MATH189_Final_Project

Walter Wong, Trisna Nguyen, Yiheng Yuan

2023-03-14

Contribution Highlighting

Walter Wong: Problem discussion and communicating, writing R code

Trisna Nguyen: Problem discussion and communicating, Online searching for answers

Yiheng Yuan: Problem discussion and communicating, RMarkDown formatting

Spam Classification

Consider an email spam dataset that consists of 4601 email messages, from which 57 features have been extracted. These features are described as follows:

- 48 features giving the percentage of certain words (e.g., "business", "free", "george") in a given message
- 6 features giving the percentage of certain characters (; ([! $ #)
- feature 55: the average length of an uninterrupted sequence of capital letters
- feature 56: the length of the longest uninterrupted sequence of capital letters
- feature 57: the sum of the lengths of uninterrupted sequences of capital letters

The data set contains a training set of size 3065 (link), and a test set of size 1536 (link). One can perform several types of preprocessing to this data. Try each of the following separately:

Data Preparation:

```
train <- read.csv("/Users/jeffyuan/Desktop/spam-train.txt", sep = ',', header = FALSE)
test <- read.csv("/Users/jeffyuan/Desktop/spam-test.txt", sep = ',', header = FALSE)
```

1) Standardize the columns so that they all have zero mean and unit variance;

```
train_scale <- as.data.frame(scale(train[,1:57]))
train_scale_all <- cbind(train_scale, data.frame(V58 = c(train[,58])))

test_scale <- as.data.frame(scale(test[,1:57]))
test_scale_all <- cbind(test_scale, data.frame(V58 = c(test[,58])))
```

2) Transform the features using $log(x_{ij} + 1)$;

```
train_log <- log(train[,1:57] + 1)
train_log_all <- cbind(train_log, data.frame(V58 = c(train[,58])))

test_log <- log(test[,1:57] + 1)
test_log_all <- cbind(test_log, data.frame(V58 = c(test[,58])))
```

3) Discretize each feature using $I(x_{ij} > 0)$.

```r
train_i <- as.data.frame(train[,1:57] > 0)
train_i_all <- cbind(train_i, data.frame(V58 = c(train[,58])))

test_i <- as.data.frame(test[,1:57] > 0)
test_i_all <- cbind(test_i, data.frame(V58 = c(test[,58])))
```

(a)

For each version of the data, visualize it using the tools introduced in the class.

Answer:

Highly Associated Pairs on Standardize Training Dataset:

```
cor_train_scale <- cor(train_scale)
subset(melt(cor_train_scale), value < 1 & value > 0.9)
```

```
## Warning in melt(cor_train_scale): The melt generic in data.table has been
## passed a matrix and will attempt to redirect to the relevant reshape2
## method; please note that reshape2 is deprecated, and this redirection is now
## deprecated as well. To continue using melt methods from reshape2 while both
## libraries are attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(cor_train_scale). In the next version, this warning will become
## an error.
```

```
##      Var1 Var2     value
## 1801  V34  V32 0.9909226
## 1913  V32  V34 0.9909226
```

Scatter Plot on Standardize Training Dataset:

```
ggplot(train_scale,aes(x=V34, y=V32)) + geom_point() + labs(title="Scatter plot on standardize training
```

Highly Associated Pairs on Standardize Testing Dataset:

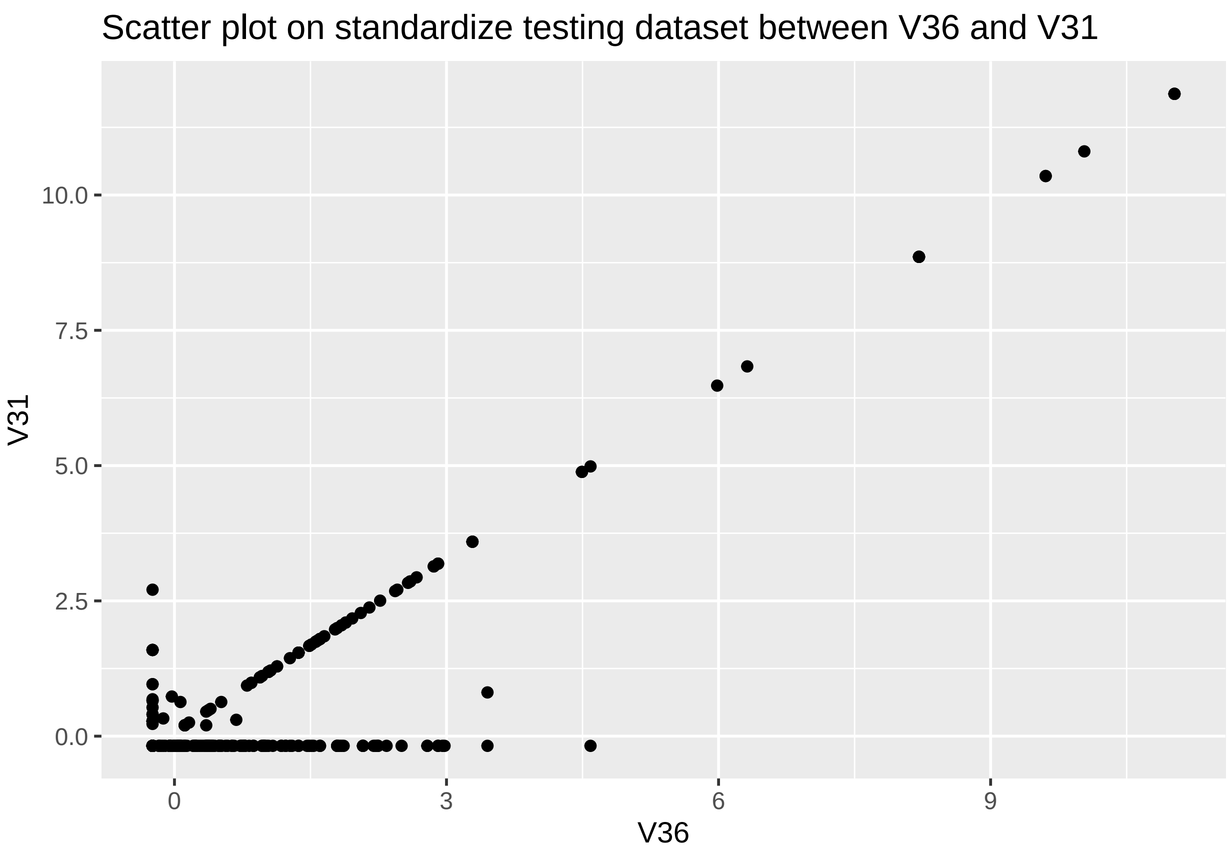
```
cor_test_scale <- cor(test_scale)
subset(melt(cor_test_scale), value < 1 & value > 0.9)
```

```
## Warning in melt(cor_test_scale): The melt generic in data.table has been
## passed a matrix and will attempt to redirect to the relevant reshape2
## method; please note that reshape2 is deprecated, and this redirection is now
## deprecated as well. To continue using melt methods from reshape2 while both
## libraries are attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(cor_test_scale). In the next version, this warning will become an
## error.
```

```
##      Var1 Var2     value
## 1746  V36  V31 0.9062135
## 1801  V34  V32 0.9968465
## 1913  V32  V34 0.9968465
## 2026  V31  V36 0.9062135
```

