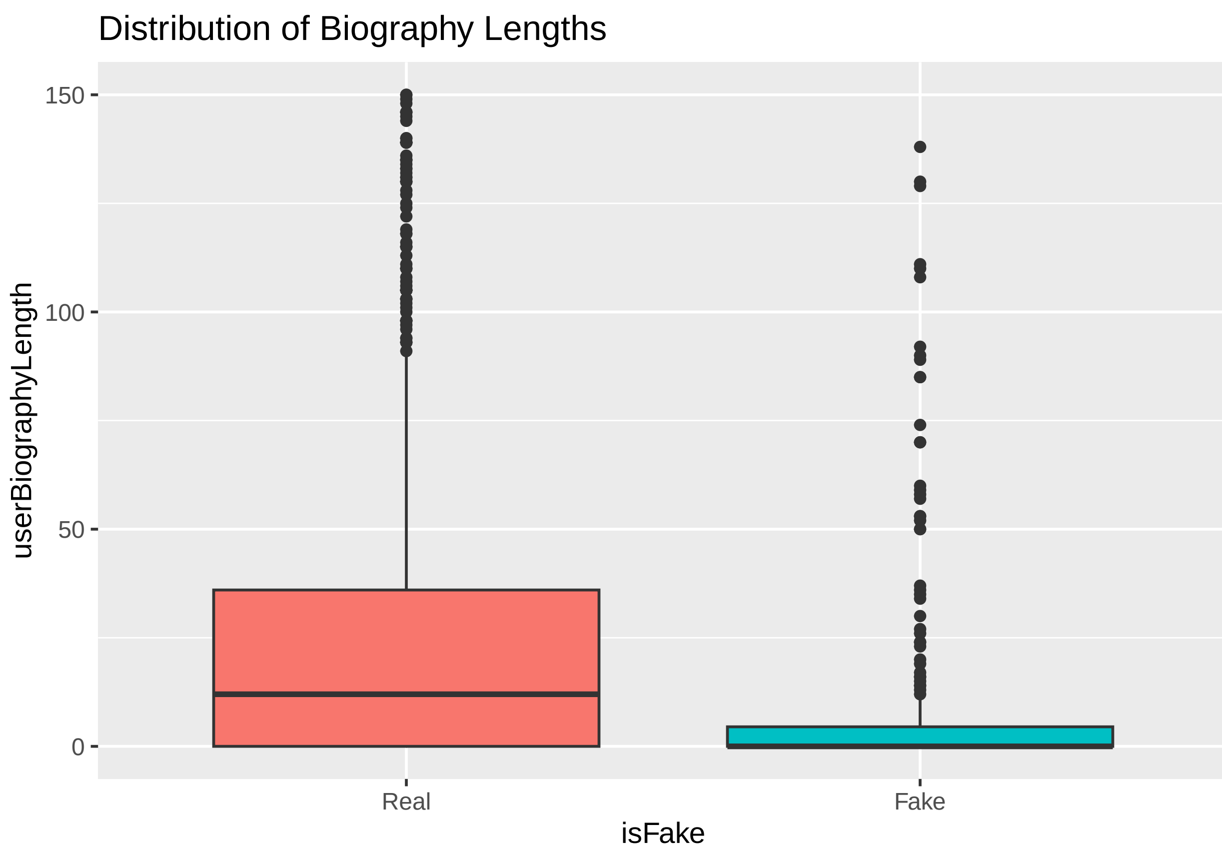
```
## Warning: The dot-dot notation (`..prop..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(prop)` instead.
```



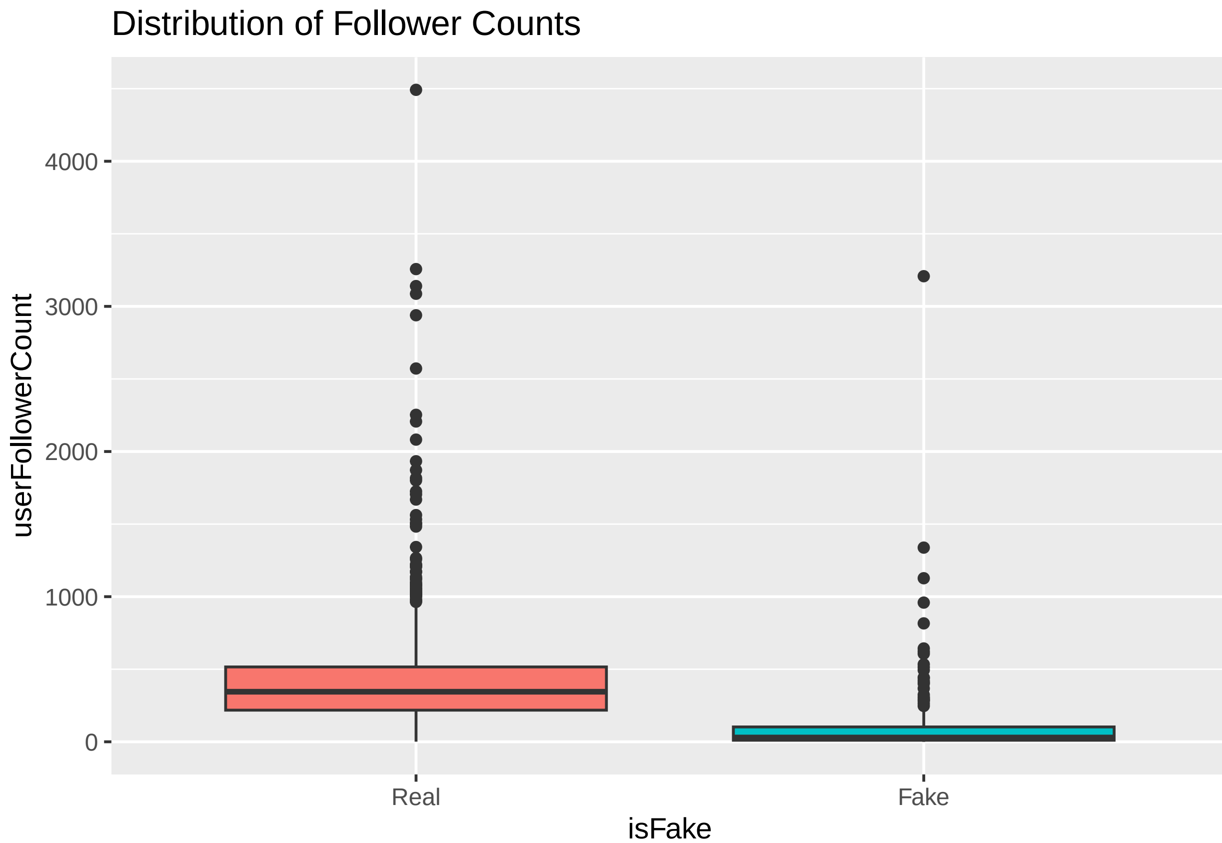

```r
ggplot(data=explore, aes(x=userHasProfilPic, group = isFake)) +
  geom_bar(aes(y=..prop.., fill = factor(..x..)),stat="count")+
  geom_text(aes(label=scales::percent(..prop..), y=..prop..), stat="count", vjust=-.5)+
  labs(y="Percent", fill="userHasProfilPic")+
  facet_grid(~isFake)+scale_y_continuous(labels=scales::percent)+
  ggtitle("Account Has Profile Picture")+
  theme(legend.position = "none")
```



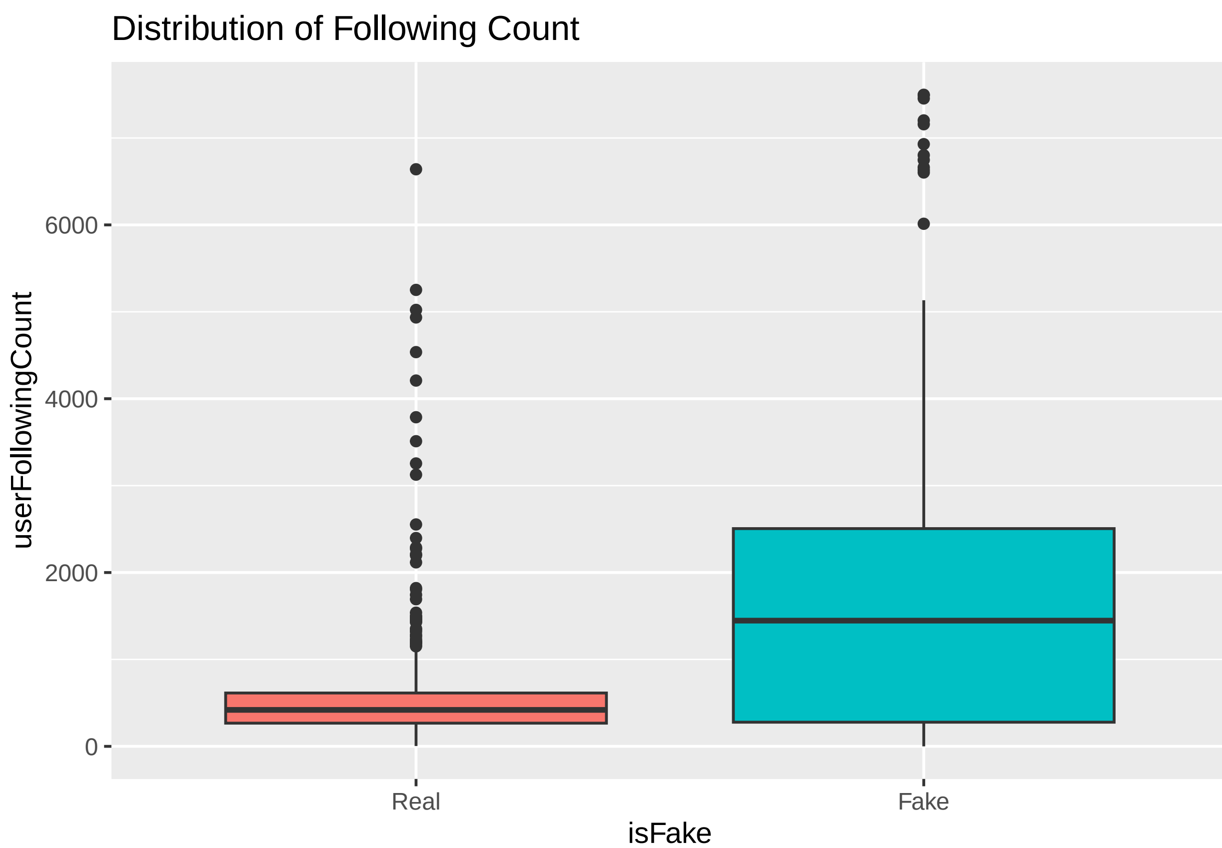

```
ggplot(explore, aes(x=userBiographyLength, y=isFake, fill=isFake))+
  geom_boxplot()+
  coord_flip()+
  ggtitle("Distribution of Biography Lengths")+
  theme(legend.position = "none")
```



