Spam Classification Project

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:

1) Standardize the columns so that they all have zero mean and unit variance;
2) Transform the features using log(xij+1);
3) Discretize each feature using I(xij>0).

##Standardize the columns

```
library(ISLR)
library(MASS)
train = read.table('/Users/lijiajia/Desktop/spam-train.txt', sep = ',', head = FALSE)
train_std = cbind(scale(train[-58]), train[,58])
test = read.table('/Users/lijiajia/Desktop/spam-test.txt', sep = ',', head = FALSE)
test_std = cbind(scale(test[-58]), test[,58])
```

##Transform the features using log(xij + 1)

```
train_log = cbind(log(train[-58] + 1), train[,58])
test_log = cbind(log(test[-58] + 1), test[,58])
```

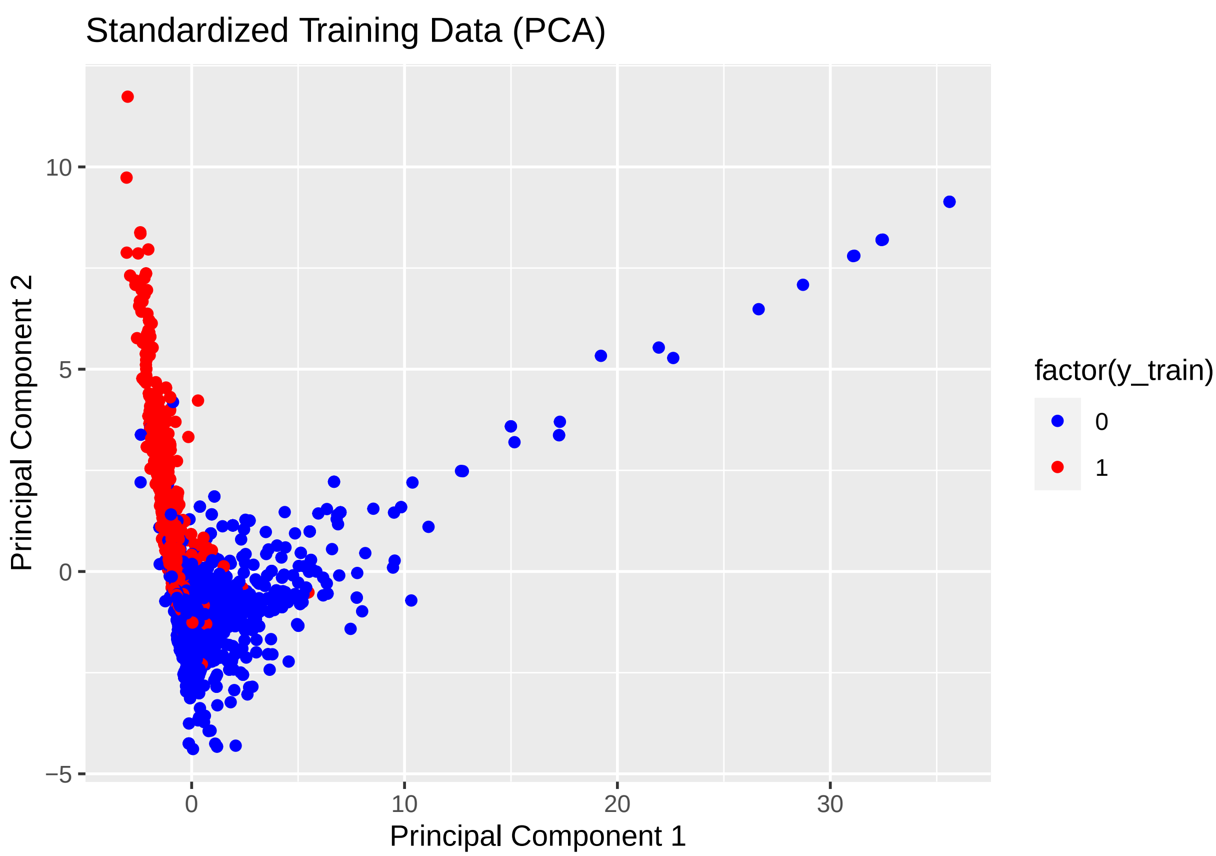
##Discretize each feature using I(xij > 0)

```
train_disc = as.data.frame(lapply(train[,1:58], function(x) as.integer(x>0)))
test_disc = as.data.frame(lapply(test[,1:58], function(x) as.integer(x>0)))
```

