Statistical Learning - Final Report - Appendix Code

James Hahn

```
# DATA READING, PREPROCESSING, BASIC STATISTICS, OUTLIER DETECTION

library(DMwR)
```

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

```
## Loading required package: grid
```

```
library(class)
library(MASS)
library(stats)

newsDataOriginal <- read.table("OnlineNewsPopularity.csv", header=TRUE, sep=",")
newsDataOriginal$shares = as.numeric(newsDataOriginal$shares)
newsDataOriginal = newsDataOriginal[sample(1:nrow(newsDataOriginal)), ]
names(newsDataOriginal)
```

```
##  [1] "url"                           "timedelta"
##  [3] "n_tokens_title"                "n_tokens_content"
##  [5] "n_unique_tokens"               "n_non_stop_words"
##  [7] "n_non_stop_unique_tokens"      "num_hrefs"
##  [9] "num_self_hrefs"                "num_imgs"
## [11] "num_videos"                    "average_token_length"
## [13] "num_keywords"                  "data_channel_is_lifestyle"
## [15] "data_channel_is_entertainment" "data_channel_is_bus"
## [17] "data_channel_is_socmed"        "data_channel_is_tech"
## [19] "data_channel_is_world"         "kw_min_min"
## [21] "kw_max_min"                    "kw_avg_min"
## [23] "kw_min_max"                    "kw_max_max"
## [25] "kw_avg_max"                    "kw_min_avg"
## [27] "kw_max_avg"                    "kw_avg_avg"
## [29] "self_reference_min_shares"     "self_reference_max_shares"
## [31] "self_reference_avg_sharess"    "weekday_is_monday"
## [33] "weekday_is_tuesday"            "weekday_is_wednesday"
## [35] "weekday_is_thursday"           "weekday_is_friday"
## [37] "weekday_is_saturday"           "weekday_is_sunday"
## [39] "is_weekend"                    "LDA_00"
## [41] "LDA_01"                        "LDA_02"
## [43] "LDA_03"                        "LDA_04"
## [45] "global_subjectivity"           "global_sentiment_polarity"
## [47] "global_rate_positive_words"    "global_rate_negative_words"
## [49] "rate_positive_words"           "rate_negative_words"
## [51] "avg_positive_polarity"         "min_positive_polarity"
## [53] "max_positive_polarity"         "avg_negative_polarity"
## [55] "min_negative_polarity"         "max_negative_polarity"
## [57] "title_subjectivity"            "title_sentiment_polarity"
## [59] "abs_title_subjectivity"        "abs_title_sentiment_polarity"
## [61] "shares"
```

