MS&E 226 - Small Data

Mini Project Report - Part II

SUNet ID: [dhruvj, tromero1]
Name: [Dhruv Joshi, Tyler Romero]

1 Prediction and Classification Tasks - Overview

At a high level, our intention is to build clever statistical models which will help us predict whether users will click on an advertisement on a page, using the data from Outbrain Inc. (an online advertising platform). The provided dataset contains information about the webpage, the users and the advertisement topics. Having done an exploratory data analysis previously, our next goal is to find predictive models for our continuous and binary response variables. Since our data was in an anonymized form, where `document_topics, document_id, traffic_source` and other metadata about documents (webpages on which the advertisements were displayed) and users (people visiting the website and hopefully clicking on ads) was given in the form of text labels (Appendix 1), there was little scope for using intuition or insights to reason at a deep level about the effect of the covariates on response. The only metadata we had about users was their country of origin and device used. In this report, we will describe a systematic approach (termed by some as 'kitchen sink analysis') we have taken to finding promising models. A smaller part would describe some intuition that we have used to reason about promising covariates.

2 A Note on Data Preprocessing

To facilitate the process of model selection and to ensure that our efforts were spent efficiently towards understanding the process of model selection rather than dealing with practical difficulties of dealing with massive and complicated datasets, we performed some preprocessing steps, outlined below.

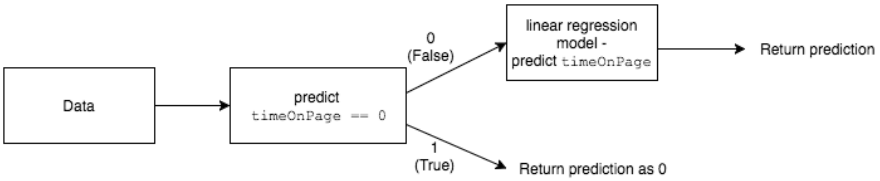
Using a smaller dataset: Since there is a massive number of rows in our dataset from the competition, we are going to restrict our analysis to a subset of them for the purposes of training (that R would be able to handle on a single machine, without having to use a very high-end machine to do this on). We chose a randomly sampled dataset of size 10,000 rows, both for the binary and the continuous response variables (randomly sampled with different seed values). Rows from all different tables (`page_views, events, documents_meta, clicks`) were unified into one single dataframe which would contain all covariates and the desired response variable. The covariate *timeOnPage* was removed from the dataset for the binary response variable, since it acts like an 'Oracle' for whether someone clicks on an advertisement.

Expansion of timestamp: The *timeStamp* feature in the dataframe was in POSIX format - it was represented as the number of milliseconds since January 1, 1970. This is clearly not very useful information, and so was converted into the day of week (0-6), day of month (1-31), hour, min, sec in the UTC timezone (year and month is not required since the dataset is from 14-28 June 2016). Conversions of timezones is not required since the classification algorithm would relate time which the user was on a page to their geolocation.

Removal of 'low utility' covariates: The userid ('uuid') was removed as this was causing regression training times to go up by large magnitudes, since it is a categorical variable taking on values almost equal to n, the number of rows (extremely few users were repeated in the dataset). Hence this would cause new binary covariates of the order $\sim n$ to be formed, which is also detrimental to the utility of the algorithm. This was removed prior to training. Similarly, `publish_time` (when the host webpage was published) and `geo_location` was not used in the initial analysis.

3 Continuous Response Variable

The continuous response variable is the time the user spends on the page (calculated by taking the difference between `eventTimestamp` and `loadTimestamp`). 58% of the values in the dataset were found to be 0s for this (most likely a scenario when the user bounces from the page), and the nonzero values were huge ($10^5 - 10^{20}$, see Appendix 2 - we used the log transform on this data for gaining better insights). Hence a linear predictor might not perform so well directly predicting `timeOnPage`, and we split up our model into two sub-parts. This is illustrated in the schematic diagram:



Stage I: predict if timeOnPage=0 For the first part of the predictive model, a new binary covariate was added (0 if `timeOnPage` == 0 and 1 otherwise). A family of models was trained by choosing covariates using regularization, forward and backward selection, we would use cross-validation to select the best performing model - since in-sample error estimation (we used AIC in this case) might get very biased if we use it as the sole criterion for model selection. We would use ROC (Receiver operating characteristic) curves to select the best threshold value for the final outcome. For cross-validation, 20% of the training data was kept in a holdout set. Using forward selection on logistic regression, we got the following order of covariates in order of increasing marginal error (and thus reducing effect on prediction) - `document_id` > `platform` > `display_id` > `topic_id` > `publisher_id` > `topic_confidence_level` > `monthDay` > `traffic_source` > `category_confidence_level`. It is interesting to note that the remaining covariates, viz those indicating the time the user accessed the page (weekday, second, hour, minute, loadTimeStamp) and `source_id, category_id`, which represented IDs classifying the source (from where the user reached the page) and the category of the document based on classification by outbrain. This might indicate that the classes/IDs that outbrain uses for document classification might need refinement. The most important were the `document_id` and `platform`, which makes complete sense - since the device the user is on and the specific document they are on would affect them bouncing from the page the highest amount. We selected the top 10 models through a combination of inputs - sharply reducing AIC through forward stepwise regression, ridge (Appendix 3) and lasso, keeping number of selected covariates as less as possible (we did not go above second order interaction terms). We compared the ROC curves for these plots, and chose the best performing one (we can tolerate false positives but we cannot tolerate false negatives - i.e. if we misclassify a user as bouncing even though they do stay on the page). Hence we will be optimizing for False Negative Rate (FNR).

A final model was selected by setting the threshold = 0.5 and comparing training and CV errors. The best performing model turned out to be the one with all covariates (no. 7 in the table in Appendix 5.3.5), hence we continued with this one. We optimized the threshold λ for FNR at $\lambda = 0.41$ (intersection point between FNR and 0-1 loss, Appendix 5.3).

Stage II: predict timeOnPage as a continuous variable A similar protocol was followed in this case, viz. the splitting of the dataset into a 'training' and cross-validation set (80-20 split). For training the continuous variable, the 'oracle' version of the training set is used, i.e. the rows with nonzero values of `timeOnPage` (since we are *conditioning* on the fact that the times are nonzero

for this classifier). This resulted in a smaller dataset, with 3189 rows (slightly more than half were 0, as discussed earlier). A similar procedure was followed in selecting the best covariates, using ridge regression, lasso and forward/backward stepwise regression (see Appendix 5.4). This allowed us to select the 10 best models based on the AIC, (while preventing the number of covariates from being too large). We evaluated training and cross-validation errors on all these models to select the best one. This turned out to be model 2, a model containing a large (21) number of covariates, including 12 interaction terms. When a residuals plot was done for this variable, it showed a lot of bias and it is clearly not able to fig the data properly. This is to be expected, since using categorical variables to predict a continuous variable is a tricky problem to solve using linear regression. The reported cross-validation RMSE is 170766770.

4 Binary Response Variable

We followed a three step process in order to choose the best model to predict our Binary Response Variable, which is whether or not the given Ad will be clicked on (`clicked`). First, we choose the best family of models for the problem. Second, we select covariates. Third, we determine how to set our selection threshold in order to optimize our model. We assume that the result of each of these steps is independent of the others.

Stage I: Select Model Family We compared three different families of models: Logistic Regression, Naive Bayes Classification, and Support Vector Machines (SVMs). We used ROC curves to compare between Model Families (Appendix 5.5.1). We used the same covariates as we generated each ROC curve. The curve for Naive Bayes make it evident that it is little better than random guessing. Logistic Regression and SVMs both offer an improvement on Naive Bayes, but they still are not great in terms of area-under-curve (AUC). This indicates that predicting if an advertisement will be clicked on is a very difficult task. Logistic Regression offers a larger AUC than SVMs, and it is significantly faster to train a logistic model, so we choose **Logistic Regression** for our model family.

Stage II: Select Covariates Armed with the knowledge that logistic regression is a good way to approach this problem, we set out to determine which covariates to include in our model. We used forward stepwise selection, ridge regularization, and lasso regularization in order to determine the best covariates. *Forwards Stepwise Selection* found that the model with the lowest AIC is one that contains platform, source_id, publisher_id, display_id, ad_id, category_confidence_level, traffic_source, and several interaction terms. *Ridge Regularization* shows that platformmobile, topic_confidence_level, category_confidence_level, traffic_sourcesearch, and traffic_sourcesocial are the most relevant covariates (Appendix 5.5.2). *Lasso Regularization* immediately pushes all covariate coefficients except platformmobile to zero. Using this information, we decided to use the covariates returned by **Forward Stepwise Selection** in our model, because this formula resulted in the lowest AIC score.

Stage III: Select Threshold Finally, we needed to choose a threshold to take the value between zero and one returned by logistic regression and use it to make a prediction. We plotted the Zero-One loss for different threshold values (Appendix 5.5.3), and saw that since the data is approximately 80% "No-Clicks", our model could guess all zeros and achieve a loss of 0.2. However, when it comes to internet advertising, false positives are completely acceptable, as long as they are accompanied by a significant portion of true positives. Therefore, instead of optimizing our

threshold for Zero-One Loss, we are going to optimize it for **Precision**. A plot of threshold vs. precision is available in Appendix 5.5.3. For a threshold value of 0.29, we were able to achieve a precision of 0.5! At this threshold value, however, there are still very few points that are classified as positive as a fraction of the total number of positives in the data, as can be seen in our plot of Sensitivity vs. Threshold. Since we really want true positives, even at the expense of more false positives, we believe that a **threshold set at 0.27** is the ideal value, as it provides a compromise between precision and sensitivity. This threshold provides a precision of 0.33, or, in other words, one-third of all of the points we classify as positive are true positives. This is a testament to how difficult it is to predict clicks.

