project

2023-01-09

```
rm(list = ls())
```

```
library(plyr)
library(tidyr)
detach(package:plyr)
library(dplyr)
library(lubridate)
library(forcats)
library(stringr)
library(readr)
library(boot)
library(caret)
library(kernlab)

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")
```

Binary Classification

In this section we attempt to fit a binary classification model, to predict given certain information whether the crime would lead to an arrest or not. Choosing only to use 2019 data, due to scale of the data set (leaving us with roughly 250,000 observations), as data recorded after is affected strongly by COVID.

The features we choose to include in this model are:

$$\mathbf{x} = (\texttt{Domestic, Primary Type, Day, Time, Week Day, District})$$

We choose `Primary Type` over `FBI Code` to determine the type of crime as during our EDA we found inconsistencies with the classification of `FBI Code`. With `Primary Type`, we merge uncommon levels of our factor variable into the level `OTHER", in which we also include the pre-existing crime type`OTHER OFFENSE".

For temporal data we include `Week Day` as a factor variable, encoding date using `Day` (1-365) and `Time` as a numeric character between 0-1.

In order to include spatial information, we consider the `District` variable. We observe that all districts experience reasonable numbers of crimes to be incorporated into analysis, with the exception of District 031.

While including more information might improve our models, during our EDA we found the inclusion of highly correlated variables to improve the model insignificantly while drastically increasing computational time.

```
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)

}
```

```
#Feature Manipulation and Selection
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) %>% dplyr::count

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`, 1000)
```

```
## [1] 8.177677
```

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

Here we plot the proportion of crimes that resulted in arrest for depending on the hour of the day. We can see that the likelihood of an Arrest Resulting from a crime might have dependence on the hour the crime was committed this could be due to the nature of the crimes). However, this seems to indicates that it would be worthwhile to include time as a predictor in our binary classification model.

```
#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)
```

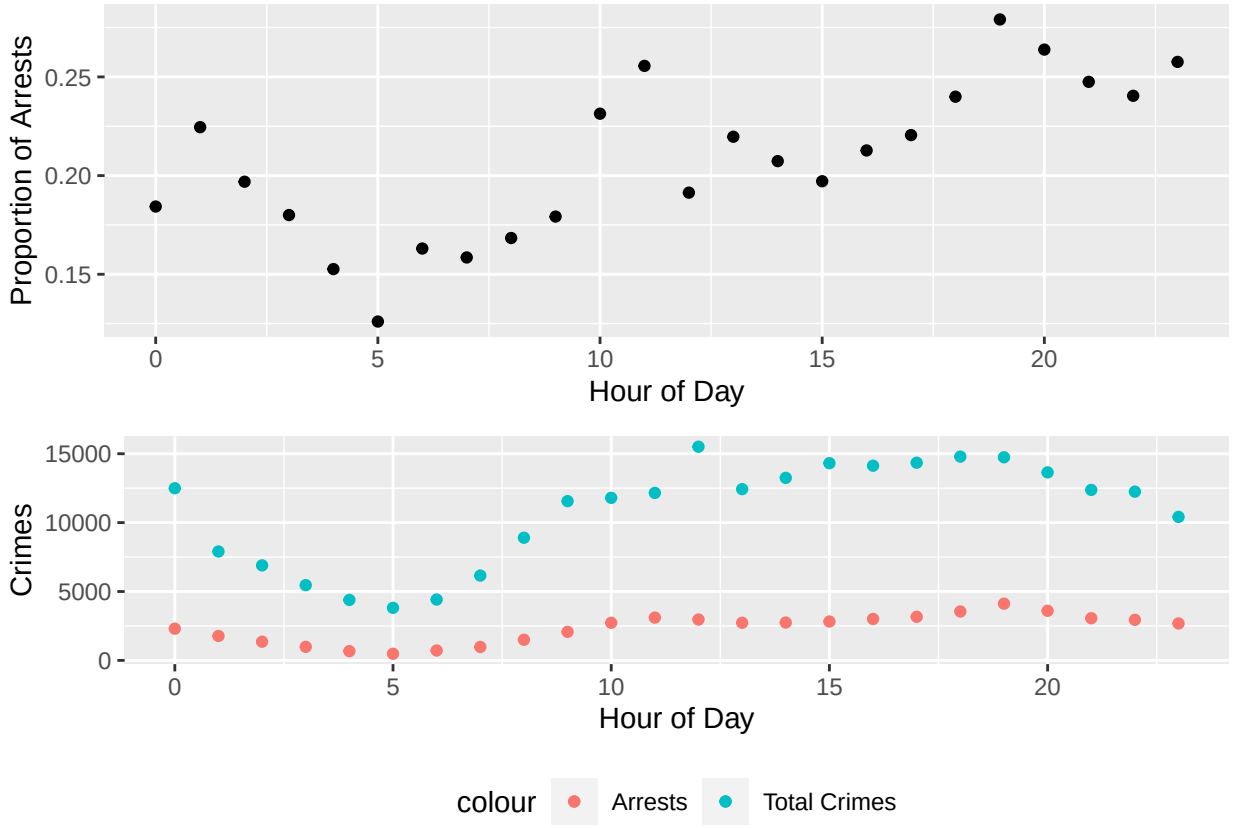
```
  arr <-c(arr, time.data$n[2*i])
  tot <- c(tot,time.data$n[2*i-1] + time.data$n[2*i])
}
```

```
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)
```



