CITS4009 PROJECT 2

Students - Daniel SANTOSA , Omri Y Ram

Student Nos. - 19315466 , 22274048

Introduction

Disclaimer : The code contained in this project has been emulated from lecture notes of Dr. Wei Liu.

The data set analyzed has been taken from the “US Accident Injury Data Set”. This data set contains information about the accidents that occurred in mine sites.

load libraries

```
library(ROCit)
library("ROCR")
library(ggthemes)
library(ggplot2)
library(pander)
library('rpart')
library(dplyr)
library(fpc)
```

```
## Warning: package 'fpc' was built under R version 4.0.3
```

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.0.3
```

```
library('grDevices')
library("dplyr")
library(gridExtra)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.3
```

Single Variable model

Load the main data and only take in relevant data used in the analysis. Then create three subsets (training testing and calibration) and divide columns in numeric or categorical variables

```
data <- read.csv('us_data.csv')
data$target <- ifelse(data$COAL_METAL_IND == "C", 1, 0)
data$DEGREE_INJURY <- ifelse(data$DEGREE_INJURY == "NO VALUE FOUND", NA, data$DEGREE_INJURY)
outcomes = c("target", "COAL_METAL_IND")

set.seed(2345)

data$rgroup <- runif(dim(data)[[1]])
dTrainAll <- subset(data, rgroup <= 0.9)
dTest <- subset(data, rgroup >0.9)


vars <- setdiff(colnames(dTrainAll), c(outcomes, "rgroup"))
catVars <- vars[sapply(dTrainAll[,vars], class) %in% c("factor", "character")]

numericVars <- vars[sapply(dTrainAll[,vars], class)%in%c('numeric', 'integer')]

outcome <- 'target'
pos <- "1"

useForCal <- rbinom(n = nrow(dTrainAll), size = 1, prob = 0.1)> 0
dCal <- subset(dTrainAll, useForCal)
dTrain <- subset(dTrainAll, !useForCal)
```

The following three functions are used for constructing the single variable model

```
mkPredC <- function(outCol, varCol, appCol){
  pPos <- sum(outCol == pos)/length(outCol)
  naTab <- table(as.factor(outCol[is.na(varCol)]))
  pPosWna <- (naTab/sum(naTab))[pos]
  vTab <- table(as.factor(outCol), varCol)
  pPosWv <- (vTab[pos, ] + 1.0e-3*pPos)/(colSums(vTab)+1.0e-3)
  pred <- pPosWv[appCol]
  pred[is.na(appCol)] <- pPosWna
  pred[is.na(pred)]<-pPos
  pred
}

mkPredN <- function(outcol, varCol, appCol){
  cuts <- unique(as.numeric(
    quantile(varCol, probs = seq(0,1,0.1), na.rm = T)))
  varC <- cut(varCol, cuts)
  appC <- cut(appCol, cuts)
  mkPredC(outcol, varC, appC)
}


calcAUC <- function(predcol, outcol){
  perf <- performance(prediction(predcol, outcol == pos), 'auc')
  as.numeric(perf@y.values)
}
```

Create a single variable model for the target variable using each column for categorical variables

```
for(v in catVars) {
  pi <- paste('pred', v, sep = "")
  dTrain[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dTrain[,v])
  dCal[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dCal[,v])
  dTest[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dTest[,v])
}
```

Create the single variable model for each numerical variable (through binning them) and then print out the auc for each variable that is above a threshold of 0.55

```
for(v in numericVars){
  pi <- paste('pred', v, sep='')
  dTrain[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dTrain[,v])
  dTest[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dTest[,v])
  dCal[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dCal[,v])
  aucTrain <- calcAUC(dTrain[,pi], dTrain[,outcome])
  if(aucTrain >= 0.55){
    aucCal <- calcAUC(dCal[,pi], dCal[,outcome])
    print(sprintf(
      "%s, trainAUC: %4.3f calibrationAUC: %4.3f",
      pi, aucTrain, aucCal
    ))
  }
}
```

```
## [1] "predMINE_ID, trainAUC: 0.815 calibrationAUC: 0.809"
## [1] "predSUBUNIT_CD, trainAUC: 0.843 calibrationAUC: 0.846"
## [1] "predACCIDENT_TIME, trainAUC: 0.620 calibrationAUC: 0.614"
## [1] "predFIPS_STATE_CD, trainAUC: 0.867 calibrationAUC: 0.864"
## [1] "predSHIFT_BEGIN_TIME, trainAUC: 0.657 calibrationAUC: 0.655"
## [1] "predNO_INJURIES, trainAUC: 0.590 calibrationAUC: 0.591"
## [1] "predTOT_EXPER, trainAUC: 0.643 calibrationAUC: 0.644"
## [1] "predMINE_EXPER, trainAUC: 0.636 calibrationAUC: 0.635"
## [1] "predJOB_EXPER, trainAUC: 0.594 calibrationAUC: 0.596"
## [1] "predDAYS_RESTRICT, trainAUC: 0.634 calibrationAUC: 0.638"
## [1] "predDAYS_LOST, trainAUC: 0.577 calibrationAUC: 0.571"
## [1] "predCLOSED_DOC_NO, trainAUC: 0.556 calibrationAUC: 0.562"
```

Print out the AUC for each categorical over a threshold of 0.55

```
for(v in catVars) {
  pi <- paste('pred',v,sep='')
  aucTrain <- calcAUC(dTrain[,pi],dTrain[,outcome])
  if(aucTrain>=0.55) {
   aucCal <- calcAUC(dCal[,pi],dCal[,outcome])
   print(sprintf(
     "%s, trainAUC: %4.3f calibrationAUC: %4.3f",
     pi, aucTrain, aucCal
   ))
  }
}
```

```
## [1] "predCONTROLLER_ID, trainAUC: 1.000 calibrationAUC: 1.000"
## [1] "predCONTROLLER_NAME, trainAUC: 1.000 calibrationAUC: 1.000"
## [1] "predOPERATOR_ID, trainAUC: 1.000 calibrationAUC: 1.000"
## [1] "predOPERATOR_NAME, trainAUC: 1.000 calibrationAUC: 1.000"
## [1] "predCONTRACTOR_ID, trainAUC: 0.589 calibrationAUC: 0.577"
## [1] "predSUBUNIT, trainAUC: 0.856 calibrationAUC: 0.859"
## [1] "predACCIDENT_DT, trainAUC: 0.613 calibrationAUC: 0.525"
## [1] "predDEGREE_INJURY_CD, trainAUC: 0.741 calibrationAUC: 0.742"
## [1] "predDEGREE_INJURY, trainAUC: 0.741 calibrationAUC: 0.742"
## [1] "predUG_LOCATION_CD, trainAUC: 0.825 calibrationAUC: 0.829"
## [1] "predUG_LOCATION, trainAUC: 0.825 calibrationAUC: 0.829"
## [1] "predUG_MINING_METHOD_CD, trainAUC: 0.804 calibrationAUC: 0.809"
## [1] "predUG_MINING_METHOD, trainAUC: 0.804 calibrationAUC: 0.809"
## [1] "predMINING_EQUIP_CD, trainAUC: 0.678 calibrationAUC: 0.678"
## [1] "predMINING_EQUIP, trainAUC: 0.678 calibrationAUC: 0.678"
## [1] "predEQUIP_MFR_CD, trainAUC: 0.619 calibrationAUC: 0.617"
## [1] "predEQUIP_MFR_NAME, trainAUC: 0.618 calibrationAUC: 0.616"
## [1] "predEQUIP_MODEL_NO, trainAUC: 0.644 calibrationAUC: 0.585"
## [1] "predCLASSIFICATION_CD, trainAUC: 0.669 calibrationAUC: 0.666"
## [1] "predCLASSIFICATION, trainAUC: 0.669 calibrationAUC: 0.666"
## [1] "predACCIDENT_TYPE_CD, trainAUC: 0.662 calibrationAUC: 0.663"
## [1] "predACCIDENT_TYPE, trainAUC: 0.662 calibrationAUC: 0.663"
## [1] "predOCCUPATION_CD, trainAUC: 0.879 calibrationAUC: 0.878"
## [1] "predOCCUPATION, trainAUC: 0.861 calibrationAUC: 0.861"
## [1] "predACTIVITY_CD, trainAUC: 0.736 calibrationAUC: 0.734"
## [1] "predACTIVITY, trainAUC: 0.736 calibrationAUC: 0.734"
## [1] "predINJURY_SOURCE_CD, trainAUC: 0.762 calibrationAUC: 0.763"
## [1] "predINJURY_SOURCE, trainAUC: 0.762 calibrationAUC: 0.763"
## [1] "predNATURE_INJURY_CD, trainAUC: 0.641 calibrationAUC: 0.646"
## [1] "predNATURE_INJURY, trainAUC: 0.641 calibrationAUC: 0.646"
## [1] "predINJ_BODY_PART_CD, trainAUC: 0.641 calibrationAUC: 0.640"
## [1] "predINJ_BODY_PART, trainAUC: 0.641 calibrationAUC: 0.640"
## [1] "predTRANS_TERM, trainAUC: 0.591 calibrationAUC: 0.592"
## [1] "predRETURN_TO_WORK_DT, trainAUC: 0.645 calibrationAUC: 0.569"
## [1] "predIMMED_NOTIFY_CD, trainAUC: 0.590 calibrationAUC: 0.595"
## [1] "predIMMED_NOTIFY, trainAUC: 0.590 calibrationAUC: 0.595"
## [1] "predINVEST_BEGIN_DT, trainAUC: 0.587 calibrationAUC: 0.541"
## [1] "predNARRATIVE, trainAUC: 1.000 calibrationAUC: 0.507"
```

Logistic Regression

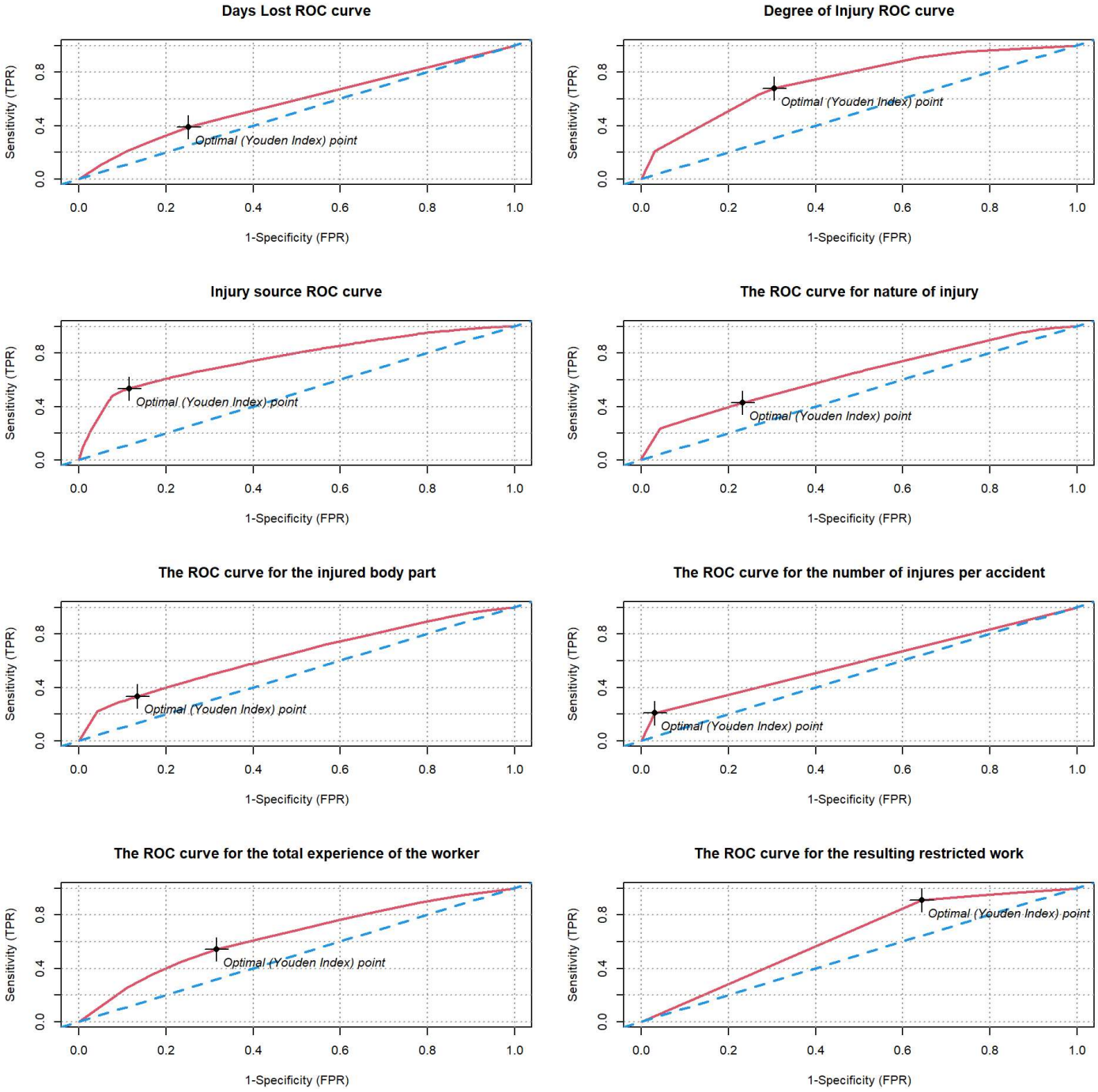
Based on the predicted variables derived from the AUC, we picked feature variables that best correspond to predicting an accident in either a metal or coal mining (COAL_METAL_IND) site. Out of the feature variables seen, some were filtered out due to them “giving away” the type of mining site (e.g UG_LOCATION and SUBUNIT) . Instead we focused on injuries due to their importance for stakeholders and ubiquitous nature.

The predicted variables were chosen to be used as the feature variables as they are numeric (required for regression) and represent the frequency of observing a certain value of a variable within each of our two classifications

The following feature variables chosen for regression