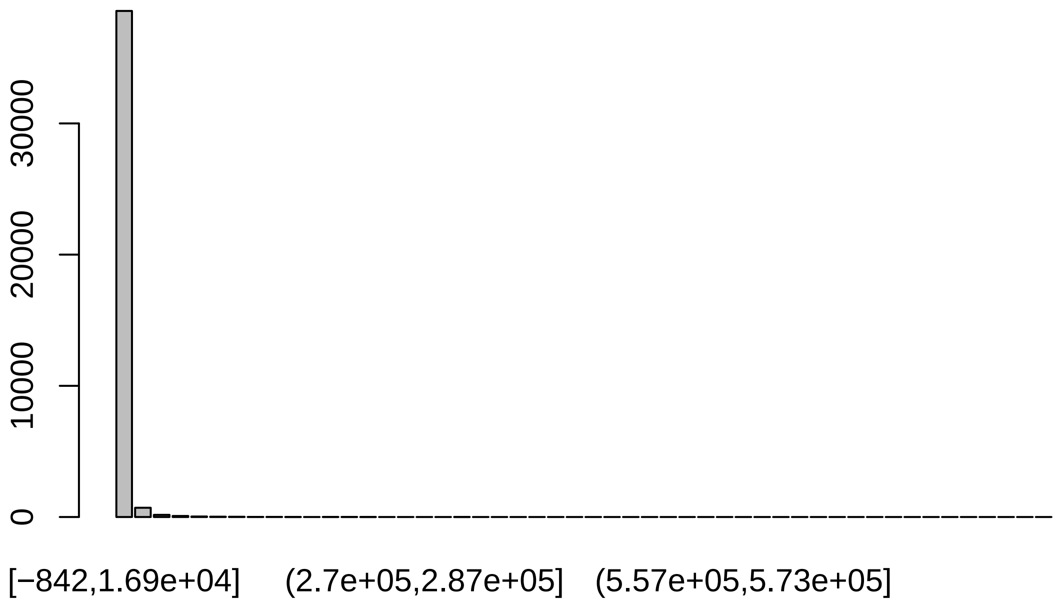
```
summary(newsDataOriginal)
```

```
##                                                              url
##  http://mashable.com/2013/01/07/amazon-instant-video-browser/  :    1
##  http://mashable.com/2013/01/07/ap-samsung-sponsored-tweets/   :    1
##  http://mashable.com/2013/01/07/apple-40-billion-app-downloads/:    1
##  http://mashable.com/2013/01/07/astronaut-notre-dame-bcs/      :    1
##  http://mashable.com/2013/01/07/att-u-verse-apps/              :    1
##  http://mashable.com/2013/01/07/beewi-smart-toys/              :    1
##  (Other)                                                       :39638
##    timedelta     n_tokens_title n_tokens_content n_unique_tokens
##  Min.   :  8.0   Min.   : 2.0   Min.   :   0.0   Min.   :  0.0000
##  1st Qu.:164.0   1st Qu.: 9.0   1st Qu.: 246.0   1st Qu.:  0.4709
##  Median :339.0   Median :10.0   Median : 409.0   Median :  0.5392
##  Mean   :354.5   Mean   :10.4   Mean   : 546.5   Mean   :  0.5482
##  3rd Qu.:542.0   3rd Qu.:12.0   3rd Qu.: 716.0   3rd Qu.:  0.6087
##  Max.   :731.0   Max.   :23.0   Max.   :8474.0   Max.   :701.0000
##
##  n_non_stop_words    n_non_stop_unique_tokens   num_hrefs
##  Min.   :   0.0000   Min.   :  0.0000         Min.   :  0.00
##  1st Qu.:   1.0000   1st Qu.:  0.6257         1st Qu.:  4.00
##  Median :   1.0000   Median :  0.6905         Median :  8.00
##  Mean   :   0.9965   Mean   :  0.6892         Mean   : 10.88
##  3rd Qu.:   1.0000   3rd Qu.:  0.7546         3rd Qu.: 14.00
##  Max.   :1042.0000   Max.   :650.0000         Max.   :304.00
##
##  num_self_hrefs       num_imgs         num_videos    average_token_length
##  Min.   :  0.000   Min.   :  0.000   Min.   : 0.00   Min.   :0.000
##  1st Qu.:  1.000   1st Qu.:  1.000   1st Qu.: 0.00   1st Qu.:4.478
##  Median :  3.000   Median :  1.000   Median : 0.00   Median :4.664
##  Mean   :  3.294   Mean   :  4.544   Mean   : 1.25   Mean   :4.548
##  3rd Qu.:  4.000   3rd Qu.:  4.000   3rd Qu.: 1.00   3rd Qu.:4.855
##  Max.   :116.000   Max.   :128.000   Max.   :91.00   Max.   :8.042
##
##   num_keywords    data_channel_is_lifestyle data_channel_is_entertainment
##  Min.   : 1.000   Min.   :0.00000           Min.   :0.000
##  1st Qu.: 6.000   1st Qu.:0.00000           1st Qu.:0.000
##  Median : 7.000   Median :0.00000           Median :0.000
##  Mean   : 7.224   Mean   :0.05295           Mean   :0.178
##  3rd Qu.: 9.000   3rd Qu.:0.00000           3rd Qu.:0.000
##  Max.   :10.000   Max.   :1.00000           Max.   :1.000
##
##  data_channel_is_bus data_channel_is_socmed data_channel_is_tech
##  Min.   :0.0000      Min.   :0.0000         Min.   :0.0000
##  1st Qu.:0.0000      1st Qu.:0.0000         1st Qu.:0.0000
##  Median :0.0000      Median :0.0000         Median :0.0000
##  Mean   :0.1579      Mean   :0.0586         Mean   :0.1853
##  3rd Qu.:0.0000      3rd Qu.:0.0000         3rd Qu.:0.0000
##  Max.   :1.0000      Max.   :1.0000         Max.   :1.0000
##
##  data_channel_is_world   kw_min_min       kw_max_min       kw_avg_min
##  Min.   :0.0000        Min.   : -1.00   Min.   :     0   Min.   :   -1.0
##  1st Qu.:0.0000        1st Qu.: -1.00   1st Qu.:   445   1st Qu.:  141.8
##  Median :0.0000        Median : -1.00   Median :   660   Median :  235.5
##  Mean   :0.2126        Mean   : 26.11   Mean   :  1154   Mean   :  312.4
##  3rd Qu.:0.0000        3rd Qu.:  4.00   3rd Qu.:  1000   3rd Qu.:  357.0
##  Max.   :1.0000        Max.   :377.00   Max.   :298400   Max.   :42827.9
##                                                                          
##    kw_min_max       kw_max_max       kw_avg_max       kw_min_avg
##  Min.   :     0   Min.   :     0   Min.   :     0   Min.   :  -1
##  1st Qu.:     0   1st Qu.:843300   1st Qu.:172847   1st Qu.:   0
##  Median :  1400   Median :843300   Median :244572   Median :1024
##  Mean   : 13612   Mean   :752324   Mean   :259282   Mean   :1117
##  3rd Qu.:  7900   3rd Qu.:843300   3rd Qu.:330980   3rd Qu.:2057
##  Max.   :843300   Max.   :843300   Max.   :843300   Max.   :3613
##                                                                 
##    kw_max_avg       kw_avg_avg    self_reference_min_shares
##  Min.   :     0   Min.   :    0   Min.   :     0           
##  1st Qu.:  3562   1st Qu.: 2382   1st Qu.:   639           
##  Median :  4356   Median : 2870   Median :  1200           
##  Mean   :  5657   Mean   : 3136   Mean   :  3999           
##  3rd Qu.:  6020   3rd Qu.: 3600   3rd Qu.:  2600           
##  Max.   :298400   Max.   :43568   Max.   :843300           
##                                                            
##  self_reference_max_shares self_reference_avg_sharess weekday_is_monday
##  Min.   :     0            Min.   :     0.0           Min.   :0.000    
##  1st Qu.:  1100            1st Qu.:   981.2           1st Qu.:0.000    
##  Median :  2800            Median :  2200.0           Median :0.000    
##  Mean   : 10329            Mean   :  6401.7           Mean   :0.168    
##  3rd Qu.:  8000            3rd Qu.:  5200.0           3rd Qu.:0.000    
##  Max.   :843300            Max.   :843300.0           Max.   :1.000    
##                                                                        
##  weekday_is_tuesday weekday_is_wednesday weekday_is_thursday
##  Min.   :0.0000     Min.   :0.0000       Min.   :0.0000     
##  1st Qu.:0.0000     1st Qu.:0.0000       1st Qu.:0.0000     
##  Median :0.0000     Median :0.0000       Median :0.0000     
##  Mean   :0.1864     Mean   :0.1875       Mean   :0.1833     
##  3rd Qu.:0.0000     3rd Qu.:0.0000       3rd Qu.:0.0000     
##  Max.   :1.0000     Max.   :1.0000       Max.   :1.0000     
##                                                             
##  weekday_is_friday weekday_is_saturday weekday_is_sunday   is_weekend    
##  Min.   :0.0000    Min.   :0.00000     Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.0000    1st Qu.:0.00000     1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.0000    Median :0.00000     Median :0.00000   Median :0.0000  
##  Mean   :0.1438    Mean   :0.06188     Mean   :0.06904   Mean   :0.1309  
##  3rd Qu.:0.0000    3rd Qu.:0.00000     3rd Qu.:0.00000   3rd Qu.:0.0000  
##  Max.   :1.0000    Max.   :1.00000     Max.   :1.00000   Max.   :1.0000  
##                                                                          
##      LDA_00           LDA_01           LDA_02           LDA_03       
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000  
##  1st Qu.:0.02505   1st Qu.:0.02501   1st Qu.:0.02857   1st Qu.:0.02857  
##  Median :0.03339   Median :0.03334   Median :0.04000   Median :0.04000  
##  Mean   :0.18460   Mean   :0.14126   Mean   :0.21632   Mean   :0.22377  
##  3rd Qu.:0.24096   3rd Qu.:0.15083   3rd Qu.:0.33422   3rd Qu.:0.37576  
##  Max.   :0.92699   Max.   :0.92595   Max.   :0.92000   Max.   :0.92653  
##                                                                         
##      LDA_04        global_subjectivity global_sentiment_polarity
##  Min.   :0.00000   Min.   :0.0000      Min.   :-0.39375         
##  1st Qu.:0.02857   1st Qu.:0.3962      1st Qu.: 0.05776         
##  Median :0.04073   Median :0.4535      Median : 0.11912         
##  Mean   :0.23403   Mean   :0.4434      Mean   : 0.11931
##  3rd Qu.:0.39999   3rd Qu.:0.5083      3rd Qu.: 0.17783
##  Max.   :0.92719   Max.   :1.0000      Max.   : 0.72784
##
##  global_rate_positive_words global_rate_negative_words rate_positive_words
##  Min.   :0.00000            Min.   :0.000000           Min.   :0.0000
##  1st Qu.:0.02838            1st Qu.:0.009615           1st Qu.:0.6000
##  Median :0.03902            Median :0.015337           Median :0.7105
##  Mean   :0.03962            Mean   :0.016612           Mean   :0.6822
##  3rd Qu.:0.05028            3rd Qu.:0.021739           3rd Qu.:0.8000
##  Max.   :0.15549            Max.   :0.184932           Max.   :1.0000
##
##  rate_negative_words avg_positive_polarity min_positive_polarity
##  Min.   :0.0000      Min.   :0.0000        Min.   :0.00000
##  1st Qu.:0.1852      1st Qu.:0.3062        1st Qu.:0.05000
##  Median :0.2800      Median :0.3588        Median :0.10000
##  Mean   :0.2879      Mean   :0.3538        Mean   :0.09545
##  3rd Qu.:0.3846      3rd Qu.:0.4114        3rd Qu.:0.10000
##  Max.   :1.0000      Max.   :1.0000        Max.   :1.00000
##
##  max_positive_polarity avg_negative_polarity min_negative_polarity
##  Min.   :0.0000        Min.   :-1.0000       Min.   :-1.0000
##  1st Qu.:0.6000        1st Qu.:-0.3284       1st Qu.:-0.7000
##  Median :0.8000        Median :-0.2533       Median :-0.5000
##  Mean   :0.7567        Mean   :-0.2595       Mean   :-0.5219
##  3rd Qu.:1.0000        3rd Qu.:-0.1869       3rd Qu.:-0.3000
##  Max.   :1.0000        Max.   : 0.0000       Max.   : 0.0000
##
##  max_negative_polarity title_subjectivity title_sentiment_polarity
##  Min.   :-1.0000       Min.   :0.0000     Min.   :-1.00000
##  1st Qu.:-0.1250       1st Qu.:0.0000     1st Qu.: 0.00000
##  Median :-0.1000       Median :0.1500     Median : 0.00000
##  Mean   :-0.1075       Mean   :0.2824     Mean   : 0.07143
##  3rd Qu.:-0.0500       3rd Qu.:0.5000     3rd Qu.: 0.15000
##  Max.   : 0.0000       Max.   :1.0000     Max.   : 1.00000
##
##  abs_title_subjectivity abs_title_sentiment_polarity     shares
##  Min.   :0.0000         Min.   :0.0000               Min.   :     1
##  1st Qu.:0.1667         1st Qu.:0.0000               1st Qu.:   946
##  Median :0.5000         Median :0.0000               Median :  1400
##  Mean   :0.3418         Mean   :0.1561               Mean   :  3395
##  3rd Qu.:0.5000         3rd Qu.:0.2500               3rd Qu.:  2800
##  Max.   :0.5000         Max.   :1.0000               Max.   :843300
##
```

```r
summary(newsDataOriginal$shares)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1     946    1400    3395    2800  843300
```

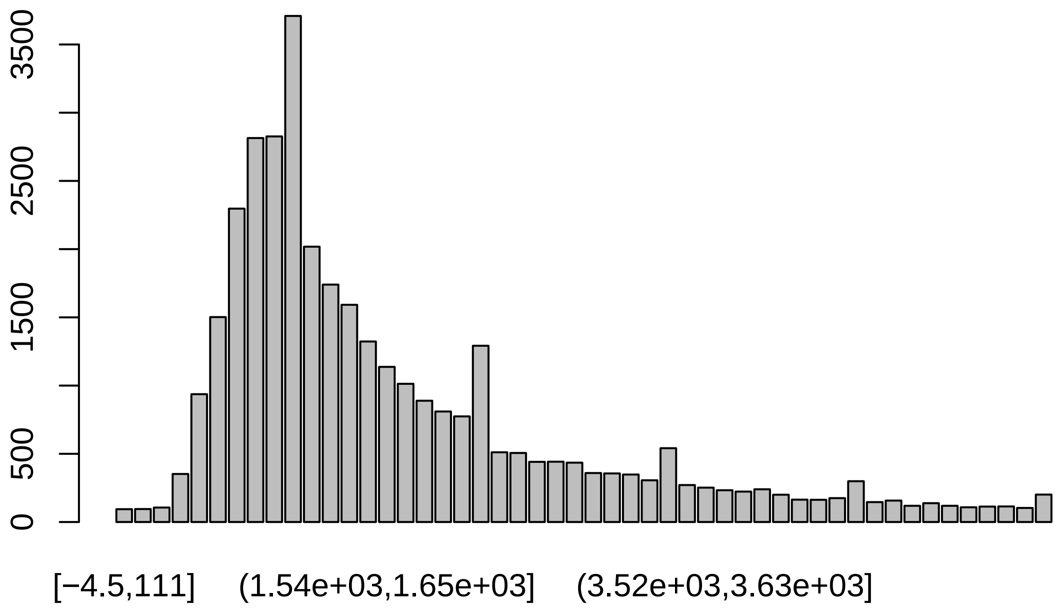
```r
newsDataLen <- nrow(newsDataOriginal)
shares_bins <- cut(newsDataOriginal$shares, 50, include.lowest=TRUE)
plot(shares_bins)
```



