homework4_isye6501

Zach Olivier

6/9/2018

Question 9.1

Question:

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2.

Answer:

Here are my steps to apply PCA to the crime dataset covered in the last homework. First we read in the data and transform it using the caret preProcess function. This function allows you to apply the centering, scaling, and principal component analysis all in one function call. After applying these steps I can extract the rotation of the preProcess object to see the linear combination of each prediction that makes up each PCA.

Now that we have our clean and tidy component dataset, we can begin the model fitting process. First I split our dataset into test and training dataset (note we have very few observations) using a split of 75% train and 25% test.

With the training and test sets ready, I use caret's train function to fit a simple linear regression on crime versus the first five principal components. Utilizing cross validation the model achieves ~71% R^2 on the training data.

Using this model to predict onto the test set - our model achieves an R^2 value of ~21%. It seems using PCA on such small data has led us to severly overfit our data

Finally we use this model to predict the same 'new state' provided in the previous homework. The PCA linear regression crime prediction is 1419.68

I noticed a sizeable difference between the linear regression and and PCA linear regression (around 750 crime prediction vs. 1030 crime prediction). My simple linear regression from the previous homework tried to manually exclude variables based on correlation, and I only fine tuned the model with a training and a test set. The applying PCA to such a small dataset caused our model to severly overfit the data. I would recomend going with the standard linear regression model in this case with so few data points.

Here is the final model represented in the form of the original unscaled predictors:

Unscaled Intercept and Coefficient Estimates:

```
    - Intercept -8217.3918
    - M          86.4716149
    - So        114.1757429
    - Ed         12.6174416
    - Po1        38.6866613
    - Po2        38.1541277
    - LF       2911.6391584
    - M.F        50.1266387
    - Pop         1.4129029
    - NW         12.9416492
    - U1       -666.5257377
    - U2         16.9883283
    - Wealth      0.0174338
    - Ineq       13.5104521
    - Prob    -1889.0022929
    - Time        6.4230641
```

```r
set.seed(110)

# read in the crime data
crime_df = read_delim('9.1uscrimeSummer2018.txt', delim = '\t') %>%
        as.data.frame()



## eigenvalue / eigenvector lesson for reference
# crime_mat <- as.matrix(crime_df)

# x_t_x <- t(crime_mat)%*%crime_mat
# ev <- eigen(x_t_x)
#ev$vectors

# for (i in 1:ncol(crime_df)) {
#         print(det(x_t_x - ev$values[i]*diag(ncol(crime_df))))
# }

## solve this system of equations to get unique eigenvalues
## does not have to be true to calculate eigenvalues in PCA
## determinant does not have to evaluate to 0
# (x_t_x) * v = lambda * v
# [x_t_x - lambda * I] * v = 0
# det[x_t_x - lambda * I] = 0





# using cool caret functions to transform predictor data - including box cox transf
ormation
transform_df = caret::preProcess(
        crime_df %>% dplyr::select(., -Crime),
        method = c('center', 'scale', 'nzv', 'pca')
        )

# look at the linear combination of the predictors for each principal component
transform_df$rotation
```

```
##                PC1         PC2           PC3         PC4         PC5
## M      -0.30371194  0.06280357  0.1724199946 -0.02035537 -0.35832737
## So     -0.33088129 -0.15837219  0.0155433104  0.29247181 -0.12061130
## Ed      0.33962148  0.21461152  0.0677396249  0.07974375 -0.02442839
## Po1     0.30863412 -0.26981761  0.0506458161  0.33325059 -0.23527680
## Po2     0.31099285 -0.26396300  0.0530651173  0.35192809 -0.20473383
## LF      0.17617757  0.31943042  0.2715301768 -0.14326529 -0.39407588
## M.F     0.11638221  0.39434428 -0.2031621598  0.01048029 -0.57877443
## Pop     0.11307836 -0.46723456  0.0770210971 -0.03210513 -0.08317034
## NW     -0.29358647 -0.22801119  0.0788156621  0.23925971 -0.36079387
## U1      0.04050137  0.00807439 -0.6590290980 -0.18279096 -0.13136873
## U2      0.01812228 -0.27971336 -0.5785006293 -0.06889312 -0.13499487
## Wealth  0.37970331 -0.07718862  0.0100647664  0.11781752  0.01167683
## Ineq   -0.36579778 -0.02752240 -0.0002944563 -0.08066612 -0.21672823
## Prob   -0.25888661  0.15831708 -0.1176726436  0.49303389  0.16562829
## Time   -0.02062867 -0.38014836  0.2235664632 -0.54059002 -0.14764767
##                 PC6         PC7         PC8         PC9
## M      -0.449132706 -0.15707378 -0.55367691  0.15474793
## So     -0.100500743  0.19649727  0.22734157 -0.65599872
## Ed     -0.008571367 -0.23943629 -0.14644678 -0.44326978
## Po1    -0.095776709  0.08011735  0.04613156  0.19425472
## Po2    -0.119524780  0.09518288  0.03168720  0.19512072
## LF      0.504234275 -0.15931612  0.25513777  0.14393498
## M.F    -0.074501901  0.15548197 -0.05507254 -0.24378252
## Pop     0.547098563  0.09046187 -0.59078221 -0.20244830
## NW      0.051219538 -0.31154195  0.20432828  0.18984178
## U1      0.017385981 -0.17354115 -0.20206312  0.02069349
## U2      0.048155286 -0.07526787  0.24369650  0.05576010
## Wealth -0.154683104 -0.14859424  0.08630649 -0.23196695
## Ineq    0.272027031  0.37483032  0.07184018 -0.02494384
## Prob    0.283535996 -0.56159383 -0.08598908 -0.05306898
## Time   -0.148203050 -0.44199877  0.19507812 -0.23551363
```

```r
# apply processing steps to data - select only the first 5 PC
crime_mod_df <- predict(transform_df, crime_df) %>%
        dplyr::select(Crime, PC1:PC5)



# set up train and testing split
train <- createDataPartition(crime_mod_df$Crime, p = .75, list = F)

# set up test and train datasets
crime_train <- crime_mod_df[train,]
crime_test <- crime_mod_df[-train,]

# check splits
dim(crime_train); dim(crime_test)
```

```
## [1] 36  6
```

```
## [1] 11  6
```

```r
# fit model
crime_fit <- train(
        Crime ~ .,
        data = crime_train,
        method = 'lm',
        trControl = trainControl(method = 'cv')
        )

summary(crime_fit)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -362.80 -102.81  -10.94  129.33  402.63
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   914.81      39.34  23.251  < 2e-16 ***
## PC1            39.38      16.51   2.385  0.02362 *
## PC2           -65.81      22.41  -2.937  0.00631 **
## PC3            30.43      27.40   1.111  0.27544
## PC4            64.21      33.93   1.892  0.06814 .
## PC5          -275.22      39.11  -7.037 8.02e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 227.7 on 30 degrees of freedom
## Multiple R-squared:  0.7174, Adjusted R-squared:  0.6702
## F-statistic: 15.23 on 5 and 30 DF,  p-value: 1.823e-07
```

```r
# check performance on validation set
crime_eval <- crime_test %>%
        add_predictions(., crime_fit) %>%
        dplyr::select('obs' = Crime, pred) %>%
        as.data.frame()

# test set performance metrics
postResample(obs = crime_eval$obs, pred = crime_eval$pred)
```

```
##        RMSE    Rsquared         MAE
## 326.9742036   0.2128949 259.3209065
```

```
# input state example from previous homework
new_state = data.frame(
        M = 14.0,
        So = 0,
        Ed = 10.0,
        Po1 = 12.0,
        Po2 = 15.5,
        LF = 0.640,
        M.F = 94.0 ,
        Pop = 150,
        NW = 1.1,
        U1 = 0.120,
        U2 = 3.6 ,
        Wealth = 3200,
        Ineq = 20.1 ,
        Prob = 0.04 ,
        Time = 39.0
)

# transform new_state data to feed it into our model based on PCs
new_state_transform = predict(transform_df, new_state)

# crime prediction for new state based on PCA model
crime_pred = predict(crime_fit, new_state_transform) %>%
        as_tibble()

print(paste('New State Crime Prediction: ', crime_pred$value))
```

```
## [1] "New State Crime Prediction:  1419.68337979051"
```

```
# format output to support unscaling calculations
crime_coef <-  summary(crime_fit)$coef %>% as.data.frame() %>% rownames_to_column() 
%>%
        filter(rowname != '(Intercept)') %>%
        dplyr::select(2) %>%
        as.matrix() %>%
        t()

crime_pca <- t(transform_df$rotation[,1:5]) %>% as.matrix()

unscale_x <- crime_coef %*% crime_pca

crime_int <- summary(crime_fit)$coef %>% as.data.frame() %>% rownames_to_column() %
>%
        filter(rowname == '(Intercept)') %>%
        dplyr::select(2)



# get the unscaled coefficients and intercept
intercept <-  crime_int$Estimate - sum(
        unscale_x * sapply(crime_df[,1:15], mean) / sapply(crime_df[,1:15],sd)
        )

print(paste('Unscaled Intercept and Coefficient Estimates: ', intercept))
```

```
## [1] "Unscaled Intercept and Coefficient Estimates:  -7304.43349884118"
```

```
(coefs <- t(unscale_x / sapply(crime_df[,1:15], sd)))
```

```
##             Estimate
## M       6.880245e+01
## So      1.040597e+02
## Ed      1.175942e+01
## Po1     3.957145e+01
## Po2     3.940328e+01
## LF      2.312172e+03
## M.F     4.493568e+01
## Pop     1.533245e+00
## NW      1.171904e+01
## U1      3.008728e+02
## U2      4.054743e+01
## Wealth  2.558634e-02
## Ineq    1.049412e+01
## Prob   -1.676499e+03
## Time    5.211152e+00
```