Binary Classification

In this section we attempt to fit a binary classification model, to predict given certain information whether the crime would lead to an arrest or not. Choosing only to use 2019 data, due to scale of the data set (leaving us with roughly 250,000 observations), as data recorded after is affected strongly by COVID.

The features we choose to include in this model are:

$$\mathbf{x} = (\texttt{Domestic, Primary Type, Day, Time, Week Day, District})$$

We choose `Primary Type` over `FBI Code` to determine the type of crime as during our EDA we found inconsistencies with the classification of `FBI Code`. With `Primary Type`, we merge uncommon levels of our factor variable into the level `OTHER", in which we also include the pre-existing crime type`OTHER OFFENSE".

For temporal data we include `Week Day` as a factor variable, encoding date using `Day` (1-365) and `Time` as a numeric character between 0-1.

In order to include spatial information, we consider the `District` variable. We observe that all districts experience reasonable numbers of crimes to be incorporated into analysis, with the exception of District 031.

While including more information might improve our models, during our EDA we found the inclusion of highly correlated variables to improve the model insignificantly while drastically increasing computational time.

Logistic Regression

```
library(caret)
library(doParallel)
```

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

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

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

```
set.seed(30700) # Setting seed for reproducibility
```

Using the package `doParallel` and `caret` we can easily fit classification models. We define a control method which allows our function to do 5-fold Cross Validation, using `doParallel` to do the cross validation in parallel to decrease computational time. Setting seeds for our cross-validation function to ensure reproducibility. The function will return the model with the highest accuracy, however we can see the accuracy for the other models in the cross validation process.

```
#Create Seeds for the Cross Validation Function
seeds <- vector(mode = "list", length = 6)
for(i in 1:6) seeds[[i]] <- sample.int(n=1000, 3)

#Define Cross Validation Function
control <- trainControl(method = 'cv', number = 5, allowParallel = TRUE, seeds=seeds)

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

```
registerDoParallel(cl)

#Fitting Logistic Regression Classifier
log.model <- train(Arrest ~ .,
                   data = crimes,
                   trControl = control,
                   method = "glm",
                   family=binomial())

stopCluster(cl)
```

Support Vector Machines

In this section we fit Support Vector Machines with the Radial Basis Kernel (due to the unclear relationship between our predictors and class).

As SVM models scale badly with data points we have to train our models on samples of our data set (which has greater than $250,000$ data points). We will be fitting 3 SVM models, each using a different sampling technique.

- The first method is to take a proportional sample, which attempts to generate a sample where the proportion of each classes emulates the original data set.
- The next is known as 'Down Sampling' where we randomly sample the majority class till we have the same number of observations as the minority class.
- Similarly we will use the method known as 'Up Sampling', randomly samples (with replacement) the minority class until we have the same number of observations as the majority class.

It is clear than in our data there is a significant imbalance in the class proportions.We see that only 21.7 of the data has `Arrest = TRUE`.

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

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

For our first SVM model, we will take a representative sample of roughly 13,000 data points. The `createDataPartition` function in `caret` allows us to easily do this.

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

crimesTest <- crimes[trainIndex$Resample,]

crimesTest %>% dplyr::count(Arrest)
```

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

Just like with the Logistic Regression Classifier we use the `train` function from `caret` using our previously defined train function - which has the exact same implementation for the SVMs we fit. The `svmRadial` method uses `kernlab::sigest` to determine a good predictor for σ. It will then test multiple values for Cost and will choose the one which maximizes accuracy. The `train` function will work in this manner for the other 2 models we train as well.