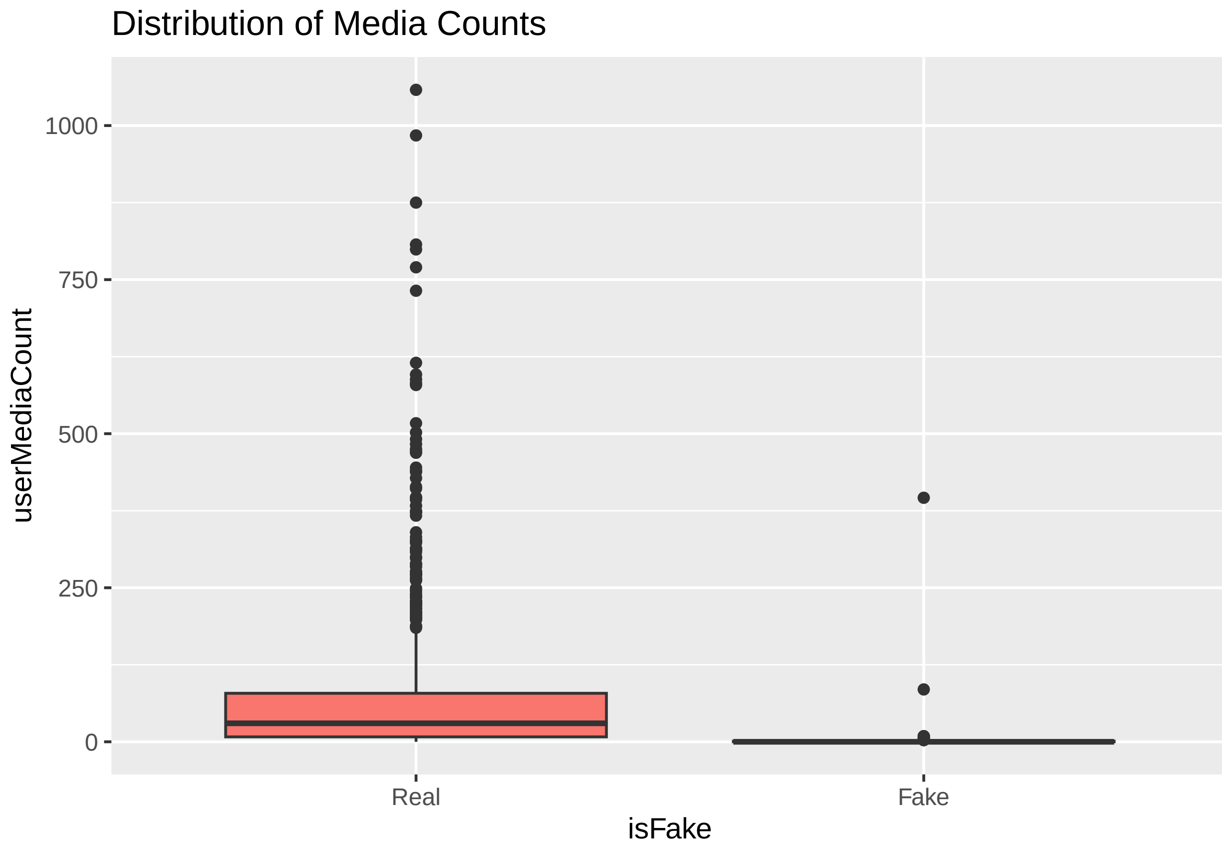

```
ggplot(explore, aes(x=userFollowerCount, y=isFake, fill=isFake))+
  geom_boxplot()+
  coord_flip()+
  ggtitle("Distribution of Follower Counts")+
  theme(legend.position = "none")
```



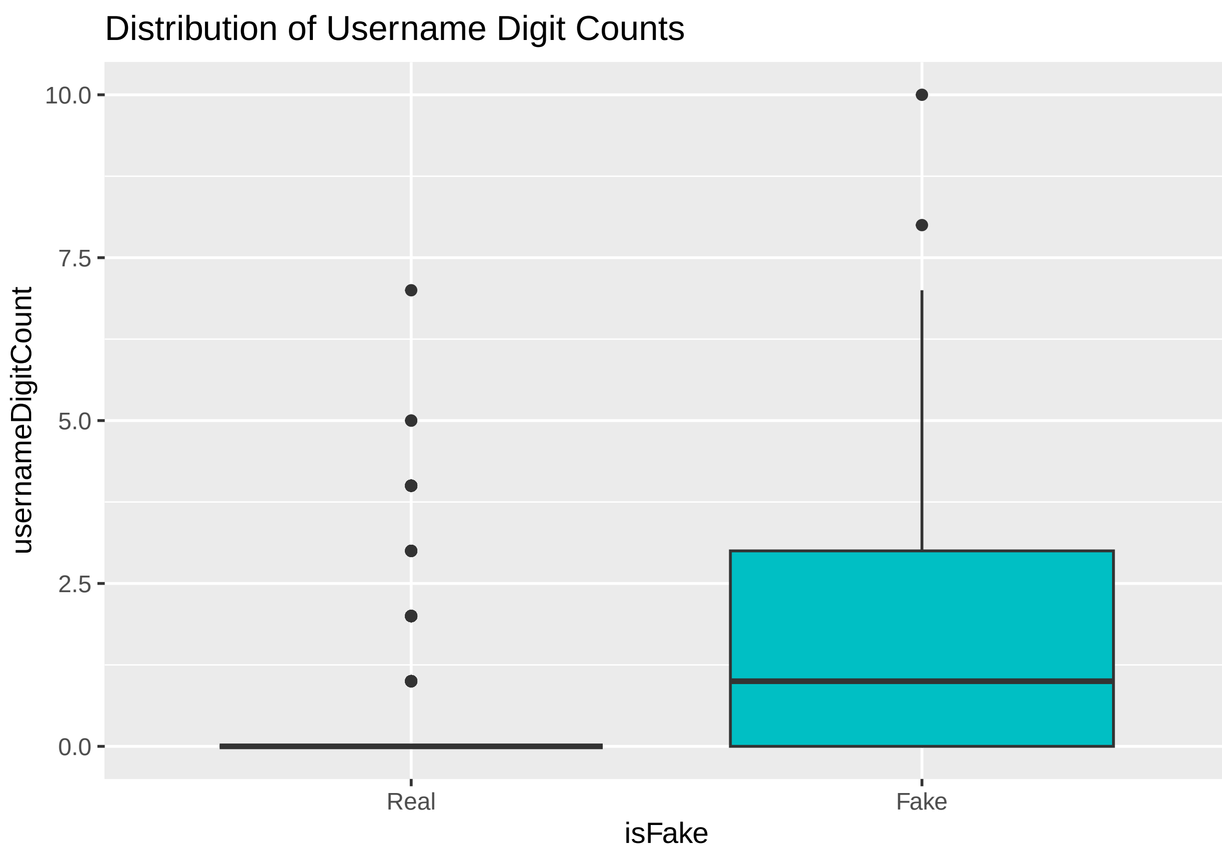

```
ggplot(explore, aes(x=userFollowingCount, y=isFake, fill=isFake))+
  geom_boxplot()+
  coord_flip()+
  ggtitle("Distribution of Following Count")+
  theme(legend.position = "none")
```



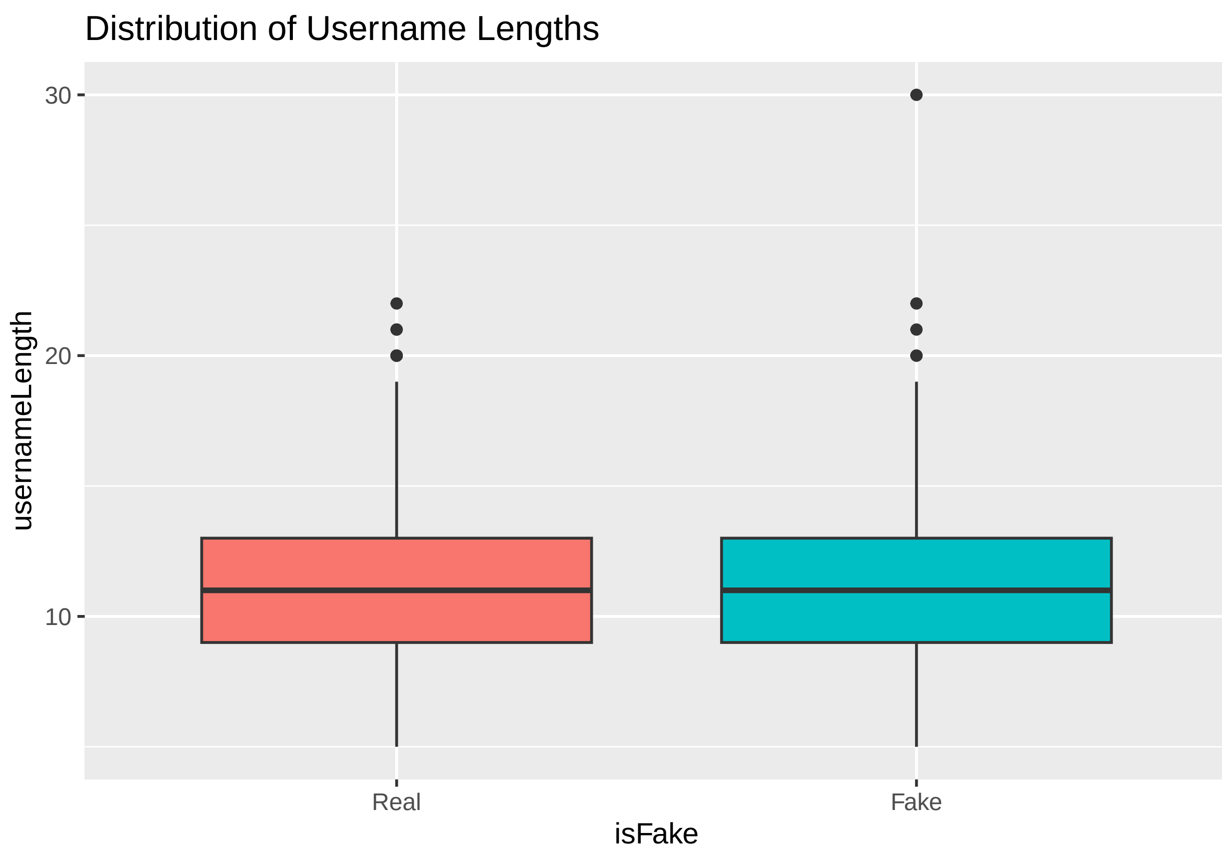

```
ggplot(explore, aes(x=userMediaCount, y=isFake, fill=isFake))+
  geom_boxplot()+
  coord_flip()+
  ggtitle("Distribution of Media Counts")+
  theme(legend.position = "none")
```




```
ggplot(explore, aes(x=usernameDigitCount, y=isFake, fill=isFake))+
  geom_boxplot()+
  coord_flip()+
  ggtitle("Distribution of Username Digit Counts")+
  theme(legend.position = "none")
```




```r
ggplot(explore, aes(x=usernameLength, y=isFake, fill=isFake))+
  geom_boxplot()+
  coord_flip()+
  ggtitle("Distribution of Username Lengths")+
  theme(legend.position = "none")
```



Summary: - Our data is imbalanced, we have more data for real accounts than fake accounts - Fake accounts are more public, real accounts are more private - Almost all real accounts had a profile picture, There was a 40/60 split with the majority of fake accounts having profile pictures - Real accounts had greater biography lengths - Real accounts had more followers - Fake accounts followed more accounts - Real accounts had more posts/media - Fake accounts had a lot more digits in their username - Real and fake accounts had about the same length in usernames

83% of the data is real while 16% is fake

```
table(explore$isFake)/sum(table(explore$isFake))*100
```

```
##
##     Real     Fake
## 83.24958 16.75042
```

```
min(explore$userFollowerCount)
```

```
## [1] 0
```

```
min(explore$userFollowingCount)
```

```
## [1] 0
```

```r
nrow(filter(explore, userFollowerCount == 0))
```

```
## [1] 18
```

```r
nrow(filter(explore, userFollowingCount == 0))
```

```
## [1] 5
```

There are several instances where the userFollowerCount and userFollowingCount is 0. This will be changed to 1 so that the followRatio column can be calculated without any undefined values.

```r
df$userFollowerCount[df$userFollowerCount == 0] = 1
df$userFollowingCount[df$userFollowingCount == 0] = 1
```

```r
nrow(filter(df, userFollowerCount == 0))
```

```
## [1] 0
```

