ISYE 6501 - Week 4 Homework

Ujjawal Madan

04/06/2020

Contents

Question 9.1

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. You can use the R function prcomp for PCA. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

Let's start by running the component analysis on the first 15 predictor variables.

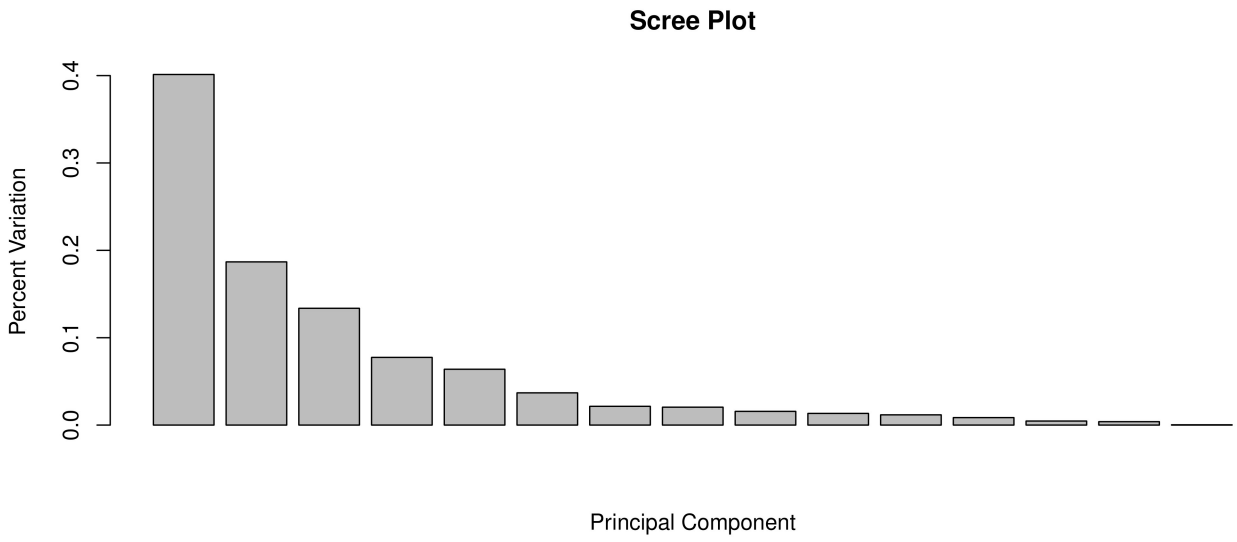
```
pca <- prcomp(us_crime[,-16], scale = TRUE)
```

Let's now examine how much variance each principal component accounts for.

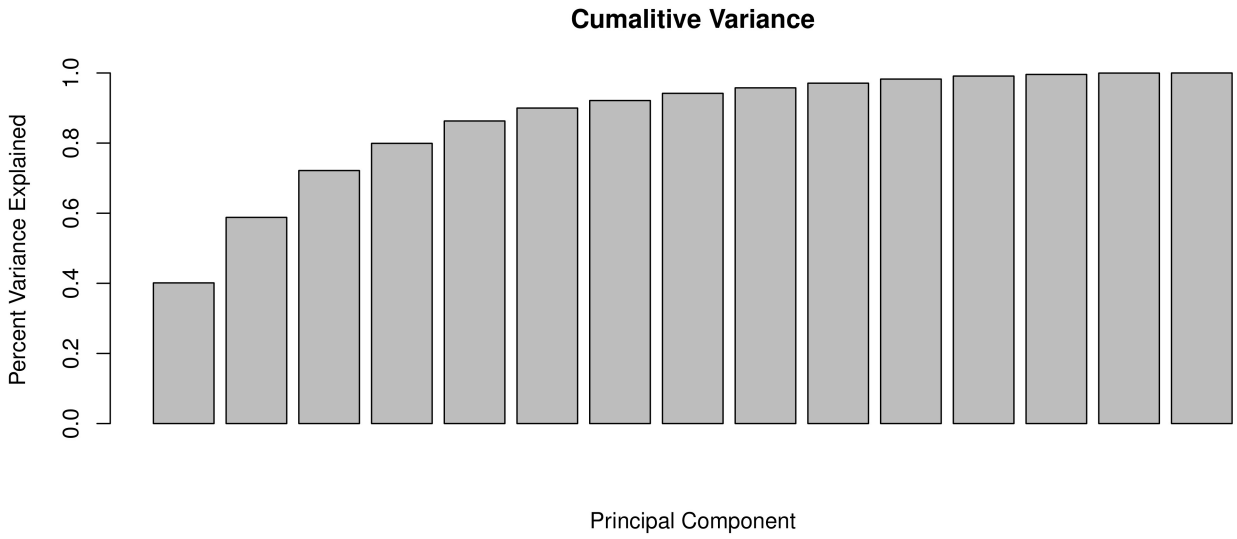
```
#Scree Plot
pca_var <- pca$sdev^2

variation <- sapply(pca_var, function(x){
  x/sum(pca_var)  })

barplot(variation, main = 'Scree Plot', xlab = 'Principal Component', ylab = 'Percent Variation')
```