```
regressionVars<- c("predDAYS_LOST", "predDEGREE_INJURY", "predINJURY_SOURCE", "predNATURE_INJUR
Y",
                   "predINJ_BODY_PART", "predNO_INJURIES", "predTOT_EXPER", "predDAYS_RESTRICT")
```

```
plot_roc <- function(predcol, outcol){
  ROCit_obj <- rocit(score=predcol, class=outcol==pos)
  plot( ROCit_obj, col = c(2,4), legend = FALSE)
  title(outer=outer, main=colnames(predcol))
}
par(mfrow = c(4, 2))
plot_roc(dTrain$predDAYS_LOST, dTrain[[outcome]])
 title(outer=outer, main="Days Lost ROC curve")
plot_roc(dTrain$predDEGREE_INJURY, dTrain[[outcome]])
  title(outer=outer, main="Degree of Injury ROC curve")
plot_roc(dTrain$predINJURY_SOURCE, dTrain[[outcome]])
  title(outer=outer, main="Injury source ROC curve")
plot_roc(dTrain$predNATURE_INJURY, dTrain[[outcome]])
  title(outer=outer, main="The ROC curve for nature of injury")
plot_roc(dTrain$predINJ_BODY_PART, dTrain[[outcome]])
  title(outer=outer, main="The ROC curve for the injured body part")
plot_roc(dTrain$predNO_INJURIES, dTrain[[outcome]])
  title(outer=outer, main="The ROC curve for the number of injures per accident")
plot_roc(dTrain$predTOT_EXPER, dTrain[[outcome]])
  title(outer=outer, main="The ROC curve for the total experience of the worker")
plot_roc(dTrain$predDAYS_RESTRICT, dTrain[[outcome]])
  title(outer=outer, main="The ROC curve for the resulting restricted work")
```



As can be seen above all single variable models have a sufficiently high AUC to serve as good predictors for a model

Building a logistic regression model

The code below constructs the glm model

```
regressionFormula <- as.formula(paste('target',
                                      paste(regressionVars, collapse = '+'),
                                      sep = '~'))
reg_model <- glm(regressionFormula, family = binomial(link = "logit"),
                 data = dTrain)
summary(reg_model)
```

```
##
## Call:
## glm(formula = regressionFormula, family = binomial(link = "logit"),
##     data = dTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5708  -0.8674  -0.2796   0.8186   2.6594
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -4.96071    0.08169 -60.727   <2e-16 ***
## predDAYS_LOST      2.13683    0.10498  20.354   <2e-16 ***
## predDEGREE_INJURY  2.49284    0.07586  32.860   <2e-16 ***
## predINJURY_SOURCE  4.41491    0.03492 126.440   <2e-16 ***
## predNATURE_INJURY  1.76201    0.08139  21.650   <2e-16 ***
## predINJ_BODY_PART -0.82716    0.09775  -8.462   <2e-16 ***
## predNO_INJURIES   -3.34403    0.12450 -26.860   <2e-16 ***
## predTOT_EXPER      1.65863    0.05617  29.531   <2e-16 ***
## predDAYS_RESTRICT  1.72812    0.07108  24.313   <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: 227617  on 164196  degrees of freedom
## Residual deviance: 168528  on 164188  degrees of freedom
## AIC: 168546
##
## Number of Fisher Scoring iterations: 4
```

From here we can see that the number of Fisher scoring iterations are not too high (less than 8) . The coefficients are all significant i.e they are not random. The coefficients for each variable can also be used to derive differences between each type of mine.

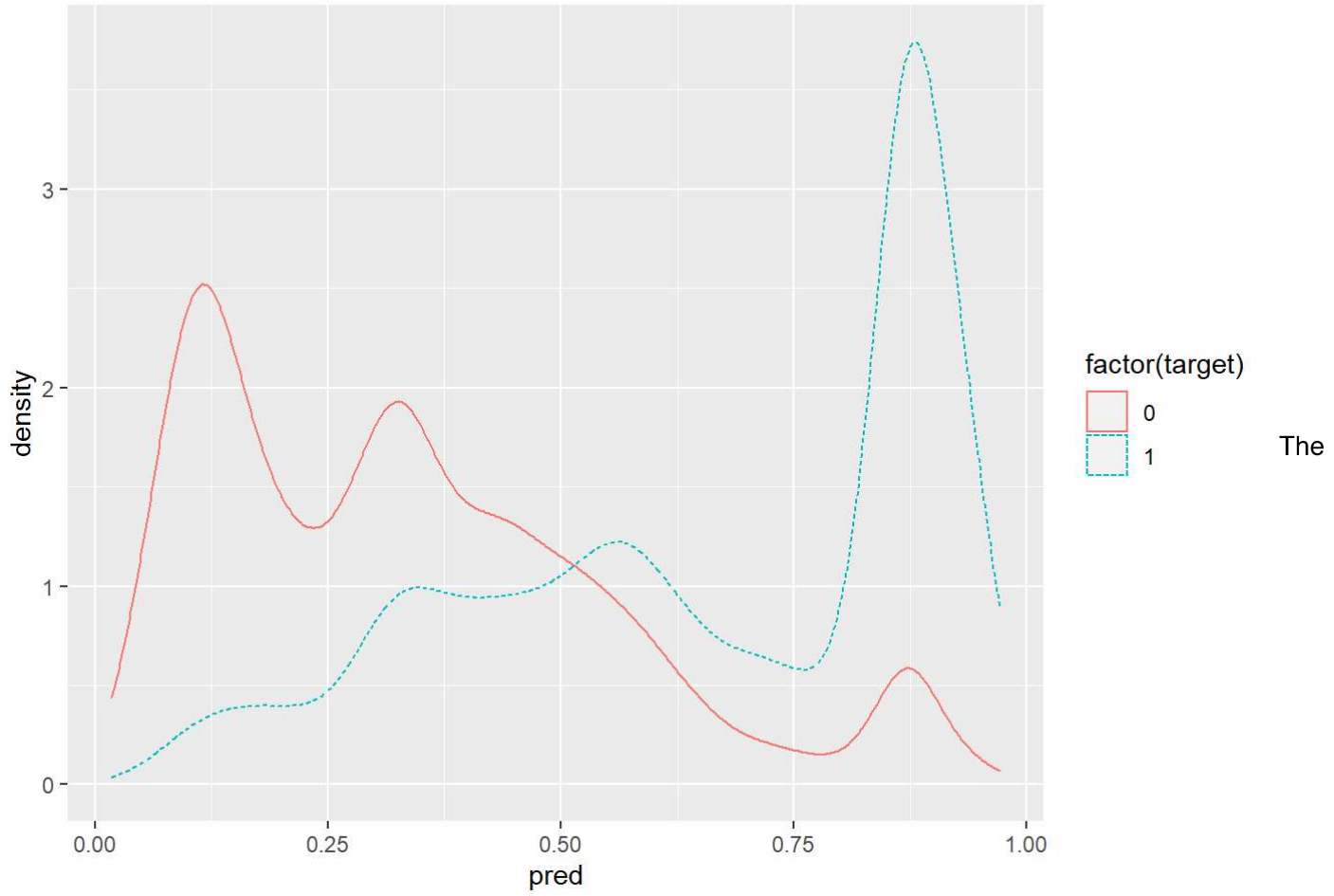
Predict for our training data and test data

```
dTrain$pred <- predict(reg_model, newdata = dTrain,
                           type = 'response')
dTest$pred <- predict(reg_model, newdata = dTest,
                      type = 'response')
```

Choosing a threshold

Out of our predicted values, those that are close to 1 represent a high probability of the accident belonging to a coal mine.

