Codes: Spotify Research Paper

(Zi Ying) Sheila Teo

```r
library(dplyr)
library(tidyr)
library(ggplot2)
library(ggthemes)
library(data.table)
library(TTR)
library(xgboost)
library(h2o)
h2o.init()
library(groupdata2)
library(caret)
library(e1071)
library(mlr)
library(ROCR)
library(MlBayesOpt)
```

```r
options(repr.matrix.max.cols=100, repr.matrix.max.rows=100)
```

1) Import data

```r
df1 = read.csv('log_mini.csv')
df2 = read.csv('tf_mini.csv')
```

```r
df = inner_join(df1, df2, by=c('track_id_clean'='track_id'))
head(df)
```

2) Split data into training and test sets

EDA to be conducted only on training set since test data must be unseen

```r
length(unique(df$session_id))
```

10000

```r
count = 0
train = data.frame()

for (id in unique(df$session_id)) {
    count = count + 1
    if (count <= 0.8*10000) {
        subset = df %>% filter(df$session_id == id)
        train = bind_rows(train, subset)
    } else {
        break
    }
}
```

```r
length(unique(train$session_id))
```

8000

```
test = setdiff(df, train)
```

```
length(unique(test$session_id))
```

2000

3) Drop irrelevant columns

```
train$skip_1 = NULL
train$skip_3 = NULL
train$not_skipped = NULL
```

4) EDA

Investigate rs between session_position and target

```
ggplot(train, aes(x = session_position, fill = skip_2)) +
  geom_density(alpha = 0.4)
```

Investigate rs between hist_user_behavior_n_seekback and target

```
count = train %>% count(skip_2, hist_user_behavior_n_seekback)
count2 = train %>% count(hist_user_behavior_n_seekback)

get_prob = function(row) {
  seekback = as.numeric(row[2])
  n = as.numeric(row[3])
  total_n = count2 %>% filter(hist_user_behavior_n_seekback == seekback) %>% pull(n) %>% as.numeric()
  return (n/total_n)
}

count$prob = count %>% apply(MARGIN=1, FUN=get_prob)  #'MARGIN=1'=apply function across rows

ggplot(count, aes(x=hist_user_behavior_n_seekback, y=prob, fill=skip_2)) +
  labs(x='number of times the user did a seek backwards within track', y='probability of occurence') +
  scale_fill_discrete(name = 'track was skipped') +
  geom_bar(stat='identity') +
  theme_economist()
```

Investigate rs between hour_of_day and target

```
ggplot(train, aes(x = hour_of_day, fill = skip_2)) +
  geom_density(alpha = 0.4)
```

Investigate rs between date and target

```
train$day_of_week = weekdays(as.Date(train$date))
```

```r
count = train %>% count(skip_2, day_of_week)
count2 = train %>% count(day_of_week)

get_prob = function(row) {
  day = row[2]
  n = as.numeric(row[3])
  total_n = count2 %>% filter(day_of_week == day) %>% pull(n) %>% as.numeric()
  return (n/total_n)
}

count$prob = count %>% apply(MARGIN=1, FUN=get_prob)  #'MARGIN=1'=apply function across rows
```

```r
count$day_of_week = factor(count$day_of_week, levels = c('Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday'))

ggplot(count, aes(x=day_of_week, y=prob, fill=factor(skip_2))) +
    labs(x='day of week', y='probability of occurence') +
    scale_fill_discrete(name = 'track was skipped') +
    geom_bar(stat='identity') +
    theme_economist()
```

Investigate rs between release_year and target

```r
count = train %>% count(skip_2, release_year)
count2 = train %>% count(release_year)

get_prob = function(row) {
  val = row[2]
  n = as.numeric(row[3])
  total_n = count2 %>% filter(release_year == val) %>% pull(n) %>% as.numeric()
  return (n/total_n)
}

count$prob = count %>% apply(MARGIN=1, FUN=get_prob)  #'MARGIN=1'=apply function across rows

ggplot(count, aes(x=release_year, y=prob, fill=skip_2)) +
  labs(x='track release year', y='probability of occurrence') +
  scale_fill_discrete(name = 'track was skipped') +
  geom_bar(stat="identity") +
  theme_economist()
```

