Analysis on Credit Approval Dataset using R Language*

Yuming Xie† Prof. Huaming Peng‡

December 11, 2022

Abstract

Credit cards are a safe and efficient way to spend money in the daily life, and one can also get a lot of cash back. The process of determining the approval of a credit card application is much the same, and the difference may come from the weight of different characteristics of the applicant. This paper uses the dataset from the archives of machine learning repository of University of California, Irvine to (1) analyze which characteristics or features are highly weighted in credit card approval and to (2) identify the best model for accurate application and prediction of credit card application outcomes. We use various statistical learning methods to analyze and summarize the data set. For consumers and financial analysts, this is a model that can be manipulated and incorporated into their own judgments about the basis for credit card approval.

Keywords: Credit approval, credit application prediction, statistical learning methods

JEL Codes: C10

*The Credit Approval dataset is obtained in the archives of machine learning repository of University of California, Irvine `http://archive.ics.uci.edu/ml/datasets/credit+approval`. The R type is R-4.2.2.

†The author of this paper.

‡The professor of ECON 4280 - Econometric Methods for Big Data.

Appendix

Contents

Appendix **1**

1 Introduction **2**

2 Exploratory Data Analysis **3**

3 Evaluating Methods and Models **5**

4 Discussion and Comparison of the Models **24**

5 Conclusion **25**

Tables **31**

R code **32**

References **38**

1 Introduction

Credit risk is a popular issue in financial risk management, which refers to the risk that the borrower (or debtor) fails to meet the contract requirements and brings economic losses to the lender (or creditor). As one know, data is a key factor for effective and accurate measurement of risks. Especially with the development of information technology and the Internet, financial risk management relies more and more on massive data. For financial enterprises, big data can solve marketing, pricing, fraud, credit and other problems caused by information asymmetry. Compared with traditional credit risk analysis, the most fundamental innovation of financial credit risk analysis under the background of big data lies in the use of a large number of non-financial data for modeling. In terms of the application of data mining technology in the banking industry in the financial field, the hidden rules and patterns can be mastered through the processing and analysis of massive collected data, and then the consumption habits and interests of a certain customer or consumer group can be found, so as to predict its future demand. We can also analyze and classify the information of a large number of loan customers to find the hidden rules and patterns behind their default, so as to evaluate and forecast the credit risk. Credit evaluation is a classification process based on statistics and data mining research methods. Discriminant analysis and regression have been the most widely used techniques to construct credit evaluation models. In addition, there are logistic regression, probability unite analysis, nonlinear smoothing methods especially KNN, optimization theory, Markov model, recursive partition, expert model, genetic algorithm and neural network, etc. To construct a credit evaluation model, firstly obtain the relevant data of the applicant, then conduct quantitative processing on the information, then select appropriate data mining technology, establish several models, analyze the fitness of data and verify appropriate model, obtain the comprehensive credit score of the customer, set a reasonable threshold, and determine whether the customer pass the evaluation. The entire analysis used in this article is completed in the open source statistical environment

of Language R. The structure of the report is as follows: Section two describes the details of the data set and the exploratory analysis, the third section discusses the methods and techniques used in the analysis and modeling, the fourth section discusses and compares the analysis and modeling for the credit risk assessment and prediction, and the fifth section summarizes the factors that affect the credit risk assessment of the applicant. The R code for the entire article can be found in the appendix.

2 Exploratory Data Analysis

1. Data Description and Abstraction

   The data set from UCI contains 690 information cases and 16 variables (V1-V16). The first 15 variables (V1-V15) represent different attributes of each case, and the 16^{th} variable (V16) represents the result of the credit card application ($'+'$ represents the application was approved and $'-'$ represents the application was rejected). The variables for this dataset are composed of continuous, letters, and symbols (class attribute). The table for each variable and the corresponding attribute can be found in the *Tables* section. The structure and composition of this data set is shown below:

```
> str(rare_data)
'data.frame': 690 obs. of  16 variables:
 $ V1 : chr  "b" "a" "a" "b" ...
 $ V2 : chr  "30.83" "58.67" "24.50" "27.83" ...
 $ V3 : num  0 4.46 0.5 1.54 5.62 ...
 $ V4 : chr  "u" "u" "u" "u" ...
 $ V5 : chr  "g" "g" "g" "g" ...
 $ V6 : chr  "w" "q" "q" "w" ...
 $ V7 : chr  "v" "h" "h" "v" ...
 $ V8 : num  1.25 3.04 1.5 3.75 1.71 ...
 $ V9 : chr  "t" "t" "t" "t" ...
 $ V10: chr  "t" "t" "f" "t" ...
```

```
$ V11: int  1 6 0 5 0 0 0 0 0 0 ...
$ V12: chr  "f" "f" "f" "t" ...
$ V13: chr  "g" "g" "g" "g" ...
$ V14: chr  "00202" "00043" "00280" "00100" ...
$ V15: int  0 560 824 3 0 0 31285 1349 314 1442 ...
$ V16: chr  "+" "+" "+" "+" ...
```

Listing 1: Structure of the data set

Among all the data, 37 cases were missing one or more attribute information, accounting for about 5.36% of the total data set. The attribute distribution of missing data is shown in the figure below:

```
> sapply(df, function(x) sum(is.na(x)))
 V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16
 12  12   0   6   6   9   9   0   0   0   0   0   0  13   0   0
```

Listing 2: Data missing status for each attribute

2. Handling the Missing Data

   The data set contains missing values over 7 of the 16 variables. These missing values are found in 37 of the 690 cases representing about 5.36% of the data. The missing values are found in the following attributes: V1(Age), V2(Gender), V4(Marital Status), V5(Bank Customer), V6(Education Level), V7(Ethnicity) and V14(Zip Code). Note V1(age) is the only continuous type and the rest are letters type. There are various methods to handle these missing values which can range from deleting the observation reporting the missing data to replacing the missing data with the average of the feature in which the data is missing. For this data set, we use the mean values to replace the missing values in Age attribute and delete the rest of the observations which reports the missing data.

3. Data Normalization and Transformation:

Since all the continuous attributes are measured with different scales along with high variances, this may affect the analysis by finding improper correlation between each pair of attributes. Thus in order to get better and more accurate analysis on the continuous attributes, V2(Age), V3(Debt), V8(Years Employed), V11(Credit Scores), and V15(Income) are normalized using the Min-Max Scaling.