##a).For each version of the data, visualize it using the tools introduced in the class. For the standardized data:

```
library(ggplot2)
library(gridExtra)
y_train = train[,58]
x_train = train[-58]
y_test = test[,58]
x_test = test[-58]
x_train_std = scale(x_train)
x_test_std = scale(x_test)
pca_train_std = prcomp(x_train_std)
```

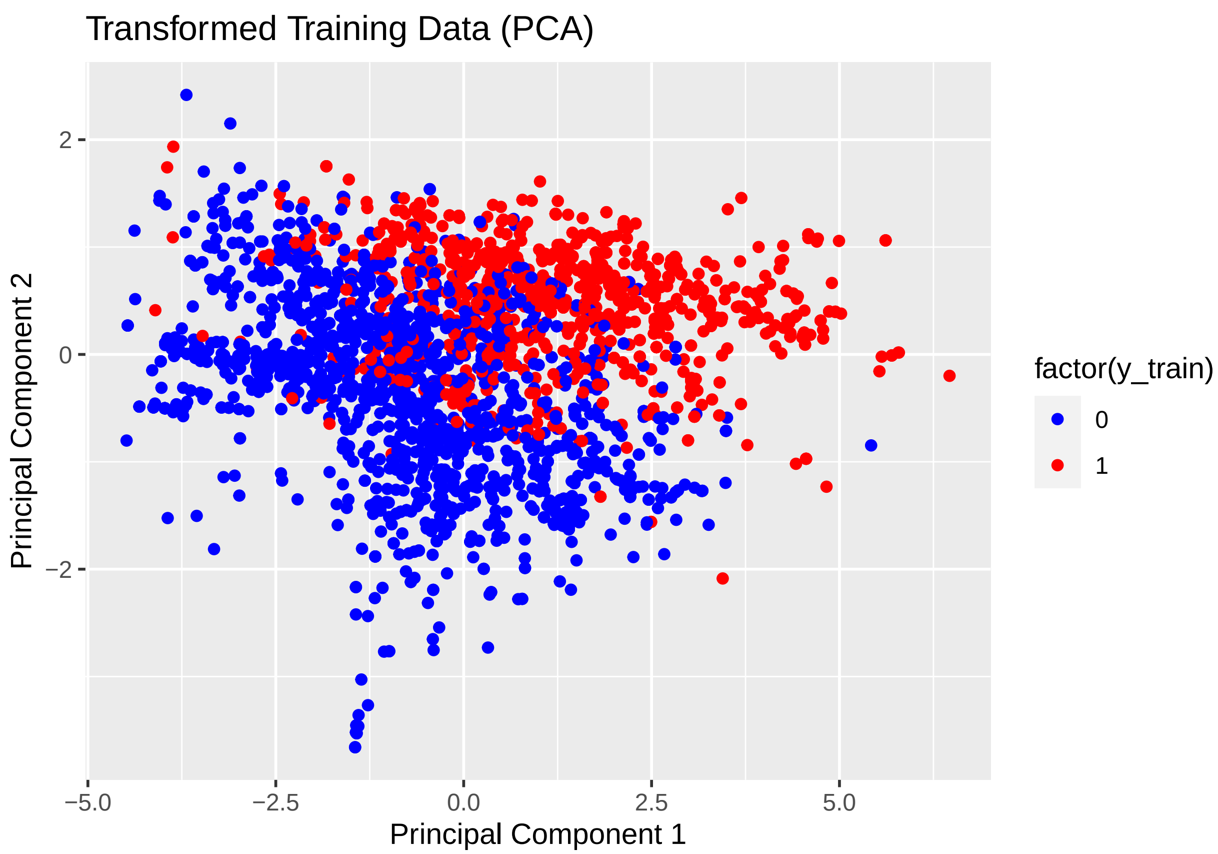
```r
pca_test_std = prcomp(x_test_std)
ggplot(data = data.frame(pca_train_std$x, y_train), aes(x = PC1, y = PC2, color = factor(y_train))) +
geom_point() +
scale_color_manual(values = c("blue", "red")) +
ggtitle("Standardized Training Data (PCA)") +
xlab("Principal Component 1") +
ylab("Principal Component 2")
```




```r
ggplot(data = data.frame(pca_test_std$x, y_test), aes(x = PC1, y = PC2, color = factor(y_test))) +
geom_point() +
scale_color_manual(values = c("blue", "red")) +
ggtitle("Standardized Test Data (PCA)") +
xlab("Principal Component 1") +
ylab("Principal Component 2")
```

