Problem Set 5

Jacob Miller, Jacob Shiohira, Reid Gahan

4/14/2018

Problem 1

```
# - Read in the Framinham data set
df <- read.csv("Data/framingham.csv", stringsAsFactors = FALSE)

# - How many observations? How many features (explanatory variables)?
str(df)
```

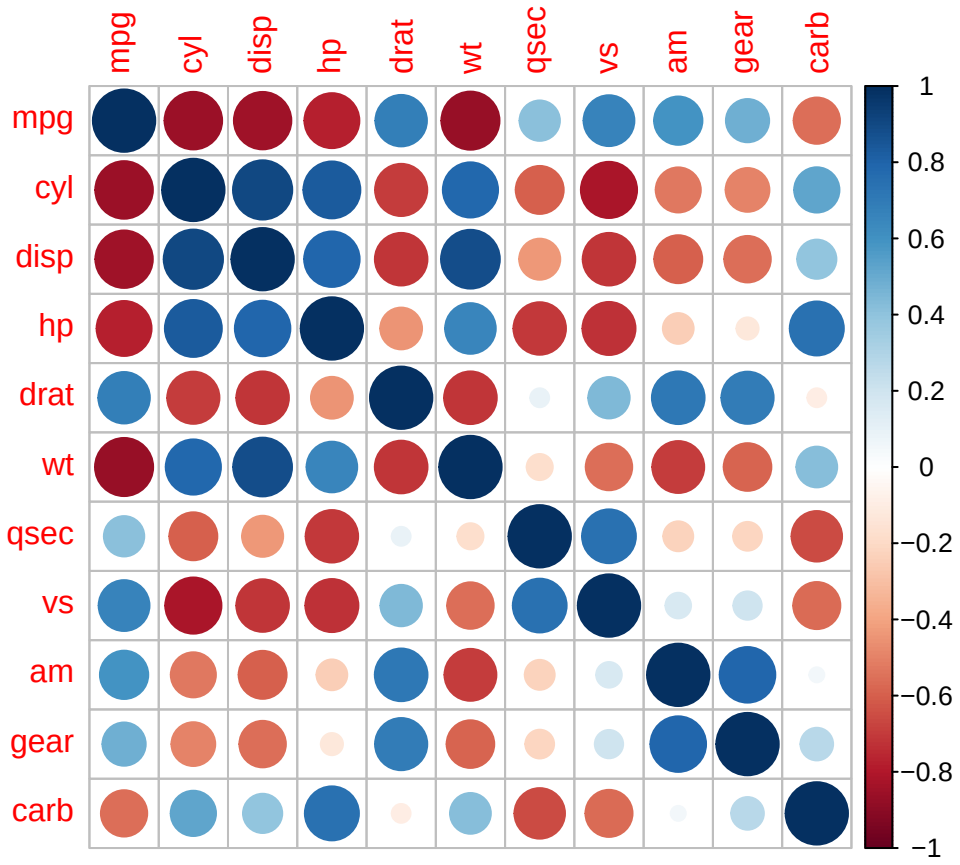
```
## 'data.frame':    4240 obs. of  16 variables:
##  $ male           : int  1 0 1 0 0 0 0 0 1 1 ...
##  $ age            : int  39 46 48 61 46 43 63 45 52 43 ...
##  $ education      : int  4 2 1 3 3 2 1 2 1 1 ...
##  $ currentSmoker  : int  0 0 1 1 1 0 0 1 0 1 ...
##  $ cigsPerDay     : int  0 0 20 30 23 0 0 20 0 30 ...
##  $ BPMeds         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ prevalentStroke: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ prevalentHyp   : int  0 0 0 1 0 1 0 0 1 1 ...
##  $ diabetes       : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ totChol        : int  195 250 245 225 285 228 205 313 260 225 ...
##  $ sysBP          : num  106 121 128 150 130 ...
##  $ diaBP          : num  70 81 80 95 84 110 71 71 89 107 ...
##  $ BMI            : num  27 28.7 25.3 28.6 23.1 ...
##  $ heartRate      : int  80 95 75 65 85 77 60 79 76 93 ...
##  $ glucose        : int  77 76 70 103 85 99 85 78 79 88 ...
##  $ TenYearCHD     : int  0 0 0 1 0 0 1 0 0 0 ...
```

The output above tells us there are a total of $4,240$ observations and 16 features in the dataset. The total number of observations in this case includes rows with possible NAN values.

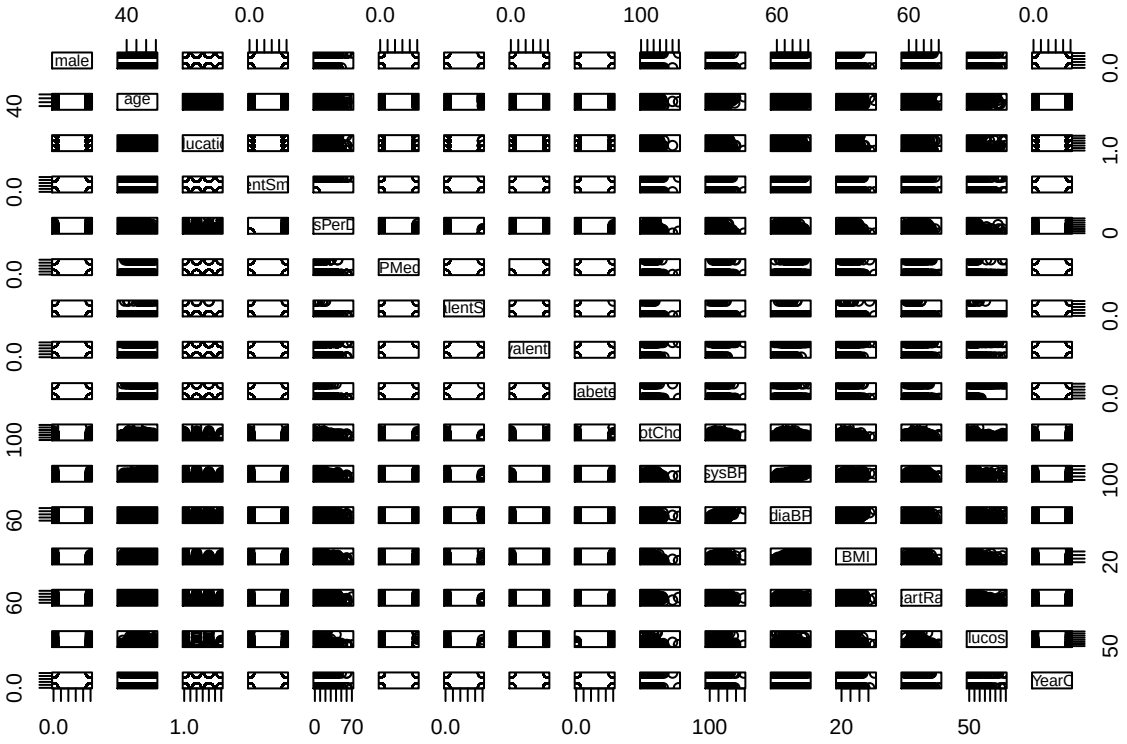
```r
summary(df)
```

```
##       male             age          education     currentSmoker
##  Min.   :0.0000   Min.   :32.00   Min.   :1.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:42.00   1st Qu.:1.000   1st Qu.:0.0000
##  Median :0.0000   Median :49.00   Median :2.000   Median :0.0000
##  Mean   :0.4292   Mean   :49.58   Mean   :1.979   Mean   :0.4941
##  3rd Qu.:1.0000   3rd Qu.:56.00   3rd Qu.:3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :70.00   Max.   :4.000   Max.   :1.0000
##                                   NA's   :105
##    cigsPerDay        BPMeds       prevalentStroke     prevalentHyp
##  Min.   : 0.000   Min.   :0.00000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.: 0.000   1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.0000
##  Median : 0.000   Median :0.00000   Median :0.000000   Median :0.0000
##  Mean   : 9.006   Mean   :0.02962   Mean   :0.005896   Mean   :0.3106
##  3rd Qu.:20.000   3rd Qu.:0.00000   3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :70.000   Max.   :1.00000   Max.   :1.000000   Max.   :1.0000
##  NA's   :29       NA's   :53
##     diabetes          totChol          sysBP           diaBP
##  Min.   :0.00000   Min.   :107.0   Min.   : 83.5   Min.   : 48.0
##  1st Qu.:0.00000   1st Qu.:206.0   1st Qu.:117.0   1st Qu.: 75.0
##  Median :0.00000   Median :234.0   Median :128.0   Median : 82.0
##  Mean   :0.02571   Mean   :236.7   Mean   :132.4   Mean   : 82.9
##  3rd Qu.:0.00000   3rd Qu.:263.0   3rd Qu.:144.0   3rd Qu.: 90.0
##  Max.   :1.00000   Max.   :696.0   Max.   :295.0   Max.   :142.5
##                    NA's   :50
##       BMI          heartRate         glucose         TenYearCHD
##  Min.   :15.54   Min.   : 44.00   Min.   : 40.00   Min.   :0.0000
##  1st Qu.:23.07   1st Qu.: 68.00   1st Qu.: 71.00   1st Qu.:0.0000
##  Median :25.40   Median : 75.00   Median : 78.00   Median :0.0000
##  Mean   :25.80   Mean   : 75.88   Mean   : 81.96   Mean   :0.1519
##  3rd Qu.:28.04   3rd Qu.: 83.00   3rd Qu.: 87.00   3rd Qu.:0.0000
##  Max.   :56.80   Max.   :143.00   Max.   :394.00   Max.   :1.0000
##  NA's   :19      NA's   :1        NA's   :388
```