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

registerDoParallel(cl)

#Fitting SVM Model with a Proportional Sample
svm.model <- train(Arrest ~ .,
                   data = crimesTest,
                   trControl = control,
                   method = 'svmRadial',
                   family = binomial(),
                   allowParallel = TRUE)

stopCluster(cl)
```

As we will be taking a somewhat small sample of our whole data set to train this model on, we will now 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)

#Training a SVM using down sampling
svm.modelD <- train(Arrest ~ .,
                    data = crimesTestD,
                    trControl = control,
                    method = 'svmRadial',
                    family = binomial())

stopCluster(cl)
```

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 in a similar manner with down sampling we will train a model.

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

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

```
set.seed(1)
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)

#Training a SVM using up sampling
svm.modelU <- train(Arrest ~ .,
                    data = crimesTestU,
                    trControl = control,
                    method = 'svmRadial',
                    family = binomial())

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

```
##    Accuracy     Kappa
## 1 0.7316809 0.4633617
## 2 0.7316809 0.4633617
## 3 0.7302091 0.4604183
```

Model Performance

To test the performance of our Models, we will simulate 10 proportional (in respect to number of arrests), testing data sets. For each of these testing sets we will some metrics. The metrics below will be ones measured for each of models.

$$Accuracy = Pr(\hat{Y} = Y)$$
$$Sensitivity = Pr(\hat{Y} = 1 | Y = 1) = Pr(\text{True Positive})$$
$$Specificity = Pr(\hat{Y} = 0 | Y = 0) = Pr(\text{True Negative})$$
$$\text{Balanced Accuracy} = \frac{Sensitivity + Specificity}{2}$$

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 proportion class. For the crimes data set that is 0.7828, however this is not some foolproof benchmark, for example if our goal is to predict to lowest proportion class consistently.

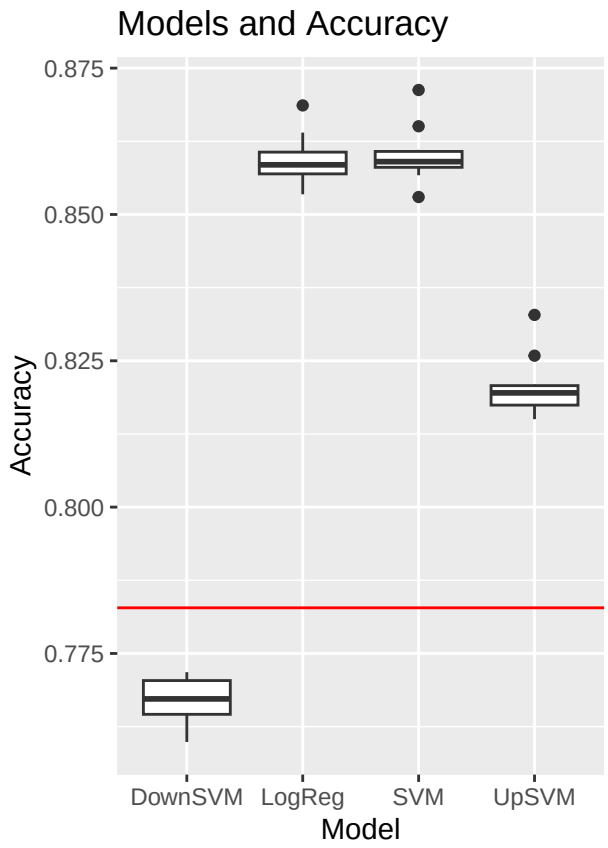
```
blank <- data.frame(Model = as.character(), ACC=numeric(0),SENS = numeric(0), SPEC = numeric(0), BALACC
up <- blank
down <- blank
logistic <- blank
repSVM <- blank

#Creating data frames of our metrics
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$Arrest, positive = 'TRUE')
  d <- confusionMatrix(data = predict(svm.modelD, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')
  l <- confusionMatrix(data = predict(log.model, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')
  r <- confusionMatrix(data = predict(svm.model, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')