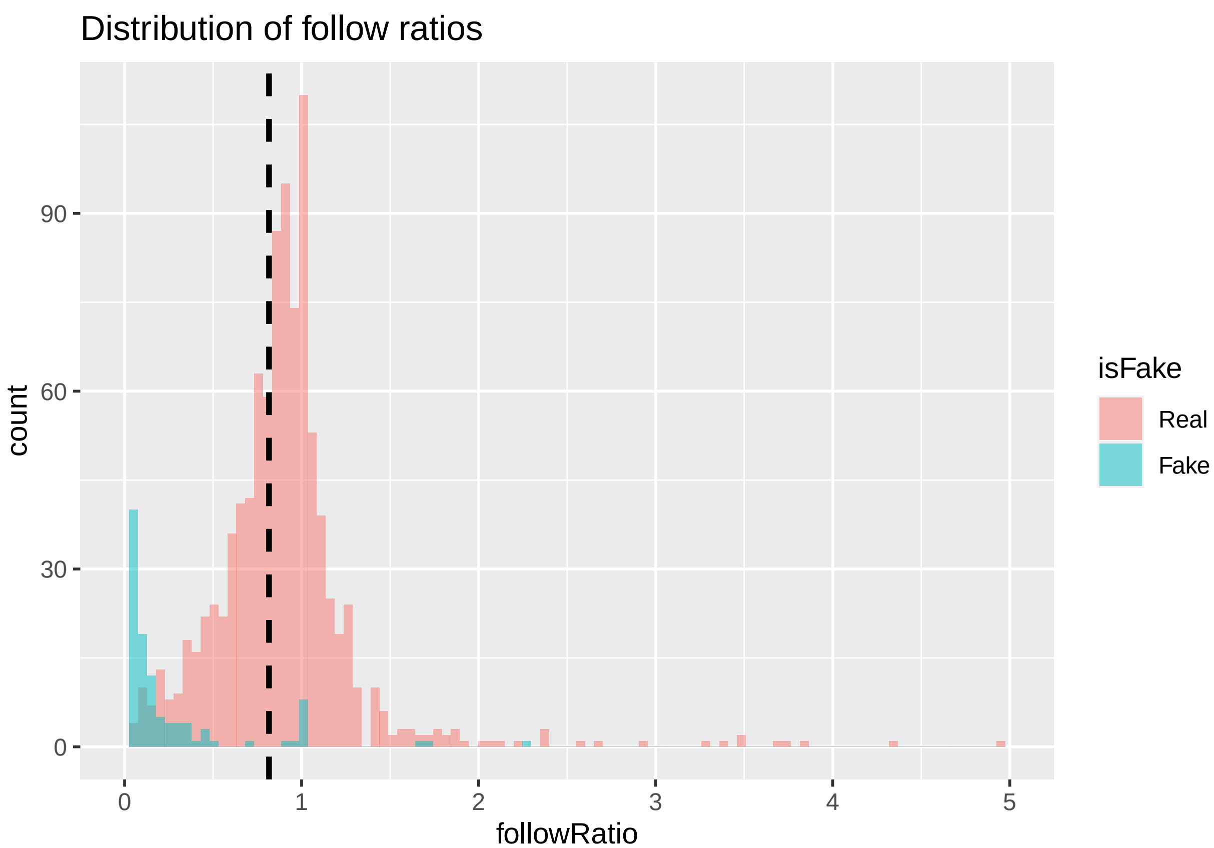
```r
nrow(filter(df, userFollowingCount ==0))
```

```
## [1] 0
```

creating an additional feature that gives the ratio of follower count vs following count

```r
df$followRatio = df$userFollowerCount / df$userFollowingCount
```

```r
explore$followRatio = df$followRatio
ggplot(explore, aes(x=followRatio, fill=isFake))+
  geom_histogram(alpha=0.5, position="identity", bins = 100)+
  ggtitle("Distribution of follow ratios")+
  geom_vline(aes(xintercept=mean(followRatio)), color="black", linetype="dashed", size=1)+
  xlim(0, 5)
```



Exploring the relationships between isFake and the categorical variables using chisquared test and mosaic plot

```
attach(df)
chisq.test(table(isFake, userHasProfilPic))
```

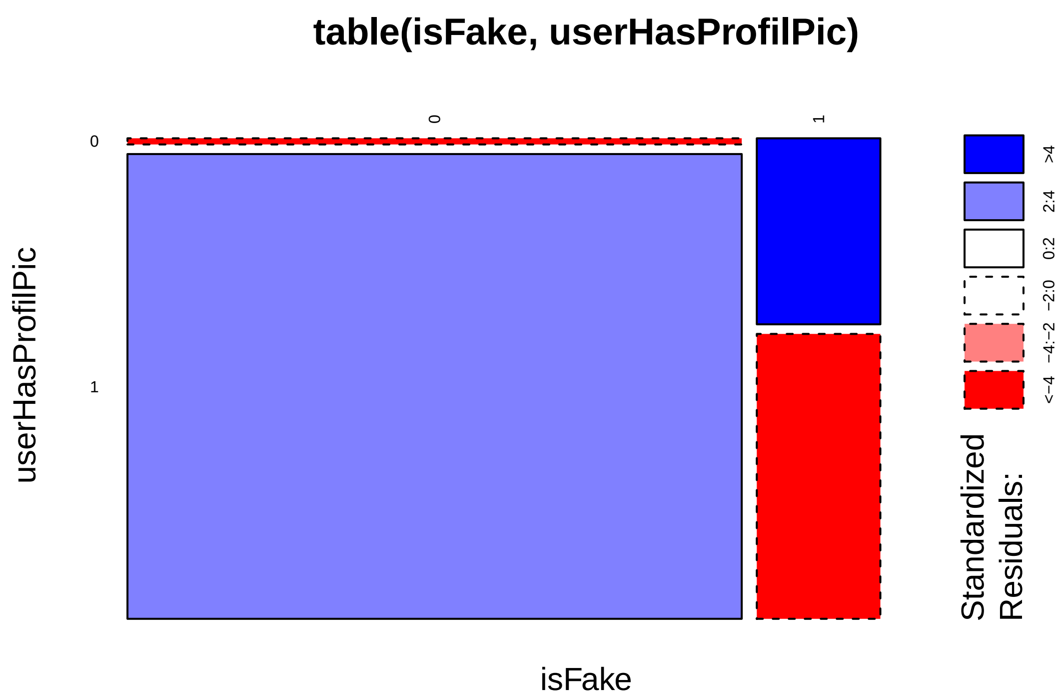
```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(isFake, userHasProfilPic)
## X-squared = 336.16, df = 1, p-value < 2.2e-16
```

```
chisq.test(table(isFake, userIsPrivate))
```

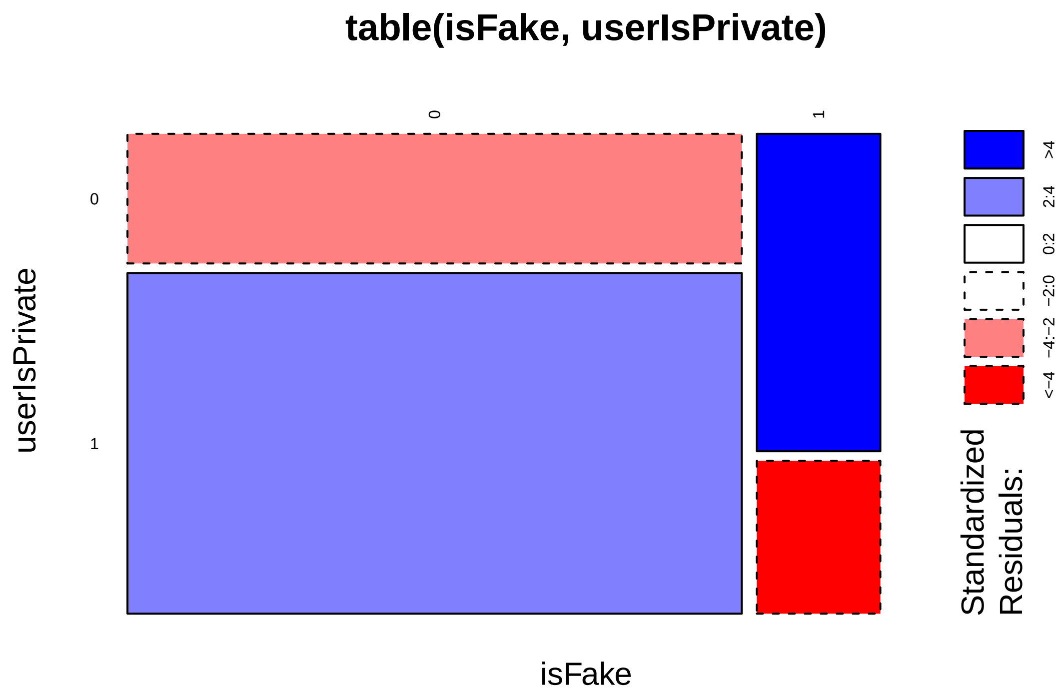
```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(isFake, userIsPrivate)
## X-squared = 116.14, df = 1, p-value < 2.2e-16
```

Using a confidence level of .95 we will use an alpha of 5 percent. Our p-values for these features are much lower than our alpha, therefore they are relevant to our analysis.

```
mosaicplot(table(isFake, userHasProfilPic), shade=TRUE, las=2, cex.axis = 0.5)
```




```
mosaicplot(table(isFake, userIsPrivate), shade=TRUE, las=2, cex.axis = 0.5)
```



t-test for numeric variables.

```
t.test(userBiographyLength~isFake)
```

```
##
##  Welch Two Sample t-test
##
## data:  userBiographyLength by isFake
## t = 5.824, df = 332.31, p-value = 1.356e-08
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##   8.644954 17.463456
## sample estimates:
## mean in group 0 mean in group 1
##        25.03421        11.98000
```

```
t.test(userFollowerCount~isFake)
```

```
##
##  Welch Two Sample t-test
##
## data:  userFollowerCount by isFake
## t = 12.861, df = 340.96, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  256.7971 349.5256
```

```
## sample estimates:
## mean in group 0 mean in group 1
##        419.8913        116.7300
```

```
t.test(userFollowingCount~isFake)
```

```
##
##  Welch Two Sample t-test
##
## data:  userFollowingCount by isFake
## t = -10.214, df = 205.17, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -1624.808 -1099.025
## sample estimates:
## mean in group 0 mean in group 1
##        516.1388       1878.0550
```

```
t.test(userMediaCount~isFake)
```

```
##
##  Welch Two Sample t-test
##
## data:  userMediaCount by isFake
## t = 15.68, df = 1150, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  56.81320 73.06449
## sample estimates:
## mean in group 0 mean in group 1
##        68.47384         3.53500
```

```
t.test(usernameDigitCount~isFake)
```

```
##
##  Welch Two Sample t-test
##
## data:  usernameDigitCount by isFake
## t = -9.9723, df = 215.3, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -1.638866 -1.097935
## sample estimates:
## mean in group 0 mean in group 1
##       0.2665996       1.6350000
```

```
t.test(usernameLength~isFake)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  usernameLength by isFake
## t = -1.2016, df = 254.77, p-value = 0.2306
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -0.8433294  0.2041745
## sample estimates:
## mean in group 0 mean in group 1
##        11.07042        11.39000
```

```
t.test(followRatio~isFake)
```

```
##
##  Welch Two Sample t-test
##
## data:  followRatio by isFake
## t = 14.713, df = 328.09, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  0.6543393 0.8563311
## sample estimates:
## mean in group 0 mean in group 1
##       0.9425634       0.1872282
```

removing usernameLength because it's p-value is less than the alpha

```
df = df[-9]
summary(df)
```

```
##      isFake       userBiographyLength userFollowerCount userFollowingCount
##  Min.   :0.0000   Min.   :  0.00      Min.   :   1.0    Min.   :   1.0
##  1st Qu.:0.0000   1st Qu.:  0.00      1st Qu.: 152.0    1st Qu.: 267.0
##  Median :0.0000   Median :  7.00      Median : 304.0    Median : 449.0
##  Mean   :0.1675   Mean   : 22.85      Mean   : 369.1    Mean   : 744.3
##  3rd Qu.:0.0000   3rd Qu.: 33.00      3rd Qu.: 481.0    3rd Qu.: 711.0
##  Max.   :1.0000   Max.   :150.00      Max.   :4492.0    Max.   :7497.0
##  userHasProfilPic userIsPrivate    userMediaCount   usernameDigitCount
##  Min.   :0.0000   Min.   :0.0000   Min.   :   0.0   Min.   : 0.0000
##  1st Qu.:1.0000   1st Qu.:0.0000   1st Qu.:   3.0   1st Qu.: 0.0000
##  Median :1.0000   Median :1.0000   Median :  20.0   Median : 0.0000
##  Mean   :0.9229   Mean   :0.6575   Mean   :  57.6   Mean   : 0.4958
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:  67.0   3rd Qu.: 0.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1058.0   Max.   :10.0000
##   followRatio
##  Min.   : 0.000398
##  1st Qu.: 0.488116
##  Median : 0.840917
##  Mean   : 0.816042
##  3rd Qu.: 1.003664
##  Max.   :16.800000
```

