# Choose Your Own Capstone Project

*Md Ishtiaque Hossain*

*9/29/2019*

## Executive Summary

The dataset used for "Choose Your Own" segment of the capstone project is procured from Kaggle, from this link: https://www.kaggle.com/loveall/clicks-conversion-tracking/download

This particular dataset is from an anonymous organisation, and corresponds to their facebook ad campaigns. It is a fairly small dataset that can be easily used on a standard laptop, without having to resort to cloud computing resouces. There are 1143 observations in total, and 11 dimensions.

My reason for picking this dataset is because of my interest in digital marketing, and also because this is a rather obscure dataset that has not been used any high profile Kaggle competitions, or other courses in this program.

The objective of this project is to use facebook campaign data to predict which ads are going to generate sales, so that sales and marketing resources can be directed towards high potential leads, rather than being wasted on leads that are unlikely to generate sales conversions.

For this particular project, let's sensitivity of the model is more important than overall accuracy. Which is often the case for high-involvment products like real estates. In such cases, the cost of missing out on a high potential lead is much higher than pursuing a lead that may not generate sales.

#### Loading Necessary Packages

#### Data Loading and Setup

After downloading the dataset from Kaggle, the .zip file is uploaded to Google Cloud Storage so that it can be accessed from anywhere. The permission for this storage bucket is set to "public".

```
data_dir <- gs_data_dir_local("gs://birds-nest-bb")
KAG_conversion_data <- read_csv(file.path(data_dir, "KAG_conversion_data.csv"))
```

There are 1143 observations and 11 dimensions in this set. Here are the descriptions of each dimension, as seen on Kaggle:

1.) ad_id: an unique ID for each ad.

2.) xyz_campaign_id: an ID associated with each ad campaign of XYZ company.

3.) fb_campaign_id: an ID associated with how Facebook tracks each campaign.

4.) age: age of the person to whom the ad is shown.

5.) gender: gender of the person to whim the add is shown

6.) interest: a code specifying the category to which the person's interest belongs (interests are as mentioned in the person's Facebook public profile).

7.) Impressions: the number of times the ad was shown.

8.) Clicks: number of clicks on for that ad.

9.) Spent: Amount paid by company xyz to Facebook, to show that ad.

10.) Total conversion: Total number of people who enquired about the product after seeing the ad.

11.) Approved conversion: Total number of people who bought the product after seeing the ad.

Once the data is loaded, to facilitate classification via machine learning, we add one more dimension called 'Sales'.

```
KAG_Bin <- KAG_conversion_data %>% mutate(Sales=ifelse(Approved_Conversion>0,"Yes","No"))
```

Now we move on to creating test and train sets:

```
set.seed(1989)
train_index <- createDataPartition(KAG_Bin$Sales,times=1,p=.7,list=FALSE)
train_part <- as.tbl(KAG_Bin) %>% dplyr::slice(train_index)
temp <- as.tbl(KAG_Bin) %>% dplyr::slice(-train_index)

# Making sure that the interests appearing in the test set are also in the train set
test_part <- temp %>%
  semi_join(train_part, by = "interest")
```

## 2. Methods and Analysis

Intially we started with a majority voting method by ensembling four classification models using adaboost, xgbtree, rf and Rborist algorithms. But then we observed that, apart from XGBoost, all other models were performing poorly (close to 50% or lower accuracy). We are not including that code chunk in this reproducible report, because it takes more than 30 minutes to run it on a standard laptop. Since XGBoost model showed above 60% accuracy on average without tuning, we decided to focus on a single XBoost model and tuning the hyperparameters for optimum performance.