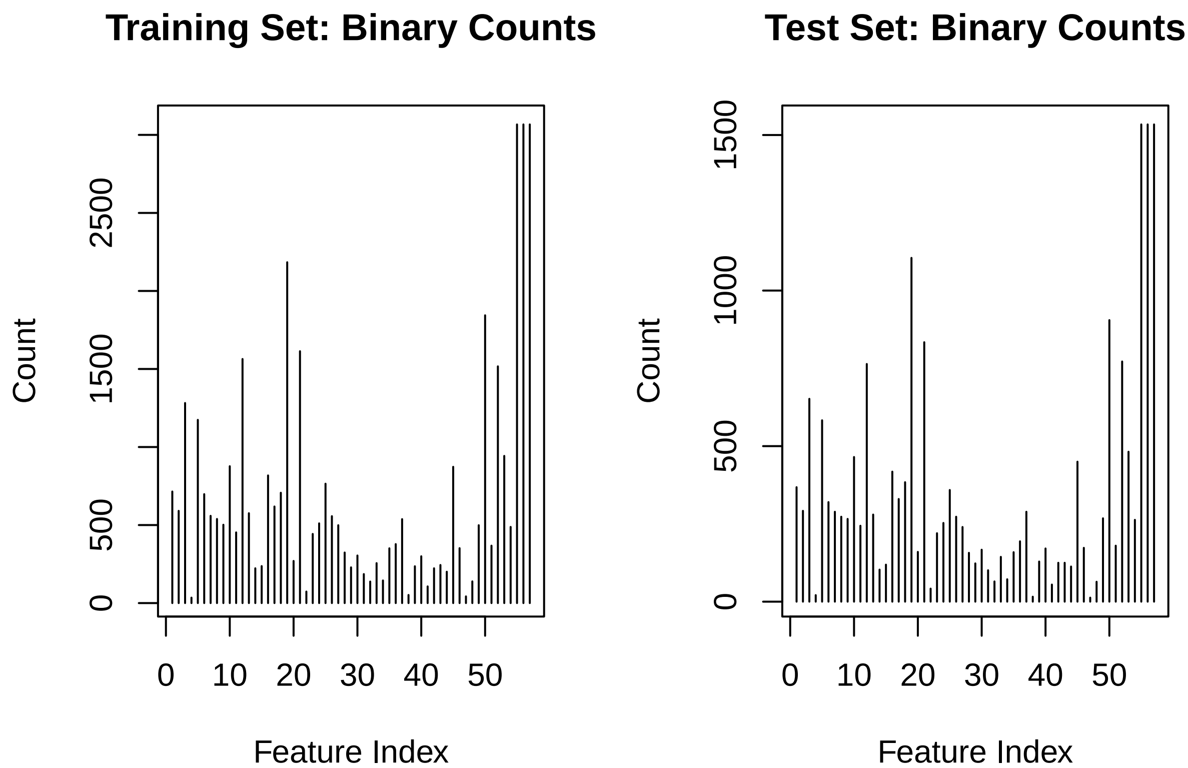
For the transformed data:

```r
x_train_log = log(x_train + 1)
x_test_log = log(x_test + 1)
pca_train_log = prcomp(x_train_log)
pca_test_log = prcomp(x_test_log)
ggplot(data = data.frame(pca_train_log$x, y_train), aes(x = PC1, y = PC2, color = factor(y_train))) +
geom_point() +
scale_color_manual(values = c("blue", "red")) +
ggtitle("Transformed Training Data (PCA)") +
xlab("Principal Component 1") +
ylab("Principal Component 2")
```

