MATH 189 Final

Zijian Su
Zelong Zhou
Xiangyi Lin

Last Updated: March 23, 2023

Concrete contributions

All problems were done by Zijian Su, Zelong Zhou, Xiangyi Lin. All contributing equally to this assignment. Everyone put in enough effort.

Overview

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:

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

Transform the features using log(xij +1);

Discretize each feature using I(xij $>$ 0).

Packages

```
#install.packages("corrplot")
#install.packages("ggplot2")
#tinytex::install_tinytex()
#install.packages("scatterplot3d")
```

```
data_train <- read.csv('spam-train.txt', header = FALSE)
data_test <- read.csv('spam-test.txt', header = FALSE)
```

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

```
V58 <- data_train[, 58]
data_train_sd <- scale(data_train[,-58])
data_train_sd <- as.data.frame(cbind(data_train_sd, V58))

V58 <- data_test[, 58]
data_test_sd <- scale(data_test[,-58])
data_test_sd <- as.data.frame(cbind(data_test_sd, V58))
```

Transform the features

```
V58 <- data_train[, 58]
data_train_log <- apply(data_train[,-58], 2, function(x) log10(x + 1))
data_train_log <- as.data.frame(cbind(data_train_log, V58))

V58 <- data_test[, 58]
data_test_log <- apply(data_test[,-58], 2, function(x) log10(x + 1))
data_test_log <- as.data.frame(cbind(data_test_log, V58))
```

Discretize each feature using I(xij >0)

```
V58 <- data_train[, 58]
train_disc <- apply(data_train[,-58], 2, function(x) ifelse(x > 0, 1, 0))
train_disc <- as.data.frame(cbind(train_disc, V58))

V58 <- data_test[, 58]
test_disc <- apply(data_test[,-58], 2, function(x) ifelse(x > 0, 1, 0))
test_disc <- as.data.frame(cbind(test_disc, V58))
```

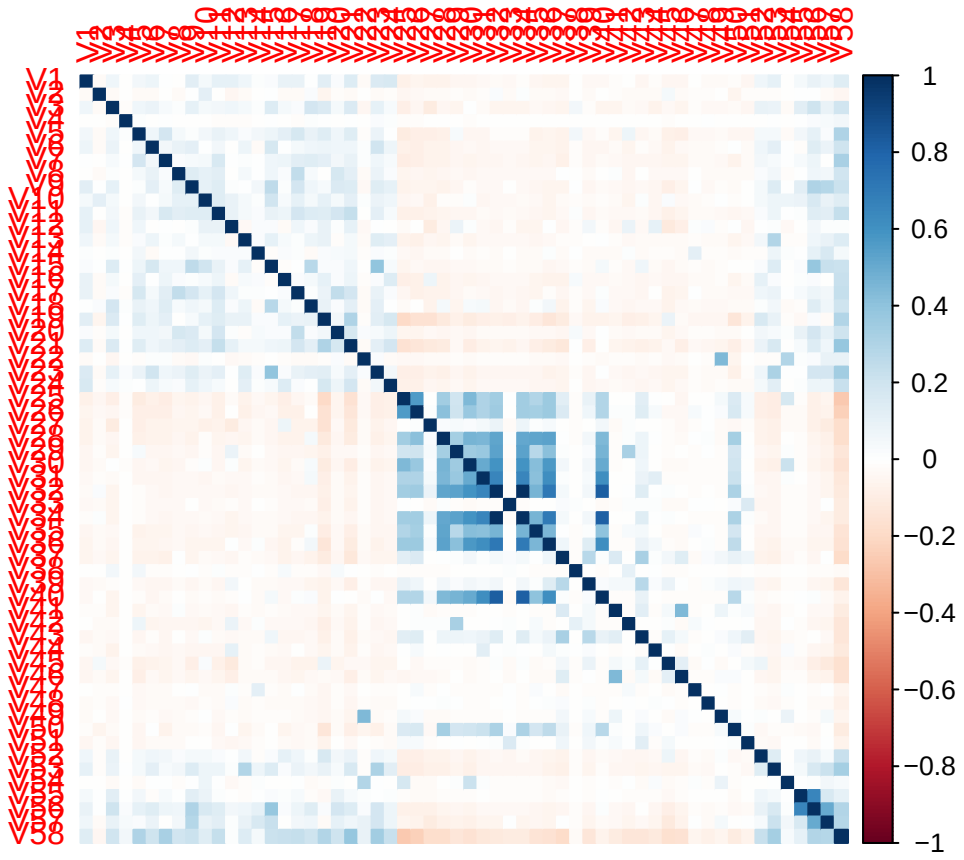
Question a

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

For each data, because there are too many data. We only take the correlation greater than **0.7** to show.

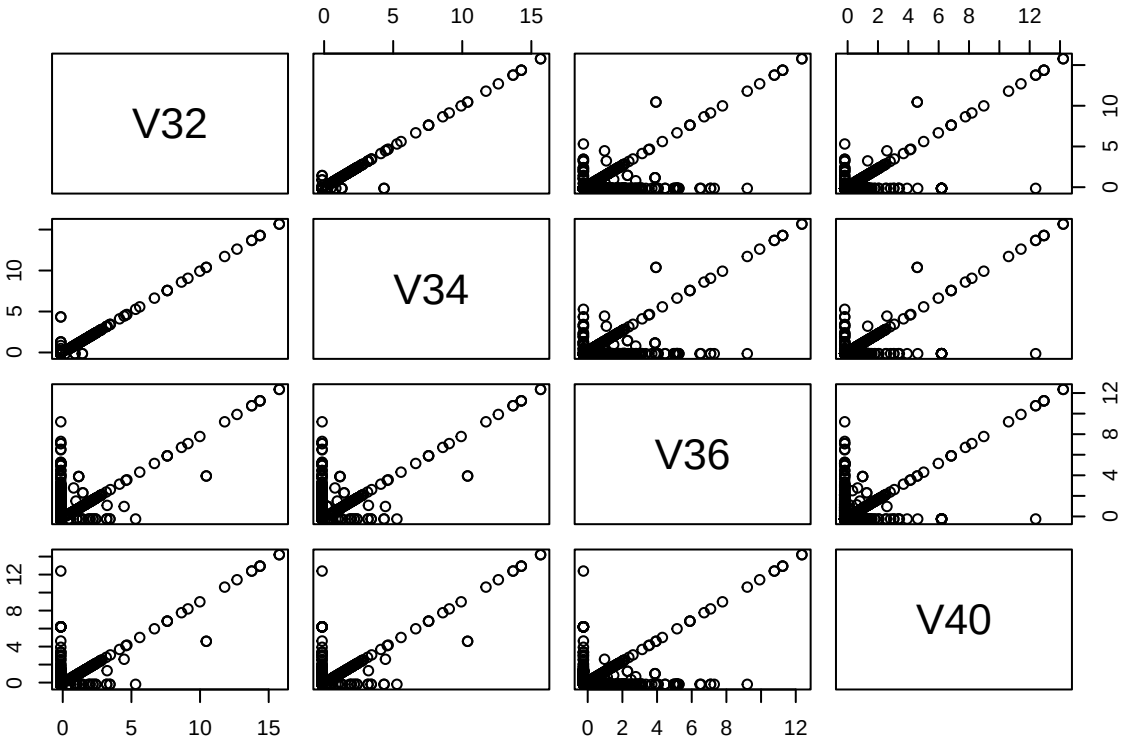
standardized train data:

```
library(corrplot)
cor_mat <- cor(data_train_sd)
corrplot(cor_mat, method = 'color')
```



