project

2023-01-09

```
library(plyr)
library(tidyr)
detach(package:plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(lubridate)
```

```
## Loading required package: timechange

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(forcats)
library(stringr)
library(readr)
library(boot)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice

##
## Attaching package: 'lattice'

## The following object is masked from 'package:boot':
##
##     melanoma
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
data_all <- read_csv('Crimes_2019.csv',show_col_types = FALSE)



data <- data_all %>% select(-c(ID, `Case Number`, Block, IUCR, Description, Beat, Ward, `Community Area

data$Date <- parse_datetime(data$Date, format = "%m/%d/%Y %I:%M:%S %p")
```

```
othering <- function(factor_vec, threshold){
  level <- levels(factor_vec)
  tab <- tabulate(factor_vec)
  other.levels <- level[ tab < threshold]
  factor_vec <- fct_collapse(factor_vec, "Other" = other.levels)
  perc <- length(factor_vec[factor_vec == 'Other'])*100/length(factor_vec)
  print(perc)
  return(factor_vec)

}
```

```
crimes <- drop_na(data_all)

crimes$Date <- parse_datetime(crimes$Date, format = "%m/%d/%Y %I:%M:%S %p")

crimes <- crimes %>% select(-c(ID, `Case Number`, Block, IUCR, Description, Beat, Ward, `Community Area

crimes[c('Primary Type', 'District', 'Arrest', 'Domestic')] <- lapply(crimes[c('Primary Type', 'District

time.data <- crimes %>% mutate(Hour = as.factor(hour(crimes$Date))) %>% group_by(Hour) %>% count(Arrest

crimes <- crimes %>% mutate(day = yday(crimes$Date), wkday = wday(crimes$Date), Hour = as.numeric(hour(

crimes[c('Primary Type', 'District', 'Arrest', 'Domestic','wkday')] <- lapply(crimes[c('Primary Type',

crimes$District[crimes$District == '031'] <- NA

crimes<- drop_na(crimes)

crimes<- droplevels(crimes)

levels(crimes$`Primary Type`)[levels(crimes$`Primary Type`)=="OTHER OFFENSE"] <- "Other"

crimes$`Primary Type` <- othering(crimes$`Primary Type`, 500)
```

```
## [1] 7.104674
```

```
crimes<- droplevels(crimes)
```

For some EDA on the importance of including time data in our model for classifying likelihood of arrest, we plot the probability of arrest for each hour (using 2019 data), we see that the proportion of arrests are inconsistent depending on time - this could be due to types of crimes expected at each time. This makes it seem significant to at least include in the model.BEEP

