Techniques

December 5, 2015

## Preparation

Here I do some analysis on the model used for tea recommendations. Note that this is an R Markdown document -- it interweaves markdown (a typesetting language) with R (a statistical programming language). It is easy to view and execute using RStudio (an IDE for R).

I'm going to source my functions and write an additional logging function.

source('../functions.R')

## Loading required package: SnowballC

library(ggplot2)  
library(scales)  
library(AUC)

## AUC 0.3.0  
## Type AUCNews() to see the change log and ?AUC to get an overview.

getAuc <- function(num.prediction, int.actual) {  
 auc(roc(num.prediction, factor(int.actual)))  
}  
  
logNote <- function(...) do.call(cat, c(list(as.character(Sys.time()), ': '), list(...), list('\n', sep = '')))

Now, pull in the data and use my function to perform the initial calculations

lst.data <- getData()

## Warning in fread("../data/Tealuxe.All.csv"): Bumped column 9 to type  
## character on data row 7, field contains '113; 2'. Coercing previously read  
## values in this column from logical, integer or numeric back to character  
## which may not be lossless; e.g., if '00' and '000' occurred before they  
## will now be just '0', and there may be inconsistencies with treatment of  
## ',,' and ',NA,' too (if they occurred in this column before the bump).  
## If this matters please rerun and set 'colClasses' to 'character' for this  
## column. Please note that column type detection uses the first 5 rows, the  
## middle 5 rows and the last 5 rows, so hopefully this message should be very  
## rare. If reporting to datatable-help, please rerun and include the output  
## from verbose=TRUE.

dt.data <- combineData(lst.data)  
mat.similarity <- computeDistanceMatrix(dt.data)  
dt.similarity <- getSimilarityTable(mat.similarity)

To calculate performance, I'm going to pull out the review data (from whichever vendors provide such data) and use that as my test set. Essentially the model is supervised, but I'm building a new model for each person.

dt.reviews <- rbindlist(lapply(lst.data, function(x) {  
 if('review' %in% names(x)) {  
 dt.reviews <- rbindlist(lapply(1:nrow(x), function(y) {  
 mxt.review <- x[y, review][[1]]  
 if(is.data.table(mxt.review)) {  
 mxt.review <- cbind(mxt.review, url = x[y, link], vendor = x[y, vendor])  
 return(mxt.review)  
 } else {  
 return(NULL)  
 }  
 }))  
 }  
}))  
# need to clean this -- these get cast as characters  
dt.reviews[, score:=as.numeric(score)]

## Performance Analysis

I expect the predictive power/accuracy will vary as a function of the training set size. Here I explicitly test various training set sizes in order to find whether/when this modeling adds value.

First, I need to restrict my review data to reviewers who have more than one review.

dt.reviews <- dt.reviews[reviewer %in%   
 dt.reviews[, list(n=.N),by=list(reviewer)][  
 n > 1, reviewer] ]  
dt.response <- dt.reviews[, list(score, reviewer, vendor, url, like = score >= median(score))]  
dt.response <- dt.response[, list(reviewer, url, like)]  
# Remove duplicates (just give me the earliest like/dislike)  
setkey(dt.response, reviewer, url)  
dt.response <- unique(dt.response)

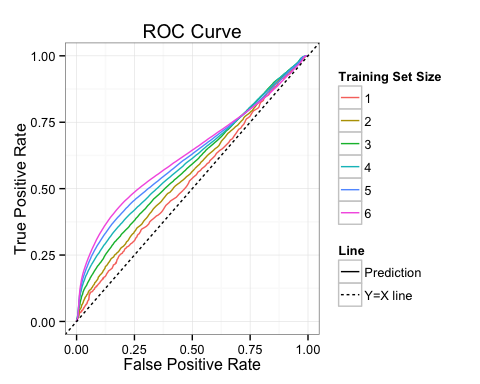
Now build the training sets

num.max.training <- dt.response[, .N, reviewer][, quantile(N,0.95)] # Want reasonable # of samples  
# dt.testing <- dt.testing[reviewer %in% sample(reviewer,15)]  
num.rows.expected <- sum(vapply(dt.response[, .N, by=list(reviewer)][, N],   
 function(x) {  
 sum(vapply(1:num.max.training, function(k) {  
 x \* choose(x-1,k) \* k  
 }, numeric(1)))  
 },  
 numeric(1)))  
dt.response[, N:=.N, by=list(reviewer)]  
dt.choose <- rbindlist(lapply(1:num.max.training, function(x) {  
 dt.response[N > x][, (function(dt) {  
 # logNote('Working on ', reviewer, ' with ', x, ' training size out of ', .N, ' possible.')  
 # Make sure I'm doing n choose k with k <= n.  
 if(.N <= x) return(data.table(  
 url\_testing = character(0),   
 training\_set\_size = numeric(0),   
 url\_training = character(0),   
 like\_training = logical(0),   
 id=numeric(0)))  
   