```
ggplot(dTest, aes(x=pred, color=factor(target), linetype = factor(target))) +
  geom_density()
```



The density diagram shown above represents the pred values for metal mines (0, red) and coal mines (1, blue) against the proportion of mines that falls within each value. We can see that the lines cross over at around 0.5 and most of the two types of mines fall outside of this cut off. We can explore this further in order to find an adequate threshold to make predictions with our model.

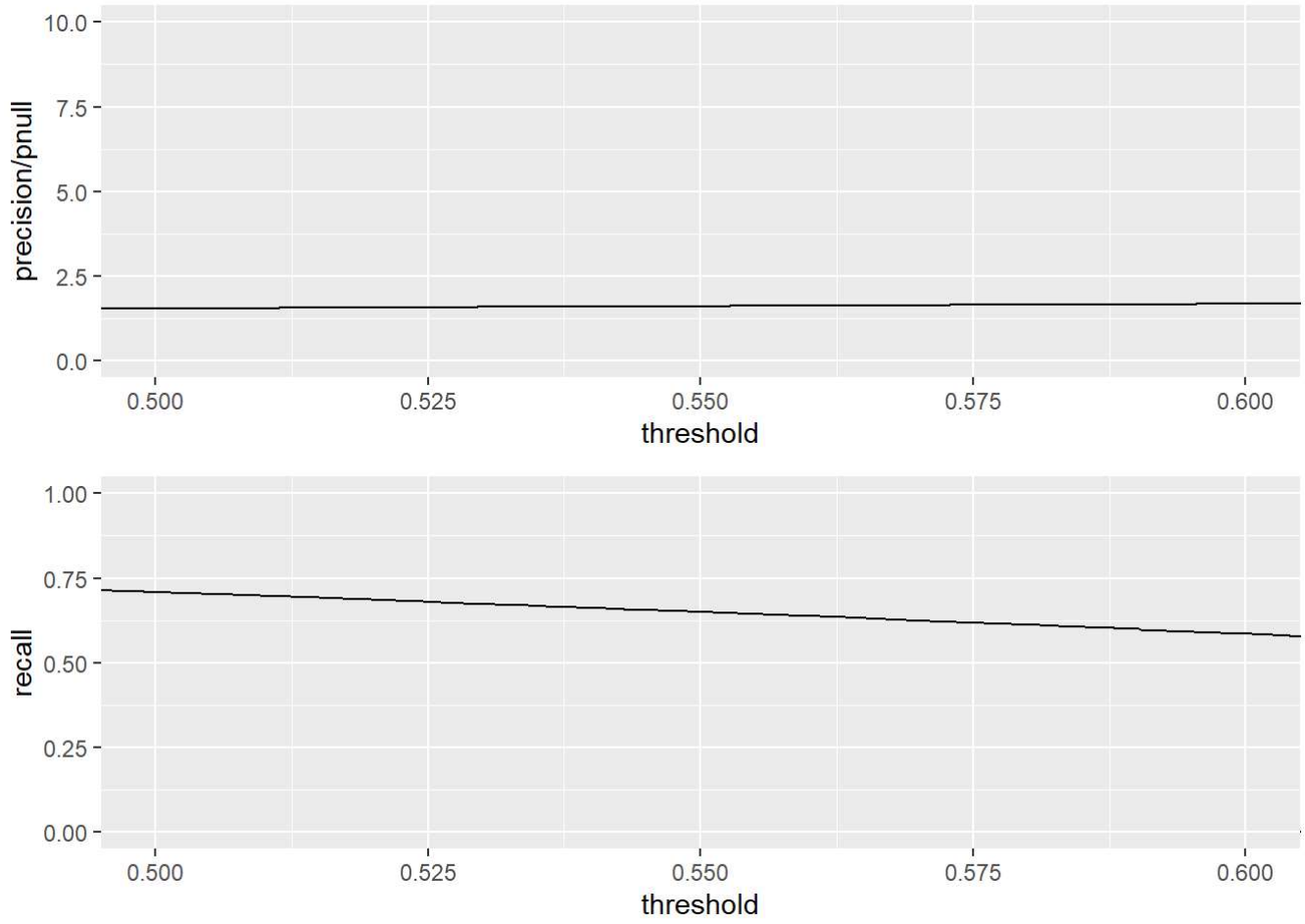
The code below creates a prediction object from the training dataset and then records its precision and recall for each possible value of the threshold

```
predObj <- prediction(dTrain$pred, factor(dTrain$target))
precObj <- performance(predObj, measure="prec")
recObj <- performance(predObj, measure="rec")
precision <- (precObj@y.values)[[1]]
prec.x <- (precObj@x.values)[[1]]
recall <- (recObj@y.values)[[1]]
rocFrame <- data.frame(threshold=prec.x,
                       precision=precision,
                       recall=recall)
```

The code below creates a null model (the average case observed) then creates two plots, one measuring how much better precision of the glm model is when compared against the null model and one measuring how well the recall performs across different thresholds.

```
predObj <- prediction(dTrain$pred, factor(dTrain$target))
precObj <- performance(predObj, measure="prec")
recObj <- performance(predObj, measure="rec")
precision <- (precObj@y.values)[[1]]
prec.x <- (precObj@x.values)[[1]]
recall <- (recObj@y.values)[[1]]
rocFrame <- data.frame(threshold=prec.x,
                       precision=precision,
                       recall=recall)


pnull <-mean(as.numeric(dTrain$target))

 p1 <- ggplot(rocFrame, aes(x=threshold)) +
  geom_line(aes(y=precision/pnull)) +
  coord_cartesian(xlim = c(0.5, 0.6), ylim=c(0,10) )

 p2 <- ggplot(rocFrame, aes(x=threshold)) +
   geom_line(aes(y=recall)) +
   coord_cartesian(xlim = c(0.5,0.6) )

 grid.arrange(p1,p2, nrow = 2)
```

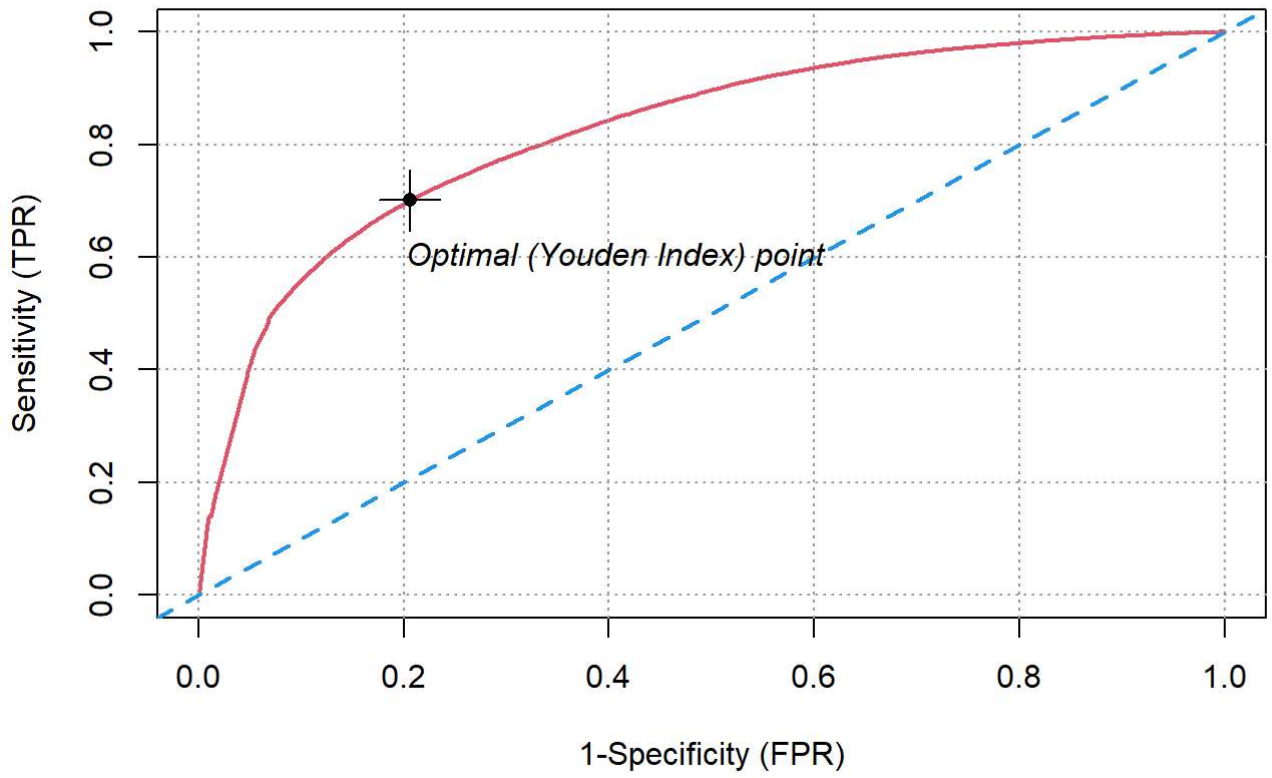
```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



Above we can see that for higher values for the threshod the recall deteriorates. However the precision only really get sufficiently better than the null model after 0.575.