4. Data Visualization

In order to better observe, interact with, and higher understand data, we apply the data visualization method. The above plots indicates The frequency distribution tables for V2(Age), V3(Debt), V8(Years Employed), V11(Credit Scores), and V15(Income). As we can observe that all these attributes have distributions that are skewed to the right which indicates that the data may not performs as a standard distribution, it follows that there are very few individuals recorded attribute values that are higher than the mean in the data set. Now for the discrete variables in the data set, we first transform the attribute to 0(False) and 1(True) and then convert the categorical variables to the corresponding numerical values. The detailed table is listed in section *Tables*. Then we obtain the Scatter plot matrix showing pairwise comparisons as below:

3 Evaluating Methods and Models

1. Logistic Regression

The model is prepared using *glm* function and binomial family. The summary of the model below shows the values of coefficients of each variable and the asterisk states the importance.

```
> summary(logitMod)

Call:
glm(formula = V16 ~ ., family = binomial(link = "logit"), data = train)
```

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3732  -0.3866  -0.2291   0.5429   2.6008

Coefficients: (1 not defined because of singularities)
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.0823605  0.8588531  -2.425  0.01533 *
V1           0.0215086  0.3278303   0.066  0.94769
V2          -0.0117128  0.0155631  -0.753  0.45169
V3          -0.0206797  0.0333531  -0.620  0.53524
V4          -0.3561885  0.3458819  -1.030  0.30310
V5                  NA         NA      NA       NA
V6          -0.0092663  0.0353997  -0.262  0.79350
V7          -0.1243808  0.0989672  -1.257  0.20883
V8           0.1584157  0.0657979   2.408  0.01606 *
V9           3.2934457  0.3429965   9.602  < 2e-16 ***
V10          0.3907150  0.4140740   0.944  0.34538
V11          0.1106513  0.0640224   1.728  0.08393 .
V12         -0.0593849  0.3046813  -0.195  0.84546
V13          0.2907790  0.2688820   1.081  0.27950
V14         -0.0009663  0.0009164  -1.054  0.29169
V15          0.0005426  0.0001957   2.773  0.00556 **
---
Signif. codes:  0  ***   0.001   **   0.01   *   0.05   .   0.1       1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 642.66  on 466  degrees of freedom
Residual deviance: 311.99  on 452  degrees of freedom
AIC: 341.99

Number of Fisher Scoring iterations: 7
```

Listing 3: Summary of the model

we then compute the McFadden's R^2, which ranges from 0 to just under 1, to see how well the model fits the data. Values close to 0 indicate that the model has no predictive power. In practice, values over 0.40 indicate that a model fits the data very well.

```
> pscl::pR2(logitMod)["McFadden"]
fitting null model for pseudo-r2
 McFadden
0.5145343
```

Listing 4: McFadden's R^2

A value of 0.5145343 is quite high for McFadden's R^2, which indicates that our model fits the data very well and has high predictive power. We can also compute the importance of each predictor variable in the model by using the *varImp* function from the *caret* package:

```
> caret::varImp(logitMod)
       Overall
V1  0.06560895
V2  0.75259753
V3  0.62002361
V4  1.02979798
V6  0.26176326
V7  1.25678803
V8  2.40760944
V9  9.60197930
V10 0.94358749
V11 1.72832206
V12 0.19490834
V13 1.08143723
V14 1.05441063
```

```
V15 2.77285323
```

Listing 5: The importance of each predictor variable in the model

Higher values indicate more importance. These results match up nicely with the p-values from the model. V9 is by far the most important predictor variable, followed by the other variables. we then create a confusion matrix which shows our predictions compared to the actual defaults:

```
> confusionMatrix(data = as.factor(as.numeric(pdata>0.5)), reference = as.factor(
    train$V16))
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 213  19
         1  44 191

               Accuracy : 0.8651
                 95% CI : (0.8307, 0.8948)
    No Information Rate : 0.5503
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7304

 Mcnemar's Test P-Value : 0.002497

            Sensitivity : 0.8288
            Specificity : 0.9095
         Pos Pred Value : 0.9181
         Neg Pred Value : 0.8128
             Prevalence : 0.5503
         Detection Rate : 0.4561
```

```
  Detection Prevalence : 0.4968
     Balanced Accuracy : 0.8692

      'Positive' Class : 0
```

Listing 6: Confusion matrix

we can plot the ROC (Receiver Operating Characteristic) Curve which displays the percentage of true positives predicted by the model as the prediction probability cutoff is lowered from 1 to 0. The higher the AUC (area under the curve), the more accurately our model is able to predict outcomes:

2. LDA

Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. The summary of the LDA model is as below:

```
> lda.model
Call:
lda(V16 ~ ., data = train)

Prior probabilities of groups:
        0         1
0.5452261 0.4547739

Group means:
         V1       V2       V3       V4       V5       V6       V7       V8
    V9
0 0.2811060 29.97534 3.987811 1.267281 1.267281 6.834101 2.258065 1.481982
    0.2350230
```

```
1 0.3370166 34.19954 6.369116 1.154696 1.154696 6.856354 1.867403 3.687376
    0.9502762
        V10       V11       V12      V13      V14       V15
0 0.2811060 0.8110599 0.4700461 1.207373 203.3180  219.6959
1 0.7071823 5.0165746 0.5138122 1.110497 161.1492 1730.2155

Coefficients of linear discriminants:
              LD1
V1  -5.901972e-02
V2   1.354140e-03
V3   2.650697e-03
V4  -7.829595e-02
V5  -7.829595e-02
V6  -9.487269e-03
V7  -4.910525e-02
V8   3.341680e-02
V9   2.448841e+00
V10  4.765693e-01
V11  2.175018e-02
V12 -8.544533e-02
V13 -8.524687e-04
V14 -9.585828e-04
V15  4.909383e-05
```

Listing 7: Summary of the model

As the next step, we will find the confusion matrix for training data.