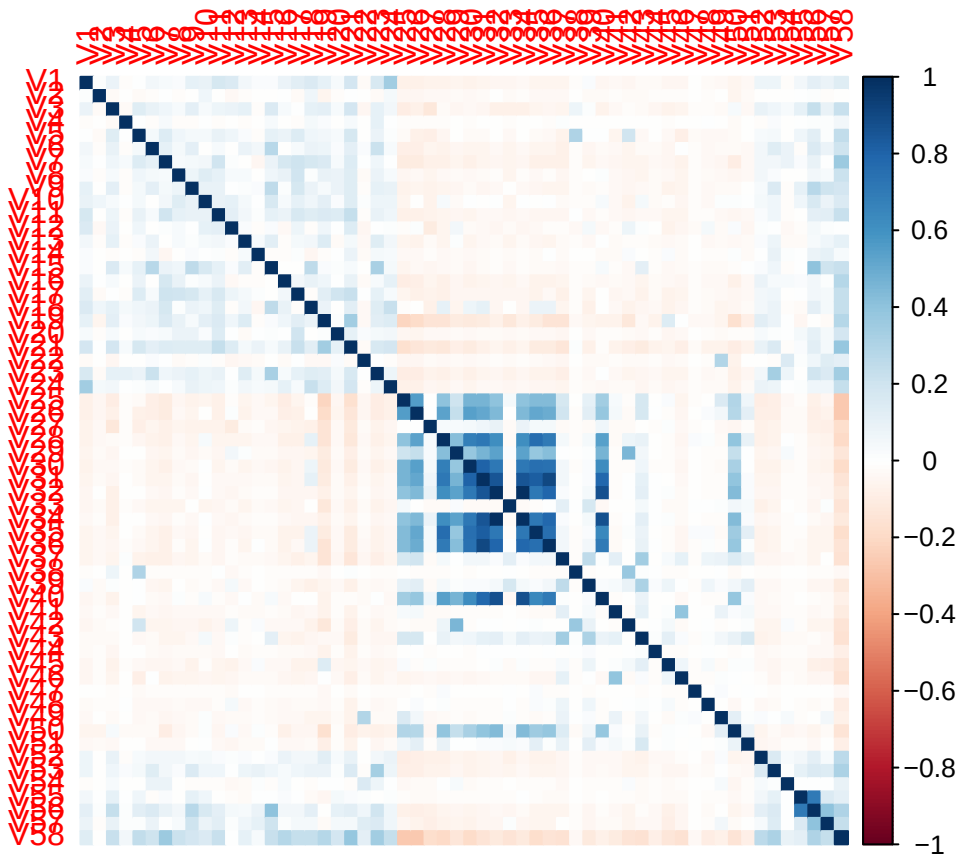
standardized train data features with correlation greater than **0.7**:

```
high_cor <- which(abs(cor_mat) > 0.7 & cor_mat != 1, arr.ind = TRUE)
vector <- unlist(lapply(high_cor, function(x) as.vector(x)))
high_cor_pair <- sort(unique(vector))
pairs(data_train_sd[,high_cor_pair], cex=1)
```



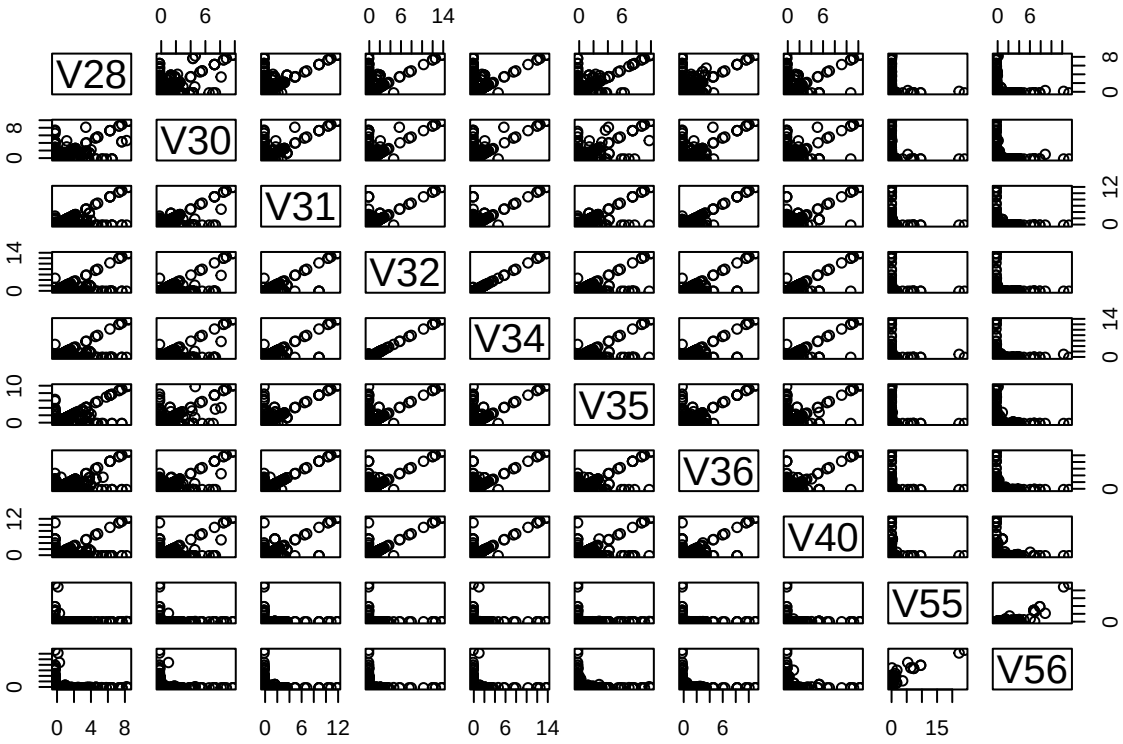
standardized test data:

```
library(corrplot)
cor_mat <- cor(data_test_sd)
corrplot(cor_mat, method = 'color')
```



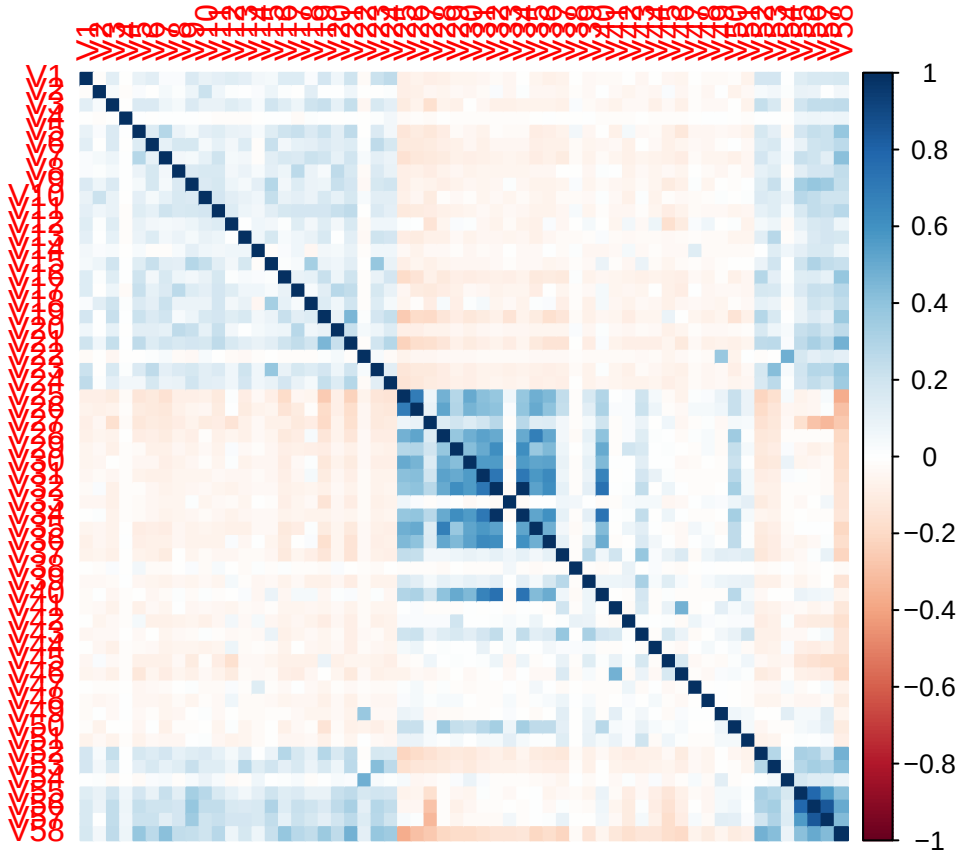
standardized test data features with correlation greater than **0.7**:

```
high_cor <- which(abs(cor_mat) > 0.7 & cor_mat != 1, arr.ind = TRUE)
vector <- unlist(lapply(high_cor, function(x) as.vector(x)))
high_cor_pair <- sort(unique(vector))
pairs(data_test_sd[,high_cor_pair], cex=1)
```