5) Create new predictors to feed sequential user action information into model

Create new var: skip_prop_prior_to_track

gives the proportion of tracks encountered prior to current track that were skipped

```r
get_prop_skipped = function(session_position, skip_2) {
  data = tibble(session_position, skip_2)
  names(data) = c('session_position', 'skip')

  data$skip = ifelse(data$skip == 'true', 1, 0)
```

```
  col = cumsum(data$skip)/data$session_position
  return(c(NA, col[1:length(col)-1]))
}

train = train %>% group_by(session_id) %>%
mutate(skip_prop_prior_to_track=get_prop_skipped(session_position, skip_2))
```

Create new var: skip_prop_prior_to_track_sd

gives the standard deviation of 'skip_prop_prior_to_track': tells us how consistent the user's skipping action was prior to current track

```
get_prop_skipped_sd = function(session_position, skip_prop_prior_to_track, skip_2) {
  data = tibble(session_position, skip_prop_prior_to_track, skip_2)
  names(data) = c('session_position', 'skip_prop', 'skip')

  cum_sd = runSD(data$skip_prop[2:length(data$skip_prop)], n=1, cumulative=TRUE)
  return(c(NA, 0, cum_sd[2:length(cum_sd)]))
}

train = train %>% group_by(session_id) %>% mutate(skip_prop_prior_to_track_sd=
get_prop_skipped_sd(session_position, skip_prop_prior_to_track, skip_2))
```

Create new var: skip_previous

gives whether or not the track right before this track was skipped by user

```
train = train %>% group_by(session_id) %>% mutate(skip_previous=lag(skip_2))
```

6) Encode categorical variables

For variables with 2 levels: Convert into binary variable

```
train$skip_2 = ifelse(train$skip_2 == 'true', 1, 0)
train$hist_user_behavior_is_shuffle = ifelse(train$hist_user_behavior_is_shuffle == 'true', 1, 0)
train$premium = ifelse(train$premium == 'true', 1, 0)
train$mode = ifelse(train$mode == 'major', 1, 0)
train$skip_previous = ifelse(train$skip_previous == 'true', 1, 0)
```

For variables with more than 2 levels: Regularized k-fold target encoding

```
#create a col 'fold' in data
train = train %>% ungroup() %>% fold(k = 5)
colnames(train)[52] = 'fold'
```

```
#convert columns from character type to factor type
train$day_of_week = as.factor(train$day_of_week)
train$session_length = as.factor(train$session_length)
train$release_year = as.factor(train$release_year)
train$hour_of_day = as.factor(train$hour_of_day)
train$key = as.factor(train$key)
train$time_signature = as.factor(train$time_signature)
```

```r
#fit target encoding model
te_model <- h2o.targetencoder(x = c('session_id', 'context_type', 'hist_user_behavior_reason_start',
'hist_user_behavior_reason_end', 'day_of_week', 'hour_of_day',
'release_year', 'key', 'time_signature', 'session_length'),
                              y = 'skip_2',
                              training_frame = as.h2o(train),
                              fold_column = 'fold',
                              data_leakage_handling = 'KFold',
                              blending = TRUE,
                              noise = 0.10,     #more noise = more regularisation
                              seed = 1234,
                              keep_original_categorical_columns = FALSE)
```

```r
#transform dataset using te model
train = h2o.transform(te_model, as.h2o(train)) %>% as.data.frame()
```