Question 10.1

Question:

Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R,you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don’t just stop when you have a good model, but interpret it too).

Answer:

Below are my attempts to fit a regression tree and a regression random forest on the crime dataset. The first thing I noticed is that this dataset is very small (only 47 total observations) - it will be very hard for either model to produce many splits of the data.

To start we read in the data and set up the test and training splits. Tree based models are predictor scale and magnitude agnostic, so I did not apply at scaling or transformations beforehand. I also choose a relatively high proportion to split the data in order to give each model the most amount of data possible.

To combat the small amount of training data, I utilize a bootstrap method with 1000 bootstrap samples instead of a typical cross validation method. My thinking is that bootstrapping will give a better model than folding the training set k times. In this case our bootstrap will randomly sample 36 observations of the training set with replacement 1000 times.

The results of each model are as follows:

```
    - Regression Tree:  test set RMSE = 558.9631
    - Random Forest: test set RMSE = 515.75088075
```

Note: with such a small test set these values can be misleading. I would expect the true RMSE to be closer to the bootstrapped RMSE taken from the training data. To remedy this situation we need more data.

Model Observations:

```
    - The regression tree splits past the terminal node on Pol > 7.8 and N < 7.65
    - This is a relatively small tree most likely limited by the size of the data
    - Random Forests are hard to interpret (multiple regression trees a built in pa
rallel with random features)
    - Noticed that random forest created 500 different trees - mostly likely all ve
ry similiar due to data constaints
```

```r
set.seed(45)

# read in the crime data
crime_df = read_delim('9.1uscrimeSummer2018.txt', delim = '\t') %>%
        as.data.frame()

# apply processing steps to data
crime_mod_df <- crime_df

# set up train and testing split
train <- createDataPartition(crime_mod_df$Crime, p = .85, list = F)

# set up test and train datasets
crime_train <- crime_mod_df[train,]
crime_test <- crime_mod_df[-train,]

# check splits
dim(crime_train); dim(crime_test)
```

```
## [1] 43 16
```

```
## [1]  4 16
```

```r
# fit regression tree model
(crime_tree_fit <- train(
        Crime ~ .,
        data = crime_train,
        method = 'rpart',
        trControl = trainControl(method = 'boot_all', number = 10),
        metric = 'RMSE'
        ))
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in the apparent performance
## measures.
```