**Results from XGBoost Default Tune**

```
fits_0 <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+Impressions
                +Clicks+Spent+Total_Conversion, method = "xgbTree", data = train_part)


# Creating Predictions

fits_0_predicts <- predict(fits_0,test_part)

confusionMatrix(as.factor(fits_0_predicts),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  129  71
##        Yes  38 104
##
##                Accuracy : 0.6813
##                  95% CI : (0.629, 0.7304)
##      No Information Rate : 0.5117
##      P-Value [Acc > NIR] : 1.519e-10
##
##                    Kappa : 0.3651
##
##   Mcnemar's Test P-Value : 0.002176
##
##              Sensitivity : 0.5943
##              Specificity : 0.7725
##           Pos Pred Value : 0.7324
##           Neg Pred Value : 0.6450
##               Prevalence : 0.5117
##           Detection Rate : 0.3041
##     Detection Prevalence : 0.4152
##        Balanced Accuracy : 0.6834
##
##         'Positive' Class : Yes
##
```

```
default_tune <- sensitivity(as.factor(fits_0_predicts),as.factor(test_part$Sales),positive = "Yes")
```

### Choosing Parameters for crossvalidation and XGBosst tuning grid

At first we use a broad range of values to optimize our hyperparameters for XGboost

```
# Trying Broad Range of Values for XGBoost Tuning Grid
tune_grid <- expand.grid(
  nrounds = seq(from = 10, to = 200, by = 10),
  eta = c(0.025, 0.05, 0.1, 0.3),
  max_depth = c(2, 3, 4, 5, 6),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

# Cross-validation

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 10, # with n folds
  verboseIter = FALSE, # no training log
  allowParallel = FALSE # FALSE for reproducible results
)
```

### Training Model on Broad Range of Hyperparameters (Step 1)

We have opted not to use dimensions that have no variability, or dimensions that are unique identifiers for ads and campaigns. Dimensions that are simply unique identifiers will not be useful in predicting ad results because they will be different for each new campaign, and unlike gender and interest dimesions, these are not factors.We are also dropping approved conversion feature, because we used it to generated the "Yes/No" binary classification in the Sales dimension.

```r
# Using the dimensions that are most relevant and variable

fits <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+Impressions
              +Clicks+Spent+Total_Conversion, method = "xgbTree", data = train_part,trControl = tune_co
```

```r
# Looking at best tuning values
fits$bestTune
```

```
##     nrounds max_depth  eta gamma colsample_bytree min_child_weight
## 130     100         3 0.05     0                1                1
##     subsample
## 130         1
```

```r
# Creating Predictions

fits_predicts <- predict(fits,test_part)

confusionMatrix(as.factor(fits_predicts),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  133  73
##        Yes  34 102
##
##                Accuracy : 0.6871
##                  95% CI : (0.6351, 0.7359)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 3.486e-11
##
##                   Kappa : 0.3773
##
##  Mcnemar's Test P-Value : 0.0002392
##
##             Sensitivity : 0.5829
##             Specificity : 0.7964
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.6456
##              Prevalence : 0.5117
##          Detection Rate : 0.2982
##    Detection Prevalence : 0.3977
##       Balanced Accuracy : 0.6896
##
##        'Positive' Class : Yes
##
```

```r
first_tune <- sensitivity(as.factor(fits_predicts),as.factor(test_part$Sales),positive = "Yes")
```

Let's look at different tuning values for maxdepth and minimum child weight

**Tuning for maxdepth and minimum child weight (Step 2)**

```
tune_grid_2 <- expand.grid(
  nrounds = seq(from = 10, to = 200, by = 10),
  eta = fits$bestTune$eta,
  max_depth = seq(1:6),
  gamma = fits$bestTune$gamma,
  colsample_bytree = 1,
  min_child_weight = seq(1:3),
  subsample = 1
)

fits_2 <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+Impressions
                +Clicks+Spent+Total_Conversion, method = "xgbTree", data = train_part,trControl = tune_
                tuneGrid = tune_grid_2)

fits_predicts_2 <- predict(fits_2,test_part)

confusionMatrix(as.factor(fits_predicts_2),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  130  72
##        Yes  37 103
##
##                Accuracy : 0.6813
##                  95% CI : (0.629, 0.7304)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 1.519e-10
##
##                   Kappa : 0.3653
##
##  Mcnemar's Test P-Value : 0.001128
##
##             Sensitivity : 0.5886
##             Specificity : 0.7784
##          Pos Pred Value : 0.7357
##          Neg Pred Value : 0.6436
##              Prevalence : 0.5117
##          Detection Rate : 0.3012
##    Detection Prevalence : 0.4094
##       Balanced Accuracy : 0.6835
##
##        'Positive' Class : Yes
##
```

```
second_tune <- sensitivity(as.factor(fits_predicts_2),as.factor(test_part$Sales),positive = "Yes")


fits_2$bestTune
```

```
##    nrounds max_depth  eta gamma colsample_bytree min_child_weight
## 68      80         2 0.05     0                1                1
##    subsample
## 68         1
```