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

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

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

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

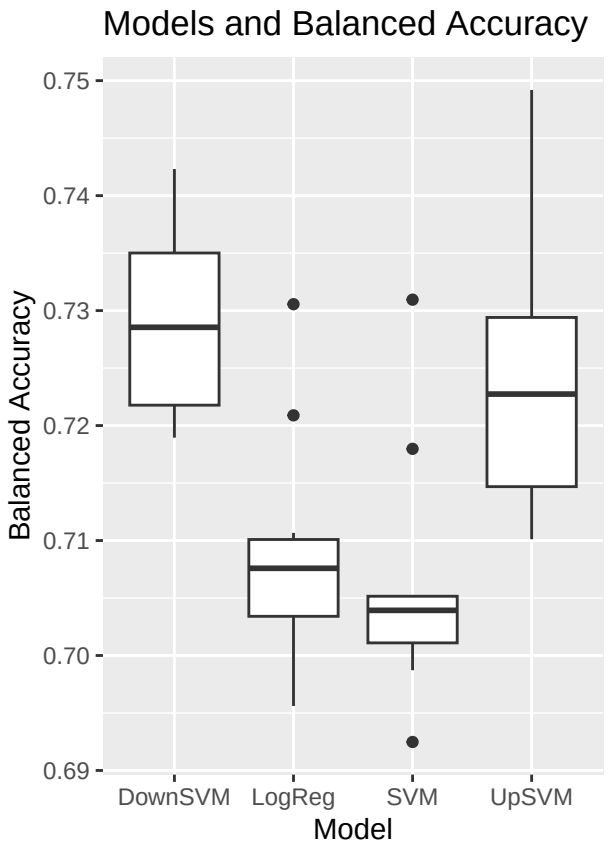
Below we have a table of the mean values of the metrics previously defined for each model.

```r
FullBinTest <- rbind(up,down,logistic,repSVM) %>%  mutate(Model = as.factor(Model),
                                                          ACC = as.numeric(ACC),
                                                          SENS = as.numeric(SENS),
                                                          SPEC = as.numeric(SPEC),
                                                          BALACC = as.numeric(BALACC))

FullBinTest %>% group_by(Model) %>% summarise(across(everything(), mean))
```

