MS&E 226 Mini-Project 2

Joseph Wu & Min Cheol Kim

Regression Task

Baseline

For the baseline regression model, we chose to always predict the sample mean, without any variable selection or transformation. We chose this method because we want to see what a result with minimal manipulation of the data would look like. Since some of the covariates are categorical and need to be transformed or dropped, we didn't want to meddle with the data too much for our baseline predictions.

Sample mean (peak viewer) = 15.18

RMSE error = 263.79

Variable Selection and Transformation

From the total of 16 variables, we first used our intuition as well as understanding of the dataset and the real work to weed out irrelevant covariates, such as 'session id' and 'channel id', which are just digits for unique identification purposes. We decided to drop 'timezone', because there are too many missing entries. We also converted 'geo' into a binary variable of whether the session took place in the U.S., since there are more than 146 geographical categories and U.S. composes more than a third of the entries, whereas some geographical categories had very few data points. We then observe the correlation matrix of the remaining variables and remove the variables that have high dependency (such as 'language_channel' and 'language_session'). In the end, 11 variables are used as covariates for regression modeling.

See Fig 1 for the GGPairs plot and Fig 2 for a covariate-response scatterplot. All of the continuous variables are heavy-tailed (outliers in the extremities) and would benefit from log transformation. We took the log transformation of the response variable ('max_viewer'), and all continuous covariates ('video_bitrate', 'uptime', 'uptime_sec', 'channel_view_count').