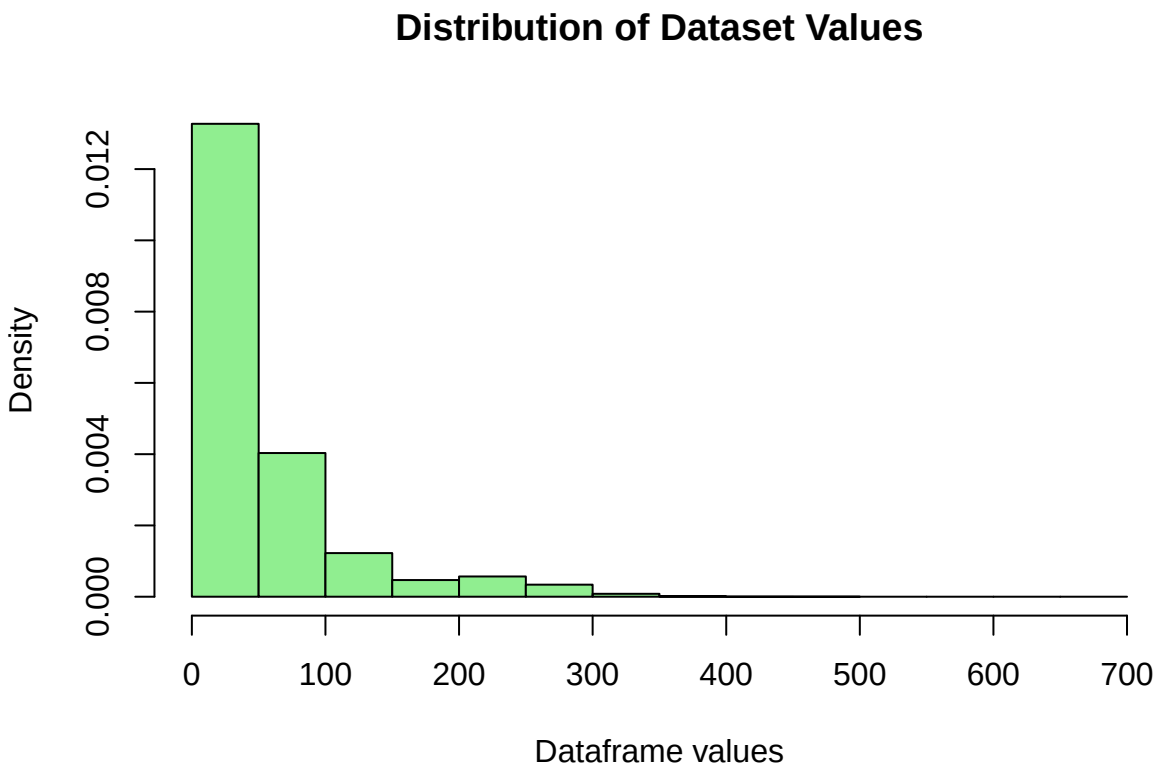
```r
# - Explore the data (plots, tables, statistics) including feature types.
M <- cor(mtcars)
corrplot(M, method = "circle")
```


mpg
cyl
disp
hp
drat
wt
qsec
vs
am
gear
carb
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

```
pairs(df)
```




```
df_as_numeric <- sapply(df, as.numeric)
hist(df_as_numeric, freq=FALSE, xlab="Dataframe values", main="Distribution of Dataset Values", col="lig
```




```
# Count the total number of rows with at least one NAN value
df_nonnan <- na.omit(df)
num_null_rows <- nrow(df) - nrow(df_nonnan)
```

```
## [1] "Total number of rows with at least one NAN value:  582"
```

```
# - Summarize any NA's by feature
na_count <-sapply(df, function(y) sum(length(which(is.na(y)))))
na_count <- data.frame(na_count)
```

Below is a table outlining the number of NAN Values per feature. Note that the sum of NAN values across features is greater than the total number of rows with at least one NAN value because a single row could have multiple NAN values.

```
##                  na_count
## male                    0
## age                     0
## education             105
## currentSmoker           0
## cigsPerDay             29
## BPMeds                 53
## prevalentStroke         0
## prevalentHyp            0
## diabetes                0
## totChol                50
## sysBP                   0
## diaBP                   0
## BMI                    19
```

```
## heartRate              1
## glucose              388
## TenYearCHD             0
```