#Classifying fake accounts using machine learning models

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.2.2
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.2
```

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

```
#randomize the rows
set.seed(123)
df = df[sample(1:nrow(df), replace = FALSE),]
```

splitting the data into about 80% training, 20% testing

```
set.seed(123)

train.index = createDataPartition(df$isFake, p=0.8, list=FALSE)
train = df[train.index, ]
test = df[-train.index, ]
```

1. Logistic Regression

```
model_glm = glm(isFake ~.,
                data=train,
                family="binomial")
```

```
summary(model_glm)
```

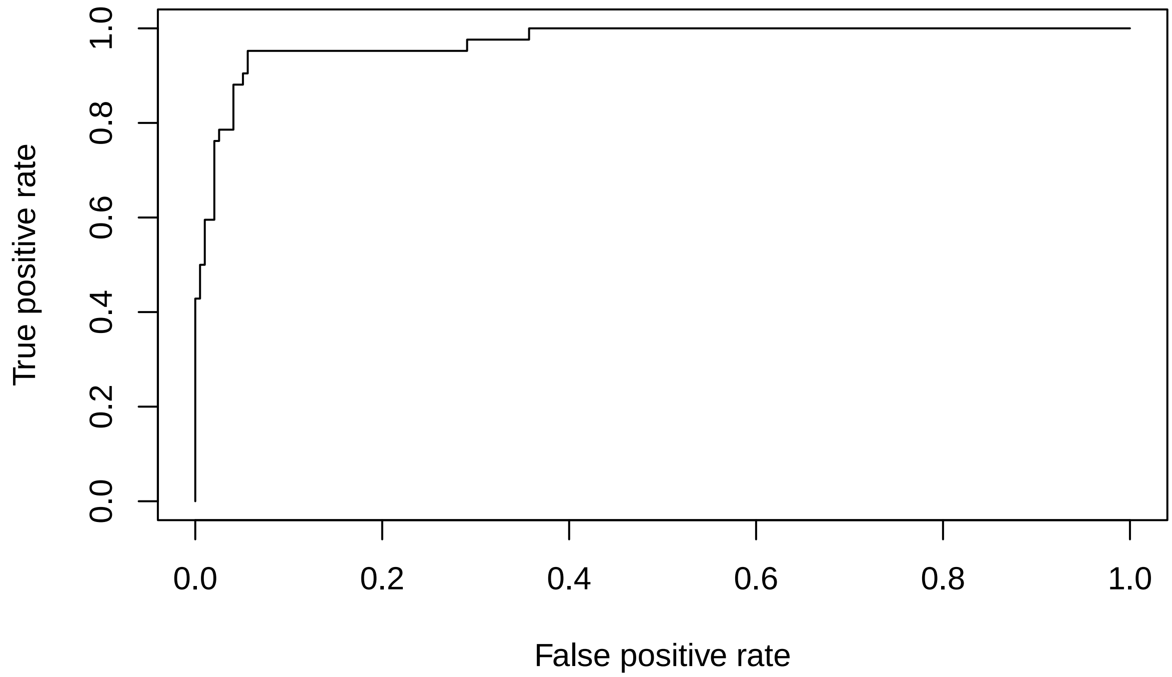
```
##
## Call:
## glm(formula = isFake ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.9129  -0.2153  -0.0860  -0.0095   6.0298
##
```

```
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)          1.6299095  0.5478602   2.975 0.002929 **
## userBiographyLength -0.0085406  0.0049919  -1.711 0.087105 .
## userFollowerCount   -0.0076606  0.0012799  -5.985 2.16e-09 ***
## userFollowingCount   0.0016872  0.0002854   5.912 3.38e-09 ***
## userHasProfilPic    -2.3226030  0.5005493  -4.640 3.48e-06 ***
## userIsPrivate       -0.9897127  0.3536567  -2.799 0.005134 **
## userMediaCount      -0.0207979  0.0058820  -3.536 0.000406 ***
## usernameDigitCount   0.4975744  0.1154804   4.309 1.64e-05 ***
## followRatio         -0.5636176  0.4619845  -1.220 0.222468
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 857.17  on 955  degrees of freedom
## Residual deviance: 268.47  on 947  degrees of freedom
## AIC: 286.47
##
## Number of Fisher Scoring iterations: 8
```

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.2.2
```

```
prob = predict(model_glm, newdata=test, type="response")
pred = prediction(prob, test$isFake)
perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
```




```
predictions = predict(model_glm, test)
pred.label=factor(ifelse(predictions>.5,"Fake", "Real"))
actual.label=factor(ifelse(test$isFake==1, "Fake", "Real"))
c.matrix = confusionMatrix(pred.label, actual.label)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction Fake Real
##       Fake   26    4
##       Real   16  192
##
##                Accuracy : 0.916
##                  95% CI : (0.8732, 0.9479)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 3.717e-05
##
##                   Kappa : 0.6743
##
##  Mcnemar's Test P-Value : 0.01391
##
##             Sensitivity : 0.6190
##             Specificity : 0.9796
##          Pos Pred Value : 0.8667
##          Neg Pred Value : 0.9231
```

```
##               Prevalence : 0.1765
##           Detection Rate : 0.1092
##     Detection Prevalence : 0.1261
##        Balanced Accuracy : 0.7993
##
##         'Positive' Class : Fake
##
```

```
glm_acc = c.matrix$overall[1]
glm_auc = multiclass.roc(as.numeric(test$isFake), as.numeric(predictions))$auc[1]
glm_precision = c.matrix$byClass[5]
glm_recall = c.matrix$byClass[6]
glm_F1 = c.matrix$byclass[7]
```

A. ADASYN

using ADASYN to balance the training data only

```
library(smotefamily)

set.seed(123)

train.adas = ADAS(train,
                  train$isFake,
                  K = 5)
```

```
train.adas = train.adas$data
as.data.frame(table(train.adas$class))
```

```
##   Var1 Freq
## 1    0  798
## 2    1  802
```

```
train.adas = train.adas[-1]
train.adas$class = as.numeric(train.adas$class)
```

```
table(train.adas$class)
```

```
##
##   0   1
## 798 802
```

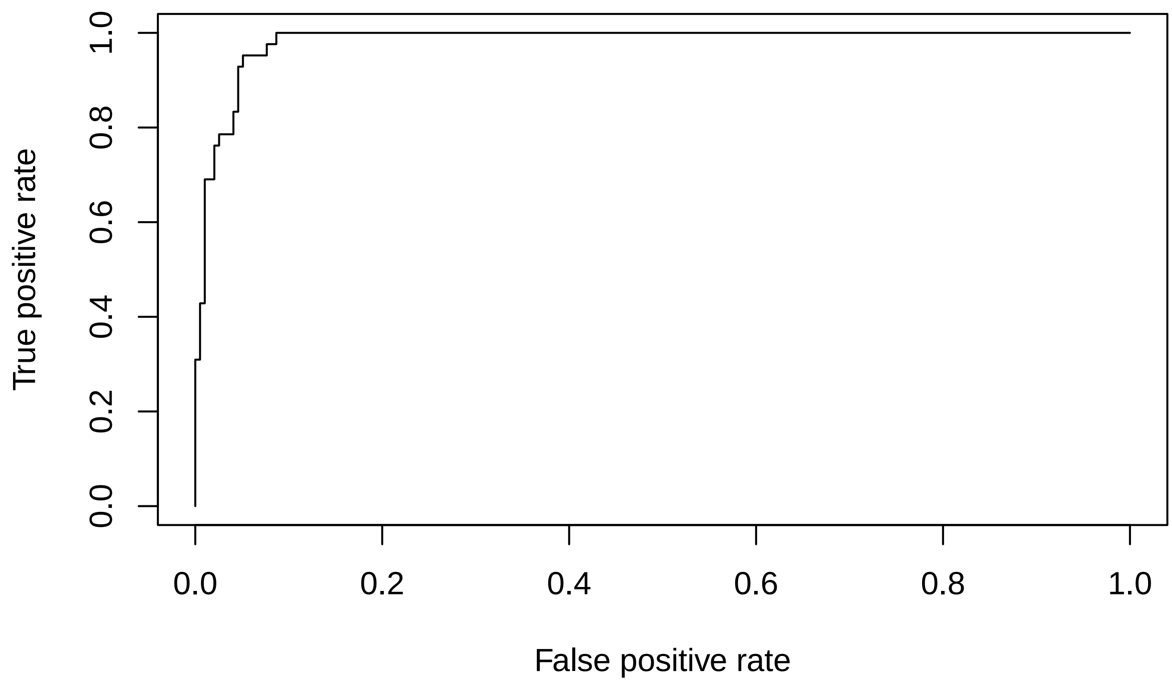
```
model_adas_glm = glm(class~.,
                     data = train.adas,
                     family="binomial")
```

```
summary(model_adas_glm)
```

```
##
## Call:
```

```
## glm(formula = class ~ ., family = "binomial", data = train.adas)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.7793  -0.2362   0.0005   0.1868   4.4429
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)           5.1200496  0.5929464   8.635  < 2e-16 ***
## userBiographyLength -0.0081143  0.0034258  -2.369   0.0179 *
## userFollowerCount   -0.0035879  0.0003031 -11.838  < 2e-16 ***
## userFollowingCount   0.0017190  0.0001907   9.014  < 2e-16 ***
## userHasProfilPic    -4.7669097  0.5649720  -8.437  < 2e-16 ***
## userIsPrivate       -1.8324671  0.2427056  -7.550 4.35e-14 ***
## userMediaCount      -0.0507210  0.0066776  -7.596 3.06e-14 ***
## usernameDigitCount   0.6039685  0.0912412   6.619 3.60e-11 ***
## followRatio         -0.0520363  0.0910866  -0.571   0.5678
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2218.06  on 1599  degrees of freedom
## Residual deviance:  684.37  on 1591  degrees of freedom
## AIC: 702.37
##
## Number of Fisher Scoring iterations: 8
```

```
prob = predict(model_adas_glm, newdata=test, type="response")
pred = prediction(prob, test$isFake)
perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
```




```
predictions = predict(model_adas_glm, test)
pred.label=factor(ifelse(predictions>.5,"Fake", "Real"))
actual.label=factor(ifelse(test$isFake==1, "Fake", "Real"))
c.matrix = confusionMatrix(pred.label, actual.label)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   36    9
##       Real    6  187
##
##                Accuracy : 0.937
##                  95% CI : (0.8982, 0.9643)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 2.455e-07
##
##                   Kappa : 0.7891
##
##  Mcnemar's Test P-Value : 0.6056
##
##             Sensitivity : 0.8571
##             Specificity : 0.9541
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.9689
```

```
##               Prevalence : 0.1765
##           Detection Rate : 0.1513
##     Detection Prevalence : 0.1891
##        Balanced Accuracy : 0.9056
##
##         'Positive' Class : Fake
##
```

```
glm2_acc = c.matrix$overall[1]
glm2_auc = multiclass.roc(as.numeric(test$isFake), as.numeric(predictions))$auc[1]
glm2_precision = c.matrix$byClass[5]
glm2_recall = c.matrix$byClass[6]
glm2_F1 = c.matrix$byclass[7]
```

