Accidents data set

1 Description

In the event of a car accident, there may be limited resources available for dealing with the ensuing property damage and injuries. In particular, there may be a limited number of people available with the ability to deliver high end medical attention if serious injuries have resulted from the accident. It would be useful if we could predict whether or not a serious injury resulted from the accident at the time the accident is reported. This could help us decide what kind of medical personnel should be sent out initially. To this end, 42,183 observations have been collected on automobile accidents. For each accident you have additional type of information, such as day of week, weather conditions, and road type.

- `HOUR_I_R`: 1 = rush hour, 0 = not (rush = 6-9 am, 4-7 pm)
- `ALCHL_I`: Alcohol involved = 1, not involved = 2
- `ALIGN_I`: 1 = straight, 2 = curve
- `STRATUM_R`: 1 = NASS Crashes Involving At Least One Passenger Vehicle (i.e., A Passenger Car, Sport Utility Vehicle, Pickup Truck Or Van) Towed Due To Damage From The Crash Scene And No Medium Or Heavy Trucks Are Involved. 0 = not
- `WRK_ZONE`: 1= yes, 0= no
- `WKDY_I_R`: 1=weekday, 0=weekend
- `INT_HWY`: Interstate? 1=yes, 0= no
- `LGTCON_I_R`: Light conditions - 1=day, 2=dark (including dawn/dusk), 3=dark, but lighted,4=dawn or dusk
- `MANCOL_I`: 0=no collision, 1=head-on, 2=other form of collision
- `PED_ACC_R`: 1=pedestrian/cyclist involved, 0=not
- `RELJCT_I_R`: 1=accident at intersection/interchange, 0=not at intersection
- `REL_RWY_R`: 1=accident on roadway, 0=not on roadway
- `PROFIL_I_R`: 1= level, 0=other
- `SPD_LIM`: Speed limit, miles per hour
- `SUR_CON`: Surface conditions (1=dry, 2=wet, 3=snow/slush, 4=ice, 5=sand/dirt/oil, 8=other, 9=unknown)
- `TRAF_CON_R`: Traffic control device: 0=none, 1=signal, 2=other (sign, officer, . . .)
- `TRAF_WAY`: 1=two-way traffic, 2=divided hwy, 3=one-way road
- `WEATHER_R`: 1=no adverse conditions, 2=rain, snow or other adverse condition
- `INJURY`: 1 = yes, 0 = no

2 Preprocessing

We download the data and preprocess it for our purposes. Original data had some additional columns that we do not care for at this stage.

```
download.file(
    'https://github.com/ChicagoBoothML/MLClassData/raw/master/TransportAccidents/Accidents.csv',
    'Accidents.csv')
accidents_df = read.csv("Accidents.csv")

n = nrow(accidents_df)

accidents_df$INJURY = rep(1, n)
accidents_df$INJURY[accidents_df$MAX_SEV_IR == 0] = 0
```

```
drops = c("MAX_SEV_IR", "FATALITIES", "PRPTYDMG_CRASH", "NO_INJ_I", "INJURY_CRASH", "VEH_INVL")
accidents_df = accidents_df[, !(names(accidents_df) %in% drops)]
```

If we care about making decisions whether to dispatch highly skillful medical staff, some information is not going to be available to us. For example, we may know if it is a rush hour or not, however, we may lack information about whether alcohol was involved or not. It is important to make sure that variables used to build a classifier are actually available at the time a classifier is used.