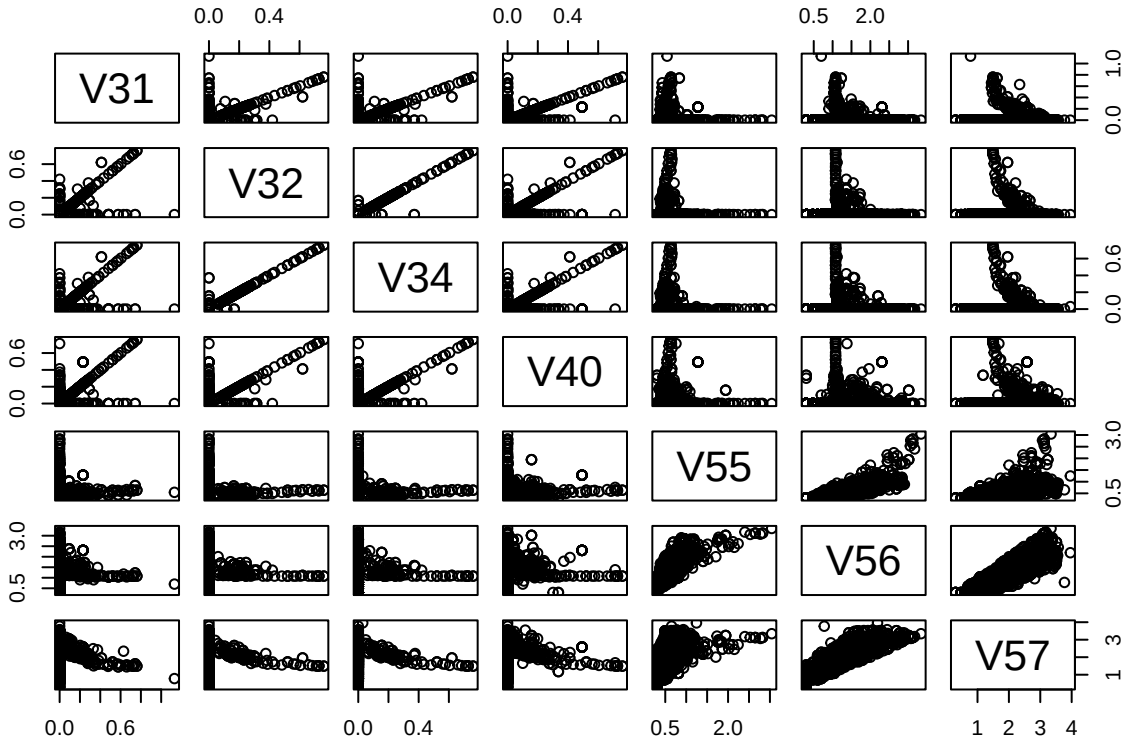
log-transformed train data

```
library(corrplot)
cor_mat <- cor(data_train_log)
corrplot(cor_mat, method = 'color')
```



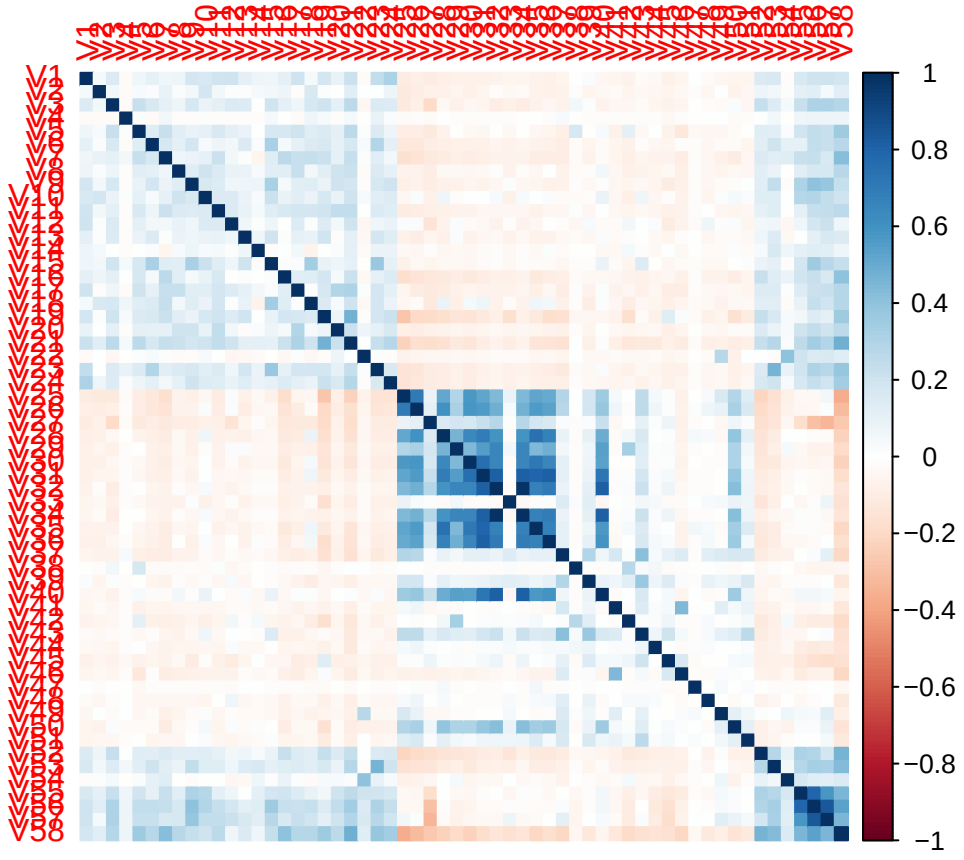
log-transformed train data features with correlation greater than **0.7**

```
high_cor <- which(abs(cor_mat) > 0.7 & cor_mat != 1, arr.ind = TRUE)
vector <- unlist(lapply(high_cor, function(x) as.vector(x)))
high_cor_pair <- sort(unique(vector))
pairs(data_train_log[,high_cor_pair], cex=1)
```



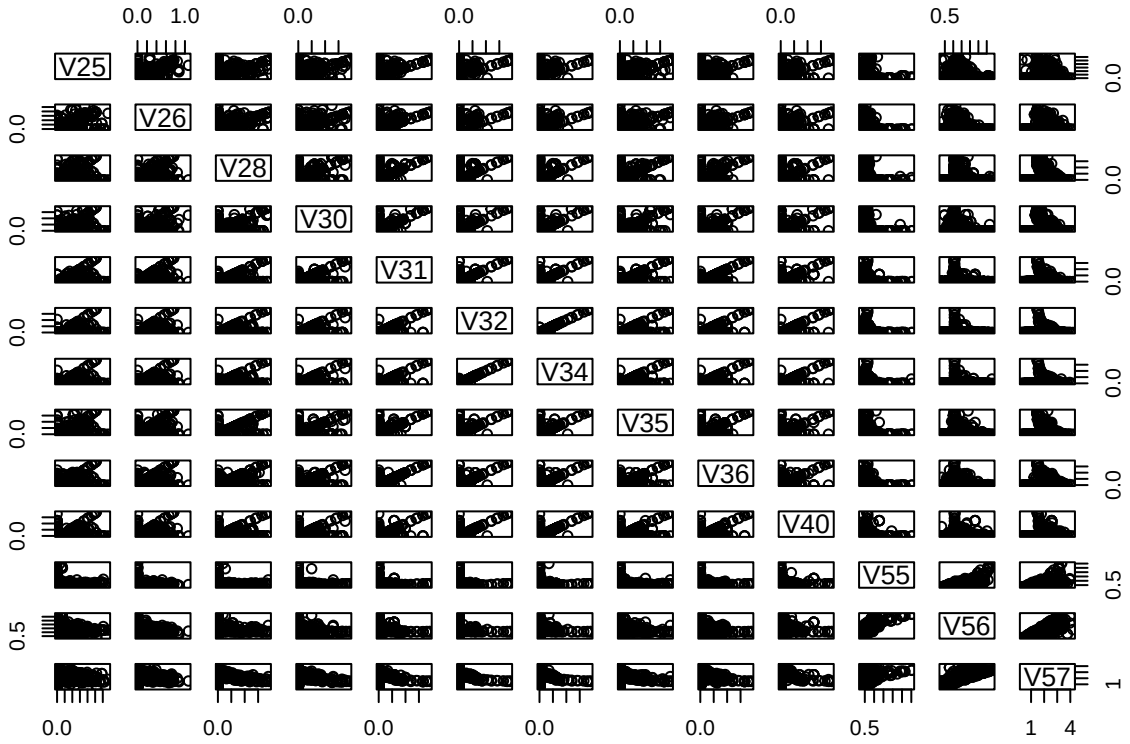
log-transformed test data

```
library(corrplot)
cor_mat <- cor(data_test_log)
corrplot(cor_mat, method = 'color')
```