How Good is Our Model?: Hold-Out Cross Validation We wish to **estimate the precision** of our model on our test-set. In order to do this, we created a hold-out set from our training data (before performing any analysis). Using this hold-out set, we plotted Precision vs. Threshold once again. On our hold-out set, our model achieves a precision of **0.36**, even higher than on our training set! This is most likely due to chance: we just selected a favorable hold-out set.

5 Appendix

5.1 A sample of the unified tables

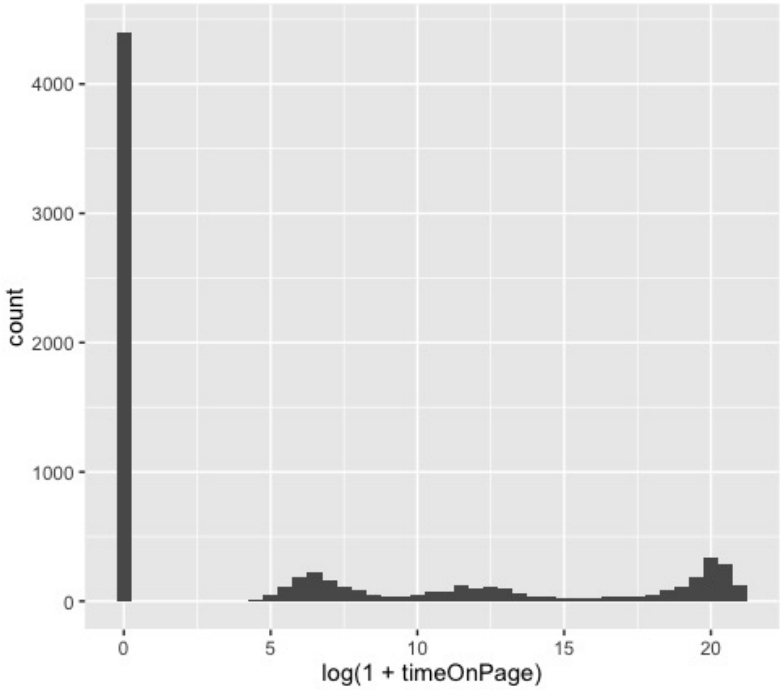
Filter

geo_location	traffic_source	display_id	timeOnPage	weekDay	monthDay	hour	min	sec	topic_id	topic_confidence_level	source_id	publisher_id	publish_time	category_id
US>SC>546	internal	9209762	607989372	3	14	20	18	44	184	0.05795561	3931	468		1907
US>FL>539	search	874239	574	1	19	3	54	36	292	0.05941360	9727	852	2015-07-23 15:00:00	1606
US>KS>616	search	16996700	35829	7	18	5	23	10	102	0.07694444	3528	170	2012-12-14 00:00:00	1100
US>OH>510	internal	17281189	115144320	5	16	6	57	51	184	0.05795561	3931	468		1907
US>IN>527	internal	299887	1982	5	16	0	28	43	20	0.14582518	8794	167	2016-04-25 15:00:00	1610
US>TX>623	search	9784520	632692368	4	15	16	24	26	200	0.07110867	10517	240	2011-08-04 10:00:00	1205
US>NC>560	internal	281561	786	4	15	21	33	0	172	0.01791358	632	324	2016-06-14 00:00:00	1907
CA>AB	search	47516	450	2	13	3	11	13	20	0.16414659	10888	637	2015-01-16 00:00:00	1610
US>VA>511	internal	14115850	864791437	1	19	23	25	33	107	0.04996685	3221	265	2015-03-26 10:00:00	1408
US>MD>511	internal	380676	57540	5	16	12	14	8	43	0.03313998	8610	1142	2016-06-02 14:00:00	1407
US>AR>612	internal	8958278	598592642	2	13	0	3	27	229	0.06322276	4218	468		1403
US>NC>560	internal	350981	256	5	16	8	15	50	249	0.08719769	842	38	2010-01-27 00:00:00	2100
US>KY>529	social	1094324	132109	2	20	8	22	2	143	0.03463938	1209	875		1702
ZW	internal	139986	10658	3	14	7	11	31	229	0.22538685	6049	714	2016-06-14 01:00:00	1912
US>NJ>501	internal	552873	303	6	17	11	35	13	200	0.07110867	10517	240	2011-08-04 10:00:00	1205
US>FL>539	social	988611	1226128	1	19	14	47	23	8	0.09863138	7267	58	2016-06-14 00:00:00	1403
US>FL>561	internal	521942	536	6	17	7	19	54	184	0.14782937	1627	714	2016-06-14 08:00:00	1902
US>GA>524	internal	7986652	519868528	5	16	14	47	50	184	0.05795561	3931	468		1907
US>IN>527	internal	11729256	775048250	1	12	20	42	57	103	0.05045177	722	925		1902
US>NY>501	internal	528350	464	6	17	8	12	33	184	0.23916160	6045	714	2016-06-13 10:00:00	1907
US>IN>509	internal	458568	468	5	16	22	49	5	292	0.08849117	6862	637	2015-09-29 00:00:00	1513
US>NC>517	internal	1721332	41180626	3	21	6	48	18	36	0.16733635	1051	435	2016-06-14 18:00:00	1702
US>NY>501	internal	1432665	1165638	4	22	11	24	29	16	0.02119969	8610	1142	2016-04-28 15:00:00	1403
US>OH>510	internal	18075895	442913412	4	15	15	23	35	184	0.05795561	3931	468		1907
US>NY>501	internal	510010	846966	6	17	3	21	38	8	0.16555751	109	84	2016-06-13 22:00:00	1702
US>FL>530	internal	1319040	52458772	4	15	18	36	54	184	0.05795561	3931	468		1907
US>FL>534	social	17007482	363652	7	18	11	36	50	8	0.07744688	1209	875	2016-06-13 11:00:00	1403
CA>ON	internal	986106	390	1	19	17	52	54	269	0.04430349	6872	41	2016-06-14 00:00:00	1706
US>CA>807	internal	161924	499	3	14	15	33	5	160	0.13740678	5792	704	2015-10-18 00:00:00	1805
US>PA>504	internal	3799881	225399396	3	14	15	43	6	184	0.05795561	3931	468		1907
US>CA>803	internal	11242724	696016651	7	18	17	55	23	200	0.07110867	10517	240	2011-08-04 10:00:00	1205
US>PA>504	search	587438	2834	6	17	16	10	36	216	0.06779961	5760	433	2016-06-14 12:00:00	1702

Showing 1 to 32 of 3,189 entries

Data is in anonymized format, hence every variable is categorical, with no deeper understanding possible through knowledge of the values of the covariates.

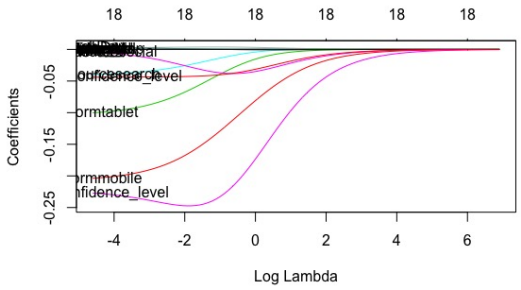
5.2 timeOnPage covariate is mostly 0s

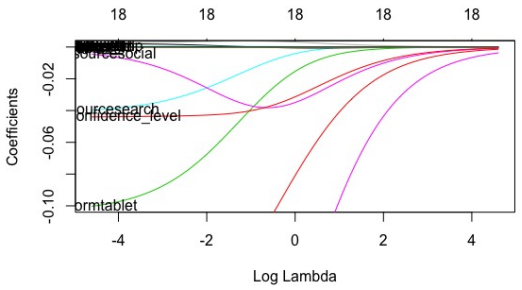


A log plot has been used for clarity. Most (58%) of the values were 0s

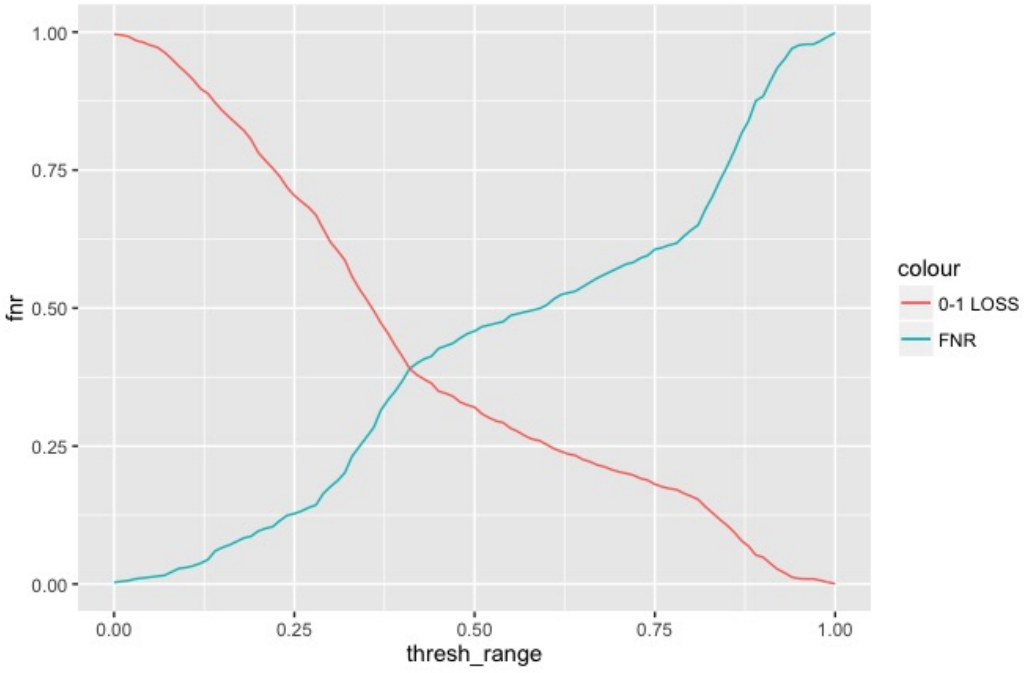
5.3 Model selection - Continuous Response Variable Stage I

