H4-ISYE6501x-Summer2018-OA

6/10/2018

Objectives

The first area of focus of Week4's homework is Principal Component Analysis and comparing that with other models like linear regression with manual predictor selection (question 8.2 from last homework). It then builds on top of that by applying regression trees and random forests to show how the results get more accurate and sharpened (albeit not as explainable). Finally it wraps up looking at logistic regression, and measurement of cost using the confusion matrix principles

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.ould you need? Would you expect the value of alpha (the first smoothing parameter) to be closer to 0 or 1, and why?

Some other tips from the question:

- 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)!)
- First we download and load the data

```
dataFile <- "uscrime.txt"
if (!file.exists(dataFile)) {
  crimeDataURL <- paste0(c("http://www.statsci.org/data/general/uscrime.txt"))
  download.file(crimeDataURL, dataFile) }

crimeDataTable <- read.table(dataFile, header = TRUE )
```

- Then, we apply the prcomp formula on the predictors, and print the summary:

```
#pcaModel <- prcomp( ~ crimeDataTable[,1:15] , scale. = TRUE)
# note, the tilda didn't work, so you shove the numeric data (x per man page directly, like so:

pcaModel2 <- prcomp(crimeDataTable[,1:15] ,  scale. = TRUE)

summary(pcaModel2)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.56729 0.55444 0.48493 0.44708 0.41915 0.35804
## Proportion of Variance 0.02145 0.02049 0.01568 0.01333 0.01171 0.00855
## Cumulative Proportion  0.92142 0.94191 0.95759 0.97091 0.98263 0.99117
##                           PC13   PC14    PC15
## Standard deviation     0.26333 0.2418 0.06793
```

```
## Proportion of Variance 0.00462 0.0039 0.00031
## Cumulative Proportion  0.99579 0.9997 1.00000
```

- some PCAs have higher variance than others, aka higher data spread (recall D1 dimension in lectures)
- these are the more important PCs, and have been ranked by the model as such. (by proportion of variance)
- now we create a NEW table, with the new Principal components and append crime , and then run regression on it:

```
CrimeDataWithPrincipalComponents <- as.data.frame(cbind(pcaModel2$x, crimeDataTable[,16]))
cn <- colnames(CrimeDataWithPrincipalComponents)
cn[16] <- "Crime"
colnames(CrimeDataWithPrincipalComponents)  <- cn

# running reg:
RegressionModelBasedonPCs <- lm(Crime ~ . , data = CrimeDataWithPrincipalComponents)
# TA used the command below, but I checked. both yield same output, and mine is cleaner
#RegressionModelBasedonPCs2 <- lm(CrimeDataWithPrincipalComponents[,16]~. , data = CrimeDataWithPrincip
```

- let's check out the specifics around this regression model:

```
summary(RegressionModelBasedonPCs)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = CrimeDataWithPrincipalComponents)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -395.74  -98.09   -6.69  112.99  512.67
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      30.50  29.680  < 2e-16 ***
## PC1            65.22      12.56   5.191 1.24e-05 ***
## PC2           -70.08      18.42  -3.806 0.000625 ***
## PC3            25.19      21.77   1.157 0.255987
## PC4            69.45      28.59   2.429 0.021143 *
## PC5          -229.04      31.49  -7.274 3.49e-08 ***
## PC6           -60.21      41.44  -1.453 0.156305
## PC7           117.26      54.34   2.158 0.038794 *
## PC8            28.72      55.60   0.517 0.609159
## PC9           -37.18      63.57  -0.585 0.562890
## PC10           56.32      68.95   0.817 0.420261
## PC11           30.59      73.54   0.416 0.680272
## PC12          289.61      86.09   3.364 0.002059 **
## PC13           81.79     117.06   0.699 0.489962
## PC14          219.19     127.48   1.719 0.095517 .
## PC15         -622.21     453.79  -1.371 0.180174
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 209.1 on 31 degrees of freedom
## Multiple R-squared:  0.8031, Adjusted R-squared:  0.7078
## F-statistic: 8.429 on 15 and 31 DF,  p-value: 3.539e-07
```

Observations:

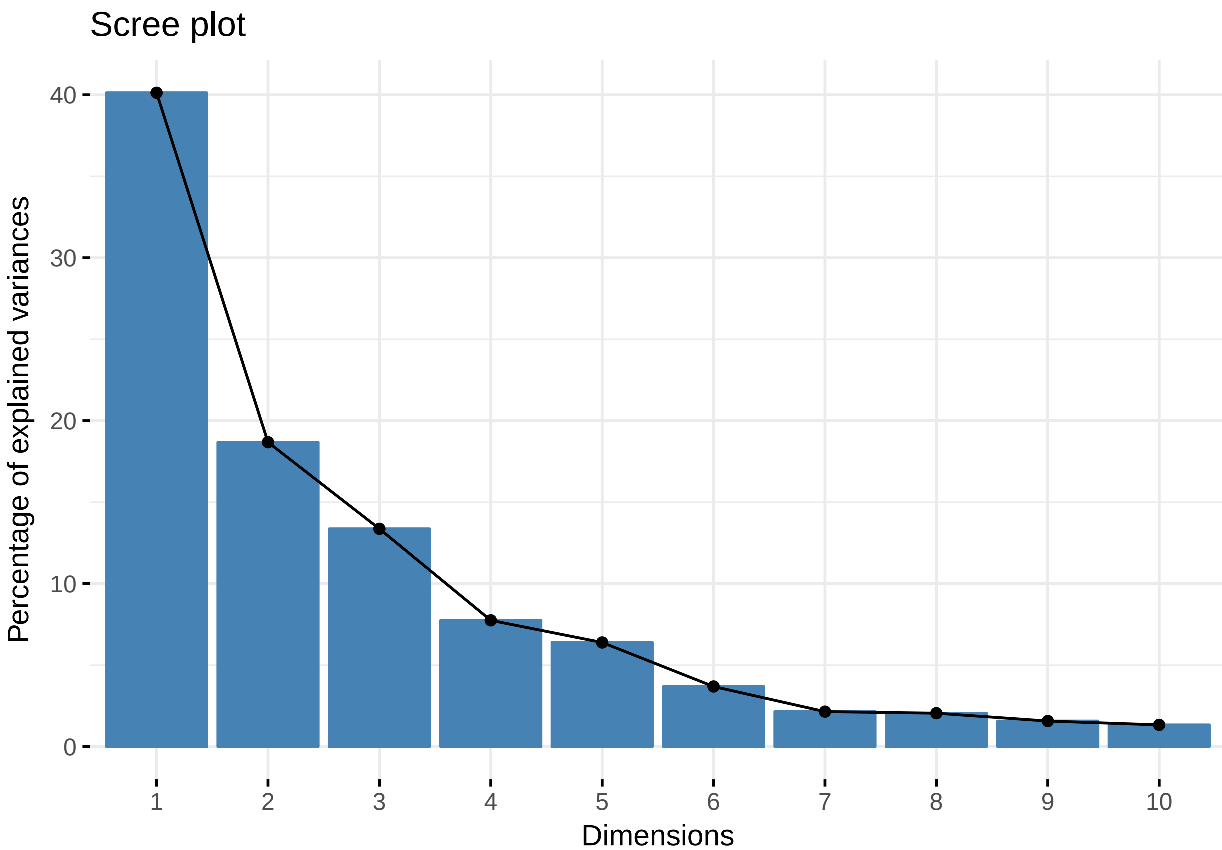
- exceptionally low p-value (which is good, shows this model is accurate)
- adjusted r-squared is 70.78%, very good, however looking at the probability column for each PCA, the question is, could this be because of over-fitting?. Specifically
- only Intercept, PC1, PC2, PC5 and PC12 have *** , however look at the probabilities. virtually all components except PC9, PC10, and PC11 and PC13 are less then 0.25 probability. those 4 have .61, .56, .42, .68 respectively
- so, let's drop those few from the model

```r
RegressionModelBasedonPCs_SHARPER <- lm(Crime ~ PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8+PC12+PC14+PC15, data = C
summary(RegressionModelBasedonPCs_SHARPER)
```

```
##
## Call:
## lm(formula = Crime ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 + PC7 +
##     PC8 + PC12 + PC14 + PC15, data = CrimeDataWithPrincipalComponents)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -381.33  -83.34  -12.22  114.81  498.15
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      29.46  30.720  < 2e-16 ***
## PC1            65.22      12.14   5.372 5.19e-06 ***
## PC2           -70.08      17.79  -3.939 0.000372 ***
## PC3            25.19      21.03   1.198 0.239022
## PC4            69.45      27.63   2.514 0.016693 *
## PC5          -229.04      30.42  -7.529 8.04e-09 ***
## PC6           -60.21      40.04  -1.504 0.141607
## PC7           117.26      52.50   2.234 0.032003 *
## PC8            28.72      53.71   0.535 0.596297
## PC12          289.61      83.18   3.482 0.001356 **
## PC14          219.19     123.16   1.780 0.083823 .
## PC15         -622.21     438.43  -1.419 0.164689
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 202 on 35 degrees of freedom
## Multiple R-squared:  0.7925, Adjusted R-squared:  0.7273
## F-statistic: 12.15 on 11 and 35 DF,  p-value: 6.004e-09
```

- our r squared eeked out better this time (73%) and p-value got smaller by a magnitude of 100! (from 10^-7 to 10^-9)
- this model is a keeper.
- however there is another way to select the appropriate principal values, aka the ELBOW diagram
- luckily the library `factoextra` has a function which can do just this!