Scatter Plot on Standardize Testing Dataset:

```r
ggplot(test_scale, aes(x=V36, y=V31)) + geom_point() + labs(title="Scatter plot on standardize testing d
```




```r
ggplot(test_scale, aes(x=V34, y=V32)) + geom_point() + labs(title="Scatter plot on standardize testing d
```

Highly Associated Pairs on Log Transform Training Dataset:

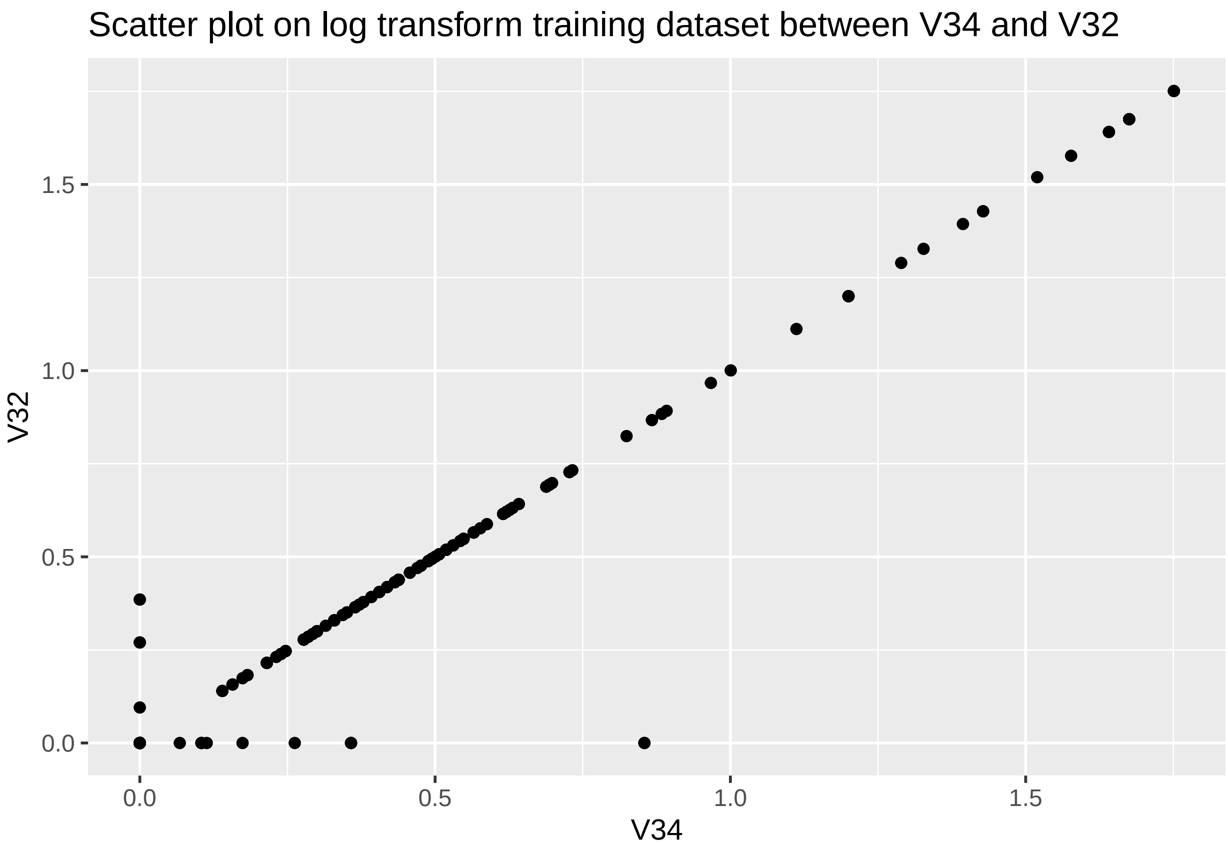
```
cor_train_log <- cor(train_log)
subset(melt(cor_train_log), value < 1 & value > 0.9)
```

```
## Warning in melt(cor_train_log): The melt generic in data.table has been
## passed a matrix and will attempt to redirect to the relevant reshape2
## method; please note that reshape2 is deprecated, and this redirection is now
## deprecated as well. To continue using melt methods from reshape2 while both
## libraries are attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(cor_train_log). In the next version, this warning will become an
## error.
```

```
##      Var1 Var2     value
## 1801  V34  V32 0.9814491
## 1913  V32  V34 0.9814491
```

Scatter Plot on Log Transform Training Dataset:

```
ggplot(train_log, aes(x=V34, y=V32)) + geom_point() + labs(title="Scatter plot on log transform training
```



Highly Associated Pairs on Log Transform Testing Dataset:

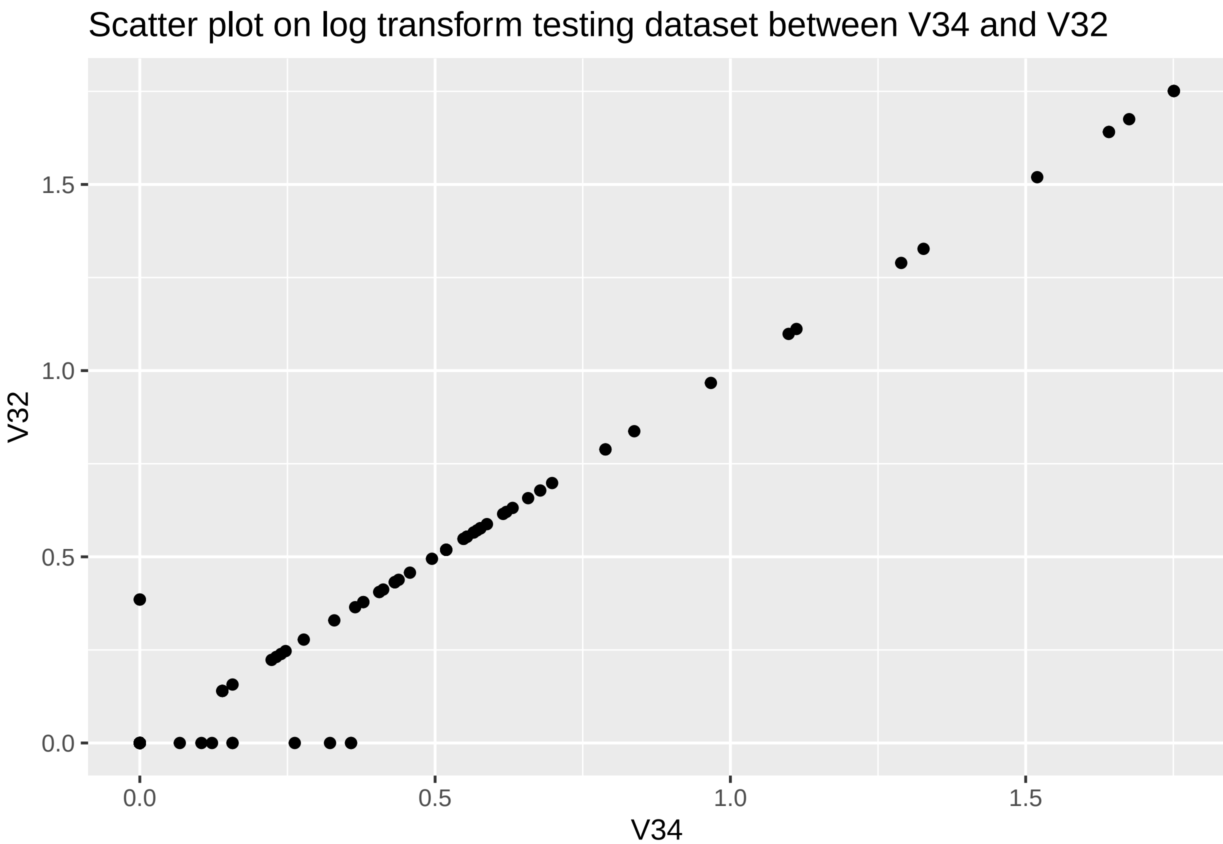
```
cor_test_log <- cor(test_log)
subset(melt(cor_test_log), value < 1 & value > 0.9)
```

```
## Warning in melt(cor_test_log): The melt generic in data.table has been passed a
## matrix and will attempt to redirect to the relevant reshape2 method; please note
## that reshape2 is deprecated, and this redirection is now deprecated as well.
## To continue using melt methods from reshape2 while both libraries are attached,
## e.g. melt.list, you can prepend the namespace like reshape2::melt(cor_test_log).
## In the next version, this warning will become an error.
```

```
##      Var1 Var2     value
## 1801  V34  V32 0.9895631
## 1913  V32  V34 0.9895631
```

Scatter Plot on Log Transform Training Dataset:

```
ggplot(test_log, aes(x=V34, y=V32)) + geom_point() + labs(title="Scatter plot on log transform testing d
```



Highly Associated Pairs on Discretize Training Dataset:

```
cor_train_i <- cor(train_i)
```

```
## Warning in cor(train_i): the standard deviation is zero
```

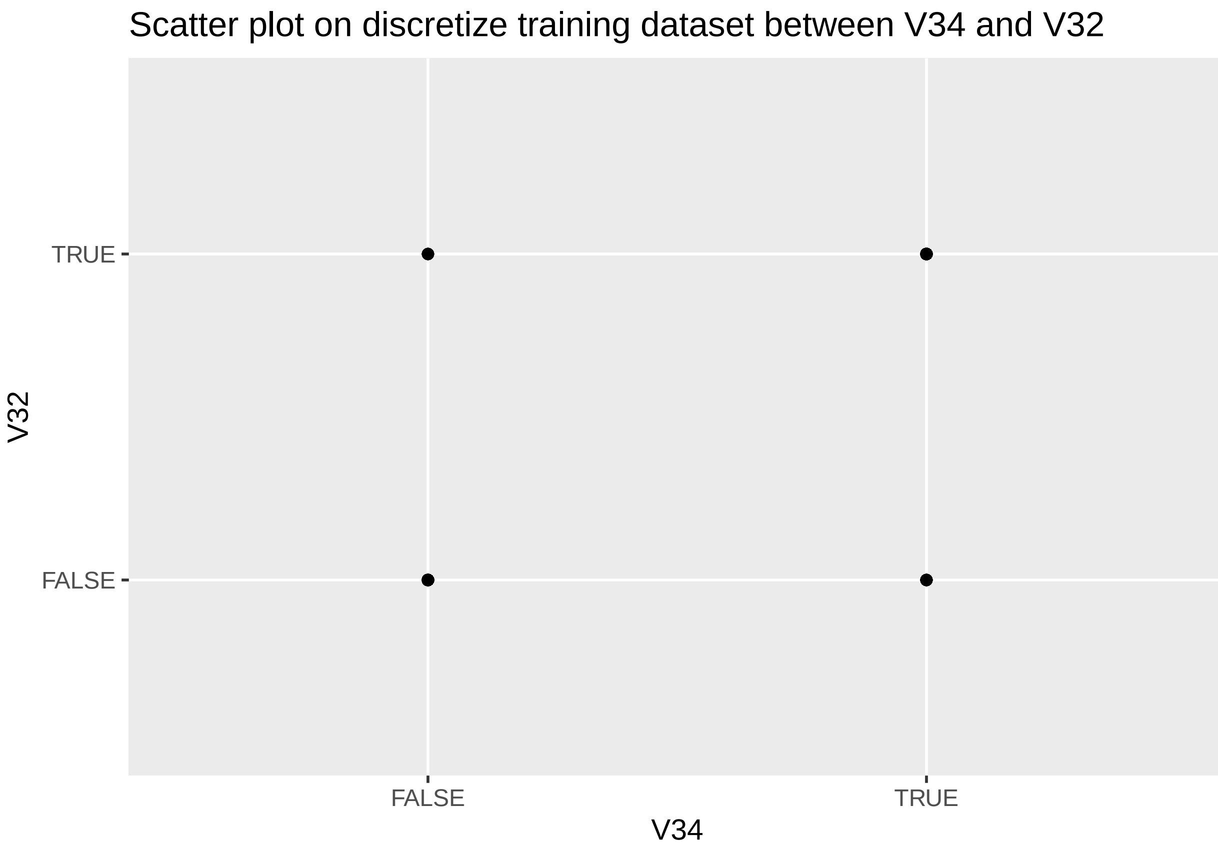
```
subset(melt(cor_train_i), value < 1 & value > 0.9)
```

```
## Warning in melt(cor_train_i): The melt generic in data.table has been passed a
## matrix and will attempt to redirect to the relevant reshape2 method; please note
## that reshape2 is deprecated, and this redirection is now deprecated as well.
## To continue using melt methods from reshape2 while both libraries are attached,
## e.g. melt.list, you can prepend the namespace like reshape2::melt(cor_train_i).
## In the next version, this warning will become an error.
```

```
##      Var1 Var2     value
## 1801  V34  V32 0.9373409
## 1913  V32  V34 0.9373409
```

Scatter Plot on Discretize Training Dataset:

```
ggplot(train_i, aes(x=V34, y=V32)) + geom_point() + labs(title="Scatter plot on discretize training data
```



Highly Associated Pairs on Discretize Testing Dataset:

```
cor_test_i <- cor(test_i)
```

```
## Warning in cor(test_i): the standard deviation is zero
```