B. Cross Validation and ADASYN

```
train.adas$class[train.adas$class == 1] = "Fake"
train.adas$class[train.adas$class == 0] = "Real"

test$isFake[test$isFake == 1] = "Fake"
test$isFake[test$isFake == 0] = "Real"

train.adas$class = as.factor(train.adas$class)
test$isFake = as.factor(test$isFake)
```

```
table(train.adas$class)
```

```
##
## Fake Real
##  802  798
```

```
table(test$isFake)
```

```
##
## Fake Real
##   42  196
```

```
ctrlspecs = trainControl(method="cv",
                         number = 5,
                         savePredictions = "all",
                         classProbs = TRUE)
```

```
set.seed(123)

model_adas_cv_glm = train(class~.,
                          data = train.adas,
                          method = "glm",
                          family = binomial,
                          trControl = ctrlspecs)
print(model_adas_cv_glm)
```

```
## Generalized Linear Model
##
## 1600 samples
##    8 predictor
##    2 classes: 'Fake', 'Real'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1280, 1281, 1279
## Resampling results:
##
##   Accuracy   Kappa
##   0.9118864  0.8237658
```

```
summary(model_adas_cv_glm)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.4429  -0.1868  -0.0005   0.2362   4.7793
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -5.1200496  0.5929464  -8.635  < 2e-16 ***
## userBiographyLength  0.0081143  0.0034258   2.369   0.0179 *
## userFollowerCount    0.0035879  0.0003031  11.838  < 2e-16 ***
## userFollowingCount  -0.0017190  0.0001907  -9.014  < 2e-16 ***
## userHasProfilPic     4.7669097  0.5649720   8.437  < 2e-16 ***
## userIsPrivate        1.8324671  0.2427056   7.550 4.35e-14 ***
## userMediaCount       0.0507210  0.0066776   7.596 3.06e-14 ***
## usernameDigitCount  -0.6039685  0.0912412  -6.619 3.60e-11 ***
## followRatio          0.0520363  0.0910866   0.571   0.5678
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2218.06  on 1599  degrees of freedom
## Residual deviance:  684.37  on 1591  degrees of freedom
## AIC: 702.37
##
## Number of Fisher Scoring iterations: 8
```

```
varImp(model_adas_cv_glm)
```

```
## glm variable importance
##
##                     Overall
## userFollowerCount    100.00
## userFollowingCount    74.93
```

```
## userHasProfilPic      69.82
## userMediaCount        62.35
## userIsPrivate         61.94
## usernameDigitCount    53.68
## userBiographyLength   15.95
## followRatio            0.00
```

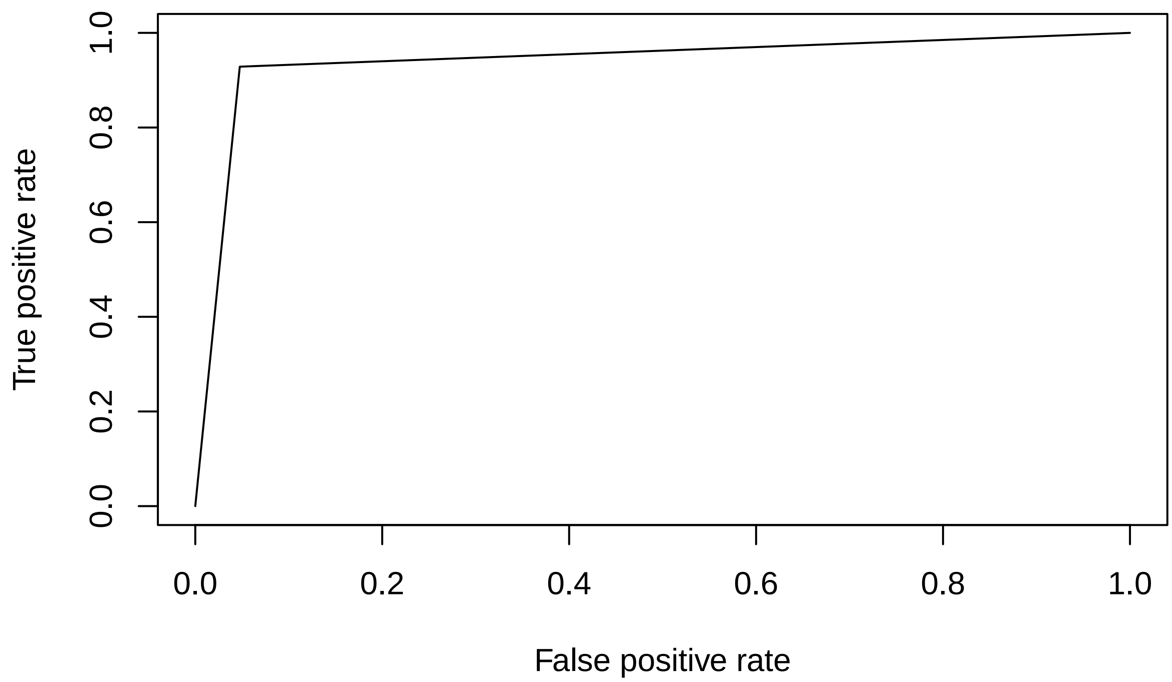
It seems like userBiographyLength and followRatio have no importance to the model.

```r
predictions = predict(model_adas_cv_glm, newdata=test)
```

```r
confusionMatrix(data=predictions, test$isFake)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   40   14
##       Real    2  182
##
##                Accuracy : 0.9328
##                  95% CI : (0.8931, 0.9611)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 7.563e-07
##
##                   Kappa : 0.792
##
##  Mcnemar's Test P-Value : 0.00596
##
##             Sensitivity : 0.9524
##             Specificity : 0.9286
##          Pos Pred Value : 0.7407
##          Neg Pred Value : 0.9891
##              Prevalence : 0.1765
##          Detection Rate : 0.1681
##    Detection Prevalence : 0.2269
##       Balanced Accuracy : 0.9405
##
##        'Positive' Class : Fake
##
```

```r
prob = predict(model_adas_cv_glm, newdata=test)
pred = prediction(as.numeric(prob), as.numeric(test$isFake))
perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
```




```
glm3_acc = c.matrix$overall[1]
glm3_auc = multiclass.roc(as.numeric(test$isFake), as.numeric(predictions))$auc[1]
glm3_precision = c.matrix$byClass[5]
glm3_recall = c.matrix$byClass[6]
glm3_F1 = c.matrix$byclass[7]
```

C. Feature Scaling

I will normalize the non-factor variables using min-max normalization

```
set.seed(123)
normalize = function(x){
  return ((x - min(x)) / max(x) - min(x))
}
```

```
table(train.adas$class)
```

```
##
## Fake Real
##  802  798
```

```
table(test$isFake)
```

```
##
## Fake Real
##   42  196
```

```
train.adas$class = as.numeric(train.adas$class)
test$isFake = as.numeric(test$isFake)
```

```
table(test$isFake)
```

```
##
##   1   2
##  42 196
```

```
table(train.adas$class)
```

```
##
##   1   2
## 802 798
```

```
test$isFake[test$isFake == 2] = 0
train.adas$class[train.adas$class == 2] = 0

table(test$isFake)
```

```
##
##   0   1
## 196  42
```

```
table(train.adas$class)
```

```
##
##   0   1
## 798 802
```

```
scale.cols.train = c(1, 2, 3, 6, 7, 8)
scale.cols.test = c(2, 3, 4, 7, 8, 9)

train.adas.norm = train.adas
test.norm = test

train.adas.norm[scale.cols.train] = lapply(train.adas.norm[scale.cols.train], normalize)
test.norm[scale.cols.test] = lapply(test.norm[scale.cols.test], normalize)
```

```
str(train.adas.norm)
```

```
## 'data.frame':    1600 obs. of  9 variables:
##  $ userBiographyLength: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ userFollowerCount  : num  -0.998 -0.999 -0.969 -0.989 -0.937 ...
##  $ userFollowingCount : num  -0.376 -0.99 -0.853 -0.669 -0.664 ...
##  $ userHasProfilPic   : num  0 1 1 1 1 1 0 1 1 1 ...
```

```
##  $ userIsPrivate      : num  0 0 0 0 0 0 1 1 0 1 ...
##  $ userMediaCount     : num  0 0 0.00343 0 0.00571 ...
##  $ usernameDigitCount : num  0.375 0.5 0 0 0.25 0.375 0.5 0.25 0.375 0 ...
##  $ followRatio        : num  -0.000154 0.007669 0.015404 0.002174 0.013557 ...
##  $ class              : num  1 1 1 1 1 1 1 1 1 1 ...
```

```
str(test.norm)
```

```
## 'data.frame':    238 obs. of  9 variables:
##  $ isFake             : num  0 0 1 1 0 0 0 0 0 0 ...
##  $ userBiographyLength: num  0.22 0.193 0 0.173 0 ...
##  $ userFollowerCount  : num  -0.788 -0.88 -1 -0.799 -0.99 ...
##  $ userFollowingCount : num  -0.91 -0.951 -0.993 -0.952 -0.994 ...
##  $ userHasProfilPic   : int  1 1 0 0 1 1 1 1 1 1 ...
##  $ userIsPrivate      : int  0 1 1 0 0 0 1 0 1 0 ...
##  $ userMediaCount     : num  0.02268 0.00945 0 0.00189 0.00378 ...
##  $ usernameDigitCount : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ followRatio        : num  5.63e-02 5.86e-02 2.83e-05 1.01e-01 3.86e-02 ...
```

```
table(test.norm$isFake)
```

```
##
##   0   1
## 196  42
```

```
table(train.adas.norm$class)
```

```
##
##   0   1
## 798 802
```

```
train.adas.norm$class[train.adas.norm$class == 1] = "Fake"
train.adas.norm$class[train.adas.norm$class == 0] = "Real"
```

```
test.norm$isFake[test.norm$isFake == 1] = "Fake"
test.norm$isFake[test.norm$isFake == 0] = "Real"
```

```
train.adas.norm$class = as.factor(train.adas.norm$class)
test.norm$isFake = as.factor(test.norm$isFake)
```

```
table(test.norm$isFake)
```

```
##
## Fake Real
##   42  196
```

```
table(train.adas.norm$class)
```

```
##
## Fake Real
##  802  798
```

```r
set.seed(123)

glm_norm = train(class~.,
                          data = train.adas.norm,
                          method = "glm",
                          family = binomial,
                          trControl = ctrlspecs)
```

```r
summary(glm_norm)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.4429  -0.1868  -0.0005   0.2362   4.7793
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -1.8816     1.2011  -1.567   0.1172
## userBiographyLength    1.2172     0.5139   2.369   0.0179 *
## userFollowerCount     16.1170     1.3614  11.838  < 2e-16 ***
## userFollowingCount   -12.8807     1.4289  -9.014  < 2e-16 ***
## userHasProfilPic       4.7669     0.5650   8.437  < 2e-16 ***
## userIsPrivate          1.8325     0.2427   7.550 4.35e-14 ***
## userMediaCount        44.3809     5.8429   7.596 3.06e-14 ***
## usernameDigitCount    -4.8317     0.7299  -6.619 3.60e-11 ***
## followRatio            0.4163     0.7287   0.571   0.5678
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2218.06  on 1599  degrees of freedom
## Residual deviance:  684.37  on 1591  degrees of freedom
## AIC: 702.37
##
## Number of Fisher Scoring iterations: 8
```

```r
varImp(glm_norm)
```

```
## glm variable importance
##
##                      Overall
## userFollowerCount     100.00
## userFollowingCount     74.93
## userHasProfilPic       69.82
## userMediaCount         62.35
## userIsPrivate          61.94
## usernameDigitCount     53.68
## userBiographyLength    15.95
## followRatio             0.00
```