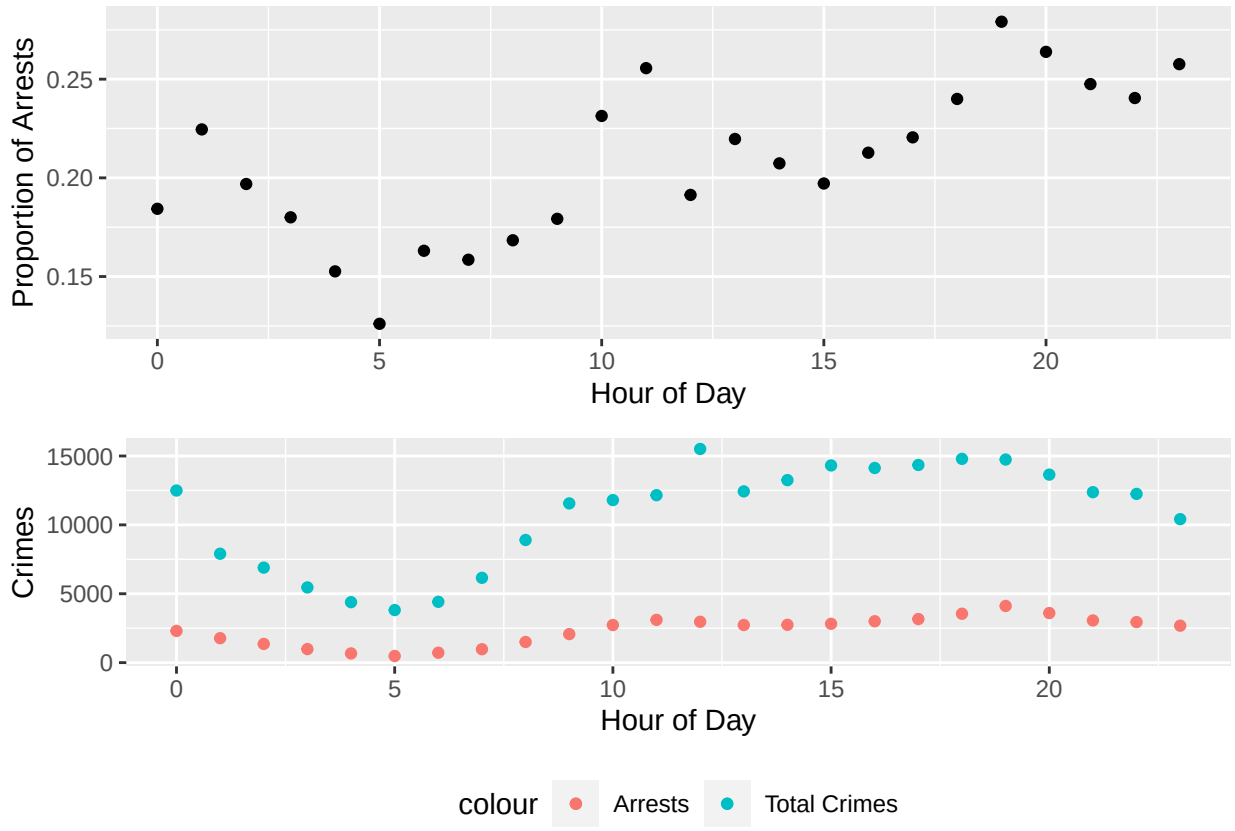
```
#calculate proportion of crimes resulted in arrest in each hour
proparr <- c()
arr <- c()
tot <- c()
for (i in 1:(nrow(time.data)/2) ){
  prop <- time.data$n[2*i]/(time.data$n[2*i-1] + time.data$n[2*i])
  proparr <- c(proparr,prop)
  arr <-c(arr, time.data$n[2*i])
  tot <- c(tot,time.data$n[2*i-1] + time.data$n[2*i])
}
```

```
propdf <- data.frame(cbind(0:23,proparr))
arrdf <- data.frame(cbind(0:23,arr, tot))
colnames(arrdf) <- c('Hour', 'Arr', 'Total')
colnames(propdf) <- c('Hour', 'ArrProp')
```

```
library(ggplot2)
library(cowplot)
```

```
##
## Attaching package: 'cowplot'

## The following object is masked from 'package:lubridate':
##
##     stamp
```

```
plot1 <- ggplot(propdf) + geom_point(aes(x =Hour, y = ArrProp)) + labs(y = 'Proportion of Arrests', x=

plot2 <- ggplot(arrdf) + geom_point(aes(x =Hour, y = Arr, color = 'Arrests' )) + geom_point(aes(x =Hour

plot_grid(plot1, plot2, nrow=2)
```



Here I use the Caret Library to Train a SVM.

```
library(e1071)
library(caret)
#taking sample as SVM scales badly with large data sets

srows <- sample(1:nrow(crimes), 25000, replace = FALSE)
ydata <- crimes$Arrest
ydata <- ydata[srows]
xdata <- crimes %>% select(-Arrest)
xdata <- xdata[srows,]

#Creating SVM model with Sample
mf <- model.frame(ydata ~ . - 1, cbind(ydata,xdata))
mt <- terms(mf)
X <- model.matrix(mt, mf)
Y <- model.response(mf)
```

We will attempt to fit our SVM using 2 different training sets. As SVM models scale badly with data points we have to train our models on samples of our data set (which has approximately $250,000$ data points). There are a few methodologies when taking these samples, we could randomly sample our data, we could also take a representative sample and we could also try having equal amount of samples from each of the classes.

```
crimes %>% count(Arrest)
```

```
## # A tibble: 2 x 2
##   Arrest      n
##   <fct>   <int>
## 1 FALSE  202086
## 2 TRUE    56068
```

We see that only 21.7 of the data has `Arrest = TRUE`. For our first SVM model, we just take a representative sample (of the same size as the previous model). The `createDataPartition` function in `caret` allows us to easily do this.

```
trainIndex <- createDataPartition(crimes$Arrest, p = .05,
                                  list = TRUE)

crimesTest <- crimes[trainIndex$Resample,]

crimesTest %>% count(Arrest)
```

```
## # A tibble: 2 x 2
##   Arrest     n
##   <fct>  <int>
## 1 FALSE  10105
## 2 TRUE    2804
```