Evaluating the model

```
lin_mod <- -4.96071 + 2.13683 * dTrain$predDAYS_LOST  +
  2.49284 * dTrain$predDEGREE_INJURY + 4.41491 * dTrain$predINJURY_SOURCE +
  1.76201 * dTrain$predNATURE_INJURY -0.82716  * dTrain$predINJ_BODY_PART +
  -3.34403 * dTrain$predNO_INJURIES + 1.65863*dTrain$predTOT_EXPER + 1.72812*dTrain$predDAYS_RES
TRICT

plot_roc(lin_mod, dTrain[[outcome]])
```




```
CM.Test <- table(truth = dTest$target, prediction = dTest$pred > 0.575 )

CM.Test
```

```
##      prediction
## truth FALSE TRUE
##     0  8908 1442
##     1  3841 6171
```

```
CM.Train <- table(truth = dTrain$target, prediction = dTrain$pred > 0.575)
CM.Train
```

```
##      prediction
## truth FALSE  TRUE
##     0 71439 11238
##     1 31170 50350
```

Above a confusion matrix can be constructed to show how accurate our model is given the threshold determined above. From this we can calculate the precision recall and accuracy of our model

```
precision.log.reg <- CM.Test[2,2]/sum(CM.Test[,2])
recall <- CM.Test[2,2]/sum(CM.Test[2,])
enrich <- precision.log.reg/mean(as.numeric(dTest$target))

enrich
```

```
## [1] 1.648539
```

We can see above that the regression model performs 1.65 times better than the created null model.

```
panderOptions("plain.ascii", TRUE)
panderOptions("keep.trailing.zeros", TRUE)
panderOptions("table.style", "simple")
perf_justify <- "lrrrr"


performanceMeasures <- function(pred, truth, name = "model") {
  ctable <- table(truth = truth, pred = (pred > 0.575))
  accuracy <- sum(diag(ctable)) / sum(ctable)
  precision <- ctable[2, 2] / sum(ctable[,2])
  recall <- ctable[2, 2] / sum(ctable[2, ])
  f1 <- 2 * precision * recall / (precision + recall)
  data.frame(model = name, precision = precision,
             recall = recall,
             f1 = f1, accuracy = accuracy)
}

trainperf <- performanceMeasures(dTrain$pred, dTrain$target,"logistic, training")

testperf <- performanceMeasures(
  dTest$pred,dTest$target, "logistic, test")

perftable <- rbind(trainperf, testperf)
pandoc.table(perftable, justify = perf_justify)
```

```
##
##
## model                  precision   recall       f1   accuracy
## ------------------- ----------- -------- -------- ----------
## logistic, training        0.8175   0.6176   0.7037     0.7417
## logistic, test            0.8106   0.6164   0.7003     0.7405
```

The above performance measures show that over 80% of the coal mines identified were in fact coal mines and 60% of the coal mines were not mislabeled as metal mines.

```
loglikelihood <- function(y, py){
  sum(y * log(py) + (1-y) * log(1-py))
}

null.dev <- -2*loglikelihood(dTest$target, pnull)
null.dev
```

```
## [1] 28223.97
```

Above the relatively high value for the null deviance can be seen

```
reg_model.dev <- -2*loglikelihood(dTest$target, dTest$pred)
reg_model.dev
```

```
## [1] 21070.27
```

Above the model deviance is shown

```
aic <- 2*length(reg_model$coefficients) - loglikelihood(dTest$target, dTest$pred)
aic
```

```
## [1] 10553.14
```

The AIC value determines whether this model behaves well considering the number of feature variables it uses. More time will be needed to try models with a different number of feature variables and the one with the lowest AIC would be selected. However, other models attempted in this project make up for this issue.

```
pr2 <- 1-(aic/null.dev)
pr2
```

```
## [1] 0.6260932
```

A relatively high pr2 is a sign that the deviance of the model constructed is significantly less than that of the null model

Random Forest Classification

Random forest is a more complicated model that can be used to classify some target variable. The model works by testing many different decision trees using different coefficients or feature variables and then eventualy landing on the best one.

preparing the data

```
all_data <- rbind(dTrain, dTest)

all_data$rgroup <- runif(dim(all_data)[[1]])
rf_train <- subset(all_data, rgroup <= 0.632)
rf_test <- subset(all_data, rgroup >0.632)
```

Preparing the formula

```
RandomForestFormula <- as.formula(paste('factor(target)',
                                        paste(regressionVars, collapse = '+'),
                                        sep = '~'))
```

Creating the model

```
rf_model <- randomForest(RandomForestFormula, data = dTrain,  ntree = 150, mtry = 4, importance
 = TRUE)

rf_model
```

```
##
## Call:
##  randomForest(formula = RandomForestFormula, data = dTrain, ntree = 150,      mtry = 4, impor
tance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 150
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 25.13%
## Confusion matrix:
##       0     1 class.error
## 0 64441 18236   0.2205692
## 1 23021 58499   0.2823970
```

```
rf_train$pred <- predict(rf_model, newdata = rf_train,
                             type = 'class')
rf_test$pred <- predict(rf_model, newdata = rf_test,
                            type = 'class')

cM_forest.test <- table(truth = rf_test$target, prediction = rf_test$pred)
cM_forest.test
```

```
##      prediction
## truth     0     1
##     0 29909  4418
##     1  5806 27876
```

```
cM_forest.train <- table(truth = rf_train$target, prediction = rf_train$pred)
cM_forest.train
```

```
##      prediction
## truth     0     1
##     0 50996  7704
##     1  9959 47891
```

The above confusion for the test and training data (respectively) demonstrate the performance of our new model.

```
performanceMeasures_forest <- function(pred, truth, name = "model") {
  ctable <- table(truth = truth, pred = (pred))
  accuracy <- sum(diag(ctable)) / sum(ctable)
  precision <- ctable[2, 2] / sum(ctable[,2])
  recall <- ctable[2, 2] / sum(ctable[2, ])
  f1 <- 2 * precision * recall / (precision + recall)
  data.frame(model = name, precision = precision,
             recall = recall,
             f1 = f1, accuracy = accuracy)
}

trainperf_forest <- performanceMeasures_forest(rf_train$pred, rf_train$target,"random forest, tr
aining")

testperf_forest <- performanceMeasures_forest(
  rf_test$pred,rf_test$target, "random forest test")

perftable <- rbind(trainperf_forest, testperf_forest)
pandoc.table(perftable, justify = perf_justify)
```

```
##
##
## model                       precision   recall       f1   accuracy
## ------------------------ ----------- -------- -------- ----------
## random forest, training        0.8614   0.8278   0.8443     0.8485
## random forest test             0.8632   0.8276   0.8450     0.8497
```

As can be seen above, the random forest classification model has a much better recall than the logistic regression model.

```
precision.forest <- cM_forest.test[2,2]/sum(cM_forest.test[2,])
enrich <- precision.forest/precision.log.reg

enrich
```

```
## [1] 1.021017
```

However we can see that the precision of the random forest model isn't significantly better than that of the glm.

Decision Tree Model Performance Evaluation

Using selected feature variables (as above) using the minimum AUC threshold from previous calculations the following variables are included in the decision tree model : categorical variables : “DEGREE_INJURY”,“INJURY_SOURCE”,“NATURE_INJURY”,“INJ_BODY_PART” numerical variables : “NO_INJURIES”,“TOT_EXPER”,“DAYS_RESTRICT”

```
library('rpart')

catVars <- c("DEGREE_INJURY","INJURY_SOURCE","NATURE_INJURY","INJ_BODY_PART")
numericVars <- c("NO_INJURIES","TOT_EXPER","DAYS_RESTRICT")

fV <- paste(outcome,'>0 ~ ',paste(c(numericVars, catVars), collapse=' + '),sep='')
tmodel <- rpart(fV,data=data)
```

Calculating the AUC using the selected feature variables for the decision tree for training, test, and calibration data sets.

```
library('ROCR')
calcAUC <- function(predcol,outcol)  {
  perf <- performance(prediction(predcol, outcol==pos), 'auc')
  as.numeric(perf@y.values)
}

 print(calcAUC(predict(tmodel,newdata=dTrain), dTrain[,outcome]))
```

```
## [1] 0.798436
```

```
 print(calcAUC(predict(tmodel,newdata=dTest), dTest[,outcome]))
```

```
## [1] 0.7973314
```

```
 print(calcAUC(predict(tmodel,newdata=dCal), dCal[,outcome]))
```

```
## [1] 0.8011721
```

It seems the model does generalise well for the selected feature variables as the training, test, and calibration data fed into the model have almost the same AUC for all of them.

To further increase the model accuracy, instead of using categorical values input into the decision tree model, reprocessed predicted numerical feature variable values were input into the model to calculate AUC.

```
tVars <- paste('pred',c(catVars, numericVars),sep='')
fV2 <- paste(outcome,'>0 ~ ',paste(tVars,collapse=' + '),sep='')
tmodel <- rpart(fV2,data=dTrain)
print(calcAUC(predict(tmodel,newdata=dTrain[,tVars]), dTrain[,outcome]))
```

```
## [1] 0.7889221
```

```
print(calcAUC(predict(tmodel,newdata=dTest[,tVars]), dTest[,outcome]))
```

```
## [1] 0.7862937
```

```
print(calcAUC(predict(tmodel,newdata=dCal[,tVars]), dCal[,outcome]))
```

```
## [1] 0.787758
```

By feeding the reprocessed numerical predicted feature variable values into the model this did not improve it further.

A function that measures performance : performanceMeasures was adopted below.

```
performanceMeasures <- function(pred, truth, name = "model") {
dev.norm <- -2 * logLikelihood(truth, pred)/length(pred)
ctable <- table(truth = truth==pos, pred = (pred > 0.5))
accuracy <- sum(diag(ctable)) / sum(ctable)
precision <- ctable[2, 2] / sum(ctable[,2])
recall <- ctable[2, 2] / sum(ctable[2, ])
f1 <- 2 * precision * recall / (precision + recall)
data.frame(model = name, precision = precision,
recall = recall,
f1 = f1, dev.norm = dev.norm)
}

logLikelihood <- function(outCol,predCol) {
sum(ifelse(outCol==pos,log(predCol),log(1-predCol)))
}
```

Selection for Pander display formatting

```
panderOpt <- function(){
library(pander)
# setting up Pander Options
panderOptions("plain.ascii", TRUE)
panderOptions("keep.trailing.zeros", TRUE)
panderOptions("table.style", "simple")
}
```

A prettier display function that shows performances in table format.