I will drop the following variables from creating a classifies: `ALCHL_I`, `STRATUM_R`, `MANCOL_I`, `PED_ACC_R`, `SUR_CON`, `TRAF_CON_R`

```
drops = c("ALCHL_I", "STRATUM_R", "MANCOL_I", "PED_ACC_R", "SUR_CON", "TRAF_CON_R")
accidents_df = accidents_df[, !(names(accidents_df) %in% drops)]
```

Next, we split data into train and test set. 80% of observations are kept in the training set.

```
set.seed(1)
train_ind = sample.int(n, floor(0.8*n))
accidents_df_train = accidents_df[train_ind,]
accidents_df_test = accidents_df[-train_ind,]

# every variable is categorical variable, tell R that
accidents_df_train[] = lapply(accidents_df_train, factor)
accidents_df_test[] = lapply(accidents_df_test, factor)
```

Finally, we will need some libraries to perform analyis.

```
library(randomForest)
library(gbm)
library(rpart)
library(rpart.plot)
library(e1071)   # for naive bayes
```

3 Predicting majority class

Without any information, we can think of accidents with injuries and without injuries as i.i.d. Bernoulli(p). Our best guess for p is fraction of accidents with injuries out of all injuries.

```
(tb_INJURY_train = table(accidents_df_train$INJURY))
```

```
##
##     0     1
## 16564 17182
```

```
# estimated probability of injury
p_INJURY = tb_INJURY_train["1"] / (sum(tb_INJURY_train)); print(p_INJURY)
```

```
##     1
## 0.509
```

It seems that the number of accidents with injury and without an injury is about the same. There are a bit more accidents with injury. Let us see what happens if we predict that every accident in the test set involves an injury.

```
tb_INJURY_test = table(predictions = rep(1, nrow(accidents_df_test)), actual = accidents_df_test$INJURY)
rownames(tb_INJURY_test) = c("predict_INJURY")
print(tb_INJURY_test)
```

```
##                 actual
## predictions        0    1
##   predict_INJURY 4157 4280
```