5.3.1 Ridge regression coefficients vs lambda (threshold)

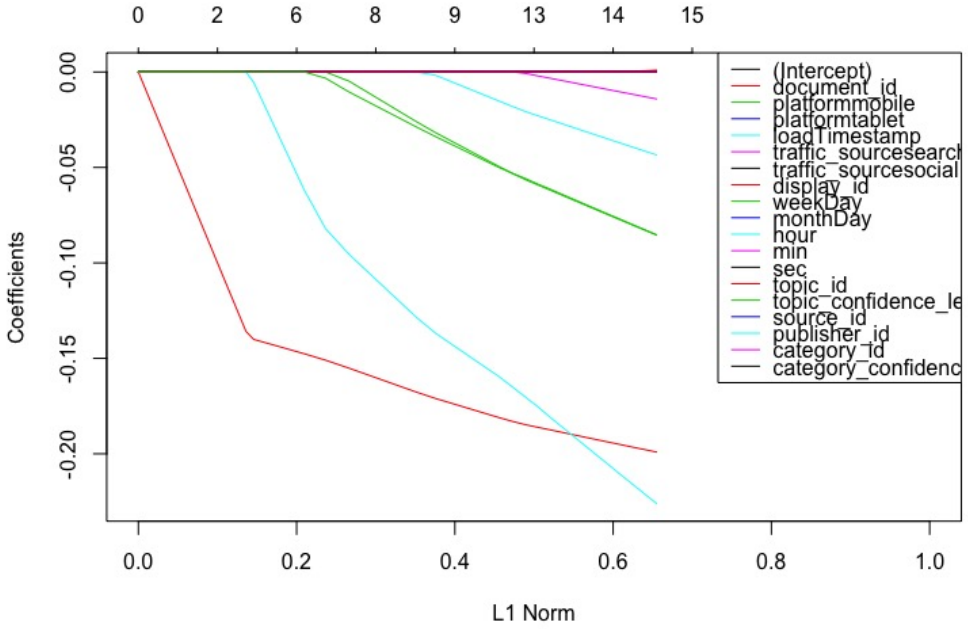




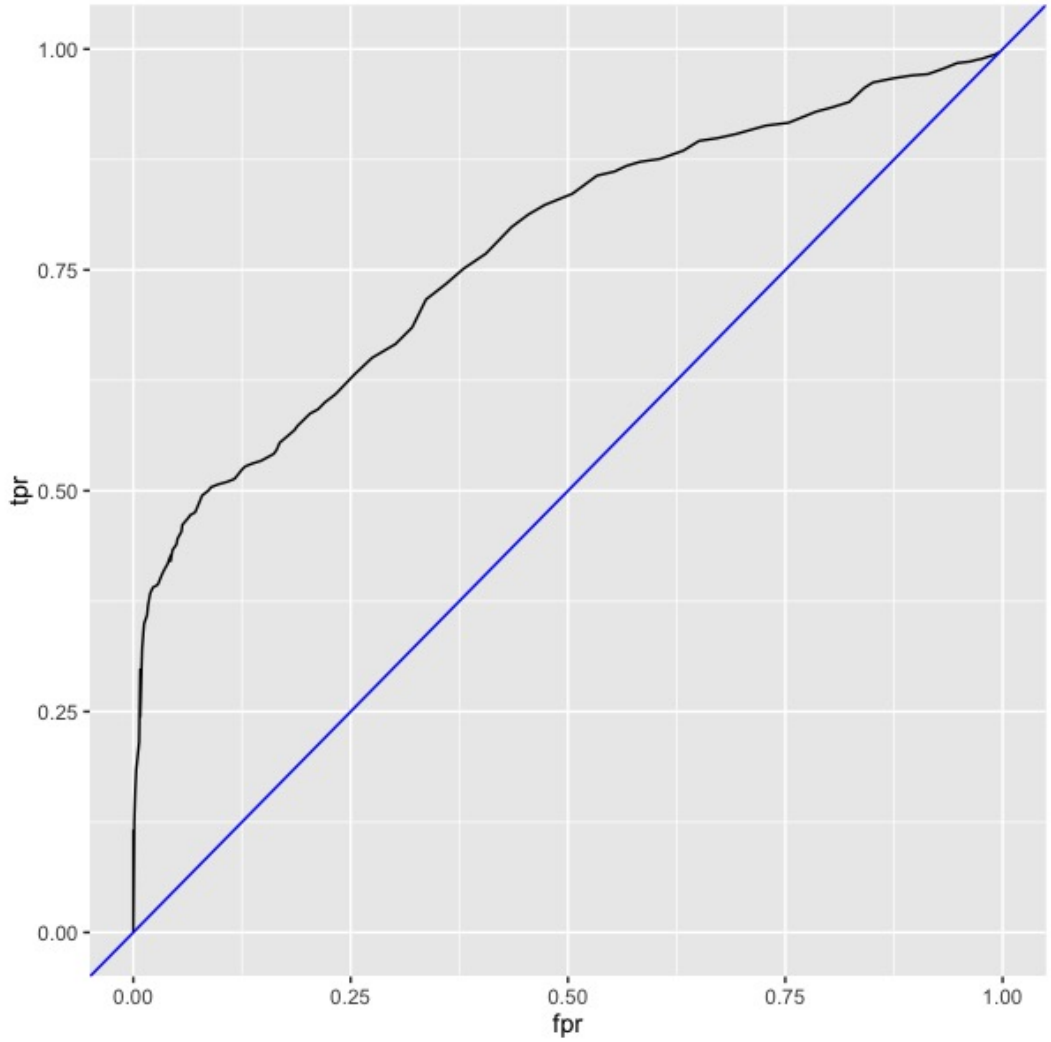
5.3.2 FNR and 0-1 loss with varying the lambda, to select the best value of threshold for the selected model



5.3.3 Lasso coefficients plot



5.3.4 ROC curve for the selected model

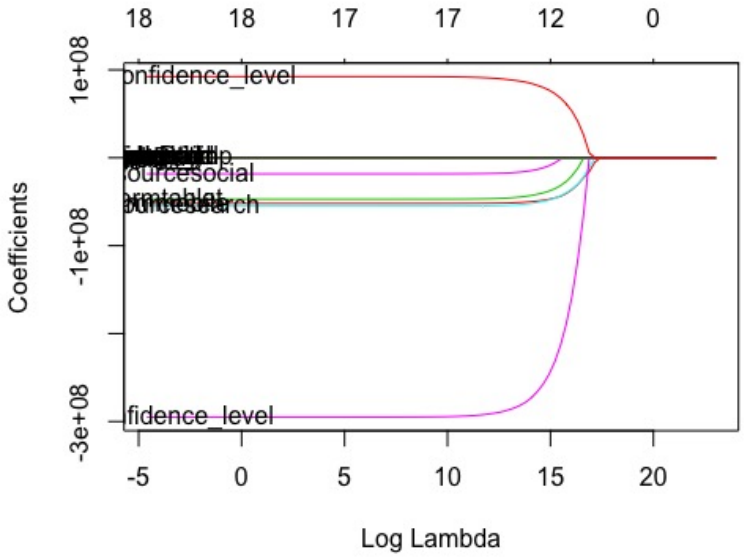


5.3.5 Model Training and Cross-validation Error

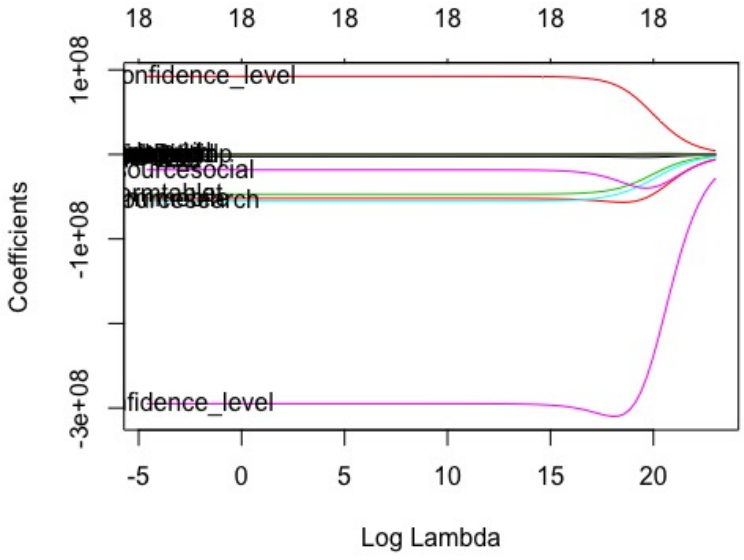
Covariates Used	Train Error	CV Error
document_id + platform + display_id	0.2600395	0.2789474
document_id + platform + display_id + document_id:display_id	0.2608624	0.2789474
document_id + platform + display_id + topic_id + document_id:display_id	0.2613562	0.2710526
document_id + platform + display_id + topic_id + publisher_id + document_id:display_id	0.2628374	0.2763158
document_id + platform + display_id + topic_id + publisher_id + document_id:display_id + document_id:topic_id	0.2583937	0.2690789
document_id + platform + display_id + topic_id + publisher_id + topic_confidence_level + category_confidence_level + document_id:display_id + document_id:topic_id + platform:topic_confidence_level + document_id:topic_confidence_level + document_id:category_confidence_level	0.2605332	0.275
.	0.2577354	0.2690789
document_id + platform + display_id + topic_id + publisher_id + topic_confidence_level + category_confidence_level + document_id:display_id + document_id:topic_id + platform:topic_confidence_level + document_id:topic_confidence_level + document_id:category_confidence_level + platform:category_confidence_level + publisher_id:category_confidence_level + publisher_id:topic_confidence_level + topic_id:category_confidence_level + topic_id:publisher_id	0.2529625	0.2782895
document_id + platform + display_id + topic_id + publisher_id + topic_confidence_level + monthDay + traffic_source	0.2621791	0.2855263
document_id + platform + display_id + topic_id + publisher_id + topic_confidence_level	0.2435813	0.2697368
document_id + platform + display_id + topic_id + publisher_id	0.2435813	0.2697368