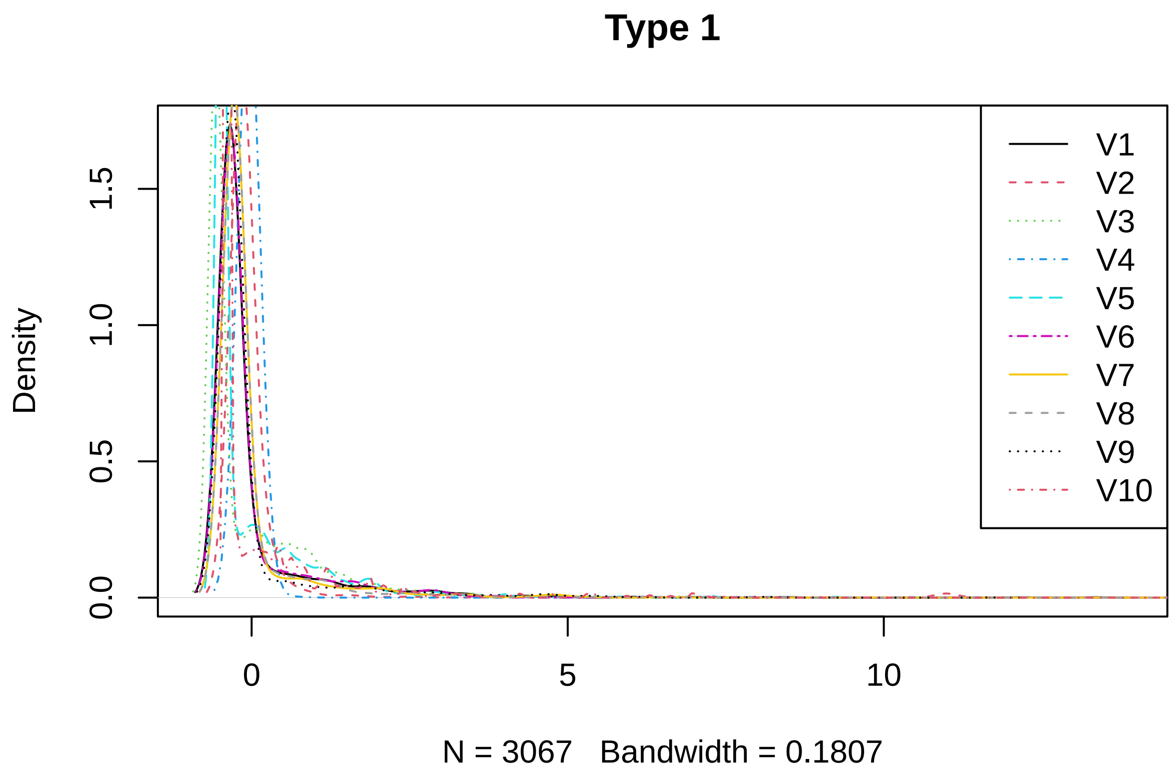


For the discretized data:

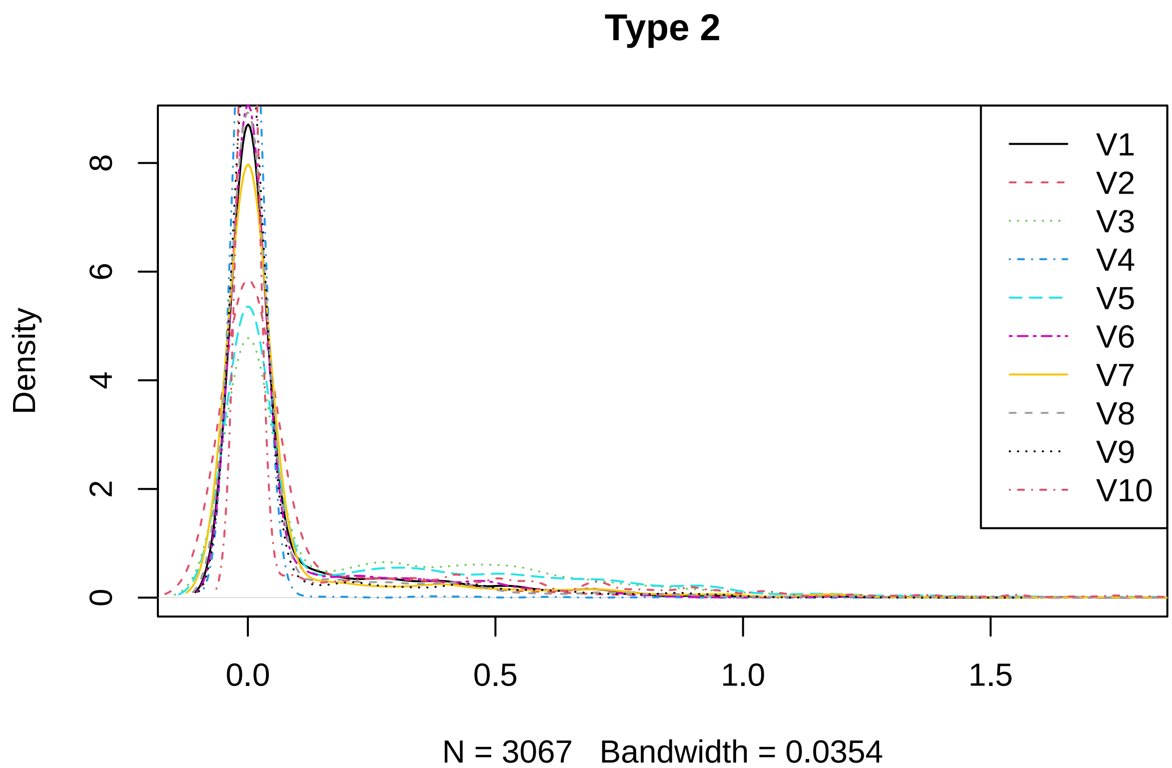
```
train_disc = apply(train[-58], 2, function(x) as.numeric(x > 0))
test_disc = apply(test[-58], 2, function(x) as.numeric(x > 0))
par(mfrow=c(1,2))
plot(colSums(train_disc), type="h", xlab="Feature Index", ylab="Count",
main="Training Set: Binary Counts")
plot(colSums(test_disc), type="h", xlab="Feature Index", ylab="Count",
main="Test Set: Binary Counts")
```