Accuracy of this approach would be

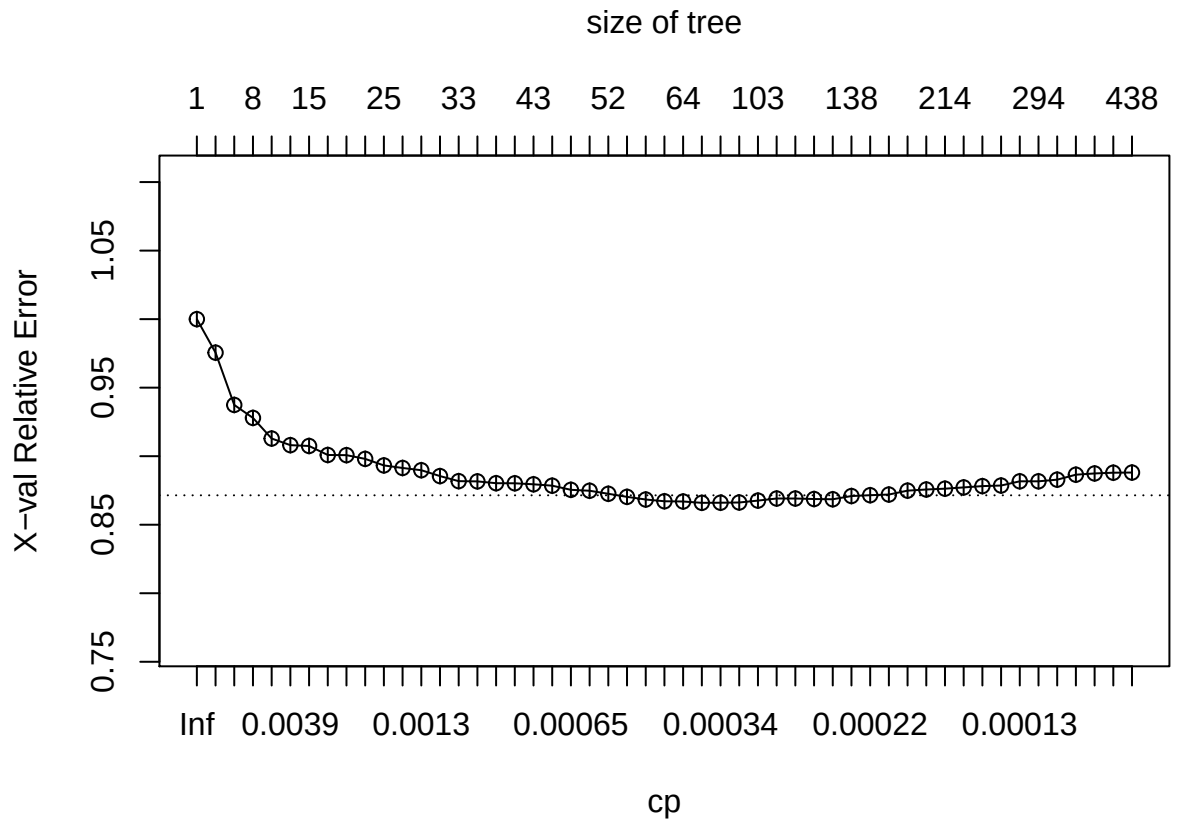
4 Decision Trees

We start by creating a big tree.

```
big_tree = rpart(INJURY~., data=accidents_df_train,
                 control=rpart.control(minsplit=10,
                                       cp=0.0001,
                                       xval=10)
                 )
```

Next, we investigate `cptable` to find the a good value for the cp parameter

```
plotcp(big_tree)
```




```
cptable = printcp(big_tree)
```

```
##
## Classification tree:
## rpart(formula = INJURY ~ ., data = accidents_df_train, control = rpart.control(minsplit = 10,
##     cp = 1e-04, xval = 10))
##
## Variables actually used in tree construction:
##  [1] ALIGN_I    HOUR_I_R   INT_HWY    LGTCON_I_R
```

```
##  [5] MANCOL_I_R PROFIL_I_R RELJCT_I_R REL_RWY_R
##  [9] SPD_LIM    SUR_COND   TRAF_WAY   WEATHER_R
## [13] WKDY_I_R   WRK_ZONE
##
## Root node error: 16564/33746 = 0.5
##
## n= 33746
##
##       CP nsplit rel error xerror  xstd
## 1  2e-02      0       1.0    1.0 0.006
## 2  1e-02      1       1.0    1.0 0.006
## 3  6e-03      3       1.0    0.9 0.006
## 4  6e-03      7       0.9    0.9 0.006
## 5  4e-03     10       0.9    0.9 0.006
## 6  4e-03     13       0.9    0.9 0.006
## 7  3e-03     14       0.9    0.9 0.006
## 8  3e-03     16       0.9    0.9 0.006
## 9  2e-03     19       0.9    0.9 0.006
## 10 2e-03     23       0.9    0.9 0.006
## 11 2e-03     24       0.9    0.9 0.006
## 12 1e-03     25       0.9    0.9 0.006
## 13 1e-03     27       0.9    0.9 0.006
## 14 1e-03     30       0.9    0.9 0.005
## 15 1e-03     32       0.9    0.9 0.005
## 16 9e-04     36       0.9    0.9 0.005
## 17 9e-04     38       0.9    0.9 0.005
## 18 9e-04     39       0.9    0.9 0.005
## 19 8e-04     42       0.9    0.9 0.005
## 20 7e-04     45       0.9    0.9 0.005
## 21 6e-04     48       0.8    0.9 0.005
## 22 6e-04     50       0.8    0.9 0.005
## 23 5e-04     51       0.8    0.9 0.005
## 24 5e-04     55       0.8    0.9 0.005
## 25 4e-04     58       0.8    0.9 0.005
## 26 4e-04     59       0.8    0.9 0.005
## 27 4e-04     63       0.8    0.9 0.005
## 28 3e-04     67       0.8    0.9 0.005
## 29 3e-04     75       0.8    0.9 0.005
## 30 3e-04     77       0.8    0.9 0.005
## 31 3e-04    102       0.8    0.9 0.005
## 32 3e-04    105       0.8    0.9 0.005
## 33 3e-04    109       0.8    0.9 0.005
## 34 3e-04    118       0.8    0.9 0.005
## 35 2e-04    124       0.8    0.9 0.005
## 36 2e-04    137       0.8    0.9 0.005
## 37 2e-04    143       0.8    0.9 0.005
## 38 2e-04    154       0.8    0.9 0.005
## 39 2e-04    160       0.8    0.9 0.005
## 40 2e-04    196       0.8    0.9 0.005
## 41 2e-04    213       0.8    0.9 0.005
## 42 2e-04    226       0.8    0.9 0.005
## 43 1e-04    263       0.8    0.9 0.005
## 44 1e-04    269       0.8    0.9 0.005
## 45 1e-04    272       0.8    0.9 0.005
```

```
## 46 1e-04    293       0.8    0.9 0.005
## 47 1e-04    367       0.8    0.9 0.005
## 48 1e-04    373       0.8    0.9 0.005
## 49 1e-04    378       0.8    0.9 0.005
## 50 1e-04    395       0.8    0.9 0.005
## 51 1e-04    437       0.8    0.9 0.005
```

```
# this is the cp parameter with smallest cv-errror
(index_cp_min = which.min(cptable[,"xerror"]))
```

```
## 28
## 28
```

```
(cp_min = cptable[ index_cp_min, "CP" ])
```

```
## [1] 0.000342
```

```
# one standard deviation rule
# need to find first cp value for which the xerror is below horizontal line on the plot
(val_h = cptable[index_cp_min, "xerror"] + cptable[index_cp_min, "xstd"])
```

```
## [1] 0.871
```

```
(index_cp_std = Position(function(x) x < val_h, cptable[, "xerror"]))
```

```
## [1] 24
```