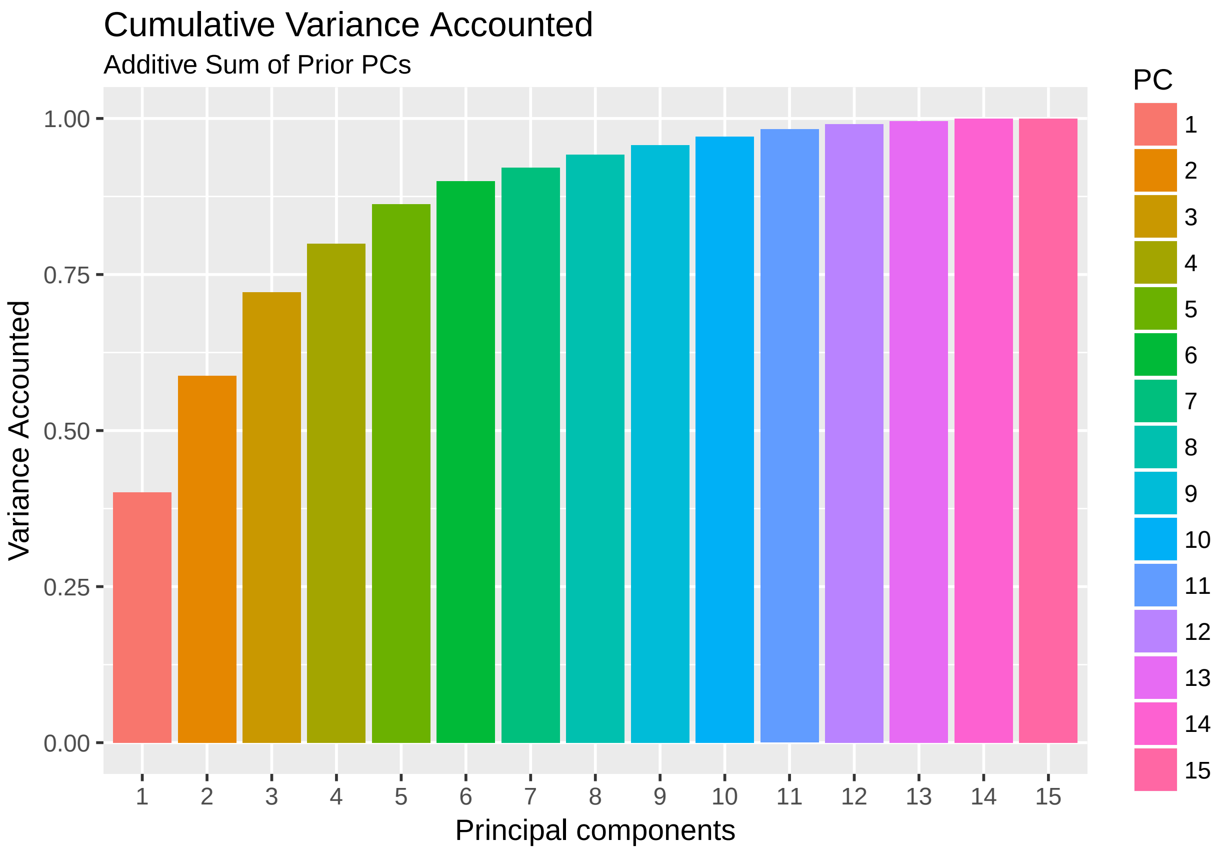
```r
fviz_eig(pcaModel2)
```



- this shows that the first 5 dimensions aka, principal components explain about 87% of the variance, and thus are the most relevant to the model, after which there is likely overfitting going on.
- another way to depict this (courtesy of Matt Nguyen on the slack channel) is using the pretty `ggplot` graph, which yields the same answer.

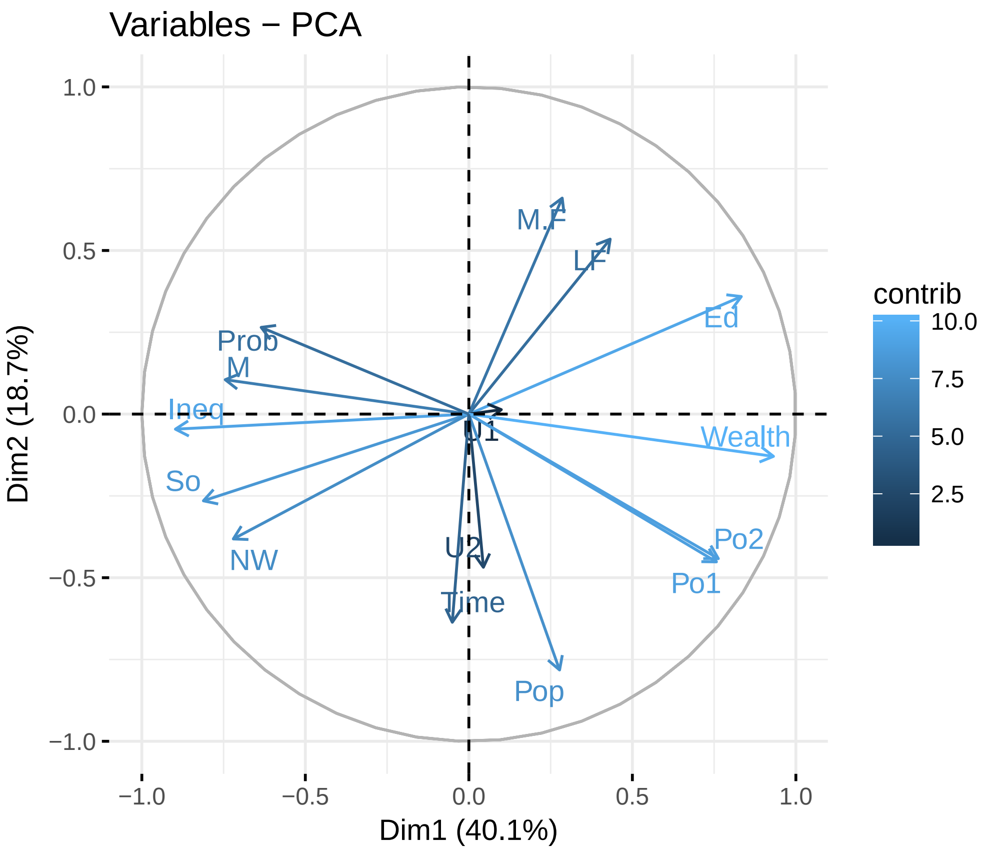
```
pcaVariance <- as.data.frame(summary(pcaModel2)$importance[3,])
PC <- 1:15
ggplot(pcaVariance, aes(x = factor(PC), y = pcaVariance, fill = factor(PC))) + geom_col() +
  labs(title = 'Cumulative Variance Accounted',
       subtitle = 'Additive Sum of Prior PCs',
       x = 'Principal components',
       y = 'Variance Accounted',
       fill = 'PC')
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```



- another interesting view from the `factoextra` library is the graph of variables and their direction on correlation
- Positive correlated variables point to the same side of the plot.
- Negative correlated variables point to opposite sides of the graph.

```
fviz_pca_var(pcaModel2, col.var = "contrib", repel = "TRUE")
```



- therefore, choosing the linear model based on the first 5 Principal components:

```
RegressionModelBasedonPCs_FINAL <- lm(Crime ~ PC1+PC2+PC3+PC4+PC5, data = CrimeDataWithPrincipalComponen
summary(RegressionModelBasedonPCs_FINAL)
```

```
##
## Call:
## lm(formula = Crime ~ PC1 + PC2 + PC3 + PC4 + PC5, data = CrimeDataWithPrincipalComponents)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -420.79 -185.01   12.21  146.24  447.86
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      35.59  25.428  < 2e-16 ***
## PC1            65.22      14.67   4.447 6.51e-05 ***
## PC2           -70.08      21.49  -3.261  0.00224 **
## PC3            25.19      25.41   0.992  0.32725
## PC4            69.45      33.37   2.081  0.04374 *
## PC5          -229.04      36.75  -6.232 2.02e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 244 on 41 degrees of freedom
## Multiple R-squared:  0.6452, Adjusted R-squared:  0.6019
## F-statistic: 14.91 on 5 and 41 DF,  p-value: 2.446e-08
```

- yes, the adjusted r square is lower now (60%) but these are the best PCs per the expained variances graph
- (what I cannot explain is the Prob of PC3 being .32)
- let's first multiply with the pcaModel2$rotation matrix to bring it back to original dimensions, and print the scaled coefficients:

```
RegCoefficientsForTop5PCs <- RegressionModelBasedonPCs_FINAL$coefficients[2:length(RegressionModelBased

ScaledFinalCoefficients <- RegCoefficientsForTop5PCs %*%head(t(pcaModel2$rotation), 5)

ScaledFinalCoefficients
```

```
##             M       So       Ed      Po1      Po2      LF      M.F
## [1,] 60.79435 37.84824 19.94776 117.3449 111.4508 76.2549 108.1266
##           Pop       NW       U1       U2   Wealth    Ineq      Prob
## [1,] 58.88024 98.07179 2.866783 32.34551 35.93336 22.1037 -34.64026
##          Time
## [1,] 27.20502
```

- now we unscale them back and print the output. This is the final output of our regression model.
- note: i'm assuming that the SCALED = true command used standard normalization theory, which is: x(normalized) = (x(real) - mean) / (standard.deviation)
- so the unscaling part is really doing everything in reverse, meaning: x(real) = x(normalized)*(std.dev) + mean

```
RegCoefficientsForTop5PCs <- RegressionModelBasedonPCs_FINAL$coefficients[2:length(RegressionModelBased

ScaledFinalCoefficients <- RegCoefficientsForTop5PCs %*%head(t(pcaModel2$rotation), 5)

#ScaledFinalCoefficients


meanForEachPredictor <- map_dbl(crimeDataTable[,1:15], mean)
standardDeviationForEachPredictor <- map_dbl(crimeDataTable[, 1:15], sd)

FinalCoefficients_Unscaled <- ScaledFinalCoefficients*standardDeviationForEachPredictor + meanForEachPr

print("Final, Unscaled coefficients")
```

```
## [1] "Final, Unscaled coefficients"
```

```
FinalCoefficients_Unscaled
```

```
##             M       So       Ed     Po1      Po2      LF      M.F      Pop
## [1,] 90.26156 18.46879 32.87938 357.237 319.6545 3.64279 416.9226 2278.258
##            NW        U1       U2   Wealth    Ineq       Prob     Time
## [1,] 1018.573 0.1471527 30.71511 39926.27 107.585 -0.7405233 219.3971
```

- final comments on the quality. this was the model from q8.2 (hw3) with a quality of 70% r squared
- the final formula used there was 8 predictors: formula = Crime ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq + Prob
- clearly our models quality from the pca metrics is only 60.6% of R squared if you scroll above, but we removed a lot of overfitting in the process
- our model does spit out all 15 predictors I was assuming the coefficients would have reduced the weighting for the useless predictors, but looking at the unscaled coefficients, none of them are tiny.

```
> summary(model2)

Call:
lm(formula = Crime ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq + Prob,
    data = crimeDataTable)

Residuals:
    Min      1Q  Median      3Q     Max
-444.70 -111.07    3.03  122.15  483.30

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -6426.10    1194.61  -5.379 4.04e-06 ***
M              93.32      33.50   2.786  0.00828 **
Ed            180.12      52.75   3.414  0.00153 **
Po1           102.65      15.52   6.613 8.26e-08 ***
M.F            22.34      13.60   1.642  0.10874
U1          -6086.63    3339.27  -1.823  0.07622 .
U2            187.35      72.48   2.585  0.01371 *
Ineq           61.33      13.96   4.394 8.63e-05 ***
Prob        -3796.03    1490.65  -2.547  0.01505 *

Residual standard error: 195.5 on 38 degrees of freedom
```

```
Multiple R-squared:  0.7888,    Adjusted R-squared:  0.7444
F-statistic: 17.74 on 8 and 38 DF,  p-value: 1.159e-10
```

Figure 1: snapshot from HW3

so looks like our model is using all 15 predictors in play. which is not a bad thing per se. I was just expecting the coefficients of atleast the 8 predictors we had selected in Q8.2 in HW3 would have been much lower..

- components derived from PCA come back in FULL force for all 15 components, if none of them is tiny (say scale of 10^-3) then model thinks each of them is germane to the prediction. That pretty much guarantees that the PCA model is over fitting.

Reference: a useful article explaining all the tips around scaling/centering etc:

LINK: PCA using prcomp and factoextra packages

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.[. . .] don't just stop when you have a good model, but interpret it too

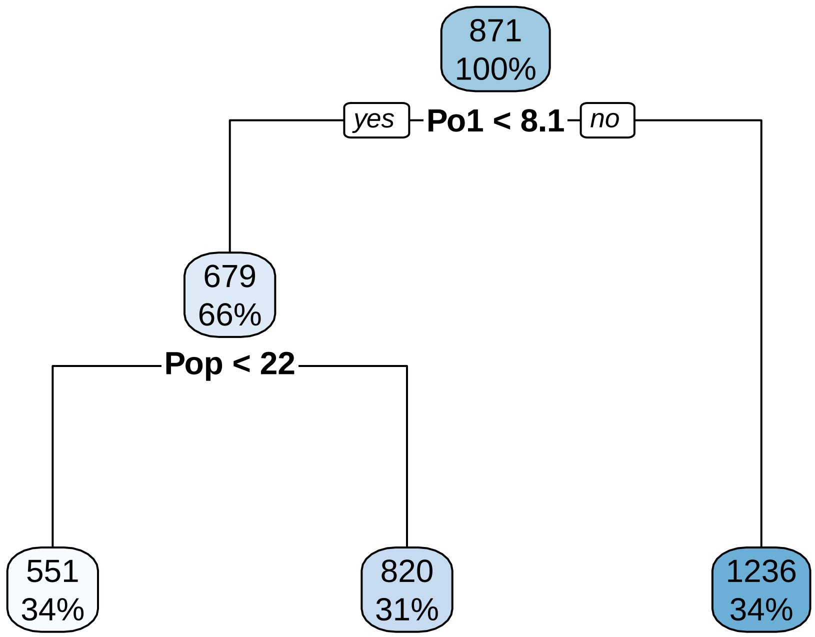
10.1a (regression tree)

- break the data apart from training and test