```
> confusionMatrix(data = as.factor(predmodel.train.lda$class), reference = as.
    factor(train$V16))
Confusion Matrix and Statistics

          Reference
Prediction   0   1
```

```
        0 168   9
        1  49 172

               Accuracy : 0.8543
                 95% CI : (0.8157, 0.8874)
    No Information Rate : 0.5452
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7114

 Mcnemar's Test P-Value : 3.04e-07

            Sensitivity : 0.7742
            Specificity : 0.9503
         Pos Pred Value : 0.9492
         Neg Pred Value : 0.7783
             Prevalence : 0.5452
         Detection Rate : 0.4221
   Detection Prevalence : 0.4447
      Balanced Accuracy : 0.8622

       'Positive' Class : 0
```

Listing 8: Confusion matrix for training data

Note we get the accuracy of 0.8543, which is lower than the logistic model. The plot below shows how the response class has been classified by the LDA classifier. The X-axis shows the value of line defined by the co-efficient of linear discriminant for LDA model. The two groups are the groups for response classes.

Then we find the confusion matrix for testing data:

```
> confusionMatrix(data = as.factor(predmodel.test.lda$class), reference = as.factor
    (test$V16))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 125   8
         1  24 109

               Accuracy : 0.8797
                 95% CI : (0.8344, 0.9162)
    No Information Rate : 0.5602
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7594

 Mcnemar's Test P-Value : 0.00801

            Sensitivity : 0.8389
            Specificity : 0.9316
         Pos Pred Value : 0.9398
         Neg Pred Value : 0.8195
             Prevalence : 0.5602
         Detection Rate : 0.4699
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.8853

       'Positive' Class : 0
```

Listing 9: Confusion matrix for testing data

As we can observe that the accuracy of the testing data is 0.8797, which is a little better than the logistic model. The figure below shows how the test data has been classified. The Predicted Group-1 and Group-2 has been colored with actual classification with red and green color. The mix of red and green color in the Group-1 and Group-2 shows

the incorrect classification prediction.

3. QDA

Quadratic Discriminant Analysis (QDA) is a generative model. QDA assumes that each class follow a Gaussian distribution. The class-specific prior is simply the proportion of data points that belong to the class. The class-specific mean vector is the average of the input variables that belong to the class. The summary of the QDA model is as below:

```
> qda.model
Call:
qda(V16 ~ V2 + V3 + V8 + V11 + V15, data = train)

Prior probabilities of groups:
        0         1
0.5452261 0.4547739

Group means:
        V2       V3       V8       V11       V15
0 29.97534 3.987811 1.481982 0.8110599  219.6959
1 34.19954 6.369116 3.687376 5.0165746 1730.2155
```

Listing 10: Summary of the model

As the next step, we will find the confusion matrix for training data.

```
> confusionMatrix(data = as.factor(predmodel.train.qda$class), reference = as.
    factor(train$V16))
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 201 103
         1  16  78
```

```
8
9                Accuracy : 0.701
10                 95% CI : (0.6534, 0.7456)
11    No Information Rate : 0.5452
12    P-Value [Acc > NIR] : 1.467e-10
13
14                  Kappa : 0.372
15
16 Mcnemar's Test P-Value : 3.181e-15
17
18            Sensitivity : 0.9263
19            Specificity : 0.4309
20         Pos Pred Value : 0.6612
21         Neg Pred Value : 0.8298
22             Prevalence : 0.5452
23         Detection Rate : 0.5050
24   Detection Prevalence : 0.7638
25      Balanced Accuracy : 0.6786
26
27       'Positive' Class : 0
```

Listing 11: Confusion matrix for training data

Note we get the accuracy of 0.701, which is lower than the logistic model and the LDA model. Then we find the confusion matrix for testing data:

```
1 > confusionMatrix(data = as.factor(predmodel.test.qda$class), reference = as.factor
    (test$V16))
2 Confusion Matrix and Statistics
3
4           Reference
5 Prediction   0   1
6          0 146  65
7          1   3  52
```

```
8
9                Accuracy : 0.7444
10                 95% CI : (0.6875, 0.7957)
11    No Information Rate : 0.5602
12    P-Value [Acc > NIR] : 3.704e-10
13
14                  Kappa : 0.4499
15
16 Mcnemar's Test P-Value : 1.389e-13
17
18            Sensitivity : 0.9799
19            Specificity : 0.4444
20         Pos Pred Value : 0.6919
21         Neg Pred Value : 0.9455
22             Prevalence : 0.5602
23         Detection Rate : 0.5489
24   Detection Prevalence : 0.7932
25      Balanced Accuracy : 0.7122
26
27       'Positive' Class : 0
```

Listing 12: Confusion matrix for testing data

As we can observe that the accuracy of the testing data is 0.7444, which is lower than the logistic model and the LDA model. The figure below shows how the test data has been classified using the QDA model. The Predicted Group-1 and Group-2 has been colored with actual classification with red and green color. The mix of red and green color in the Group-1 and Group-2 shows the incorrect classification prediction.

4. KNN

In statistics, the k-nearest neighbors (KNN) is a non-parametric supervised learning method that is a type of classification where the function is only approximated locally

and all computation is deferred until function evaluation. The attributes in the training and testing set are chosen randomly, and the continuous variables are normalized. By using the knn function in the *Class* library with initial value of k=1 on test and training dataset, we iterate through until k = 30 and store the accuracy of each k. The plot below shows the visualization of the model assessment:

As we can observe that the optimal k is 5 and it's corresponding accuracy is 0.697561:

```
> r[which.max(r$accuracy),]
  k accuracy
5 5 0.697561
```

Listing 13: The accuracy for K

5. Ridge Regression

Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated, in other words, multicollinearity is present in the data. We use the glmnet function to fit the ridge regression model and specify alpha = 0. Below is the summary of the model:

```
> summary(model)
           Length Class     Mode
a0          100   -none-    numeric
beta       1500   dgCMatrix S4
df          100   -none-    numeric
dim           2   -none-    numeric
lambda      100   -none-    numeric
dev.ratio   100   -none-    numeric
nulldev       1   -none-    numeric
npasses       1   -none-    numeric
jerr          1   -none-    numeric
offset        1   -none-    logical
call          4   -none-    call
```

```
nobs         1   -none-    numeric
```

Listing 14: Summary of the ridge model

Then we identify the lambda value that produces the lowest test mean squared error (MSE) by using k-fold cross-validation:

```
> best_lambda
[1] 0.03643625
```

Listing 15: Best lambda value for the ridge regression

And the plot of test MSE by lambda value:

Then we obtain the coefficient estimates for this model:

```
> coef(best_model)
16 x 1 sparse Matrix of class "dgCMatrix"
                        s0
(Intercept)  1.717795e-01
V1           4.336520e-03
V2           1.182444e-04
V3          -8.681321e-04
V4          -2.607551e-02
V5          -2.225078e-02
V6           3.016006e-06
V7          -1.416337e-02
V8           1.108899e-02
V9           5.569921e-01
V10          1.292432e-01
V11          7.858049e-03
V12         -1.733449e-02
V13         -6.163051e-03
V14         -1.518647e-04
V15          1.092836e-05
```

Listing 16: Coefficients of the ridge model

And we have the trace plot to visualize how the coefficient estimates changed as a result of increasing lambda:

Then we calculate the R-squared of the model on the training data:

```
> rsq
[1] 0.5880085
```

Listing 17: R^2 of the ridge model

6. Lasso Regression

In statistics and machine learning, lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model. We use the glmnet function to fit the lasso regression model and specify alpha = 1. To determine what value to use for lambda, we'll perform k-fold cross-validation and identify the lambda value that produces the lowest test mean squared error (MSE). And we obtain that

```
> best_lambda
[1] 0.007320891
```

Listing 18: Best lambda value for the ridge regression

And the plot of test MSE by lambda value:

Then we obtain the coefficient estimates for this model:

```
> coef(best_model)
16 x 1 sparse Matrix of class "dgCMatrix"
                        s0
(Intercept)  1.134121e-01
V1           .
```

```
V2            .
V3            .
V4           -3.217445e-02
V5           -2.314544e-15
V6            .
V7           -9.399675e-03
V8            8.161286e-03
V9            6.025895e-01
V10           1.199181e-01
V11           6.696885e-03
V12          -5.052363e-03
V13           .
V14          -1.163461e-04
V15           9.704320e-06
```

Listing 19: Coefficients of the ridge model

No coefficient is shown for the predictor gender, age, debt, edu, and citizen because the lasso regression shrunk the coefficient all the way to zero. This means it was completely dropped from the model because it wasn't influential enough. Note that this is a key difference between ridge regression and lasso regression. Ridge regression shrinks all coefficients towards zero, but lasso regression has the potential to remove predictors from the model by shrinking the coefficients completely to zero. Then we calculate the R-squared of the model on the training data:

```
> rsq
[1] 0.5882604
```

Listing 20: R^2 of the lasso model

7. PCR Model

Principal components regression (PCR) is a regression technique based on principal component analysis (PCA). The basic idea behind PCR is to calculate the principal

components and then use some of these components as predictors in a linear regression model fitted using the typical least squares procedure. First we need to determine the number of principal components worth keeping. The way to do so is by looking at the test root mean squared error (test RMSE) calculated by the k-fold cross-validation:

```
> summary(model)
Data:   X dimension: 664 15
  Y dimension: 664 1
Fit method: svdpc
Number of components considered: 15

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
CV          0.4981   0.4004   0.3988   0.3855   0.3819   0.3812   0.3789   0.3782
adjCV       0.4981   0.4006   0.3989   0.3853   0.3814   0.3808   0.3786   0.3787
       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
CV      0.3790   0.3790    0.3771    0.3364    0.3326    0.3275    0.3270    0.3275
adjCV   0.3779   0.3786    0.3764    0.3354    0.3322    0.3270    0.3265    0.3266

TRAINING: % variance explained
     1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9
    comps
X      18.60    31.39    42.06    50.71    58.34    64.84    70.86    76.67
    81.73
V16    35.05    35.93    41.22    42.86    43.41    44.08    44.26    45.74
    47.36
     10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X       86.64     91.03     94.52     97.51       100    100.00
V16     48.31     56.46     57.32     58.82        59     59.02
```

Listing 21: Summary of the model

As we can see there are to tables in the output:

(a) VALIDATION: RMSEP

This table tells us the test RMSE calculated by the k-fold cross validation.

i. If we only use the intercept term in the model, the test RMSE is 0.4981.

ii. If we add 1 principal component, the test RMSE drops to 0.4006.

iii. If we add 2 principal components, the test RMSE drops to 0.3989.

We can see that adding additional principal components actually leads to a decrease in test RMSE. Thus, it appears that it would be optimal to use as many principal components as possible in the model.

(b) TRAINING: % variance explained This table tells us the percentage of the variance in the response variable explained by the principal components.

i. If we only use the intercept term in the model, we can explain 18.6% of the variation in the response variable.

ii. If we add 1 principal component, we can explain 31.39% of the variation in the response variable.

iii. If we add 2 principal components, we can explain 42.06% of the variation in the response variable.

Note that we can explain more variance by using more principal components.

The plot below is the visualization the test RMSE (along with the test MSE and R-squared) based on the number of principal components by using the validationplot function.

As we can observe that the optimal model includes all the 15 principal components. Now we run the obtained model on the testing data set and have the following test RMSE:

```
> sqrt(mean((pcr_pred - test$V16)^2))#calculate RMSE
[1] 0.3002703
```

Listing 22: Test RMSE on the PCR model

We can see that the test RMSE turns out to be 0.3. This is the average deviation between the predicted value for result and the observed value for result for the observations in the testing set.

8. PLS Model

The Partial Least Squares regression (PLS) is a method which reduces the variables, used to predict, to a smaller set of predictors. These predictors are then used to perfom a regression. in other words, PLS is the supervised version of PCR. First we need to determine the number of PLS components worth keeping.. The way to do so is by looking at the test root mean squared error (test RMSE) calculated by the k-fold cross-validation:

```
> summary(model)
Data:   X dimension: 664 15
  Y dimension: 664 1
Fit method: kernelpls
Number of components considered: 15

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
CV          0.4981   0.3608   0.3350   0.3275   0.3269   0.3268   0.3268   0.3269
adjCV       0.4981   0.3605   0.3342   0.3270   0.3264   0.3263   0.3264   0.3264
       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
CV      0.3269   0.3269    0.3269    0.3269    0.3269    0.3269    0.3269    0.3269
adjCV   0.3264   0.3264    0.3264    0.3264    0.3264    0.3264    0.3264    0.3264

TRAINING: % variance explained
```

```
17      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9
    comps
18 X      17.72    25.91    33.96    42.37    50.31    57.47    63.65    68.76
    74.06
19 V16    48.95    57.43    58.85    58.97    59.00    59.00    59.00    59.00
    59.00
20      10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
21 X       79.33     83.98     89.04     94.52       100     106.4
22 V16     59.00     59.00     59.00     59.00        59      59.0
```

Listing 23: Summary of the model

As we can see there are to tables in the output:

(a) VALIDATION: RMSEP

This table tells us the test RMSE calculated by the k-fold cross validation.

i. If we only use the intercept term in the model, the test RMSE is 0.4981.

ii. If we add 1 principal component, the test RMSE drops to 0.3605.

iii. If we add 2 principal components, the test RMSE drops to 0.3342.

We can see that adding additional PLS components actually leads to a decrease in test RMSE until the number of components is 5. Then it remains unchanged. Thus, it appears that it would be optimal to use more than or equal to 5 components as possible in the model.

(b) TRAINING: % variance explained This table tells us the percentage of the variance in the response variable explained by the PLS components.

i. If we only use the intercept term in the model, we can explain 17.72% of the variation in the response variable.

ii. If we add 1 principal component, we can explain 25.91% of the variation in the response variable.

iii. If we add 2 principal components, we can explain 33.96% of the variation in the response variable.

Note that we can explain more variance by using more PLS components.

The plot below is the visualization the test RMSE (along with the test MSE and R-squared) based on the number of PLS components by using the validationplot function.

As we can observe that the optimal model includes any number than is greater than 5 components. In this case, we choose 15. Now we run the obtained model on the testing data set and have the following test RMSE:

```
> sqrt(mean((pls_pred - test$V16)^2))#calculate RMSE
[1] 0.2997435
```

Listing 24: Test RMSE on the PLS model

We can see that the test RMSE turns out to be about 0.3. This is the average deviation between the predicted value for result and the observed value for result for the observations in the testing set.

4 Discussion and Comparison of the Models

First we see that the data set has multicollinearity, which occurs when two or more predictor variables in a dataset are highly correlated. Now based on the obtained methods and models, we can see that PLS and PCR models performs best among all the models, and the rest of the models are at the same level. By using the visualization techniques that we displayed above, we can see that prior(V9) has the most weight of all the data, followed by income(V15), then emp(V10) and then credit(V11).

5 Conclusion

In the process of credit card application, the most significant attributes that affect the approval of the application are prior(V9), income(V15), emp(V10) and credit(V11). The other variables in the data set has a relatively less significant influence in deciding the result of the application.

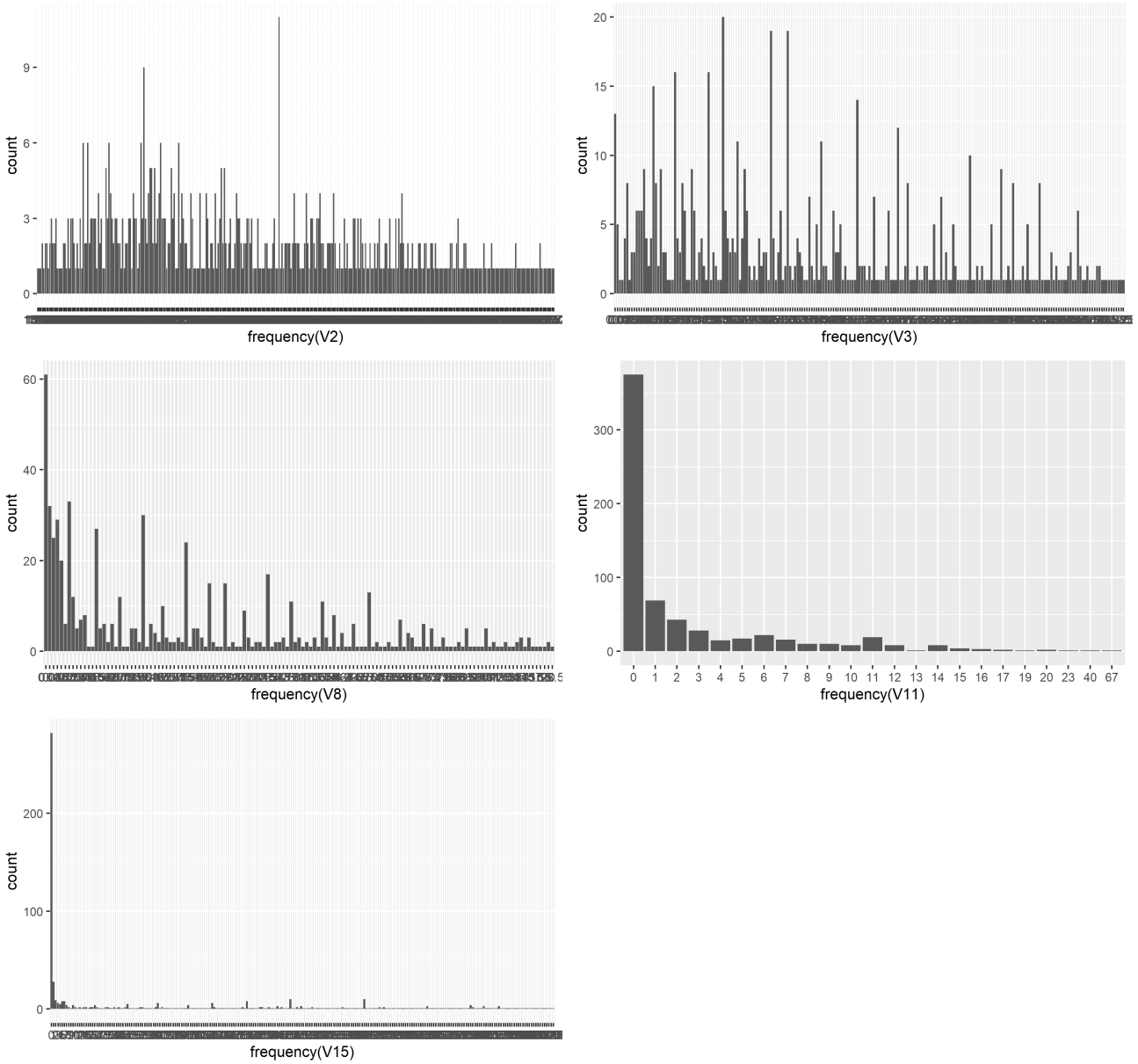

Figure 1: The frequency distribution tables for V2(Age), V3(Debt), V8(Years Employed), V11(Credit Scores), and V15(Income).

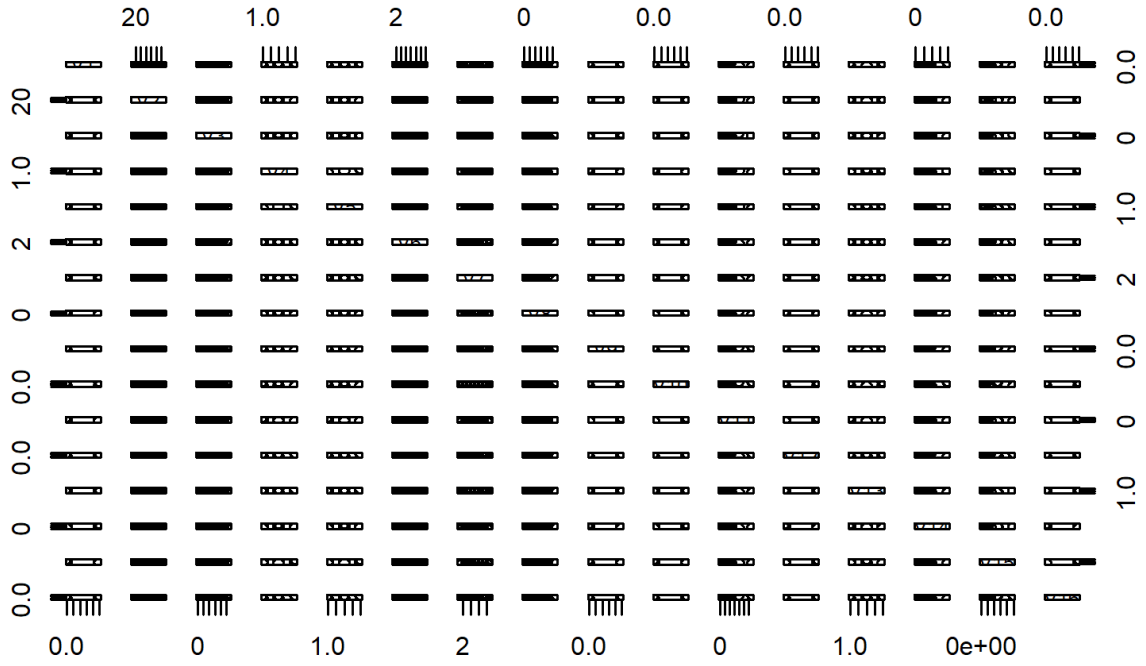

Figure 2: the Scatter plot matrix showing pairwise comparisons.

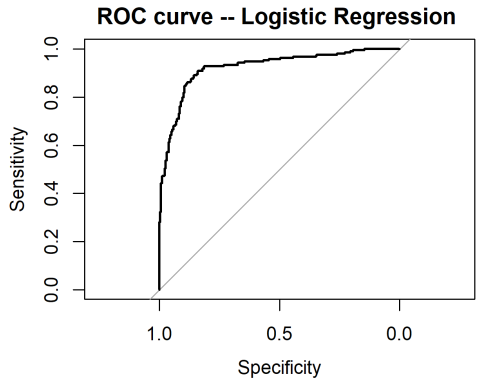

Figure 3: ROC curve for logistic model.

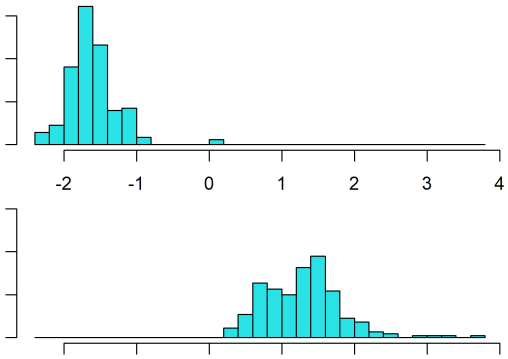

Figure 4: classification for the LDA training model.