Problem 2

```
n <- nrow(df)

# Documentation: https://www.rdocumentation.org/packages/mice/versions/2.25/topics/mice
# Using mice: http://web.maths.unsw.edu.au/~dwarton/missingDataLab.html, search for "pmm" method
# NOTE: changing the maxit param will affect the time it takes for mice() to run
imputed_df <- complete(mice(df, m=5, maxit=50, meth="pmm", printFlag=FALSE, seed=101))
summary(imputed_df)
```

```
##       male             age          education     currentSmoker
##  Min.   :0.0000   Min.   :32.00   Min.   :1.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:42.00   1st Qu.:1.000   1st Qu.:0.0000
##  Median :0.0000   Median :49.00   Median :2.000   Median :0.0000
##  Mean   :0.4292   Mean   :49.58   Mean   :1.979   Mean   :0.4941
##  3rd Qu.:1.0000   3rd Qu.:56.00   3rd Qu.:3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :70.00   Max.   :4.000   Max.   :1.0000
##    cigsPerDay         BPMeds        prevalentStroke     prevalentHyp
##  Min.   : 0.000   Min.   :0.00000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.: 0.000   1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.0000
##  Median : 0.000   Median :0.00000   Median :0.000000   Median :0.0000
##  Mean   : 9.051   Mean   :0.03019   Mean   :0.005896   Mean   :0.3106
##  3rd Qu.:20.000   3rd Qu.:0.00000   3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :70.000   Max.   :1.00000   Max.   :1.000000   Max.   :1.0000
##     diabetes          totChol          sysBP            diaBP
##  Min.   :0.00000   Min.   :107.0   Min.   : 83.5   Min.   : 48.0
##  1st Qu.:0.00000   1st Qu.:206.0   1st Qu.:117.0   1st Qu.: 75.0
##  Median :0.00000   Median :234.0   Median :128.0   Median : 82.0
##  Mean   :0.02571   Mean   :236.7   Mean   :132.4   Mean   : 82.9
##  3rd Qu.:0.00000   3rd Qu.:263.0   3rd Qu.:144.0   3rd Qu.: 90.0
##  Max.   :1.00000   Max.   :696.0   Max.   :295.0   Max.   :142.5
##       BMI           heartRate         glucose         TenYearCHD
##  Min.   :15.54   Min.   : 44.00   Min.   : 40.00   Min.   :0.0000
##  1st Qu.:23.07   1st Qu.: 68.00   1st Qu.: 71.00   1st Qu.:0.0000
##  Median :25.39   Median : 75.00   Median : 78.00   Median :0.0000
##  Mean   :25.80   Mean   : 75.88   Mean   : 81.86   Mean   :0.1519
##  3rd Qu.:28.04   3rd Qu.: 83.00   3rd Qu.: 87.00   3rd Qu.:0.0000
##  Max.   :56.80   Max.   :143.00   Max.   :394.00   Max.   :1.0000
```

```
# -- Uniformly then Normally scaling the dataset. Reference the histogram plots to verify the best type
UnifDF <- ScaleUnif(imputed_df)
NormDF <- data.frame(apply(imputed_df, MARGIN = 2, FUN = function(X) (X - min(X))/diff(range(X))))
```

```
# -- Shuffle the dataset then split data with 80% in the training set and 20% in the test set.
set.seed(1)
shuffled_df <- NormDF[sample(n),]
trainset <- shuffled_df[1:round(0.8 * n), ]
testset <- shuffled_df[(round(0.8 * n) + 1):n, ]

# Baseline model - predict the mean of the training data with the TenYearCHD binary response variable
```

```r
trainset_mean <- mean(trainset$TenYearCHD)

# Evaluate RMSE and MAE on the testing data
RMSE_baseline <- sqrt(mean((trainset_mean-testset$TenYearCHD)^2))
psp("Baseline RMSE: ", RMSE_baseline, 2)
```

```
## [1] "Baseline RMSE:  35.57 %"
```

```r
MAE_baseline <- mean(abs(trainset_mean-testset$TenYearCHD))
psp("Baseline Mean Absolute Error: ", MAE_baseline, 2)
```

```
## [1] "Baseline Mean Absolute Error:  25.59 %"
```

```r
# Apply the "activation function" to the mean of the training set to use in all of the statistics
trainset_mean <- round(trainset_mean)

# -- What is the postive predictive rate (precision) of your baseline prediction ?
TenYearCHDAsTable <- table(testset$TenYearCHD)

truepositive <- TenYearCHDAsTable[names(TenYearCHDAsTable)==trainset_mean]
falsepositive <- TenYearCHDAsTable[names(TenYearCHDAsTable)!=trainset_mean]

psp("Postive Predictive Rate (precision): ", (truepositive / (truepositive + falsepositive)), 2)
```

```
## [1] "Postive Predictive Rate (precision):  85.14 %"
```

```r
baseline_accuracy <- Metrics::accuracy(testset$TenYearCHD, trainset_mean)
psp("Baseline Accuracy: ", baseline_accuracy, 2)
```

```
## [1] "Baseline Accuracy:  85.14 %"
```

Question: Which error type seems worse and thus should be considered along with accuracy in your predictions below? Why? \

The worse type of error here is not predicting cancer when there is actually cancer. It is possible that a model could have very high accuracy but just guess all 0's, as demonstrated by our baseline model. However, the importance in an application like this is the ability to precisely predict positive cases. Thus, the precision value, or Postive Predictive Rate, should also cbe considered alongside accuracy in the predictions below.

Problem 3

A note on the output from `anova(...)`. The difference between the null deviance and the residual deviance shows how our model is doing against the null model (a model with only the intercept). The wider this gap, the better. Analyzing the table we can see the drop in deviance when adding each variable one at a time.

```r
# Logistic Regression: http://ww2.coastal.edu/kingw/statistics/R-tutorials/logistic.html

# - Create a 'NULL' logistic regression model with glm() using only the intercept feature. Use glm's bi
modelNull <- glm(TenYearCHD ~ 1, family=binomial(link='logit'), data=trainset)
summary(modelNull)
```

```
##
## Call:
## glm(formula = TenYearCHD ~ 1, family = binomial(link = "logit"),
##     data = trainset)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -0.5757  -0.5757  -0.5757  -0.5757   1.9387
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.71348    0.04773   -35.9   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2899.4  on 3391  degrees of freedom
## Residual deviance: 2899.4  on 3391  degrees of freedom
## AIC: 2901.4
##
## Number of Fisher Scoring iterations: 3
```

```
modelNullAIC <- AIC(modelNull)
psi("Null Model AIC: ", modelNullAIC)
```

```
## [1] "Null Model AIC:  2901.38036502798"
```

```
modelNullLogLike <- logLik(modelNull)
psi("Null Model Log likelihood: ", modelNullLogLike)
```

```
## [1] "Null Model Log likelihood:  -1449.69018251399"
```

```
modelNullDeviance <- modelNull$deviance
psi("Null Model Deviance: ", modelNullDeviance)
```

```
## [1] "Null Model Deviance:  2899.38036502798"
```

```
# - Create a 'FULL' logistic regression model with glm() using all of the features including the interc
modelFull <- glm(TenYearCHD ~., family=binomial(link='logit'), data=trainset)
summary(modelFull)
```

```
##
## Call:
## glm(formula = TenYearCHD ~ ., family = binomial(link = "logit"),
##     data = trainset)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0024  -0.6002  -0.4327  -0.2927   2.8099
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -4.46083    0.32696 -13.643  < 2e-16 ***
## male             0.56903    0.11190   5.085 3.67e-07 ***
## age              2.34368    0.26209   8.942  < 2e-16 ***
## education       -0.05469    0.15160  -0.361 0.718301
## currentSmoker   -0.15341    0.16448  -0.933 0.350976
## cigsPerDay       1.67288    0.45760   3.656 0.000256 ***
## BPMeds           0.19016    0.24179   0.786 0.431589
## prevalentStroke  0.89000    0.46135   1.929 0.053716 .
## prevalentHyp     0.25135    0.14516   1.732 0.083360 .
## diabetes        -0.04598    0.33264  -0.138 0.890051
```

```
## totChol          0.57207    0.67625   0.846 0.397578
## sysBP            2.93981    0.83564   3.518 0.000435 ***
## diaBP           -0.47517    0.63390  -0.750 0.453495
## BMI              0.29419    0.54432   0.540 0.588871
## heartRate       -0.15189    0.42717  -0.356 0.722157
## glucose          2.57503    0.80967   3.180 0.001471 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2899.4  on 3391  degrees of freedom
## Residual deviance: 2581.1  on 3376  degrees of freedom
## AIC: 2613.1
##
## Number of Fisher Scoring iterations: 5
```

```
# Calculates and outputs logLik(model), model$deviance, and AIC(model) for full model
modelFullAIC <- AIC(modelFull)
psi("Full Model AIC: ", modelFullAIC)
```

```
## [1] "Full Model AIC:  2613.14640639939"
```

```
modelFullLogLike <- logLik(modelFull)
psi("Full Model Log likelihood: ", modelFullLogLike)
```

```
## [1] "Full Model Log likelihood:  -1290.5732031997"
```

```
modelFullDeviance <- modelFull$deviance
psi("Full Model Deviance: ", modelFullDeviance)
```

```
## [1] "Full Model Deviance:  2581.14640639939"
```