7) Create pipeline to transform test set

```r
pipeline = function(test) {
    #1) drop all other skip cols: useless information
    test$skip_1 = NULL
    test$skip_3 = NULL
    test$not_skipped = NULL

    #2) create new predictors to feed sequential user action information into model
        #skip_prop_prior_to_track
    get_prop_skipped = function(session_position, skip_2) {
        data = tibble(session_position, skip_2)
        names(data) = c('session_position', 'skip')

        data$skip = ifelse(data$skip == 'true', 1, 0)
        col = cumsum(data$skip)/data$session_position
        return(c(NA, col[1:length(col)-1]))
        }

    test = test %>% group_by(session_id) %>% mutate(skip_prop_prior_to_track=
    get_prop_skipped(session_position, skip_2))

        #skip_prop_prior_to_track_sd
    get_prop_skipped_sd = function(session_position, skip_prop_prior_to_track, skip_2) {
        data = tibble(session_position, skip_prop_prior_to_track, skip_2)
        names(data) = c('session_position', 'skip_prop', 'skip')

        cum_sd = runSD(data$skip_prop[2:length(data$skip_prop)], n=1, cumulative=TRUE)
        return(c(NA, 0, cum_sd[2:length(cum_sd)]))
    }

    test = test %>% group_by(session_id) %>% mutate(skip_prop_prior_to_track_sd=
    get_prop_skipped_sd(session_position, skip_prop_prior_to_track, skip_2))

        #skip_previous
    test = test %>% group_by(session_id) %>% mutate(skip_previous=lag(skip_2))
```

```r
    #3) transform existing predictors to increase utility for model
    test$day_of_week = weekdays(as.Date(test$date))

    #4) encode categorical variables
        #for variables with 2 levels: convert into binary variables
    test$skip_2 = ifelse(test$skip_2 == 'true', 1, 0)
    test$hist_user_behavior_is_shuffle = ifelse(test$hist_user_behavior_is_shuffle == 'true', 1, 0)
    test$premium = ifelse(test$premium == 'true', 1, 0)
    test$mode = ifelse(test$mode == 'major', 1, 0)
    test$skip_previous = ifelse(test$skip_previous == 'true', 1, 0)

        #for variables with more than 2 levels: use k-fold target encoding
    test = test %>% ungroup() %>% fold(k = 5)
    colnames(test)[length(test)] = 'fold'

    test$day_of_week = as.factor(test$day_of_week)
    test$session_length = as.factor(test$session_length)
    test$release_year = as.factor(test$release_year)
    test$hour_of_day = as.factor(test$hour_of_day)
    test$key = as.factor(test$key)
    test$time_signature = as.factor(test$time_signature)

    test = h2o.transform(te_model, as.h2o(test), as_training = FALSE, blending = FALSE, noise = 0)
    %>% as.data.frame()
    #using 'te_model' created from training data to transform test data -> don't need to apply any of
    #the overfitting prevention techniques because target encoding map used was created on
    #training data not testing data

    return(test)
}
```

```r
test = pipeline(test)
```

8) Create baseline XGBoost

```r
#subset train df to retain only columns to be used in model
train_to_use = train %>% select(-c('track_id_clean', 'date', 'fold',
'hist_user_behavior_reason_end_te'))
```

```r
#subset test df to retain only columns to be used in model
test_to_use = test %>% select(-c('track_id_clean', 'date', 'fold', 'hist_user_behavior_reason_end_te'))
```

Convert dataframe into matrix

```r
options(na.action='na.pass')
train_to_use_matrix = model.matrix(skip_2~.-1,data = train_to_use)
test_to_use_matrix = model.matrix(skip_2~.-1,data = test_to_use)
```

```r
train_matrix = xgb.DMatrix(data = train_to_use_matrix, label=train_to_use$skip_2)
test_matrix = xgb.DMatrix(data = test_to_use_matrix,label=test_to_use$skip_2)
```