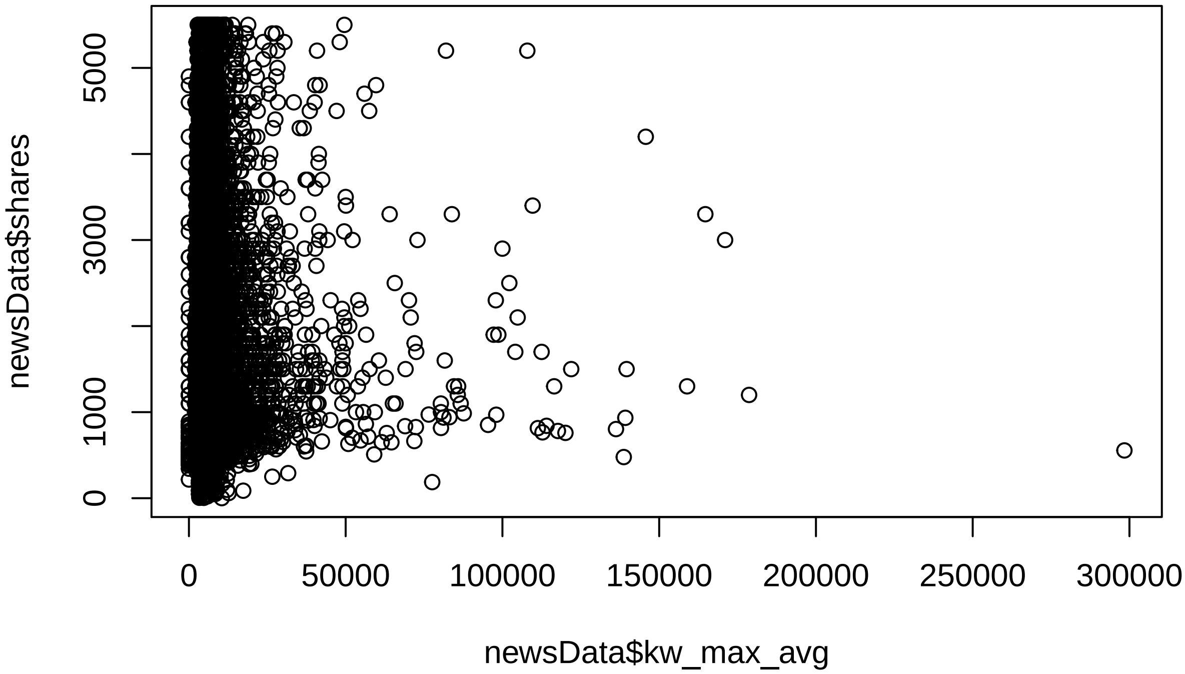
```
sharesIqr <- IQR(newsDataOriginal$shares)
shares75Quant <- quantile(newsDataOriginal$shares, 0.75)
shares25Quant <- quantile(newsDataOriginal$shares, 0.25)
newsData <- newsDataOriginal[newsDataOriginal$shares < (1.5*sharesIqr + shares75Quant) & newsDataOrigina
summary(newsData$shares)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1     903    1300    1672    2100    5500
```

```
numOutliers <- (newsDataLen - nrow(newsData))

shares_bins <- cut(newsData$shares, 50, include.lowest=TRUE)
plot(shares_bins) # plot the shares distribution AFTER outlier removal so it isn't as skewed
```



```r
plot(newsData$kw_max_avg, newsData$shares)
```



```r
newsDataQuant <- newsData[, sapply(newsData, class) == "numeric"]
names(newsDataQuant)
```

```
##  [1] "timedelta"                     "n_tokens_title"
##  [3] "n_tokens_content"              "n_unique_tokens"
##  [5] "n_non_stop_words"              "n_non_stop_unique_tokens"
##  [7] "num_hrefs"                     "num_self_hrefs"
##  [9] "num_imgs"                      "num_videos"
## [11] "average_token_length"          "num_keywords"
## [13] "data_channel_is_lifestyle"     "data_channel_is_entertainment"
## [15] "data_channel_is_bus"           "data_channel_is_socmed"
## [17] "data_channel_is_tech"          "data_channel_is_world"
## [19] "kw_min_min"                    "kw_max_min"
## [21] "kw_avg_min"                    "kw_min_max"
## [23] "kw_max_max"                    "kw_avg_max"
## [25] "kw_min_avg"                    "kw_max_avg"
## [27] "kw_avg_avg"                    "self_reference_min_shares"
## [29] "self_reference_max_shares"     "self_reference_avg_sharess"
## [31] "weekday_is_monday"             "weekday_is_tuesday"
## [33] "weekday_is_wednesday"          "weekday_is_thursday"
## [35] "weekday_is_friday"             "weekday_is_saturday"
## [37] "weekday_is_sunday"             "is_weekend"
## [39] "LDA_00"                        "LDA_01"
## [41] "LDA_02"                        "LDA_03"
## [43] "LDA_04"                        "global_subjectivity"
## [45] "global_sentiment_polarity"     "global_rate_positive_words"
## [47] "global_rate_negative_words"    "rate_positive_words"
## [49] "rate_negative_words"           "avg_positive_polarity"
## [51] "min_positive_polarity"         "max_positive_polarity"
## [53] "avg_negative_polarity"         "min_negative_polarity"
## [55] "max_negative_polarity"         "title_subjectivity"
## [57] "title_sentiment_polarity"      "abs_title_subjectivity"
## [59] "abs_title_sentiment_polarity"  "shares"
```

```r
cor(as.matrix(newsData[, 61]), as.matrix(newsData[,-1])) # correlations with 'shares' and every other v
```

```
##       timedelta n_tokens_title n_tokens_content n_unique_tokens
## [1,] 0.03657173    -0.04204983       0.04782074     -0.04909971
##      n_non_stop_words n_non_stop_unique_tokens num_hrefs num_self_hrefs
## [1,]      -0.01318755              -0.05080723 0.0776524     0.04316208
##        num_imgs   num_videos average_token_length num_keywords
## [1,] 0.05592683 -0.002898373          -0.02555183   0.06553517
##      data_channel_is_lifestyle data_channel_is_entertainment
## [1,]                0.03143692                    -0.1054218
##      data_channel_is_bus data_channel_is_socmed data_channel_is_tech
## [1,]         0.001639743              0.1149444           0.09737915
##      data_channel_is_world kw_min_min kw_max_min kw_avg_min  kw_min_max
## [1,]             -0.137431 0.03989283 0.02247175 0.03162921 0.007840949
##       kw_max_max kw_avg_max kw_min_avg kw_max_avg kw_avg_avg
## [1,] -0.02491639 0.01602475 0.08951021 0.06315745  0.1476776
##      self_reference_min_shares self_reference_max_shares
## [1,]                0.04458771                0.05480541
##      self_reference_avg_sharess weekday_is_monday weekday_is_tuesday
## [1,]                 0.05719484       -0.02269312        -0.03918078
##      weekday_is_wednesday weekday_is_thursday weekday_is_friday
## [1,]          -0.04175593         -0.02504239       0.009257667
##      weekday_is_saturday weekday_is_sunday is_weekend     LDA_00
## [1,]            0.101764        0.08975654  0.1399974 0.07562637
##           LDA_01     LDA_02     LDA_03     LDA_04 global_subjectivity
## [1,] -0.07674991 -0.1366928 0.03927561 0.08673353          0.05829045
##      global_sentiment_polarity global_rate_positive_words
## [1,]                0.06326113                 0.06326329
##      global_rate_negative_words rate_positive_words rate_negative_words
## [1,]                -0.02546071          0.04474235         -0.06757323
##      avg_positive_polarity min_positive_polarity max_positive_polarity
## [1,]             0.0190065           -0.03203813            0.03322045
##      avg_negative_polarity min_negative_polarity max_negative_polarity
## [1,]          -0.003511512          -0.004899245           0.003100281
##      title_subjectivity title_sentiment_polarity abs_title_subjectivity
## [1,]         0.02585881                0.0452892            0.004831034
##      abs_title_sentiment_polarity shares
## [1,]                   0.02951515      1
```