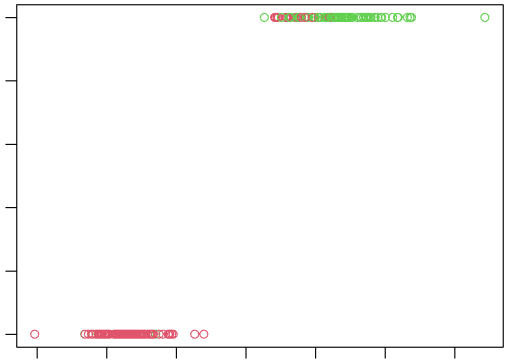

Figure 5: classification for the LDA testing model.

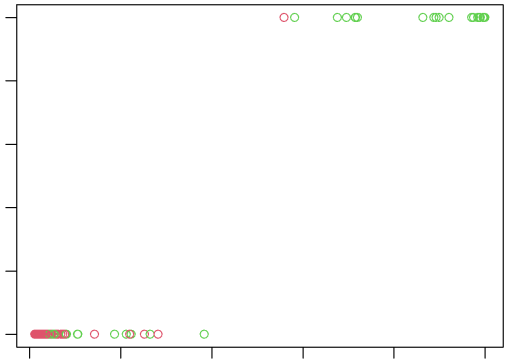

Figure 6: classification for the LDA testing model.

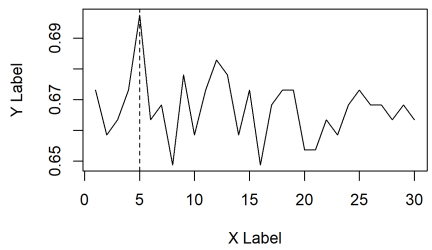

Figure 7: Visualization of the KNN assessment.

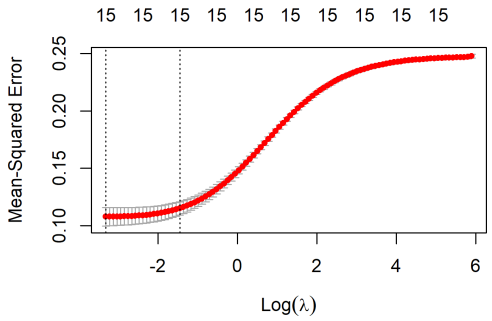

Figure 8: Plot of test MSE by lambda value.

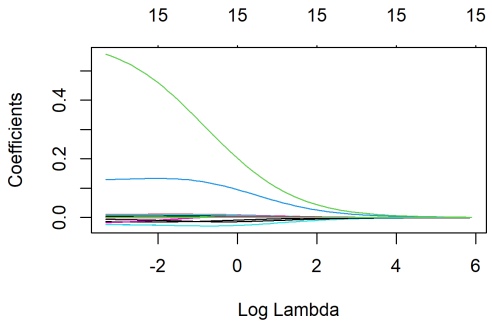

Figure 9: Trace plot to visualize how the coefficient estimates changed as a result of increasing lambda.

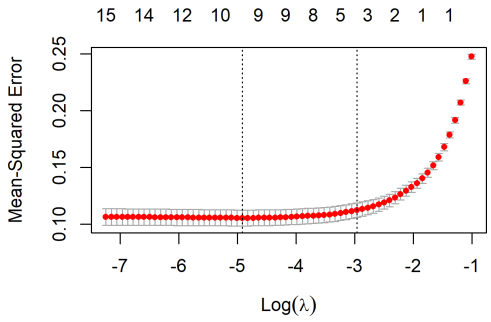

Figure 10: Plot of test MSE by lambda value.

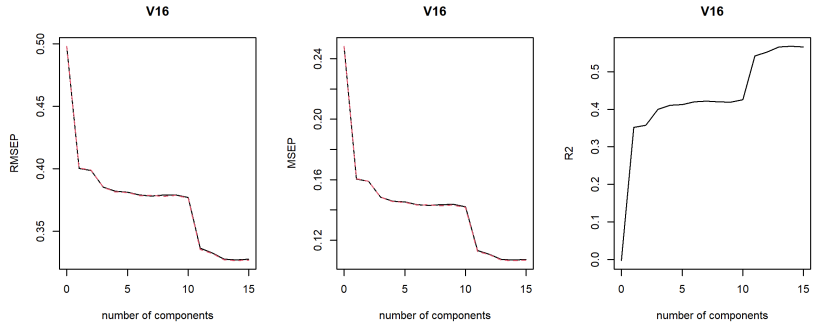

Figure 11: Plot of test RMSE, test MSE, R^2 based on the number of principal components.

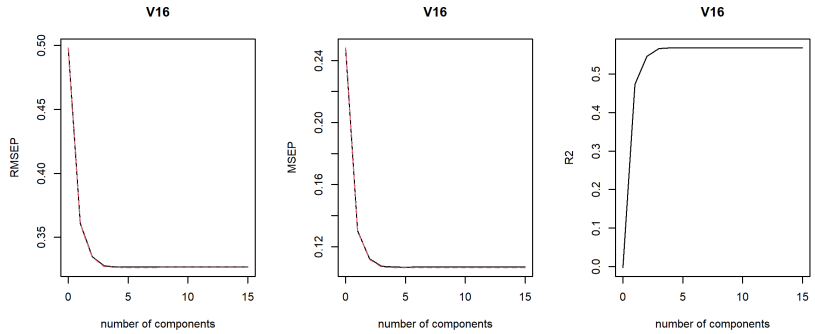

Figure 12: Plot of test RMSE, test MSE, R^2 based on the number of principal components.

Tables

Variables Description				
Var. Name	Type	Var. Discription	New name	Mod.
V1	character	Gender	gender	a,b → 1,0
V2	character	Age	age	NA
V3	number	Debt	debt	NA
V4	character	Marital Status	marital	u,y,l,t → 1,2,3,4
V5	character	Bank Customer	bankcus	g, p, gg → 1,2,3
V6	character	Education Level	edu	c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff → 1-14
V7	character	Ethnicity	ethnicity	v, h, bb, j, n, z, dd, ff, o → 1-9
V8	number	Years Employed	y_emp	NA
V9	character	Prior Default	prior	t, f → 1,0
V10	character	Employed	emp	t, f → 1,0
V11	integer	Credit Score	credit	NA
V12	character	Drivers License	driver	t, f → 1,0
V13	character	Citizen	citizen	g, p, s → 1,2,3
V14	character	Zip Code	zip	NA
V15	integer	Income	income	NA
V16	character	Approved	result	+,- → 1,0

R code