5.4 Model selection - Continuous Response Variable Stage II

5.4.1 Lasso coefficients



5.4.2 Ridge coefficients

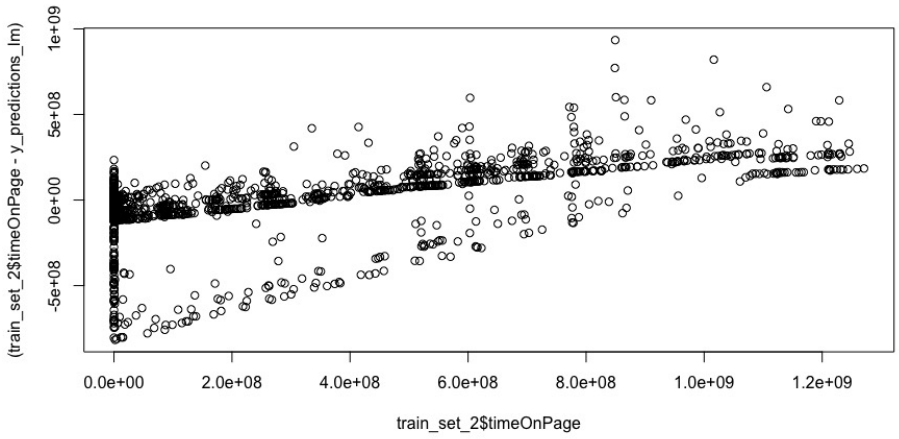


5.4.3 Model Training and Cross-validation Error (RMSE)

Note: Please see the corresponding models in the R-code corresponding to this section, lines 231-269 (since the models are too large to type in this table)

Model	Train Error	CV Error
m1	169265614	171507376
m2	169871320	170766770
m3	171056729	171757336
m4	171908600	171793614
m5	172307759	172727073
m6	172615740	173380438
m7	173812344	175220985
m8	176505170	178996538
m9	177422278	180248952
m10	208943551	208271940

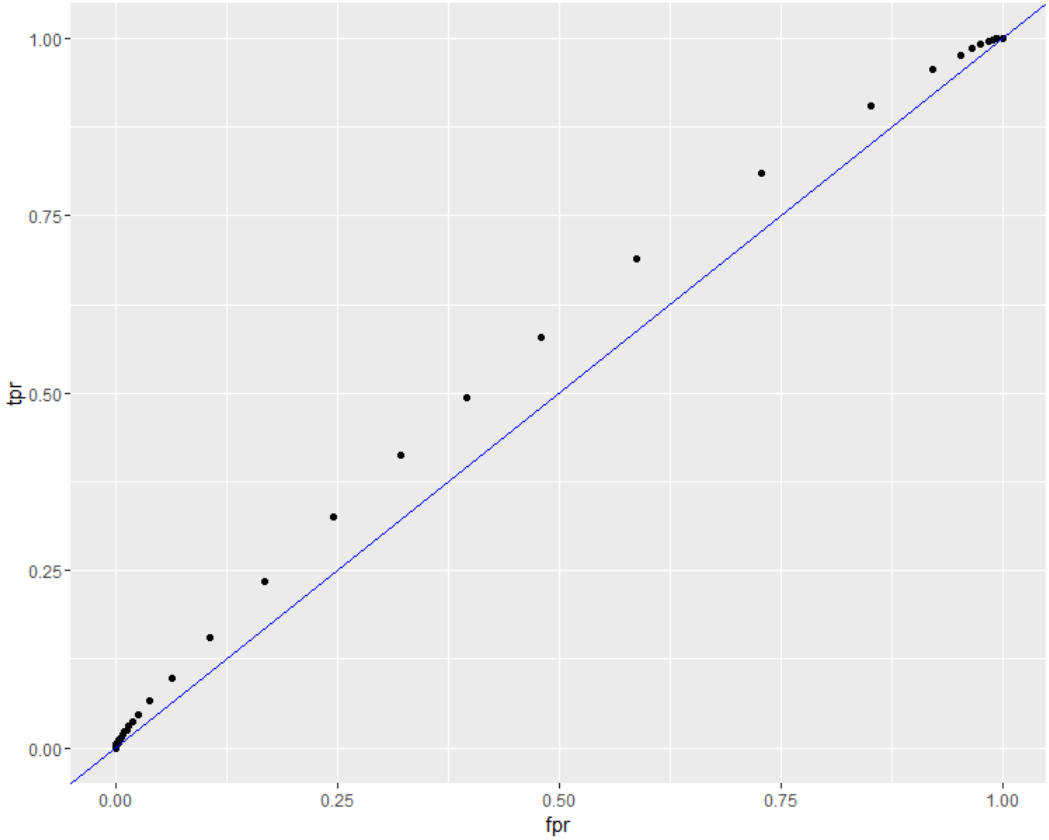
5.4.4 Residuals Plot - Continuous Response Variable



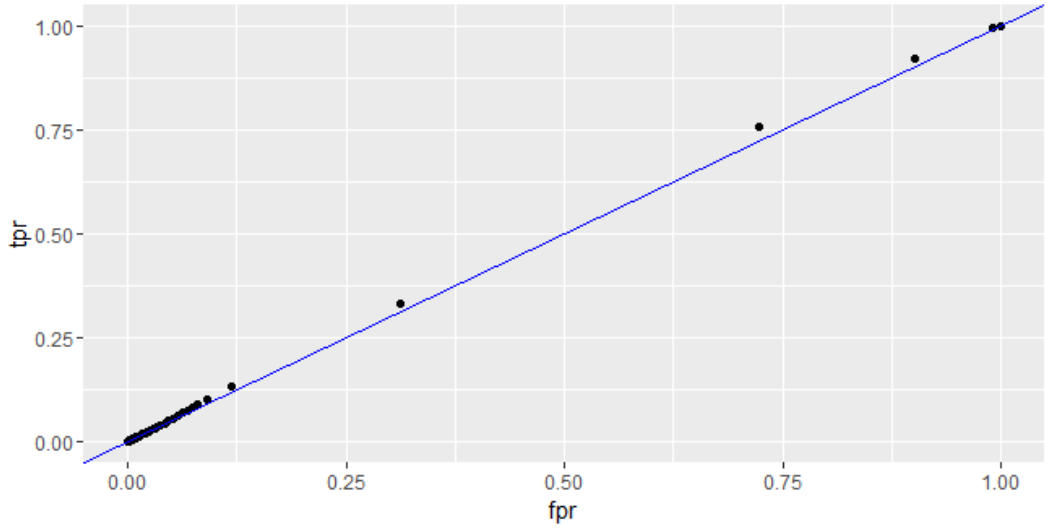
5.5 Model selection - Binary Response Variable

5.5.1 Model Class Selection

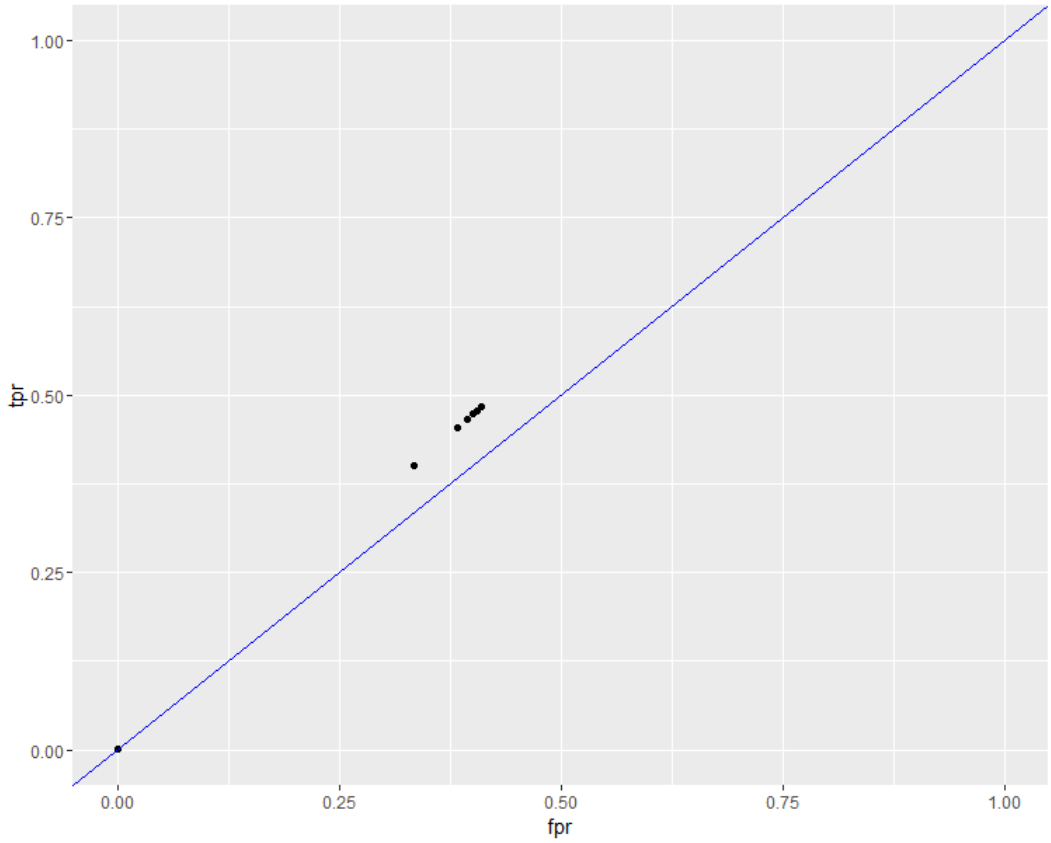
Logistic Regression ROC Plot:



Naive Bayes Classifier ROC Plot:



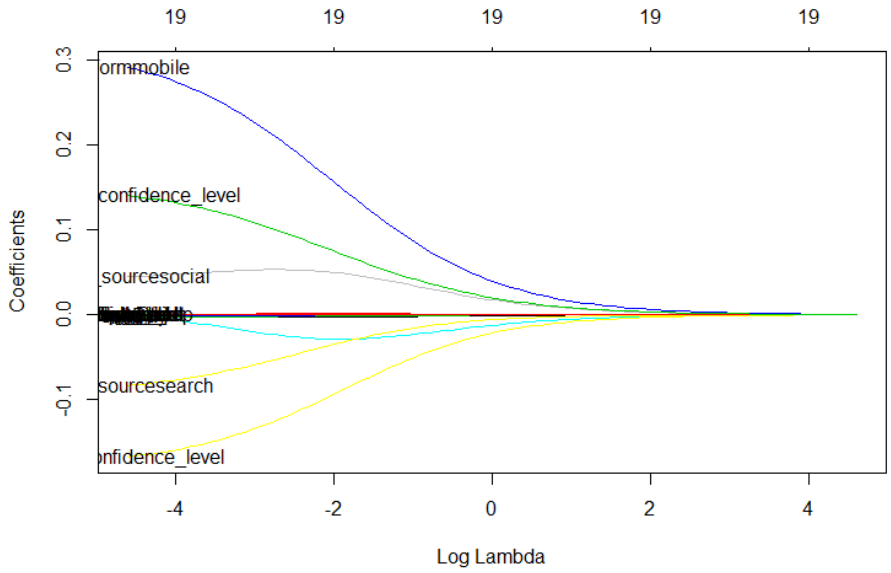
Support Vector Machines ROC Plot:



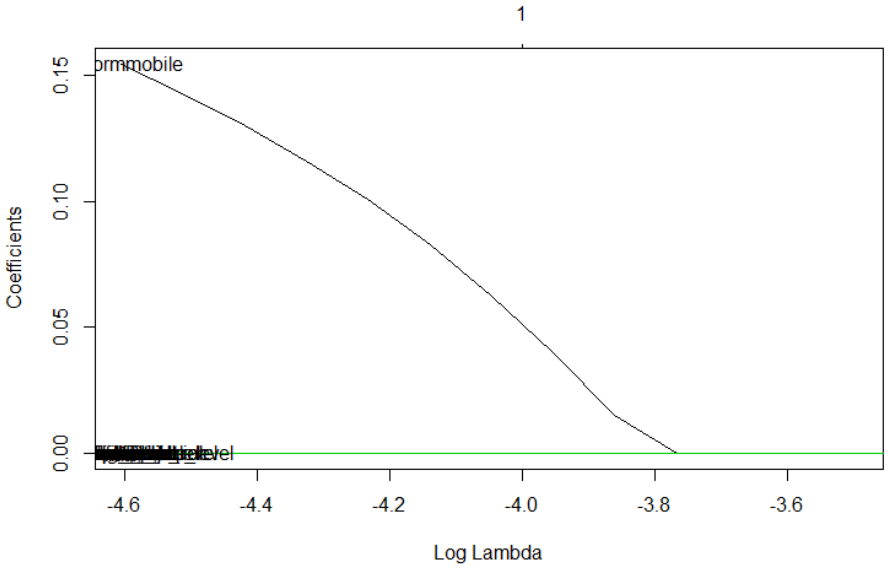
The Support Vector Machine ROC plot has very few points because training an SVM takes a very long time.

5.5.2 Covariate Selection

Ridge Regularization Lambda Plot:

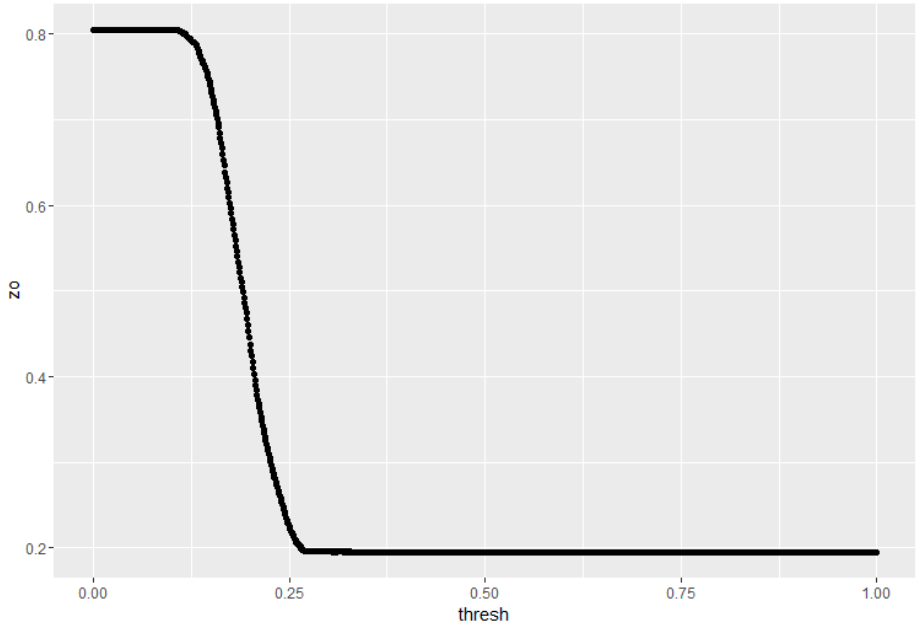


Lasso Regularization Lambda Plot:

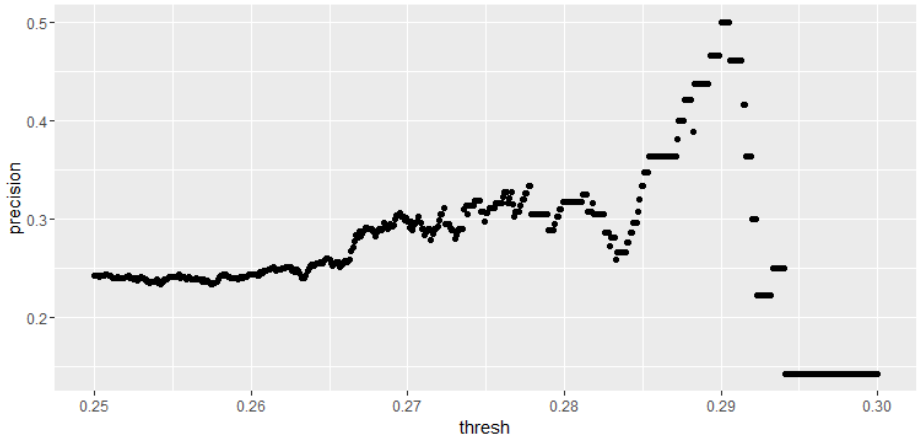


5.5.3 Threshold Selection

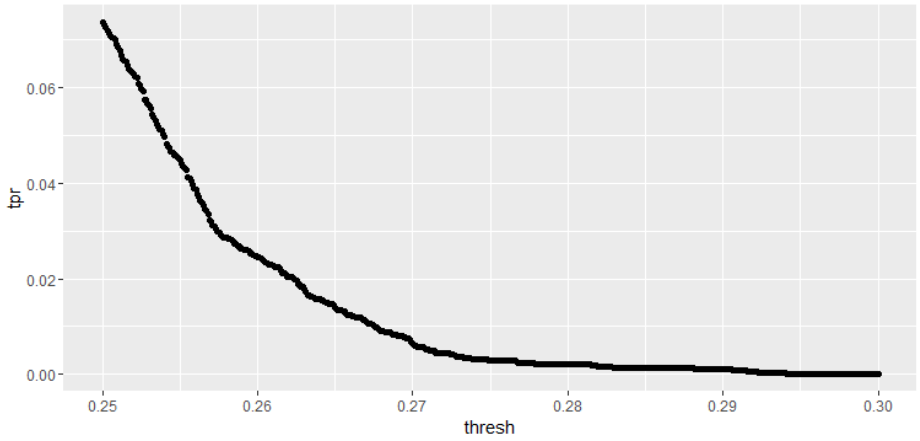
Zero-One Loss on Training Set



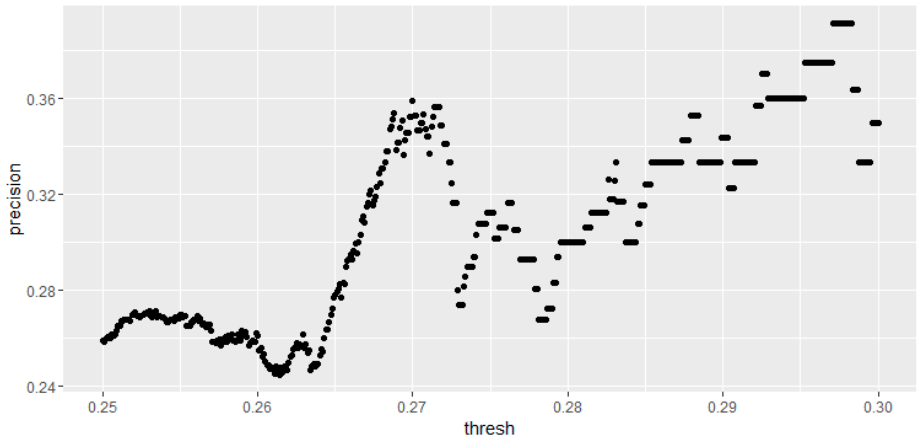
Precision vs. Threshold on Training Set



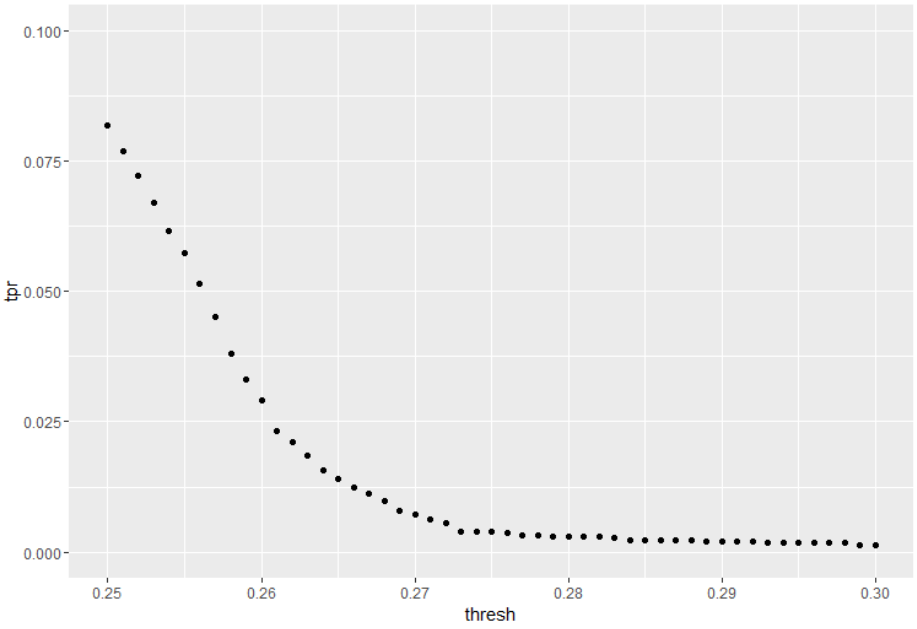
Sensitivity vs. Threshold on Training Set