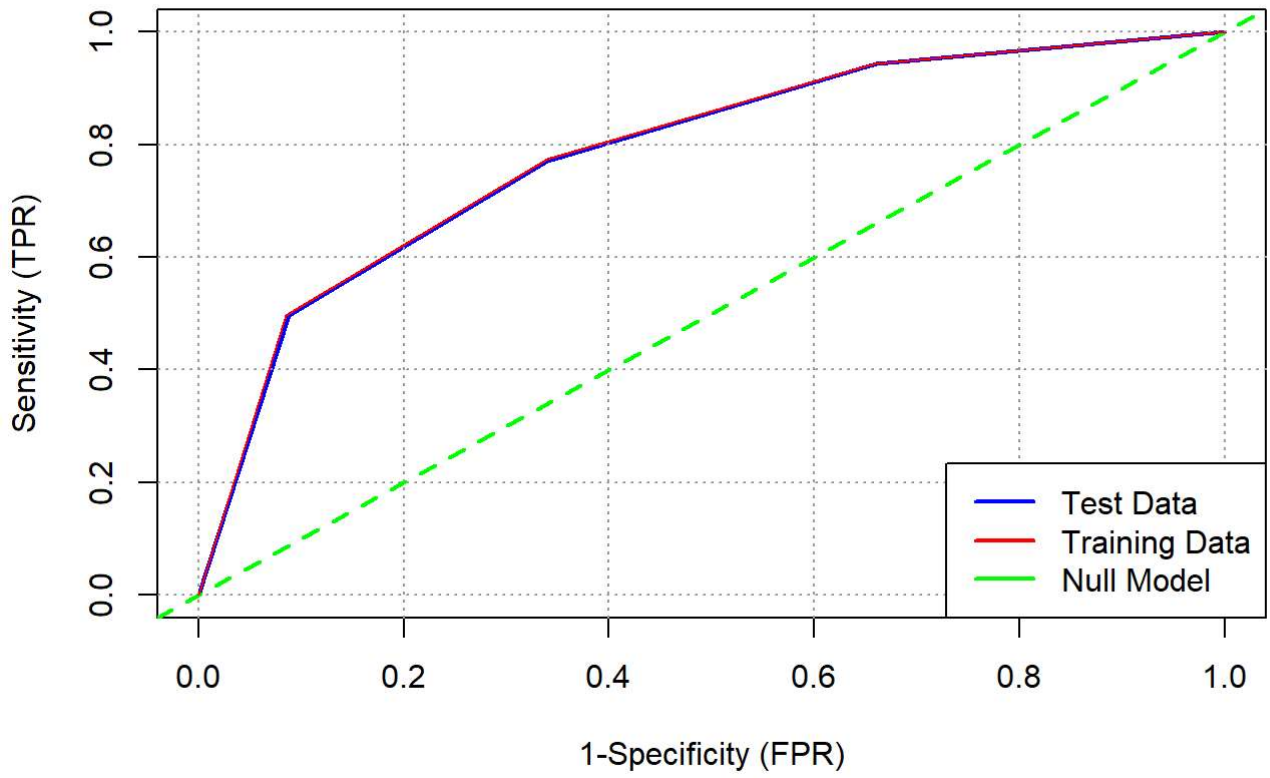
```
pretty_perf_table <- function(model,training,test){
# Option setting for Pander
panderOpt()
perf_justify <- "lrrrr"
# comparing performance on training vs. test
pred_train<-predict(model,newdata=training)
truth_train <- training[,outcome]
pred_test <- predict(model,newdata=test)
truth_test <- test[,outcome]
trainperf_tree <- performanceMeasures(
pred_train,truth_train,"training")
testperf_tree <- performanceMeasures(
pred_test,truth_test, "test")
perftable <- rbind(trainperf_tree, testperf_tree)
pandoc.table(perftable, justify = perf_justify)
}

pretty_perf_table(tmodel, dTrain, dTest)
```

```
##
##
## model        precision   recall       f1   dev.norm
## ---------- ----------- -------- -------- ----------
## training        0.6919   0.7749   0.7311      1.087
## test            0.6871   0.7713   0.7268      1.093
```

The performance shows the model does generalise well to testing data with average performance measures for precision, recall, and f1.

The ROC curve of the decision tree model performance is plotted below :

```
plot_roc <- function(predcol1, outcol1, predcol2, outcol2){
roc_1 <- rocit(score=predcol1,class=outcol1==pos)
roc_2 <- rocit(score=predcol2,class=outcol2==pos)
plot(roc_1, col = c("blue","green"), lwd = 3,
legend = FALSE,YIndex = FALSE, values = TRUE)
lines(roc_2$TPR ~ roc_2$FPR, lwd = 1,
col = c("red","green"))
legend("bottomright", col = c("blue","red", "green"),
c("Test Data", "Training Data", "Null Model"), lwd = 2)
}
pred_test_roc<-predict(tmodel,newdata=dTest)
pred_train_roc<-predict(tmodel,newdata=dTrain)

plot_roc(pred_test_roc, dTest[[outcome]],
pred_train_roc, dTrain[[outcome]])
```



The decision tree model performs much better than the Null model. The model does not overfit the training data and does generalise well by having the test data performance almost the same as the training data.

To improve the decision model further, the decision tree model's internal algorithm hyperparameters are modified below :

```
tmodel <- rpart(fV2,data=dTrain,control=rpart.control(cp=0.001,minsplit=1000,minbucket=1000,maxd
epth=5))
print(calcAUC(predict(tmodel,newdata=dTrain), dTrain[,outcome]))
```

```
## [1] 0.8178838
```

```
print(calcAUC(predict(tmodel,newdata=dTest), dTest[,outcome]))
```

```
## [1] 0.8151517
```

```
print(calcAUC(predict(tmodel,newdata=dCal), dCal[,outcome]))
```

```
## [1] 0.8169509
```

```
pretty_perf_table(tmodel, dTrain, dTest)
```

```
##
##
## model        precision   recall       f1   dev.norm
## ---------- ----------- -------- -------- ----------
## training        0.8001   0.6265   0.7027      1.031
## test            0.7963   0.6263   0.7012      1.038
```

By modifying the decision tree model's hyperparameters improvements in all AUC values and all performance measures are accomplished. By tweaking the decision tree model's internal algorithm hyperparameters the model's performance has been improved.

Comparing Logistic Regression, Random Forest, and Decision Tree model Performances :

The detailed performances of each model are located in each model section. Below shows a brief summary of all 3 model performances for comparison.

Logistic regression model :

model precision recall f1 dev.norm

training 0.8175 0.6176 0.7037 0.7417

test 0.8106 0.6164 0.7003 0.7405

Random forest model :

model precision recall f1 dev.norm

training 0.8614 0.8278 0.8443 0.8485

test 0.8632 0.8276 0.8450 0.8497

Decision tree model :

model precision recall f1 dev.norm

training 0.8001 0.6265 0.7027 1.031

test 0.7963 0.6263 0.7012 1.038

It can be seen from the results above that all the models were able to generalise well with the testing dataset having similar values to the training data set. In terms of precision the best was random forest then logistic regression and finally decision tree.

With recall the best was random forest then decision tree slightly ahead of logistic regression.

For F1 score the best was random forest with decision tree and logistic regression being around the same.

For deviance the lowest was random forest followed by logistic regression and the highest was decision tree model.

Overall the best performing model was the random forest model then logistic regression and last was decicion tree model.

Clustering Investigation and Explanations

1st part clustering with k-means

The clustering feature variable is the data set : DEGREE_INJURY This is the degree of injury recorded for an incident.

The numerical feature variables to calculate clustering are : TOT_EXPER , MINE_EXPER, and JOB_EXPER These numerical feature variables are the experience levels in time of the person that had the injury incident. With TOT_EXP being the total working experience, MINE_EXPER is the total mine experience, and JOB_EXP is the experience for that particular job.

```
library(dplyr)

myvars <- c("DEGREE_INJURY", "TOT_EXPER", "MINE_EXPER", "JOB_EXPER")

injury <- data[myvars]

summary(injury)
```

```
##  DEGREE_INJURY        TOT_EXPER       MINE_EXPER      JOB_EXPER
##  Length:202814      Min.   : 0.01   Min.   : 0.01   Min.   : 0.01
##  Class :character   1st Qu.: 2.00   1st Qu.: 0.69   1st Qu.: 1.00
##  Mode  :character   Median : 7.00   Median : 2.77   Median : 3.23
##                     Mean   :11.32   Mean   : 6.63   Mean   : 6.98
##                     3rd Qu.:20.00   3rd Qu.: 8.77   3rd Qu.:10.00
##                     Max.   :65.00   Max.   :65.00   Max.   :65.00
##                     NA's   :37400   NA's   :34325   NA's   :33746
```

Data cleaning is performed by removing empty values and “NO VALUE FOUND” values from all feature variables.

To perform clustering calculations the mean average for all the different experiences levels were calculated for each DEGREE_INJURY classification. This data has been prepared for hierarchical clustering and calculation of K-means to determine the best value of k, the number of clusters.

```
injury <- injury %>%
  filter(!is.na(TOT_EXPER))

injury <- injury %>%
  filter(!is.na(MINE_EXPER))

injury <- injury %>%
  filter(!is.na(JOB_EXPER))

injury <- injury[injury$DEGREE_INJURY != "NO VALUE FOUND",]

meanmine <- aggregate(MINE_EXPER ~ DEGREE_INJURY, data = injury, FUN = mean)
meantot <- aggregate(TOT_EXPER ~ DEGREE_INJURY, data = injury, FUN = mean)
meanjob <- aggregate(JOB_EXPER ~ DEGREE_INJURY, data = injury, FUN = mean)
combined <- merge(meanmine, meantot)

combineall <- merge(combined, meanjob, all = TRUE)

injury <- combineall
injury
```

DEGREE_INJURY <chr>	MINE_EXPER <dbl>	TOT_EXPER <dbl>	JOB_EXPER <dbl>
ALL OTHER CASES (INCL 1ST AID)	7.364799	10.819431	7.512778
DAYS AWAY FROM WORK ONLY	5.947447	12.000627	6.801209
DAYS RESTRICTED ACTIVITY ONLY	6.952202	9.254767	6.310709
DYS AWY FRM WRK & RESTRCTD ACT	6.802740	9.893754	6.595558
FATALITY	5.762639	13.749504	8.829009
INJURIES DUE TO NATURAL CAUSES	8.766588	15.292523	10.602366
INJURIES INVOLVNG NONEMPLOYEES	5.199245	9.031698	9.978868
NO DYS AWY FRM WRK,NO RSTR ACT	6.622901	10.779483	6.956360
OCCUPATNAL ILLNESS NOT DEG 1-6	12.030022	20.090797	11.458320
PERM TOT OR PERM PRTL DISABLTY	6.218908	11.530799	7.184802

1-10 of 10 rows

The results show the mean average for each experience for the different categories of DEGREE_INJURY.

