Predicting loan approval

Hanah Chang

```
library(glmnet)
## Loading required package: Matrix
## Loaded glmnet 3.0-2
library(MASS)
```

Introduction

In this project, we are going to find out which model produces the most accurate prediction in terms of deciding whether an individual is get approved for a loan. The dataset is from Kaggle and includes variables such as

- Amount.Requested: The proposed amount for the loan
- Debt.To.Income.Ratio: The ratio of the applicant's debt payments each month to the applicant's stated monthly income
- Zip.Code: The 3-digit zip code of the applicant
- State: The state where the applicant lives
- Employment.Length: The number of years that the applicant has worked at the same job. 10 indicates at least ten years, 0 indicates less than one year, and -1 indicates unemployed.
- y: A binary variable indicating whether the loan was approved

1. Initial Model: GLM, LDA, QDA, glment

I chose three variables (Amount.Requested, Employment.Length, Debt.To.Income.Ratio) to run GLM, LDA, QDA, glment model.

```
training <-dataset[1:5000, ]
testing <- dataset[-c(1:5000), ]
```

(Generalized linear model) The GLM correctly predicted that the loan would be approved 18 times, and that it would be disapproved for 4,299 times. In this case, the logistic regression correctly predicted the approval of the loan 86.34% of the time.

```
logit <- glm(y ~ Amount.Requested +
               Employment.Length + Debt.To.Income.Ratio, data = training, family = binomial(link = "log
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

y_hat_logit <- fitted(logit)
summary(y_hat_logit)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.03168 0.03778 0.08920 0.04282 0.69162
```

```r
z_logit <- as.integer(y_hat_logit > 0.5)
table(testing$y, z_logit)
##    z_logit
##        0    1
##   0 4299  285
##   1  398   18

mean((testing$y) == z_logit)
## [1] 0.8634
```

(Linear Discriminant Analysis) The LDA correctly predicted that the loan would be approved 35 times, and that it would be disapproved for 4,194 times. In this case, the LDA correctly predicted the approval of the loan 84.58% of the time.

```r
LDA <- lda(y ~ Amount.Requested + Employment.Length+ Debt.To.Income.Ratio, data = training)
y_hat_LDA <- predict(LDA)
summary(y_hat_LDA$posterior)
##        0                 1
##  Min.   :0.02495   Min.   :0.002573
##  1st Qu.:0.99342   1st Qu.:0.006191
##  Median :0.99374   Median :0.006263
##  Mean   :0.89765   Mean   :0.102347
##  3rd Qu.:0.99381   3rd Qu.:0.006582
##  Max.   :0.99743   Max.   :0.975054

z_LDA <- y_hat_LDA$class
table(testing$y, z_LDA)
##    z_LDA
##        0    1
##   0 4194  390
##   1  381   35

mean((testing$y) == z_LDA)
## [1] 0.8458
```

(Quadratic Discriminant Analysis)

It looks like the QDA predictions does not capture the true relationship between variables

```r
QDA <- qda(y ~ Amount.Requested + Employment.Length +Debt.To.Income.Ratio , data = training)
y_hat_QDA <- predict(QDA)
summary(y_hat_QDA$posterior)
##        0                   1
##  Min.   :0.0000096   Min.   :0.0000
##  1st Qu.:0.1563475   1st Qu.:0.3205
##  Median :0.2834514   Median :0.7165
##  Mean   :0.4077262   Mean   :0.5923
##  3rd Qu.:0.6794894   3rd Qu.:0.8437
##  Max.   :1.0000000   Max.   :1.0000

z_QDA <- y_hat_QDA$class
table(testing$y, z_QDA)
```

```
##    z_QDA
##        0    1
##   0 1600 2984
##   1  144  272

mean((testing$y) == z_QDA)
## [1] 0.3744
```

(glmnet)