**Tuning for column and row sampling (Step 3)**

```
tune_grid_3 <- expand.grid(
  nrounds = seq(from = 10, to = 200, by = 10),
  eta = fits$bestTune$eta,
  max_depth = fits_2$bestTune$max_depth,
  gamma = fits$bestTune$gamma,
  colsample_bytree = c(0.4, 0.6, 0.8, 1.0),
  min_child_weight = fits_2$bestTune$min_child_weight,
  subsample = c(0.5, 0.75, 1.0)
)

fits_3 <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+Impressions
                +Clicks+Spent+Total_Conversion, method = "xgbTree", data = train_part,trControl = tune_
                tuneGrid = tune_grid_3)

fits_predicts_3 <- predict(fits_3,test_part)

confusionMatrix(as.factor(fits_predicts_3),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  130  72
##        Yes  37 103
##
##                Accuracy : 0.6813
##                  95% CI : (0.629, 0.7304)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 1.519e-10
##
##                   Kappa : 0.3653
##
##  Mcnemar's Test P-Value : 0.001128
##
##             Sensitivity : 0.5886
##             Specificity : 0.7784
##          Pos Pred Value : 0.7357
##          Neg Pred Value : 0.6436
##              Prevalence : 0.5117
##          Detection Rate : 0.3012
##    Detection Prevalence : 0.4094
##       Balanced Accuracy : 0.6835
##
##        'Positive' Class : Yes
##
```

```
third_tune <- sensitivity(as.factor(fits_predicts_3),as.factor(test_part$Sales),positive = "Yes")

fits_3$bestTune
```

```
##     nrounds max_depth  eta gamma colsample_bytree min_child_weight
## 227      70         2 0.05     0                1                1
##     subsample
## 227         1
```

### Tuning for Regularization (Step 4)

Now that we have the best tunes, we refine it a bit further by tuning for gamma and keeping other hyper-parameters static.

```
tune_grid_4 <- expand.grid(
  nrounds = seq(from = 10, to = 200, by = 10),
  eta = fits$bestTune$eta,
  max_depth = fits_2$bestTune$max_depth,
  gamma = c(0,5,10,15,20,25,30,35,40,45,50),
  colsample_bytree = fits_3$bestTune$colsample_bytree,
  min_child_weight = fits_2$bestTune$min_child_weight,
  subsample = fits_3$bestTune$subsample
)

# New fit to detect best value for gamma

fits_4 <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+Impressions
                +Clicks+Spent+Total_Conversion, method = "xgbTree", data = train_part,trControl = tune_
                tuneGrid = tune_grid_4)

fits_predicts_4 <- predict(fits_4,test_part)

confusionMatrix(as.factor(fits_predicts_4),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  129  70
##        Yes  38 105
##
##                Accuracy : 0.6842
##                  95% CI : (0.632, 0.7332)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 7.325e-11
##
##                   Kappa : 0.3708
##
##  Mcnemar's Test P-Value : 0.002855
##
##             Sensitivity : 0.6000
##             Specificity : 0.7725
##          Pos Pred Value : 0.7343
##          Neg Pred Value : 0.6482
##              Prevalence : 0.5117
##          Detection Rate : 0.3070
##    Detection Prevalence : 0.4181
##       Balanced Accuracy : 0.6862
##
##        'Positive' Class : Yes
##
```

```
fourth_tune <- sensitivity(as.factor(fits_predicts_4),as.factor(test_part$Sales),positive = "Yes")


fits_4$bestTune
```

```
##    nrounds max_depth  eta gamma colsample_bytree min_child_weight
## 24      40         2 0.05     5                1                1
##    subsample
## 24         1
```