Build baseline XGBoost: Use default hyperparameter values

```r
train %>% count(skip_2)
```

```r
test %>% count(skip_2)
```

from above, we see that our dataset is balanced -> no need to further consider for imbalanced data when deciding model evaluation metric

```r
#default hyparameter values from documentation
default_params = list(booster = 'gbtree', objective = 'binary:logistic', eta=0.3, gamma=0, max_depth=6,
min_child_weight=1, subsample=1, colsample_bytree=1)
```

```r
#conduct 5-fold CV to determine optimal nrounds
default_xgbcv = xgb.cv(params = default_params, data = train_matrix, nrounds = 200, nfold = 5,
showsd = T, metrics = list('logloss'), stratified = T, print_every_n = 10,
early_stopping_rounds = 30)

#note: 'test' in the output here is the validation from k-fold CV! not the actual test set
```

```
[1] train-logloss:0.588686+0.000303 test-logloss:0.589863+0.000436
Multiple eval metrics are present. Will use test_logloss for early stopping.
Will train until test_logloss hasn't improved in 30 rounds.

[11]    train-logloss:0.424116+0.001007 test-logloss:0.435552+0.002326
[21]    train-logloss:0.402416+0.001065 test-logloss:0.423159+0.003587
[31]    train-logloss:0.390672+0.001470 test-logloss:0.419368+0.002406
[41]    train-logloss:0.382689+0.002156 test-logloss:0.418169+0.003003
[51]    train-logloss:0.375001+0.002466 test-logloss:0.417502+0.002582
[61]    train-logloss:0.367820+0.001867 test-logloss:0.417031+0.002876
[71]    train-logloss:0.361355+0.001364 test-logloss:0.416475+0.003100
[81]    train-logloss:0.354957+0.001401 test-logloss:0.416129+0.003019
[91]    train-logloss:0.349463+0.001795 test-logloss:0.416292+0.003096
[101]   train-logloss:0.344487+0.002356 test-logloss:0.416901+0.003129
Stopping. Best iteration:
[79]    train-logloss:0.356258+0.001331 test-logloss:0.416067+0.003195
```

```r
#min holdout set log loss

min(default_xgbcv$evaluation_log$test_logloss_mean)
```

0.4160674

```r
#plot logloss against nrounds for both the training set and test set, with both log loss means and std

df_to_plot = default_xgbcv$evaluation_log %>%
            gather(key=dataset, value=logloss_mean, test_logloss_mean, train_logloss_mean) %>%
            gather(key=dataset2, value=logloss_std, test_logloss_std, train_logloss_std)

df_to_plot$dataset = ifelse(df_to_plot$dataset == 'test_logloss_mean', 'holdout set', 'training set')
df_to_plot$dataset2 <- NULL
```

```r
png(filename='default_xgb_cv.png', res=80)

df_to_plot %>%
  ggplot(aes(x = iter, group = dataset, color = dataset)) +
  geom_line(aes(y = logloss_mean, color = dataset), size = 1) +
```

```r
  geom_ribbon(aes(y = logloss_mean, ymin = logloss_mean - logloss_std, ymax = logloss_mean +
  logloss_std, fill = dataset), alpha = .2, colour = NA) +
  theme_bw()

dev.off()
```

```r
#fit model on training set using best nrounds obtained from CV and
#observe whether this best nrounds is true for our actual test set

default_xgb = xgb.train(params = default_params, data = train_matrix,
nrounds = default_xgbcv$best_iteration, watchlist =
list(train=train_matrix, test=test_matrix),
print_every_n = 10, eval_metric = 'logloss')
```

```
[1] train-logloss:0.589062  test-logloss:0.615843
[11]    train-logloss:0.425705  test-logloss:0.552938
[21]    train-logloss:0.406696  test-logloss:0.591446
[31]    train-logloss:0.393713  test-logloss:0.621367
[41]    train-logloss:0.382507  test-logloss:0.651339
[51]    train-logloss:0.376088  test-logloss:0.660107
[61]    train-logloss:0.372198  test-logloss:0.662259
[71]    train-logloss:0.365273  test-logloss:0.674581
[79]    train-logloss:0.361314  test-logloss:0.673856
```

```r
#min test set log loss

min(default_xgb$evaluation_log$test_logloss)
```