```r
rm(list = ls())

library(dplyr)
library(caret)
library("ggplot2")
library(tidyverse)

#read the data
rare_data <- read.table("4280data.data", fileEncoding = "UTF-8", sep = ",")

#checking the missing data, replace all "?" with NA
df <- rare_data %>%
  mutate_all(na_if, "?")
sapply(df, function(x) sum(is.na(x)))

#handle the missing data, replace NA in Age with mean
df$V2 = as.numeric(df$V2)
df$V2 = ifelse(is.na(df$V2),ave(df$V2, FUN = function (x)mean(x, na.rm = TRUE)),
        df$V2)

#remove the rest of the missing values
df<-na.omit(df)

#obtain each attribute from the set
gender <- df$V1
age <- df$V2;
debt <- df$V3;
marital <- df$V4;
bankcus <- df$V5;
edu <- df$V6;
ethnicity <- df$V7;
y_emp <- df$V8;
prior <- df$V9;
emp <- df$V10;
credit <- df$V11;
driver <- df$V12;
citizen <- df$V13;
zip <- df$V14;
income <- df$V15;
result = df$V16

```

```r
#normalize data
process <- preProcess(as.data.frame(age), method=c("range"))
df$V2 <- predict(process, as.data.frame(age))
process <- preProcess(as.data.frame(debt), method=c("range"))
df$V3 <- predict(process, as.data.frame(debt))
process <- preProcess(as.data.frame(y_emp), method=c("range"))
df$V8 <- predict(process, as.data.frame(y_emp))
process <- preProcess(as.data.frame(credit), method=c("range"))
df$V11 <- predict(process, as.data.frame(credit))
process <- preProcess(as.data.frame(income), method=c("range"))
df$V15 <- predict(process, as.data.frame(income))

#data visualization for continuous variables
frequency <- function(x) {
  factor(x, levels = names(table(x)))
}
ggplot(df, aes(x = frequency(`V2`))) + geom_bar()
ggplot(df, aes(x = frequency(`V3`))) + geom_bar()
ggplot(df, aes(x = frequency(`V8`))) + geom_bar()
ggplot(df, aes(x = frequency(`V11`))) + geom_bar()
ggplot(df, aes(x = frequency(`V15`))) + geom_bar()

#data visualization for discrete variables
df$V1<-ifelse(df$V1=="a",1,0)
df$V9<-ifelse(df$V9=="t",1,0)
df$V10<-ifelse(df$V10=="t",1,0)
df$V12<-ifelse(df$V12=="t",1,0)

df$V4 <- with(df, ifelse(V4 == "u", 1, ifelse(V4 == "y", 2, ifelse(V4 == "l", 3, ifelse(
    V4 == "6", 4, V4)))))
df$V5 <- with(df, ifelse(V5 == "g", 1, ifelse(V5 == "p", 2, ifelse(V5 == "gg", 3, V5))))
df$V6 <- with(df, ifelse(V6 == "c", 1, ifelse(V6 == "d", 2, ifelse(V6 == "cc", 3, ifelse(
    V6 == "i", 4, ifelse(V6 == "j", 5, ifelse(V6 == "k", 6, ifelse(V6 == "m", 7, V6))))))
    ))
df$V6 <- with(df, ifelse(V6 == "r", 8, ifelse(V6 == "q", 9, ifelse(V6 == "w", 10, ifelse(
    V6 == "x", 11, ifelse(V6 == "e", 12, ifelse(V6 == "aa", 13, ifelse(V6 == "ff", 14, V6
    ))))))))
df$V7 <- with(df, ifelse(V7 == "v", 1, ifelse(V7 == "h", 2, ifelse(V7 == "bb", 3, ifelse(
    V7 == "j", 4, ifelse(V7 == "n", 5, ifelse(V7 == "z", 6, ifelse(V7 == "dd", 7, ifelse(
    V7 == "ff", 8, ifelse(V7 == "o", 9, V7)))))))))
df$V13 <- with(df, ifelse(V13 == "g", 1, ifelse(V13 == "p", 2, ifelse(V13 == "s", 3, V13)
    )))
df$V16 <- with(df, ifelse(V16 == "+", 1, ifelse(V16 == "-", 0, V16)))
```