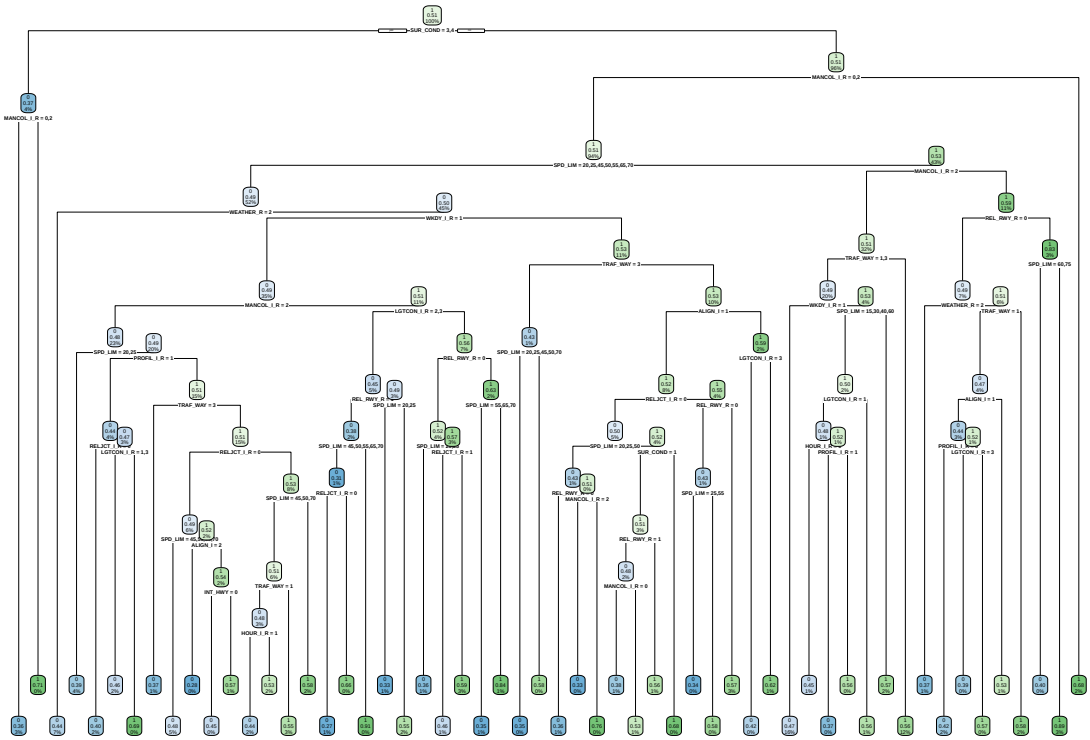
```
(cp_std = cptable[ index_cp_std, "CP" ])
```

```
## [1] 0.000483
```