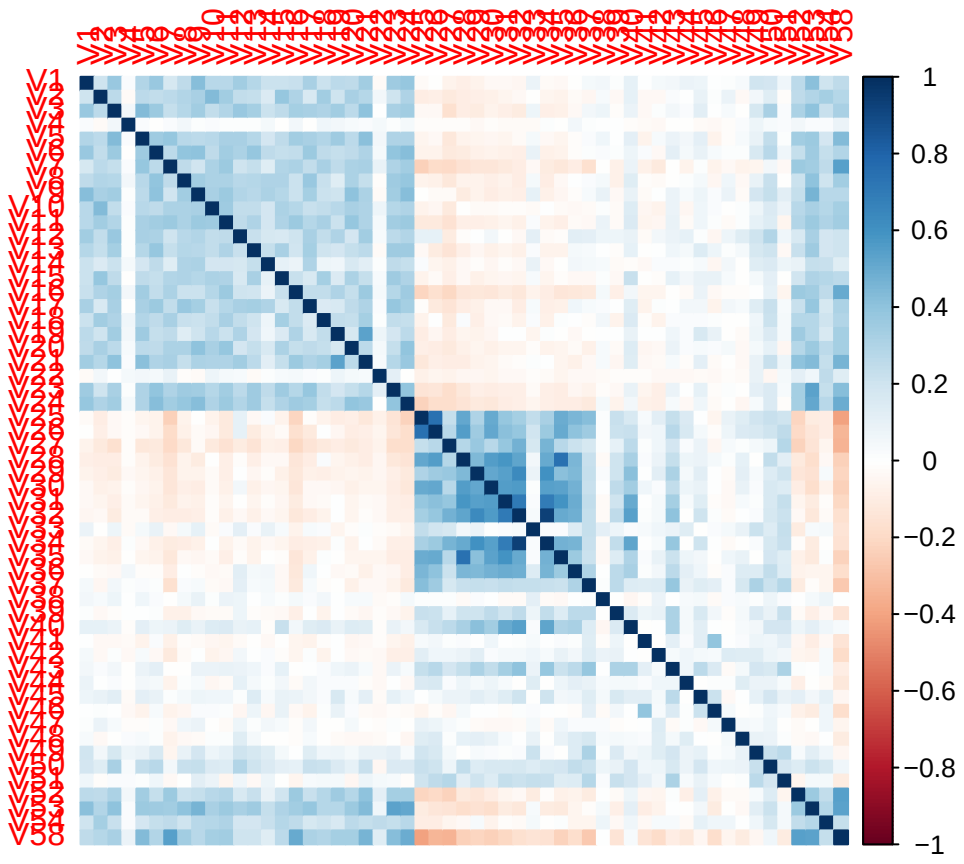
log-transformed test data features with correlation greater than **0.7**

```
high_cor <- which(abs(cor_mat) > 0.7 & cor_mat != 1, arr.ind = TRUE)
vector <- unlist(lapply(high_cor, function(x) as.vector(x)))
high_cor_pair <- sort(unique(vector))
pairs(data_test_log[,high_cor_pair], cex=1)
```



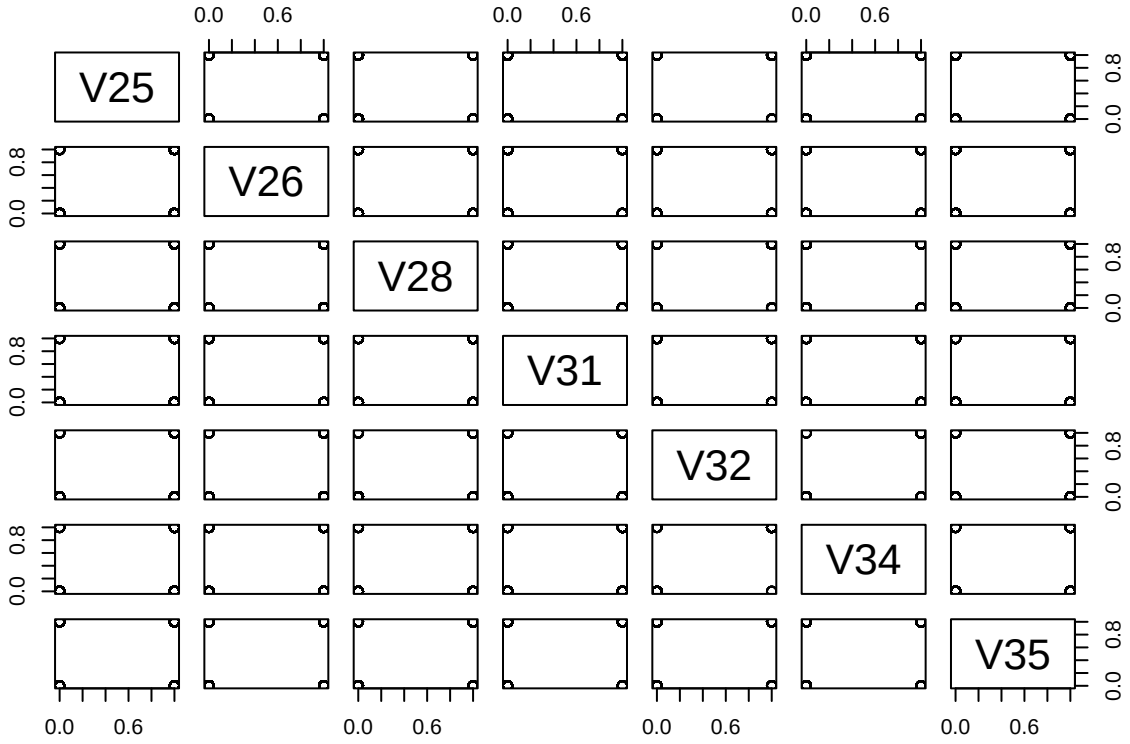
train data after discretize (Since column 55 56 57 always greater than 0, Here we don't consider them.)

```
library(corrplot)
cor_mat <- cor(train_disc[,-c(55,56,57)])
corrplot(cor_mat, method = 'color')
```



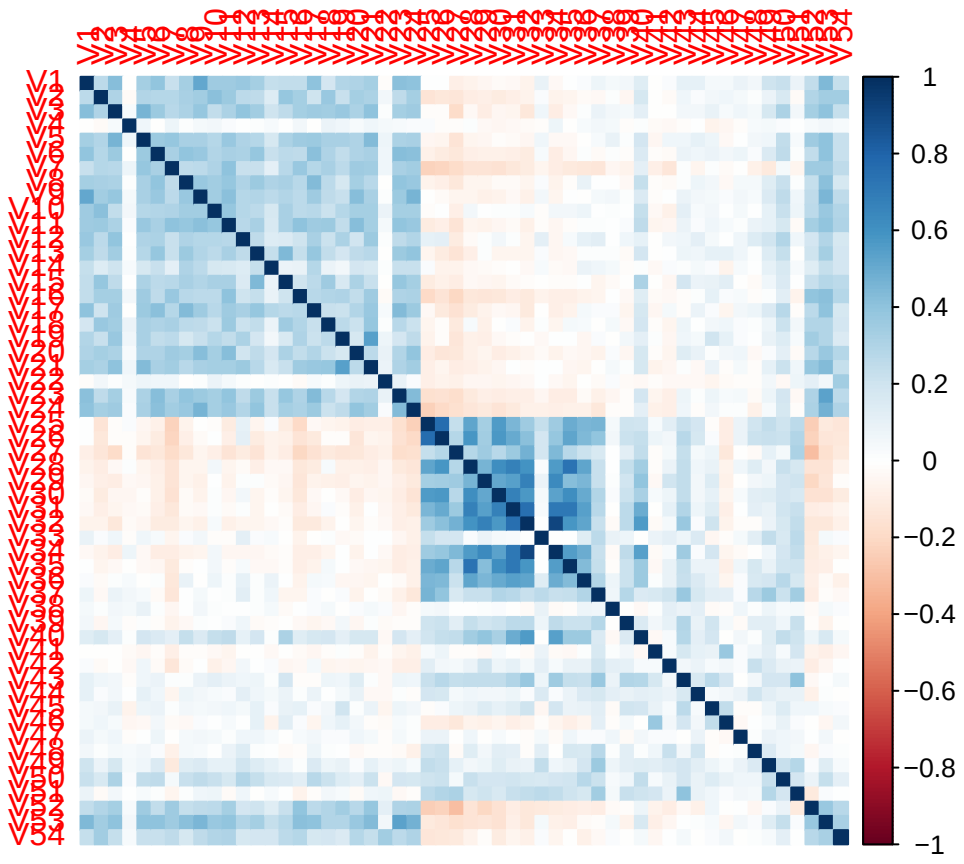
train data after discretize features with correlation greater than **0.7** (Since column 55 56 57 always greater than 0, Here we don't consider them.)

```
high_cor <- which(abs(cor_mat) > 0.7 & cor_mat != 1, arr.ind = TRUE)
vector <- unlist(lapply(high_cor, function(x) as.vector(x)))
high_cor_pair <- sort(unique(vector))
pairs(train_disc[,high_cor_pair], cex=1)
```