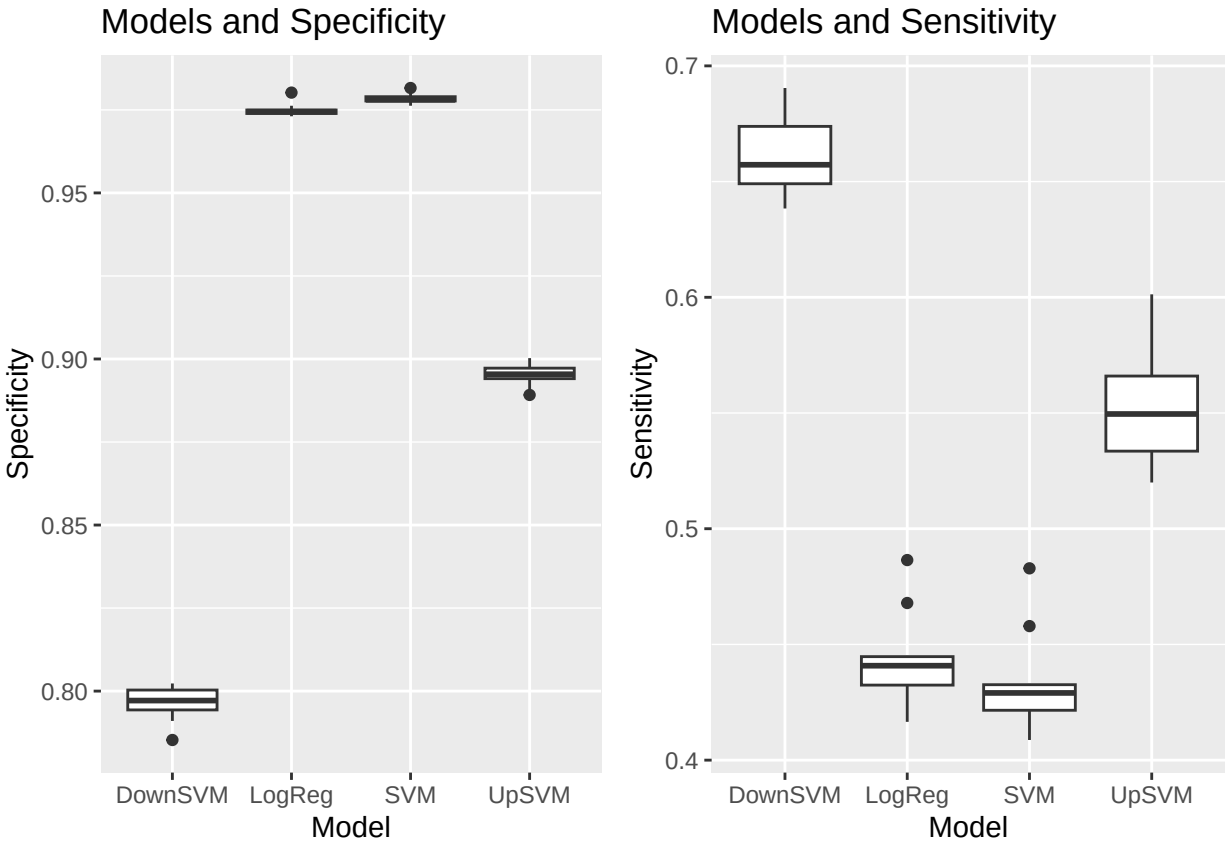
```
## # A tibble: 4 x 5
##   Model     ACC  SENS  SPEC BALACC
##   <fct>   <dbl> <dbl> <dbl>  <dbl>
## 1 DownSVM 0.767 0.662 0.796  0.729
## 2 LogReg  0.860 0.443 0.975  0.709
## 3 SVM     0.860 0.434 0.978  0.706
## 4 UpSVM   0.821 0.552 0.895  0.724
```

```r
ACCPlot <- ggplot(FullBinTest, aes(x = Model, y = ACC)) + geom_boxplot() +
  geom_hline(mapping = aes(yintercept = 0.7828), color = 'red') +
  labs(title = 'Models and Accuracy', y = 'Accuracy')
SENSPlot <- ggplot(FullBinTest, aes(x = Model, y = SENS)) + geom_boxplot() +
  labs(title = 'Models and Sensitivity', y = 'Sensitivity')
SPECPlot <- ggplot(FullBinTest, aes(x = Model, y = SPEC)) + geom_boxplot() +
  labs(title = 'Models and Specificity', y = 'Specificity')
BALACCPlot <- ggplot(FullBinTest, aes(x = Model, y = BALACC)) + geom_boxplot() +
  labs(title = 'Models and Balanced Accuracy', y = 'Balanced Accuracy')

plot_grid(ACCPlot,BALACCPlot , nrow=1, greedy = TRUE)
```






```
plot_grid(SPECPlot, SENSPlot, nrow=1, greedy = TRUE)
```



We can see that both the proportional SVM Model and Logistic Model performed very similarly, both with high accuracy and specificity, this is to be expected, as both models were trained and tested with far more `FALSE` data points. We can see that with the trade-off being in sensitivity, both performing poorly at distinguishing positive data points.

Another important point, is that we might have expected the SVM model to perform better than the Logistic Regression Model, due to it's abilities to take into account non-linear relationships; however, we can see they performed very similarly in nearly all aspects, this is probably due to SVM being trained on only 5% of the 2019 data set while our Logistic Model was trained on the full 2019 data set.

As expected up and down sampling had a positive affect on the sensitivity of the model due to the training data involving a larger proportion of `TRUE` values. If we look at the down sampled model, we see that the high sensitivity gives a large trade off with specificity and accuracy (pushing it below the no information rate), this model is far more likely than the other models to give a false positive. This could be due to the down sample involving sampling the `FALSE` data points possibly giving a less informative sample.

Compared to down sampling, up sampling did not increase the sensitivity as much, but instead found a more balanced trade off with specificity and accuracy - this is interesting given that both models were trained on the same amount of `TRUE` and `FALSE` data points, further analysis would be needed to determine whether this was due to an intrinsic property of the sampling technique or just due to randomness.

The better balanced accuracy of the up and down sampled models, shows that accuracy of the Logistic and proportional models comes heavily from it's lack of sensitivity combined with the imbalanced class proportions, and that depending on what the goal of our models it might be preferable to pick a less overall accurate model.