###Optimizing learning rate (Step 5)

Now that we know the best tune for gamma, we will look for the best value for learning rate hyperparameter.

```
# Tuning for Learning Rate

tune_grid_5 <- expand.grid(
  nrounds = seq(from = 10, to = 2000, by = 10),
  eta = c(0.01, 0.015, 0.025, 0.05, 0.1),
  max_depth = fits_2$bestTune$max_depth,
  gamma = fits_4$bestTune$gamma,
  colsample_bytree = fits_3$bestTune$colsample_bytree,
  min_child_weight = fits_2$bestTune$min_child_weight,
  subsample = fits_3$bestTune$subsample
)

# New fit with improved learning rate

fits_5 <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+Impressions
                +Clicks+Spent+Total_Conversion, method = "xgbTree", data = train_part,trControl = tune_
                tuneGrid = tune_grid_5)

fits_predicts_5 <- predict(fits_5,test_part)

confusionMatrix(as.factor(fits_predicts_5),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  129  70
##        Yes  38 105
##
```

```
##                Accuracy : 0.6842
##                  95% CI : (0.632, 0.7332)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 7.325e-11
##
##                   Kappa : 0.3708
##
##  Mcnemar's Test P-Value : 0.002855
##
##             Sensitivity : 0.6000
##             Specificity : 0.7725
##          Pos Pred Value : 0.7343
##          Neg Pred Value : 0.6482
##              Prevalence : 0.5117
##          Detection Rate : 0.3070
##    Detection Prevalence : 0.4181
##       Balanced Accuracy : 0.6862
##
##        'Positive' Class : Yes
##
```

```
fifth_tune <- sensitivity(as.factor(fits_predicts_5),as.factor(test_part$Sales),positive = "Yes")


fits_5$bestTune
```

```
##     nrounds max_depth  eta gamma colsample_bytree min_child_weight
## 604      40         2 0.05     5                1                1
##     subsample
## 604         1
```

## Fitting the final model

We are now going to fit the final model by using the best tunes for each hyperparameters

```
final_grid <- expand.grid(
  nrounds = fits_5$bestTune$nrounds,
  eta = fits_5$bestTune$eta,
  max_depth = fits_2$bestTune$max_depth,
  gamma = fits_4$bestTune$gamma,
  colsample_bytree = fits_3$bestTune$colsample_bytree,
  min_child_weight = fits_2$bestTune$min_child_weight,
  subsample = fits_3$bestTune$subsample
)

# Final Fit

fits_final <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)
                    +Impressions+Clicks+Spent+Total_Conversion,
                    method = "xgbTree", data = train_part,trControl = tune_control,
                tuneGrid = final_grid)
```

```r
fits_final_pred <- predict(fits_final,test_part)

confusionMatrix(as.factor(fits_final_pred),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  129  70
##        Yes  38 105
##
##                Accuracy : 0.6842
##                  95% CI : (0.632, 0.7332)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 7.325e-11
##
##                   Kappa : 0.3708
##
##  Mcnemar's Test P-Value : 0.002855
##
##             Sensitivity : 0.6000
##             Specificity : 0.7725
##          Pos Pred Value : 0.7343
##          Neg Pred Value : 0.6482
##              Prevalence : 0.5117
##          Detection Rate : 0.3070
##    Detection Prevalence : 0.4181
##       Balanced Accuracy : 0.6862
##
##        'Positive' Class : Yes
##
```

```r
final_tune <- sensitivity(as.factor(fits_final_pred),as.factor(test_part$Sales),positive = "Yes")


# Trying with a larger cross-validation set

tune_control_f <- caret::trainControl(
  method = "cv", # cross-validation
  number = 25, # with n folds
  #index = createFolds(tr_treated$Id_clean), # fix the folds
  verboseIter = FALSE, # no training log
  allowParallel = FALSE # FALSE for reproducible results
)

fits_final_f <- train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)
                      +Impressions+Clicks+Spent+Total_Conversion,
                      method = "xgbTree", data = train_part,
                      trControl = tune_control_f,
                      tuneGrid = final_grid)

fits_final_pred_f <- predict(fits_final_f,test_part)
```

```
confusionMatrix(as.factor(fits_final_pred_f),as.factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  129  70
##        Yes  38 105
##
##                Accuracy : 0.6842
##                  95% CI : (0.632, 0.7332)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 7.325e-11
##
##                   Kappa : 0.3708
##
##  Mcnemar's Test P-Value : 0.002855
##
##             Sensitivity : 0.6000
##             Specificity : 0.7725
##          Pos Pred Value : 0.7343
##          Neg Pred Value : 0.6482
##              Prevalence : 0.5117
##          Detection Rate : 0.3070
##    Detection Prevalence : 0.4181
##       Balanced Accuracy : 0.6862
##
##        'Positive' Class : Yes
##
```

```
final_tune_with_larger_CV <- sensitivity(as.factor(fits_final_pred_f),as.factor(test_part$Sales),positi
```