The injury dataset is summarised below.

```
str(injury)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ DEGREE_INJURY: chr  "ALL OTHER CASES (INCL 1ST AID)" "DAYS AWAY FROM WORK ONLY" "DAYS REST
RICTED ACTIVITY ONLY" "DYS AWY FRM WRK & RESTRCTD ACT" ...
##  $ MINE_EXPER   : num  7.36 5.95 6.95 6.8 5.76 ...
##  $ TOT_EXPER    : num  10.82 12 9.25 9.89 13.75 ...
##  $ JOB_EXPER    : num  7.51 6.8 6.31 6.6 8.83 ...
```

```
vars.to.use <- colnames(injury)[-1]

pmatrix <- scale(injury[,vars.to.use])
```

Inputting data through k-means model. Summary of the clusters from k-means model.

```
kbest.p <- 2
pclusters <- kmeans(pmatrix, kbest.p,nstart=100, iter.max=100)
summary(pclusters)
```

```
##              Length Class  Mode
## cluster      10     -none- numeric
## centers       6     -none- numeric
## totss         1     -none- numeric
## withinss      2     -none- numeric
## tot.withinss  1     -none- numeric
## betweenss     1     -none- numeric
## size          2     -none- numeric
## iter          1     -none- numeric
## ifault        1     -none- numeric
```

The cluster centers.

```
pclusters$centers
```

```
##   MINE_EXPER  TOT_EXPER  JOB_EXPER
## 1 -0.4099549 -0.4032364 -0.3775973
## 2  1.6398197  1.6129456  1.5103891
```

The extracted cluster groupings.

```
groups <- pclusters$cluster
print_clusters <- function(labels, k) {
                    for(i in 1:k) {
                    print(paste("cluster", i))
                    print(injury[labels==i,c("DEGREE_INJURY","MINE_EXPER","TOT_EXPER","JOB_EXPE
R")])
                    }
}

print_clusters(groups, 2)
```

```
## [1] "cluster 1"
##                     DEGREE_INJURY MINE_EXPER TOT_EXPER JOB_EXPER
## 1  ALL OTHER CASES (INCL 1ST AID)   7.364799 10.819431  7.512778
## 2        DAYS AWAY FROM WORK ONLY   5.947447 12.000627  6.801209
## 3   DAYS RESTRICTED ACTIVITY ONLY   6.952202  9.254767  6.310709
## 4  DYS AWY FRM WRK & RESTRCTD ACT   6.802740  9.893754  6.595558
## 5                        FATALITY   5.762639 13.749504  8.829009
## 7  INJURIES INVOLVNG NONEMPLOYEES   5.199245  9.031698  9.978868
## 8  NO DYS AWY FRM WRK,NO RSTR ACT   6.622901 10.779483  6.956360
## 10 PERM TOT OR PERM PRTL DISABLTY   6.218908 11.530799  7.184802
## [1] "cluster 2"
##                    DEGREE_INJURY MINE_EXPER TOT_EXPER JOB_EXPER
## 6 INJURIES DUE TO NATURAL CAUSES   8.766588  15.29252  10.60237
## 9 OCCUPATNAL ILLNESS NOT DEG 1-6  12.030022  20.09080  11.45832
```

Using the fpc package, it contains a function called “kmeansruns” that estimates the best k.

```
library(fpc)

clustering.ch <- kmeansruns(
pmatrix, krange=1:4, criterion="ch")
clustering.ch$bestk
```

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

```
clustering.asw <- kmeansruns(
pmatrix, krange=1:4, criterion="asw")
clustering.asw$bestk
```

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

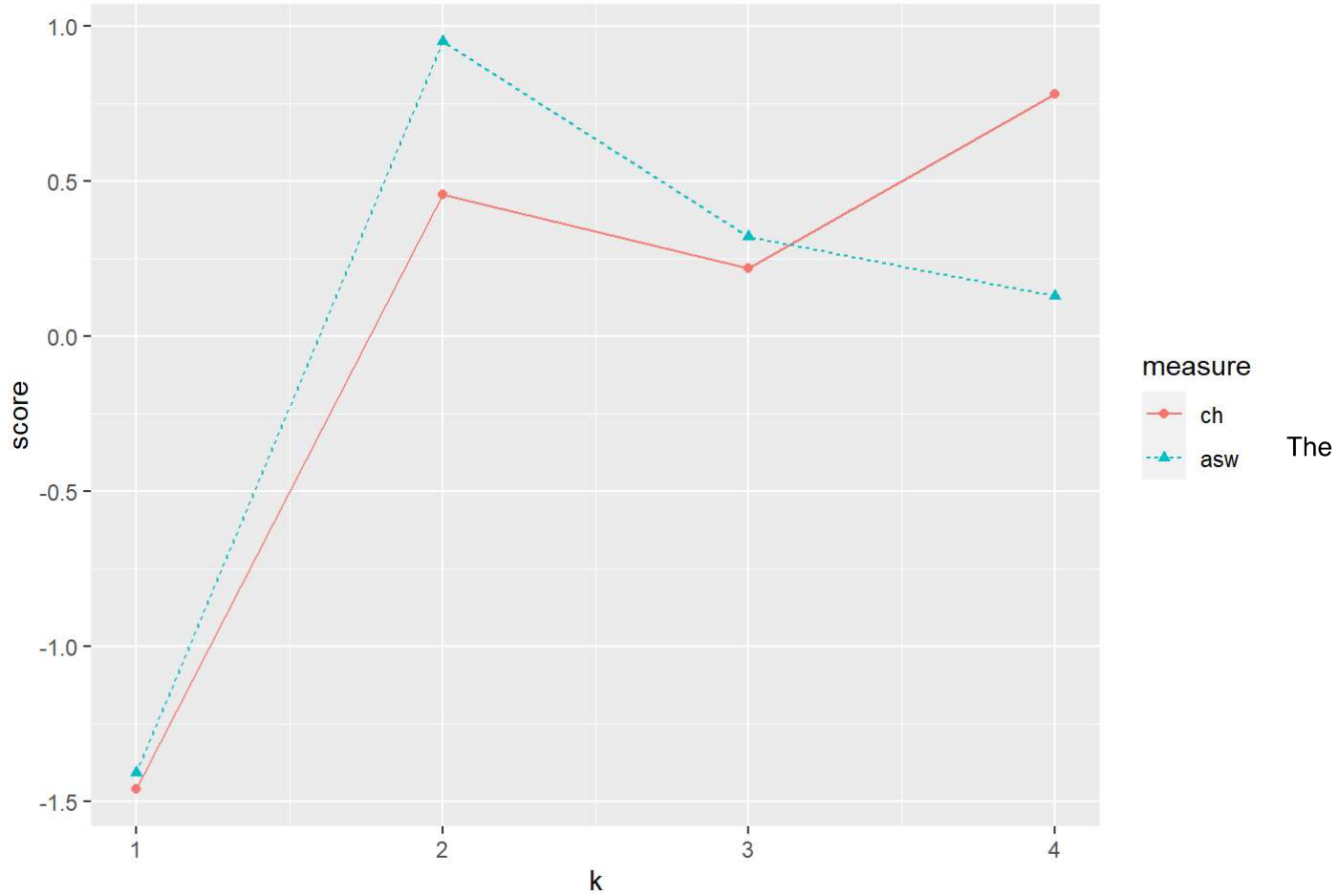
Comparing CH values for kmeans and hclust

```
clustering.ch$crit
```

```
## [1]  0.00000 18.76478 16.41784 21.91489
```

Plotting k against Calinski Harabasz Index(“ch”) and Average Silhouette Width(“asw”)

```
library(reshape2)

critframe <- data.frame(k=1:4, ch=scale(clustering.ch$crit),asw=scale(clustering.asw$crit))
critframe <- melt(critframe, id.vars=c("k"), variable.name="measure",value.name="score")

p <- ggplot(critframe, aes(x=k, y=score, color=measure)) + geom_point(aes(shape=measure)) + geom
_line(aes(linetype=measure)) +
scale_x_continuous(breaks=1:4, labels=1:4)

p
```