Compare and comment on the AUC of the models: By looking at AIC values, the 'full' logistic regression model using all the features is the better model. The null model has an AIC of 2901 while the full model has one of 2613.

Problem 4

```
# - Create the lowest AIC ('best') model using backward elimination of parameters.
modelBest <- step(modelFull, direction = c("backward"), trace = 0)

# - Print the summary(), the log likelihood, the deviance, and the AIC for this model and compare with
summary(modelBest)
```

```
##
## Call:
## glm(formula = TenYearCHD ~ male + age + cigsPerDay + prevalentStroke +
##     prevalentHyp + sysBP + glucose, family = binomial(link = "logit"),
##     data = trainset)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0233  -0.5955  -0.4367  -0.2986   2.7673
##
```

```
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.5052     0.2061 -21.855  < 2e-16 ***
## male              0.5519     0.1082   5.101 3.38e-07 ***
## age               2.4424     0.2501   9.765  < 2e-16 ***
## cigsPerDay        1.3511     0.3041   4.442 8.90e-06 ***
## prevalentStroke   0.9471     0.4543   2.085   0.0371 *
## prevalentHyp      0.2524     0.1416   1.783   0.0746 .
## sysBP             2.6742     0.6265   4.268 1.97e-05 ***
## glucose           2.5565     0.5947   4.299 1.72e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2899.4  on 3391  degrees of freedom
## Residual deviance: 2584.6  on 3384  degrees of freedom
## AIC: 2600.6
##
## Number of Fisher Scoring iterations: 5
```

```
modelBestLogLike <- logLik(modelBest)
psi("Full Model Log likelihood: ", modelBestLogLike)
```

```
## [1] "Full Model Log likelihood:  -1292.29950397636"
```

```
modelBestAIC <- AIC(modelBest)
psi("Full Model AIC: ", modelBestAIC)
```

```
## [1] "Full Model AIC:  2600.59900795271"
```

```
# - Print these 'best' parameters (coefficients) along with their 95% confidence intervals in a single
cbind(coef(modelBest), suppressMessages(confint(modelBest)))
```

```
##                                 2.5 %     97.5 %
## (Intercept)     -4.5051547 -4.91614608 -4.1077603
## male             0.5518505  0.34015779  0.7644300
## age              2.4423847  1.95490424  2.9357135
## cigsPerDay       1.3510675  0.75213681  1.9451622
## prevalentStroke  0.9470964  0.03629289  1.8373190
## prevalentHyp     0.2524334 -0.02621069  0.5290944
## sysBP            2.6741506  1.44723355  3.9053357
## glucose          2.5565420  1.40352323  3.7445832
```

Problem 5

```
activationFunction <- function(x)(ifelse(x >= 0.5, 1, 0))

# - Using the 'best' model, â predictâ  the 10YrCHD on the training set.
trainingSetPredictions <- activationFunction(predict(modelBest, newdata=trainset, type="response"))
trainingSetPredictions <- sigmoid::sigmoid(predict(modelBest, newdata=trainset), method = "logistic", i

# -- Display the confusion matrix, the accuracy, as well as the true positive and true negative rates.
t = table(actual = trainset$TenYearCHD, predict = trainingSetPredictions > 0.5)
acc(t, FALSE)
```

```
##            predict
## actual      pred false pred true
##   act false       2857        17
##   act true         476        42
##
## [1] "Num of Observ" "3392"
## [1] "Sensitivity or Recall" "0.0811"
## [1] "Specificity" "0.9941"
## [1] "Overall Accuracy" "0.8547"
```

Question: Compare to the baseline model from Question 2. \

The two models had the same overall accuracy of 85.14%. This shows that even the best model still struggles to properly predict correctly.

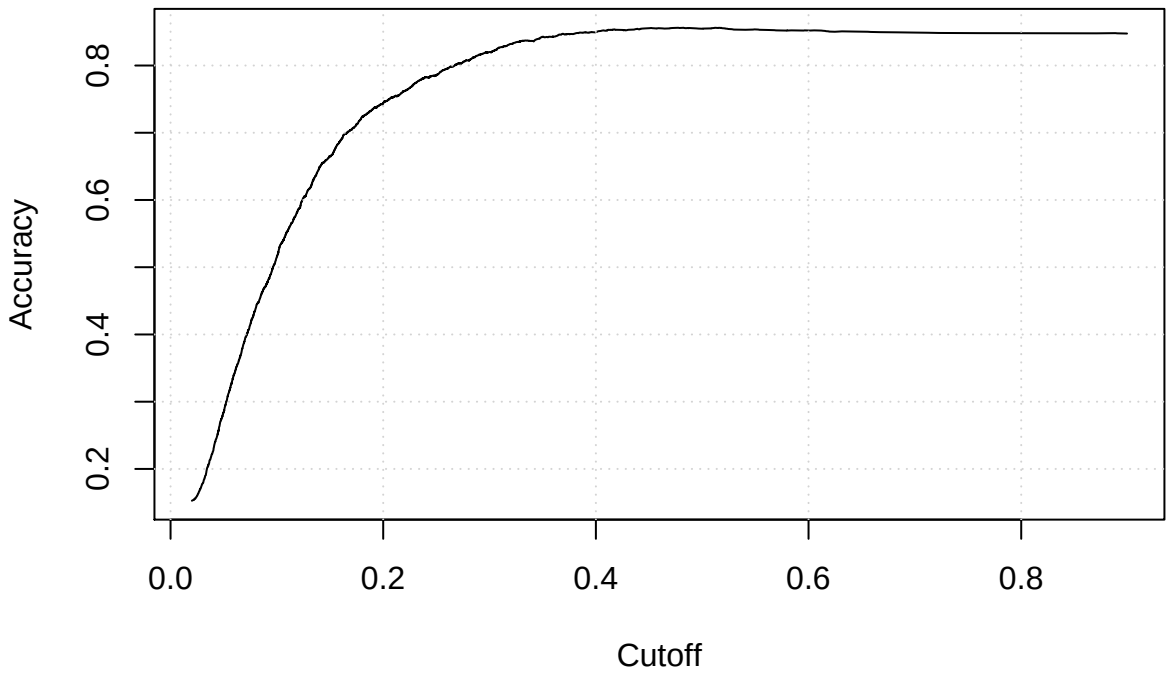
```
# - Using the 'best' model, predict the 10YrCHD on the test set
# -- Display the confusion matrix, the accuracy, as well as the true positive and true negative rates
testSetPredictions <- activationFunction(predict(modelBest, newdata=testset, type="response"))
testSetPredictions <- sigmoid::sigmoid(predict(modelBest, newdata=testset), method = "logistic", inverse

# -- Display the confusion matrix, the accuracy, as well as the true positive and true negative rates.
t = table(actual = testset$TenYearCHD, predict = testSetPredictions > 0.5)
acc(t, FALSE)
```

```
##            predict
## actual      pred false pred true
##   act false        715         7
##   act true         119         7
##
## [1] "Num of Observ" "848"
## [1] "Sensitivity or Recall" "0.0556"
## [1] "Specificity" "0.9903"
## [1] "Overall Accuracy" "0.8514"
```