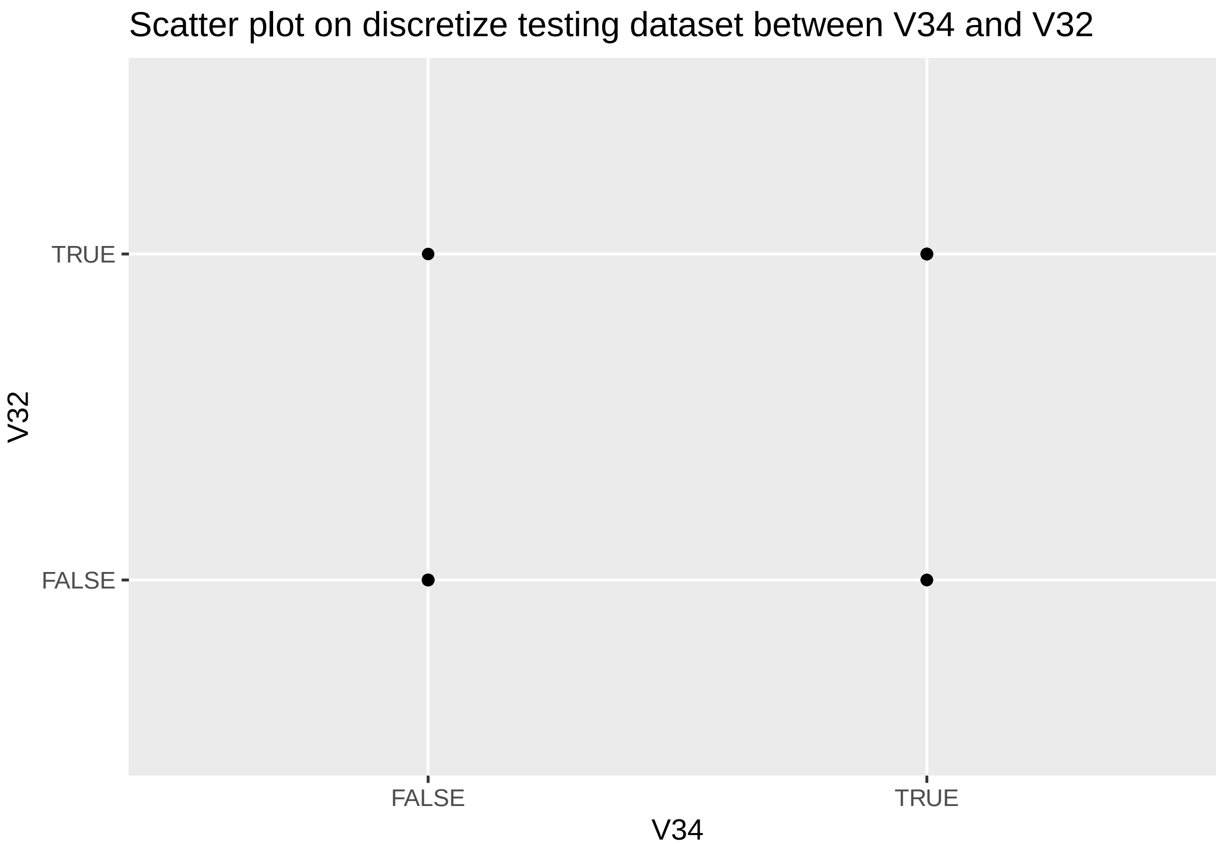
```
subset(melt(cor_test_i), value < 1 & value > 0.9)
```

```
## Warning in melt(cor_test_i): The melt generic in data.table has been passed a
## matrix and will attempt to redirect to the relevant reshape2 method; please note
## that reshape2 is deprecated, and this redirection is now deprecated as well.
## To continue using melt methods from reshape2 while both libraries are attached,
## e.g. melt.list, you can prepend the namespace like reshape2::melt(cor_test_i).
## In the next version, this warning will become an error.
```

```
##      Var1 Var2     value
## 1801  V34  V32 0.9172778
## 1913  V32  V34 0.9172778
```

Scatter Plot on Discretize Testing Dataset:

```
ggplot(test_i, aes(x=V34, y=V32)) + geom_point() + labs(title="Scatter plot on discretize testing datas
```


Scatter plot on discretize testing dataset between V34 and V32
TRUE
FALSE
V32
FALSE
TRUE
V34

(b)

For each version of the data, fit a logistic regression model. Interpret the results, and report the classification errors on both the training and test sets. Do any of the 57 features/ predictors appear to be statistically significant? If so, which ones?

Answer:

Standardize Training Dataset:

```
glm_fits_scale <- glm(V58 ~ ., data = train_scale_all, family="binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm_fits_scale)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = "binomial", data = train_scale_all)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.3245  -0.1988  -0.0001   0.0940   3.6053
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.36294    1.76165  -4.180 2.92e-05 ***
## V1           -0.07047    0.08544  -0.825 0.409508
## V2           -0.21268    0.13656  -1.557 0.119379
## V3            0.02573    0.07472   0.344 0.730612
## V4            5.42487    2.63430   2.059 0.039464 *
## V5            0.41029    0.08897   4.611 4.00e-06 ***
## V6            0.08488    0.05780   1.469 0.141965
## V7            1.30763    0.19827   6.595 4.24e-11 ***
## V8            0.20112    0.07309   2.752 0.005931 **
## V9            0.21642    0.10039   2.156 0.031095 *
## V10           0.05737    0.06090   0.942 0.346145
## V11          -0.19561    0.07523  -2.600 0.009319 **
## V12          -0.03552    0.07302  -0.486 0.626655
## V13          -0.13217    0.11069  -1.194 0.232431
## V14          -0.00339    0.06296  -0.054 0.957058
## V15           0.31084    0.23239   1.338 0.181023
## V16           1.10038    0.16449   6.690 2.24e-11 ***
## V17           0.59641    0.13999   4.260 2.04e-05 ***
## V18          -0.02993    0.08391  -0.357 0.721327
## V19           0.15357    0.07781   1.974 0.048423 *
## V20           1.80199    0.50899   3.540 0.000400 ***
## V21           0.49973    0.08500   5.879 4.13e-09 ***
## V22           0.10473    0.15871   0.660 0.509332
## V23           1.17267    0.24101   4.866 1.14e-06 ***
## V24           0.09945    0.06169   1.612 0.106930
## V25          -3.27164    0.58150  -5.626 1.84e-08 ***
## V26          -0.44855    0.39100  -1.147 0.251312
## V27         -18.55268    3.80185  -4.880 1.06e-06 ***
## V28           0.24526    0.17081   1.436 0.151031
```

```
## V29           -2.42887    1.66214  -1.461 0.143936
## V30            0.01145    0.09666   0.118 0.905705
## V31           -0.08296    0.25709  -0.323 0.746941
## V32           -0.37441    0.95348  -0.393 0.694553
## V33           -0.46280    0.24665  -1.876 0.060610 .
## V34            0.85386    1.01167   0.844 0.398662
## V35           -0.61202    0.35339  -1.732 0.083302 .
## V36            0.07618    0.16958   0.449 0.653264
## V37           -0.26049    0.14890  -1.749 0.080214 .
## V38           -0.15147    0.12133  -1.248 0.211871
## V39           -0.02633    0.15297  -0.172 0.863349
## V40           -0.15745    0.17675  -0.891 0.373028
## V41          -18.56408   12.22870  -1.518 0.128996
## V42           -1.69535    0.58310  -2.907 0.003644 **
## V43           -0.45417    0.23919  -1.899 0.057599 .
## V44           -0.73394    0.35711  -2.055 0.039857 *
## V45           -0.88579    0.17727  -4.997 5.83e-07 ***
## V46           -1.08493    0.25513  -4.252 2.11e-05 ***
## V47           -0.64235    0.32519  -1.975 0.048234 *
## V48           -0.50262    0.38329  -1.311 0.189745
## V49           -0.20714    0.10111  -2.049 0.040502 *
## V50            0.04754    0.06007   0.791 0.428765
## V51           -0.06586    0.12898  -0.511 0.609646
## V52            0.24800    0.05813   4.266 1.99e-05 ***
## V53            1.01664    0.16220   6.268 3.66e-10 ***
## V54            0.59058    0.33572   1.759 0.078551 .
## V55           -0.56200    0.22976  -2.446 0.014445 *
## V56            1.08271    0.29373   3.686 0.000228 ***
## V57            0.61655    0.14118   4.367 1.26e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4121.0  on 3066  degrees of freedom
## Residual deviance: 1157.4  on 3009  degrees of freedom
## AIC: 1273.4
##
## Number of Fisher Scoring iterations: 13
```

```
Probs_train_scale <- predict(glm_fits_scale, type='response')
Pred_trend_train_scale <- ifelse(Probs_train_scale > 0.5, 1, 0)

glm_train_error_scale <- mean(Pred_trend_train_scale != train_scale_all$V58)
```