Using the `train` function from `caret` we can train our classifier using SVM with a Radial Basis Kernel, I use the package `doParallel` to allow us to do 5 fold cross validation in parallel.

```
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

svm.model <- train(Arrest ~ .,
                   data = crimesTest,
                   trControl = trainControl(method = 'cv',5, allowParallel = TRUE),
                   method = 'svmRadial',
                   family = binomial(),
                   allowParallel = TRUE)

stopCluster(cl)
svm.model$results[c('Accuracy', 'Kappa')]
```

```
##    Accuracy     Kappa
## 1 0.8612596 0.5121629
## 2 0.8621891 0.5137884
## 3 0.8619568 0.5108436
```

As we will be taking a somewhat small sample of our whole data set to train this model on, we can try a method called down sampling, this a method of sampling a data set in which we take all of the data points from our lowest proportion class, and sample an equal amount of the other proportions of the classes.

First we use the `downSample` function from `caret` this takes a sample of our data set, giving us an equal sample of both `Arrest = TRUE` and `Arrest = FALSE`.

```
down_sample <- downSample(x = crimes, y = crimes$Arrest)
down_sample %>% count(Arrest)
```

```
##   Arrest     n
## 1  FALSE 56068
## 2   TRUE 56068
```

We see that the sample involves all the TRUE data points and gives a random sample of our FALSE data points, we can now use the `createDataPartition` function to take a representative sample of `down_sample` (of equal size to our other sample.

```
trainIndexD <- createDataPartition(down_sample$Arrest, p = nrow(crimesTest)/nrow(down_sample),
                                   list = TRUE)

crimesTestD <- down_sample[trainIndexD$Resample,] %>% select(-Class)

crimesTestD %>% count(Arrest)
```

```
##   Arrest    n
## 1  FALSE 6455
## 2   TRUE 6455
```

```
cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

svm.modelD <- train(Arrest ~ .,
                    data = crimesTestD,
                    trControl = trainControl(method = 'cv', 5, allowParallel = TRUE),
                    method = 'svmRadial',
                    family = binomial())

stopCluster(cl)
svm.modelD$results[c('Accuracy', 'Kappa')]
```

```
##    Accuracy     Kappa
## 1 0.7388846 0.4777692
## 2 0.7348567 0.4697134
## 3 0.7347018 0.4694036
```

There is another method of sampling in a similar vein to down sampling, known as up sampling where we sample with replacement the non majority classes until they are of the same length as the majority class.

```
up_sample <- upSample(x = crimes, y = crimes$Arrest)
up_sample %>% count(Arrest)
```

```
##   Arrest      n
## 1  FALSE 202086
## 2   TRUE 202086
```

```
trainIndexU <- createDataPartition(up_sample$Arrest, p = nrow(crimesTest)/nrow(up_sample),
                                   list = TRUE)

crimesTestU <- up_sample[trainIndexU$Resample,] %>% select(-Class)

crimesTestU %>% count(Arrest)
```

```
##   Arrest    n
## 1  FALSE 6455
## 2   TRUE 6455
```

```
cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

svm.modelU <- train(Arrest ~ .,
                    data = crimesTestU,
                    trControl = trainControl(method = 'cv', 5, allowParallel = TRUE),
                    method = 'svmRadial',
                    family = binomial())

stopCluster(cl)
svm.modelU$results[c('Accuracy', 'Kappa')]
```

```
##    Accuracy     Kappa
## 1 0.7221534 0.4443067
## 2 0.7208366 0.4416731
## 3 0.7218435 0.4436871
```

We can similarly train a logistic regression model using 5-fold cross validation.

```
cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

log.model <- train(Arrest ~ .,
                   data = crimes,
                   trControl = trainControl(method = 'cv',5, allowParallel = TRUE),
                   method = "glm",
                   family=binomial())

stopCluster(cl)
log.model$results[c('Accuracy', 'Kappa')]
```

```
##    Accuracy     Kappa
## 1 0.8631786 0.5201491
```