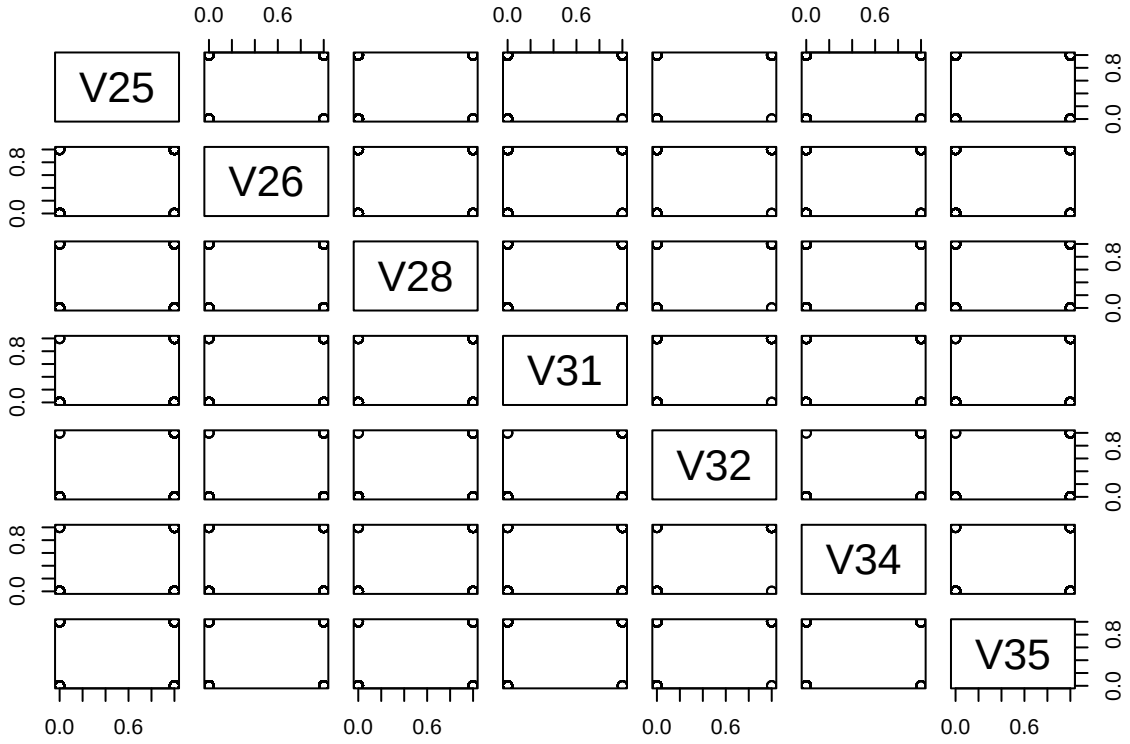
test data after discretize (Since column 55 56 57 always greater than 0, Here we don't consider them.)

```
library(corrplot)
cor_mat <- cor(test_disc[,-c(55,56,57,58)])
corrplot(cor_mat, method = 'color')
```



test data after discretize features with correlation greater than **0.7(**Since column 55 56 57 always greater than 0, Here we don't consider them.**)**

```
high_cor <- which(abs(cor_mat) > 0.7 & cor_mat != 1, arr.ind = TRUE)
vector <- unlist(lapply(high_cor, function(x) as.vector(x)))
high_cor_pair <- sort(unique(vector))
pairs(test_disc[,high_cor_pair], cex=1)
```



Question 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?

Standardized data:

```
sd_fit <- glm(V58 ~ ., data = data_train_sd, family = "binomial")
summary(sd_fit)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = "binomial", data = data_train_sd)
##
## 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_sd = predict(sd_fit,data_train_sd, type='response')
Pred_sd = ifelse(Probs_sd>0.5, 1, 0)
table(Pred_sd, data_train_sd$V58)
```

```
##
## Pred_sd    0    1
##       0 1762  133
##       1   87 1085
```

```
train_errors_sd <- mean(Pred_sd != data_train_sd$V58)
cat("\ntrain errors (Standardize):", train_errors_sd, "\n")
```

```
##
## train errors (Standardize): 0.07173133
```

```
Probs_sd = predict(sd_fit,data_test_sd, type='response')
Pred_sd = ifelse(Probs_sd>0.5, 1, 0)
table(Pred_sd, data_test_sd$V58)
```

```
##
## Pred_sd   0   1
##       0 877  70
##       1  39 548
```

```
test_errors_sd <- mean(Pred_sd != data_test_sd$V58)
cat("\ntese errors (Standardize):", test_errors_sd, "\n")
```

```
##
## tese errors (Standardize): 0.07105606
```

As can be seen from the above results, the prediction error rate in training is 0.07173133 and in test set is 0.07105606 . can be considered accurate.

```
significant_params <- rownames(summary(sd_fit)$coefficients
                               [summary(sd_fit)$coefficients[,4] < 0.05, ])
for (param_name in significant_params) {
  if(param_name != "(Intercept)"){
        cat("\n",param_name)
  }
  else{
    cat("statistically significant V:")
  }
}
```

```
## statistically significant V:
##  V4
##  V5
##  V7
##  V8
##  V9
##  V11
##  V16
##  V17
##  V19
##  V20
##  V21
##  V23
##  V25
##  V27
##  V42
##  V44
##  V45
##  V46
##  V47
##  V49
##  V52
##  V53
```

```
##  V55
##  V56
##  V57
```

The output above is for all parameters with a P value less than 0.05 in **Standardized data**, they reject the null hypothesis. So the above parameters are statistically significant.