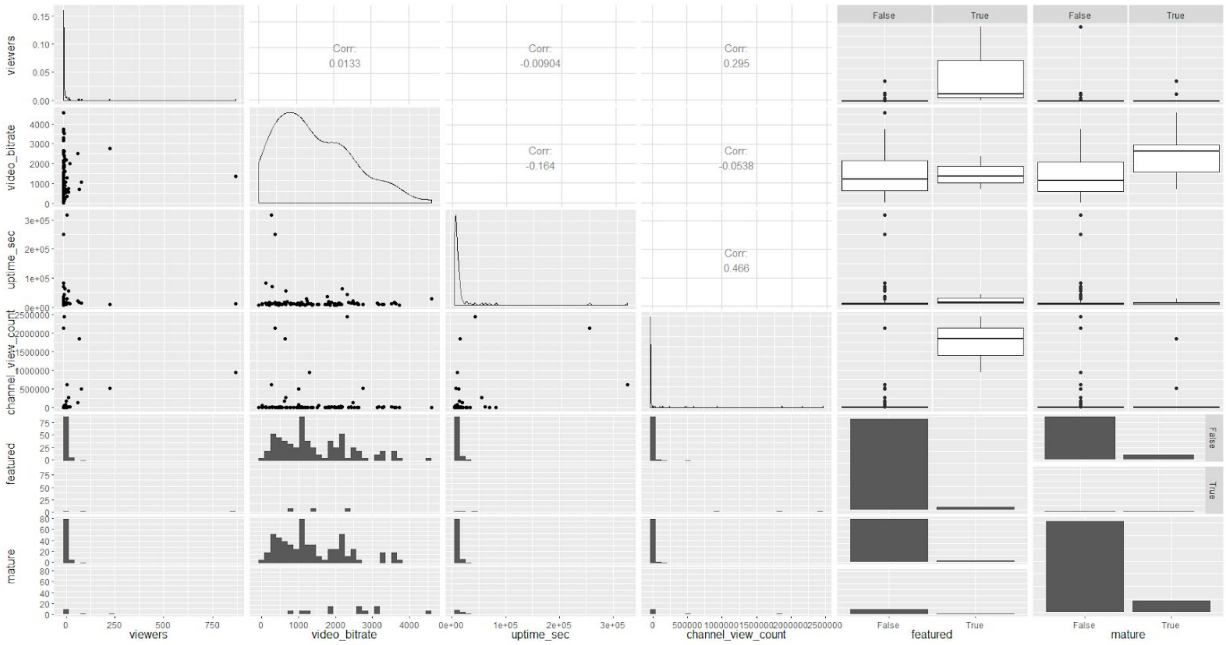


Fig 1: GGPairs Plot of all relevant variables

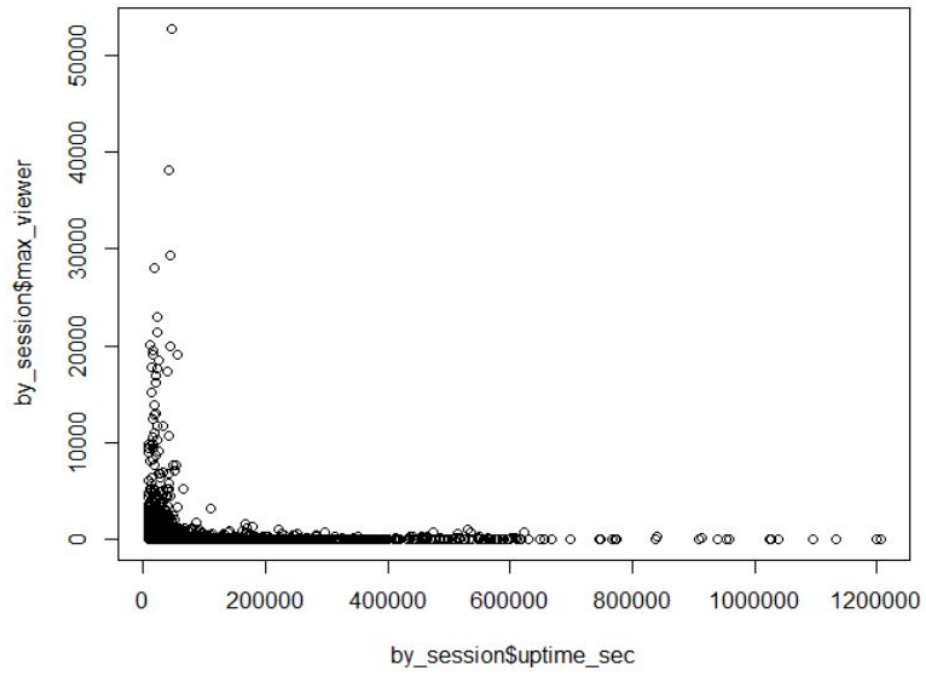


Fig 2: scatterplot of a covariate vs. response variable

Modeling approaches

For regression modeling, we tried 3 approaches and recorded the 5-fold CV error for each method:

- Linear Regression

  Coefficients:

(Intercept) video_bitrate uptime uptime_sec language_session channel_view_count broadcaster
1.125e+05 1.543e-03 -7.093e+00 1.411e-04 -9.864e-02 5.913e-05 -8.161e-02
featured channel_subscription mature US
4.427e+01 6.848e+02 -4.305e-01 -4.060e+00

CV error = 157.44

- Lasso Regression
  After standardizing the covariates and the response variable, we applied Lasso Regression, with varying penalty threshold, to see if we could get an improvement over OLS on the CV error

Lambda	0.1	0.5	1
CV Error	158.89	160.63	183.06

- Random Forest

We also tried a discrete classification method, random forest, to model the number of peak viewers in response to the 11 attributes of the streaming session. We experimented with different number of buckets, n_tree = 5, 10, 20. This method turned out very inaccurate for all the thresholds. We postulate that because of the spread of the data, where most sessions receive double-digit viewers whereas some reach tens of thousands of viewers, dividing the response variable into buckets fails to distinguish the difference between most of the data points.

Best Model Evaluation

Based on the 5-fold CV error, we chose Linear Regression to be our best model because it yielded the lowest CV error. Adding penalty thresholds in Lasso did not seem to improve accuracy of prediction, most likely because we have already pre-selected and removed irrelevant and highly dependent variables. The discrete method we tried, Random Forest, led to inaccurate results.

Since the model was trained and tested on the same set of data (data over a week) and the test set we plan on using will be data from another week, we expect the generalization/test error to increase because of the differences between train and test data, as they are taken from slightly different time period. However, the train set is the most faithful representation we could find so we don't expect any unreasonable spike in generalization error.