Seeing it in terms of of culimative variance would allow us to select the number of principal components based on how much variance we are looking for.



For example, if we want to explain 90 percent of the variance, we should use approximately 6 principal components.

Using the principal components, let's create a linear model using only the first 6 principal components.

```
ncomps <- 6
data <- data.frame(cbind(pca$x[,1:ncomps], us_crime[,16]))
colnames(data) <- c(as.vector(colnames(pca$x[,1:ncomps])), 'Crime')
model <- lm(Crime ~ ., data)
summary(model)$r.squared
```

```
## [1] 0.6586023
```

Above is the R^2 value of the model. While this R^2 value may not be very high, it captures a little over 80 percent of the variance that our linear model from the previous homework assignment did. What if we tried using all 15 components, since we used 15 attributes in the linear model from last week?

```
## [1] 0.8030868
```

We do indeed end up achieving an R^2 value that is precisely the same as our previous model.

I have now created a function below that creates a linear model using the principal components and is then used to predict Crime from the dataset. This function takes in the number of principal components one wants to employ in the linear model and returns the prediction of the response value based on the input vector it receives (in terms of the original variables before scaling).

IMPORTANT - After the model is created using the principal components, the final calculation is indeed done using the original variables. The input vector consists of original variables before scaling rather than principal components.

```
input <- function(ncomps, inputvector = us_crime[-16,1]){

  pca <- prcomp(us_crime[,-16], scale = TRUE)
  mat <- as.matrix(us_crime[,-16])
  sds <- apply(mat, 2, sd)
  means <- apply(mat, 2, mean)
  scaled <- scale(as.matrix(us_crime[,-16]), center = TRUE, scale = TRUE)
  newpcavars <- scaled %*% as.matrix(pca$rotation[,1:ncomps])
  #How many of those vars you want to use is up to you
  data <- data.frame(cbind(newpcavars, us_crime[,16]))
  model <- lm(Crime ~ ., data)
  intercept <- model$coefficients[1]
  coefficients <- model$coefficients[2:length(model$coefficients)]

  original_variables <<- pca$rotation %*% coefficients

  after_scaling_equations <<- vector()
  #newcoef <- apply(pca$rotation*coefficients, 1, mean)
  for (i in seq(1, 15)){
    after_scaling_equations[i] <<- paste(toString(original_variables[i] / sds[i]),
                                ' *x', i, ' - ', toString((original_variables[i]/sds[i])*means[i]), sep
  }
  after_scaling_equations[16] <<- intercept

  scaled_input <- (inputvector - means)/sds
  transformed_input <- scaled_input %*% as.matrix(pca$rotation[,1:ncomps])
  output <- intercept + (transformed_input %*% coefficients)
  return (output)
}
```

```
inputvector <- as.vector(as.matrix(us_crime[1,-16]))
us_crime[1,16]
```

```
## # A tibble: 1 x 1
##   Crime
##   <dbl>
## 1   791
```

```
input(15, inputvector)
```

```
##          [,1]
## [1,] 755.0322
```

For example, if we use all 15 principal components, then we achieve a regression value of 755 for row 1 when the actual value was 791. Below is the linear model transformed in terms of the original variables after scaling.