```
set.seed(1)
crimeDataTrainingIndices <- sample(nrow(crimeDataTable), size = floor(nrow(crimeDataTable)*0.7))
crimeDataTraining <- crimeDataTable[crimeDataTrainingIndices,]

# will split the remaining 30% into validation and testing data sets
restOfData <- crimeDataTable[-crimeDataTrainingIndices,]
crimeDataTesting <- restOfData[1:floor(nrow(restOfData)),]
#crimeDataTesting <- restOfData[ceiling(nrow(restOfData)*0.5):nrow(restOfData),]
```

- First we fit the regression tree function to the *training* crime data, essentially growing the tree based on recursive partitioning (big boy words taken from man page)

```
#treeData <- tree(Crime~., data = crimeDataTraining)
#summary(treeData)

treeDataFromRPart_JustTrainingData <- rpart(Crime~., data = crimeDataTraining)
rpart.plot(treeDataFromRPart_JustTrainingData)
```



- this table looks pretty sparse, and I suspect we lose valuable information by splitting like this
- However, since for this homework, the TAs are ok with splitting this sparse data so we can test the model with clean, independent data.
- the output above shows that the variables used are Po1, POP, LF, and NW:
- Po1 per capita expenditure on police protection in 1960. If its greater than 8.1M its classified separately vs when its less
- Pop state population in 1960 in hundred thousands
- going back to the training data

```
treeDataFromRPart_JustTrainingData$frame
```

```
##      var  n wt       dev      yval complexity ncompete nsurrogate
## 1    Po1 32 32 5092168.0  870.7500  0.4397941        4          5
## 2    Pop 21 21  815660.3  679.2857  0.0743391        4          5
## 4 <leaf> 11 11  243732.2  551.2727  0.0100000        0          0
## 5 <leaf> 10 10  193380.9  820.1000  0.0100000        0          0
## 3 <leaf> 11 11 2037002.2 1236.2727  0.0100000        0          0
```

- we are interested in leaf #4, leaf #5 and leaf #3
- lets now iterate through each branch of the tree and run linear regression on each to identify a good model
- we use the rpart.predict.leaves to cleanly separate the data for each leaf. this is part of the treeClust package

```
leaves <- rpart.predict.leaves(treeDataFromRPart_JustTrainingData, crimeDataTraining, type = "where" )

leaf4 <- vector()
leaf5 <- vector()
leaf3 <- vector()

for (valueWithinLeaf in 1:length(leaves)) {

  # figuring this out took really long!!!
```

```
  if(leaves[[valueWithinLeaf]] == 4) { leaf4 <- c(leaf4, valueWithinLeaf) }
  if(leaves[[valueWithinLeaf]] == 5) { leaf5 <- c(leaf5, valueWithinLeaf) }
  if(leaves[[valueWithinLeaf]] == 3) { leaf3 <- c(leaf3, valueWithinLeaf) }

}
```

- now that we have split the leaves out, we run the lm function on them:

```
leaf3LinearModel <- lm(Crime ~ ., data = crimeDataTraining[leaf3,])
leaf4LinearModel <- lm(Crime ~ ., data = crimeDataTraining[leaf4,])
leaf5LinearModel <- lm(Crime ~ ., data = crimeDataTraining[leaf5,])
```

- lets get a result of predicted responses from a fitted rpart object

```
predictedCrimeRatesLeaf3 <- predict(leaf3LinearModel, crimeDataTesting[,1:15], type = "response")
predictedCrimeRatesLeaf4 <- predict(leaf4LinearModel, crimeDataTesting[,1:15], type = "response")
predictedCrimeRatesLeaf5 <- predict(leaf4LinearModel, crimeDataTesting[,1:15], type = "response")


#predict(treeDataFromRPart_JustTrainingData, crimeDataTraining[,1:15], type = "vector")
#predictedCrimeRates
```

- lets compare the actual vs predicted data (I started to normalize this data, but realized there is no value in doing that since we are just evaluating the Crime parameter so there is no other scale to contend with)

```
#meanForCrime <- mean(crimeDataTable$Crime)
#sdForCrime <- sd(crimeDataTable$Crime)
#normalizedPredictedCrimeRates <- (predictedCrimeRates - meanForCrime)/ sdForCrime

differenceInPredictedAndActualInPercentage_leaf3 <- (predictedCrimeRatesLeaf3 - crimeDataTesting$Crime)/

errorPercentageForLeaf3 <- mean(abs(differenceInPredictedAndActualInPercentage_leaf3))
cat("The error % between predicted and real (testing) data for leaf3 is", errorPercentageForLeaf3, "\n")
```

```
## The error % between predicted and real (testing) data for leaf3 is 1.956582
```

```
differenceInPredictedAndActualInPercentage_leaf4 <- (predictedCrimeRatesLeaf4 - crimeDataTesting$Crime)/

errorPercentageForLeaf4 <- mean(abs(differenceInPredictedAndActualInPercentage_leaf4))

cat("The error % between predicted and real (testing) data for leaf4 is", errorPercentageForLeaf4, "\n")
```

```
## The error % between predicted and real (testing) data for leaf4 is 1.14088
```

```
differenceInPredictedAndActualInPercentage_leaf5 <- (predictedCrimeRatesLeaf5 - crimeDataTesting$Crime)/

errorPercentageForLeaf5 <- mean(abs(differenceInPredictedAndActualInPercentage_leaf5))

cat("The error % between predicted and real (testing) data for leaf5 is", errorPercentageForLeaf5, "\n")
```

```
## The error % between predicted and real (testing) data for leaf5 is 1.14088
```

- essentially 21% is the average difference of each predicted vs actual crime value using the tree method.
- now let's run the random forest: (i use the type = "response" here) **If object$type is classification, the object returned depends on the argument type: response - predicted classes (the classes with majority vote).**