Precision vs. Threshold on Hold-Out Set



Sensitivity vs. Threshold on Hold-Out Set


0.100
0.075
0.050
0.025
0.000
tpr
0.25
0.26
0.27
0.28
0.29
0.30
thresh

6 Code Snippets in R

6.1 Code in R for generating the smaller datasets (seperate for Binary and Continous response variable)

```
# Unifies data with the other tables, then segments into training and
   test set
library(data.table)
library(ggplot2)
library(dplyr)

# Set working directory to the one where your data (csv files) are
   stored

#================= Continuous Response Variable
   ========================
page_views <- fread("page_views_sample.csv")
events <- fread("events.csv")
colnames(page_views)[[3]] <- "loadTimestamp"
colnames(events)[[4]] <- "eventTimestamp"
events$platform <- as.integer(events$platform)
page_views$platform <- as.integer(page_views$platform)

page_events <- merge(page_views, events, by = c("uuid", "document_id",
   "platform"))
# add new covariate which is the time on page (difference between
   timestamps)
page_events <- mutate(page_events, timeOnPage = eventTimestamp -
   loadTimestamp)
page_events$traffic_source <- as.integer(page_events$traffic_source)

# remove loaded tables as we don't need them (memory save)
rm(page_views)
rm(events)

# convert integer categories to string for ease of interpretation as
   well as avoiding errors later
convert_platform <- function(x) {
  if(x == 1) return(as.factor("desktop"))
  if(x == 2) return(as.factor("mobile"))
  if(x == 3) return(as.factor("tablet"))
}

convert_traffic <- function(x) {
  if(x == 1) return(as.factor("internal"))
  if(x == 2) return(as.factor("search"))
  if(x == 3) return(as.factor("social"))
}

page_events$platform <- sapply(page_events$platform, convert_platform)
```

```
page_events$traffic_source <- sapply(page_events$traffic_source,
    convert_traffic)

# Sample only 10,000 rows from original training data
set.seed(129)
n = 10000 # rowcount
samp.ind = sample(nrow(page_events), n) # sample n row indices at
    random
page_events.samp = page_events[samp.ind,]
rm(page_events)

# Clean data
# remove geo_location.y as it is a repeat
# remove eventTimeStamp as it is an Oracle (it's the same as our
    continuous response in essence)
formatTime <- function(epoch) {
  as.POSIXlt(as.POSIXct(round(epoch/100), origin="2016-06-13"))
}
page_events.samp <- mutate(page_events.samp,
                    weekDay = wday(formatTime(loadTimestamp)),
                    monthDay = mday(formatTime(loadTimestamp)),
                    hour = hour(formatTime(loadTimestamp)),
                    min = formatTime(loadTimestamp)$min,
                    sec = formatTime(loadTimestamp)$sec)
page_events.samp <- select(page_events.samp, -c(geo_location.y,
    eventTimestamp))
page_events.samp <- rename(page_events.samp, geo_location =
    geo_location.x)

# Unify those rows with the other relational data tables
#!!! There are multiple topics per document, this selects only the
    topic given the most confidence
doc_topics <- fread("documents_topics.csv") %>%
  rename(topic_confidence_level = confidence_level) %>%
  group_by(document_id) %>%
  mutate(r = min_rank(desc(topic_confidence_level))) %>%
  filter(r == 1) %>%
  select(-c(r))

#!!! Two categories per document, this selects only the category given
    the most confidence
doc_categories <- fread("documents_categories.csv") %>%
  rename(category_confidence_level = confidence_level) %>%
  group_by(document_id) %>%
  mutate(r = min_rank(desc(category_confidence_level))) %>%
  filter(r == 1) %>%
  select(-c(r))

doc_meta <- fread("documents_meta.csv")
```

```
page_events.samp <- merge(page_events.samp, doc_topics, by =
   c("document_id"), all.x = TRUE)
page_events.samp <- merge(page_events.samp, doc_meta, by =
   c("document_id"), all.x = TRUE)
page_events.samp <- merge(page_events.samp, doc_categories, by =
   c("document_id"), all.x = TRUE)

page_events.samp <- page_events.samp[complete.cases(page_events.samp),]

# write the file to disk, crv == continuous reponse variable
write.csv(page_events.samp, file = "crv.csv")

# remove unnecessary files, free memory
rm(doc_meta)
rm(doc_topics)
rm(doc_categories)
rm(page_events.samp)

#================== Discrete Response Variable ========================
clicks <- fread("clicks_train.csv")
page_events.samp <- fread("crv.csv")

training_clicks <- clicks
clicked_on <- filter(training_clicks, training_clicks$clicked == 1)
   %>% select(display_id)

#Sample 10000 random rows from original training data
set.seed(675)
samp.ind = sample(nrow(clicked_on), n)
clicked_on = clicked_on[samp.ind,]

training_clicks.samp <- merge(clicked_on, training_clicks, by =
   c("display_id"))
# training_clicks.samp <- merge(training_clicks.samp,
   promoted_content, by = c("ad_id"), all.x = TRUE)

#Remove a variable that we wouldnt have access to in the real world
page_events.samp <- select(page_events.samp, -c(timeOnPage))

#Merge brv with our existing crv data
training_clicks.samp <- merge(training_clicks.samp, page_events.samp,
   by = c("display_id"))
training_clicks.samp <- select(training_clicks.samp, -c(V1, uuid))

write.csv(training_clicks.samp, file = "brv.csv")

# remove unused
rm(clicked_on)
rm(training_clicks.samp)
rm(page_events.samp)
```