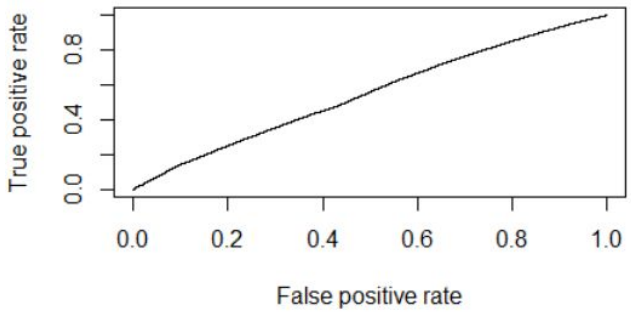
Classification Task

Baseline

The simplest baseline is always returning False for primetime, and it would actually achieve > 70% accuracy (average 0-1 loss) in the training dataset, since there are so much more sessions not in primetime than the sessions within primetime.

We created another baseline of including all covariates that make sense, with no interaction or higher order terms.

This baseline created a following ROC curve using our training dataset:



As you can see, the curve is very close to the line y=x and the model is not able to really distinguish the two possibilities (primetime vs otherwise).

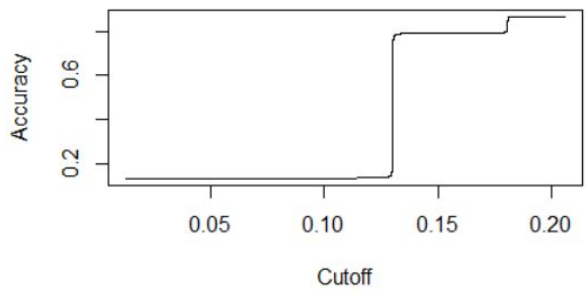
As expected, the AUC was very close to 0.5. Our calculated AUC for this baseline was 0.5469.

Modeling Approach & Evaluation

We did not intuitively think that k-nearest-neighbors algorithm applies well to our situation and some of our important covariates were numerical, which excluded Naive Bayes (at least the form we learned so far). Therefore, we built several models, all using logistic regression but with different interaction terms and transformations. We present one of the models below.
Difference from baseline: Interaction terms with max_viewer:channel_view_count, mature:featured, and max_viewer:uptime_sec. This modification seemed to make sense since these variables most likely cooperate to have some correlation on whether the session was streamed at primetime or not.

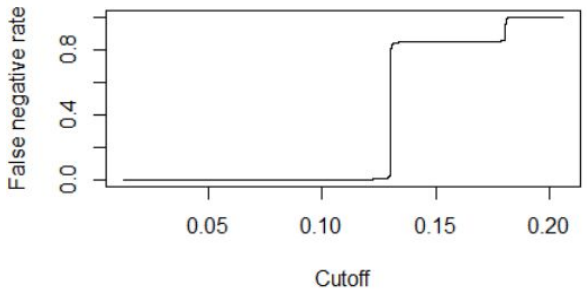
However, most interaction terms/transformations we performed did not really seem to affect the performance too well, at least what we could glean from the ROC curve.

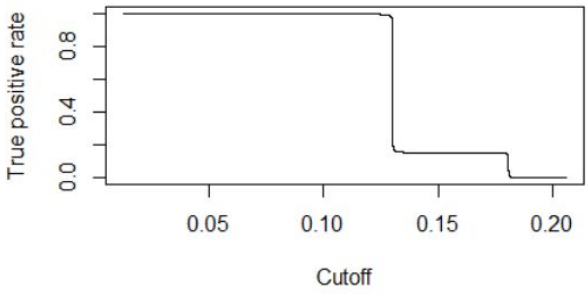
We plotted the accuracy for different cutoff values (the minimum output of regression needed to classify as a “true” value, or a 1):