There are many different ways to measure

$$Accuracy = Pr(\hat{Y} = Y)$$

$$Sensitivity = Pr(\hat{Y} = 1|Y = 1) = Pr(\text{True Positive})$$

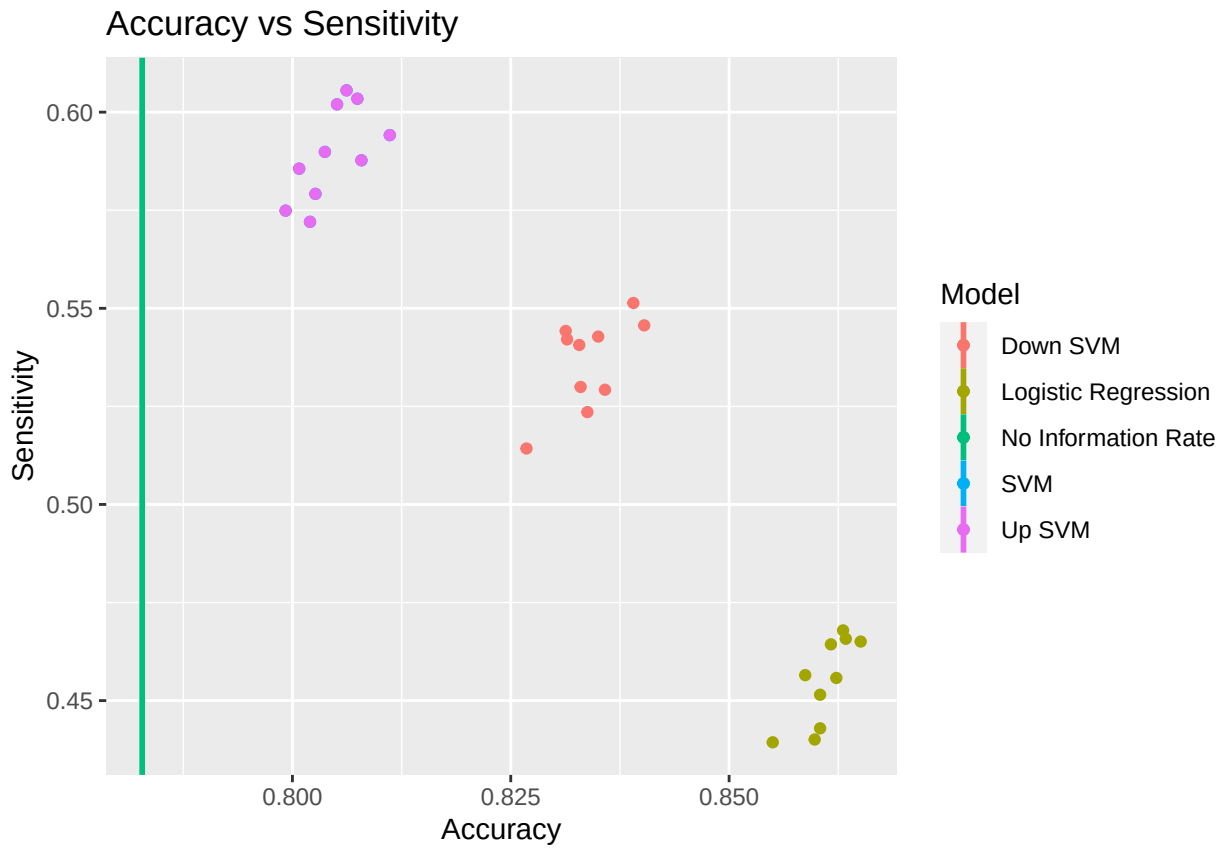
$$Specificity = Pr(\hat{Y} = 0|Y = 0) = Pr(\text{True Negative})$$

An interesting benchmark for the usefulness of the model is the No-Information Rate which is the accuracy if we just guess for each data point is the largest percentage data point. For the crimes dataset that is 0.7828.