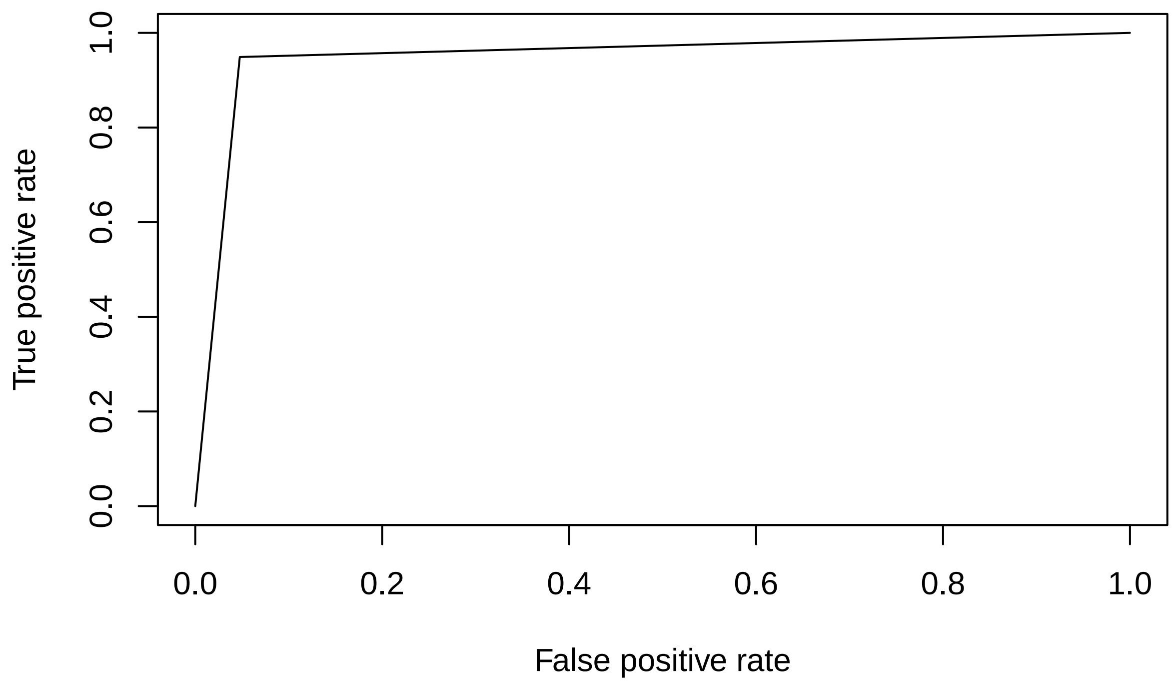
Even with feature scaling, followRatio and userBiographyLength still had no variable importance to the model.

```
predictions = predict(glm_norm, newdata=test.norm)
```

```
confusionMatrix(data=predictions, test.norm$isFake)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   40   10
##       Real    2  186
##
##                Accuracy : 0.9496
##                  95% CI : (0.9136, 0.9737)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 5.482e-09
##
##                   Kappa : 0.8386
##
##  Mcnemar's Test P-Value : 0.04331
##
##             Sensitivity : 0.9524
##             Specificity : 0.9490
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.9894
##              Prevalence : 0.1765
##          Detection Rate : 0.1681
##    Detection Prevalence : 0.2101
##       Balanced Accuracy : 0.9507
##
##        'Positive' Class : Fake
##
```

```
prob = predict(glm_norm, newdata=test.norm)
pred = prediction(as.numeric(prob), as.numeric(test.norm$isFake))
perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
```




```
glm4_acc = c.matrix$overall[1]
glm4_auc = multiclass.roc(as.numeric(test.norm$isFake), as.numeric(predictions))$auc[1]
glm4_precision = c.matrix$byClass[5]
glm4_recall = c.matrix$byClass[6]
glm4_F1 = c.matrix$byclass[7]
```

* Logistic Regression Results

```
log_reg_performance = matrix(c(glm_acc, glm_auc, glm_precision, glm_recall, glm_F1,
                               glm2_acc, glm2_auc, glm2_precision, glm2_recall, glm2_F1,
                               glm3_acc, glm3_auc, glm3_precision, glm3_recall, glm3_F1,
                               glm4_acc, glm4_auc, glm4_precision, glm4_recall, glm4_F1
                               ),
                             ncol = 5, byrow = FALSE)

colnames(log_reg_performance) = c("Accuracy", "AUC", "Precision", "Recall", "F1")
rownames(log_reg_performance) = c("LR",
                                  "LR ADASYN",
                                  "LR ADASYN CV",
                                  "LR ADASYN CV NORM")
as.table(log_reg_performance)
```

```
##                    Accuracy       AUC Precision    Recall        F1
## LR                0.9159664 0.9369748 0.9369748 0.9369748 0.9159664
## LR ADASYN         0.9714529 0.9832362 0.9404762 0.9506803 0.9714529
```

```
## LR ADASYN CV      0.8666667 0.8000000 0.8000000 0.8000000 0.8666667
## LR ADASYN CV NORM 0.6190476 0.8571429 0.8571429 0.8571429 0.6190476
```

2. Simple SVM

```
table(train.adas.norm$class)
```

```
##
## Fake Real
##  802  798
```

```
table(test.norm$isFake)
```

```
##
## Fake Real
##   42  196
```

```
set.seed(123)
t.grid = expand.grid(C = seq(0,2,length=20))
model_svm = train(class~.,
                  data = train.adas.norm,
                  method = "svmLinear",
                  trControl = ctrlspecs,
                  tuneGrid = t.grid)
model_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 1600 samples
##    8 predictor
##    2 classes: 'Fake', 'Real'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1280, 1281, 1279
## Resampling results across tuning parameters:
##
##   C          Accuracy   Kappa
##   0.0000000        NaN        NaN
##   0.1052632  0.9156345  0.8312697
##   0.2105263  0.9175134  0.8350283
##   0.3157895  0.9181365  0.8362739
##   0.4210526  0.9181404  0.8362783
##   0.5263158  0.9175154  0.8350312
##   0.6315789  0.9181423  0.8362871
##   0.7368421  0.9206384  0.8412772
##   0.8421053  0.9200154  0.8400326
##   0.9473684  0.9187615  0.8375239
##   1.0526316  0.9193904  0.8387862
##   1.1578947  0.9225154  0.8450326
```

```
##   1.2631579  0.9206384  0.8412800
##   1.3684211  0.9225154  0.8450362
##   1.4736842  0.9225154  0.8450362
##   1.5789474  0.9225154  0.8450362
##   1.6842105  0.9231404  0.8462844
##   1.7894737  0.9231404  0.8462826
##   1.8947368  0.9218923  0.8437888
##   2.0000000  0.9218904  0.8437844
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.684211.
```

```r
varImp(model_svm)
```

```
## ROC curve variable importance
##
##                     Importance
## userMediaCount          100.00
## followRatio              80.33
## userFollowerCount        77.59
## userHasProfilPic         67.40
## usernameDigitCount       64.01
## userIsPrivate            46.31
## userBiographyLength      23.27
## userFollowingCount        0.00
```

```r
predictions = predict(model_svm, test.norm)
c.matrix = confusionMatrix(predictions, test.norm$isFake)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   36    9
##       Real    6  187
##
##                Accuracy : 0.937
##                  95% CI : (0.8982, 0.9643)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 2.455e-07
##
##                   Kappa : 0.7891
##
##  Mcnemar's Test P-Value : 0.6056
##
##             Sensitivity : 0.8571
##             Specificity : 0.9541
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.9689
##              Prevalence : 0.1765
##          Detection Rate : 0.1513
##    Detection Prevalence : 0.1891
```

```
##       Balanced Accuracy : 0.9056
##
##        'Positive' Class : Fake
##
```

```
svm_acc = c.matrix$overall[1]
svm_auc = multiclass.roc(as.numeric(test.norm$isFake), as.numeric(predictions))$auc[1]
```

```
## Setting direction: controls < cases
```

```
svm_precision = c.matrix$byClass[5]
svm_recall = c.matrix$byClass[6]
svm_F1 = c.matrix$byclass[7]
```

A. Without the normalized data

```
table(train.adas$class)
```

```
##
##   0   1
## 798 802
```

```
table(test$isFake)
```

```
##
##   0   1
## 196  42
```

```
train.adas$class[train.adas$class == 1] = "Fake"
train.adas$class[train.adas$class == 0] = "Real"

test$isFake[test$isFake == 1] = "Fake"
test$isFake[test$isFake == 0] = "Real"

train.adas$class = as.factor(train.adas$class)
test$isFake = as.factor(test$isFake)
```

```
table(train.adas$class)
```

```
##
## Fake Real
##  802  798
```

```
table(test$isFake)
```

```
##
## Fake Real
##   42  196
```

```r
set.seed(123)
t.grid = expand.grid(C = seq(0,2,length=20))

model_svm_2 = train(class~.,
                   data = train.adas,
                   method = "svmLinear",
                   trControl = ctrlspecs,
                   tuneGrid = t.grid)
model_svm_2
```

```
## Support Vector Machines with Linear Kernel
##
## 1600 samples
##    8 predictor
##    2 classes: 'Fake', 'Real'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1280, 1281, 1279
## Resampling results across tuning parameters:
##
##   C          Accuracy   Kappa
##   0.0000000        NaN        NaN
##   0.1052632  0.9156345  0.8312697
##   0.2105263  0.9175134  0.8350283
##   0.3157895  0.9181365  0.8362739
##   0.4210526  0.9181404  0.8362783
##   0.5263158  0.9175154  0.8350312
##   0.6315789  0.9181423  0.8362871
##   0.7368421  0.9206384  0.8412772
##   0.8421053  0.9200154  0.8400326
##   0.9473684  0.9187615  0.8375239
##   1.0526316  0.9193904  0.8387862
##   1.1578947  0.9225154  0.8450326
##   1.2631579  0.9206384  0.8412800
##   1.3684211  0.9225154  0.8450362
##   1.4736842  0.9225154  0.8450362
##   1.5789474  0.9225154  0.8450362
##   1.6842105  0.9231404  0.8462844
##   1.7894737  0.9231404  0.8462826
##   1.8947368  0.9218923  0.8437888
##   2.0000000  0.9218904  0.8437844
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.684211.
```

```r
varImp(model_svm_2)
```

```
## ROC curve variable importance
##
##                       Importance
## userMediaCount            100.00
```

```
## followRatio              80.33
## userFollowerCount        77.59
## userHasProfilPic         67.40
## usernameDigitCount       64.01
## userIsPrivate            46.31
## userBiographyLength      23.27
## userFollowingCount        0.00
```

```
predictions = predict(model_svm_2, test)
c.matrix = confusionMatrix(predictions, test$isFake)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   37   15
##       Real    5  181
##
##                Accuracy : 0.916
##                  95% CI : (0.8732, 0.9479)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 3.717e-05
##
##                   Kappa : 0.7356
##
##  Mcnemar's Test P-Value : 0.04417
##
##             Sensitivity : 0.8810
##             Specificity : 0.9235
##          Pos Pred Value : 0.7115
##          Neg Pred Value : 0.9731
##              Prevalence : 0.1765
##          Detection Rate : 0.1555
##    Detection Prevalence : 0.2185
##       Balanced Accuracy : 0.9022
##
##        'Positive' Class : Fake
##
```

```
svm2_acc = c.matrix$overall[1]
svm2_auc = multiclass.roc(as.numeric(test$isFake), as.numeric(predictions))$auc[1]
```

```
## Setting direction: controls < cases
```

```
svm2_precision = c.matrix$byClass[5]
svm2_recall = c.matrix$byClass[6]
svm2_F1 = c.matrix$byclass[7]
```

```
svm_performance = matrix(c(svm_acc, svm_auc, svm_precision, svm_recall, svm_F1,
                          svm2_acc, svm2_auc, svm2_precision, svm2_recall, svm2_F1),
                             ncol = 5, byrow = FALSE)
```