Column V5, V7, V16, V17, V20, V21, V23, V25, V27, V45, V46, V52, V53, V56, V57 appear to be statistically significant.

Error of logistic regression discriminant on standardized training data: 0.0717313

Standardize Testing Dataset:

```
Probs_test_scale <- predict(glm_fits_scale, test_scale_all, type='response')
Pred_trend_test_scale <- ifelse(Probs_test_scale > 0.5, 1, 0)
```

```r
glm_test_error_scale <- mean(Pred_trend_test_scale != test_scale_all$V58)
```

Error of logistic regression discriminant on standardized testing data: 0.0710561

Log-transform Training Dataset:

```r
glm_fits_log <- glm(V58 ~ ., data = train_log_all, family="binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(glm_fits_log)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = "binomial", data = train_log_all)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0831  -0.1646  -0.0010   0.0738   3.7853
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.55361    0.47536 -11.683  < 2e-16 ***
## V1          -0.50525    0.52078  -0.970 0.331955
## V2          -0.48375    0.41287  -1.172 0.241325
## V3          -0.34268    0.32461  -1.056 0.291122
## V4           2.49036    2.49963   0.996 0.319109
## V5           1.68052    0.26735   6.286 3.26e-10 ***
## V6           0.49007    0.49976   0.981 0.326779
## V7           3.81919    0.63656   6.000 1.98e-09 ***
## V8           1.11891    0.39254   2.850 0.004366 **
## V9           0.22162    0.61448   0.361 0.718349
## V10          0.20794    0.26664   0.780 0.435466
## V11         -1.73051    0.64790  -2.671 0.007563 **
## V12         -0.13019    0.21705  -0.600 0.548628
## V13         -1.47819    0.59699  -2.476 0.013284 *
## V14          0.49815    0.49244   1.012 0.311724
## V15          2.35454    1.31509   1.790 0.073389 .
## V16          2.00188    0.30550   6.553 5.64e-11 ***
## V17          2.00033    0.49917   4.007 6.14e-05 ***
## V18         -0.62599    0.34041  -1.839 0.065927 .
## V19          0.04966    0.17069   0.291 0.771075
## V20          4.74708    1.75988   2.697 0.006989 **
## V21          0.92793    0.20837   4.453 8.46e-06 ***
## V22          0.19783    0.59582   0.332 0.739860
## V23          3.39784    0.89163   3.811 0.000139 ***
## V24          1.27695    0.41124   3.105 0.001902 **
## V25         -3.97126    0.60152  -6.602 4.06e-11 ***
## V26         -0.43395    0.74531  -0.582 0.560401
## V27         -5.92242    1.42772  -4.148 3.35e-05 ***
## V28          1.27690    0.58913   2.167 0.030202 *
## V29         -5.52545    3.47037  -1.592 0.111344
## V30         -0.08833    0.47636  -0.185 0.852892
## V31         -1.17924    2.44793  -0.482 0.629997
```

```
## V32          -4.26131    4.43665  -0.960 0.336814
## V33          -1.44590    0.73243  -1.974 0.048368 *
## V34           0.86735    4.05419   0.214 0.830595
## V35          -2.60252    1.20495  -2.160 0.030784 *
## V36           0.44061    0.70994   0.621 0.534840
## V37          -1.55260    0.59961  -2.589 0.009615 **
## V38          -1.10219    1.36375  -0.808 0.418971
## V39           0.09940    0.80741   0.123 0.902025
## V40          -1.66152    1.14748  -1.448 0.147622
## V41         -45.30209   35.39198  -1.280 0.200542
## V42          -4.12654    1.24565  -3.313 0.000924 ***
## V43          -5.08561    1.94170  -2.619 0.008815 **
## V44          -2.90440    1.49695  -1.940 0.052354 .
## V45          -2.02986    0.41499  -4.891 1.00e-06 ***
## V46          -2.21581    0.52201  -4.245 2.19e-05 ***
## V47          -7.41904    4.88356  -1.519 0.128715
## V48          -2.02099    1.39842  -1.445 0.148405
## V49          -1.58851    0.79263  -2.004 0.045059 *
## V50          -0.01172    0.62116  -0.019 0.984945
## V51          -3.40426    2.64864  -1.285 0.198693
## V52           2.24783    0.29972   7.500 6.39e-14 ***
## V53           4.93003    0.88667   5.560 2.70e-08 ***
## V54          -0.01276    2.13277  -0.006 0.995225
## V55           0.57047    0.33492   1.703 0.088513 .
## V56           0.09317    0.19497   0.478 0.632744
## V57           0.75138    0.13167   5.707 1.15e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4121.01  on 3066  degrees of freedom
## Residual deviance:  930.67  on 3009  degrees of freedom
## AIC: 1046.7
##
## Number of Fisher Scoring iterations: 12
```

```r
Probs_train_log <- predict(glm_fits_log, type='response')
Pred_trend_train_log <- ifelse(Probs_train_log > 0.5, 1, 0)

glm_train_error_log <- mean(Pred_trend_train_log != train_log_all$V58)
```

Column V5, V7, V16, V17, V21, V23, V25, V27, V42, V45, V46, V52, V53, V57 appear to be statistically significant.