M	110.381746
So	-1.821758
Ed	210.678383
Po1	572.994704
Po2	-305.958132
LF	-26.826419
M.F	51.293417
Pop	-27.906491
NW	43.233976
U1	-105.055568
U2	141.714373
Wealth	92.791716
Ineq	281.953836
Prob	-110.394044
Time	-24.655435

Below is a table which consists of the linear equations one can employ for unscaled variables. For example, entering the exact original values of row 1 in the us_crime dataset into the equations below and adding them will get you the 755 value we saw just previously.

```
kable(after_scaling_equations)
```

x
87.8301732430492 *x1 - 1217.10195389783
-3.80345029611548 *x2 - -1.29479159016697
188.32431475042 *x3 - 1989.42600582093
192.80433827659 *x4 - 1638.83687535101
-109.421925381632 *x5 - -877.936341732199
-663.82614507977 *x6 - -372.533583034554
17.406855527635 *x7 - 1711.13093422935
-0.733008149584922 *x8 - -26.8405750092692
4.2044610019414 *x9 - 42.5187300898457
-5827.10272440477 *x10 - -556.302338817111
167.799672221837 *x11 - 570.16186497505
0.0961662430048673 *x12 - 505.241072025359
70.6720994522304 *x13 - 1371.03872937327
-4855.26581547547 *x14 - -228.641181980059
-3.47901784343313 *x15 - -92.5346427195064
905.085106382979

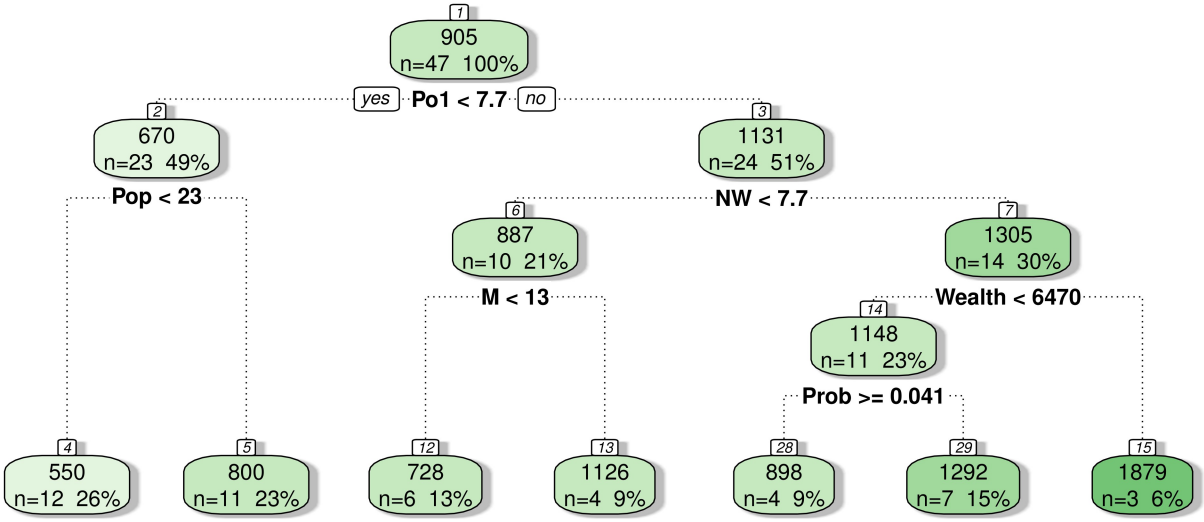
Question 10.1

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).

Let's start by building the classification tree using the rpart package. There are a number of tuning parameters which allow us to configure our model. In our case, the ones we should consider include minsplit (the smallest number of observations in the parent node that could be split further), minbucket (the smallest number of observations that are allowed in a terminal nod)and cp (minimum improvement in the model needed at each node).

If we set minbucket and minsplit to be 1, then our model will be overfitted. As mentioned in the lecture, each leaf node should have at least 5 percent of the data points. Therefore let's set our minbucket to be 3. Our cp parameter is a bit more difficult to determine since there is no predefined formula for what the minimum improvement should be. However, after some experimentation, 0.05 seems to be a reasonable value. If the cost is too low for any reason, we can always prune the three (I have included the code)



Taking a look at our model, it seems that we have 7 leaf nodes. There are a total of only 6 features variables utilized in our splits with Po1 used for our first split and the most deep split using prob.