```r
numberOfPredictorsToConsider <- 4
randomForestData <- randomForest(Crime ~. , data = crimeDataTable, mtry = numberOfPredictorsToConsider,

predictedCrimeRatesUsingForest <- predict(randomForestData, crimeDataTable[,1:15], type = "response")

predictedCrimeRatesUsingForest
```

```
##         1         2         3         4         5         6         7
##  775.3397 1409.2062  592.0599 1653.0481 1142.8008  974.4300  955.2319
##         8         9        10        11        12        13        14
## 1332.7955  825.3619  719.6603 1532.3647  831.2218  596.5663  668.3178
##        15        16        17        18        19        20        21
##  770.3942  937.0739  588.8492  966.7051  904.1522 1198.0483  787.7971
##        22        23        24        25        26        27        28
##  556.4696 1137.3089  931.2019  588.6606 1631.6596  561.2467 1139.4348
##        29        30        31        32        33        34        35
## 1195.7711  744.7713  522.1789  883.9358  942.8059  930.8134  850.1271
##        36        37        38        39        40        41        42
## 1207.1365  808.3832  582.5623  797.8942 1146.8862  861.4706  538.5456
##        43        44        45        46        47
##  841.0936 1043.6361  511.9112  730.2876  891.7954
```

- now let's compare predicted vs real for the forest output:

```r
# meanForCrime <- mean(crimeDataTable$Crime)
#sdForCrime <- sd(crimeDataTable$Crime)

#normalizedPredictedCrimeRates <- (predictedCrimeRates - meanForCrime)/ sdForCrime

differenceInPredictedAndActualInPercentage_FOREST <- (predictedCrimeRatesUsingForest - crimeDataTable$C

mean(abs(differenceInPredictedAndActualInPercentage_FOREST))
```

```
## [1] 0.1110734
```

- essentially 10% is the average difference of each predicted vs actual crime value using the tree method.
- so clearly the random forest method does a better job in prediction (10% vs 21%) than the solitary tree method.

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.

The whole value of logistic regression is the ability to predict the PROBABILITY of an event happen . The event result is also binary furthermore. Meaning, either it will happen or it won't happen. I would apply this in my startup context as follows:

I am a co-founder of waada (http://waada.org). The purpose of this non profit is to help folks with mental illness using technology. The reason why CUSUM / Change Detection is so germane to this organization is that I can can use the CUSUM algorithm to detect mood changes. Unless the person has bi-polar depression , where the changes are obvious, depression in people who are prone to it creeps in gradually until its too late for the care giver to make an impact. In this situation, the "slippery slope" hits the depressed person and they stay depressed for weeks or months. Sometimes crude measures like medicine have to be taken to lift them out, but those are mostly artificial and there is no way to measure the exact quantity to be taken by the person to get better since measuring the "extent" of depression is so subjective. Therefore there is almost always a slight overdose of the medicine, which in the long term is severely adverse to the health of the patient, since he or she invariably becomes dependent on that medicine, akin to a drug addict.

The concept I have is as follows. Suppose we are able to take in physiological information (heartbeats through fitbits or iWatch wearables), phone mobility (through its gyroscope), # of calls made, length of calls, we can start creating a pattern around this person. We can also allow this person to directly enter into the phone via an app if they are feeling down or not (taking care to give something back in return, like a calming remedy , a song, breathing techniques etc so we can motivate the person to enter the data). I would take the input from each of these predictors, and calculate via logistic regression a way to detect the probability that a person has fallen into depression so that remedial actions can be taken for immediate efficacy.

Prompt identification of the onset of depression would allow us to engage in proactive measures like involving the caregiver much sooner, or providing special services through the mobile app around improving breathing techniques and a more engaged package of activities. if the onset is pretty severe, healthcare and even emergency services (suicide hotline) could be put on notice.

Question 10.3.1

Using the GermanCredit data set 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.

```
dataFile <- "germancredit.txt"
if (!file.exists(dataFile)) {
  germanCreditURL <- paste0(c("https://prod-edxapp.edx-cdn.org/assets/courseware/v1/a145a478beb6f64b59e
  download.file(germanCreditURL, dataFile) }

germanCreditTable <- read.table(dataFile, header = FALSE )

# changing the response variabl to 0 or 1 since glm binomial will be expecting that...
germanCreditTable$V21[germanCreditTable$V21 == 1] <- 0
germanCreditTable$V21[germanCreditTable$V21 == 2] <- 1
```

- use the legend here to undersand this data: http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29
- first we use the glm model...

```
logisticRegModel <- glm(V21 ~ . , family = binomial(link = "logit"), data = germanCreditTable)

summary(logisticRegModel)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = germanCreditTable)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3410  -0.6994  -0.3752   0.7095   2.6116
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.005e-01  1.084e+00   0.369 0.711869
```