Log-transformed data:

```
sd_fit_2 <- glm(V58 ~ ., data = data_train_log, family = "binomial")
summary(sd_fit_2)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = "binomial", data = data_train_log)
##
## 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            -1.16339    1.19915  -0.970 0.331955
## V2            -1.11388    0.95067  -1.172 0.241325
## V3            -0.78904    0.74744  -1.056 0.291122
## V4             5.73427    5.75562   0.996 0.319109
## V5             3.86955    0.61559   6.286 3.26e-10 ***
## V6             1.12843    1.15073   0.981 0.326779
## V7             8.79401    1.46573   6.000 1.98e-09 ***
## V8             2.57639    0.90385   2.850 0.004366 **
## V9             0.51030    1.41490   0.361 0.718349
## V10            0.47881    0.61396   0.780 0.435466
## V11           -3.98465    1.49184  -2.671 0.007563 **
## V12           -0.29977    0.49977  -0.600 0.548628
## V13           -3.40365    1.37462  -2.476 0.013284 *
## V14            1.14704    1.13387   1.012 0.311724
## V15            5.42152    3.02810   1.790 0.073389 .
## V16            4.60949    0.70343   6.553 5.64e-11 ***
## V17            4.60592    1.14937   4.007 6.14e-05 ***
## V18           -1.44140    0.78383  -1.839 0.065927 .
## V19            0.11436    0.39302   0.291 0.771075
## V20           10.93055    4.05228   2.697 0.006989 **
## V21            2.13664    0.47979   4.453 8.46e-06 ***
## V22            0.45553    1.37192   0.332 0.739860
## V23            7.82380    2.05305   3.811 0.000139 ***
## V24            2.94028    0.94692   3.105 0.001902 **
## V25           -9.14416    1.38506  -6.602 4.06e-11 ***
## V26           -0.99921    1.71613  -0.582 0.560401
## V27          -13.63687    3.28745  -4.148 3.35e-05 ***
## V28            2.94016    1.35652   2.167 0.030202 *
## V29          -12.72282    7.99082  -1.592 0.111344
```

```
## V30           -0.20339    1.09686  -0.185 0.852892
## V31           -2.71530    5.63656  -0.482 0.629997
## V32           -9.81203   10.21577  -0.960 0.336814
## V33           -3.32932    1.68648  -1.974 0.048368 *
## V34            1.99714    9.33512   0.214 0.830595
## V35           -5.99252    2.77450  -2.160 0.030784 *
## V36            1.01454    1.63469   0.621 0.534840
## V37           -3.57499    1.38065  -2.589 0.009615 **
## V38           -2.53789    3.14014  -0.808 0.418971
## V39            0.22887    1.85913   0.123 0.902025
## V40           -3.82580    2.64216  -1.448 0.147622
## V41         -104.31192   81.49304  -1.280 0.200542
## V42           -9.50171    2.86821  -3.313 0.000924 ***
## V43          -11.71006    4.47092  -2.619 0.008815 **
## V44           -6.68762    3.44686  -1.940 0.052354 .
## V45           -4.67392    0.95554  -4.891 1.00e-06 ***
## V46           -5.10208    1.20197  -4.245 2.19e-05 ***
## V47          -17.08296   11.24480  -1.519 0.128715
## V48           -4.65349    3.21999  -1.445 0.148405
## V49           -3.65767    1.82509  -2.004 0.045059 *
## V50           -0.02699    1.43028  -0.019 0.984945
## V51           -7.83860    6.09872  -1.285 0.198693
## V52            5.17582    0.69013   7.500 6.39e-14 ***
## V53           11.35181    2.04164   5.560 2.70e-08 ***
## V54           -0.02939    4.91089  -0.006 0.995225
## V55            1.31357    0.77119   1.703 0.088513 .
## V56            0.21453    0.44893   0.478 0.632744
## V57            1.73012    0.30318   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
```

```
Probs_log = predict(sd_fit_2,data_train_log, type='response')
Pred_log = ifelse(Probs_log>0.5, 1, 0)
table(Pred_log, data_train_log$V58)
```

```
##
## Pred_log    0    1
##        0 1766   94
##        1   83 1124
```

```
train_errors_log <- mean(Pred_log != data_train_log$V58)
cat("\ntrain errors (log):", train_errors_log, "\n")
```

```
##
## train errors (log): 0.05771112
```

```
Probs_log = predict(sd_fit_2,data_test_log, type='response')
Pred_log = ifelse(Probs_log>0.5, 1, 0)
table(Pred_log, data_test_log$V58)
```

```
##
## Pred_log   0   1
##        0 879  50
##        1  37 568
```

```
test_errors_log <- mean(Pred_log != data_test_log$V58)
cat("\ntese errors (log):", test_errors_log, "\n")
```

```
##
## tese errors (log): 0.05671447
```

As can be seen from the above results, the prediction error rate in training is 0.05771112 and in test set is 0.05671447 . can be considered accurate.

```
significant_params <- rownames(summary(sd_fit_2)$coefficients
                               [summary(sd_fit_2)$coefficients[,4] < 0.05, ])
for (param_name in significant_params) {
  if(param_name != "(Intercept)"){
        cat("\n",param_name)
  }
  else{
    cat("statistically significant V:")
  }
}
```

```
## statistically significant V:
##  V5
##  V7
##  V8
##  V11
##  V13
##  V16
##  V17
##  V20
##  V21
##  V23
##  V24
##  V25
##  V27
##  V28
##  V33
##  V35
##  V37
##  V42
##  V43
##  V45
##  V46
##  V49
```

```
##  V52
##  V53
##  V57
```

The output above is for all parameters with a P value less than 0.05 in **Log-transformed data**, they reject the null hypothesis. So the above parameters are statistically significant.

Data after discretize:

```
sd_fit_3 <- glm(V58 ~ ., data = train_disc, family = "binomial")
summary(sd_fit_3)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = "binomial", data = train_disc)
##
## 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 ***
## V1          -0.303292   0.289818  -1.046 0.295335
## V2          -0.378470   0.275804  -1.372 0.169989
## V3          -0.199095   0.212662  -0.936 0.349167
## V4           1.096282   0.824259   1.330 0.183511
## V5           1.268090   0.216147   5.867 4.44e-09 ***
## V6           0.251840   0.273000   0.922 0.356271
## V7           2.986605   0.386285   7.732 1.06e-14 ***
## V8           0.875957   0.316310   2.769 0.005618 **
## V9           0.228813   0.325213   0.704 0.481695
## V10          0.742343   0.238269   3.116 0.001836 **
## V11         -1.162239   0.334525  -3.474 0.000512 ***
## V12         -0.078381   0.194282  -0.403 0.686624
## V13         -1.161887   0.311432  -3.731 0.000191 ***
## V14          0.941421   0.452030   2.083 0.037283 *
## V15          2.006003   0.693342   2.893 0.003813 **
## V16          1.984579   0.226463   8.763  < 2e-16 ***
## V17          1.096497   0.319793   3.429 0.000606 ***
## V18         -0.857063   0.264975  -3.235 0.001219 **
## V19          0.006163   0.224878   0.027 0.978137
## V20          1.670892   0.554536   3.013 0.002586 **
## V21          0.834548   0.210275   3.969 7.22e-05 ***
## V22          0.811703   0.555363   1.462 0.143859
## V23          1.787937   0.392435   4.556 5.21e-06 ***
## V24          1.385796   0.343260   4.037 5.41e-05 ***
## V25         -3.611845   0.473164  -7.633 2.29e-14 ***
## V26         -0.640878   0.497465  -1.288 0.197646
## V27         -4.432733   0.740612  -5.985 2.16e-09 ***
## V28          1.981086   0.457457   4.331 1.49e-05 ***
## V29         -1.174992   0.668922  -1.757 0.078996 .
```

```
## V30         -0.183166   0.519469  -0.353 0.724387
## V31         -1.558298   1.033703  -1.507 0.131685
## V32         -2.211046   1.150863  -1.921 0.054706 .
## V33         -0.926369   0.562091  -1.648 0.099337 .
## V34          0.536636   1.068210   0.502 0.615408
## V35         -0.973451   0.565672  -1.721 0.085273 .
## V36          0.636619   0.417226   1.526 0.127050
## V37         -1.440826   0.348518  -4.134 3.56e-05 ***
## V38          1.173486   0.741369   1.583 0.113453
## V39          0.037749   0.413235   0.091 0.927214
## V40         -0.611572   0.557756  -1.096 0.272866
## V41         -5.823151   3.179731  -1.831 0.067051 .
## V42         -2.410825   0.508741  -4.739 2.15e-06 ***
## V43         -1.500599   0.638114  -2.352 0.018692 *
## V44         -1.301660   0.521227  -2.497 0.012514 *
## V45         -1.391117   0.235936  -5.896 3.72e-09 ***
## V46         -1.789877   0.363562  -4.923 8.52e-07 ***
## V47         -0.695873   1.130612  -0.615 0.538235
## V48         -1.512213   0.617515  -2.449 0.014331 *
## V49         -0.070815   0.275485  -0.257 0.797135
## V50          0.185424   0.196176   0.945 0.344561
## V51         -0.056805   0.409948  -0.139 0.889793
## V52          1.476322   0.186253   7.926 2.26e-15 ***
## V53          1.858618   0.250030   7.434 1.06e-13 ***
## V54         -0.794196   0.338409  -2.347 0.018933 *
## V55               NA         NA      NA       NA
## V56               NA         NA      NA       NA
## V57               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_disc = predict(sd_fit_3,train_disc, type='response')
Pred_disc = ifelse(Probs_disc>0.5, 1, 0)
table(Pred_disc, train_disc$V58)
```

```
##
## Pred_disc    0    1
##         0 1779  105
##         1   70 1113
```

```
train_errors_disc <- mean(Pred_disc != train_disc$V58)
cat("\ntrain errors (discretize):", train_errors_disc, "\n")
```

```
##
## train errors (discretize): 0.05705902
```

```
Probs_disc = predict(sd_fit_3,test_disc, type='response')
Pred_disc = ifelse(Probs_disc>0.5, 1, 0)
table(Pred_disc, test_disc$V58)
```

```
##
## Pred_disc   0   1
##         0 859  67
##         1  57 551
```

```
test_errors_disc <- mean(Pred_disc != test_disc$V58)
cat("\ntese errors (discretize):", test_errors_disc, "\n")
```

```
##
## tese errors (discretize): 0.08083442
```

As can be seen from the above results, the prediction error rate in training is 0.05705902 and in test set is 0.08083442 . can be considered accurate.

```
significant_params <- rownames(summary(sd_fit_3)$coefficients
                               [summary(sd_fit_3)$coefficients[,4] < 0.05, ])
for (param_name in significant_params) {
  if(param_name != "(Intercept)"){
        cat("\n",param_name)
  }
  else{
    cat("statistically significant V:")
  }
}
```

```
## statistically significant V:
##  V5
##  V7
##  V8
##  V10
##  V11
##  V13
##  V14
##  V15
##  V16
##  V17
##  V18
##  V20
##  V21
##  V23
##  V24
##  V25
##  V27
##  V28
##  V37
##  V42
##  V43
##  V44
```

```
##  V45
##  V46
##  V48
##  V52
##  V53
##  V54
```

The output above is for all parameters with a P value less than 0.05 in **Data after discretize**, they reject the null hypothesis. So the above parameters are statistically significant.