We can also take a look at our predicted values vs actual values. A measure like RMSE would not be particularly useful to run on our predicted values since we are not running our model on a testing or validation set. We could create a model so that it completely fit the training data and RMSE equaled 0 on our training set if we wanted. Neverthless, it gives an idea of to what degree our model has fit the training data.

799.5455	791
1291.5714	1635
550.5000	578
1878.6667	1969
1126.0000	1234
727.5000	682
898.0000	963
1291.5714	1555
799.5455	856
550.5000	705
1878.6667	1674
799.5455	849
799.5455	511
550.5000	664
799.5455	798
898.0000	946
550.5000	539
898.0000	929
727.5000	750
1291.5714	1225
799.5455	742
550.5000	439
1291.5714	1216
1126.0000	968

550.5000	523
1878.6667	1993
550.5000	342
1291.5714	1216
1291.5714	1043
799.5455	696
550.5000	373
898.0000	754
799.5455	1072
727.5000	923
727.5000	653
1126.0000	1272
799.5455	831
550.5000	566
799.5455	826
1291.5714	1151
550.5000	880
550.5000	542
799.5455	823
1126.0000	1030
550.5000	455
727.5000	508
727.5000	849

Let's also take a look at variable importance.

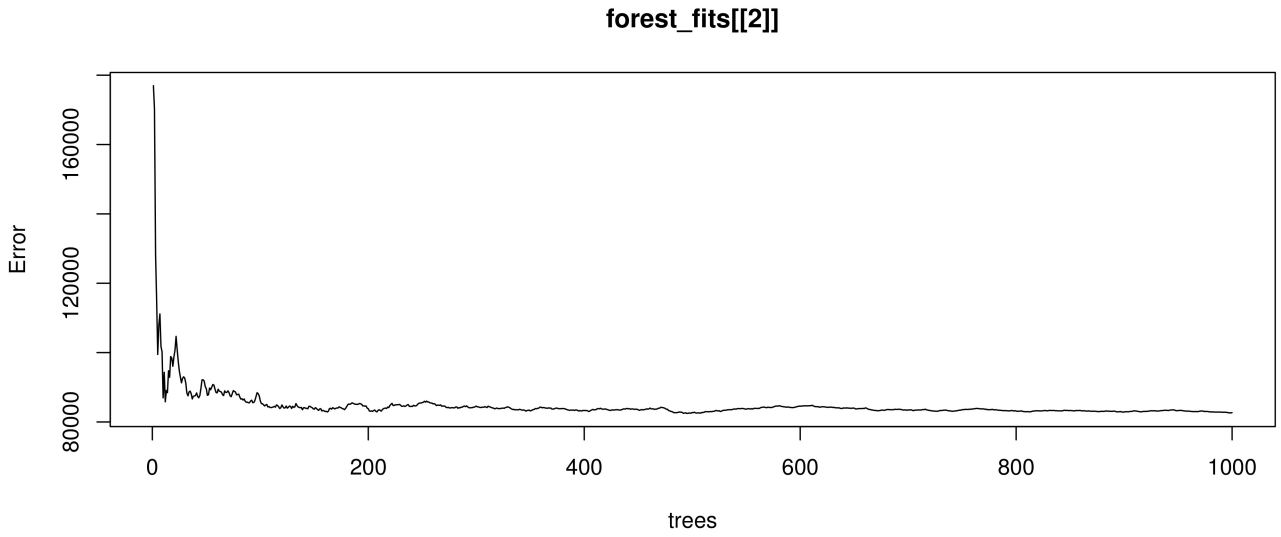
	x
Po2	3335422.7
Wealth	3076232.7
Po1	3011753.5
Prob	2752418.6
Ineq	1792774.7
M	1769753.2
Ed	1502728.2
NW	1443027.3
LF	915247.9
Time	799049.5
Pop	661770.6
So	260372.5
U2	197143.5