```
## V1A12       -3.749e-01  2.179e-01  -1.720 0.085400 .
## V1A13       -9.657e-01  3.692e-01  -2.616 0.008905 **
## V1A14       -1.712e+00  2.322e-01  -7.373 1.66e-13 ***
## V2           2.786e-02  9.296e-03   2.997 0.002724 **
## V3A31        1.434e-01  5.489e-01   0.261 0.793921
## V3A32       -5.861e-01  4.305e-01  -1.362 0.173348
## V3A33       -8.532e-01  4.717e-01  -1.809 0.070470 .
## V3A34       -1.436e+00  4.399e-01  -3.264 0.001099 **
## V4A41       -1.666e+00  3.743e-01  -4.452 8.51e-06 ***
## V4A410      -1.489e+00  7.764e-01  -1.918 0.055163 .
## V4A42       -7.916e-01  2.610e-01  -3.033 0.002421 **
## V4A43       -8.916e-01  2.471e-01  -3.609 0.000308 ***
## V4A44       -5.228e-01  7.623e-01  -0.686 0.492831
## V4A45       -2.164e-01  5.500e-01  -0.393 0.694000
## V4A46        3.628e-02  3.965e-01   0.092 0.927082
## V4A48       -2.059e+00  1.212e+00  -1.699 0.089297 .
## V4A49       -7.401e-01  3.339e-01  -2.216 0.026668 *
## V5           1.283e-04  4.444e-05   2.887 0.003894 **
## V6A62       -3.577e-01  2.861e-01  -1.250 0.211130
## V6A63       -3.761e-01  4.011e-01  -0.938 0.348476
## V6A64       -1.339e+00  5.249e-01  -2.551 0.010729 *
## V6A65       -9.467e-01  2.625e-01  -3.607 0.000310 ***
## V7A72       -6.691e-02  4.270e-01  -0.157 0.875475
## V7A73       -1.828e-01  4.105e-01  -0.445 0.656049
## V7A74       -8.310e-01  4.455e-01  -1.866 0.062110 .
## V7A75       -2.766e-01  4.134e-01  -0.669 0.503410
## V8           3.301e-01  8.828e-02   3.739 0.000185 ***
## V9A92       -2.755e-01  3.865e-01  -0.713 0.476040
## V9A93       -8.161e-01  3.799e-01  -2.148 0.031718 *
## V9A94       -3.671e-01  4.537e-01  -0.809 0.418448
## V10A102      4.360e-01  4.101e-01   1.063 0.287700
## V10A103     -9.786e-01  4.243e-01  -2.307 0.021072 *
## V11          4.776e-03  8.641e-02   0.055 0.955920
## V12A122      2.814e-01  2.534e-01   1.111 0.266630
## V12A123      1.945e-01  2.360e-01   0.824 0.409743
## V12A124      7.304e-01  4.245e-01   1.721 0.085308 .
## V13         -1.454e-02  9.222e-03  -1.576 0.114982
## V14A142     -1.232e-01  4.119e-01  -0.299 0.764878
## V14A143     -6.463e-01  2.391e-01  -2.703 0.006871 **
## V15A152     -4.436e-01  2.347e-01  -1.890 0.058715 .
## V15A153     -6.839e-01  4.770e-01  -1.434 0.151657
## V16          2.721e-01  1.895e-01   1.436 0.151109
## V17A172      5.361e-01  6.796e-01   0.789 0.430160
## V17A173      5.547e-01  6.549e-01   0.847 0.397015
## V17A174      4.795e-01  6.623e-01   0.724 0.469086
## V18          2.647e-01  2.492e-01   1.062 0.288249
## V19A192     -3.000e-01  2.013e-01  -1.491 0.136060
## V20A202     -1.392e+00  6.258e-01  -2.225 0.026095 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1221.73  on 999  degrees of freedom
```

```
## Residual deviance:  895.82  on 951  degrees of freedom
## AIC: 993.82
##
## Number of Fisher Scoring iterations: 5
```

- the above are way too many coefficients, and GLM doesn't give r squared either
- recall Dr Sokol's lecture that R sq is not really possible for Logistic regression
- searching on the net, found them rms (heavy) package which provides a pseudo quality of fit
- first we carve the data out to training and test data, train the model using the former

```r
set.seed(1)
germanDataTrainingIndices <- sample(nrow(germanCreditTable), size = floor(nrow(germanCreditTable)*0.7))
germanDataTraining <- germanCreditTable[germanDataTrainingIndices,]

# will split the remaining 30% into validation and testing data sets
restOfData <- germanCreditTable[-germanDataTrainingIndices,]
germanDataTesting <- restOfData[1:floor(nrow(restOfData)),]
#crimeDataTesting <- restOfData[ceiling(nrow(restOfData)*0.5):nrow(restOfData),]

fitlogisticRegModel_RMS <- lrm(V21 ~ . ,  data = germanDataTraining)

print(fitlogisticRegModel_RMS)
```

```
## Logistic Regression Model
##
##  lrm(formula = V21 ~ ., data = germanDataTraining)
##
##                        Model Likelihood     Discrimination    Rank Discrim.
##                           Ratio Test           Indexes           Indexes
##  Obs           700    LR chi2     263.68    R2       0.447    C       0.857
##   0            495    d.f.            48    g        2.107    Dxy     0.714
##   1            205    Pr(> chi2) <0.0001    gr       8.225    gamma   0.714
##  max |deriv| 2e-09                          gp       0.296    tau-a   0.296
##                                             Brier    0.135
##
##            Coef    S.E.   Wald Z Pr(>|Z|)
##  Intercept  1.7188 1.3649  1.26  0.2079
##  V1=A12    -0.1509 0.2780 -0.54  0.5872
##  V1=A13    -1.7290 0.5490 -3.15  0.0016
##  V1=A14    -1.4231 0.2886 -4.93  <0.0001
##  V2         0.0340 0.0115  2.97  0.0030
##  V3=A31    -0.5534 0.7179 -0.77  0.4408
##  V3=A32    -0.9670 0.5689 -1.70  0.0892
##  V3=A33    -1.2999 0.5929 -2.19  0.0284
##  V3=A34    -2.0783 0.5780 -3.60  0.0003
##  V4=A41    -2.2394 0.5122 -4.37  <0.0001
##  V4=A410   -3.1461 1.0935 -2.88  0.0040
##  V4=A42    -1.1240 0.3287 -3.42  0.0006
##  V4=A43    -0.9062 0.3064 -2.96  0.0031
##  V4=A44    -0.2663 0.8437 -0.32  0.7523
##  V4=A45    -0.5171 0.8131 -0.64  0.5248
##  V4=A46     0.0097 0.4976  0.02  0.9844
##  V4=A48    -2.0125 1.2200 -1.65  0.0990
##  V4=A49    -1.0071 0.4283 -2.35  0.0187
##  V5         0.0001 0.0001  2.14  0.0324
```