The result of the glmnet function is 91.68% accuracy. The penalization of the fit in the testing data improved the classification accuracy in the testing data.

```
x <- model.matrix(logit)
y <- testing$y

path2 <- glmnet(x[,-1], y, family = "binomial")
path2
##
## Call:  glmnet(x = x[, -1], y = y, family = "binomial")
##
##    Df      %Dev    Lambda
## 1   0 0.000e+00 0.0033820
## 2   1 4.459e-05 0.0030810
## 3   1 8.188e-05 0.0028080
## 4   2 1.301e-04 0.0025580
## 5   3 2.030e-04 0.0023310
## 6   3 2.696e-04 0.0021240
## 7   3 3.283e-04 0.0019350
## 8   3 3.803e-04 0.0017630
## 9   3 4.268e-04 0.0016070
## 10  3 4.688e-04 0.0014640
## 11  3 5.071e-04 0.0013340
## 12  3 5.425e-04 0.0012150
## 13  3 5.757e-04 0.0011070
## 14  3 6.072e-04 0.0010090
## 15  3 6.376e-04 0.0009194
## 16  3 6.673e-04 0.0008377
## 17  3 6.968e-04 0.0007633
## 18  3 7.260e-04 0.0006955
## 19  3 7.551e-04 0.0006337
## 20  3 7.837e-04 0.0005774
## 21  3 8.114e-04 0.0005261
## 22  3 8.377e-04 0.0004794
## 23  3 8.624e-04 0.0004368
## 24  3 8.854e-04 0.0003980
## 25  3 9.066e-04 0.0003626
## 26  3 9.261e-04 0.0003304
## 27  3 9.439e-04 0.0003011
## 28  3 9.602e-04 0.0002743
## 29  3 9.751e-04 0.0002499
## 30  3 9.886e-04 0.0002277
## 31  3 1.001e-03 0.0002075
## 32  3 1.012e-03 0.0001891
```

```
## 33  3 1.022e-03 0.0001723
## 34  3 1.031e-03 0.0001570

y_hat_path2 <- predict(path2, newx = x[,-1], type = "response")
z_path2<- y_hat_path2 > 0.5
s <- which.max(colSums(apply(z_path2, MARGIN = 2, FUN = `==`, e2 = y)))
table(y, as.integer(z_path2[,s]))
##
## y      0
##   0 4584
##   1  416

mean((testing$y) == as.integer(z_path2[,s]))
## [1] 0.9168
```

2. Extended Model: Ramdpmforest, Bartmachine

(Randomforest)

Randomforest accurately predicts 93.54% of the testing data.

```
stopifnot(require(randomForest))
## Loading required package: randomForest
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.

bagged <- randomForest(y ~ Amount.Requested + Employment.Length  + Debt.To.Income.Ratio, data = trainin
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range

bagged
##
## Call:
##  randomForest(formula = y ~ Amount.Requested + Employment.Length +      Debt.To.Income.Ratio, data =
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.04915343
##                     % Var explained: 39.5
```

```
yhat_bagged <- predict(bagged, newdata = testing, type = "class")
correct_bg <- mean((testing$y == 1) == (yhat_bagged > 0.5))
z_bg <- as.integer(yhat_bagged > 0.5)
table(testing$y, z_bg)
##    z_bg
##        0    1
##   0 4423  161
##   1  162  254
```

(bartmachine)

for bartmachine, I'm going to use all variables. let's take a look.

```
options( java.parameters = "-Xmx4g" )
stopifnot(require(bartMachine))
## Loading required package: bartMachine
## Loading required package: rJava
## Loading required package: bartMachineJARs
## Loading required package: car
## Loading required package: carData
## Loading required package: missForest
## Loading required package: foreach
## Loading required package: itertools
## Loading required package: iterators
## Welcome to bartMachine v1.2.3! You have 3.82GB memory available.
##
## If you run out of memory, restart R, and use e.g.
## 'options(java.parameters = "-Xmx5g")' for 5GB of RAM before you call
## 'library(bartMachine)'.
set_bart_machine_num_cores(parallel::detectCores())
## bartMachine now using 8 cores.

bart <- bartMachine(X = training[, c("Amount.Requested", "Employment.Length",  "Debt.To.Income.Ratio")]
## bartMachine initializing with 50 trees...
## bartMachine vars checked...
## bartMachine java init...
## bartMachine factors created...
## bartMachine before preprocess...
## bartMachine after preprocess... 4 total features...
## bartMachine sigsq estimated...
## bartMachine training data finalized...
## Now building bartMachine for regression ...
## evaluating in sample data...done

bart
## bartMachine v1.2.3 for regression
##
## training data n = 5000 and p = 3
## built in 14.5 secs on 8 cores, 50 trees, 250 burn-in and 1000 post. samples
##
## sigsq est for y beforehand: 0.06
## avg sigsq estimate after burn-in: 0.03789
##
## in-sample statistics:
##  L1 = 423.22
##  L2 = 182.98
##  rmse = 0.19
##  Pseudo-Rsq = 0.5496
## p-val for shapiro-wilk test of normality of residuals: 0
## p-val for zero-mean noise: 0.96415
```

```
yhat_bart <- predict(bart, new_data = testing[, c("Amount.Requested", "Employment.Length",  "Debt.To.In
correct_bt <- mean((testing$y == 1) == (yhat_bart > 0.5))
```

```r
z_bt <- as.integer(yhat_bart > 0.5)
table(testing$y, z_bt)
##    z_bt
##        0    1
##   0 4420  164
##   1  150  266
```

Conclusion

In this project, Bartmachine showed the highest proportion of correct predictions, followed closely by randomforest.