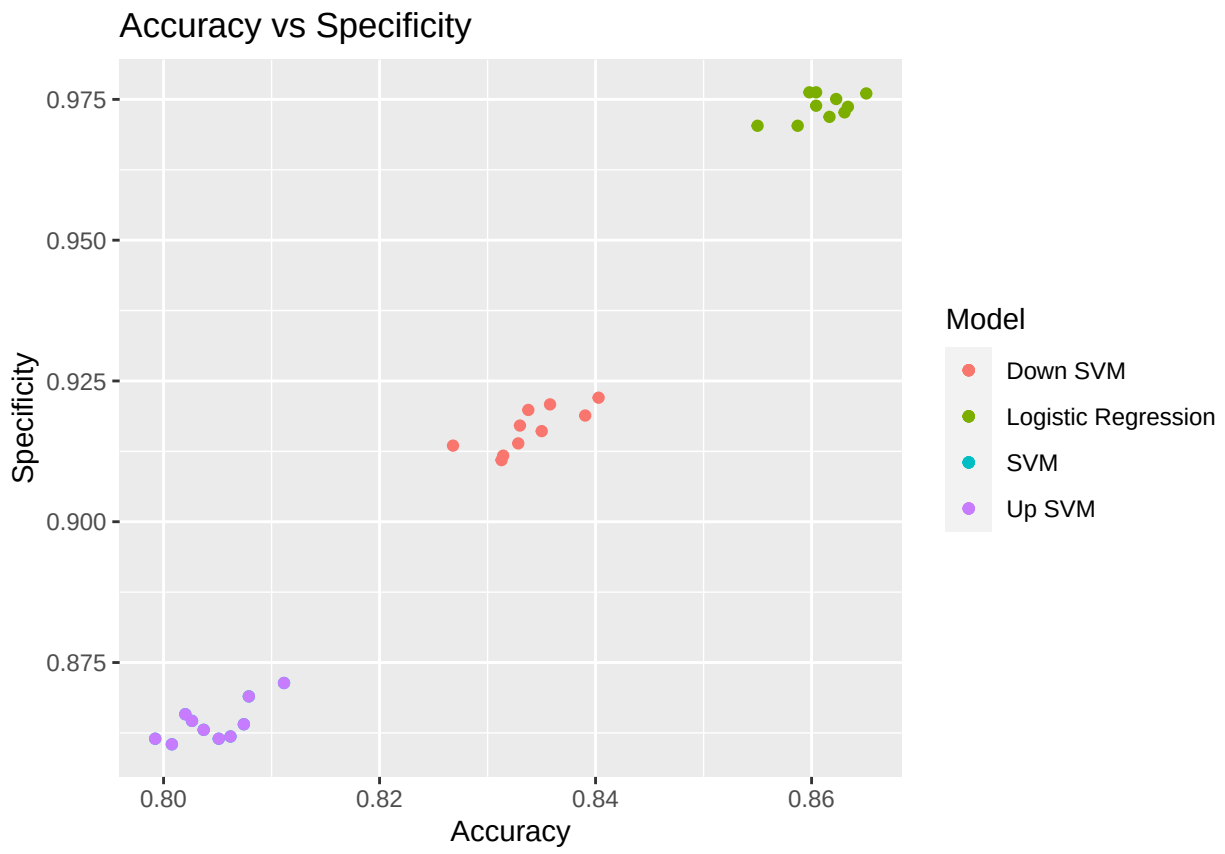
```
blank <- data.frame(ACC=numeric(0),SENS = numeric(0), SPEC = numeric(0), BALACC = numeric(0))
up <- blank
down <- blank
logistic <- blank
repSVM <- blank

for (testrow in 1:10){
  sampIndex <- createDataPartition(crimes$Arrest, p = .025,
                                   list = TRUE)

  sampTest <- crimes[sampIndex$Resample,]

  u <- confusionMatrix(data = predict(svm.modelU, newdata = sampTest), reference = sampTest$Arrest, posi
  d <- confusionMatrix(data = predict(svm.modelD, newdata = sampTest), reference = sampTest$Arrest, posi
  l <- confusionMatrix(data = predict(log.model, newdata = sampTest), reference = sampTest$Arrest, posit
  r <- confusionMatrix(data = predict(svm.modelU, newdata = sampTest), reference = sampTest$Arrest, posi

  up[testrow, ] <- c(u[["overall"]][["Accuracy"]], u[["byClass"]][["Sensitivity"]],
              u[["byClass"]][["Specificity"]], u[["byClass"]][["Balanced Accuracy"]])

  down[testrow, ] <- c(d[["overall"]][["Accuracy"]], d[["byClass"]][["Sensitivity"]],
              d[["byClass"]][["Specificity"]], d[["byClass"]][["Balanced Accuracy"]])

  logistic[testrow, ] <- c(l[["overall"]][["Accuracy"]], l[["byClass"]][["Sensitivity"]],
              l[["byClass"]][["Specificity"]], l[["byClass"]][["Balanced Accuracy"]])

  repSVM[testrow, ] <- c(r[["overall"]][["Accuracy"]], r[["byClass"]][["Sensitivity"]],
              r[["byClass"]][["Specificity"]], r[["byClass"]][["Balanced Accuracy"]])
}
```

```
ggplot(mapping = aes(ACC,SENS, color = Model)) +
  geom_point(data = repSVM, mapping = aes(x = ACC, y = SENS, color = 'SVM')) +
  geom_point(data = down, mapping = aes(x = ACC, y = SENS, color = 'Down SVM')) +
  geom_point(data = logistic, mapping = aes(x = ACC, y = SENS, color = 'Logistic Regression')) +
  geom_point(data = up, mapping = aes(x = ACC, y = SENS, color = 'Up SVM')) +
  labs(title = 'Accuracy vs Sensitivity', x = 'Accuracy', y = 'Sensitivity') +
  geom_vline(mapping = aes(xintercept = 0.7828, colour = 'No Information Rate'), size = 1)
```