Let's jump to the random forest model now. The two parameters that are of concern to us (if we use the randomForest package) are mtry and ntrees. As Joel has stated, any number between 500 and 1000 trees ought to suffice. So let's split it and pick 750. Mtry (the number of factors considered during a split) is typically log(m) with m being number of feature variables. After some research of my own, it seems that another formula sometimes used is sqrt(m). Let's try different values of mtry and see what we get.

```
## [[1]]
##
## Call:
##  randomForest(formula = Crime ~ ., data = us_crime, mtry = factors,      ntree = 1000, importance = TRU
##                Type of random forest: regression
##                      Number of trees: 1000
```

```
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 85111.76
##                     % Var explained: 41.86
##
## [[2]]
##
## Call:
##  randomForest(formula = Crime ~ ., data = us_crime, mtry = factors,      ntree = 1000, importance = TRU
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 82672.35
##                     % Var explained: 43.53
##
## [[3]]
##
## Call:
##  randomForest(formula = Crime ~ ., data = us_crime, mtry = factors,      ntree = 1000, importance = TRU
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 84017.69
##                     % Var explained: 42.61
##
## [[4]]
##
## Call:
##  randomForest(formula = Crime ~ ., data = us_crime, mtry = factors,      ntree = 1000, importance = TRU
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 84018.45
##                     % Var explained: 42.61
```

As we can see, the model that explains the most variance is the one with mtry at 4. So let's choose that. Although we know that ntree set to a 1000 is an appropriate number of trees, we can take a look to confirm.



We can see that adding additional trees after 500 does not do much to reduce our error. Nor does it increase it.

One of the great things about the randomforest model in the randomForest package is that we can take a look at variable importance. Using a randomForest model in a machine learning task can provide us with the variable importance and help us select features or at least determine how much predictive power each variable has on it's own.

	%IncMSE	IncNodePurity
M	2.2600096	225520.44
So	3.8362162	27316.31
Ed	5.7770601	246575.56
Po1	16.8635203	1131510.45
Po2	15.5338738	1039145.33
LF	4.0972324	317800.90
M.F	2.8692776	306029.68
Pop	0.9434895	364820.65
NW	13.0976256	548031.40
U1	0.5120302	146942.36
U2	2.4809782	196500.31
Wealth	5.4284870	600589.63
Ineq	2.5304634	228978.04
Prob	11.3834887	791344.58
Time	3.8866389	214979.28

It seems that Po1 is one of the most important variables which would in fact line up with our previous decision tree that used Po1 as a variable for the first split. However, our classification tree also used M and Pop which do not seems to be very important variables according to our random forest model.

We can also take a look at our predicted values vs our actual values just as we did on for the rpart model.

755.0517	791
1121.9381	1635
635.6933	578
1339.0928	1969
994.0646	1234
1301.0177	682
955.6146	963

1049.7898	1555
806.7510	856
738.8128	705
1211.9061	1674
845.5022	849
711.0347	511
653.9585	664
716.6700	798
957.3608	946
667.5650	539
1027.8847	929
1128.4743	750
1164.5911	1225
853.5079	742
723.1974	439
1043.9543	1216
913.2009	968
686.1632	523
1174.2580	1993
849.2567	342
1070.6305	1216
1357.0752	1043
810.0192	696
739.7717	373
1034.8521	754
721.3368	1072
922.5684	923
1098.8919	653
1040.0286	1272
772.3242	831
613.4472	566
794.2351	826
1130.3409	1151
811.8312	880
544.2689	542
885.4112	823
1081.0714	1030
619.0232	455
988.9023	508
946.2084	849

Question 10.2

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.

I believe an interesting use case for logistic regression would be in a credit card fraud detection algorithm. For example, predictors could include location (with crossreference to previous locations the individual made purchases at) , cost of purchase (with crossreferences with previous purchases made), type of purchase (with crossreference to previous type of purchases), date and time of day etc. The advantages of logistic regression vs SVM or KNN would be that the algorithm would generate a probability for there being a fraud. And the threshold for what is marked as fraud and what is not could be adjusted based on the costs associated with False Negatives and False Positives. In such a scenario, I would imagine that the threshold might be a bit lower to be safe, since the cost of overlooking a fraud is much higher than the cost of alerting the customer that a certain suspicious transcation was made.

Question 10.3

Question 10.3.1

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.