Finally, we use this cp value to prune the big tree.

```
optimal.tree = prune(big_tree, cp=cp_std)
rpart.plot(optimal.tree)
```

```
## Warning: labs do not fit even at cex 0.15, there
## may be some overplotting
```



```
length(unique(optimal.tree$where))    # number of leaves
```

```
## [1] 56
```

The following command will obtain predictions for the tree. It directly outputs the class label of the test examples.

```
optimal_tree_predictions = predict(optimal.tree, accidents_df_test,
                                   type="class"  # this parameter tells R to predict classes
                                   )
```

The optimal tree has the following miss-classification rate

```
(1 - mean(optimal_tree_predictions == accidents_df_test$INJURY))  # error rate using all variables
```

```
## [1] 0.433
```

and the corresponding confusion matrix

```
tb_tree = table(predictions = optimal_tree_predictions,
                actual = accidents_df_test$INJURY)
rownames(tb_tree) = c("predict_NO_INJURY", "predict_INJURY")
print(tb_tree)
```

```
##                    actual
## predictions           0    1
##   predict_NO_INJURY 2499 1994
##   predict_INJURY    1658 2286
```

5 Naive Bayes classifier

This is a popular classifier that uses Bayes theorem to make decisions. We will talk more about this classifier later in the class. This is a linear clasifier.

```
nb_model = naiveBayes(INJURY ~ ., accidents_df_train)
```

Next, we compute error on test data.

```
nb_test_predictions = predict(nb_model, accidents_df_test)
```

Miss-classification rate is

```
1 - mean(nb_test_predictions == accidents_df_test$INJURY)
```

```
## [1] 0.452
```

and the corresponding confusion matrix

```
tb_tree = table(predictions = nb_test_predictions,
                actual = accidents_df_test$INJURY)
rownames(tb_tree) = c("predict_NO_INJURY", "predict_INJURY")
print(tb_tree)
```

```
##                    actual
## predictions           0    1
##   predict_NO_INJURY 1775 1432
##   predict_INJURY    2382 2848
```

We can observe that the miss-classification rate is worse compared to decision trees. However, notice that the type of predictions is quite different as well.

6 Random Forest Model

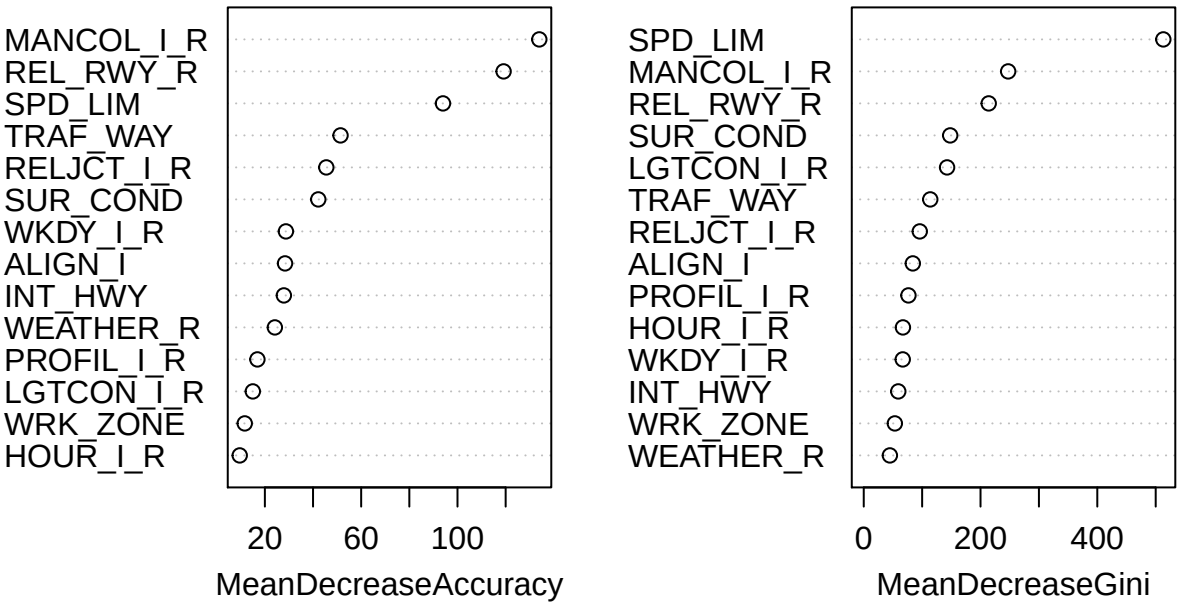
We build a model a random tree model. There are two parameters:

- number of variables to use when building each tree
- total number of trees

However, we also control the size of grown trees.

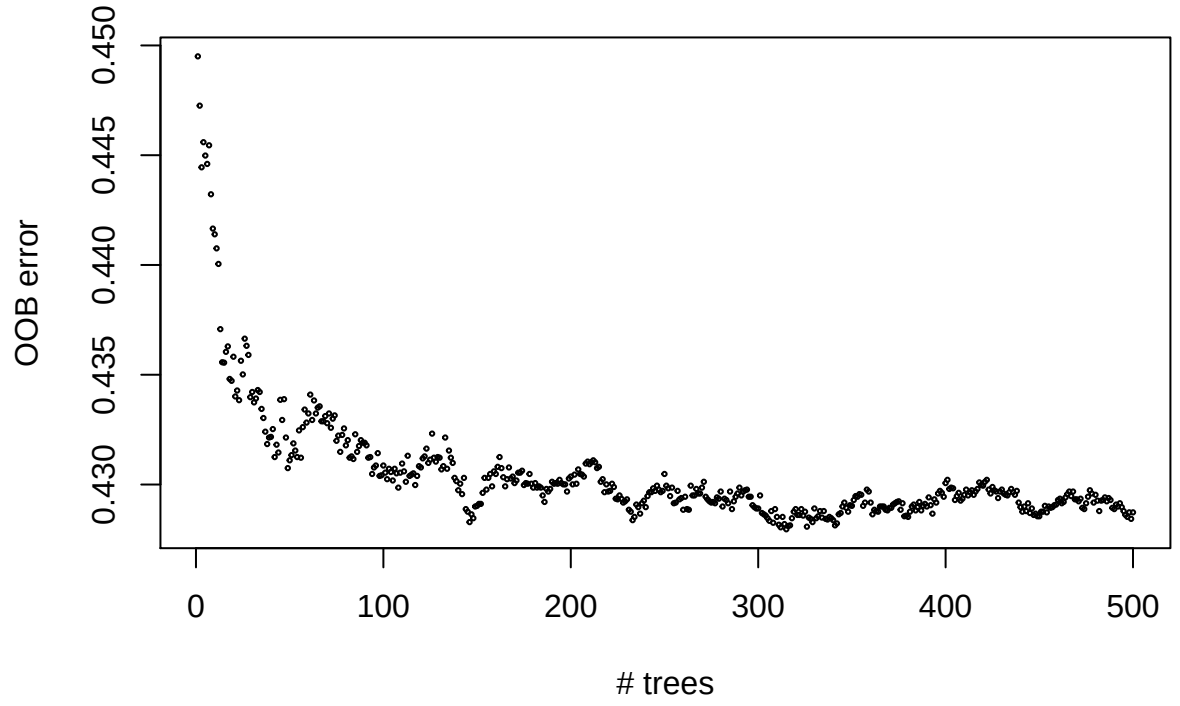
```
rffit = randomForest(INJURY~.,data=accidents_df_train,
                     mtry=5,
                     ntree=500,
                     nodesize=50,
                     importance=T
                     )

varImpPlot(rffit)
```