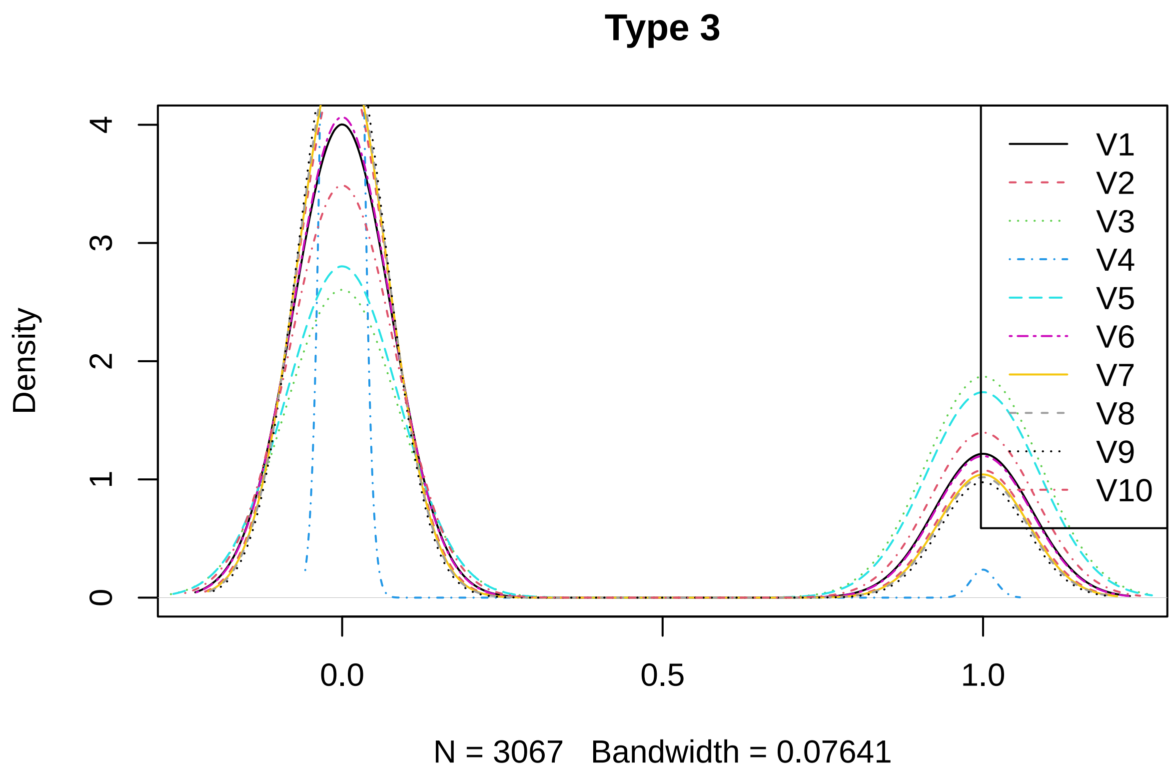

```
plot(density(train_std[,1]), main = 'Type 1')
for(i in 2:10){
  lines(density(train_std[,i]), col = i, lty = i)
}
legend('topright',legend = colnames(train_std)[1:10], col = 1:10, lty = 1:10)
```




```
plot(density(train_log[,1]), main = 'Type 2')
for(i in 2:10){
  lines(density(train_log[,i]), col = i, lty = i)
}
legend('topright',legend = colnames(train_std)[1:10], col = 1:10, lty = 1:10)
```




```
plot(density(train_disc[,1]), main = 'Type 3')
for(i in 2:10){
  lines(density(train_disc[,i]), col = i, lty = i)
}
legend('topright',legend = colnames(train_std)[1:10], col = 1:10, lty = 1:10)
```