Refer to above code. The above code does a lot of work. I have done some preprocessing on the data. For example, I plotted the original news data with 20 histogram bins and immediately realized the distribution was significantly skewed to the right. I concluded there were definitely outliers in the data, so I went into further analysis. I did a summary of the shares data, which is the target/predicted label, and saw the first quartile was at 946 shares, third quartile was at 2800 shares, and then the min and max were 1 and 843,300 respectively. Therefore, with an IQR of 1854, I calculated outliers as being outside the range (946 - IQR*1.5, 2800 + IQR*1.5). There were 4541 outliers in the data, taking the dataset from 39644 samples to 35103 samples. This had an immediate impact on the calculation of correlations. Although not depicted in the code above, I did analysis before removing the outliers and the correlations between shares and all other features were in the range [-0.07, +0.08]. As such, there were no strong correlations. After removing the outliers, the range increased to [-0.137, +0.148] with the strongest positive and negative relationships being with data_channel_is_entertainment (-0.105), data_channel_is_socmed (0.115), data_channel_is_world (-0.137), kw_avg_avg (0.148), weekday_is_saturday (0.102), is_weekend (0.140), and LDA_02 (-0.137).

```
# BASIC STATISTICS ON SIGNIFICANT PREDICTORS

summary(newsDataQuant$kw_max_avg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0    3531    4230    5460    5852  298400
```

```
summary(newsDataQuant$kw_avg_avg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0    2351    2816    3055    3490   37608
```

```
summary(newsDataQuant$LDA_00)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01818 0.02506 0.03341 0.18630 0.24487 0.92699
```

```
summary(newsDataQuant$LDA_03)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01818 0.02553 0.04000 0.21044 0.31880 0.92653
```

```
# FORWARD SUBSET SELECTION
library(leaps)

newsClassif <- head(newsDataQuant, 35000) # current classification set we're using for all classificati

trainIndex <- sample(1:nrow(newsClassif), 1*nrow(newsClassif)) # train indices
testIndex <- setdiff(1:nrow(newsClassif), trainIndex) # test indices
train <- newsClassif[trainIndex,]
test <- newsClassif[testIndex,]
trainX <- newsClassif[trainIndex, -61]
trainY <- newsClassif[trainIndex, "shares"]
testX <- as.data.frame(newsClassif[testIndex, -61])
testY <- as.data.frame(newsClassif[testIndex, "shares"])

regfit.full = regsubsets(shares ~ ., data = train, method = "forward")
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 3 linear dependencies found

## Reordering variables and trying again:
```

```
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(shares ~ ., data = train, method = "forward")
## 59 Variables  (and intercept)
##                                 Forced in Forced out
## timedelta                           FALSE      FALSE
## n_tokens_title                      FALSE      FALSE
## n_tokens_content                    FALSE      FALSE
## n_unique_tokens                     FALSE      FALSE
## n_non_stop_words                  FALSE      FALSE
## n_non_stop_unique_tokens          FALSE      FALSE
## num_hrefs                         FALSE      FALSE
## num_self_hrefs                    FALSE      FALSE
## num_imgs                          FALSE      FALSE
## num_videos                        FALSE      FALSE
## average_token_length              FALSE      FALSE
## num_keywords                      FALSE      FALSE
## data_channel_is_lifestyle         FALSE      FALSE
## data_channel_is_entertainment     FALSE      FALSE
## data_channel_is_bus               FALSE      FALSE
## data_channel_is_socmed            FALSE      FALSE
## data_channel_is_tech              FALSE      FALSE
## data_channel_is_world             FALSE      FALSE
## kw_min_min                        FALSE      FALSE
## kw_max_min                        FALSE      FALSE
## kw_avg_min                        FALSE      FALSE
## kw_min_max                        FALSE      FALSE
## kw_max_max                        FALSE      FALSE
## kw_avg_max                        FALSE      FALSE
## kw_min_avg                        FALSE      FALSE
## kw_max_avg                        FALSE      FALSE
## kw_avg_avg                        FALSE      FALSE
## self_reference_min_shares         FALSE      FALSE
## self_reference_max_shares         FALSE      FALSE
## self_reference_avg_sharess        FALSE      FALSE
## weekday_is_monday                 FALSE      FALSE
## weekday_is_tuesday                FALSE      FALSE
## weekday_is_wednesday              FALSE      FALSE
## weekday_is_thursday               FALSE      FALSE
## weekday_is_friday                 FALSE      FALSE
## weekday_is_saturday               FALSE      FALSE
## LDA_00                            FALSE      FALSE
## LDA_01                            FALSE      FALSE
## LDA_02                            FALSE      FALSE
## LDA_03                            FALSE      FALSE
## global_subjectivity               FALSE      FALSE
## global_sentiment_polarity         FALSE      FALSE
## global_rate_positive_words        FALSE      FALSE
## global_rate_negative_words        FALSE      FALSE
## rate_positive_words               FALSE      FALSE
## rate_negative_words               FALSE      FALSE
## avg_positive_polarity             FALSE      FALSE
## min_positive_polarity             FALSE      FALSE
## max_positive_polarity             FALSE      FALSE
## avg_negative_polarity             FALSE      FALSE
## min_negative_polarity             FALSE      FALSE
## max_negative_polarity             FALSE      FALSE
## title_subjectivity                FALSE      FALSE
## title_sentiment_polarity          FALSE      FALSE
## abs_title_subjectivity            FALSE      FALSE
## abs_title_sentiment_polarity      FALSE      FALSE
## weekday_is_sunday                 FALSE      FALSE
## is_weekend                        FALSE      FALSE
## LDA_04                            FALSE      FALSE
## 1 subsets of each size up to 9
## Selection Algorithm: forward
##          timedelta n_tokens_title n_tokens_content n_unique_tokens
## 1  ( 1 ) " "       " "            " "              " "            
## 2  ( 1 ) " "       " "            " "              " "            
## 3  ( 1 ) " "       " "            " "              " "            
## 4  ( 1 ) " "       " "            " "              " "            
## 5  ( 1 ) " "       " "            " "              " "            
## 6  ( 1 ) " "       " "            " "              " "            
## 7  ( 1 ) " "       " "            " "              " "            
## 8  ( 1 ) " "       " "            " "              " "            
## 9  ( 1 ) " "       " "            "*"              " "            
##          n_non_stop_words n_non_stop_unique_tokens num_hrefs
## 1  ( 1 ) " "              " "                      " "      
## 2  ( 1 ) " "              " "                      " "      
## 3  ( 1 ) " "              " "                      " "      
## 4  ( 1 ) " "              " "                      " "      
## 5  ( 1 ) " "              " "                      " "      
## 6  ( 1 ) " "              " "                      " "      
## 7  ( 1 ) " "              " "                      " "      
## 8  ( 1 ) " "              " "                      " "      
## 9  ( 1 ) " "              " "                      " "      
##          num_self_hrefs num_imgs num_videos average_token_length
## 1  ( 1 ) " "            " "      " "        " "                 
## 2  ( 1 ) " "            " "      " "        " "                 
## 3  ( 1 ) " "            " "      " "        " "                 
## 4  ( 1 ) " "            " "      " "        " "                 
## 5  ( 1 ) " "            " "      " "        " "                 
## 6  ( 1 ) " "            " "      " "        " "                 
## 7  ( 1 ) " "            " "      " "        " "                 
## 8  ( 1 ) " "            " "      " "        " "                 
## 9  ( 1 ) " "            " "      " "        " "                 
##          num_keywords data_channel_is_lifestyle
## 1  ( 1 ) " "          " "                      
## 2  ( 1 ) " "          " "                      
## 3  ( 1 ) " "          " "                      
## 4  ( 1 ) " "          " "                      
## 5  ( 1 ) " "          " "                      
## 6  ( 1 ) " "          " "                      
## 7  ( 1 ) " "          " "                      
## 8  ( 1 ) " "          " "                      
## 9  ( 1 ) " "          " "                      
##          data_channel_is_entertainment data_channel_is_bus
## 1  ( 1 ) " "                           " "                
## 2  ( 1 ) " "                           " "                
## 3  ( 1 ) " "                           " "                
## 4  ( 1 ) " "                           " "                
## 5  ( 1 ) " "                           " "                
## 6  ( 1 ) " "                           " "                
## 7  ( 1 ) " "                           " "                
## 8  ( 1 ) "*"                           " "                
## 9  ( 1 ) "*"                           " "                
##          data_channel_is_socmed data_channel_is_tech data_channel_is_world
## 1  ( 1 ) " "                    " "                  " "
## 2  ( 1 ) " "                    " "                  " "
## 3  ( 1 ) " "                    "*"                  " "
## 4  ( 1 ) "*"                    "*"                  " "
## 5  ( 1 ) "*"                    "*"                  " "
## 6  ( 1 ) "*"                    "*"                  " "
## 7  ( 1 ) "*"                    "*"                  " "
## 8  ( 1 ) "*"                    "*"                  " "
## 9  ( 1 ) "*"                    "*"                  " "
##          kw_min_min kw_max_min kw_avg_min kw_min_max kw_max_max kw_avg_max
## 1  ( 1 ) " "        " "        " "        " "        " "        " "
## 2  ( 1 ) " "        " "        " "        " "        " "        " "
## 3  ( 1 ) " "        " "        " "        " "        " "        " "
## 4  ( 1 ) " "        " "        " "        " "        " "        " "
## 5  ( 1 ) " "        " "        " "        " "        " "        " "
## 6  ( 1 ) " "        " "        " "        " "        " "        " "
## 7  ( 1 ) " "        " "        " "        " "        " "        "*"
## 8  ( 1 ) " "        " "        " "        " "        " "        "*"
## 9  ( 1 ) " "        " "        " "        " "        " "        "*"
##          kw_min_avg kw_max_avg kw_avg_avg self_reference_min_shares
## 1  ( 1 ) " "        " "        "*"        " "
## 2  ( 1 ) " "        " "        "*"        " "
## 3  ( 1 ) " "        " "        "*"        " "
## 4  ( 1 ) " "        " "        "*"        " "
## 5  ( 1 ) " "        " "        "*"        " "
## 6  ( 1 ) " "        "*"        "*"        " "
## 7  ( 1 ) " "        "*"        "*"        " "
## 8  ( 1 ) " "        "*"        "*"        " "
## 9  ( 1 ) " "        "*"        "*"        " "
##          self_reference_max_shares self_reference_avg_sharess
## 1  ( 1 ) " "                       " "
## 2  ( 1 ) " "                       " "
## 3  ( 1 ) " "                       " "
## 4  ( 1 ) " "                       " "
## 5  ( 1 ) " "                       " "
## 6  ( 1 ) " "                       " "
## 7  ( 1 ) " "                       " "
## 8  ( 1 ) " "                       " "
## 9  ( 1 ) " "                       " "
##          weekday_is_monday weekday_is_tuesday weekday_is_wednesday
## 1  ( 1 ) " "               " "                " "
## 2  ( 1 ) " "               " "                " "
## 3  ( 1 ) " "               " "                " "
## 4  ( 1 ) " "               " "                " "
## 5  ( 1 ) " "               " "                " "
## 6  ( 1 ) " "               " "                " "
## 7  ( 1 ) " "               " "                " "
## 8  ( 1 ) " "               " "                " "
## 9  ( 1 ) " "               " "                " "
##          weekday_is_thursday weekday_is_friday weekday_is_saturday
## 1  ( 1 ) " "                 " "               " "
## 2  ( 1 ) " "                 " "               " "
## 3  ( 1 ) " "                 " "               " "
## 4  ( 1 ) " "                 " "               " "
## 5  ( 1 ) " "                 " "               " "
## 6  ( 1 ) " "                 " "               " "
## 7  ( 1 ) " "                 " "               " "
## 8  ( 1 ) " "                 " "               " "
## 9  ( 1 ) " "                 " "               " "
##          weekday_is_sunday is_weekend LDA_00 LDA_01 LDA_02 LDA_03 LDA_04
## 1  ( 1 ) " "               " "        " "    " "    " "    " "    " "
## 2  ( 1 ) " "               "*"        " "    " "    " "    " "    " "
## 3  ( 1 ) " "               "*"        " "    " "    " "    " "    " "
## 4  ( 1 ) " "               "*"        " "    " "    " "    " "    " "
## 5  ( 1 ) " "               "*"        "*"    " "    " "    " "    " "
## 6  ( 1 ) " "               "*"        "*"    " "    " "    " "    " "
## 7  ( 1 ) " "               "*"        "*"    " "    " "    " "    " "
## 8  ( 1 ) " "               "*"        "*"    " "    " "    " "    " "
## 9  ( 1 ) " "               "*"        "*"    " "    " "    " "    " "
##          global_subjectivity global_sentiment_polarity
## 1  ( 1 ) " "                 " "
## 2  ( 1 ) " "                 " "
## 3  ( 1 ) " "                 " "
## 4  ( 1 ) " "                 " "
## 5  ( 1 ) " "                 " "
## 6  ( 1 ) " "                 " "
## 7  ( 1 ) " "                 " "
## 8  ( 1 ) " "                 " "
## 9  ( 1 ) " "                 " "
##          global_rate_positive_words global_rate_negative_words
## 1  ( 1 ) " "                        " "
## 2  ( 1 ) " "                        " "
## 3  ( 1 ) " "                        " "
## 4  ( 1 ) " "                        " "
## 5  ( 1 ) " "                        " "
## 6  ( 1 ) " "                        " "
## 7  ( 1 ) " "                        " "
## 8  ( 1 ) " "                        " "
## 9  ( 1 ) " "                        " "
##          rate_positive_words rate_negative_words avg_positive_polarity
## 1  ( 1 ) " "                 " "                 " "
## 2  ( 1 ) " "                 " "                 " "
## 3  ( 1 ) " "                 " "                 " "
## 4  ( 1 ) " "                 " "                 " "
## 5  ( 1 ) " "                 " "                 " "
## 6  ( 1 ) " "                 " "                 " "
## 7  ( 1 ) " "                 " "                 " "
## 8  ( 1 ) " "                 " "                 " "
## 9  ( 1 ) " "                 " "                 " "
##          min_positive_polarity max_positive_polarity avg_negative_polarity
## 1  ( 1 ) " "                   " "                   " "
## 2  ( 1 ) " "                   " "                   " "
## 3  ( 1 ) " "                   " "                   " "
## 4  ( 1 ) " "                   " "                   " "
## 5  ( 1 ) " "                   " "                   " "
## 6  ( 1 ) " "                   " "                   " "
## 7  ( 1 ) " "                   " "                   " "
## 8  ( 1 ) " "                   " "                   " "
## 9  ( 1 ) " "                   " "                   " "
##          min_negative_polarity max_negative_polarity title_subjectivity
## 1  ( 1 ) " "                   " "                   " "
## 2  ( 1 ) " "                   " "                   " "
## 3  ( 1 ) " "                   " "                   " "
## 4  ( 1 ) " "                   " "                   " "
## 5  ( 1 ) " "                   " "                   " "
## 6  ( 1 ) " "                   " "                   " "
## 7  ( 1 ) " "                   " "                   " "
## 8  ( 1 ) " "                   " "                   " "
## 9  ( 1 ) " "                   " "                   " "
##          title_sentiment_polarity abs_title_subjectivity
## 1  ( 1 ) " "                      " "
## 2  ( 1 ) " "                      " "
## 3  ( 1 ) " "                      " "
## 4  ( 1 ) " "                      " "
## 5  ( 1 ) " "                      " "
## 6  ( 1 ) " "                      " "
## 7  ( 1 ) " "                      " "
## 8  ( 1 ) " "                      " "
## 9  ( 1 ) " "                      " "
##          abs_title_sentiment_polarity
## 1  ( 1 ) " "
## 2  ( 1 ) " "
## 3  ( 1 ) " "
## 4  ( 1 ) " "
## 5  ( 1 ) " "
## 6  ( 1 ) " "
## 7  ( 1 ) " "
## 8  ( 1 ) " "
## 9  ( 1 ) " "
```

```
# KNN REGRESSION

set.seed(1)
printf <- function(...) cat(sprintf(...))

newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0) # n

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel

newsClassif <- head(newsDataQuant, 2000) # current classification set we're using for all classificatio

gc()
```

```
##            used (Mb) gc trigger  (Mb) max used (Mb)
## Ncells   705599 37.7    2098485 112.1  1215585   65
## Vcells 11184890 85.4   33170445 253.1 28691946  219
```

```
library(StatMatch)
```

```
## Warning: package 'StatMatch' was built under R version 3.5.3
```

```
## Loading required package: proxy
## Warning: package 'proxy' was built under R version 3.5.3

##
## Attaching package: 'proxy'

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: clue

## Warning: package 'clue' was built under R version 3.5.3

## Loading required package: survey

## Warning: package 'survey' was built under R version 3.5.3

## Loading required package: Matrix

## Loading required package: survival

##
## Attaching package: 'survey'

## The following object is masked from 'package:graphics':
##
##     dotchart

## Loading required package: RANN

## Warning: package 'RANN' was built under R version 3.5.3

## Loading required package: lpSolve
```

```r
library(FastKNN)
```

```
## Warning: package 'FastKNN' was built under R version 3.5.3
```

```r
library(caret)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'caret'

## The following object is masked from 'package:survival':
##
##     cluster
```