*** SVM Results**

```
## Warning in matrix(c(svm_acc, svm_auc, svm_precision, svm_recall, svm_F1, : data
## length [8] is not a sub-multiple or multiple of the number of columns [5]
```

```
colnames(svm_performance) = c("Accuracy", "AUC", "Precision", "Recall", "F1")
rownames(svm_performance) = c("SVM w/ Normalization",
                                  "SVM w/o Normalization")
as.table(svm_performance)
```

```
##                        Accuracy       AUC Precision    Recall        F1
## SVM w/ Normalization  0.9369748 0.8000000 0.9159664 0.7115385 0.9369748
## SVM w/o Normalization 0.9056122 0.8571429 0.9022109 0.8809524 0.9056122
```

3. Random Forest

```
table(train.adas$class)
```

```
##
## Fake Real
##  802  798
```

```
table(test$isFake)
```

```
##
## Fake Real
##   42  196
```

```
set.seed(123)
t.grid = expand.grid(mtry = c(1,2,3,4,5,7,8))
model_rf = train(class ~.,
                 data = train.adas,
                 trControl = ctrlspecs,
                 tuneGrid = t.grid,
                 method = "rf")
model_rf
```

```
## Random Forest
##
## 1600 samples
##    8 predictor
##    2 classes: 'Fake', 'Real'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1280, 1281, 1279
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.9825058  0.9650091
##   2     0.9825078  0.9650134
##   3     0.9825078  0.9650135
##   4     0.9818828  0.9637631
##   5     0.9812558  0.9625087
##   7     0.9825039  0.9650047
##   8     0.9806269  0.9612499
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
varImp(model_rf)
```

```
## rf variable importance
##
##                     Overall
## userMediaCount       100.00
## followRatio           90.50
## userFollowerCount     58.63
## userHasProfilPic      42.64
## userFollowingCount    32.54
## usernameDigitCount    27.38
## userIsPrivate         12.85
## userBiographyLength    0.00
```

```
predictions = predict(model_rf, test)
c.matrix = confusionMatrix(predictions, test$isFake)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   35    3
##       Real    7  193
##
##                Accuracy : 0.958
##                  95% CI : (0.9241, 0.9797)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 2.888e-10
##
##                   Kappa : 0.8498
##
##  Mcnemar's Test P-Value : 0.3428
##
##             Sensitivity : 0.8333
##             Specificity : 0.9847
```

```
##          Pos Pred Value : 0.9211
##          Neg Pred Value : 0.9650
##              Prevalence : 0.1765
##          Detection Rate : 0.1471
##    Detection Prevalence : 0.1597
##       Balanced Accuracy : 0.9090
##
##        'Positive' Class : Fake
##
```

```
rf_acc = c.matrix$overall[1]
rf_auc = multiclass.roc(as.numeric(test$isFake), as.numeric(predictions))$auc[1]
```

```
## Setting direction: controls < cases
```

```
rf_precision = c.matrix$byClass[5]
rf_recall = c.matrix$byClass[6]
rf_F1 = c.matrix$byclass[7]
```

```
rf_performance = matrix(c(rf_acc, rf_auc, rf_precision, rf_recall, rf_F1),
                        ncol = 5, byrow = FALSE)
```

*** Random Forest Results**

```
## Warning in matrix(c(rf_acc, rf_auc, rf_precision, rf_recall, rf_F1), ncol = 5, :
## data length [4] is not a sub-multiple or multiple of the number of columns [5]
```

```
colnames(rf_performance) = c("Accuracy", "AUC", "Precision", "Recall", "F1")
rownames(rf_performance) = c("Random Forest")
as.table(rf_performance)
```

```
##                Accuracy       AUC Precision    Recall        F1
## Random Forest 0.9579832 0.9090136 0.9210526 0.8333333 0.9579832
```

4. Naive Bayes

```
table(train.adas.norm$class)
```

```
##
## Fake Real
##  802  798
```

```
set.seed(123)
model_nb = train(class~.,
                 data = train.adas.norm,
                 method = "nb",
                 trControl = ctrlspecs)
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 82

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 83

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 101

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 102

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 208

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 249

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 266

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 82

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 83

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 101

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 102

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 208

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 249

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 266

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 102

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 102
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 85

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 85

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 152

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 152

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 82

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 205

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 216

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 232

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 260

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 299

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 82

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 205

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 216
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 232

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 260

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 299

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 62

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 67

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 275

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 62

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 67

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 275
```

```
model_nb
```

```
## Naive Bayes
##
## 1600 samples
##    8 predictor
##    2 classes: 'Fake', 'Real'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1280, 1281, 1279
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.8906266  0.7812761
##    TRUE      0.9531229  0.9062349
##
## Tuning parameter 'fL' was held constant at a value of 0
```

```
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE and adjust
##  = 1.
```

```
varImp(model_nb)
```

```
## ROC curve variable importance
##
##                     Importance
## userMediaCount          100.00
## followRatio              80.33
## userFollowerCount        77.59
## userHasProfilPic         67.40
## usernameDigitCount       64.01
## userIsPrivate            46.31
## userBiographyLength      23.27
## userFollowingCount        0.00
```

```
predictions = predict(model_nb, test.norm)
c.matrix = confusionMatrix(predictions, test.norm$isFake)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   39   17
##       Real    3  179
##
##                Accuracy : 0.916
##                  95% CI : (0.8732, 0.9479)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 3.717e-05
##
##                   Kappa : 0.7444
##
##  Mcnemar's Test P-Value : 0.00365
##
##             Sensitivity : 0.9286
##             Specificity : 0.9133
##          Pos Pred Value : 0.6964
##          Neg Pred Value : 0.9835
##              Prevalence : 0.1765
##          Detection Rate : 0.1639
##    Detection Prevalence : 0.2353
##       Balanced Accuracy : 0.9209
##
##        'Positive' Class : Fake
##
```

```r
nb_acc = c.matrix$overall[1]
nb_auc = multiclass.roc(as.numeric(test.norm$isFake), as.numeric(predictions))$auc[1]
```

```
## Setting direction: controls < cases
```

```r
nb_precision = c.matrix$byClass[5]
nb_recall = c.matrix$byClass[6]
nb_F1 = c.matrix$byclass[7]
```

Naive Bayes with out Normalization

```r
set.seed(123)
model_nb2 = train(class~.,
                  data = train.adas,
                  method = "nb",
                  trControl = ctrlspecs)
model_nb2
```

```
## Naive Bayes
##
## 1600 samples
##    8 predictor
##    2 classes: 'Fake', 'Real'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1280, 1280, 1280, 1281, 1279
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.8900036  0.7800294
##    TRUE      0.9524979  0.9049834
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE and adjust
##  = 1.
```

```r
varImp(model_nb2)
```

```
## ROC curve variable importance
##
##                      Importance
## userMediaCount           100.00
## followRatio               80.33
## userFollowerCount         77.59
## userHasProfilPic          67.40
```

```
## usernameDigitCount       64.01
## userIsPrivate            46.31
## userBiographyLength      23.27
## userFollowingCount        0.00
```

```
predictions = predict(model_nb2, test)
c.matrix = confusionMatrix(predictions, test$isFake)
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fake Real
##       Fake   40    8
##       Real    2  188
##
##                Accuracy : 0.958
##                  95% CI : (0.9241, 0.9797)
##     No Information Rate : 0.8235
##     P-Value [Acc > NIR] : 2.888e-10
##
##                   Kappa : 0.8631
##
##  Mcnemar's Test P-Value : 0.1138
##
##             Sensitivity : 0.9524
##             Specificity : 0.9592
##          Pos Pred Value : 0.8333
##          Neg Pred Value : 0.9895
##              Prevalence : 0.1765
##          Detection Rate : 0.1681
##    Detection Prevalence : 0.2017
##       Balanced Accuracy : 0.9558
##
##        'Positive' Class : Fake
##
```

```
nb2_acc = c.matrix$overall[1]
nb2_auc = multiclass.roc(as.numeric(test$isFake), as.numeric(predictions))$auc[1]
```

```
## Setting direction: controls < cases
```

```
nb2_precision = c.matrix$byClass[5]
nb2_recall = c.matrix$byClass[6]
nb2_F1 = c.matrix$byclass[7]
```

```
nb_performance = matrix(c(nb_acc, nb_auc, nb_precision, nb_recall, nb_F1,
                          nb2_acc, nb2_auc, nb2_precision, nb2_recall, nb2_F1),
                           ncol = 5, byrow = FALSE)
```

```r
colnames(nb_performance) = c("Accuracy", "AUC", "Precision", "Recall", "F1")
rownames(nb_performance) = c("Naive Bayes", "Naive Bayes w/o Normalization")
as.table(nb_performance)
```

*** Naive Bayes Results**

```
##                                Accuracy       AUC Precision    Recall        F1
## Naive Bayes                   0.9159664 0.6964286 0.9579832 0.8333333 0.9159664
## Naive Bayes w/o Normalization 0.9209184 0.9285714 0.9557823 0.9523810 0.9209184
```

5. Neural Network

```r
table(train.adas.norm$class)
```

```
## 
## Fake Real 
##  802  798
```

```r
table(test.norm$isFake)
```

```
## 
## Fake Real 
##   42  196
```

```r
train.adas.norm$class = as.numeric(train.adas.norm$class)
test.norm$isFake = as.numeric(test.norm$isFake)

table(train.adas.norm$class)
```

```
## 
##   1   2 
## 802 798
```

```r
table(test.norm$isFake)
```

```
## 
##   1   2 
##  42 196
```

```r
train.adas.norm$class[train.adas.norm$class == 2] = 0
test.norm$isFake[test.norm$isFake == 2] = 0

table(train.adas.norm$class)
```

```
## 
##   0   1 
## 798 802
```

```r
table(test.norm$isFake)
```

```
##
##   0   1
## 196  42
```