##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

```r
train_std_df= as.data.frame(train_std)
test_std_df= as.data.frame(test_std)
train_disc = as.data.frame(lapply(train[,1:58], function(x) as.integer(x>0)))
test_disc = as.data.frame(lapply(test[,1:58], function(x) as.integer(x>0)))

# Fit logistic regression model on standardized data
logit_model_std = glm(V58 ~ .,family = binomial, data = as.data.frame(train_std))
```

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

```r
# Model summary
summary(logit_model_std)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = binomial, data = as.data.frame(train_std))
##
## 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
```

```
# Training set predictions
train_pred_std = predict (logit_model_std, newdata = as.data.frame(train_std), type = 'response')

train_pred_std = ifelse(train_pred_std > 0.5, 1, 0)
train_error_std = mean (train_pred_std != train$V58)
cat('Training erorr (standardized:', train_error_std, '\n')
```

```
## Training erorr (standardized: 0.07173133
```

```
# Test set predictions
test_pred_std = predict(logit_model_std, newdata = as.data.frame(test_std), type = 'response')
test_pred_std = ifelse(test_pred_std > 0.5, 1, 0)
test_error_std = mean (test_pred_std != test$V58)
cat('Training erorr (standardized:', test_error_std, '\n')
```

```
## Training erorr (standardized: 0.07105606
```

Log Transformation

```
names(train_log)[58] = 'V58'
logit_model_log = glm(V58 ~., family = binomial, train_log)
```

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

```
summary (logit_model_log)
```

```
##
## Call:
## glm(formula = V58 ~ ., family = binomial, 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          -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
```

```
# Training set predictions
train_pred_disc = predict(logit_model_log, train_log, type = 'response')
train_error_disc = mean((train_pred_disc > 0.5) != train$V58)

# Test set predictions
test_pred_disc = predict(logit_model_log, test_log, type = 'response')
test_error_disc = mean((test_pred_disc > 0.5) != test$V58)

cat('Training error(log transform):', train_error_disc, '\n')
```

```
## Training error(log transform): 0.05771112
```

```
cat('Training error(log_transform):', test_error_disc, '\n')
```

```
## Training error(log_transform): 0.05671447
```

Discretized

```
logit_model_disc = glm(V58 ~ ., train_disc, family = 'binomial')
summary(logit_model_disc)
```

```
##
## 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
```

Training set predictions

```r
train_pred_disc = predict(logit_model_disc, newdata = train_disc, type ='response')
```

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

```r
train_error_disc = mean((train_pred_disc > 0.5) != train$V58)
```

Test set predictions

```r
test_pred_disc = predict(logit_model_disc, newdata = test_disc, type ='response')
```

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

```r
test_error_disc = mean((test_pred_disc > 0.5) != test$V58)
```

```r
cat('Train error (discretized):', train_error_disc, '\n')
```

```
## Train error (discretized): 0.05705902
```

```r
cat('Test error (discretized):', test_error_disc, '\n')
```

```
## Test error (discretized): 0.08083442
```

According to the logistic regression model, the training error for each data are respectively 0.102541, 0.0609716, and 0.4734268, and the test error are respectively 0.565189,0.4843546, and 0.1056063. And the ones are appear to be statistically significant than others are marked significant in all three types, namely,the following predictors: V1-8,V12,V16-27,V29,V33-35,V37-38,V42,V44-49,V52-53, and V57.

##(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)?