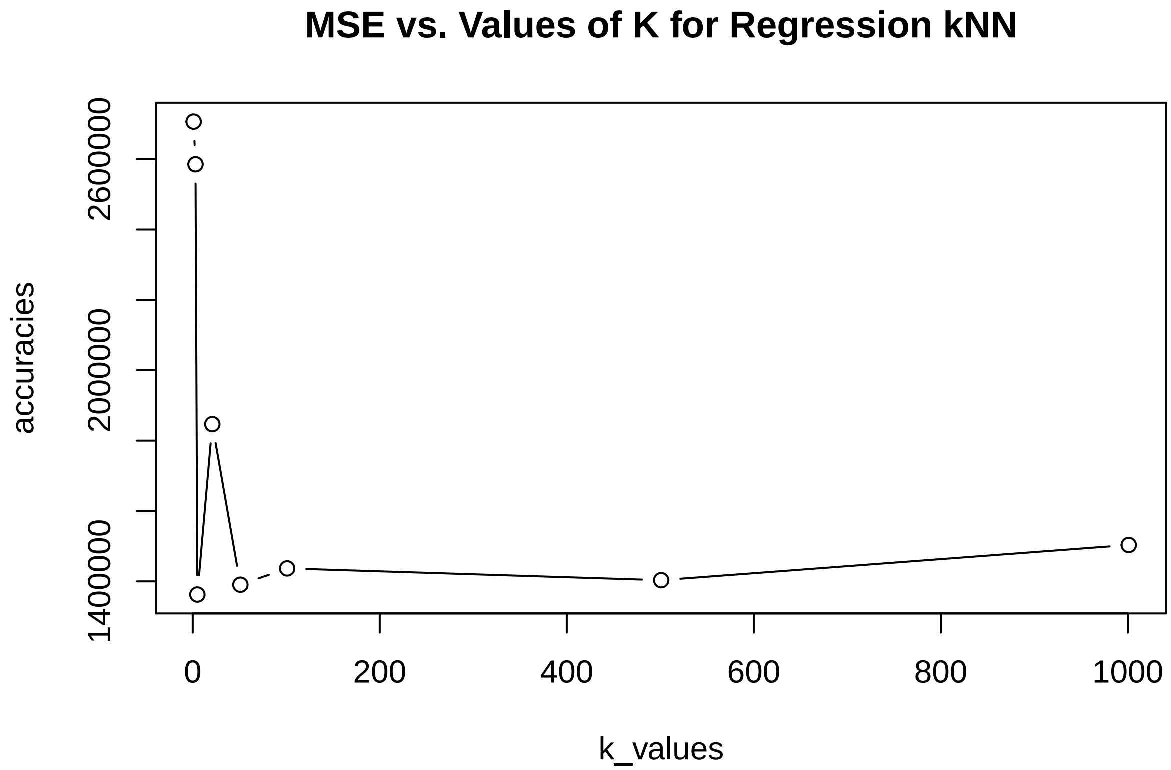
```r
library(FactoMineR)
```

```
## Warning: package 'FactoMineR' was built under R version 3.5.3
```

```r
fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
k_values <- c(1, 3, 5, 21, 51, 101, 501, 1001)
accuracies <- c()
for(j in k_values){
  acc <- 0
  for(i in 1:fold_n){
    # grab the i-th fold
    testIndices <- which(folds == i, arr.ind=TRUE)
    test <- newsClassif[testIndices,]
    train <- newsClassif[-testIndices,]
    trainX <- newsClassif[-testIndices, -61]
    trainY <- newsClassif[-testIndices, "shares"]
    testX <- as.data.frame(newsClassif[testIndices, -61])
    testY <- as.data.frame(newsClassif[testIndices, "shares"])

    trainYDF <- as.data.frame(trainY)

    gower.mat <- gower.dist(testX, trainX)
    newsKnn <- knn_test_function(trainX, testX, gower.mat, trainY, k = j)

    running_avg <- 0
    for(m in 1:length(testY)){
      nn <- k.nearest.neighbors(m, gower.mat, k = j)
      avg <- mean(trainYDF[nn, ])
      se <- (avg - testY[m, ])^2
      running_avg <- running_avg + se
    }
    running_avg <- running_avg / length(testY)

    acc <- acc + running_avg
  }
  acc <- acc/fold_n
  printf("kNN with k = %d accuracy: %f\n", j, acc)
  accuracies <- c(accuracies, acc)
}
```

```
## kNN with k = 1 accuracy: 2706673.800000
## kNN with k = 3 accuracy: 2585432.422222
## kNN with k = 5 accuracy: 1362651.776000
## kNN with k = 21 accuracy: 1846831.116100
## kNN with k = 51 accuracy: 1390526.200923
## kNN with k = 101 accuracy: 1436841.161827
## kNN with k = 501 accuracy: 1403364.750739
## kNN with k = 1001 accuracy: 1503440.271418
```

```
plot(k_values, accuracies, type="b", main="MSE vs. Values of K for Regression kNN")
```



```r
# KNN BINARY CLASSIFICATION

set.seed(1)
printf <- function(...) cat(sprintf(...))

newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0) # n

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel

newsClassif <- head(newsDataBinary, 2000) # current classification set we're using for all classificati

gc()
```