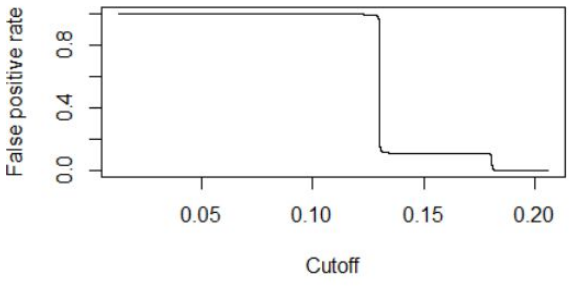


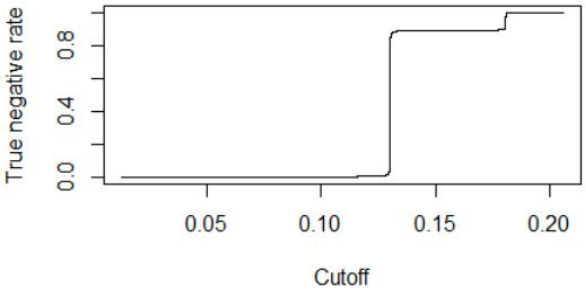
Because the data is heavily skewed to non-primetime sessions, accuracy does not really reflect a good measure of performance. If the primetime classification is used anywhere, it will most likely be used to figure out factors a person might consider when deciding when to hold a Twitch TV streaming session.

With that in mind, it is important to be able to interpret the coefficients to our model. In addition, goal should be to reduce false negative rate as much as possible, so that no shows that are good for primetime showing are ignored, even if some shows that might not fit the bill make it to the list. Reducing false negatives while increasing true positives is important even if it costs us true negatives and false positives. Therefore, I generated confusion matrix values vs cutoff graphs:

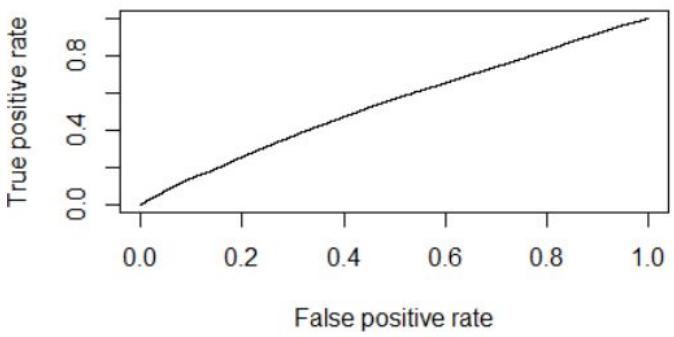








This suggested that this model in general had no good separation between primetime and non-primetime shows. I then checked the ROC curve on the training data and the corresponding AUC:



As expected, the ROC curve did not look like it was representing a good classifier, and the AUC value was 0.544. Regardless, a good cutoff value for this model certainly lies between 0.12 and 0.15, where it has some hope of discriminating true from false primetime tags.

Approximation of Prediction Error

The best model is the model presented above, which was not much better than the baseline in many ways. Other modeling approaches gave similar profiles for AUC/ROC curve and optimal cutoff points.

We were not yet completely sure what the best measure of success for this classification problem, so we report the accuracy as a validation measure here. We approximate our general accuracy to be around 0.79 in the test set. This number was calculated by dividing the training data into a validation and a training set, and a K-fold approach would improve this number. This time we were not able to perform a K-fold validation approach, and we will have to think of better performance metric/methods.