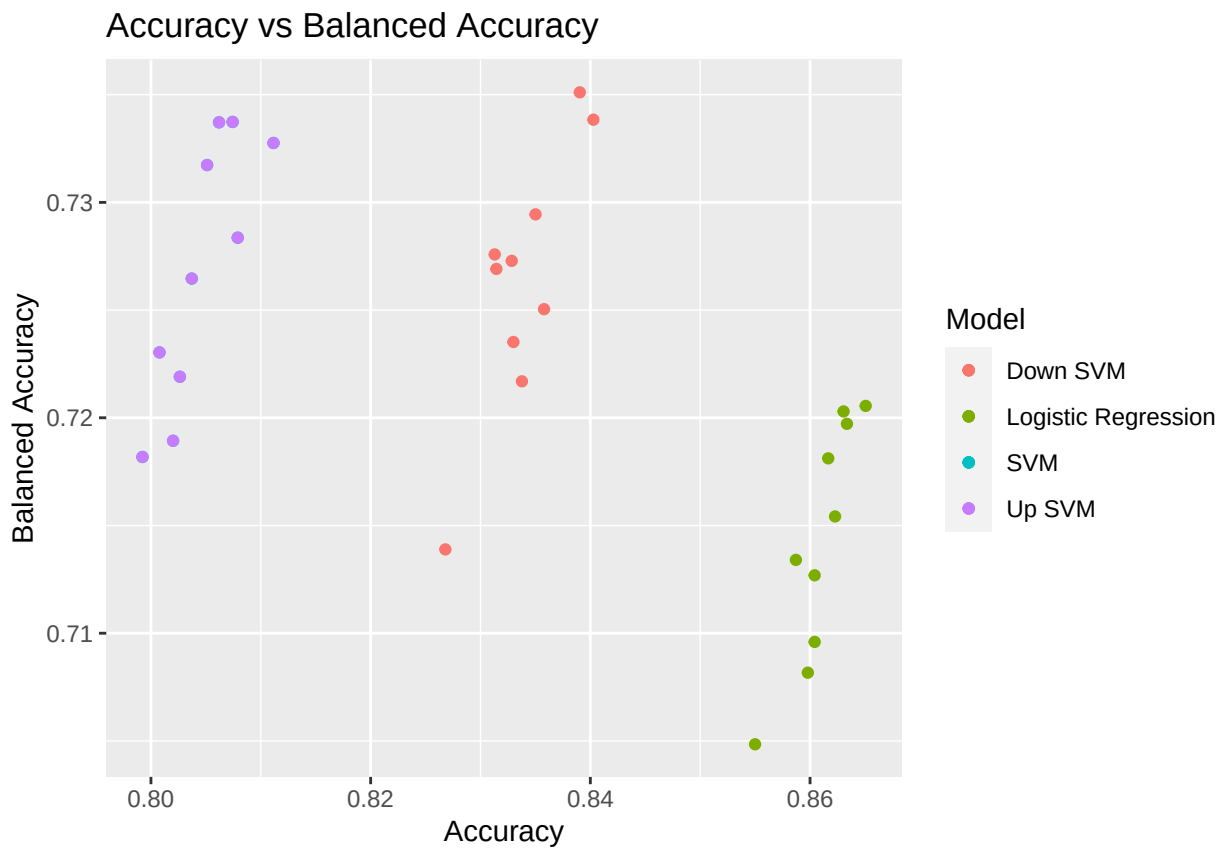
```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
```




```
ggplot(mapping = aes(ACC,SPEC, color = Model)) +
  geom_point(data = repSVM, mapping = aes(x = ACC, y = SPEC, color = 'SVM')) +
  geom_point(data = down, mapping = aes(x = ACC, y = SPEC, color = 'Down SVM')) +
  geom_point(data = logistic, mapping = aes(x = ACC, y = SPEC, color = 'Logistic Regression')) +
  geom_point(data = up, mapping = aes(x = ACC, y = SPEC, color = 'Up SVM')) +
  labs(title = 'Accuracy vs Specificity', x = 'Accuracy', y = 'Specificity')
```




```
ggplot(mapping = aes(ACC,BALACC, color = Model)) +
  geom_point(data = repSVM, mapping = aes(x = ACC, y = BALACC, color = 'SVM')) +
  geom_point(data = down, mapping = aes(x = ACC, y = BALACC, color = 'Down SVM')) +
  geom_point(data = logistic, mapping = aes(x = ACC, y = BALACC, color = 'Logistic Regression')) +
  geom_point(data = up, mapping = aes(x = ACC, y = BALACC, color = 'Up SVM')) +
  labs(title = 'Accuracy vs Balanced Accuracy', x = 'Accuracy', y = 'Balanced Accuracy')
```