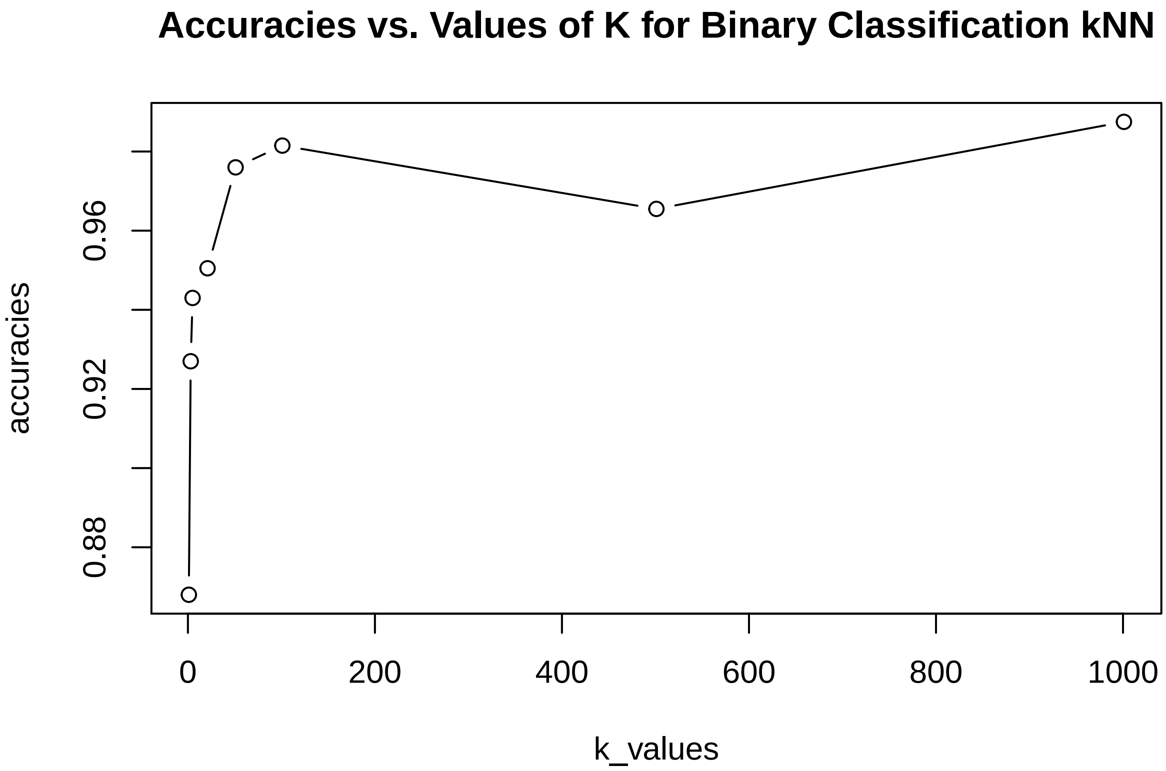
```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2114513 113.0    3487092 186.3  3487092 186.3
## Vcells 10064272  76.8   33170513 253.1 33170513 253.1
```

```r
library(StatMatch)
library(FastKNN)
library(caret)
library(FactoMineR)

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
k_values <- c(1, 3, 5, 21, 51, 101, 501, 1001)
```

```
accuracies <- c()
for(j in k_values){
  acc <- 0
  for(i in 1:fold_n){
    #printf("i: %d\n", i) # print the current fold iteration

    # grab the i-th fold
    testIndices <- which(folds == i, arr.ind=TRUE)
    test <- newsClassif[testIndices,]
    train <- newsClassif[-testIndices,]
    trainX <- newsClassif[-testIndices, -61]
    trainY <- newsClassif[-testIndices, "shares"]
    testX <- as.data.frame(newsClassif[testIndices, -61])
    testY <- as.data.frame(newsClassif[testIndices, "shares"])

    trainYDF <- as.data.frame(trainY)

    gower.mat <- gower.dist(testX, trainX)
    newsKnn <- knn_test_function(trainX, testX, gower.mat, trainY, k = j)

    conf_matrix <- table(newsKnn, t(testY)) # confusion matrix
    acc <- acc + sum(diag(conf_matrix))/sum(conf_matrix)
  }
  acc <- acc/fold_n
  printf("kNN with k = %d accuracy: %f\n", j, acc)
  accuracies <- c(accuracies, acc)
}
```

```
## kNN with k = 1 accuracy: 0.868000
## kNN with k = 3 accuracy: 0.927000
## kNN with k = 5 accuracy: 0.943000
## kNN with k = 21 accuracy: 0.950500
## kNN with k = 51 accuracy: 0.976000
## kNN with k = 101 accuracy: 0.981500
## kNN with k = 501 accuracy: 0.965500
## kNN with k = 1001 accuracy: 0.987500
```

```
plot(k_values, accuracies, type="b", main="Accuracies vs. Values of K for Binary Classification kNN")
```



```
# KNN THREE-WAY CLASSIFICATION

set.seed(1)
printf <- function(...) cat(sprintf(...))

newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0) # n

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel

newsClassif <- head(newsDataTrinary, 2000) # current classification set we're using for all classificat

gc()
```