going to split the test data in half to use as a validation set

```r
set.seed(123)

val.index.nn = createDataPartition(test.norm$isFake, p=0.5, list=FALSE)
test.norm_nn = as.data.frame(test.norm[val.index.nn, ])
val.norm_nn = as.data.frame(test.norm[-val.index.nn, ])
```

```r
str(val.norm_nn)
```

```
## 'data.frame':    119 obs. of  9 variables:
##  $ isFake             : num  0 0 1 0 0 0 0 0 0 0 ...
##  $ userBiographyLength: num  0.22 0.193 0 0 0.553 ...
##  $ userFollowerCount  : num  -0.788 -0.88 -1 -0.99 -0.952 ...
##  $ userFollowingCount : num  -0.91 -0.951 -0.993 -0.994 -0.89 ...
##  $ userHasProfilPic   : int  1 1 0 1 1 1 1 1 1 1 ...
##  $ userIsPrivate      : int  0 1 1 0 0 1 0 1 0 1 ...
##  $ userMediaCount     : num  0.02268 0.00945 0 0.00378 0.00567 ...
##  $ usernameDigitCount : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ followRatio        : num  5.63e-02 5.86e-02 2.83e-05 3.86e-02 9.72e-03 ...
```

```r
str(test.norm_nn)
```

```
## 'data.frame':    119 obs. of  9 variables:
##  $ isFake             : num  1 0 0 0 0 1 0 0 0 0 ...
##  $ userBiographyLength: num  0.1733 0.0467 0 0.26 0 ...
##  $ userFollowerCount  : num  -0.799 -0.891 -0.884 -0.394 -0.916 ...
##  $ userFollowingCount : num  -0.952 -0.961 -0.949 -0.759 -0.945 ...
##  $ userHasProfilPic   : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ userIsPrivate      : int  0 0 1 1 1 1 0 1 1 0 ...
##  $ userMediaCount     : num  0.00189 0.06144 0.37524 0.3535 0.00284 ...
##  $ usernameDigitCount : num  0 0 0 0 0 0 0.2 0 0.4 0 ...
##  $ followRatio        : num  0.1006 0.0667 0.0543 0.0606 0.0367 ...
```

```r
str(train.adas.norm)
```

```
## 'data.frame':    1600 obs. of  9 variables:
##  $ userBiographyLength: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ userFollowerCount  : num  -0.998 -0.999 -0.969 -0.989 -0.937 ...
##  $ userFollowingCount : num  -0.376 -0.99 -0.853 -0.669 -0.664 ...
##  $ userHasProfilPic   : num  0 1 1 1 1 1 0 1 1 1 ...
##  $ userIsPrivate      : num  0 0 0 0 0 0 1 1 0 1 ...
##  $ userMediaCount     : num  0 0 0.00343 0 0.00571 ...
##  $ usernameDigitCount : num  0.375 0.5 0 0 0.25 0.375 0.5 0.25 0.375 0 ...
##  $ followRatio        : num  -0.000154 0.007669 0.015404 0.002174 0.013557 ...
##  $ class              : num  1 1 1 1 1 1 1 1 1 1 ...
```

```
train.labels = train.adas.norm$class
train.nn = train.adas.norm[-9]
str(train.nn)
```

```
## 'data.frame':    1600 obs. of  8 variables:
##  $ userBiographyLength: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ userFollowerCount  : num  -0.998 -0.999 -0.969 -0.989 -0.937 ...
##  $ userFollowingCount : num  -0.376 -0.99 -0.853 -0.669 -0.664 ...
##  $ userHasProfilPic   : num  0 1 1 1 1 1 0 1 1 1 ...
##  $ userIsPrivate      : num  0 0 0 0 0 0 1 1 0 1 ...
##  $ userMediaCount     : num  0 0 0.00343 0 0.00571 ...
##  $ usernameDigitCount : num  0.375 0.5 0 0 0.25 0.375 0.5 0.25 0.375 0 ...
##  $ followRatio        : num  -0.000154 0.007669 0.015404 0.002174 0.013557 ...
```

```
test.labels = test.norm_nn[1]
test.norm_nn = test.norm_nn[-1]
str(test.norm_nn)
```

```
## 'data.frame':    119 obs. of  8 variables:
##  $ userBiographyLength: num  0.1733 0.0467 0 0.26 0 ...
##  $ userFollowerCount  : num  -0.799 -0.891 -0.884 -0.394 -0.916 ...
##  $ userFollowingCount : num  -0.952 -0.961 -0.949 -0.759 -0.945 ...
##  $ userHasProfilPic   : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ userIsPrivate      : int  0 0 1 1 1 1 0 1 1 0 ...
##  $ userMediaCount     : num  0.00189 0.06144 0.37524 0.3535 0.00284 ...
##  $ usernameDigitCount : num  0 0 0 0 0 0 0.2 0 0.4 0 ...
##  $ followRatio        : num  0.1006 0.0667 0.0543 0.0606 0.0367 ...
```

```
val.labels = val.norm_nn[1]
val.norm_nn = val.norm_nn[-1]
str(val.norm_nn)
```

```
## 'data.frame':    119 obs. of  8 variables:
##  $ userBiographyLength: num  0.22 0.193 0 0 0.553 ...
##  $ userFollowerCount  : num  -0.788 -0.88 -1 -0.99 -0.952 ...
##  $ userFollowingCount : num  -0.91 -0.951 -0.993 -0.994 -0.89 ...
##  $ userHasProfilPic   : int  1 1 0 1 1 1 1 1 1 1 ...
##  $ userIsPrivate      : int  0 1 1 0 0 1 0 1 0 1 ...
##  $ userMediaCount     : num  0.02268 0.00945 0 0.00378 0.00567 ...
##  $ usernameDigitCount : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ followRatio        : num  5.63e-02 5.86e-02 2.83e-05 3.86e-02 9.72e-03 ...
```

```
train.labels = as.data.frame(train.labels)
```

```
nrow(val.labels)
```

```
## [1] 119
```

```
nrow(test.labels)
```

```
## [1] 119
```

```r
nrow(train.labels)
```

```
## [1] 1600
```

```r
dim(val.norm_nn)
```

```
## [1] 119   8
```

```r
dim(test.norm_nn)
```

```
## [1] 119   8
```

```r
dim(train.nn)
```

```
## [1] 1600    8
```

```r
val.norm_nn = as.matrix(val.norm_nn)
test.norm_nn = as.matrix(test.norm_nn)
train.nn = as.matrix(train.nn)
```

```r
val.labels = val.labels$isFake
test.labels = test.labels$isFake
train.labels = train.labels$train.labels
```

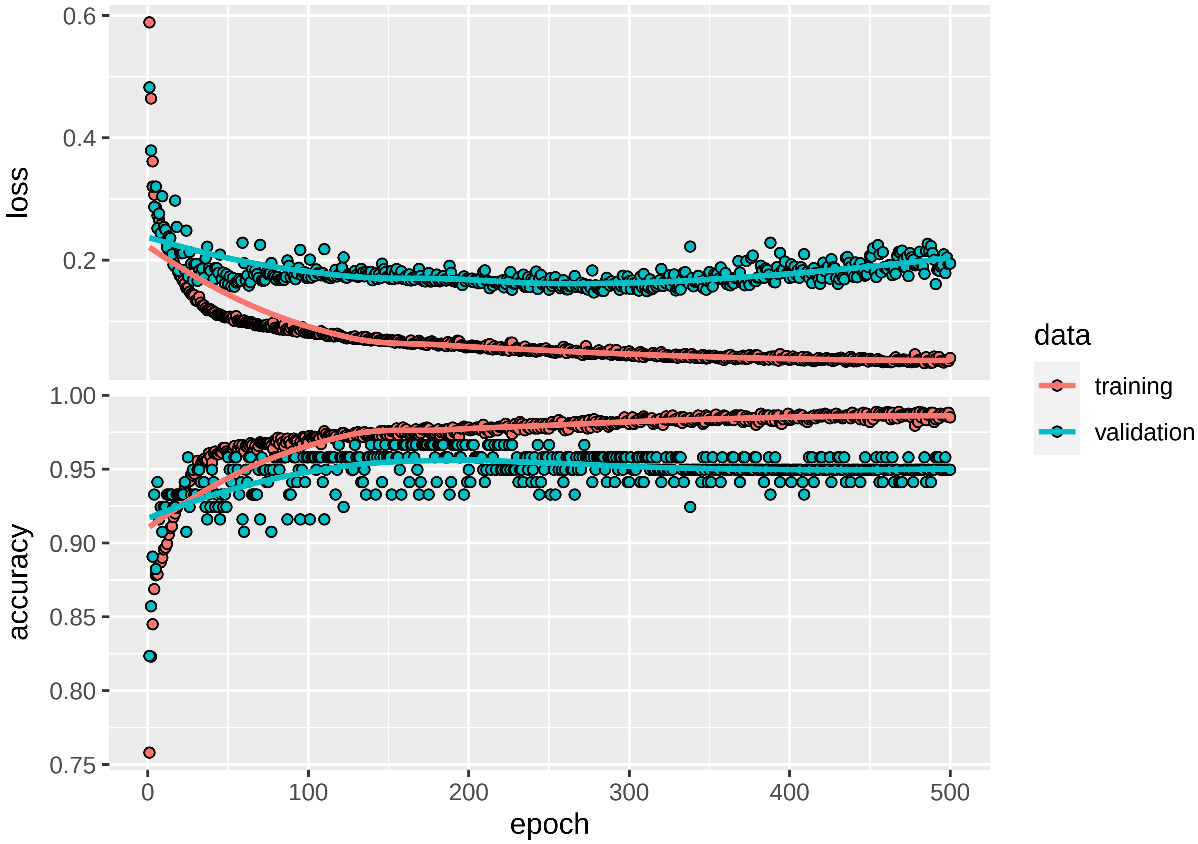
```r
library(keras)
library(dplyr)
library(tfruns)
```

```r
set.seed(123)

model = keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu", input_shape = dim(train.nn)[2])%>%
  layer_dense(units = 32, activation = "relu")%>%
  layer_dense(units = 1, activation = "sigmoid")
```

```
## Loaded Tensorflow version 2.9.2
```

```r
model %>% compile(loss = "binary_crossentropy",
                  optimizer = "adam",
                  metrics = "accuracy")

history = model %>% fit(train.nn, train.labels,
                        batch_size = 32, epochs = 500,
                        verbose = 0,
                        validation_data = list(val.norm_nn, val.labels))
```

```r
plot(history)
```




```
history
```

```
##
## Final epoch (plot to see history):
##         loss: 0.03981
##     accuracy: 0.985
##     val_loss: 0.1943
## val_accuracy: 0.9496
```

```
predictions = model %>% predict(test.norm_nn) %>% k_argmax()
```

```
predictions = as.array(predictions)
```

```
MLmetrics::Accuracy(predictions, test.labels)
```

```
## [1] 0.8403361
```

```
MLmetrics::AUC(predictions, test.labels)
```

```
## [1] 0.5
```

```r
c.matrix = confusionMatrix(factor(predictions), factor(test.labels))
```

```
## Warning in confusionMatrix.default(factor(predictions), factor(test.labels)):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```r
c.matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 100  19
##          1   0   0
##
##                Accuracy : 0.8403
##                  95% CI : (0.7619, 0.901)
##     No Information Rate : 0.8403
##     P-Value [Acc > NIR] : 0.5608
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 3.636e-05
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.8403
##          Neg Pred Value :    NaN
##              Prevalence : 0.8403
##          Detection Rate : 0.8403
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```r
nn_acc = c.matrix$overall[1]
nn_auc = multiclass.roc(as.numeric(test.labels), as.numeric(predictions))$auc[1]
```

```
## Setting direction: controls < cases
```

```r
nn_precision = c.matrix$byClass[5]
nn_recall = c.matrix$byClass[6]
nn_F1 = c.matrix$byclass[7]
```

```r
nn_performance = matrix(c(nn_acc, nn_auc, nn_precision, nn_recall, nn_F1),
                        ncol = 5, byrow = FALSE)
```

```
## Warning in matrix(c(nn_acc, nn_auc, nn_precision, nn_recall, nn_F1), ncol = 5, :
## data length [4] is not a sub-multiple or multiple of the number of columns [5]
```

```r
colnames(nn_performance) = c("Accuracy", "AUC", "Precision", "Recall", "F1")
rownames(nn_performance) = c("Neural Network")
as.table(nn_performance)
```

```
##                 Accuracy       AUC Precision    Recall        F1
## Neural Network 0.8403361 0.5000000 0.8403361 1.0000000 0.8403361
```

Final Results

```r
a = rbind(log_reg_performance, svm_performance)
b = rbind(a, rf_performance)
c = rbind(b, nn_performance)
d = rbind(c, nb_performance)
d
```

```
##                               Accuracy       AUC Precision    Recall        F1
## LR                           0.9159664 0.9369748 0.9369748 0.9369748 0.9159664
## LR ADASYN                    0.9714529 0.9832362 0.9404762 0.9506803 0.9714529
## LR ADASYN CV                 0.8666667 0.8000000 0.8000000 0.8000000 0.8666667
## LR ADASYN CV NORM            0.6190476 0.8571429 0.8571429 0.8571429 0.6190476
## SVM w/ Normalization         0.9369748 0.8000000 0.9159664 0.7115385 0.9369748
## SVM w/o Normalization        0.9056122 0.8571429 0.9022109 0.8809524 0.9056122
## Random Forest                0.9579832 0.9090136 0.9210526 0.8333333 0.9579832
## Neural Network               0.8403361 0.5000000 0.8403361 1.0000000 0.8403361
## Naive Bayes                  0.9159664 0.6964286 0.9579832 0.8333333 0.9159664
## Naive Bayes w/o Normalization 0.9209184 0.9285714 0.9557823 0.9523810 0.9209184
```

- best accuracy : LR ADASYN
- best AUC : LR ADASYN
- best precision: LR ADASYN
- best recall : LR ADASYN
- best F1 : LR ADASYN