0.550488

```r
df_to_plot2 = default_xgb$evaluation_log %>%
              gather(key=dataset, value=logloss, test_logloss, train_logloss)

df_to_plot2$dataset = ifelse(df_to_plot2$dataset == 'test_logloss', 'test set', 'training set')
```

```r
#plot logloss against nrounds for both the training set and test set

png(filename='default_xgb_actual_test_set.png', res=100)

df_to_plot2 %>%
  ggplot(aes(x = iter, y = logloss, group = dataset, color = dataset)) +
  geom_line() +
  theme_bw()

dev.off()
```

```r
#predict on test set using default cut-off threshold of 0.5

default_xgb_pred = predict(default_xgb, test_matrix)
default_xgb_pred_target = ifelse(default_xgb_pred > 0.5, 1, 0)
```

```r
#obtain confusion matrix and performance metrics scores

confusionMatrix(as.factor(default_xgb_pred_target), as.factor(test_to_use$skip_2), positive='1')
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction     0     1
         0 11292  6955
         1  4812 10460

               Accuracy : 0.6489
                 95% CI : (0.6438, 0.6541)
    No Information Rate : 0.5196
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3003

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6006
            Specificity : 0.7012
         Pos Pred Value : 0.6849
         Neg Pred Value : 0.6188
             Prevalence : 0.5196
         Detection Rate : 0.3121
   Detection Prevalence : 0.4556
      Balanced Accuracy : 0.6509

       'Positive' Class : 1
```

```r
#roc curve
png(filename='default_xgb_roc.png', res=80)

default_xgb_rocr_prediction = prediction(default_xgb_pred, test_to_use$skip_2)
perf = performance(default_xgb_rocr_prediction, "tpr", "fpr")
roc_auc = performance(default_xgb_rocr_prediction, measure = "auc")

plot(perf, colorize=TRUE, main=paste('ROC-AUC = ', signif(roc_auc@y.values[[1]], 4)))

dev.off()
```

9) Tune XGBoost: Bayesian optimization with k-fold cross validation

```r
set.seed(1)

start.time = Sys.time()

tune = xgb_cv_opt(data = train_to_use,
                   label = 'skip_2',
                   objectfun = 'binary:logistic',
                   evalmetric = 'logloss',
                   n_folds = 5,  #use 5-fold CV
                   init_points = 2,  #no. of steps of random exploration to perform
                   n_iter = 30,  #no. of steps of bayesian optimization to perform
                   eta_range = c(0.1, 0.3L),
                   nrounds_range = c(80, 500L),
```

```
                   max_depth_range = c(1L, 4L),
                   subsample_range = c(0, 1L),
                   bytree_range = c(0, 1L)
                  )

end.time = Sys.time()
end.time - start.time
```

```
tune$Best_Par
```

eta_opt

0.3

max_depth_opt

1

nrounds_opt

473.202133104604

subsample_opt

0.96421950027094

bytree_opt

0.997748645492159

```
optimal_params = list(booster = 'gbtree', objective = 'binary:logistic', eta=0.3, gamma=0, max_depth=1,
subsample=0.96421950027094, colsample_bytree=0.997748645492159)
```

```
#train model using optimal hyperparameters obtained above

tuned_xgb = xgb.train(params = optimal_params, data = train_matrix, nrounds=473, watchlist =
list(train=train_matrix, test=test_matrix),
eval_metric = 'logloss', print_every_n=50)
```

```
[1] train-logloss:0.622441  test-logloss:0.620888
[51]    train-logloss:0.469634  test-logloss:0.530802
[101]   train-logloss:0.464444  test-logloss:0.537501
[151]   train-logloss:0.462356  test-logloss:0.541976
[201]   train-logloss:0.461035  test-logloss:0.543950
[251]   train-logloss:0.460085  test-logloss:0.545610
[301]   train-logloss:0.459344  test-logloss:0.546551
[351]   train-logloss:0.458737  test-logloss:0.548522
[401]   train-logloss:0.458245  test-logloss:0.550250
[451]   train-logloss:0.457815  test-logloss:0.551563
[473]   train-logloss:0.457637  test-logloss:0.552077
```

```
#min test set log loss
min(tuned_xgb$evaluation_log$test_logloss)
```