6.2 Code in R for Modeling BRV

```
library(data.table)
library(ggplot2)
library(dplyr)
library(MLmetrics)
library(cvTools)
library(glmnet)

# FUNCTION DEFINITIONS
# ----------------------------------------
# based on a threshold, decide which elements to call '0' and '1'
makePrediction = function(x, thresh = 0.5) {
  # @param x: This is the value/probability that your predictor outputs
  # @param thresh: the threshold that decides whether you classify as 1
     or 0
  # Outputs 1 or 0 depending on whether value is more than thresh or not
  # make a prediction on the probability of y, return 1 if greater than
     thresh, 0 otherwise
  if (x >= thresh)
    return(1)
  else
    return(0)
}

#
   http://stackoverflow.com/questions/30566788/legend-label-errors-with-glmnet-plo
# prints the legend in the coefficients plot (for lasso and ridge)
lbs_fun <- function(legend_location, fit, ...) {
  L <- length(fit$lambda)
  x <- log(fit$lambda[L])
  y <- fit$beta[, L]
  labs <- names(y)
  text(x, y, labels=labs, ...)
  # legend(legend_location, legend=labs, col=1:6, lty=1) # <- labels
     are hard to see, hence commented out
}

# Now will find the ROC by changing thresholds
findCM = function(y_true, y, thresh) {
  # apply this function to the predicted values, and return a numeric
     vector
  y_observed <- as.numeric(lapply(y_pred, function(x)
     {makePrediction(x, thresh = thresh)}))

  # return confusion matrix
  cm <- ConfusionMatrix(y_observed, y_true)
  # print(cm)
  return(cm)
}
# ----------------------------------------
```

```
# Manually set input directory
crv.master <- fread("crv_train.csv")
crv.master <- select(crv.master, -V1) # remove the index column which
   automatically gets added

# Two step model will be used:
# 1. Does person spend time > 0 on page
# 2. How much time does person spend on page

# preprocessing step: add new column which is a binarized version (1
   if nonzero) of time on page
crv.master <- mutate(crv.master, notBounced = ifelse(timeOnPage > 0,
   1, 0))

# PART 1: FINDING THE BEST MODEL FOR PREDICTING WHETHER PEOPLE STAY ON
   PAGE OR NOT
# TODO: Running lasso/ridge
# the left out covariates take the longest time (-publish_time,-uuid,
   geo_location)
# timeOnPage is removed because it would be cheating
X <- select(crv.master, -uuid, -publish_time, -geo_location,
   -timeOnPage) # this is our actual p+1 covariates set

# Divide into training and CV datasets
samp.ind = sample(nrow(X), 0.8*nrow(X)) # sample n row indices at
   random
train_set = X[samp.ind,]
test_set = X[-samp.ind,]

# Choose covariates through a variety of methods
# A. Forward stepwise selection - on logistic regression
# min.model <- lm(notBounced ~ 1, data=train_set)
# biggest <- formula(lm(notBounced ~ .:., train_set)) # first and
   second order terms
min.model <- glm(notBounced ~ 1, data = train_set)
biggest <- formula(glm(notBounced ~ .:., train_set, family =
   binomial()))
fwd.model = step(min.model, direction='forward', scope=biggest)
summary(fwd.model)

# B. Backward stepwise selection
lmfull1 <- lm(notBounced ~ ., data=train_set)
lmfull2 <- lm(notBounced ~ .:., data=train_set)
bkwd.model = step(lmfull2, direction='backward')

# C. Identification of best covariates using regsubsets
library(leaps)
allsubsets <- regsubsets(notBounced ~ ., data=train_set, nbest=10)

# D. Regression with regularization
```

```
X <- model.matrix(notBounced ~ ., train_set)
y <- train_set$notBounced
grid=10^seq(2, -2, length = 100) # choosing lambda in the range of -2
   to 10

# ridge
ridge.mod = glmnet (X, y, alpha=0, lambda=grid) # alpha=0, hence ridge
   model is fit
plot(ridge.mod, xvar="lambda", col=1:dim(coef(ridge.mod))[1], ylim =
   c(-0.3, 0)) # Get the plot of coefficients w.r.t. lambda
lbs_fun('topright', ridge.mod)

# lasso
lasso.mod = glmnet (X, y, alpha=1) # alpha=1, hence lasso
plot(lasso.mod, ylim = c(-0.3, 0)) # Get the plot of coefficients
   w.r.t. lambda
lbs_fun('topright', lasso.mod)

# Find training error and CV error for the top 10 most promising models
# create list of models, iterate over them
# manually enter preferred models
m3 <- notBounced ~ document_id + platform + display_id
m4 <- notBounced ~ document_id + platform + display_id +
   document_id:display_id
m5 <- notBounced ~ document_id + platform + display_id + topic_id +
  document_id:display_id
m6 <- notBounced ~ document_id + platform + display_id + topic_id +
  publisher_id + document_id:display_id
m7 <- notBounced ~ document_id + platform + display_id + topic_id +
  publisher_id + document_id:display_id + document_id:topic_id
m8 <- notBounced ~ document_id + platform + display_id + topic_id +
  publisher_id + topic_confidence_level + category_confidence_level +
  document_id:display_id + document_id:topic_id +
     platform:topic_confidence_level +
  document_id:topic_confidence_level +
     document_id:category_confidence_level
m9 <- notBounced ~ .
m10 <- notBounced ~ document_id + platform + display_id + topic_id +
  publisher_id + topic_confidence_level + category_confidence_level +
  document_id:display_id + document_id:topic_id +
     platform:topic_confidence_level +
  document_id:topic_confidence_level +
     document_id:category_confidence_level +
  platform:category_confidence_level +
     publisher_id:category_confidence_level +
  publisher_id:topic_confidence_level +
     topic_id:category_confidence_level +
  topic_id:publisher_id
m1 <- notBounced ~ document_id + platform + display_id + topic_id +
  publisher_id + topic_confidence_level + monthDay + traffic_source
m2 <- notBounced ~ document_id + platform + display_id + topic_id +
```

```
  publisher_id + topic_confidence_level
m3 <-notBounced ~ document_id + platform + display_id + topic_id +
  publisher_id
models_list <- list(m1, m2, m3, m4, m5, m6, m7, m8, m9, m10)

y_true <- train_set$notBounced
j = 0
for (m in models_list) {
  # iterate over all models, train
  binary_timeOnPage <- glm(m, data = train_set)
  y_pred_train <- predict(binary_timeOnPage, train_set, type =
     "response") # these are probabilities

  # Find training error and testing error for each of these models
  # Convert to 0 and 1 based on threshold level 0.5 (as dataframe, then
     convert back into numeric vector)
  Y_train <- select(mutate(as.data.frame(y_pred_train), predicted =
     ifelse(y_pred_train > 0.5, 1, 0)), predicted)
  training_error <- sum(abs(Y_train -
     train_set$notBounced))/nrow(train_set)
  cat('model', j, 'training error:', training_error, '\n')

  y_pred_test <- predict(binary_timeOnPage, test_set, type = "response")
  Y_test <- select(mutate(as.data.frame(y_pred_test), predicted =
     ifelse(y_pred_test > 0.5, 1, 0)), predicted)
  testing_error <- sum(abs(Y_test - test_set$notBounced))/nrow(test_set)
  cat('model', j, 'testing error:', testing_error, '\n')

  # increment j
  j = j + 1
}

# Now find the plot of False negative rate vs. lambda
thresh_range <- seq(0,1,0.01)
binary_timeOnPage <- glm(m9, data = train_set) # best model
y_pred <- predict(binary_timeOnPage, test_set, type = "response") #
   predicted response
y_truth <- as.numeric(test_set$notBounced)

# empty variables which will store values in for loop
fnr <- rep(0, length(thresh_range))
tpr <- rep(0, length(thresh_range))
fpr <- rep(0, length(thresh_range))
y_zeroOne_bestModel <- rep(0, length(thresh_range))

for (i in 1:length(thresh_range)) {
  # find FNR and 0-1 loss on test set only
  y_zeroOne_bestModel[i] <- mean(as.numeric(lapply(y_pred, function(x)
     {makePrediction(x, thresh = thresh_range[i])})))
  cm <- findCM(y = y_pred, y_true = y_truth, thresh = thresh_range[i])
  # extract values of confusion matrix
```

```
  tn = cm[1]
  fn = cm[2]
  fp = cm[3]
  tp = cm[4]

  # find the TPR and FPR
  fnr[i] <- fn/(tp + fn)

  # Also find the Precision and Recall, for the ROC curve
  tpr[i] <- tp/(tp + fn)
  fpr[i] <- fp/(fp + tn)
}
# plot the FPR vs lambda curve and the 0-1 loss vs lambda curve
ggplot() + geom_line(aes(x=thresh_range, y=fnr, color = 'FNR')) +
    geom_line(aes(x=thresh_range, y=y_zeroOne_bestModel, color='0-1
    LOSS')) + xlim(0,1) + ylim(0,1)

# plot the ROC curve for the selected model
ggplot() + geom_line(aes(x=fpr, y=tpr)) + xlim(0,1) + ylim(0,1) +
    geom_abline(slope = 1, intercept = 0, color = 'blue')

# PART 2: CONTINUOUS RESPONSE VARIABLE
# linear regression on the same matrix for those values for which the
    time on page != 0
# Given that time on page != 0, what is the time on page?
# Use the 'Oracle' of part 1 as the input for training
timePage.train <- filter(crv.master, notBounced == 1)
timePage.train <- select(timePage.train, -notBounced) # get rid of the
    variable as it is not being used
# y_true <- as.numeric(timePage.train$timeOnPage)

# train linear regression models on it
X_linear <- select(timePage.train, -uuid, -publish_time,
    -geo_location) # the left out covariates take the longest time
    (-publish_time,-uuid, geo_location)
# fit <- glmnet(X_linear, y_true, family="gaussian", alpha=0,
    lambda=0.001)
# lm_ols <- lm(formula = timeOnPage ~ ., data = X_linear)
# print(summary(lm1))
# lm_ridge <- lm.ridge(formula = timeOnPage ~ ., data = X_linear,
    lambda = 0.5)
# Using glmnet from
    http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf
# create new X and y for training
# Divide into training and CV datasets
samp.ind = sample(nrow(X), 0.8*nrow(X)) # sample n row indices at
    random
train_set_2 = X_linear[samp.ind,]
test_set_2 = X_linear[-samp.ind,]

X <- model.matrix(timeOnPage ~ ., train_set_2)
```

```
y <- train_set_2$timeOnPage

# A. Find ridge coefficients
grid=10^seq(10,-2, length = 100) # choosing lambda in the range of -2
   to 10
ridge.mod = glmnet (X, y, alpha=0, lambda=grid) # alpha=0, hence ridge
   model is fit
plot(ridge.mod, xvar="lambda", col=1:dim(coef(ridge.mod))[1]) # Get
   the plot of coefficients w.r.t. lambda
lbs_fun('topright', ridge.mod)

# B. Find ridge coefficients
grid=10^seq(10,-2, length = 100)
lasso.mod = glmnet (X, y, alpha=1, lambda=grid) # alpha=1, hence lasso
   model is fit
plot(lasso.mod, xvar="lambda", col=1:dim(coef(lasso.mod))[1]) # Get the
lbs_fun('topright', lasso.mod)

# C. Forward stepwise regression
min.model <- lm(timeOnPage ~ 1, data = train_set_2)
biggest <- formula(lm(timeOnPage ~ .:., train_set_2))
fwd.model = step(min.model, direction='forward', scope=biggest)
summary(fwd.model)

# Models selected based on these three methods and insights
m1 <- timeOnPage ~ display_id + document_id + traffic_source +
   category_confidence_level +
  platform + loadTimestamp + topic_id + topic_confidence_level +
  display_id:document_id + display_id:traffic_source +
     document_id:platform +
  display_id:platform + display_id:category_confidence_level +
  traffic_source:category_confidence_level + document_id:traffic_source
     +
  document_id:loadTimestamp + document_id:category_confidence_level +
  platform:loadTimestamp + display_id:topic_id + document_id:topic_id +
  document_id:topic_confidence_level + display_id:topic_confidence_level
m2 <- timeOnPage ~ display_id + document_id + traffic_source +
   category_confidence_level +
  platform + loadTimestamp + topic_id + topic_confidence_level +
  display_id:document_id + display_id:traffic_source +
     document_id:platform +
  display_id:platform + display_id:category_confidence_level +
  traffic_source:category_confidence_level + document_id:traffic_source
     +
  document_id:loadTimestamp + document_id:category_confidence_level +
  platform:loadTimestamp + display_id:topic_id + document_id:topic_id +
  document_id:topic_confidence_level
m3 <- timeOnPage ~ display_id + document_id + traffic_source +
   category_confidence_level +
  platform + loadTimestamp + display_id:document_id +
     display_id:traffic_source +
```

```
  document_id:platform + display_id:platform +
     display_id:category_confidence_level +
  traffic_source:category_confidence_level + document_id:traffic_source
     +
  document_id:loadTimestamp + document_id:category_confidence_level
m4 <- timeOnPage ~ display_id + document_id + traffic_source +
    category_confidence_level +
  platform + display_id:document_id + display_id:traffic_source +
  document_id:platform + display_id:platform +
     display_id:category_confidence_level +
  traffic_source:category_confidence_level + document_id:traffic_source
m5 <- timeOnPage ~ display_id + document_id + traffic_source +
    category_confidence_level +
  platform + display_id:document_id + display_id:traffic_source +
  document_id:platform + display_id:platform +
     display_id:category_confidence_level +
  traffic_source:category_confidence_level
m6 <- timeOnPage ~ display_id + document_id + traffic_source +
    category_confidence_level +
  platform + display_id:document_id + display_id:traffic_source +
  document_id:platform + display_id:platform +
     display_id:category_confidence_level
m7 <- timeOnPage ~ display_id + document_id + traffic_source +
    category_confidence_level +
  platform + display_id:document_id + display_id:traffic_source +
  document_id:platform
m8 <- timeOnPage ~ display_id + document_id + traffic_source +
    display_id:document_id +
  display_id:traffic_source
m9 <- timeOnPage ~ display_id + document_id + traffic_source +
    display_id:document_id
m10 <- timeOnPage ~ .

# Make a list of these models
models_list <- list(m1, m2, m3, m4, m5, m6, m7, m8, m9, m10)

y_true <- train_set_2$timeOnPage

j = 0
for (m in models_list) {
  # iterate over all models, train
  lm_timeOnPage <- lm(m, data = train_set_2)

  # Find training error and testing error for each of these models
  # Convert to 0 and 1 based on threshold level 0.5 (as dataframe, then
     convert back into numeric vector)
  y_pred_train <- predict(lm_timeOnPage, train_set_2, type =
     "response") # these are probabilities
  training_error <- sqrt(mean((y_pred_train -
     train_set_2$timeOnPage)^2))
  cat('model', j, 'training error:', training_error, '\n')
```

```
  y_pred_test <- predict(lm_timeOnPage, test_set_2, type = "response")
  testing_error <- sqrt(mean((y_pred_test - test_set_2$timeOnPage)^2))
  cat('model', j, 'testing error:', testing_error, '\n')

  # increment j
  j = j + 1
}

# Plot residuals of the best model
best_model <- lm(m2, data = train_set_2) # this is manually put!
y_predictions_lm <- predict(best_model, train_set_2, type="response")

# Plot residuals
plot(train_set_2$timeOnPage, (train_set_2$timeOnPage -
   y_predictions_lm))
```