Out-of-bag error plot

```
plot(rffit$err.rate[,"OOB"], xlab="# trees", ylab="OOB error", cex=0.3)
```



We make predictions as usual.

```
rf_test_predictions = predict(rffit, accidents_df_test)
(1 - mean(rf_test_predictions == accidents_df_test$INJURY))
```

```
## [1] 0.433
```

```
tb_rf = table(predictions = rf_test_predictions,
              actual = accidents_df_test$INJURY)
rownames(tb_rf) = c("predict_NO_INJURY", "predict_INJURY")
print(tb_rf)
```

```
##                    actual
## predictions           0    1
##   predict_NO_INJURY 2315 1810
##   predict_INJURY    1842 2470
```

7 Boosting

We build a boosting model. The following parameters are needed:

- shrinkage parameter λ
- total number of trees
- how big trees to build

IMPORTANT: The gbm package requires us to use numeric value for Y!

```
# we need to make INJURY a numberic variable with values equal to 0 and 1
accidents_df_train$INJURY = as.numeric(accidents_df_train$INJURY)-1
accidents_df_test$INJURY = as.numeric(accidents_df_test$INJURY)-1

boostfit = gbm(INJURY~.,data=accidents_df_train,
               distribution='bernoulli',
               interaction.depth=4,
               n.trees=500,
               shrinkage=.2)
```

Variable importance plot

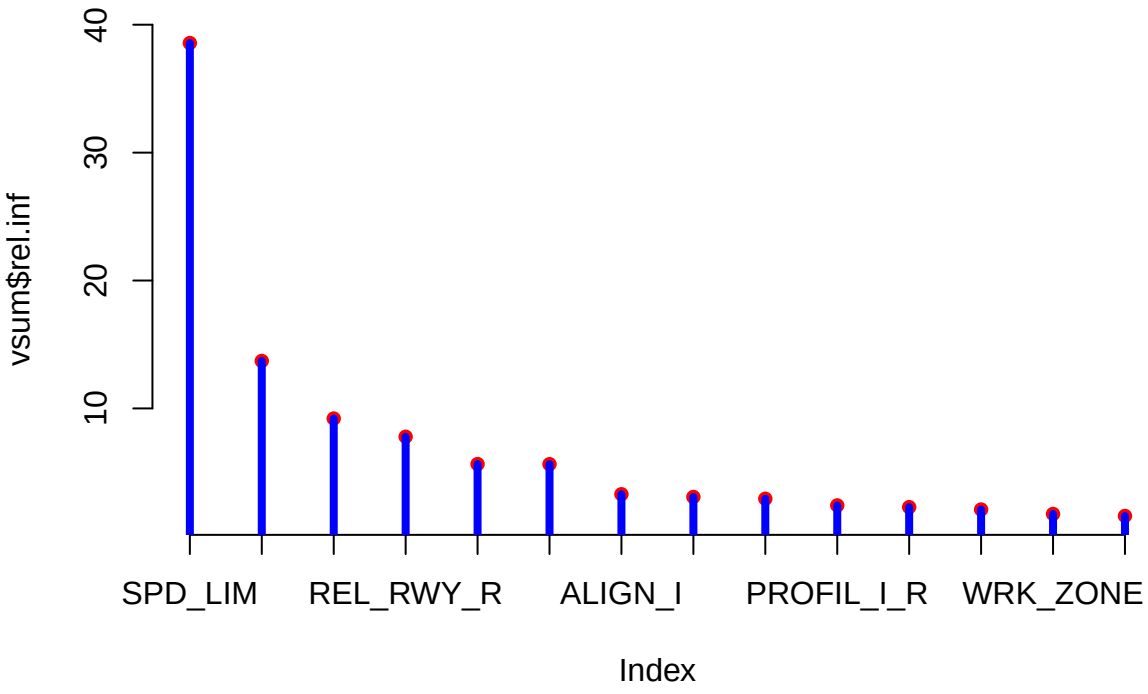
```
p=ncol(accidents_df_train)-1
vsum=summary(boostfit, plotit=F) #this will have the variable importance info

#write variable importance table
print(vsum)
```

```
##                    var rel.inf
## SPD_LIM         SPD_LIM   38.57
## MANCOL_I_R   MANCOL_I_R   13.71
## SUR_COND       SUR_COND    9.21
## REL_RWY_R     REL_RWY_R    7.79
## LGTCON_I_R   LGTCON_I_R    5.65
## TRAF_WAY       TRAF_WAY    5.64
## ALIGN_I         ALIGN_I    3.29
## WKDY_I_R       WKDY_I_R    3.08
## RELJCT_I_R   RELJCT_I_R    2.94
## PROFIL_I_R   PROFIL_I_R    2.42
## WEATHER_R     WEATHER_R    2.28
```

```
## HOUR_I_R     HOUR_I_R    2.09
## WRK_ZONE     WRK_ZONE    1.75
## INT_HWY       INT_HWY    1.59
```

```
#plot variable importance
#the package does this automatically, but I did not like the plot
plot(vsum$rel.inf,axes=F,pch=16,col='red')
axis(1,labels=vsum$var,at=1:p)
axis(2)
for(i in 1:p) lines(c(i,i),c(0,vsum$rel.inf[i]),lwd=4,col='blue')
```