Question 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)?

Standardized data:

linear discriminant analysis

```
library(MASS)
lda_sd <- lda(V58 ~ ., data = data_train_sd)
lda_pred_sd <- predict(lda_sd, data_train_sd)$class
error_sd_lin <- mean(lda_pred_sd != data_train_sd$V58)
lda_pred_sd2 <- predict(lda_sd, data_test_sd)$class
error_sd_lin2 <- mean(lda_pred_sd2 != data_test_sd$V58)

cat(paste("\nThe training error of LDA model in Standardized data is :",error_sd_lin))
```

```
##
## The training error of LDA model in Standardized data is : 0.10172807303554
```

```
cat(paste("\nThe test error of LDA model in Standardized data is     :",error_sd_lin2))
```

```
##
## The test error of LDA model in Standardized data is     : 0.102998696219035
```

quadratic discriminant analysis

```
qda_sd <- qda(V58 ~ ., data = data_train_sd)
qda_pred_sd <- predict(qda_sd, data_train_sd)$class
error_sd_qda <- mean(qda_pred_sd != data_train_sd$V58)
qda_pred_sd2 <- predict(qda_sd, data_test_sd)$class
error_sd_qda2 <- mean(qda_pred_sd2 != data_test_sd$V58)

cat(paste("\nThe training error of QDA model in Standardized data is :",error_sd_qda))
```

```
##
## The training error of QDA model in Standardized data is : 0.178676230844473
```

```
cat(paste("\nThe test error of QDA model in Standardized data is     :",error_sd_qda2))
```

```
##
## The test error of QDA model in Standardized data is     : 0.17470664928292
```

log-transformed data:

linear discriminant analysis

```
lda_log <- lda(V58 ~ ., data = data_train_log)
lda_pred_log <- predict(lda_log, data_train_log)$class
error_log_lin <- mean(lda_pred_log != data_train_log$V58)
lda_pred_log2 <- predict(lda_log, data_test_log)$class
error_log_lin2 <- mean(lda_pred_log2 != data_test_log$V58)

cat(paste("\nThe training error of LDA model in log-transformed data is :",error_log_lin))
```

```
##
## The training error of LDA model in log-transformed data is : 0.0603195304858168
```

```
cat(paste("\nThe test error of LDA model in log-transformed data is     :",error_log_lin2))
```

```
##
## The test error of LDA model in log-transformed data is     : 0.0651890482398957
```

quadratic discriminant analysis

```
qda_log <- qda(V58 ~ ., data = data_train_log)
qda_pred_log <- predict(qda_log, data_train_log)$class
error_log_qda <- mean(qda_pred_log != data_train_log$V58)
qda_pred_log2 <- predict(qda_log, data_test_log)$class
error_log_qda2 <- mean(qda_pred_log2 != data_test_log$V58)

cat(paste("\nThe training error of QDA model in log-transformed data is :",error_log_qda))
```

```
##
## The training error of QDA model in log-transformed data is : 0.158787088359961
```

```
cat(paste("\nThe test error of QDA model in log-transformed data is     :",error_log_qda2))
```

```
##
## The test error of QDA model in log-transformed data is     : 0.157105606258149
```

Question d

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

Standardized data:

linear SVM model

```
library(e1071)
data_train_sd_SVM <-data_train_sd
data_test_sd_SVM <-data_test_sd
data_train_sd_SVM$V58 = as.factor(data_train_sd_SVM$V58)
data_test_sd_SVM$V58 = as.factor(data_test_sd_SVM$V58)

svm_sd <- svm(V58~ ., data = data_train_sd_SVM, kernel = "linear", cost=5)

svm_pred_sd <- predict(svm_sd,data_train_sd_SVM)
error_lin_svm_sd <- mean(svm_pred_sd != data_train_sd_SVM$V58)

svm_pred_sd2 <- predict(svm_sd, data_test_sd_SVM)
error_lin_svm_sd2 <- mean(svm_pred_sd2 != data_test_sd_SVM$V58)

cat(paste("\nThe training error of linear SVM model in Standardized data is :",error_lin_svm_sd))
```

```
##
## The training error of linear SVM model in Standardized data is : 0.061623736550375
```

```
cat(paste("\nThe test error of linear SVM model in Standardized data is :",error_lin_svm_sd2))
```

```
##
## The test error of linear SVM model in Standardized data is : 0.0697522816166884
```

nonlinear

```
svm_sd <- svm(V58~.,data = data_train_sd_SVM, kernel = "radial", cost=1 , gamma = 0.1)

svm_pred_sd <- predict(svm_sd,data_train_sd_SVM)
error_non_svm_sd <- mean(svm_pred_sd != data_train_sd_SVM$V58)

svm_pred_sd2 <- predict(svm_sd, data_test_sd_SVM)
error_non_svm_sd2 <- mean(svm_pred_sd2 != data_test_sd_SVM$V58)

cat(paste("\nThe training error of nonlinear SVM model in Standardized data is :",error_non_svm_sd))
```

```
##
## The training error of nonlinear SVM model in Standardized data is : 0.0215194000652103
```

```
cat(paste("\nThe test error of SVM nonlinear model in Standardized data is     :",error_non_svm_sd2))
```

```
##
## The test error of SVM nonlinear model in Standardized data is     : 0.0651890482398957
```

Log-transformed data:

linear SVM model

```
data_train_log_SVM <-data_train_log
data_test_log_SVM <-data_test_log
data_train_log_SVM$V58 = as.factor(data_train_log_SVM$V58)
data_test_log_SVM$V58 = as.factor(data_test_log_SVM$V58)

svm_log <- svm(V58~.,data = data_train_log_SVM, kernel = "linear", cost=5)

svm_pred_log <- predict(svm_log,data_train_log_SVM)
error_lin_svm_log <- mean(svm_pred_log != data_train_log_SVM$V58)

svm_pred_log2 <- predict(svm_log, data_test_log_SVM)
error_lin_svm_log2 <- mean(svm_pred_log2 != data_test_log_SVM$V58)

cat(paste("\nThe training error of linear SVM model in Log-transformed data is :",error_lin_svm_log))
```

```
##
## The training error of linear SVM model in Log-transformed data is : 0.0531463971307467
```

```
cat(paste("\nThe test error of linear SVM model in Log-transformed data is :",error_lin_svm_log2))
```

```
##
## The test error of linear SVM model in Log-transformed data is : 0.0495436766623207
```

nonlinear

```
svm_log <- svm(V58~.,data = data_train_log_SVM, kernel = "radial", cost=5 , gamma =0.1)

svm_pred_log <- predict(svm_log,data_train_log_SVM)
error_non_svm_log <- mean(svm_pred_log != data_train_log_SVM$V58)

svm_pred_log2 <- predict(svm_log, data_test_log_SVM)
error_non_svm_log2 <- mean(svm_pred_log2 != data_test_log_SVM$V58)

cat(paste("\nThe training error of nonlinear SVM model in Log-transformed data is :",error_non_svm_log))
```

```
##
## The training error of nonlinear SVM model in Log-transformed data is : 0.0039126181936746
```

```
cat(paste("\nThe test error of nonlinear SVM model in Log-transformed data is :",error_non_svm_log2))
```

```
##
## The test error of nonlinear SVM model in Log-transformed data is : 0.0462842242503259
```