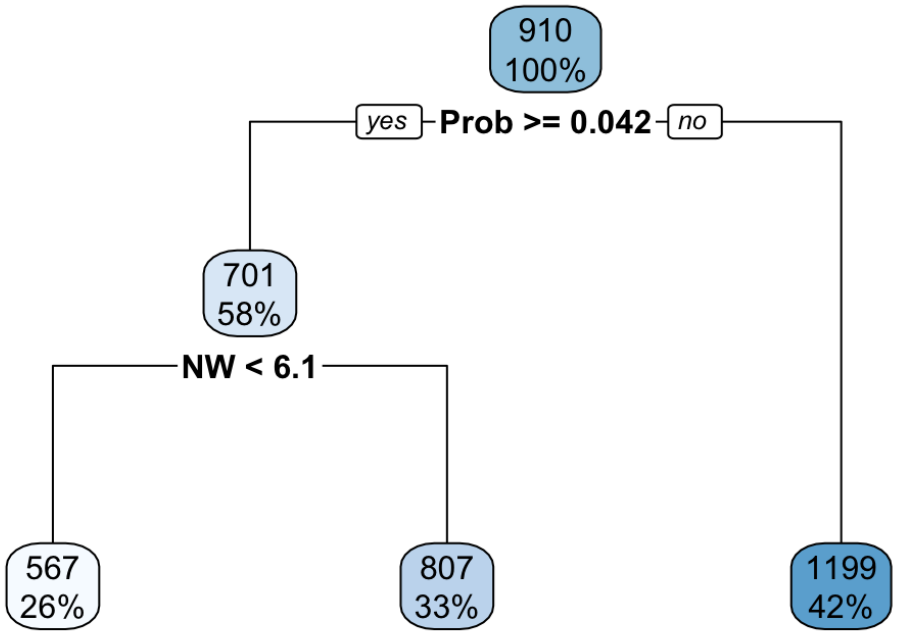
```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
## CART
##
## 43 samples
## 15 predictors
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 43, 43, 43, 43, 43, 43, ...
## Resampling results across tuning parameters:
##
##   cp          RMSE      Rsquared   MAE       RMSE_632  Rsquared_632
##   0.00000000  346.0381  0.3080665  268.7309  318.2400  0.3725179
##   0.05851138  346.3912  0.3071555  270.9927  323.9456  0.3504169
##   0.42475215  370.7773  0.2934444  285.8957  372.7957         NA
##   MAE_632   RMSE_OptBoot  Rsquared_OptBoot  MAE_OptBoot
##   237.4961  334.5677      0.3288689         228.2725
##   251.2199  349.1902      0.2693252         262.2148
##   284.8724  425.1610             NA         317.1475
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.
```

```
crime_tree_fit$finalModel
```

```
## n= 43
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 43 6087701.0  909.5581
##   2) Prob>=0.0418485 25  860010.2  701.4800
##     4) NW< 6.05 11  168119.6  566.8182 *
##     5) NW>=6.05 14  335690.9  807.2857 *
##   3) Prob< 0.0418485 18 2641926.0 1198.5560 *
```

```
library(rpart.plot)

# view final decision tree
rpart.plot::rpart.plot(crime_tree_fit$finalModel)
```



```r
# check performance on validation set
crime_eval <- crime_test %>%
        add_predictions(., crime_tree_fit) %>%
        dplyr::select('obs' = Crime, pred) %>%
        as.data.frame()

# test set performance metrics
postResample(obs = crime_eval$obs, pred = crime_eval$pred)
```

```
##      RMSE Rsquared      MAE
## 558.9631       NA 519.7778
```

```r
# fit regression random forest tree model
(crime_forest_fit <- train(
        Crime ~ .,
        data = crime_train,
        method = 'rf',
        trControl = trainControl(method = 'boot_all', number = 10),
        metric = 'RMSE'
        ))
```

```
## Random Forest
##
## 43 samples
## 15 predictors
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 43, 43, 43, 43, 43, 43, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE       RMSE_632  Rsquared_632  MAE_632
##    2    260.8183  0.5987735  194.9023  208.5593  0.7301108     156.1072
##    8    256.1590  0.5721235  181.7228  201.3337  0.7119081     142.2421
##   15    257.8345  0.5526890  182.2714  201.4774  0.7000094     142.1698
##   RMSE_OptBoot  Rsquared_OptBoot  MAE_OptBoot
##   213.9921      0.8028898         153.0989
##   206.8930      0.7928452         135.9704
##   205.3191      0.7892572         135.3101
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 8.
```

```
crime_forest_fit$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 68861.88
##                     % Var explained: 51.36
```

```
# check performance on validation set
crime_eval <- crime_test %>%
        add_predictions(., crime_forest_fit) %>%
        dplyr::select('obs' = Crime, pred) %>%
        as.data.frame()

# test set performance metrics
postResample(obs = crime_eval$obs, pred = crime_eval$pred)
```

```
##        RMSE    Rsquared         MAE
## 498.4150986   0.0523848 447.4705417
```