Error of logistic regression discriminant on log-transform training data: 0.0577111

Log-transform Testing Dataset:

```r
Probs_test_log <- predict(glm_fits_log, test_log_all, type='response')
Pred_trend_test_log <- ifelse(Probs_test_log > 0.5, 1, 0)

glm_test_error_log <- mean(Pred_trend_test_log != test_log_all$V58)
```

Error of logistic regression discriminant on log-transform testing data: 0.0567145

Discretize Training Dataset:

```r
glm_fits_i <- glm(V58 ~ ., data = train_i_all, family="binomial")
summary(glm_fits_i)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = "binomial", data = train_i_all)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6393  -0.1904  -0.0130   0.0600   3.9295
##
## Coefficients: (3 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.102414   0.189853 -11.074  < 2e-16 ***
## V1TRUE      -0.303292   0.289818  -1.046 0.295335
## V2TRUE      -0.378470   0.275804  -1.372 0.169989
## V3TRUE      -0.199095   0.212662  -0.936 0.349167
## V4TRUE       1.096282   0.824259   1.330 0.183511
## V5TRUE       1.268090   0.216147   5.867 4.44e-09 ***
## V6TRUE       0.251840   0.273000   0.922 0.356271
## V7TRUE       2.986605   0.386285   7.732 1.06e-14 ***
## V8TRUE       0.875957   0.316310   2.769 0.005618 **
## V9TRUE       0.228813   0.325213   0.704 0.481695
## V10TRUE      0.742343   0.238269   3.116 0.001836 **
## V11TRUE     -1.162239   0.334525  -3.474 0.000512 ***
## V12TRUE     -0.078381   0.194282  -0.403 0.686624
## V13TRUE     -1.161887   0.311432  -3.731 0.000191 ***
## V14TRUE      0.941421   0.452030   2.083 0.037283 *
## V15TRUE      2.006003   0.693342   2.893 0.003813 **
## V16TRUE      1.984579   0.226463   8.763  < 2e-16 ***
## V17TRUE      1.096497   0.319793   3.429 0.000606 ***
## V18TRUE     -0.857063   0.264975  -3.235 0.001219 **
## V19TRUE      0.006163   0.224878   0.027 0.978137
## V20TRUE      1.670892   0.554536   3.013 0.002586 **
## V21TRUE      0.834548   0.210275   3.969 7.22e-05 ***
## V22TRUE      0.811703   0.555363   1.462 0.143859
## V23TRUE      1.787937   0.392435   4.556 5.21e-06 ***
## V24TRUE      1.385796   0.343260   4.037 5.41e-05 ***
## V25TRUE     -3.611845   0.473164  -7.633 2.29e-14 ***
## V26TRUE     -0.640878   0.497465  -1.288 0.197646
## V27TRUE     -4.432733   0.740612  -5.985 2.16e-09 ***
## V28TRUE      1.981086   0.457457   4.331 1.49e-05 ***
## V29TRUE     -1.174992   0.668922  -1.757 0.078996 .
## V30TRUE     -0.183166   0.519469  -0.353 0.724387
## V31TRUE     -1.558298   1.033703  -1.507 0.131685
## V32TRUE     -2.211046   1.150863  -1.921 0.054706 .
## V33TRUE     -0.926369   0.562091  -1.648 0.099337 .
## V34TRUE      0.536636   1.068210   0.502 0.615408
## V35TRUE     -0.973451   0.565672  -1.721 0.085273 .
## V36TRUE      0.636619   0.417226   1.526 0.127050
## V37TRUE     -1.440826   0.348518  -4.134 3.56e-05 ***
## V38TRUE      1.173486   0.741369   1.583 0.113453
```

```
## V39TRUE      0.037749   0.413235   0.091 0.927214    
## V40TRUE     -0.611572   0.557756  -1.096 0.272866    
## V41TRUE     -5.823151   3.179731  -1.831 0.067051 .  
## V42TRUE     -2.410825   0.508741  -4.739 2.15e-06 ***
## V43TRUE     -1.500599   0.638114  -2.352 0.018692 *  
## V44TRUE     -1.301660   0.521227  -2.497 0.012514 *  
## V45TRUE     -1.391117   0.235936  -5.896 3.72e-09 ***
## V46TRUE     -1.789877   0.363562  -4.923 8.52e-07 ***
## V47TRUE     -0.695873   1.130612  -0.615 0.538235    
## V48TRUE     -1.512213   0.617515  -2.449 0.014331 *  
## V49TRUE     -0.070815   0.275485  -0.257 0.797135    
## V50TRUE      0.185424   0.196176   0.945 0.344561    
## V51TRUE     -0.056805   0.409948  -0.139 0.889793    
## V52TRUE      1.476322   0.186253   7.926 2.26e-15 ***
## V53TRUE      1.858618   0.250030   7.434 1.06e-13 ***
## V54TRUE     -0.794196   0.338409  -2.347 0.018933 *  
## V55TRUE            NA         NA      NA       NA    
## V56TRUE            NA         NA      NA       NA    
## V57TRUE            NA         NA      NA       NA    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 4121.0  on 3066  degrees of freedom
## Residual deviance: 1014.6  on 3012  degrees of freedom
## AIC: 1124.6
## 
## Number of Fisher Scoring iterations: 9
```

```
Probs_train_i <- predict(glm_fits_i, type='response')
Pred_trend_train_i <- ifelse(Probs_train_i > 0.5, 1, 0)

glm_train_error_i <- mean(Pred_trend_train_i != train_i_all$V58)
```

Column V5, V7, V11, V13, V16, V17, V21, V23, V24, V25, V27, V28, V37, V42, V45, V46, V52, V53 appear to be statistically significant.

Error of logistic regression discriminant on discretize training data: 0.057059

Discretize Testing Dataset:

```
Probs_test_i <- predict(glm_fits_i, test_i_all, type='response')
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
Pred_trend_test_i <- ifelse(Probs_test_i > 0.5, 1, 0)

glm_test_error_i <- mean(Pred_trend_test_i != test_log_all$V58)
```

Error of logistic regression discriminant on discretize testing data: 0.0808344

(c)

Apply both linear and quadratic discriminant analysis methods to the standardized data, and the log-transformed data. What are the classification errors (training and test)?