Turns out, despite such extensive tuning, the accuracy doesn't improve much from the initial fit.

## Trying out the ensemble method

```
#Model Stacking

models <- c("kknn","adaboost","rf","Rborist")

# Training HULK model


fits_ensemble <- lapply(models, function(model){
  print(model)
  train(Sales~as.factor(age)+as.factor(gender)+as.factor(interest)+
          Impressions+Clicks+Spent+Total_Conversion,
        method = "xgbTree", data = train_part)
})
```

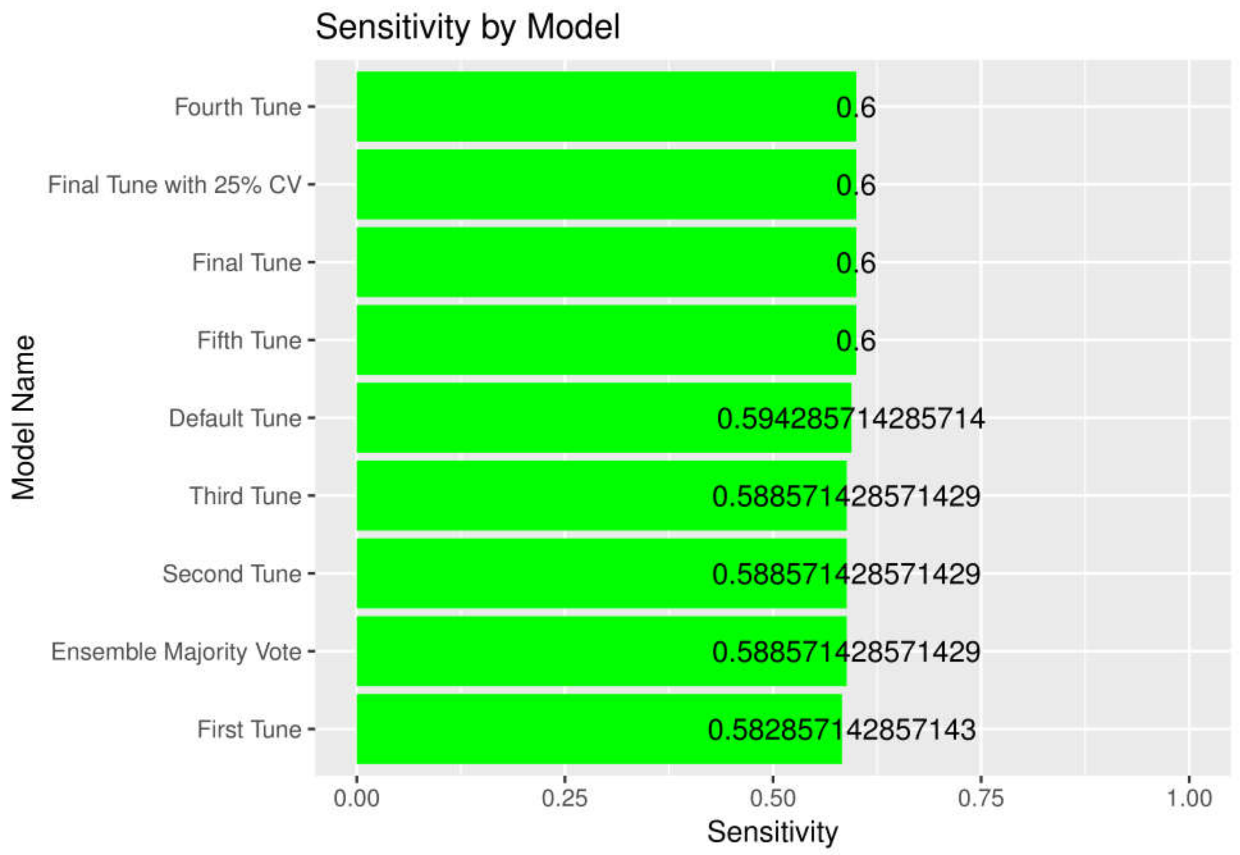
```
## [1] "kknn"
## [1] "adaboost"
## [1] "rf"
## [1] "Rborist"
```

```
names(fits_ensemble) <- models

# Creating a matrix of predictions

fits_ensemble_pred <- sapply(fits_ensemble, function(fits){
  predict(fits,test_part)
  })

total_fits <- as.data.frame(fits_ensemble_pred) %>% mutate(xgbTree=fits_final_pred_f)

# Using majority voting method

df <- data.frame(matrix(unlist(total_fits), nrow=length(total_fits), byrow=T))
colnames(df) <- seq(1:nrow(total_fits))
rownames(df) <- c("kknn","adaboost","rf","Rborist","xgbTree")

col_index <- seq(1,ncol(df), 1)
predict_vote <- map_df(col_index, function(j){
  vote <- ifelse(test = sum(df[,j] == "Yes") > 3, yes = "Yes", no = "No")
  return(tibble(vote = vote))
})   # returns a df

predict_vote <- as.factor(predict_vote$vote) #  as factor

confusionMatrix(factor(predict_vote),  factor(test_part$Sales),positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  132  72