Problem 6

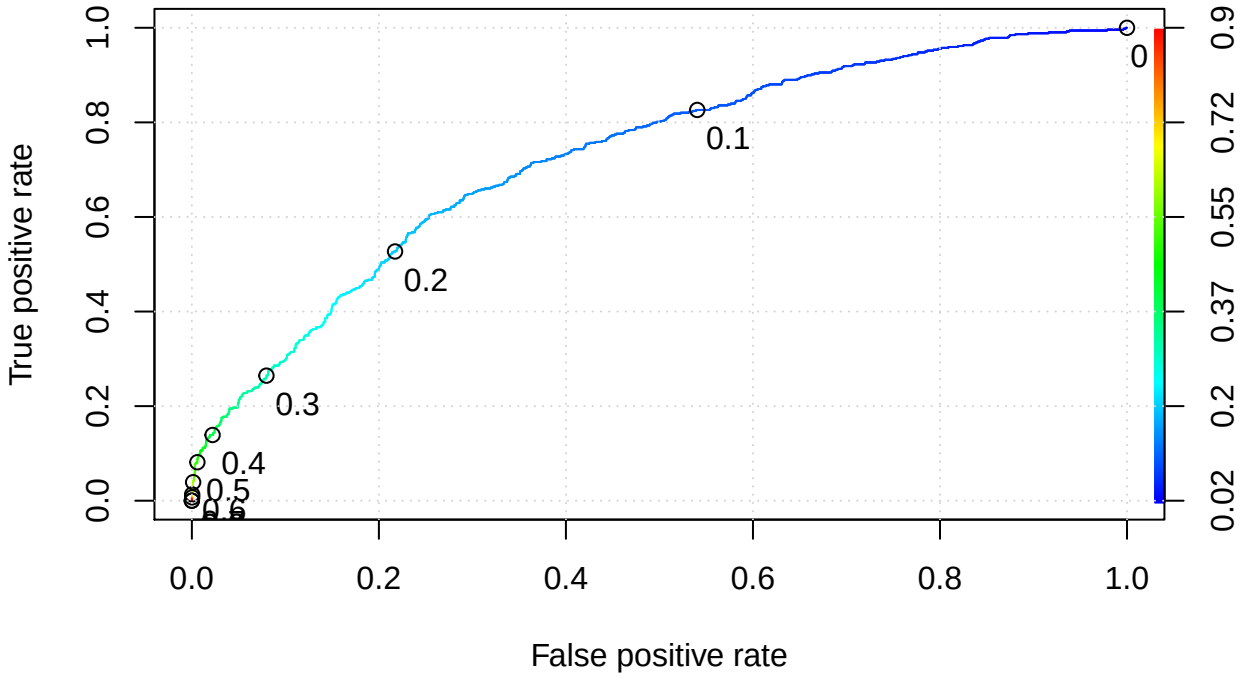
```
# Now use ROC curves to reconsider a threshold (cutoff) of .5 using the ROCR (or other) package
# -- Create the prediction object for the training set
rocrPred <- ROCR::prediction(data.frame(trainingSetPredictions)[,1], trainset$TenYearCHD)
```

```
# - Plot the accuracy v. the threshold (cutoff)
perf <- ROCR::performance(rocrPred, measure = "acc")
plot(perf, colorize=FALSE, colorize.palette=rev(rainbow(256)))
grid()
```



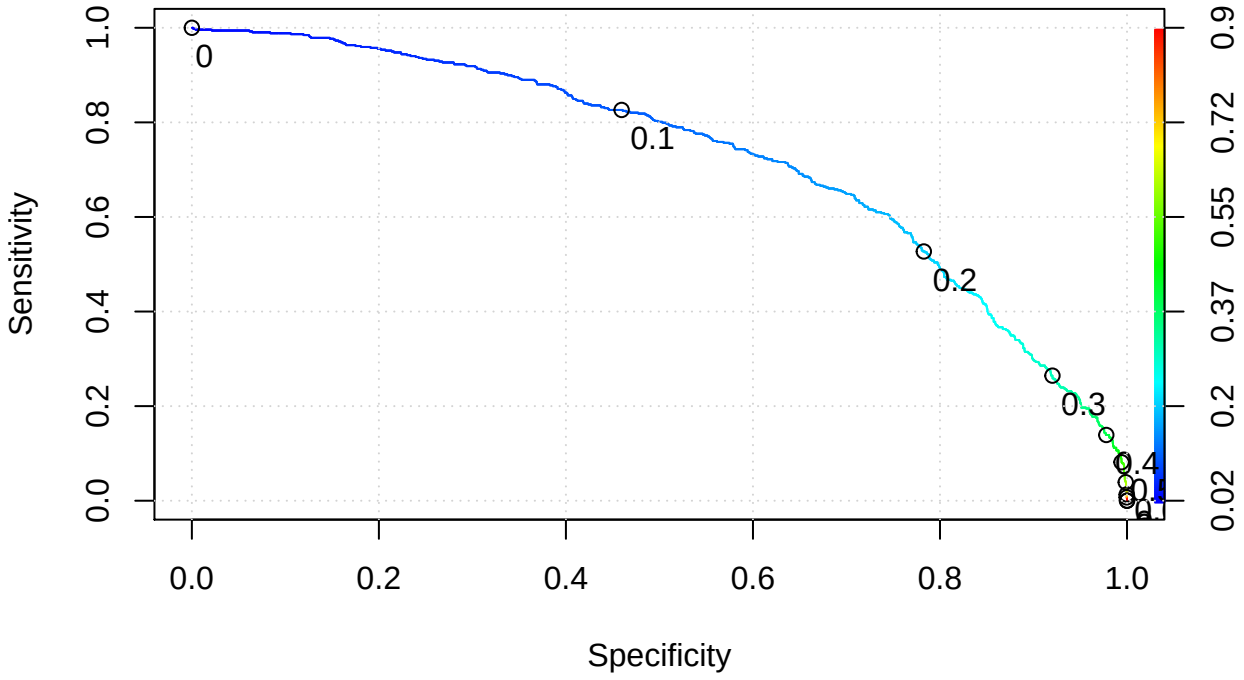
Comment on accuracy v. the threshold: We can see on the above graph that our model rapidly gains accuracy between the cutoff range of 0.0 and 0.2. However, after that point, we approach the threshold of approximately 0.85. Thus, this tells us that there is really not much to gain in terms of accuracy past the cutoff value of 0.2, which we should take into consideration when evaluating the ROC plots below.

```
# - Plot the true positive rate v. the false positive rate and label the threshold values
tprFprPerf <- ROCR::performance(rocrPred, measure = "tpr", x.measure = "fpr")
plot(tprFprPerf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))
grid()
```