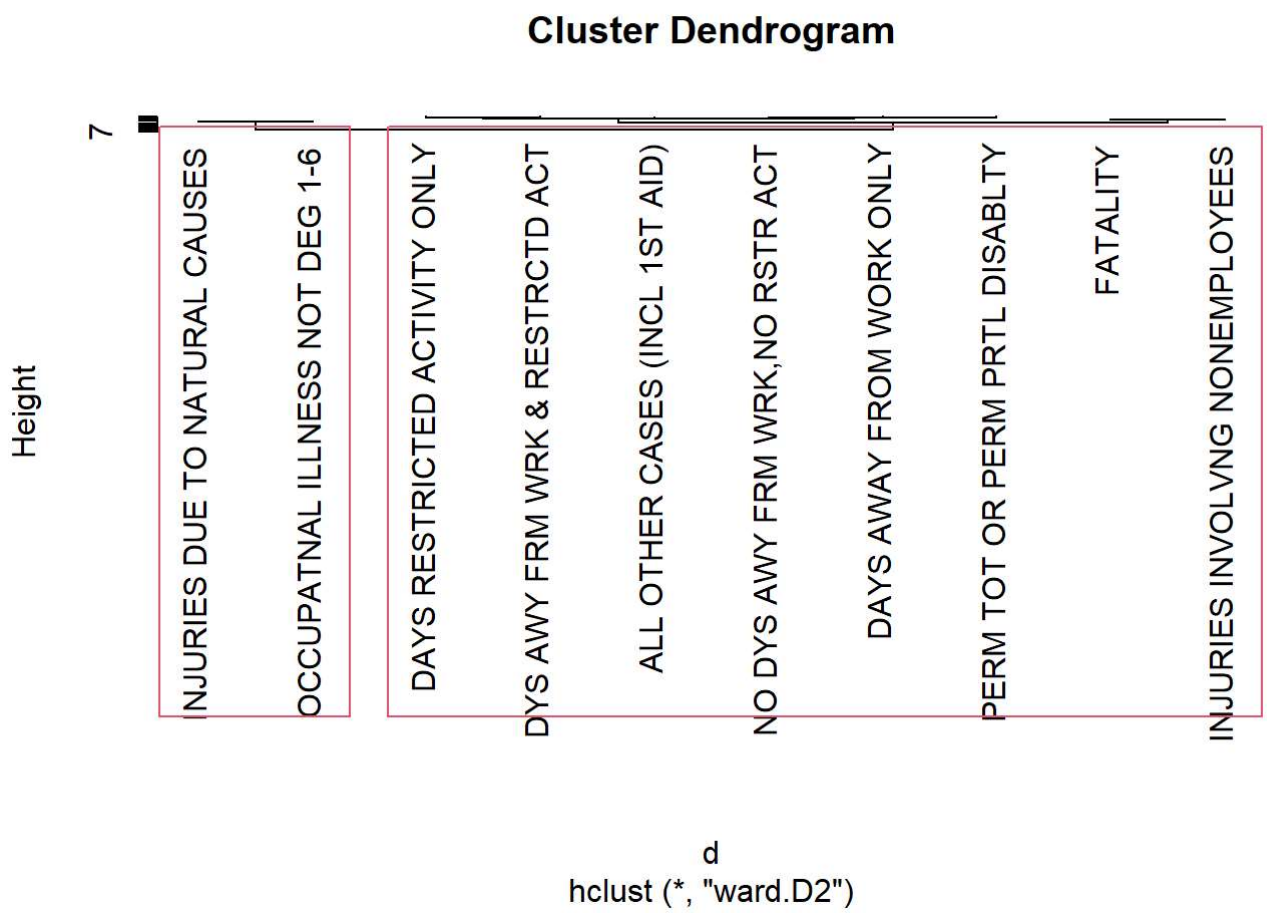
The plot shows k = 2 as the best value that both have correspondingly high Calinski Harabasz Index(“ch”) and Average Silhouette Width(“asw”) values.

From the plotted graph of the k values using 2 scoring methods, a k value of k = 2 was chosen.

2nd part clustering with hierarchical clustering

The euclidean method was chosen because the experiences are numerical values. Calculating scaled values center and standard deviation scaled. Using the hclust function for hierarchical clustering and using the ward.D2 method to cluster. From the k-means the best k = 2 clustering of 2 groups was chosen with cluster dendogram.

```
vars.to.use <- colnames(injury)[-1]
pmatrix <- scale(injury[,vars.to.use])


pcenter <- attr(pmatrix, "scaled:center")

pscale <- attr(pmatrix, "scaled:scale")

d <- dist(pmatrix, method="euclidean")

pfit <- hclust(d, method="ward.D2")
plot(pfit, labels=injury$DEGREE_INJURY)
rect.hclust(pfit, k=2)
```



The cutree function returns a vector of cluster group assignment for each row by specifying the cluster group number between 1-2.

```
groups <- cutree(pfit, k=2)
groups
```

```
##  [1] 1 1 1 1 1 2 1 1 2 1
```

Showing the 2 cluster groups.

```
print_clusters(groups, 2)
```

```
## [1] "cluster 1"
##                      DEGREE_INJURY MINE_EXPER TOT_EXPER JOB_EXPER
## 1  ALL OTHER CASES (INCL 1ST AID)   7.364799 10.819431  7.512778
## 2        DAYS AWAY FROM WORK ONLY   5.947447 12.000627  6.801209
## 3   DAYS RESTRICTED ACTIVITY ONLY   6.952202  9.254767  6.310709
## 4  DYS AWY FRM WRK & RESTRCTD ACT   6.802740  9.893754  6.595558
## 5                         FATALITY   5.762639 13.749504  8.829009
## 7  INJURIES INVOLVNG NONEMPLOYEES   5.199245  9.031698  9.978868
## 8  NO DYS AWY FRM WRK,NO RSTR ACT   6.622901 10.779483  6.956360
## 10 PERM TOT OR PERM PRTL DISABLTY   6.218908 11.530799  7.184802
## [1] "cluster 2"
##                     DEGREE_INJURY MINE_EXPER TOT_EXPER JOB_EXPER
## 6 INJURIES DUE TO NATURAL CAUSES   8.766588  15.29252  10.60237
## 9 OCCUPATNAL ILLNESS NOT DEG 1-6  12.030022  20.09080  11.45832
```

Preparing data for visualising cluster.

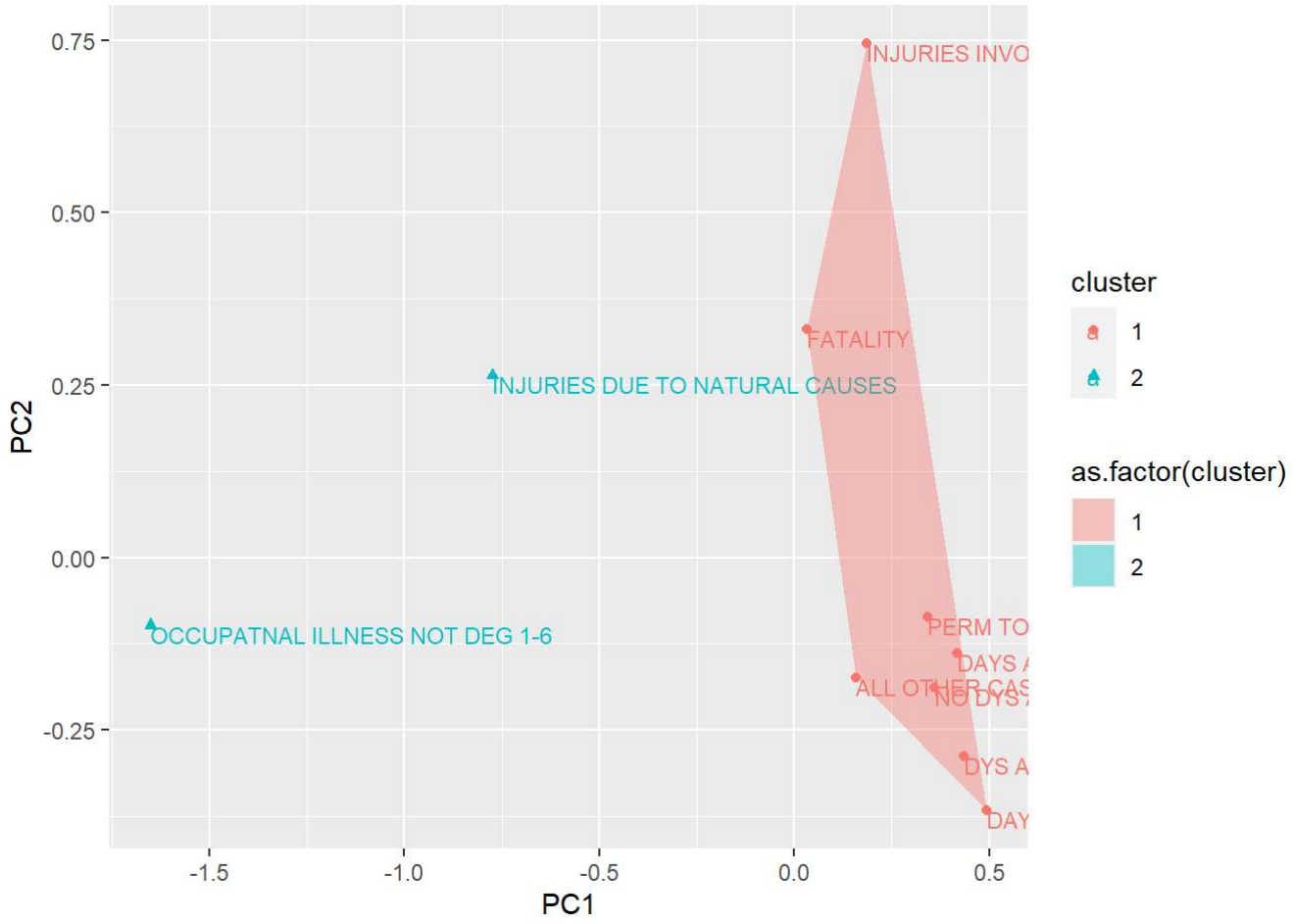
```
library(ggplot2)
# Calculate the principle components of pmatrix
princ <- prcomp(pmatrix)
nComp <- 2
project <- as.data.frame(predict(princ, newdata=pmatrix)[,1:nComp])
project.plus <- cbind(project,cluster=as.factor(groups),DegreeInjury=injury$DEGREE_INJURY)
```

Finding convex hull of cluster.

```
# finding convex hull
library('grDevices')
h <- do.call(rbind,lapply(unique(groups), function(c)
                { f <- subset(project.plus,cluster==c);f[chull(f),]
                }
                        )
            )
```

Plotting the 2 clusters.

```
p <- ggplot(project.plus, aes(x=PC1, y=PC2)) + geom_point(aes(shape=cluster, color=cluster)) + g
eom_text(aes(label=DegreeInjury, color=cluster),
hjust=0, vjust=1, size=3) + geom_polygon(data=h,aes(group=cluster,fill=as.factor(cluster)),alpha
=0.4,linetype=0)
p
```



We will assess whether these groupings are stable and true groupings using Jaccard coefficient and clusterboot's algorithm which is available from the fpc package.