Let's start by importing the data and preparing it.

```
german <- read_table2("http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data
names(german)[length(names(german))]<-'Response'

cols <- c("A11", 'A34', 'A43', 'A65', 'A75', 'A93', 'A101', 'A121', 'A143', 'A152', 'A173', 'A192', 'A201'
german[cols] <- lapply(german[cols], factor)
german$Response <- german$Response -1
german$Response <- as.factor(german$Response)
```

Let's now run the model with default parameters and threshold set to 0.5

```
##
## Call:  glm(formula = Response ~ ., family = binomial, data = german)
##
## Coefficients:
## (Intercept)       A11A12       A11A13       A11A14          '6'       A34A31
##   0.3987009   -0.3757199   -0.9667093   -1.7130808    0.0278266    0.1431906
##      A34A32       A34A33       A34A34       A43A41       A43A410       A43A42
##  -0.5865718   -0.8535097   -1.4349503   -1.6659574   -1.4878803   -0.7909389
##      A43A43       A43A44       A43A45       A43A46       A43A48       A43A49
##  -0.8899120   -0.5230845   -0.2160893    0.0366085   -2.0585145   -0.7394822
##      '1169'       A65A62       A65A63       A65A64       A65A65       A75A72
##   0.0001283   -0.3568564   -0.3760575   -1.3392650   -0.9443435   -0.0665079
##      A75A73       A75A74       A75A75          '4'       A93A92       A93A93
##  -0.1828704   -0.8309884   -0.2762359    0.3300631   -0.2751169   -0.8152092
##      A93A94      A101A102     A101A103        '4_1'     A121A122     A121A123
##  -0.3669916    0.4356453   -0.9790208    0.0047983    0.2801328    0.1934917
##    A121A124         '67'     A143A142     A143A143     A152A152     A152A153
##   0.7289470   -0.0144631   -0.1232388   -0.6458519   -0.4434850   -0.6841435
##         '2'      A173A172     A173A173     A173A174          '1'     A192A192
##   0.2719673    0.5361826    0.5554362    0.4793417    0.2640211   -0.2991574
##    A201A202
##  -1.3924049
##
## Degrees of Freedom: 998 Total (i.e. Null);  950 Residual
## Null Deviance:       1221
## Residual Deviance: 895.7     AIC: 993.7
```

```
##
## Call:
## glm(formula = Response ~ ., family = binomial, data = german)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.3408  -0.6988  -0.3760   0.7121   2.6116
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.987e-01  1.084e+00   0.368 0.713081