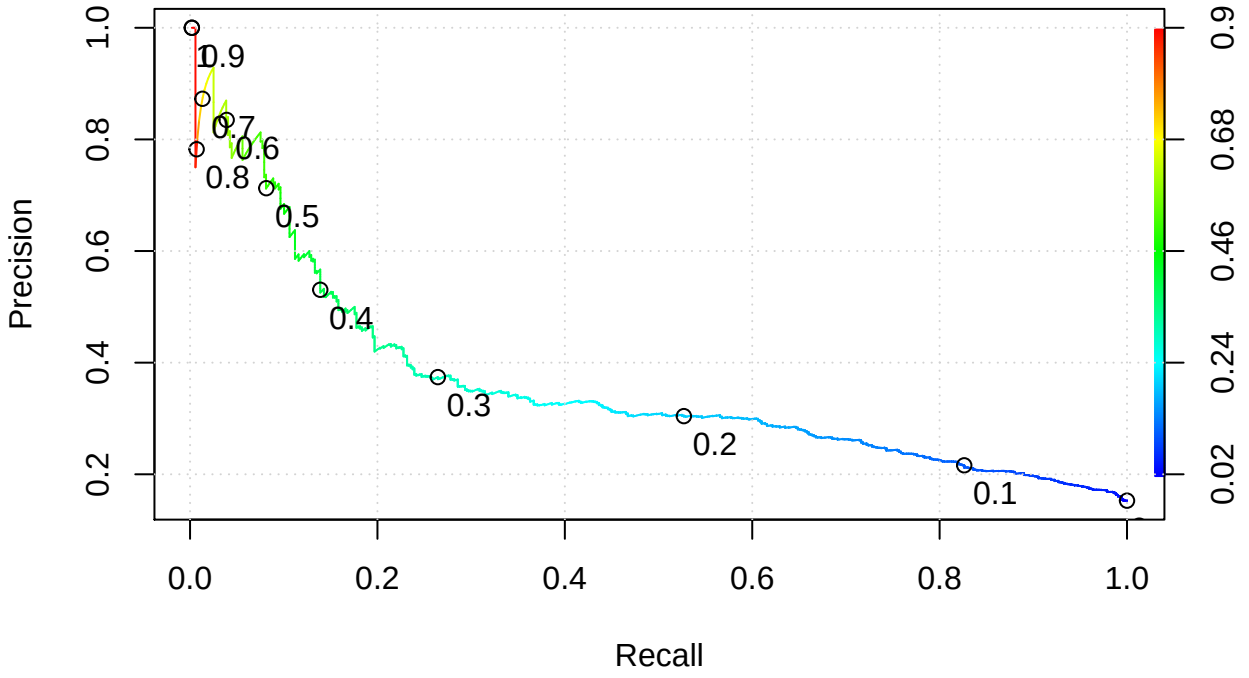
Comment on TPR v. the FPR: The graph above shows us that any increase in sensitivity will be accompanied by a decrease in specificity. More specifically, the model is more accurate the closer the curve follows the left-hand and top border of the ROC space. Alternatively, we see that the model is less accurate the closer the curve comes to the 45-degree diagonal of the ROC space. In the range between 0.0 and 0.2 on the false positive rate axis, we gain a significant amount in the value of the true positive rate without losing much of the value from the false positive rate. However, after 0.2 on the false positive rate axis, we start to lose a more significant amount for any gain in the true positive rate. Thus, we must consider which statisitic is more important based on the context of our problem.

```
# - Plot sensitivity v. specificity and label the threshold values
sensVsSpecPerf <- ROCR::performance(rocrPred, measure = "sens", x.measure = "spec")
plot(sensVsSpecPerf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))
grid()
```



Comment on sensitivity v. specificity: Recall that sensitivity is the measure of correctly classified true values and specificity is the measure of correctly classified negative values. Thus, the above graph shows us we slowly start to misclassify true values as we rapidly start to correctly classify negative values. Depending on the application of the model, one might value a higher sensitivity measure over a higher specificity value. In this problem set, where it is objectively more important to correctly classify those with a medical condition, we value sensitivity over specificity. However, we might accept some trade off for a higher specificity value to the range of 0.4 to 0.8.

```
# - Plot precision v. recall and label the threshold values
precVsRecallPerf <- ROCR::performance(rocrPred, measure = "prec", x.measure = "rec")
plot(precVsRecallPerf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))
grid()
```



Comment on precision v. recall: Recall is essentially the ability of our model to find all the relevant true values within the dataset. More specifically, recall is the number of true positives divided by the number of true positives plus the number of false negatives. In contrast to recall, precision expresses the proportion of the data points our model says was relevant actually were relevant. More specifically, precision is the number of true positives divided by the number of true positives plus the number of false positives. So, in the context of the above graph, we can see there is a major point of inversion between 0.4 and 0.3 on the precision axis. This is the range we should consider interesting because outside of this range, the cost of tradeoff between precision and recall is too great unless we are specifically concerned with maximizing one value over the other.

```
# Using best
best.auc.perf <- ROCR::performance(rocrPred, measure="auc")
psr("AUC value for the 'best' model", best.auc.perf@y.values[[1]], 4)
```

```
## [1] "AUC value for the 'best' model 0.7270"
```

```
# Create predictions using ROCR for null and full
null_trainingSetPredictions <- sigmoid::sigmoid(predict(modelNull, newdata=trainset), method = "logistic
null_rocrPred <- ROCR::prediction(data.frame(null_trainingSetPredictions)[,1], trainset$TenYearCHD)

null.auc.perf <- ROCR::performance(null_rocrPred, measure="auc")
psr("AUC value for the 'null' model", null.auc.perf@y.values[[1]], 4)
```

```
## [1] "AUC value for the 'null' model 0.5000"
```

```
full_trainingSetPredictions <- sigmoid::sigmoid(predict(modelFull, newdata=trainset), method = "logistic
full_rocrPred <- ROCR::prediction(data.frame(full_trainingSetPredictions)[,1], trainset$TenYearCHD)
```

```r
full.auc.perf <- ROCR::performance(full_rocrPred, measure="auc")
psr("AUC value for the 'full' model", full.auc.perf@y.values[[1]], 4)
```

```
## [1] "AUC value for the 'full' model 0.7284"
```

Question: Commment on AUC. \

An AUC of .5 is considered worthless, so, as expected, the 'null' model is not a desirable model for this problem. Interestingly, in our AUC calculations, the 'full' model had a higher, and therefore better, AUC than the 'best' model did, make the 'full' model the best performing model under the AUC test.