Answer:

Linear and quadratic discriminant analysis methods to the standardized data:

```
lda.fit.scale <- lda(V58~., data=train_scale_all)
lda.pred.train.scale <- predict(lda.fit.scale, train_scale_all)
lda_train_error_scale <- mean(lda.pred.train.scale$class != train_scale_all$V58)
lda.pred.test.scale = predict(lda.fit.scale, test_scale_all)
lda_test_error_scale <- mean(lda.pred.test.scale$class != test_scale_all$V58)

qda.fit.scale <- qda(V58~., data=train_scale_all)
qda.pred.train.scale <- predict(qda.fit.scale, train_scale_all)
qda_train_error_scale <- mean(qda.pred.train.scale$class != train_scale_all$V58)
qda.pred.test.scale = predict(qda.fit.scale, test_scale_all)
qda_test_error_scale <- mean(qda.pred.test.scale$class != test_scale_all$V58)
```

Error of linear discriminant on standardized training data: 0.1017281

Error of linear discriminant on standardized testing data: 0.1029987

Error of quadratic discriminant on standardized training data: 0.1786762

Error of quadratic discriminant on standardized testing data: 0.1747066

Linear and quadratic discriminant analysis methods to the log-transformed data:

```
lda.fit.log <- lda(V58~., data=train_log_all)
lda.pred.train.log <- predict(lda.fit.log, train_log_all)
lda_train_error_log <- mean(lda.pred.train.log$class != train_log_all$V58)
lda.pred.test.log = predict(lda.fit.log, test_log_all)
lda_test_error_log <- mean(lda.pred.test.log$class != test_log_all$V58)

qda.fit.log <- qda(V58~., data=train_log_all)
qda.pred.train.log <- predict(qda.fit.log, train_log_all)
qda_train_error_log <- mean(qda.pred.train.log$class != train_log_all$V58)
qda.pred.test.log = predict(qda.fit.log, test_log_all)
qda_test_error_log <- mean(qda.pred.test.log$class != test_log_all$V58)
```

Error of linear discriminant on log-transformed training data: 0.0603195

Error of linear discriminant on log-transformed testing data: 0.065189

Error of quadratic discriminant on log-transformed training data: 0.1587871

Error of quadratic discriminant on log-transformed testing data: 0.1571056

(d)

Apply linear and nonlinear support vector machine classifiers to each version of the data. What are the classification errors (training and test)?

Answer:

Linear support vector machine classifiers to standardized data:

```
train_scale_all$V58 <- as.factor(train_scale_all$V58)

svmfit.linear.scale <- svm(V58~., data=train_scale_all , kernel="linear",
                    cost=0.1, scale=FALSE)

ypred.linear.train.scale <- predict(svmfit.linear.scale, train_scale_all)
svm_linear_train_error_scale <- mean(ypred.linear.train.scale != train_scale_all$V58)

ypred.linear.test.scale <- predict(svmfit.linear.scale, test_scale_all)
svm_linear_test_error_scale <- mean(ypred.linear.test.scale != test_scale_all$V58)
```

Error of linear support vector machine classifiers to standardized training data: 0.0727095

Error of linear support vector machine classifiers to standardized testing data: 0.0677966

Nonlinear support vector machine classifiers to standardized data:

```
svmfit.poly.scale = svm(V58~., data=train_scale_all , kernel="polynomial",
                        degree=1, cost=10)

ypred.nonlinear.train.scale <- predict(svmfit.poly.scale, train_scale_all)
svm_nonlinear_train_error_scale <- mean(ypred.nonlinear.train.scale != train_scale_all$V58)

ypred.nonlinear.test.scale <- predict(svmfit.poly.scale, test_scale_all)
svm_nonlinear_test_error_scale <- mean(ypred.nonlinear.test.scale != test_scale_all$V58)
```

Error of nonlinear support vector machine classifiers to standardized training data: 0.0717313

Error of nonlinear support vector machine classifiers to standardized testing data: 0.0684485

Linear support vector machine classifiers to log-transformed data:

```
train_log_all$V58 <- as.factor(train_log_all$V58)

svmfit.linear.log <- svm(V58~., data=train_log_all , kernel="linear",
                    cost=0.1, scale=FALSE)

ypred.linear.train.log <- predict(svmfit.linear.log, train_log_all)
svm_linear_train_error_log <- mean(ypred.linear.train.log != train_log_all$V58)

ypred.linear.test.log <- predict(svmfit.linear.log, test_log_all)
svm_linear_test_error_log <- mean(ypred.linear.test.log != test_log_all$V58)
```

Error of linear support vector machine classifiers to log-transformed training data: 0.057059

Error of linear support vector machine classifiers to log-transformed testing data: 0.0580183

Nonlinear support vector machine classifiers to log-transformed data:

```
svmfit.poly.log = svm(V58~., data=train_log_all , kernel="polynomial",
                        degree=1, cost=10)

ypred.nonlinear.train.log <- predict(svmfit.poly.log, train_log_all)
svm_nonlinear_train_error_log <- mean(ypred.nonlinear.train.log != train_log_all$V58)

ypred.nonlinear.test.log <- predict(svmfit.poly.log, test_log_all)
svm_nonlinear_test_error_log <- mean(ypred.nonlinear.test.log != test_log_all$V58)
```

Error of nonlinear support vector machine classifiers to log-transformed training data: 0.0557548

Error of nonlinear support vector machine classifiers to log-transformed testing data: 0.0521512

Linear support vector machine classifiers to discretize data:

```
train_i_all$V58 <- as.factor(train_i_all$V58)

svmfit.linear.i <- svm(V58~., data=train_i_all , kernel="linear",
                    cost=0.1, scale=FALSE)

ypred.linear.train.i <- predict(svmfit.linear.i, train_i_all)
svm_linear_train_error_i <- mean(ypred.linear.train.i != train_i_all$V58)

ypred.linear.test.i <- predict(svmfit.linear.i, test_i_all)
svm_linear_test_error_i <- mean(ypred.linear.test.i != test_i_all$V58)
```

Error of linear support vector machine classifiers to discretize training data: 0.0639061

Error of linear support vector machine classifiers to discretize testing data: 0.0743155

Nonlinear support vector machine classifiers to discretize data:

```
svmfit.poly.i = svm(V58~., data=train_i_all , kernel="polynomial",
                        degree=1, cost=10)

ypred.nonlinear.train.i <- predict(svmfit.poly.i, train_i_all)
svm_nonlinear_train_error_i <- mean(ypred.nonlinear.train.i != train_i_all$V58)

ypred.nonlinear.test.i <- predict(svmfit.poly.i, test_i_all)
svm_nonlinear_test_error_i <- mean(ypred.nonlinear.test.i != test_i_all$V58)
```

Error of nonlinear support vector machine classifiers to discretize training data: 0.0596674

Error of nonlinear support vector machine classifiers to discretize testing data: 0.0710561

(e)

Apply tree-based classifiers to this data. What are the classification errors (training and test)?