Assessing whether the cluster represents true structure. Running clusterboot.

```
library(fpc)
kbest.p <- 2
cboot.hclust <- clusterboot(
pmatrix, clustermethod=hclustCBI,
method="ward.D2", k=kbest.p)
```

```
## boot 1
## boot 2
## boot 3
## boot 4
## boot 5
## boot 6
## boot 7
## boot 8
## boot 9
## boot 10
## boot 11
## boot 12
## boot 13
## boot 14
## boot 15
## boot 16
## boot 17
## boot 18
## boot 19
## boot 20
## boot 21
## boot 22
## boot 23
## boot 24
## boot 25
## boot 26
## boot 27
## boot 28
## boot 29
## boot 30
## boot 31
## boot 32
## boot 33
## boot 34
## boot 35
## boot 36
## boot 37
## boot 38
## boot 39
## boot 40
## boot 41
## boot 42
## boot 43
## boot 44
## boot 45
## boot 46
## boot 47
## boot 48
## boot 49
## boot 50
## boot 51
## boot 52
```

```
## boot 53
## boot 54
## boot 55
## boot 56
## boot 57
## boot 58
## boot 59
## boot 60
## boot 61
## boot 62
## boot 63
## boot 64
## boot 65
## boot 66
## boot 67
## boot 68
## boot 69
## boot 70
## boot 71
## boot 72
## boot 73
## boot 74
## boot 75
## boot 76
## boot 77
## boot 78
## boot 79
## boot 80
## boot 81
## boot 82
## boot 83
## boot 84
## boot 85
## boot 86
## boot 87
## boot 88
## boot 89
## boot 90
## boot 91
## boot 92
## boot 93
## boot 94
## boot 95
## boot 96
## boot 97
## boot 98
## boot 99
## boot 100
```

Summary of clusterboot results.

```
summary(cboot.hclust$result)
```

```
##               Length Class  Mode
## result         7     hclust list
## noise          1     -none- logical
## nc             1     -none- numeric
## clusterlist    2     -none- list
## partition     10     -none- numeric
## clustermethod  1     -none- character
## nccl           1     -none- numeric
```

Printing out groupings.

```
groups <- cboot.hclust$result$partition
print_clusters(groups, 2)
```

```
## [1] "cluster 1"
##                      DEGREE_INJURY MINE_EXPER TOT_EXPER JOB_EXPER
## 1  ALL OTHER CASES (INCL 1ST AID)   7.364799 10.819431  7.512778
## 2        DAYS AWAY FROM WORK ONLY   5.947447 12.000627  6.801209
## 3   DAYS RESTRICTED ACTIVITY ONLY   6.952202  9.254767  6.310709
## 4  DYS AWY FRM WRK & RESTRCTD ACT   6.802740  9.893754  6.595558
## 5                        FATALITY   5.762639 13.749504  8.829009
## 7  INJURIES INVOLVNG NONEMPLOYEES   5.199245  9.031698  9.978868
## 8  NO DYS AWY FRM WRK,NO RSTR ACT   6.622901 10.779483  6.956360
## 10 PERM TOT OR PERM PRTL DISABLTY   6.218908 11.530799  7.184802
## [1] "cluster 2"
##                    DEGREE_INJURY MINE_EXPER TOT_EXPER JOB_EXPER
## 6 INJURIES DUE TO NATURAL CAUSES   8.766588  15.29252  10.60237
## 9 OCCUPATNAL ILLNESS NOT DEG 1-6  12.030022  20.09080  11.45832
```

Calculating Jaccard coefficient over all the bootstrap iterations.

```
1-cboot.hclust$bootbrd/100
```

```
## [1] 0.98 0.83
```

```
sqr_edist <- function(x, y) {
             sum((x-y)^2)
             }

wss.cluster <- function(clustermat) {
             c0 <- apply(clustermat, 2, FUN=mean)
             sum(apply(clustermat, 1,FUN=function(row){sqr_edist(row,c0)}))
             }

wss.total <- function(dmatrix, labels) {
             wsstot <- 0
             k <- length(unique(labels))
               for(i in 1:k){
                 wsstot <- wsstot +
                 wss.cluster(subset(dmatrix, labels==i))
                 }
                 wsstot
             }

totss <- function(dmatrix) {
        grandmean <- apply(dmatrix, 2, FUN=mean)
          sum(apply(dmatrix, 1,FUN=function(row){
          sqr_edist(row, grandmean)
          }
          )
        )
        }
```

Unstable and low certainty clusters have Jaccard coefficients : 0.6 < Jaccard coefficients < 0.75 Highly stable real clusters have Jaccard coefficients > 0.85 These results shown above of 1.00 for Group 1 and 0.81 for Group 2 show that both the cluster groups are highly stable.

Function to calculate the Calinski-Harabasz Index

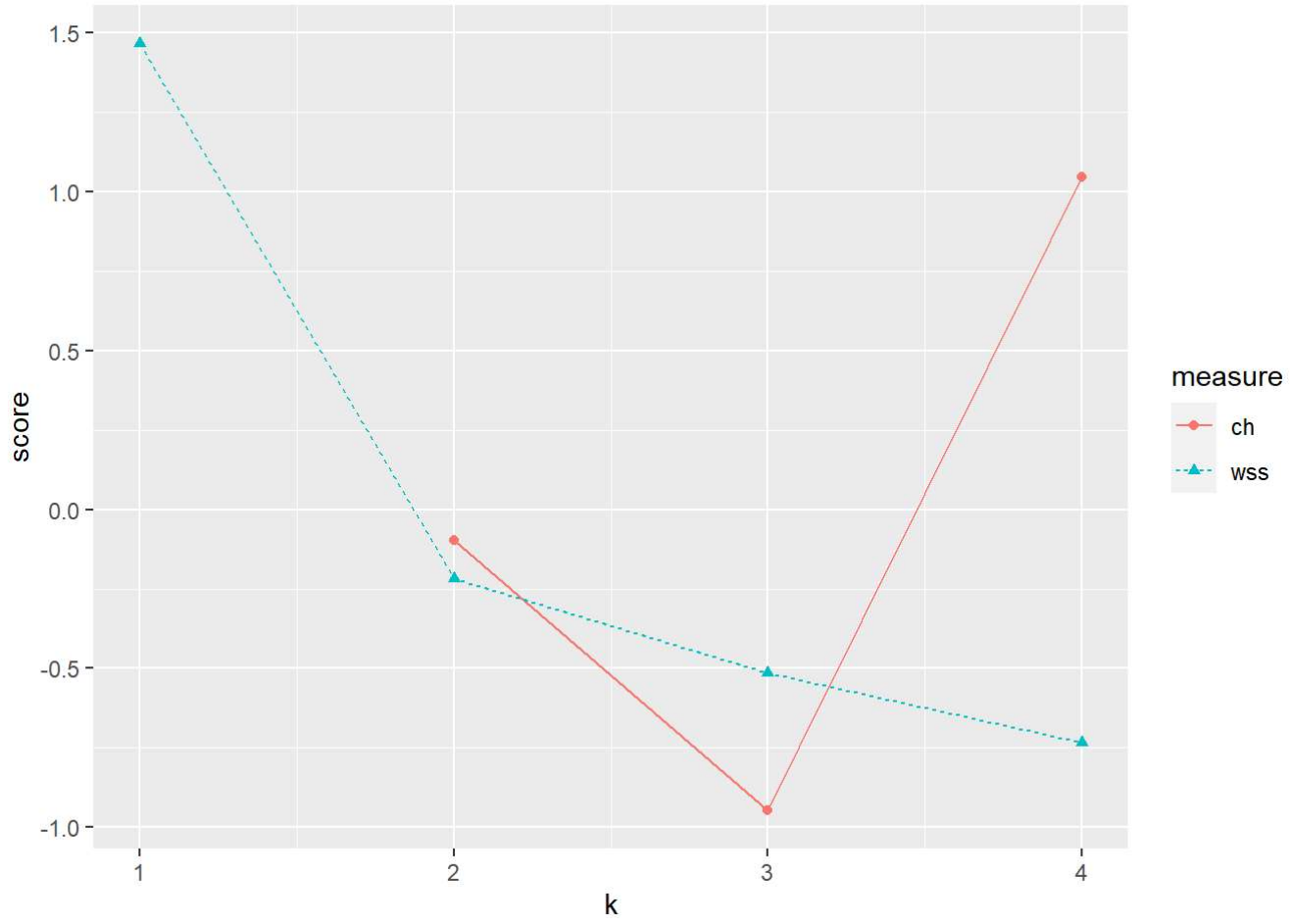
```
ch_criterion <- function(dmatrix, kmax, method="kmeans") {
if(!(method %in% c("kmeans", "hclust"))){
stop("method must be one of c('kmeans', 'hclust')")
}
npts <- dim(dmatrix)[1] # number of rows.
totss <- totss(dmatrix)
wss <- numeric(kmax)
crit <- numeric(kmax)
wss[1] <- (npts-1)*sum(apply(dmatrix, 2, var))
for(k in 1:kmax) {
if(method=="kmeans") {
clustering<-kmeans(dmatrix, k, nstart=4, iter.max=100)
wss[k] <- clustering$tot.withinss
}else { # hclust
d <- dist(dmatrix, method="euclidean")
pfit <- hclust(d, method="ward.D2")
labels <- cutree(pfit, k=k)
wss[k] <- wss.total(dmatrix, labels)
}
}
bss <- totss - wss
crit.num <- bss/(0:(kmax-1))
crit.denom <- wss/(npts - 1:kmax)
list(crit = crit.num/crit.denom, wss = wss, totss = totss)
}
```

Plotting both the Calinski-Harabasz index and Average Silhouette Width(“asw”)

```
library(reshape2)
clustcrit <- ch_criterion(pmatrix, 4, method="hclust")
critframe <- data.frame(k=1:4, ch=scale(clustcrit$crit),
wss=scale(clustcrit$wss))
critframe <- melt(critframe, id.vars=c("k"),
variable.name="measure",
value.name="score")
p<-ggplot(critframe, aes(x=k, y=score, color=measure)) +
geom_point(aes(shape=measure)) +
geom_line(aes(linetype=measure)) +
scale_x_continuous(breaks=1:4, labels=1:4)

p
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



The plot shows k = 2 as the best value which also corresponds well with the k-means plot that also shows k = 2 as the best cluster number.

From the plotted graph of the k values using both hierarchical and k-means methods, a k value of k = 2 would be the best clustering grouping.

From the Jaccard coefficients : 0.98 0.83 both groupings have high Jaccard coefficients that indicate highly stable clusters.

Since this data is numeric, a numerical distance measurement was chosen. The distance measure used is Euclidean distance with the mean value and standard deviation scaled center.