Question 10.2

Question:

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

Answer:

Logistic Regression can be very useful for many business analytic problems. Being able to assign and interpret probabilities of some outcome can solve the need for prediction and inference. In my own experience, I used logistic regression to determine which benefits are most revelant to a customer segmentation we wanted to shift our product into. Using sales data I treated the response as a binary - did our target customer make this purchase - and feed in the product features as the predictors. This allowed us to assign importance to difference features and also have a indication of the impact adding these features to other products would have on our target buyer.

Here are some predictors I used:

```
    - Price
    - Discount Level
    - Package purchased
    - Upgrades purchased
    - Purchase profile of previous products
```

Question 10.3

Question:

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.
2. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

Answer:

Here are my steps to fit a logistic regression model to the German Credit data. As always we read in the data and create our experiment design splits to prepare for modeling. I utilize the caret package again for a clean output and efficiencies in setting up the data.

By specifying the method as 'glm' and the family as 'binomial' in the train function - caret will know to transform the response variable into a binary outcome. I will utilize 10-fold cross validation and also instruct the train function to keep the class probabilities the logistic model generates.

Here are the results of the model on the test dataset. Coefficients are included in the output below.

```
    - Accuracy = 0.7267; Sensitivity = 0.5488; Specificity : 0.7936
    - This model correctly classified 79% of good customers correctly (negative cas
e is good)
```