0.527136

```
df_to_plot3 = tuned_xgb$evaluation_log %>%
              gather(key=dataset, value=logloss, test_logloss, train_logloss)

df_to_plot3$dataset = ifelse(df_to_plot3$dataset == 'test_logloss', 'test set', 'training set')
```

```r
#plot logloss against nrounds for both the training set and test set

png(filename='tuned_xgb_actual_test_set.png', res=100)

df_to_plot3 %>%
  ggplot(aes(x = iter, y = logloss, group = dataset, color = dataset)) +
  geom_line() +
  theme_bw()

dev.off()
```

Regularize XGBoost

```r
#create tasks
traintask = makeClassifTask(data = train_to_use, target = 'skip_2')
testtask = makeClassifTask(data = test_to_use, target = 'skip_2')
```

```r
#create learner

xgb_learner = makeLearner('classif.xgboost', predict.type = 'prob')
xgb_learner$par.vals = list(
             objective='binary:logistic',
             eval_metric='logloss',
             eta=0.3,
             nrounds=473,
             max_depth=1,
             subsample=0.96421950027094,
             colsample_bytree=0.997748645492159
)
```

```r
#create parameter grid
params = makeParamSet(
         makeNumericParam("gamma", lower = 1, upper = 100)
)
```

```r
#set resampling strategy
rdesc = makeResampleDesc("CV", stratify=T, iters=5L)  #'stratify=T' ensures the distribution of target
#class is maintained in the resampled data sets during CV
```

```r
#set search strategy: a random search using 5 models with different gamma values

control = makeTuneControlRandom(maxit = 5L)
```

```r
#set parallel backend for faster computation
library(parallel)
library(parallelMap)
parallelStartSocket(cpus = detectCores())
```

```r
#conduct tuning and time it
start.time = Sys.time()

tuned_params = tuneParams(learner = xgb_learner,
                   task = traintask,
                   resampling = rdesc,
                   measures = acc,
```

```
                   par.set = params,
                   control = control,
                   show.info = TRUE)

end.time = Sys.time()
end.time - start.time
```

10) Evaluate performance of final XGBoost

```
final_params = list(booster = 'gbtree', objective = 'binary:logistic', eta=0.3, gamma=14, max_depth=1,
subsample=0.96421950027094, colsample_bytree=0.997748645492159)
```

```
#train model using optimal hyperparameters obtained above

final_xgb = xgb.train(params = final_params, data = train_matrix, nrounds=473, watchlist =
list(train=train_matrix, test=test_matrix),
eval_metric = 'logloss', print_every_n=50)
```

```
[1] train-logloss:0.622533  test-logloss:0.620939
[51]    train-logloss:0.469418  test-logloss:0.530802
[101]   train-logloss:0.464249  test-logloss:0.538970
[151]   train-logloss:0.462193  test-logloss:0.541858
[201]   train-logloss:0.461510  test-logloss:0.542963
[251]   train-logloss:0.461331  test-logloss:0.543862
[301]   train-logloss:0.461222  test-logloss:0.543981
[351]   train-logloss:0.461222  test-logloss:0.543986
[401]   train-logloss:0.461170  test-logloss:0.543872
[451]   train-logloss:0.461120  test-logloss:0.543814
[473]   train-logloss:0.461120  test-logloss:0.543873
```

```
#min test set log loss

min(final_xgb$evaluation_log$test_logloss)
```