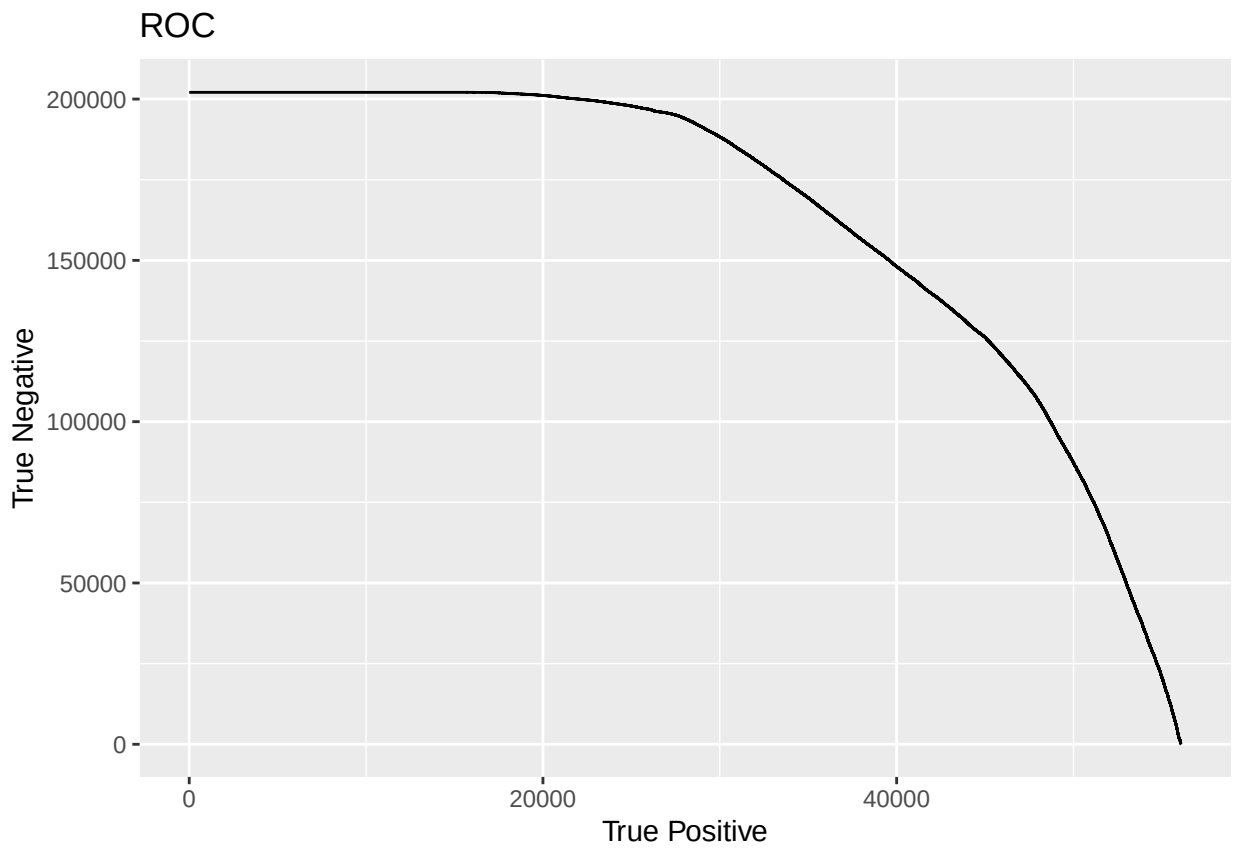

```
library(ROCit)
```

```
##
## Attaching package: 'ROCit'
```