If cost is the only thing we care about - we can push the probability parameter to only give a 'good' rating to customers with higher than 95% probability. This allows us to only predict 3 customers as good when they are really bad. However, values for revenue are not given, our threshold might change as we are more strictly classifying good customers as bad - this would be opportunity loss for our firm.

```
set.seed(9)

data("GermanCredit")

# read in the crime data
credit_df <- GermanCredit %>% as.data.frame()

# training and test data sets
partition <- createDataPartition(credit_df$Class, p = .8, list = F)

# training and test data sets
train_credit <- credit_df[partition,]
test_credit <- credit_df[-partition,]

dim(train_credit); dim(test_credit)
```

```
## [1] 800  62
```

```
## [1] 200  62
```

```
# fit model with cross validation - basic logistic regression
(credit_mod_glm <- train(
        Class ~ .,
        data = train_credit,
        method = 'glm',
        family = 'binomial',
        trControl = trainControl(method = 'cv', classProbs = T),
        metric = 'Accuracy'
        ))
```

```
## Generalized Linear Model
##
## 800 samples
##  61 predictor
##   2 classes: 'Bad', 'Good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 720, 720, 720, 720, 720, 720, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.76125   0.4047356
```

```
coef(credit_mod_glm$finalModel)
```

```
##                          (Intercept)
##                         9.5986272174
##                             Duration
##                        -0.0201031481
##                               Amount
##                        -0.0001864236
##            InstallmentRatePercentage
##                        -0.4553392358
##                    ResidenceDuration
##                        -0.0327971497
##                                  Age
##                         0.0204236364
##                NumberExistingCredits
##                        -0.1911303739
##              NumberPeopleMaintenance
##                        -0.3736222433
##                            Telephone
##                        -0.3661301200
##                        ForeignWorker
##                        -2.5611379195
##           CheckingAccountStatus.lt.0
##                        -1.5656013164
##       CheckingAccountStatus.0.to.200
##                        -1.0601688709
##         CheckingAccountStatus.gt.200
##                        -0.4076078560
##           CheckingAccountStatus.none
##                                   NA
##       CreditHistory.NoCredit.AllPaid
##                        -1.3832693536
##       CreditHistory.ThisBank.AllPaid
##                        -1.3228270054
##              CreditHistory.PaidDuly
##                        -0.8257734035
##                 CreditHistory.Delay
##                        -0.8626171608
##              CreditHistory.Critical
##                                   NA
##                     Purpose.NewCar
##                          -1.4519698719
##                        Purpose.UsedCar
##                           0.3410287428
##            Purpose.Furniture.Equipment
##                          -0.3710752157
##               Purpose.Radio.Television
##                          -0.4358780734
##              Purpose.DomesticAppliance
##                          -1.0312962943
##                        Purpose.Repairs
##                          -1.0898541987
##                      Purpose.Education
##                          -1.4666589813
##                       Purpose.Vacation
##                                     NA
##                     Purpose.Retraining
##                           0.8583717441
##                       Purpose.Business
##                          -0.6865988648
##                          Purpose.Other
##                                     NA
##             SavingsAccountBonds.lt.100
##                          -1.1688009402
##         SavingsAccountBonds.100.to.500
##                          -0.8230813521
##        SavingsAccountBonds.500.to.1000
##                          -1.0771013497
##            SavingsAccountBonds.gt.1000
##                          -0.3998324399
##            SavingsAccountBonds.Unknown
##                                     NA
##                EmploymentDuration.lt.1
##                           0.0300158388
##              EmploymentDuration.1.to.4
##                           0.3154963944
##              EmploymentDuration.4.to.7
##                           0.9677103746
##                EmploymentDuration.gt.7
##                           0.3255539427
##         