```
##  V6=A62    -0.6178 0.3644 -1.70  0.0900
##  V6=A63    -0.8173 0.5472 -1.49  0.1353
##  V6=A64    -0.6326 0.5896 -1.07  0.2833
##  V6=A65    -1.2840 0.3474 -3.70  0.0002
##  V7=A72     0.0598 0.5171  0.12  0.9079
##  V7=A73    -0.3262 0.4931 -0.66  0.5082
##  V7=A74    -0.9421 0.5448 -1.73  0.0838
##  V7=A75    -0.5208 0.5104 -1.02  0.3075
##  V8         0.3228 0.1150  2.81  0.0050
##  V9=A92    -0.5424 0.4800 -1.13  0.2585
##  V9=A93    -0.9459 0.4679 -2.02  0.0432
##  V9=A94    -0.7790 0.5753 -1.35  0.1757
##  V10=A102   0.4178 0.4918  0.85  0.3956
##  V10=A103  -1.0792 0.5364 -2.01  0.0442
##  V11        0.0158 0.1098  0.14  0.8856
##  V12=A122   0.6148 0.3220  1.91  0.0563
##  V12=A123   0.4082 0.2999  1.36  0.1735
##  V12=A124   1.2284 0.5467  2.25  0.0246
##  V13       -0.0168 0.0118 -1.42  0.1551
##  V14=A142  -0.5909 0.5155 -1.15  0.2517
##  V14=A143  -1.0860 0.2972 -3.65  0.0003
##  V15=A152  -0.7838 0.3021 -2.59  0.0095
##  V15=A153  -1.3504 0.6194 -2.18  0.0292
##  V16        0.4381 0.2499  1.75  0.0795
##  V17=A172   0.4812 0.8505  0.57  0.5715
##  V17=A173   0.4486 0.8190  0.55  0.5838
##  V17=A174   0.2924 0.8288  0.35  0.7243
##  V18        0.0321 0.3177  0.10  0.9196
##  V19=A192  -0.2567 0.2560 -1.00  0.3160
##  V20=A202  -1.6784 0.8816 -1.90  0.0569
##
```

- as we can see above the R^2 computed is ~ 44.7% only. this model is not a great quality of fit.

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.

From the data set details: This dataset requires use of a cost matrix (see below)

. . . .. 1 2

1 0 1

2 5 0

(1 = Good, 2 = Bad)

The rows represent the actual classification and the columns the predicted classification.

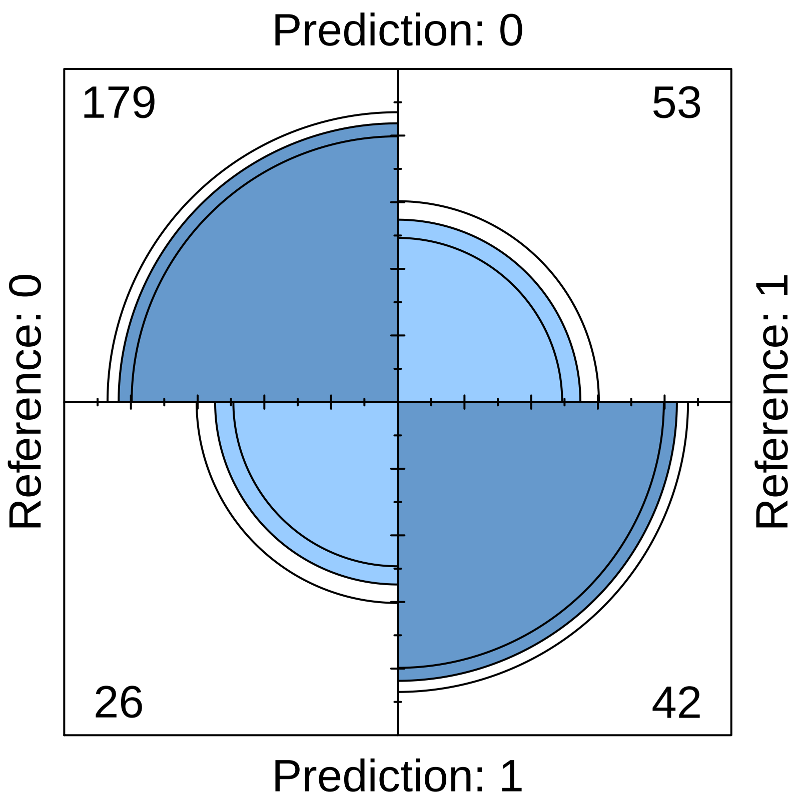
It is worse to class a customer as good when they are bad (5), than it is to class a customer as bad when they are good (1).

- now we calculate the confusion matrix, and plot it:

```
predictedGermanCreditResult_TESTDATA <- predict(fitlogisticRegModel_RMS, germanDataTesting[, 1:20])

probs <- round(exp(predictedGermanCreditResult_TESTDATA)/(1+exp(predictedGermanCreditResult_TESTDATA)))

cMatrix <- confusionMatrix(
  factor(probs),
  factor(germanDataTesting[, 21])
)
cMatrix$table
```

```
##           Reference
## Prediction   0   1
##          0 179  53
##          1  26  42
```

```
TP <- cMatrix$table[1,1]
FP <- cMatrix$table[1,2]
FN <- cMatrix$table[2,1]
TN <- cMatrix$table[2,2]

fourfoldplot(cMatrix$table)
```



- incorrectly identifying a bad customer as good (FP) is 5x WORSE than identirying a good customer as bad (TN)
- there is 0 cost identifying a bad customer as bad (FN) and good customer as good (TP)

- just an example to read this FP, FN mumbo jumbo:
- FP = false positive, meaning the model thinks is positive, or a good customer, but in reality its false, i.e. , its a bad customer
- therefore the overall cost is calculated as:

```
Cost <- 0*TP + 1*TN + 0*FN + 5*FP

Cost
```

```
## [1] 307
```

- to figure out the cutoff, I used the ROCR package (as per a piazza post):

```
pred <- prediction(probs,germanDataTesting[, 21])
cost.perf = performance(pred, "cost")
#pred@cutoffs[[1]][which.min(cost.perf@y.values[[1]])]

cost.perf@y.values
```

```
## [[1]]
## [1] 0.3166667 0.2633333 0.6833333
```

- from the result we can see that the minimum cost is 0.2633. that is what i'd set as the threshold