Answer:

Single Regression Tree and Random Forest classifiers to standardized data:

```
tree.train.scale <- tree(V58~., data=train_scale_all)

yhat.single.test.scale <- predict(tree.train.scale, newdata = test_scale_all)

pred_tree_test_scale <- ifelse(yhat.single.test.scale[,1] > 0.5, 0, 1)
single_tree_test_error_scale <- mean(pred_tree_test_scale != test_scale_all$V58)

yhat.single.train.scale <- predict(tree.train.scale, newdata = train_scale_all)

pred_tree_train_scale <- ifelse(yhat.single.train.scale[,1] > 0.5, 0, 1)
single_tree_train_error_scale <- mean(pred_tree_train_scale != train_scale_all$V58)

RF.train.scale <- randomForest(V58~., data=train_scale_all,
                         mtry=4, importance =TRUE, ntree=100)

yhat.RF.test.scale <- predict(RF.train.scale, newdata = test_scale_all)

RF_test_error_scale <- mean(yhat.RF.test.scale != test_scale_all$V58)

yhat.RF.train.scale <- predict(RF.train.scale, newdata = train_scale_all)

RF_train_error_scale <- mean(yhat.RF.train.scale != train_scale_all$V58)
```

Error of Single Regression Tree classifier to standardized training data: 0.0877079

Error of Single Regression Tree classifier to standardized testing data: 0.1277705

Error of Random Forest classifiers to standardized training data: 0.0078252

Error of Random Forest classifier to standardized testing data: 0.0319426

Single Regression Tree and Random Forest classifiers to log-transform data:

```
tree.train.log <- tree(V58~., data=train_log_all)

yhat.single.test.log <- predict(tree.train.log, newdata = test_log_all)

pred_tree_test_log <- ifelse(yhat.single.test.log[,1] > 0.5, 0, 1)
single_tree_test_error_log <- mean(pred_tree_test_log != test_log_all$V58)

yhat.single.train.log <- predict(tree.train.log, newdata = train_log_all)

pred_tree_train_log <- ifelse(yhat.single.train.log[,1] > 0.5, 0, 1)
single_tree_train_error_log <- mean(pred_tree_train_log != train_log_all$V58)

RF.train.log <- randomForest(V58~., data=train_log_all,
                         mtry=4, importance =TRUE, ntree=100)
```

```r
yhat.RF.test.log <- predict(RF.train.log, newdata = test_log_all)

RF_test_error_log <- mean(yhat.RF.test.log != test_log_all$V58)

yhat.RF.train.log <- predict(RF.train.log, newdata = train_log_all)

RF_train_error_log <- mean(yhat.RF.train.log != train_log_all$V58)
```

Error of Single Regression Tree classifier to log-transform training data: 0.0877079

Error of Single Regression Tree classifier to log-transform testing data: 0.0912647

Error of Random Forest classifiers to log-transform training data: 0.006195

Error of Random Forest classifier to log-transform testing data: 0.0267275

Single Regression Tree and Random Forest classifiers to discretize data:

```r
tree.train.i <- tree(V58~., data=train_i_all)

yhat.single.test.i <- predict(tree.train.i, newdata = test_i_all)

pred_tree_test_i <- ifelse(yhat.single.test.i[,1] > 0.5, 0, 1)
single_tree_test_error_i <- mean(pred_tree_test_i != test_i_all$V58)

yhat.single.train.i <- predict(tree.train.i, newdata = train_i_all)

pred_tree_train_i <- ifelse(yhat.single.train.i[,1] > 0.5, 0, 1)
single_tree_train_error_i <- mean(pred_tree_train_i != train_i_all$V58)

RF.train.i <- randomForest(V58~., data=train_i_all,
                         mtry=4, importance =TRUE, ntree=100)

yhat.RF.test.i <- predict(RF.train.i, newdata = test_i_all)

RF_test_error_i <- mean(yhat.RF.test.i != test_i_all$V58)

yhat.RF.train.i <- predict(RF.train.i, newdata = train_i_all)

RF_train_error_i <- mean(yhat.RF.train.i != train_i_all$V58)
```

Error of Single Regression Tree classifier to discretize training data: 0.1170525

Error of Single Regression Tree classifier to discretize testing data: 0.1170525

Error of Random Forest classifiers to discretize training data: 0.031627

Error of Random Forest classifier to discretize testing data: 0.0547588

Report classification errors using different methods and different preprocessed data in a table, and comment on the different performances.

```
df <- data.frame(Logistic_Regression = c(glm_train_error_scale, glm_test_error_scale,
                                         glm_train_error_log, glm_test_error_log,
                                         glm_train_error_i, glm_test_error_i),
                 Linear = c(lda_train_error_scale, lda_test_error_scale,
                            lda_train_error_log, lda_test_error_log,
                            NaN, NaN),
                 Quadratic = c(qda_train_error_scale, qda_test_error_scale,
                               qda_train_error_log, qda_test_error_log,
                               NaN, NaN),
                 Linear_SVM = c(svm_linear_train_error_scale, svm_linear_test_error_scale,
                                svm_linear_train_error_log, svm_linear_test_error_log,
                                svm_linear_train_error_i, svm_linear_test_error_i),
                 Nonlinear_SVM = c(svm_nonlinear_train_error_scale, svm_nonlinear_test_error_scale,
                                   svm_nonlinear_train_error_log, svm_nonlinear_test_error_log,
                                   svm_nonlinear_train_error_i, svm_nonlinear_test_error_i),
                 Single_Tree = c(single_tree_train_error_scale, single_tree_test_error_scale,
                                 single_tree_train_error_log, single_tree_test_error_log,
                                 single_tree_train_error_i, single_tree_test_error_i),
                 Random_Forest = c(RF_train_error_scale, RF_test_error_scale,
                                   RF_train_error_log, RF_test_error_log,
                                   RF_train_error_i, RF_test_error_i))

rownames(df) <- c('Standardized_Train', 'Standardized_Test',
                  'Log_Transform_Train', 'Log_Transform_Test',
                  'Discretize_Train', 'Discretize_Test')
dt <- as.data.table(df, TRUE)
dt
```

```
##                     rn Logistic_Regression     Linear Quadratic Linear_SVM
## 1:  Standardized_Train          0.07173133 0.10172807 0.1786762 0.07270949
## 2:   Standardized_Test          0.07105606 0.10299870 0.1747066 0.06779661
## 3: Log_Transform_Train          0.05771112 0.06031953 0.1587871 0.05705902
## 4:  Log_Transform_Test          0.05671447 0.06518905 0.1571056 0.05801825
## 5:    Discretize_Train          0.05705902        NaN       NaN 0.06390610
## 6:     Discretize_Test          0.08083442        NaN       NaN 0.07431551
##    Nonlinear_SVM Single_Tree Random_Forest
## 1:    0.07173133  0.08770786   0.007825236
## 2:    0.06844850  0.12777053   0.031942634
## 3:    0.05575481  0.08770786   0.006194979
## 4:    0.05215124  0.09126467   0.026727510
## 5:    0.05966743  0.11705249   0.031626997
## 6:    0.07105606  0.12255541   0.054758801
```

From the above table, we can see that log-transform produced the lowest errors compared to standardization and discretization. And Random Forest Classification produced the lowest errors among all other classifications. Note, in the table there are four NaN values, it is because we never perform the linear and quadratic discriminant on the discretizated data.

Finally, use either a single method with properly chosen tuning parameter or a combination of several methods to design a classifier with test error rate as small as possible. Describe your recommended method and its performance.

As the table showed above, we can see that the Random Forest Classification with log-transform produce the lowest classification error among all others, Error rate of 0.006195 on training set and 0.0267275 on testing set. Therefore, we recommend to use Random Forest Classification as our model method and also with log-transform as our data preprocessing.