## A11A12      -3.757e-01  2.179e-01  -1.724 0.084697 .
## A11A13      -9.667e-01  3.692e-01  -2.619 0.008828 **
## A11A14      -1.713e+00  2.322e-01  -7.376 1.63e-13 ***
## `6`          2.783e-02  9.297e-03   2.993 0.002762 **
## A34A31       1.432e-01  5.488e-01   0.261 0.794169
## A34A32      -5.866e-01  4.305e-01  -1.363 0.172988
## A34A33      -8.535e-01  4.716e-01  -1.810 0.070347 .
## A34A34      -1.435e+00  4.399e-01  -3.262 0.001105 **
## A43A41      -1.666e+00  3.743e-01  -4.451 8.54e-06 ***
## A43A410     -1.488e+00  7.762e-01  -1.917 0.055256 .
## A43A42      -7.909e-01  2.610e-01  -3.031 0.002441 **
## A43A43      -8.899e-01  2.472e-01  -3.601 0.000318 ***
## A43A44      -5.231e-01  7.622e-01  -0.686 0.492559
## A43A45      -2.161e-01  5.500e-01  -0.393 0.694385
## A43A46       3.661e-02  3.964e-01   0.092 0.926423
## A43A48      -2.059e+00  1.212e+00  -1.698 0.089435 .
## A43A49      -7.395e-01  3.339e-01  -2.215 0.026782 *
## `1169`       1.283e-04  4.443e-05   2.887 0.003887 **
## A65A62      -3.569e-01  2.861e-01  -1.247 0.212254
## A65A63      -3.761e-01  4.011e-01  -0.938 0.348468
## A65A64      -1.339e+00  5.248e-01  -2.552 0.010718 *
## A65A65      -9.443e-01  2.626e-01  -3.596 0.000324 ***
## A75A72      -6.651e-02  4.269e-01  -0.156 0.876200
## A75A73      -1.829e-01  4.104e-01  -0.446 0.655928
## A75A74      -8.310e-01  4.454e-01  -1.866 0.062069 .
## A75A75      -2.762e-01  4.133e-01  -0.668 0.503926
## `4`          3.301e-01  8.827e-02   3.739 0.000184 ***
## A93A92      -2.751e-01  3.865e-01  -0.712 0.476530
## A93A93      -8.152e-01  3.799e-01  -2.146 0.031889 *
## A93A94      -3.670e-01  4.536e-01  -0.809 0.418493
## A101A102     4.356e-01  4.101e-01   1.062 0.288116
## A101A103    -9.790e-01  4.242e-01  -2.308 0.021003 *
## `4_1`        4.798e-03  8.639e-02   0.056 0.955709
## A121A122     2.801e-01  2.534e-01   1.106 0.268940
## A121A123     1.935e-01  2.360e-01   0.820 0.412293
## A121A124     7.289e-01  4.245e-01   1.717 0.085946 .
## `67`        -1.446e-02  9.227e-03  -1.568 0.116992
## A143A142    -1.232e-01  4.119e-01  -0.299 0.764790
## A143A143    -6.459e-01  2.391e-01  -2.701 0.006911 **
## A152A152    -4.435e-01  2.347e-01  -1.890 0.058768 .
## A152A153    -6.841e-01  4.769e-01  -1.434 0.151435
## `2`          2.720e-01  1.895e-01   1.435 0.151174
## A173A172     5.362e-01  6.795e-01   0.789 0.430035
## A173A173     5.554e-01  6.548e-01   0.848 0.396331
## A173A174     4.793e-01  6.622e-01   0.724 0.469134
## `1`          2.640e-01  2.492e-01   1.059 0.289416
## A192A192    -2.992e-01  2.013e-01  -1.486 0.137186
## A201A202    -1.392e+00  6.257e-01  -2.225 0.026051 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1221.01  on 998  degrees of freedom
## Residual deviance:  895.75  on 950  degrees of freedom
## AIC: 993.75
##
## Number of Fisher Scoring iterations: 5
```