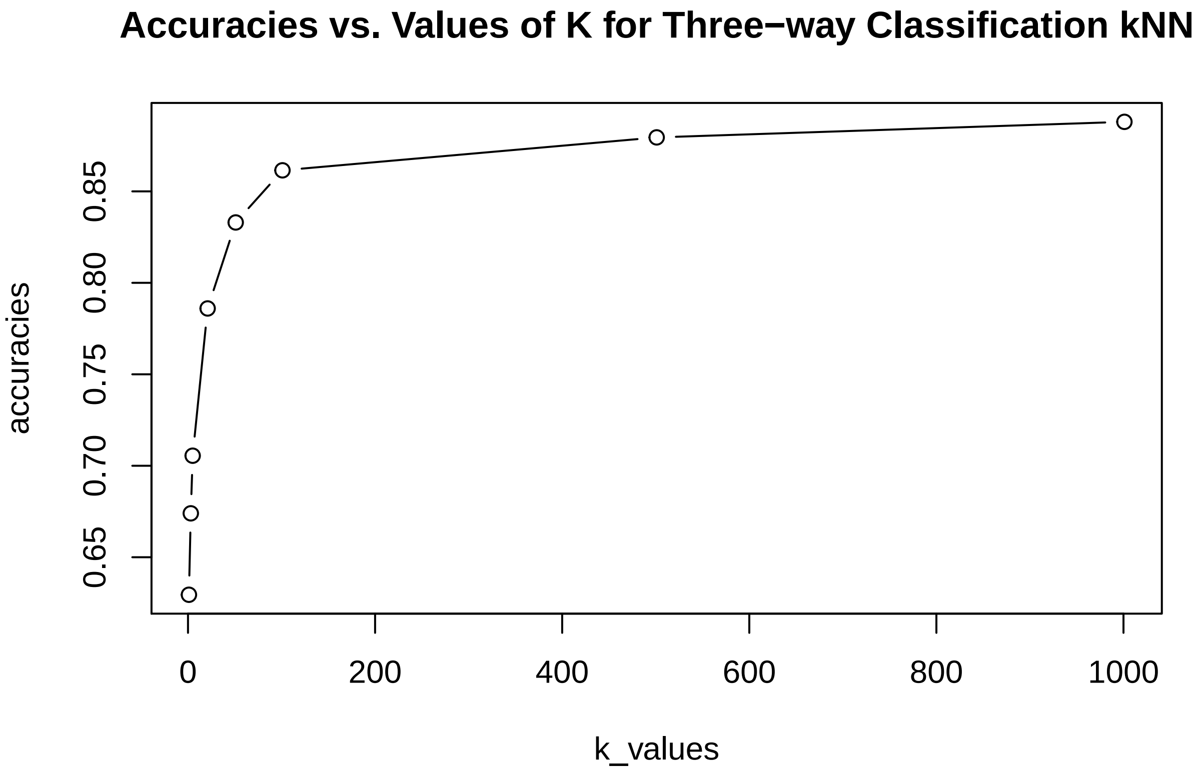
```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2117407 113.1    3487092 186.3  3487092 186.3
## Vcells 10078592  76.9   33170513 253.1 33170513 253.1
```

```
library(StatMatch)
library(FastKNN)
library(caret)
library(FactoMineR)

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
k_values <- c(1, 3, 5, 21, 51, 101, 501, 1001)
```

```
accuracies <- c()
for(j in k_values){
  acc <- 0
  for(i in 1:fold_n){
    # grab the i-th fold
    testIndices <- which(folds == i, arr.ind=TRUE)
    test <- newsClassif[testIndices,]
    train <- newsClassif[-testIndices,]
    trainX <- newsClassif[-testIndices, -61]
    trainY <- newsClassif[-testIndices, "shares"]
    testX <- as.data.frame(newsClassif[testIndices, -61])
    testY <- as.data.frame(newsClassif[testIndices, "shares"])

    trainYDF <- as.data.frame(trainY)

    gower.mat <- gower.dist(testX, trainX)
    newsKnn <- knn_test_function(trainX, testX, gower.mat, trainY, k = j)

    conf_matrix <- table(newsKnn, t(testY)) # confusion matrix
    acc <- acc + sum(diag(conf_matrix))/sum(conf_matrix)
  }
  acc <- acc/fold_n
  printf("kNN with k = %d accuracy: %f\n", j, acc)
  accuracies <- c(accuracies, acc)
}
```

```
## kNN with k = 1 accuracy: 0.629500
## kNN with k = 3 accuracy: 0.674000
## kNN with k = 5 accuracy: 0.705500
## kNN with k = 21 accuracy: 0.786000
## kNN with k = 51 accuracy: 0.833000
## kNN with k = 101 accuracy: 0.861500
## kNN with k = 501 accuracy: 0.879500
## kNN with k = 1001 accuracy: 0.888000
```

```
plot(k_values, accuracies, type="b", main="Accuracies vs. Values of K for Three-way Classification kNN")
```



```
# RANDOM FORESTS REGRESSION

library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.3
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

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

```
library(leaps)
set.seed(10)
gc()
```

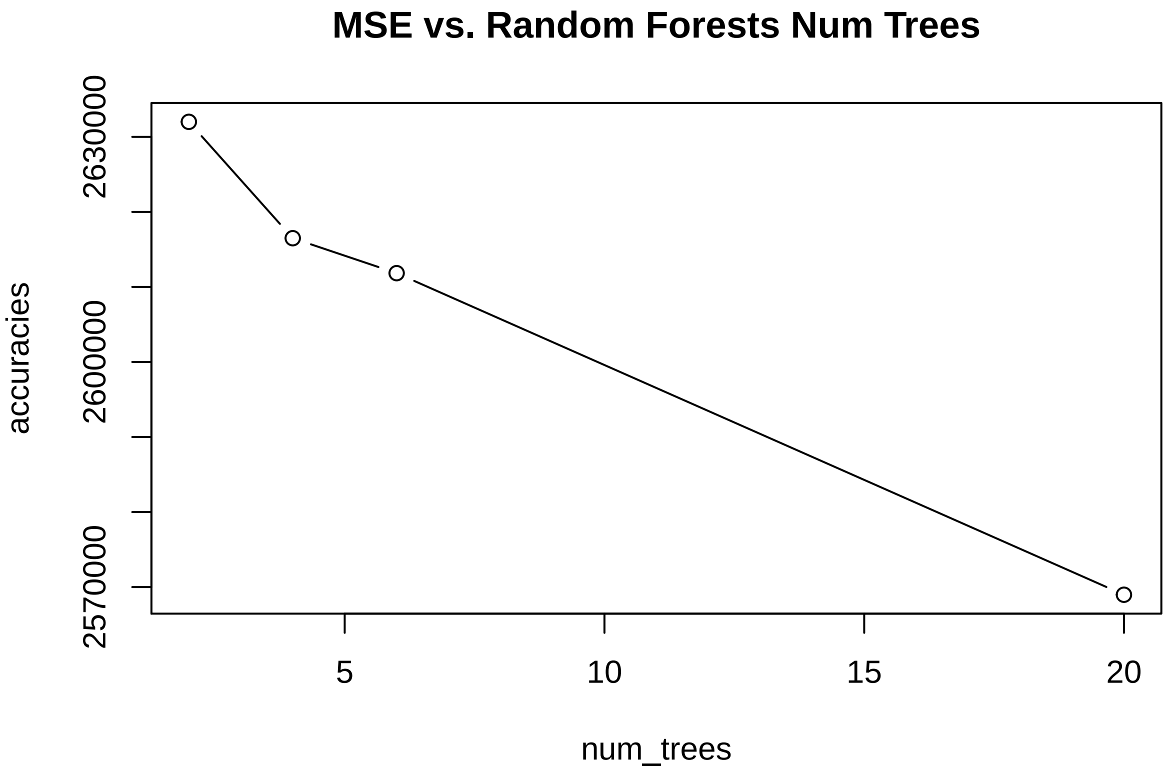
```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2120066 113.3    3487092 186.3  3487092 186.3
## Vcells 10087085  77.0   33170513 253.1 33170513 253.1
```

```
newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0)

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel
```

```r
newsClassif <- head(newsDataQuant, 5000) # current classification set we're using for all classificatio

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
num_trees <- c(2, 4, 6, 20)#, 50, 100, 500, 1000)
rand_factor <- c(2, 4, 8, 16, 32, 55)
#num_trees <- c(8)
accuracies <- c()
rsqs <- c()
for(j in num_trees){
  acc <- 0 # mse
  rs <- 0 # r-squared
  for(i in 1:fold_n){
    # grab the i-th fold
    testIndices <- which(folds == i, arr.ind=TRUE)
    test <- newsClassif[testIndices,]
    train <- newsClassif[-testIndices,]
    trainX <- newsClassif[-testIndices, -61]
    trainY <- as.factor(newsClassif[-testIndices, "shares"])
    testX <- as.data.frame(newsClassif[testIndices, -61])
    testY <- as.data.frame(newsClassif[testIndices, "shares"])

    forest <- randomForest(x = trainX, y = trainY, ntree = j, mtry = 8)
    forestPred <- as.numeric(predict(forest, newdata = testX))

    acc <- acc + mean((forestPred - t(testY))^2) # get the MSE
    #rs <- rs + mean(forest$rsq)
  }
  acc <- acc/fold_n
  #rs <- rs/fold_n
  printf("Random Forests with num trees = %d MSE: %f, r-squared: %f\n", j, acc, rs)
  accuracies <- c(accuracies, acc)
  #rsqs <- c(rsqs, rs)
}
```

```
## Random Forests with num trees = 2 MSE: 2632002.943800, r-squared: 0.000000
## Random Forests with num trees = 4 MSE: 2616497.318000, r-squared: 0.000000
## Random Forests with num trees = 6 MSE: 2611837.957200, r-squared: 0.000000
## Random Forests with num trees = 20 MSE: 2568978.719200, r-squared: 0.000000
```

```r
plot(num_trees, accuracies, type="b", main="MSE vs. Random Forests Num Trees")
```



```
#plot(num_trees, rsqs, type="b", main="R squared vs. Random Forests Num Trees")
```