Problem 7

```r
# Using gradient descent, solve for the parameters from the 'best backward eliminaton' data set and ver
hypothesisVector = function(X, theta) {
  # sigmoid of the dot product between X and theta
  return ( as.vector( 1 / (1+exp(-(X %*% theta))) ))
}

Jcost = function(y, h) {
  epsilon = 0.0000000001
  n = nrow(y)

  # Take the error when label=0
  ifelse(h < epsilon, epsilon, h)
  # Take the error when label=1
  ifelse(h > 1-epsilon, 1-epsilon, h)

  return( (1/n)*((-y*log(h)) - ((1-y)*log(1-h))) )
}

gradientDescent <- function(X, y){
  X = as.matrix(X)
  y = as.vector(y)

  theta = runif(ncol(X), -1, 1)
  alpha = 1
  maxIterations = 25000
  n = nrow(y)
  epsilon = 0.0000000001
  oldSSE = 999999999999

  for ( i in 1:maxIterations){
      hypothesis = hypothesisVector(X, theta)
      # psi("cols: ", length(hypothesis))

      cost = Jcost(y, hypothesis)
      loss = hypothesis - y

      # avg gradient per example
      gradient = (t(X) %*% as.matrix(loss)) / n
      # gradient = (t(X) %*% as.matrix(hypothesis * y)) / n
```

```
      # update
      theta = theta - alpha * gradient

      newSSE = sum(loss^2)

      errn = abs((newSSE - oldSSE)/oldSSE)
      if(errn < epsilon){
        break
      }
      oldSSE = newSSE
  }

  psi("Total number of iterations: ", i)
  return (theta)
}

# - Create the X and y test and training sets
trainY <- data.frame(trainset$TenYearCHD)
trainX <- trainset[,c("male", "age", "cigsPerDay", "prevalentStroke", "prevalentHyp", "sysBP", "glucose
trainX <- cbind(rep(1,nrow(trainX)), trainX)
colnames(trainX)[1] = "(Intercept)"

testY <- data.frame(testset$TenYearCHD)
testX <- testset[,c("male", "age", "cigsPerDay", "prevalentStroke", "prevalentHyp", "sysBP", "glucose")]
testX <- cbind(rep(1,nrow(testX)), testX)
colnames(testX)[1] = "(Intercept)"

# - Compute the parameters (coefficients)
trainThetas <- gradientDescent(trainX, trainY)
```

```
## [1] "Total number of iterations:  13143"
```

```
testThetas <- gradientDescent(testX, testY)
```

```
## [1] "Total number of iterations:  6001"
```

```
# - Compare to glm()'s coefficients in a single table
comparisons = cbind(modelBest$coefficients[1:8], cbind(trainThetas,testThetas))
colnames(comparisons) = c("ModelBest", "GD Train", "GD Test")
comparisons
```

```
##                  ModelBest   GD Train    GD Test
## (Intercept)     -4.5051547 -4.5051246 -4.8656038
## male             0.5518505  0.5518463  0.1724351
## age              2.4423847  2.4423982  2.5610519
## cigsPerDay       1.3510675  1.3510678  2.0505032
## prevalentStroke  0.9470964  0.9470935  1.8013169
## prevalentHyp     0.2524334  0.2524613  0.2279601
## sysBP            2.6741506  2.6739681  3.5923585
## glucose          2.5565420  2.5565503  3.3783737
```

Problem 8

```
# Section of auxiliary functions used in calculating the statistics for question 8. See the section bel

calculate_ssr <- function(actual, predictions){
  # returns value residual sum of squared errors
  ssr = sum((actual - predictions)^2)
  return (ssr)
}

calculate_sse <- function(actual, predictions){
    # returns value sum of squared errors
    ybar = sum(actual)/nrow(actual)
    sse = sum((predictions-ybar)^2)
    return (sse)
}

calculate_sst <- function(actual, predictions){
    # returns value total sum of squared errors
    ybar = sum(actual)/nrow(actual)
    sst = sum((actual - ybar)^2)
    return (sst)
}

LogL = function(y,h) {
  return ( (t(y) %*% log(h) + t(1-y) %*% log(1-h)) )
}

manualAIC <- function(n, sse, ssr, dfm, actual, predictions){
    # returns value for Akaike's information criteria
    ll = loglike(n, sse, ssr, dfm, actual, predictions)
    aic = 2*(dfm+1)-2*ll
    return (aic)
}

compute_standard_error <- function(X){
    # returns calculated value of standard errors of matrix X in an array
    mf <- function(x){sd(X[,x])/sqrt(length(X[,x]))}
    return(sapply(1:length(X[1,]),mf))
}

compute_t_statistic <- function(std_err_arr, coefficients){
    # returns calculated value of t-statistics for each coefficent in an array
    t_stat_arr = c()
    for( i in 1:length(std_err_arr)){
        t_stat = (coefficients[i] / std_err_arr[i])
        t_stat_arr = c(t_stat_arr,t_stat)
    }
    return ( t_stat_arr )
}

compute_p_statistic <- function(t_statistic, n){
    # returns calculated value of p-statistics for each coefficent in an array
```

```
    p_stat_arr = c()
    for( i in 1:length(t_statistic)){
        pval = pt(-abs(t_statistic[i]), df = n-1) # stats.t.sf(np.abs(t_statistic[i]), n-1)*2
        p_stat_arr = c(p_stat_arr, pval)
    }

    return (p_stat_arr)
}
```

Part 1: Gradient Descent Statistics on Training Data \

The training data log likelihood, deviance, and AIC are pretty much the same from our own gradient descent model to the glm model's, which can be seen in the summary of question #4.

```
std_err = compute_standard_error(trainX)
t_stat = compute_t_statistic(std_err, trainThetas)
p_stat = compute_p_statistic(t_stat, nrow(trainX))
train_tab <- cbind(trainThetas, cbind(std_err, cbind(t_stat, p_stat)))
colnames(train_tab) <- c("Train_Coeffs", "Std_Err_Train", "t_stat_Train", "p_stat_Train")
print(train_tab)
```

```
##                 Train_Coeffs Std_Err_Train t_stat_Train  p_stat_Train
## (Intercept)       -4.5051246   0.000000000         -Inf  0.000000e+00
## male               0.5518463   0.008499171     64.92942  0.000000e+00
## age                2.4423982   0.003899117    626.39771  0.000000e+00
## cigsPerDay         1.3510678   0.002897355    466.31076  0.000000e+00
## prevalentStroke    0.9470935   0.001409272    672.04471  0.000000e+00
## prevalentHyp       0.2524613   0.007932773     31.82511 5.120066e-195
## sysBP              2.6739681   0.001814542   1473.63252  0.000000e+00
## glucose            2.5565503   0.001201845   2127.18870  0.000000e+00
```

```
predictions_grad_desc_train = as.vector( 1 / (1+exp(-as.matrix(trainX) %*% as.vector(trainThetas))) )

ssr = calculate_ssr(trainY, predictions_grad_desc_train)
sse = calculate_sse(trainY, predictions_grad_desc_train)
sst = calculate_sst(trainY, predictions_grad_desc_train)
psi("SSR for training data using gradient descent: ", ssr)
```

```
## [1] "SSR for training data using gradient descent:  393.281606039443"
```

```
psi("SSE for training data using gradient descent:", sse)
```

```
## [1] "SSE for training data using gradient descent: 45.1467747563782"
```

```
psi("SST for training data using gradient descent:", sst)
```

```
## [1] "SST for training data using gradient descent: 438.895047169811"
```

```
log_life_train = LogL(trainY,predictions_grad_desc_train)
dev_train = -2 * log_life_train
aic_train = dev_train + 2 * ncol(trainX)
psi("Log likelihood for training data using gradient descent: ", log_life_train)
```

```
## [1] "Log likelihood for training data using gradient descent:  -1292.29950401886"
```

```
psi("Deviance for training data using gradient descent: ", dev_train)
```

```
## [1] "Deviance for training data using gradient descent:  2584.59900803772"
```

```
psi("AIC for training data using gradient descent: ", aic_train)
```

```
## [1] "AIC for training data using gradient descent:  2600.59900803772"
```