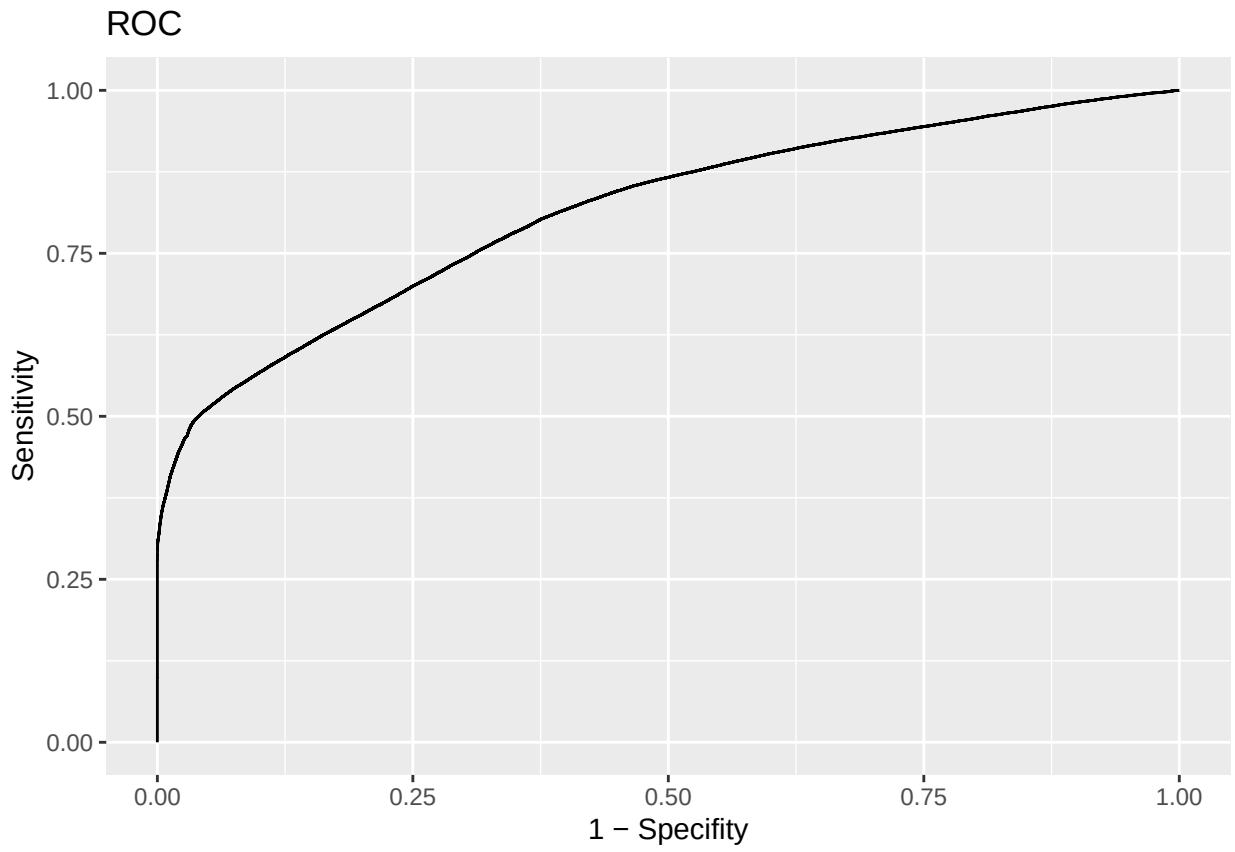
```
## The following object is masked from 'package:boot':
##
##     logit
```

```
roc <- rocit(score = log.model[["finalModel"]][["fitted.values"]], class = log.model[["finalModel"]][["y
measure <- measureit(roc,measure = c("ACC", "SENS", "SPEC"))
```

```
ggplot() + geom_line(aes(x = measure$TP, y = measure$TN)) + labs(title = 'ROC',x = 'True Positive', y =
```




```
ggplot() + geom_line(aes(x = 1 - measure$SPEC, y = measure$SENS)) + labs(title = 'ROC',x = '1 - Specifi
```




```
ggplotROC <- function(fitval, y){
  roc <- rocit(score = fitval, class = y)
  meas <- measureit(roc,measure = c("ACC", "SENS", "SPEC"))
  return(ggplot() + geom_line(aes(x = 1 - meas$SPEC, y = measure$SENS)) + labs(title = 'ROC',x = '1 - S
}

AUC <- function(fitval,y){
  roc_mod <- rocit(score = fitval , class = y)
  return(roc_mod[['AUC']])
}
```

Using ROC in this case has some theoretical advantages, it does tend to prefer models that obtain True positives- this wouldn't be frowned upon as an overestimation allows redundancy preparation for the services needed to deal with arrests.