0.526838

```
df_to_plot4 = final_xgb$evaluation_log %>%
              gather(key=dataset, value=logloss, test_logloss, train_logloss)

df_to_plot4$dataset = ifelse(df_to_plot4$dataset == 'test_logloss', 'test set', 'training set')
```

```
#plot logloss against nrounds for both the training set and test set

png(filename='final_xgb_actual_test_set.png', res=100)

df_to_plot4 %>%
  ggplot(aes(x = iter, y = logloss, group = dataset, color = dataset)) +
  geom_line() +
  theme_bw()

dev.off()
```

```
#predict on test set using default cut-off threshold of 0.5
final_xgb_pred = predict(final_xgb, test_matrix)
final_xgb_pred_target = ifelse(final_xgb_pred > 0.5, 1, 0)
```

```r
#obtain new confusion matrix and performance metrics scores

confusionMatrix(as.factor(final_xgb_pred_target), as.factor(test_to_use$skip_2), positive='1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 11504  3177
         1  4600 14238

               Accuracy : 0.768
                 95% CI : (0.7634, 0.7725)
    No Information Rate : 0.5196
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5337

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8176
            Specificity : 0.7144
         Pos Pred Value : 0.7558
         Neg Pred Value : 0.7836
             Prevalence : 0.5196
         Detection Rate : 0.4248
   Detection Prevalence : 0.5620
      Balanced Accuracy : 0.7660

       'Positive' Class : 1
```

Plot ROC

```r
png(filename='final_xgb_roc.png', res=80)

final_xgb_rocr_prediction = prediction(final_xgb_pred, test_to_use$skip_2)
perf = performance(final_xgb_rocr_prediction, "tpr", "fpr")
roc_auc = performance(final_xgb_rocr_prediction, measure = "auc")

plot(perf, colorize=TRUE, main=paste('ROC-AUC = ', signif(roc_auc@y.values[[1]], 4)))

dev.off()
```

Feature importance plot

```r
#impt: we use gain and not frequency as the metric for computing feature importance to avoid bias

png(filename='final_xgb_feature_imptance.png', res=80)

mat = xgb.importance(feature_names = colnames(train_to_use_matrix), model = final_xgb)
xgb.importance(model=final_xgb) %>% xgb.ggplot.importance(top_n=10, measure='Gain', rel_to_first = F)
```

```
dev.off()
```

11) Threshold-moving

```
expected_cost_list = NULL
threshold_list = NULL

for (threshold in seq(from=0, to=1, by=0.0005)) {
    classification = ifelse(final_xgb_pred > threshold, 1, 0)
    confusion_matrix = confusionMatrix(factor(classification, levels = c('0', '1')),
    as.factor(test_to_use$skip_2), positive='1')
    fp = confusion_matrix$table[2]
    fn = confusion_matrix$table[3]
    expected_cost = fp*1 + fn*2  #assume fn is 2 times the cost of fp
    expected_cost_list = append(expected_cost_list, expected_cost)
    threshold_list = append(threshold_list, threshold)
}

dat = data.frame(expected_cost_list, threshold_list)
```

```
png(filename='threshold_moving.png', res=100)

ggplot(dat, aes(x=threshold_list, y=expected_cost_list)) +
  labs(x='threshold for classifying positive', y='expected cost to spotify') +
  geom_line() +
  theme_bw()

dev.off()
```

```
dat %>% filter(expected_cost_list == min(expected_cost_list))

#optimal threshold is 0.2655
```

```
final_xgb_pred_target = ifelse(final_xgb_pred > 0.2655, 1, 0)
confusionMatrix(as.factor(final_xgb_pred_target), as.factor(test_to_use$skip_2), positive='1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0  7744   891
         1  8360 16524

               Accuracy : 0.724
                 95% CI : (0.7192, 0.7288)
    No Information Rate : 0.5196
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4373

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9488
```

```
         Specificity : 0.4809
      Pos Pred Value : 0.6640
      Neg Pred Value : 0.8968
          Prevalence : 0.5196
      Detection Rate : 0.4930
Detection Prevalence : 0.7424
   Balanced Accuracy : 0.7149

    'Positive' Class : 1
```