QDA and LDA:

```r
train_x <- train[, 1:57]
train_y <- train[, 58]
train_x_std <- scale(train_x)
train_x_log <- log(train_x + 1)
library(MASS)
lda_model_std <- lda(train_x_std, train_y)
lda_model_log <- lda(train_x_log, train_y)
qda_model_std <- qda(train_x_std, train_y)
qda_model_log <- qda(train_x_log, train_y)
```

Evaluate the models on the training set:

```r
lda_error_train_std = mean(lda_model_std$class != train_y)
lda_error_train_log = mean(lda_model_log$class != train_y)
qda_error_train_std = mean(qda_model_std$class != train_y)
qda_error_train_log = mean(qda_model_log$class != train_y)
```

```r
test_x = test[, 1:57]
test_y = test[, 58]
test_x_std = scale(test_x)
test_x_log = log(test_x + 1)
library(MASS)
lda_model_std = lda(train_x_std, train_y)
lda_model_log = lda(train_x_log, train_y)
qda_model_std = qda(train_x_std, train_y)
qda_model_log = qda(train_x_log, train_y)
```

```r
lda_error_test_std = mean(predict(lda_model_std,test_x_std)$class != test_y)
lda_error_test_log = mean(predict(lda_model_log, test_x_log)$class != test_y)
qda_error_test_std = mean(predict(qda_model_std, test_x_std)$class != test_y)
qda_error_test_log = mean(predict(qda_model_log, test_x_log)$class != test_y)
cat("Linear discriminant analysis (standardized data):\n")
```

```
## Linear discriminant analysis (standardized data):
```

```r
## Linear discriminant analysis (standardized data):
cat(paste("Classification error:", lda_error_test_std, "\n\n"))
```

```
## Classification error: 0.102998696219035
```

```r
cat("Linear discriminant analysis (log-transformed data):\n")
```

```
## Linear discriminant analysis (log-transformed data):
```

```r
## Linear discriminant analysis (log-transformed data):
cat(paste("Classification error:", lda_error_test_log, "\n\n"))
```

```
## Classification error: 0.0651890482398957
```

```r
cat("Quadratic discriminant analysis (standardized data):\n")
```

```
## Quadratic discriminant analysis (standardized data):
```

```r
## Quadratic discriminant analysis (standardized data):
cat(paste("Classification error:", qda_error_test_std, "\n\n"))
```

```
## Classification error: 0.17470664928292
```

```r
cat("Quadratic discriminant analysis (log-transformed data):\n")
```

```
## Quadratic discriminant analysis (log-transformed data):
```

```r
## Quadratic discriminant analysis (log-transformed data):
cat(paste("Classification error:", qda_error_test_log, "\n"))
```

```
## Classification error: 0.157105606258149
```

We deal with the LDA and QDA, so we evaluate the models on the training and test sets, separately. Then we print out the classification errors.

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

For standardized data:

```r
library(rpart)
tree_model_std <- rpart(as.factor(V58) ~ ., data=as.data.frame(train_std))
train_pred_std <- predict(tree_model_std, as.data.frame(train_std), type="class")
train_err_std <- mean(train_pred_std != train$V58)
cat("Training error (standardized):", train_err_std, "\n")
```

```
## Training error (standardized): 0.09325073
```

```r
test_pred_std <- predict(tree_model_std, as.data.frame(test_std), type="class")
test_err_std <- mean(test_pred_std != test$V58)
cat("Test error (standardized):", test_err_std, "\n")
```

```
## Test error (standardized): 0.1043025
```

For log transformed data:

```
library(rpart)
tree_model_std <- rpart(as.factor(V58) ~ ., data=as.data.frame(train_log))
train_pred_std <- predict(tree_model_std, as.data.frame(train_log), type="class")
train_err_std <- mean(train_pred_std != train$V58)
cat("Training error (standardized):", train_err_std, "\n")
```

```
## Training error (standardized): 0.09325073
```

```
test_pred_std <- predict(tree_model_std, as.data.frame(test_std), type="class")
test_err_std <- mean(test_pred_std != test$V58)
cat("Test error (standardized):", test_err_std, "\n")
```

```
## Test error (standardized): 0.1655802
```

For the discretized data:

```
library(rpart)
tree_model_std <- rpart(as.factor(V58) ~ ., data=as.data.frame(train_disc))
train_pred_std <- predict(tree_model_std, as.data.frame(train_disc), type="class")
train_err_std <- mean(train_pred_std != train$V58)
cat("Training error (standardized):", train_err_std, "\n")
```

```
## Training error (standardized): 0.09846756
```

```
test_pred_std <- predict(tree_model_std, as.data.frame(test_disc), type="class")
test_err_std <- mean(test_pred_std != test$V58)
cat("Test error (standardized):", test_err_std, "\n")
```

```
## Test error (standardized): 0.1003911
```