Part 2: Gradient Descent Statistics on Test Data

```
std_err = compute_standard_error(testX)
t_stat = compute_t_statistic(std_err, testThetas)
p_stat = compute_p_statistic(t_stat, nrow(testX))
train_tab <- cbind(trainThetas, cbind(std_err, cbind(t_stat, p_stat)))
colnames(train_tab) <- c("Test_Coeffs", "Std_Err_Test", "t_stat_Test", "p_stat_Test")
print(train_tab)
```

```
##                 Test_Coeffs Std_Err_Test t_stat_Test  p_stat_Test
## (Intercept)      -4.5051246  0.000000000        -Inf 0.000000e+00
## male              0.5518463  0.017013060    10.13546 3.640171e-23
## age               2.4423982  0.007543182   339.51878 0.000000e+00
## cigsPerDay        1.3510678  0.006050703   338.88678 0.000000e+00
## prevalentStroke   0.9470935  0.001666720  1080.75564 0.000000e+00
## prevalentHyp      0.2524613  0.016006937    14.24133 1.020955e-41
## sysBP             2.6739681  0.003364471  1067.73350 0.000000e+00
## glucose           2.5565503  0.001874728  1802.06067 0.000000e+00
```

```
predictions_grad_desc_test = as.vector( 1 / (1+exp(-as.matrix(testX) %*% as.vector(testThetas))) )

ssr = calculate_ssr(testY, predictions_grad_desc_test)
sse = calculate_sse(testY, predictions_grad_desc_test)
sst = calculate_sst(testY, predictions_grad_desc_test)
psi("SSR for testing data using gradient descent: ", ssr)
```

```
## [1] "SSR for testing data using gradient descent:  95.4303675293807"
```

```
psi("SSE for testing data using gradient descent:", sse)
```

```
## [1] "SSE for testing data using gradient descent: 12.6682226172947"
```

```
psi("SST for testing data using gradient descent:", sst)
```

```
## [1] "SST for testing data using gradient descent: 107.278301886792"
```

```
log_life_test = LogL(testY,predictions_grad_desc_test)
dev_test = -2 * log_life_test
aic_test = dev_test + 2 * ncol(testX)
psi("Log likelihood for testing data using gradient descent: ", log_life_test)
```

```
## [1] "Log likelihood for testing data using gradient descent:  -312.515407603527"
```

```
psi("Deviance for testing data using gradient descent: ", dev_test)
```

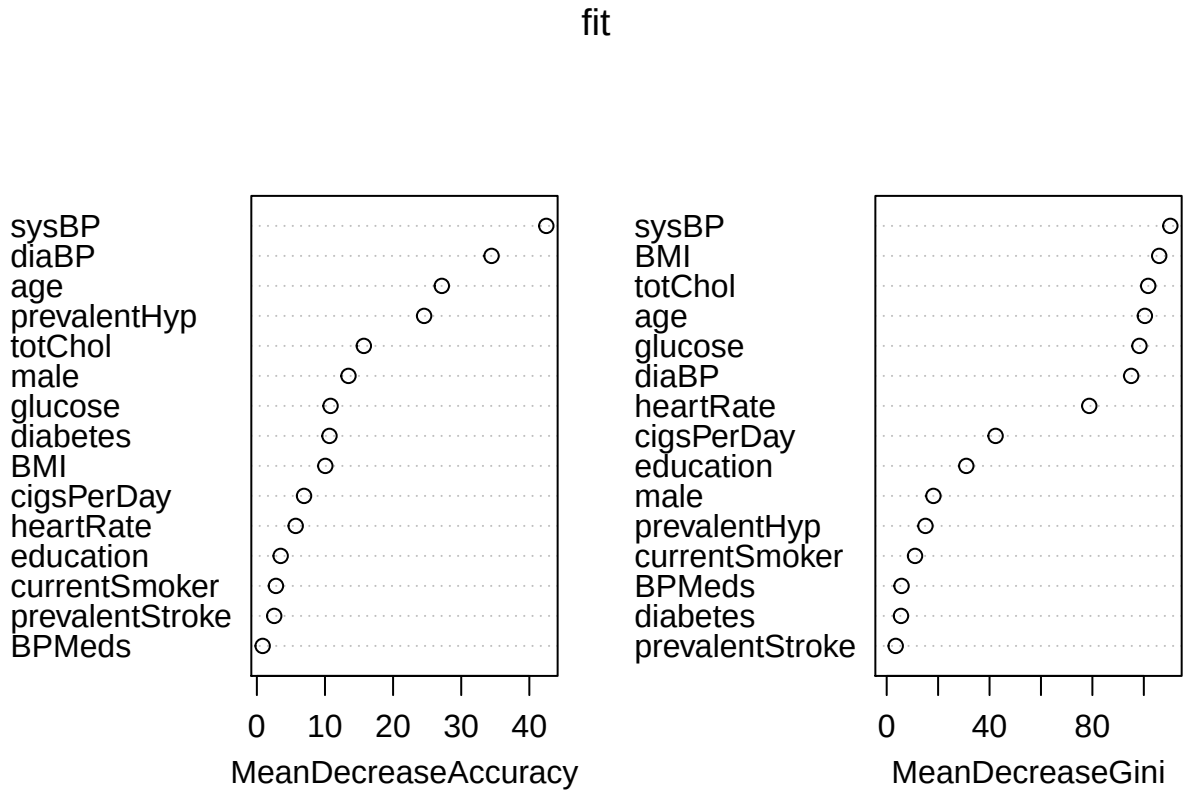
```
## [1] "Deviance for testing data using gradient descent:  625.030815207055"
```

```
psi("AIC for testing data using gradient descent: ", aic_test)
```

```
## [1] "AIC for testing data using gradient descent:  641.030815207055"
```

Problem 9

```
# - Implement using library functions on the training set
fit <- randomForest(as.factor(TenYearCHD) ~., data=trainset, importance=TRUE, ntree=1000)
varImpPlot(fit)
```




```
pred <- predict(fit, testset)

# - Create confusion tables for predicting on the test set. You may use any solution parameters or tuni
# - Display the accuracy and the true positive and negative error rates
t = table(actual = testset$TenYearCHD, predict = pred)
acc(t, FALSE)
```

```
##            predict
## actual      pred false pred true
##   act false        718         4
##   act true         118         8
##
## [1] "Num of Observ" "848"
## [1] "Sensitivity or Recall" "0.0635"
## [1] "Specificity" "0.9945"
## [1] "Overall Accuracy" "0.8561"
```

By looking at the varImpPlot, we can clearly see that some variables are much more significant than others. By not pruning the random forest and passing it all the features, some of the trees could have been made with features that were not useful, making them give improper predictions. In the situation of the Framingham study, this model is much worse because the sensitivity of it is very low.

Problem 10

Question: How is that method different from logistic regression? One paragraph is sufficient, full understanding is not required, just the essential idea.

Excerpt from Study: Statistical tests included age-adjusted linear regression or logistic regression to test for trends across blood pressure, TC, LDL-C, and HDL-C categories. Age-adjusted Cox proportional hazards regression and its accompanying c statistic were used to test for the relation between various independent variables and the CHD outcome and to evaluate the discriminatory ability of various prediction models. \

The prediction of CHD has taken the form of sex-specific equations that were developed from a single study and applied to other populations or individuals. Age, TC, HDL-C, and blood pressure were used in the equations as continuous variables, in contrast to dichotomous variables (yes/no) such as smoking, diabetes, and left ventricular hypertrophy. The present study builds on the prior experience of CHD prediction with continuous variables and integrates the categorical approaches that have become part of the framework of blood pressure (JNC-V) and cholesterol (NCEP) programs in the United States. As suggested in an earlier NCEP report, their approach integrates blood pressure and cholesterol information and estimates both relative and absolute CHD risk with a risk factor weighting approach.