Regression Code

library(GGally)

```
library(chron)
library(cvTools)
load('C:/Users/Joseph/Documents/R/cleanData')

#removing NA and empty entries#
by_session = by_session[-which(by_session$uptime == ""), ]
by_session = by_session[-which(is.na(by_session$max_viewer)), ]
by_session$US = by_session$geo == 'US'
by_session = subset(by_session, select = -c(geo))

cleaned_timestamp = c()
for (i in 1:nrow(by_session)) {
  timestamp = toString(by_session$uptime[i])
  ts = unlist(strsplit(timestamp, "\\s+"))
  cleaned_timestamp[i] = chron(paste(ts[2], ts[3], ts[5]), ts[4], format = c('mon d y', 'h:m:s'))
}

by_session$uptime = cleaned_timestamp


# sample mean predictions #
sample_mean = mean(by_session$max_viewer)
RMSE_sm = sqrt(mean((by_session$max_viewer - sample_mean)^2))

# observe correlation of covariates#
cor(data.matrix(by_session))

# linear regression #
by_session.reg = subset(by_session, select = -c(session_id, channel_id, language_channel,
timezone, producer))
by_session.reg = as.data.frame(data.matrix(by_session.reg))
fm = lm(max_viewer ~ ., data = by_session.reg)
cv.lm = cvFit(fm, data=by_session.reg, y=by_session.reg$max_viewer, K=5)


##### Classification Code ######
library(chron)
library(ROCR)
library(boot)
# Clean Data ----
by_session = by_session[-which(by_session$uptime == ""), ]

cleaned_timestamp = rep(chron(), nrow(by_session))
```

```
primetime = vector(length=nrow(by_session))
for (i in 1:nrow(by_session)) {by
  timestamp = toString(by_session$uptime[i])
  ts = unlist(strsplit(timestamp, "\\s+"))
  temp = chron(paste(ts[2], ts[3], ts[5]), ts[4], format = c('mon d y', 'h:m:s'))
  time = strsplit(ts[4], ':')
  if (as.numeric(time[[1]][1]) > 18 && as.numeric(time[[1]][1]) < 22) {
    primetime[i] = 1;
  }
  cleaned_timestamp[i] = temp
}

by_session$uptime = cleaned_timestamp
by_session$primetime = primetime
classification_data = by_session
classification_data$mature = as.logical(classification_data$mature)*1
classification_data$producer = as.logical(classification_data$producer)*1
classification_data$channel_subscription =
as.logical(classification_data$channel_subscription)*1
classification_data$featured = as.logical(classification_data$featured)*1
classification_data$viewed = (classification_data$max_viewer > 3)*1


# Primetime Classification Task -----

# Baseline

fm_baseline = glm(primetime ~ max_viewer + video_bitrate + uptime_sec + channel_view_count
+ featured + mature + channel_subscription, family='binomial',data=classification_data)

prob = predict(fm_baseline1, classification_data, type='response')
pred = prediction(prob, classification_data$primetime)

fnr.perf = performance(pred, measure = "fnr")
tpr.perf = performance(pred, measure = "tpr")
tnr.perf = performance(pred, measure = "tnr")
fpr.perf = performance(pred, measure = "fpr")
plot(fnr.perf)
plot(tpr.perf)
plot(fpr.perf)
plot(fpr.perf)

# Get ROC curve and AUC
```

```
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
auc <- performance(pred, measure = "auc")
auc <- auc@y.values[[1]]
auc

# Model 1

fm_model_1 = glm(primetime ~ max_viewer + uptime_sec + channel_view_count + featured +
mature + max_viewer:channel_view_count + mature:featured + max_viewer:uptime_sec,
family='binomial',data=classification_data)

prob = predict(fm_model_1, classification_data, type='response')
pred = prediction(prob, classification_data$primetime)

acc.perf = performance(pred, measure = "acc")
plot(acc.perf)

fnr.perf = performance(pred, measure = "fnr")
tpr.perf = performance(pred, measure = "tpr")
tnr.perf = performance(pred, measure = "tnr")
fpr.perf = performance(pred, measure = "fpr")
plot(fnr.perf)
plot(tpr.perf)
plot(fpr.perf)
plot(tnr.perf)

# Get ROC curve and AUC
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
auc <- performance(pred, measure = "auc")
auc <- auc@y.values[[1]]
auc

# Divide into train/validate
val_rows = sample(nrow(classification_data), 50000)
val_set = classification_data[val_rows, ]
train_set = classification_data[-val_rows, ]
fm_model_1 = glm(primetime ~ max_viewer + uptime_sec + channel_view_count + featured +
mature + max_viewer:channel_view_count + mature:featured + max_viewer:uptime_sec,
family='binomial',data=train_set)
prob = predict(fm_model_1, val_set, type='response')
preds = (prob > 0.15)*1
```

```
correct = 0
for (i in 1:nrow(val_set)){
  if (!is.na(preds[i]) && !is.na(val_set$primetime[i])){
    if (preds[i] == val_set$primetime[i]) {
      correct = correct + 1
    }
  }
}
accuracy = correct/50000
accuracy
```