```
# RANDOM FORESTS REGRESSION

library(randomForest)
library(leaps)
set.seed(10)
gc()
```

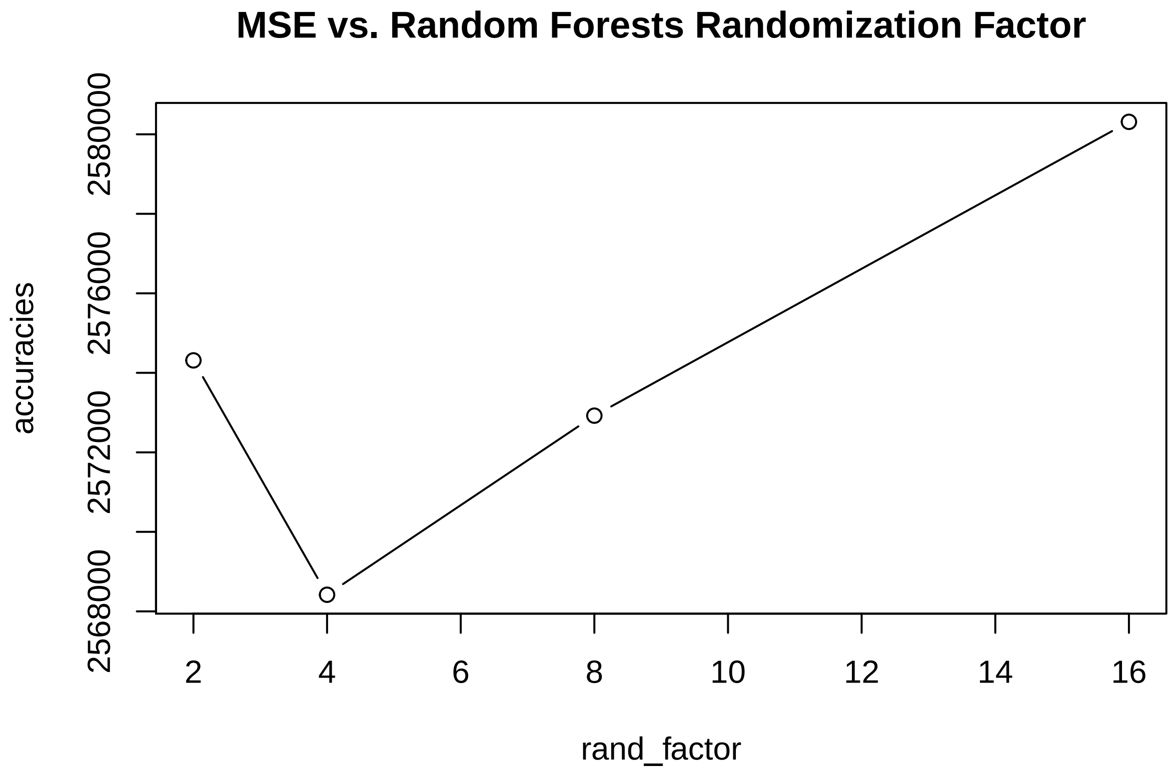
```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2132546 113.9    3487092 186.3  3487092 186.3
## Vcells 13665588 104.3   39884615 304.3 39884592 304.3
```

```
newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0)

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel

newsClassif <- head(newsDataQuant, 5000) # current classification set we're using for all classificatio

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
#num_trees <- c(2, 4, 6, 20, 50, 100, 500, 1000)
rand_factor <- c(2, 4, 8, 16)#, 32, 55)
accuracies <- c()
#rsqs <- c()
```

```
for(j in rand_factor){
  acc <- 0 # mse
  rs <- 0 # r-squared
  for(i in 1:fold_n){
    # grab the i-th fold
    testIndices <- which(folds == i, arr.ind=TRUE)
    test <- newsClassif[testIndices,]
    train <- newsClassif[-testIndices,]
    trainX <- newsClassif[-testIndices, -61]
    trainY <- as.factor(newsClassif[-testIndices, "shares"])
    testX <- as.data.frame(newsClassif[testIndices, -61])
    testY <- as.data.frame(newsClassif[testIndices, "shares"])

    forest <- randomForest(x = trainX, y = trainY, ntree = 20, mtry = j)
    forestPred <- as.numeric(predict(forest, newdata = testX))

    acc <- acc + mean((forestPred - t(testY))^2) # get the MSE
    #rs <- rs + mean(forest$rsq)
  }
  acc <- acc/fold_n
  #rs <- rs/fold_n
  printf("Random Forests with rand factor = %d MSE: %f, r-squared: %f\n", j, acc, rs)
  accuracies <- c(accuracies, acc)
  #rsqs <- c(rsqs, rs)
}
```

```
## Random Forests with rand factor = 2 MSE: 2574313.128400, r-squared: 0.000000
## Random Forests with rand factor = 4 MSE: 2568418.018600, r-squared: 0.000000
## Random Forests with rand factor = 8 MSE: 2572922.369800, r-squared: 0.000000
## Random Forests with rand factor = 16 MSE: 2580313.041400, r-squared: 0.000000
```

```
plot(rand_factor, accuracies, type="b", main="MSE vs. Random Forests Randomization Factor")
```



```r
#plot(rand_factor, rsqs, type="b", main="R squared vs. Random Forests Randomization Factor")
```

```r
# RANDOM FORESTS REGRESSION

library(randomForest)
library(leaps)
set.seed(10)
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2132487 113.9    3487092 186.3  3487092 186.3
## Vcells 13637111 104.1   39884615 304.3 39884592 304.3
```

```r
newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0)

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel

newsClassif <- head(newsDataQuant, 5000) # current classification set we're using for all classificatio

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
accuracies <- c()
rsqs <- c()
acc <- 0 # mse
#rs <- 0 # r-squared
```

```r
for(i in 1:fold_n){
  # grab the i-th fold
  testIndices <- which(folds == i, arr.ind=TRUE)
  test <- newsClassif[testIndices,]
  train <- newsClassif[-testIndices,]
  trainX <- newsClassif[-testIndices, -61]
  trainY <- as.factor(newsClassif[-testIndices, "shares"])
  testX <- as.data.frame(newsClassif[testIndices, -61])
  testY <- as.data.frame(newsClassif[testIndices, "shares"])

  forest <- randomForest(x = trainX, y = trainY, ntree = 20, mtry = 8)
  forestPred <- as.numeric(predict(forest, newdata = testX))

  acc <- acc + mean((forestPred - t(testY))^2) # get the MSE
  #rs <- rs + mean(forest$rsq)
}
acc <- acc/fold_n
#rs <- rs/fold_n
printf("Random Forests with num trees = 20, randomization factor = 8; MSE: %f, r-squared: %f\n", acc, r
```

```
## Random Forests with num trees = 20, randomization factor = 8; MSE: 2573930.791600, r-squared: 0.00000
```

```r
# RANDOM FORESTS BINARY CLASSIFICATION

library(randomForest)
library(leaps)
set.seed(10)
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2132469 113.9    3487092 186.3  3487092 186.3
## Vcells 13658669 104.3   39884615 304.3 39884592 304.3
```

```r
newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0)

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifel

newsClassif <- head(newsDataBinary, 5000) # current classification set we're using for all classificati

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
acc <- 0 # mse
for(i in 1:fold_n){
  # grab the i-th fold
  testIndices <- which(folds == i, arr.ind=TRUE)
  test <- newsClassif[testIndices,]
  train <- newsClassif[-testIndices,]
  trainX <- newsClassif[-testIndices, -61]
  trainY <- as.factor(newsClassif[-testIndices, "shares"])
  testX <- as.data.frame(newsClassif[testIndices, -61])
  testY <- as.data.frame(newsClassif[testIndices, "shares"])

  forest <- randomForest(x = trainX, y = trainY, ntree = 20, mtry = 8)
  forestPred <- predict(forest, newdata = testX)

  conf_matrix <- table(forestPred, t(testY)) # confusion matrix
  acc <- acc + sum(diag(conf_matrix))/sum(conf_matrix)
}
acc <- acc/fold_n
printf("Random Forests with num trees = 20, randomization factor = 8; accuracy: %f\n", acc)
```

```
## Random Forests with num trees = 20, randomization factor = 8; accuracy: 1.000000
```

```
# RANDOM FORESTS THREE-WAY CLASSIFICATION

library(randomForest)
library(leaps)
set.seed(10)
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2133028 114.0    3487092 186.3  3487092 186.3
## Vcells 10728963  81.9   39884615 304.3 39884592 304.3
```

```
newsDataBinary <- data.frame(newsDataQuant) # make a copy
newsDataBinary$shares <- ifelse(newsDataBinary$shares > quantile(newsDataBinary$shares, 0.5), 1, 0)

newsDataTrinary <- data.frame(newsDataQuant) # make a copy
newsDataTrinary$shares <- ifelse(newsDataTrinary$shares > quantile(newsDataTrinary$shares, 0.333), ifels

newsClassif <- head(newsDataTrinary, 5000) # current classification set we're using for all classificat

fold_n = 5
folds <- cut(seq(1, nrow(newsClassif)), breaks = fold_n, labels = FALSE)
acc <- 0 # mse
for(i in 1:fold_n){
  # grab the i-th fold
  testIndices <- which(folds == i, arr.ind=TRUE)
  test <- newsClassif[testIndices,]
  train <- newsClassif[-testIndices,]
  trainX <- newsClassif[-testIndices, -61]
  trainY <- as.factor(newsClassif[-testIndices, "shares"])
  testX <- as.data.frame(newsClassif[testIndices, -61])
  testY <- as.data.frame(newsClassif[testIndices, "shares"])

  forest <- randomForest(x = trainX, y = trainY, ntree = 20, mtry = 8)
  forestPred <- predict(forest, newdata = testX)

  conf_matrix <- table(forestPred, t(testY)) # confusion matrix
  acc <- acc + sum(diag(conf_matrix))/sum(conf_matrix)
}
acc <- acc/fold_n
printf("Random Forests with num trees = 20, randomization factor = 8; accuracy: %f\n", acc)
```

```
## Random Forests with num trees = 20, randomization factor = 8; accuracy: 0.998200
```