```
77
78 df$V4 <- as.numeric(df$V4)
79 df$V5 <- as.numeric(df$V5)
80 df$V6 <- as.numeric(df$V6)
81 df$V7 <- as.numeric(df$V7)
82 df$V13 <- as.numeric(df$V13)
83 df$V14 <- as.numeric(df$V14)
84 df$V16 <- as.numeric(df$V16)
85 str(df)
86 pairs(df)
87
88 #split the data into train and test set
89 sample <- sample(c(TRUE, FALSE), nrow(df), replace=TRUE, prob=c(0.7,0.3))
90 train <- df[sample, ]
91 test <- df[!sample, ]
92
93 #logistic
94 model <- glm(V16 ~ ., family="binomial"(link="logit"), data=train)
95 library(caret)
96 pdata <- predict(model, newdata = train, type = "response")
97 confusionMatrix(data = as.numeric(pdata>0.5), reference = train$V16)
98 summary(logitMod)
99 library(pscl)
100 pscl::pR2(logitMod)["McFadden"]
101 caret::varImp(logitMod)
102 library(pROC)
103 roc_score=roc(train$V16, pdata) #AUC score
104 plot(roc_score ,main ="ROC curve -- Logistic Regression ")
105
106 #LDA
107 set.seed(1)
108 row.number = sample(1:nrow(df), 0.6*nrow(df))
109 train = df[row.number,]
110 test = df[-row.number,]
111 dim(train)
112 dim(test)
113
114 library(MASS)
115 lda.model = lda (V16~., data=train)
116 lda.model
117 predmodel.train.lda = predict(lda.model, data=train)
118 confusionMatrix(data = as.factor(predmodel.train.lda$class), reference = as.factor(train$
    V16))
```

```
par(mar = c(1, 1, 1, 1))
ldahist(predmodel.train.lda$x[,1], g= predmodel.train.lda$class)

predmodel.test.lda = predict(lda.model, newdata=test)
confusionMatrix(data = as.factor(predmodel.test.lda$class), reference = as.factor(test$
    V16))
par(mfrow=c(1,1))
plot(predmodel.test.lda$x[,1], predmodel.test.lda$class, col=test$V16+10)

#QDA
qda.model = qda (V16~ V2 + V3 + V8 + V11 + V15, data=train)
qda.model
predmodel.train.qda = predict(qda.model, data=train)
confusionMatrix(data = as.factor(predmodel.train.qda$class), reference = as.factor(train$
    V16))

predmodel.test.qda = predict(qda.model, newdata=test)
confusionMatrix(data = as.factor(predmodel.test.qda$class), reference = as.factor(test$
    V16))
par(mfrow=c(1,1))
plot(predmodel.test.qda$posterior[,2], predmodel.test.qda$class, col=test$V16+10)

#KNN, K = 1~30
p <- c("V1","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12","V13", "V14","V15")
y <- "V16"
library(class)
r <- data.frame(array(NA, dim = c(0, 2), dimnames = list(NULL, c("k","accuracy"))))
for (k in 1:30) {
  set.seed(60402)
  predictions <- knn(train = train[,p],
                     test = test[,p],
                     cl = train[,y],
                     k = k)
  t <- table(pred = predictions, ref = test[,y])
  a <- sum(diag(t)) / sum(t)
  r <- rbind(r, data.frame(k = k, accuracy = a))
}
# find best k
r[which.max(r$accuracy),]
(k.best <- r[which.max(r$accuracy),"k"])
# plot
with(r, plot(k, accuracy, type = "l", xlab="X Label", ylab="Y Label"))
abline(v = k.best, lty = 2)
```

```
159
160 #ridge regression
161 y <- df$V16
162 x <- data.matrix(df[, c("V1","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12","
    V13", "V14","V15")])
163 library(glmnet)
164 model <- glmnet(x, y, alpha = 0)
165 summary(model)
166
167 cv_model <- cv.glmnet(x, y, alpha = 0)
168 best_lambda <- cv_model$lambda.min#find optimal lambda value that minimizes test MSE
169 best_lambda
170 plot(cv_model)
171
172 best_model <- glmnet(x, y, alpha = 0, lambda = best_lambda)
173 coef(best_model)
174
175 plot(model, xvar = "lambda")#produce Ridge trace plot
176
177 #find R-Squared
178 y_predicted <- predict(model, s = best_lambda, newx = x)
179 sst <- sum((y - mean(y))^2)
180 sse <- sum((y_predicted - y)^2)
181 rsq <- 1 - sse/sst
182 rsq
183
184 #Lasso model
185 y <- df$V16
186 x <- data.matrix(df[, c("V1","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11","V12","
    V13", "V14","V15")])
187 library(glmnet)
188 cv_model <- cv.glmnet(x, y, alpha = 1)
189 best_lambda <- cv_model$lambda.min#find optimal lambda value that minimizes test MSE
190 best_lambda
191
192 plot(cv_model) #produce plot of test MSE by lambda value
193
194 best_model <- glmnet(x, y, alpha = 1, lambda = best_lambda)#find coefficients of best
    model
195 coef(best_model)
196
197 y_predicted <- predict(best_model, s = best_lambda, newx = x)#find R-Squared
198 sst <- sum((y - mean(y))^2)
```

```r
sse <- sum((y_predicted - y)^2)
rsq <- 1 - sse/sst
rsq

#PCR
library(pls)
set.seed(1)
model <- pcr(V16~., data=df, scale=TRUE, validation="CV")#fit PCR model
summary(model)

par(mfrow=c(1,3))
validationplot(model)#visualize cross-validation plots
validationplot(model, val.type="MSEP")
validationplot(model, val.type="R2")

pcr_pred <- predict(model, test, ncomp=15)
sqrt(mean((pcr_pred - test$V16)^2))#calculate RMSE

#PLS model
set.seed(1)
model <- plsr(V16~., data=df, scale=TRUE, validation="CV")
summary(model)

par(mfrow=c(1,3))
validationplot(model)
validationplot(model, val.type="MSEP")
validationplot(model, val.type="R2")

pls_pred <- predict(model, test, ncomp=15)
sqrt(mean((pls_pred - test$V16)^2))#calculate RMSE
```

Listing 25: R code for this paper

References

1. Deepesh Khaneja, "Credit Approval Analysis using R", *Technical Report*, November 2017

2. Zach. "Ridge Regression in R (Step-by-Step)." *Statology*, 13 Nov. 2020, https://www.statology.org/ridge-regression-in-r/.