Boosting model predicts probabilities. It is important to specify that you want `type = "response"`

```
b_test_predictions = predict(boostfit, accidents_df_test, n.trees = 500, type = "response")
```

We tranform the probabilites using a naive threshold of 0.5.

```
class0_ind =  b_test_predictions < 0.5
class1_ind =  b_test_predictions >= 0.5
b_test_predictions[class0_ind] = 0
b_test_predictions[class1_ind] = 1
(1 - mean(b_test_predictions == accidents_df_test$INJURY))
```

```
## [1] 0.431
```

```
tb_rf = table(predictions = b_test_predictions,
                  actual = accidents_df_test$INJURY)
rownames(tb_rf) = c("predict_NO_INJURY", "predict_INJURY")
print(tb_rf)
```

```
##                    actual
## predictions           0    1
##   predict_NO_INJURY 2428 1906
##   predict_INJURY    1729 2374
```

7.1 Try using most important variables only

Refit the model

```
accidents_df_train_vs = accidents_df_train

keeps = c("SPD_LIM", "MANCOL_I_R", "REL_RWY_R", "SUR_COND", "INJURY")
accidents_df_train_vs = accidents_df_train_vs[, (names(accidents_df) %in% keeps)]

boostfit = gbm(INJURY~.,data=accidents_df_train_vs,
               distribution='bernoulli',
               interaction.depth=4,
               n.trees=500,
               shrinkage=.2)
```

Predict again

```
b_test_predictions = predict(boostfit, accidents_df_test, n.trees = 500, type = "response")

class0_ind =  b_test_predictions < 0.5
class1_ind =  b_test_predictions >= 0.5
b_test_predictions[class0_ind] = 0
b_test_predictions[class1_ind] = 1
(1 - mean(b_test_predictions == accidents_df_test$INJURY))
```

```
## [1] 0.438
```

```
tb_rf = table(predictions = b_test_predictions,
                  actual = accidents_df_test$INJURY)
rownames(tb_rf) = c("predict_NO_INJURY", "predict_INJURY")
print(tb_rf)
```

```
##                    actual
## predictions           0    1
##   predict_NO_INJURY 2066 1608
##   predict_INJURY    2091 2672
```