EmploymentDuration.Unemployed
##                                     NA
##        Personal.Male.Divorced.Seperated
##                          -0.2721415375
##              Personal.Female.NotSingle
##                           0.1507802891
##                   Personal.Male.Single
##                           0.7887707344
##          Personal.Male.Married.Widowed
##                                     NA
##                 Personal.Female.Single
##                                     NA
##            OtherDebtorsGuarantors.None
##                          -0.9647546652
##     OtherDebtorsGuarantors.CoApplicant
##                          -1.6916033431
##       OtherDebtorsGuarantors.Guarantor
##                                     NA
##                    Property.RealEstate
##                           0.7040225703
##                     Property.Insurance
##                           0.1436394616
##                      Property.CarOther
##                           0.1308177089
##                       Property.Unknown
##                                     NA
##             OtherInstallmentPlans.Bank
##                          -0.7201887371
##           OtherInstallmentPlans.Stores
##                          -0.6271818062
##             OtherInstallmentPlans.None
##                                     NA
##                           Housing.Rent
##                          -0.5935887960
##                            Housing.Own
##                          -0.0028297254
##                        Housing.ForFree
##                                     NA
##                Job.UnemployedUnskilled
##                           0.0054078796
##                  Job.UnskilledResident
##                          -0.0908905142
##                    Job.SkilledEmployee
##                          -0.1213852828
## Job.Management.SelfEmp.HighlyQualified
##                                     NA
```

```
# predict back onto test data set
credit_pred_glm <- test_credit %>% add_predictions(credit_mod_glm) %>%
        dplyr::select(Class, pred) %>%
        as.data.frame()

confusionMatrix(credit_pred_glm$Class, credit_pred_glm$pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   20   40
##       Good  12  128
##
##                Accuracy : 0.74
##                  95% CI : (0.6734, 0.7993)
##     No Information Rate : 0.84
##     P-Value [Acc > NIR] : 0.999895
##
##                   Kappa : 0.2857
##  Mcnemar's Test P-Value : 0.000181
##
##             Sensitivity : 0.6250
##             Specificity : 0.7619
##          Pos Pred Value : 0.3333
##          Neg Pred Value : 0.9143
##              Prevalence : 0.1600
##          Detection Rate : 0.1000
##    Detection Prevalence : 0.3000
##       Balanced Accuracy : 0.6935
##
##        'Positive' Class : Bad
##
```

```
# finding the best tuning parameter based on cost
# estimate that incorrectly identifying a bad customer as good is 5 times worse
preds_p <- stats::predict(credit_mod_glm, test_credit, type = 'prob') %>% cbind(Cla
ss = as.character(test_credit$Class)) %>%
        mutate(prob_pred =  ifelse(Good > .95, 'Good', 'Bad'))

confusionMatrix(preds_p$Class, preds_p$prob_pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   57    3
##       Good  97   43
##
##                Accuracy : 0.5
##                  95% CI : (0.4287, 0.5713)
##     No Information Rate : 0.77
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1776
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.3701
##             Specificity : 0.9348
##          Pos Pred Value : 0.9500
##          Neg Pred Value : 0.3071
##              Prevalence : 0.7700
##          Detection Rate : 0.2850
##    Detection Prevalence : 0.3000
##       Balanced Accuracy : 0.6525
##
##        'Positive' Class : Bad
##
```

```
# fit model with cross validation - bonus ! regularized logistic regression using g
lmnet!
(credit_mod_glmnet <- train(
        Class ~ .,
        data = train_credit,
        method = 'glmnet',
        family = 'binomial',
        trControl = trainControl(method = 'cv', classProbs = T),
        metric = 'Accuracy'
        ))
```

```
## glmnet
##
## 800 samples
##  61 predictor
##   2 classes: 'Bad', 'Good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 720, 720, 720, 720, 720, 720, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy  Kappa
##   0.10   0.0002774894  0.75250   0.3767625
##   0.10   0.0027748935  0.75375   0.3791359
##   0.10   0.0277489353  0.75250   0.3662899
##   0.55   0.0002774894  0.75250   0.3767625
##   0.55   0.0027748935  0.75250   0.3763333
##   0.55   0.0277489353  0.74625   0.3198880
##   1.00   0.0002774894  0.75250   0.3767625
##   1.00   0.0027748935  0.75125   0.3751167
##   1.00   0.0277489353  0.74750   0.2769884
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda
##  = 0.002774894.
```

```
# view coefficients of best tuned glmnet model
coef(credit_mod_glmnet$finalModel, credit_mod_glmnet$bestTune$lambda)
```

```
## 62 x 1 sparse Matrix of class "dgCMatrix"
##                                                1
## (Intercept)                         5.2816202641
## Duration                           -0.0204897224
## Amount                             -0.0001729812
## InstallmentRatePercentage          -0.4270024625
## ResidenceDuration                  -0.0273609640
## Age                                 0.0194583710
## NumberExistingCredits              -0.1701130727
## NumberPeopleMaintenance            -0.3438711995
## Telephone                          -0.3405365026
## ForeignWorker                      -2.3242769842
## CheckingAccountStatus.lt.0         -0.7310816632
## CheckingAccountStatus.0.to.200     -0.2328443285
## CheckingAccountStatus.gt.200        0.4036801140
## CheckingAccountStatus.none          0.8028349814
## CreditHistory.NoCredit.AllPaid     -0.6072512081
## CreditHistory.ThisBank.AllPaid     -0.5316055862
## CreditHistory.PaidDuly             -0.0468581837
## CreditHistory.Delay                -0.0891441680
## CreditHistory.Critical              0.7401388266
## Purpose.NewCar                     -0.7850434242
## Purpose.UsedCar                     0.9328936073
## Purpose.Furniture.Equipment         0.2460575737
## Purpose.Radio.Television            0.1926812874
## Purpose.DomesticAppliance          -0.3698374671
## Purpose.Repairs                        -0.4401197420
## Purpose.Education                      -0.8036913444
## Purpose.Vacation                        .
## Purpose.Retraining                      1.3724077200
## Purpose.Business                       -0.0481241950
## Purpose.Other                           0.5892134858
## SavingsAccountBonds.lt.100             -0.3290921835
## SavingsAccountBonds.100.to.500          .
## SavingsAccountBonds.500.to.1000        -0.2055022507
## SavingsAccountBonds.gt.1000             0.4037749774
## SavingsAccountBonds.Unknown             0.7891181151
## EmploymentDuration.lt.1                -0.2872554149
## EmploymentDuration.1.to.4               .
## EmploymentDuration.4.to.7               0.6183033763
## EmploymentDuration.gt.7                 0.0081412791
## EmploymentDuration.Unemployed          -0.2892658681
## Personal.Male.Divorced.Seperated       -0.5113319074
## Personal.Female.NotSingle              -0.1149992397
## Personal.Male.Single                    0.4934496152
## Personal.Male.Married.Widowed          -0.2402252661
## Personal.Female.Single                  .
## OtherDebtorsGuarantors.None             .
## OtherDebtorsGuarantors.CoApplicant     -0.6910981988
## OtherDebtorsGuarantors.Guarantor        0.9334851955
## Property.RealEstate                     0.5322148283
## Property.Insurance                      .
## Property.CarOther                      -0.0230135112
## Property.Unknown                       -0.1271839685
## OtherInstallmentPlans.Bank             -0.2866277516
## OtherInstallmentPlans.Stores           -0.2063734442
## OtherInstallmentPlans.None              0.4043912989
## Housing.Rent                           -0.4608631802
## Housing.Own                             0.1134099097
## Housing.ForFree                         0.0783434454
## Job.UnemployedUnskilled                 0.0629224324
## Job.UnskilledResident                   .
## Job.SkilledEmployee                    -0.0269212382
## Job.Management.SelfEmp.HighlyQualified  0.0709239711
```

```
# predict back onto test data set
credit_pred_glmnet <- test_credit %>% add_predictions(credit_mod_glmnet) %>%
        dplyr::select(Class, pred) %>%
        as.data.frame()

confusionMatrix(credit_pred_glmnet$Class, credit_pred_glmnet$pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   20   40
##       Good   8  132
##
##                Accuracy : 0.76
##                  95% CI : (0.6947, 0.8174)
##     No Information Rate : 0.86
##     P-Value [Acc > NIR] : 0.9999
##
##                   Kappa : 0.3258
##  Mcnemar's Test P-Value : 7.66e-06
##
##             Sensitivity : 0.7143
##             Specificity : 0.7674
##          Pos Pred Value : 0.3333
##          Neg Pred Value : 0.9429
##              Prevalence : 0.1400
##          Detection Rate : 0.1000
##    Detection Prevalence : 0.3000
##       Balanced Accuracy : 0.7409
##
##        'Positive' Class : Bad
##
```