##        Yes  35 103
##
##                Accuracy : 0.6871
##                  95% CI : (0.6351, 0.7359)
##     No Information Rate : 0.5117
##     P-Value [Acc > NIR] : 3.486e-11
##
##                   Kappa : 0.3771
##
##  Mcnemar's Test P-Value : 0.0005009
##
##             Sensitivity : 0.5886
##             Specificity : 0.7904
##          Pos Pred Value : 0.7464
##          Neg Pred Value : 0.6471
##              Prevalence : 0.5117
##          Detection Rate : 0.3012
##    Detection Prevalence : 0.4035
##         Balanced Accuracy : 0.6895
##
##          'Positive' Class : Yes
##
```

```
ensemble_method <- sensitivity(factor(predict_vote),  factor(test_part$Sales),positive = "Yes")
```

## Results



If we compare the model performances, we observe that tuning hyperparameters of XGBoost resulted in a slight increase in sensitivity. However, since in the final tune since we zeroed in on the optimum values for hyperparameters, training time was reduced significantly. So even though we didn't improve sensitivity by a large margin, we achieved higher performance in terms of reducing model training time. The majority voting ensemble method was slightly lower than the XGBoost models.

## Conclusion

This dataset was originally not intended for machine learning practices. Therefore the necessary features needed in order to make high accuracy predictions might not be present. Feature engineering might improve accuracy and sensitivity, but not by much, since features like spent, impressions and clicks are highly correlated to each other.The feature indicating gender has low variability in approved conversions.

In order to get greater sensitivity (aka true positives) one might have to sacrifice specificity. Using a large number of classification algorithms in majority voting ensemble may result in a high sensitivity output model, but in that case specificity and overall accuracy will suffer.