Let's take a look at the confusionMatrix for this model

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 625 140
##          1  74 160
##
##                Accuracy : 0.7858
##                  95% CI : (0.759, 0.8109)
##     No Information Rate : 0.6997
##     P-Value [Acc > NIR] : 5.687e-10
##
##                   Kappa : 0.4561
##
##  Mcnemar's Test P-Value : 8.859e-06
##
##             Sensitivity : 0.8941
##             Specificity : 0.5333
##          Pos Pred Value : 0.8170
##          Neg Pred Value : 0.6838
##              Prevalence : 0.6997
##          Detection Rate : 0.6256
##    Detection Prevalence : 0.7658
##       Balanced Accuracy : 0.7137
##
##        'Positive' Class : 0
##
```

Question 10.3.2

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.

With the threshold set to 0.5, the model incorrectly predicts 140 of a 1000 data points as good customers. And incorrectly predicts good customers as bad 74/1000 cases. Let's see if we can optimize this.

```
#False Negative (incorrectly predicted as good customers)
cm$table[1,2]
```

```
## [1] 140
```

```
#False Positive (incorrectly predicted as bad cusomters)
cm$table[2,1]
```

```
## [1] 74
```

Our false negative and our false positive can both be found from the confusion matrix. We can use this to come up with a linear equation.

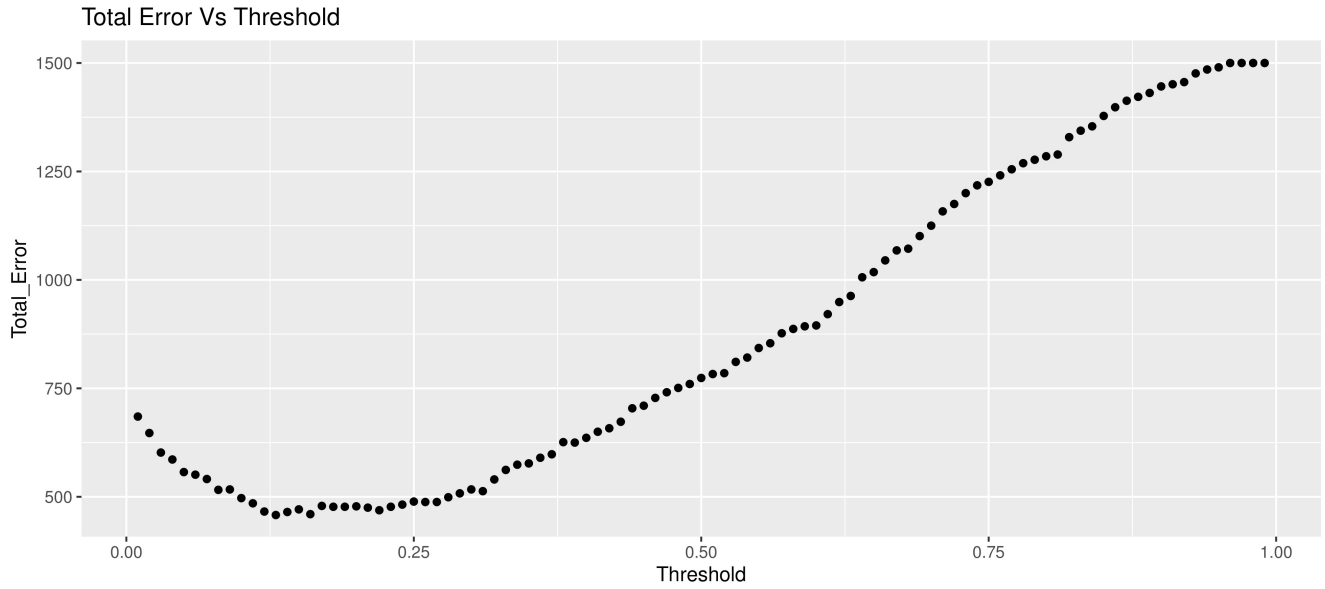
Y = 5 * False Negative(incorrectly predicted as good customers) + False Positive (incorrectly predicted as bad cusomters)

```
#Linear Equation
5 * cm$table[1,2] + cm$table[2,1]
```

```
## [1] 774
```

Let's run different probability thresholds and see what probability threshold most reduces total error.

```
counter <- 1
value <- vector()
prediction <- predict(fit, german[,-ncol(german)], type = 'response')
thresholds <- seq(0.01, 0.99, 0.01)
for (i in thresholds){
  prediction1 <- ifelse (prediction > i, 1, 0)
  cm <- caret::confusionMatrix(factor(prediction1), german$Response)
  value[counter] <- 5 * cm$table[1,2] + cm$table[2,1]
  counter <- counter + 1
}
table <- cbind(thresholds, value)
colnames(table) <- c('Threshold', 'Total_Error')
as.data.frame(table) %>% ggplot(aes(Threshold, Total_Error)) + geom_point() + ggtitle('Total Error Vs Thre
```



As we can see, a probability threhsold of approximately 0.2 would most reduce our total error. Let's find the exact minimum of this plot.

```
thresholds[which.min(value)]
```

```
## [1] 0.13
```

The exact minimum is 0.13. Let's see what our new model looks like along with the confusion matrix when the probability threshold is 0.13

```
prediction <- predict(fit, german[,-ncol(german)], type = 'response')
prediction_optimized <- ifelse (prediction > 0.13, 1, 0)
caret::confusionMatrix(factor(prediction_optimized), german$Response)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 341  20
##          1 358 280
##
##                Accuracy : 0.6216
##                  95% CI : (0.5907, 0.6518)
##     No Information Rate : 0.6997
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3187
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.4878
##             Specificity : 0.9333
##          Pos Pred Value : 0.9446
##          Neg Pred Value : 0.4389
##              Prevalence : 0.6997
##          Detection Rate : 0.3413
##    Detection Prevalence : 0.3614
##       Balanced Accuracy : 0.7106
##
##        'Positive' Class : 0
##
```

As we can see, while our false positive hav greatly risen, there are very few instances where our model is now incorrectly predicting a bad customer as good. In layman's terms, the bank is now very conservative in lending out loans. They are denying a large number of applications, many of which may actually not be risky. However, for those of whom they are lending to, the probability of any one of them being a credit risk is very very low.