Data after discretize:

linear SVM model

```
data_train_disc_SVM <-train_disc[,-c(55,56,57)]
data_test_disc_SVM <-test_disc[,-c(55,56,57)]
data_train_disc_SVM$V58 = as.factor(data_train_disc_SVM$V58)
data_test_disc_SVM$V58 = as.factor(data_test_disc_SVM$V58)

svm_disc <- svm(V58~.,data = data_train_disc_SVM, kernel = "linear", cost=5)

svm_pred_disc <- predict(svm_disc,data_train_disc_SVM)
error_lin_svm_disc <- mean(svm_pred_disc != data_train_disc_SVM$V58)

svm_pred_disc2 <- predict(svm_disc, data_test_disc_SVM)
error_lin_svm_disc2 <- mean(svm_pred_disc2 != data_test_disc_SVM$V58)

cat(paste("\nThe training error of linear SVM model in discretized data is :",error_lin_svm_disc))
```

```
##
## The training error of linear SVM model in discretized data is : 0.0583632213889795
```

```
cat(paste("\nThe test error of linear SVM model in discretized data is :",error_lin_svm_disc2))
```

```
##
## The test error of linear SVM model in discretized data is : 0.0749674054758801
```

nonlinear

```
svm_disc <- svm(V58~.,data = data_train_disc_SVM, kernel = "radial", cost=5 , gamma =0.1)

svm_pred_disc <- predict(svm_disc,data_train_disc_SVM)
error_non_svm_disc <- mean(svm_pred_disc != data_train_disc_SVM$V58)

svm_pred_disc2 <- predict(svm_disc, data_test_disc_SVM)
error_non_svm_disc2 <- mean(svm_pred_disc2 != data_test_disc_SVM$V58)

cat(paste("\nThe training error of nonlinear SVM model in discretized data is :",error_non_svm_disc))
```

```
##
## The training error of nonlinear SVM model in discretized data is : 0.00749918487120965
```

```
cat(paste("\nThe test error of nonlinear SVM model in discretized data is :",error_non_svm_disc2))
```

```
##
## The test error of nonlinear SVM model in discretized data is : 0.0586701434159061
```

Question e

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

Standardized data:

```
## standard data
library(tree)
library(rpart)
#tree_sd <- tree(V58 ~., data_train_sd)
tree_sd <- rpart(V58 ~., data = data_train_sd, method = 'class')

#summary(tree_sd)
pred_tree_sd <- predict(tree_sd, new_data = data_train_sd, type = 'class')
error_tree_sd <- mean(pred_tree_sd != data_train_sd$V58)

pred_tree_sd2 <- predict(tree_sd, newdata = data_test_sd, type = 'class')
error_tree_sd2 <-  mean(pred_tree_sd2 != data_test_sd$V58)

cat(paste("\nThe training error of classifier tree in standardize data is :",error_tree_sd))
```

```
##
## The training error of classifier tree in standardize data is : 0.0932507336159113
```

```
cat(paste("\nThe test error ofclassifier tree in standardize data is :",error_tree_sd2))
```

```
##
## The test error ofclassifier tree in standardize data is : 0.104302477183833
```

Log-transformed data:

```
## transform data

tree_log <- rpart(V58 ~., data = data_train_log, method = 'class')

#summary(tree_sd)
pred_tree_log <- predict(tree_log, new_data = data_train_log, type = 'class')
error_tree_log <- mean(pred_tree_log != data_train_log$V58)

pred_tree_log2 <- predict(tree_log, newdata = data_test_log, type = 'class')
error_tree_log2 <-  mean(pred_tree_log2 != data_test_log$V58)

cat(paste("\nThe training error of classifier tree in Log-transformed data is :",error_tree_log))
```

```
##
## The training error of classifier tree in Log-transformed data is : 0.0932507336159113
```

```
cat(paste("\nThe test error of classifier tree in Log-transformed data is :",error_tree_log2))
```

```
##
## The test error of classifier tree in Log-transformed data is : 0.0925684485006519
```

Data after discretize:

```
## dicretize data

tree_disc <- rpart(V58 ~., data = train_disc, method = 'class')

#summary(tree_sd)
pred_tree_disc <- predict(tree_disc, new_data = train_disc, type = 'class')
error_tree_disc <- mean(pred_tree_disc != train_disc$V58)

pred_tree_disc2 <- predict(tree_disc, newdata = test_disc, type = 'class')
error_tree_disc2 <-  mean(pred_tree_disc2 != test_disc$V58)

cat(paste("\nThe training error of classifier tree model in discretize data is :",error_tree_disc))
```

```
##
## The training error of classifier tree model in discretize data is : 0.0984675578741441
```

```
cat(paste("\nThe test error of classifier tree model in discretize data is :",error_tree_disc2))
```

```
##
## The test error of classifier tree model in discretize data is : 0.100391134289439
```

Report

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

```
error_table = data.frame(Method = c("Logistic", "LDA", "QDA", "Linear SVM", "NonLinear SVM", "Tree-base
                         SD_Tr_E = c(train_errors_sd, error_sd_lin, error_sd_qda, error_lin_svm_sd, err
                         SD_Te_E = c(test_errors_sd, error_sd_lin2, error_sd_qda2, error_lin_svm_sd2, e
                         Trans_Tr_E = c(train_errors_log, error_log_lin, error_log_qda, error_lin_svm_l
                         Trans_Te_E = c(test_errors_log, error_log_lin2,
                                        error_log_qda2, error_lin_svm_log2,
                                        error_non_svm_log2, error_tree_log2),

                         Disc_Tr_E = c(round(train_errors_disc,7), "NULL", "NULL",
                                       round(error_lin_svm_disc,7),
                                       round(error_non_svm_disc,7),
                                       round(error_tree_disc,7)),
                         Disc_Te_E = c(round(test_errors_disc,7), "NULL", "NULL",
                                       round(error_lin_svm_disc2,7),
                                       round(error_non_svm_disc2,7),
                                       round(error_tree_disc2,7)))

#error_table
knitr::kable(error_table)
```

Method	SD_Tr_E	SD_Te_E	Trans_Tr_E	Trans_Te_E	Disc_Tr_E	Disc_Te_E
Logistic	0.0717313	0.0710561	0.0577111	0.0567145	0.057059	0.0808344
LDA	0.1017281	0.1029987	0.0603195	0.0651890	NULL	NULL
QDA	0.1786762	0.1747066	0.1587871	0.1571056	NULL	NULL
Linear SVM	0.0616237	0.0697523	0.0531464	0.0495437	0.0583632	0.0749674
NonLinear SVM	0.0215194	0.0651890	0.0039126	0.0462842	0.0074992	0.0586701
Tree-based	0.0932507	0.1043025	0.0932507	0.0925684	0.0984676	0.1003911

From the table above we can see the error rates of different methods. We found that the one with the lowest error rate was the non-linear SVM and the highest error rate is QDA.. It has lower error rates than all other methods on all three data sets. Therefore, we can consider using non-linear SVM as our classifier.

Finally

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.

```
## using non-linear SVM

tune_fit_r = tune(svm, V58 ~., data = data_train_log_SVM, kernel = "radial", ranges = list(cost= c(0.01

best_c_r = tune_fit_r$best.model

svm_pred_sd_b_c <- predict(best_c_r,data_train_log_SVM)
error_non_svm_sd_b_c <- mean(svm_pred_sd_b_c != data_train_log_SVM$V58)

svm_pred_sd_b_c2 <- predict(best_c_r, data_test_log_SVM)
error_non_svm_sd_b_c2 <- mean(svm_pred_sd_b_c2 != data_test_log_SVM$V58)

#cat(paste("\nThe training error of nonlinear SVM model in Standardized data is :",error_non_svm_sd_b_c
#cat(paste("\nThe test error of SVM nonlinear model in Standardized data is     :",error_non_svm_sd_b_c
```

```
# The above code needs to run for a long, long time, and here we directly give the result of the operat

cat("\nThe training error of nonlinear SVM model in Standardized data is : 0.0058689272905119")
```

```
##
## The training error of nonlinear SVM model in Standardized data is : 0.0058689272905119
```

```
cat("\nThe test error of SVM nonlinear model in Standardized data is     : 0.0358539765319426")
```

```
##
## The test error of SVM nonlinear model in Standardized data is     : 0.0358539765319426
```

To compare different methods, our recommended method of choice is **nonlinear SVM**. By testing different cost and gamma and using tune( ), we choose the result with the best error rate, and its error rate is as follows:

training error: 0.0058689

test error : 0.0358539

We are now getting lower error rates than our previous lowest error rates, and this approach can be considered effective.