 # Now find all possible training sets of size x.  
 # Return as a data.table  
 lst.combn <- lapply(seq\_along(dt$url), function(y) {  
  
 lst.combinations <- combn(  
 setdiff(dt$url,dt$url[y]),  
 x,  
 simplify=FALSE)  
   
 lst.combinations <- lapply(lst.combinations, function(chr.urls) {  
 data.table(url\_testing = dt$url[y],   
 training\_set\_size = x,   
 url\_training = chr.urls,   
 like\_training = dt[match(chr.urls, url), like])  
 })  
 lst.combinations <- lapply(seq\_along(lst.combinations), function(num.id) {  
 cbind(lst.combinations[[num.id]], id = num.id)  
 })  
 dt.combn <- rbindlist(lst.combinations)  
  
 return(dt.combn[, list(  
 url\_testing,  
 training\_set\_size = as.double(training\_set\_size),  
 url\_training,  
 like\_training,  
 id = as.double(id)  
 )])  
 })  
 dt.combn <- rbindlist(lst.combn)  
 return(dt.combn)  
 })(.SD), by=list(reviewer)]  
}))  
  
if(nrow(dt.choose) != num.rows.expected) stop('Error in building training samples -- rows didn\'t match expectations')  
  
# Use a hash to avoid repeated calculations.  
# The hash is the entire training set (each tea in the training set and whether it is liked or not)  
# Clearly I'm using a memory-infefficient but easily-reversible hashing function  
dt.choose <- dt.choose[order(url\_training)]  
dt.choose[, hash:=paste0(url\_training, ' ', like\_training, collapse = ','), by=list(reviewer, training\_set\_size, id, url\_testing)]  
  
dt.hash <- unique(dt.choose[, list(url\_training, like\_training, hash)])  
dt.hash <- dt.hash[, recommendTea(chr.liked.teas = url\_training[like\_training], url\_training[!like\_training], dt.similarity), by=list(hash)]  
dt.predict <- merge(unique(dt.choose[, list(hash, url\_testing, reviewer, training\_set\_size, id)]),   
 dt.hash[, list(hash,   
 url\_testing = recommendation,   
 likelihood,   
 most\_similar\_tea,   
 least\_similar\_tea,   
 rank)],   
 by=c('hash','url\_testing'))  
  
dt.predict[, hash:=NULL]  
dt.predict <- merge(dt.predict,  
 dt.response[, list(reviewer, url\_testing = url, like\_testing = like)],  
 by=c('reviewer','url\_testing'))

Now check the performance.

Since there's a new 'model' for each person in order to predict which teas they'd like, we also wanted to see how useful it is to increase the number of teas used to train each person's model. These are the differently colored lines in the graph.

dt.roc <- rbindlist(lapply(seq(min(dt.predict$likelihood),max(dt.predict$likelihood),length.out = 1000),   
 function(x) dt.predict[, list(training\_set\_size, like\_prediction = likelihood >= x, like\_testing)][  
 , list(true\_positive = sum(like\_prediction & like\_testing),  
 true\_negative = sum(!like\_prediction & !like\_testing),  
 false\_positive = sum(like\_prediction & !like\_testing),  
 false\_negative = sum(!like\_prediction & like\_testing),  
 threshold = x),  
 by=list(training\_set\_size)]))  
dt.roc[, false\_positive\_rate := false\_positive / (false\_positive + true\_negative)]  
dt.roc[, true\_positive\_rate := true\_positive / (true\_positive + false\_negative)]  
ggplot(dt.roc, aes(false\_positive\_rate, true\_positive\_rate, linetype = 'Prediction', colour = factor(training\_set\_size))) +   
 geom\_line() +   
 geom\_abline(slope = 1, intercept = 0, aes(linetype = 'Y=X line')) +   
 labs(title = 'ROC Curve', colour = 'Training Set Size', x = 'False Positive Rate', y = 'True Positive Rate', linetype = 'Line') +   
 theme\_bw()



dt.predict[, list(`Area Under the ROC Curve` = getAuc(likelihood, like\_testing)), list(`Training Set Size` = training\_set\_size)]

## Training Set Size Area Under the ROC Curve  
## 1: 1 0.5348271  
## 2: 2 0.5582116  
## 3: 3 0.5849796  
## 4: 4 0.6040885  
## 5: 5 0.6184453  
## 6: 6 0.6308236

Here we can see that our results appear to be better than a coin-flip, although not by a huge margin. Results do improve as we increase our training size.

## Conclusion

The techniques used here are a great starting point for the business plan, allowing us to bootstrap recommendations. As an operating business, we'd in fact be able to improve our predictive abilities frequently by leveraging the proprietary data we'd gain by having users submit their own reviews, ratings, and opinions. In the same way that Google initially relied on PageRank (an algorithm applied to an essentially public data set) and has eventually moved to leverage their data on user behavior to improve results, we could initially rely on this data and eventually improve our modeling through our proprietary data.