6.3 Code in R for Modeling BRV

```
library(data.table)
library(ggplot2)
library(dplyr)
library(e1071)
library(cvTools)
library(MLmetrics)
library(leaps)
library(glmnet)

#input_directory <- ".\\input"
#setwd(input_directory)

#----------- Get the training data ------------
brv <- fread("brv.csv")
brv <- select(brv, -c(V1))
brv$clicked <- as.factor(brv$clicked)

#split data
set.seed(851)
train.ind = sample(nrow(brv), 4*round(nrow(brv)/5)) #80% of data is
   for training, 20% for test
brv.train = brv[train.ind,]
brv.test = brv[-train.ind,]
train.ind = sample(nrow(brv.train), 4*round(nrow(brv.train)/5)) #20%
   of data for cv hold out
brv.train = brv.train[train.ind,]
brv.holdout = brv.train[-train.ind,]


#----------- Helper Functions ------------
#Prediction Threshold for naive bayes
```

```
predictThresh <- function(prediction, K) {
  df <- as.data.frame(prediction)
  return(as.numeric(df[2]/df[1] > K))
}

predictThreshGlm <- function(prediction, K) {
  return(as.numeric(prediction > K))
}

#
   http://stackoverflow.com/questions/30566788/legend-label-errors-with-glmnet-plo
# prints the legend in the coefficients plot (for lasso and ridge)
lbs_fun <- function(legend_location, fit, ...) {
  L <- length(fit$lambda)
  x <- log(fit$lambda[L])
  y <- fit$beta[, L]
  labs <- names(y)
  text(x, y, labels=labs, ...)
  # legend(legend_location, legend=labs, col=1:6, lty=1) # <- labels
     are hard to see, hence commented out
}

#----------- Compare Between Model Classes: ROC Plots ------------
f1 <- factor(clicked) ~ platform + geo_location + traffic_source +
   factor(weekDay) + loadTimestamp +
  display_id + document_id + ad_id

glmROC <- function() {
  glm.m1 <- glm(f1, data = brv.train, family = binomial())
  rawPre <- predict(glm.m1, brv.train, type="response")
  glm.roc <- data.frame()
  for(i in seq(0, 1, 0.01)){
    pre1 <- predictThreshGlm(rawPre, K=i)
    cm <- as.matrix(ConfusionMatrix(y_pred = pre1, y_true =
       brv.train$clicked))
    if(!("0" %in% colnames(cm))) cm <- cbind(c(0,0),cm)
    else if(!("1" %in% colnames(cm))) cm <- cbind(cm, c(0,0))
    print(cm)
    tn <- cm[1]
    fn <- cm[2]
    fp <- cm[3]
    tp <- cm[4]
    tpr <- tp/(tp+fn)
    fpr <- fp/(fp+tn)
    zeroOne <- ZeroOneLoss(pre1, brv.train$clicked)
    newRow <- data.frame(thresh=i, tn=tn, fp=fp, fn=fn, tp=tp, tpr=tpr,
       fpr=fpr, zo=zeroOne)

    glm.roc <- bind_rows(glm.roc, newRow)
  }
  loss.plot <- ggplot(glm.roc, aes(thresh, zo)) +
```

```
    geom_point(aes(color=glm.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    labs(color='Threshold')
  print(loss.plot)
  roc.plot <- ggplot(glm.roc, aes(fpr, tpr)) +
    geom_point(aes(color=glm.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    geom_abline(slope=1, intercept=0, color="blue") +
    labs(color='Threshold')
  print(roc.plot)
  return(glm.roc)
}
#glm.roc <- glmROC()


naivebayesROC <- function() {
  naiveBayes.m1 <- naiveBayes(f1, data=brv.train)
  rawPre <- predict(naiveBayes.m1, brv.train, type = "raw")
  bayes.roc <- data.frame()
  for(i in seq(0.2, 0.8, 0.005)){
    pre1 <- predictThresh(rawPre, K=i)
    cm <- as.matrix(ConfusionMatrix(y_pred = pre1, y_true =
       brv.train$clicked))
    if(!("0" %in% colnames(cm))) cm <- cbind(c(0,0),cm)
    else if(!("1" %in% colnames(cm))) cm <- cbind(cm, c(0,0))
    print(cm)
    tn <- cm[1]
    fn <- cm[2]
    fp <- cm[3]
    tp <- cm[4]
    tpr <- tp/(tp+fn)
    fpr <- fp/(fp+tn)
    zeroOne <- ZeroOneLoss(pre1, brv.train$clicked)
    newRow <- data.frame(thresh=i, tn=tn, fp=fp, fn=fn, tp=tp, tpr=tpr,
       fpr=fpr, zo=zeroOne)
    bayes.roc <- bind_rows(bayes.roc, newRow)
  }

  loss.plot <- ggplot(bayes.roc, aes(thresh, zo)) +
    geom_point(aes(color=bayes.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    labs(color='Threshold')
  print(loss.plot)
  roc.plot <- ggplot(bayes.roc, aes(fpr, tpr)) +
    geom_point(aes(color=bayes.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    geom_abline(slope=1, intercept=0, color="blue") +
    labs(color='Threshold')
  print(roc.plot)
  return(bayes.roc)
}
```

```
#bayes.roc <- naivebayesROC()


svmROC <- function() {
  svm.roc <- data.frame()
  for(i in seq(4.5, 5, 0.1)){
    svm.m1 <- svm(f1, data=brv.train, class.weights = c("0"=1, "1"=i))
    pre1 <- predict(svm.m1, brv.train)
    cm <- as.matrix(ConfusionMatrix(y_pred = pre1, y_true =
       brv.train$clicked))
    if(!("0" %in% colnames(cm))) cm <- cbind(c(0,0),cm)
    else if(!("1" %in% colnames(cm))) cm <- cbind(cm, c(0,0))
    print(cm)
    tn <- cm[1]
    fn <- cm[2]
    fp <- cm[3]
    tp <- cm[4]
    tpr <- tp/(tp+fn)
    fpr <- fp/(fp+tn)
    zeroOne <- ZeroOneLoss(pre1, brv.train$clicked)
    newRow <- data.frame(thresh=i, tn=tn, fp=fp, fn=fn, tp=tp, tpr=tpr,
       fpr=fpr, zo=zeroOne)
    svm.roc <- bind_rows(svm.roc, newRow)
  }

  loss.plot <- ggplot(svm.roc, aes(thresh, zo)) +
    geom_point(aes(color=svm.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    labs(color='Threshold')
  print(loss.plot)
  roc.plot <- ggplot(svm.roc, aes(fpr, tpr)) +
    geom_point(aes(color=svm.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    geom_abline(slope=1, intercept=0, color="blue") +
    labs(color='Threshold')
  print(roc.plot)
  return(svm.roc)
}
#svm.roc <- svmROC()

#The AOC of the ROC plots indicate that Logistic Regression and SVMs
   work better than Naive Bayes
#in this context. SVMs take prohibitively long to train on this data
   with more than
#a few covariates, so we will take Logisitic Regression as the best
   model class for
#this application

#We can also use ROC plots and Loss vs. Threshold plots to tune our
   threshold
#for this application, as we will see later.
```

```
#----------- Choose Covariates for Logistic Regression ------------
#Remove HUGE factor covariates
brv.train <- select(brv.train, -publish_time, -geo_location)

# 1. Forward and Backwards Stepwise Selection
min.model <- glm(factor(clicked) ~ 1, data = brv.train, family =
   binomial())
biggest <- formula(glm(factor(clicked) ~ .:., brv.train, family =
   binomial()))
fwd.model <- step(min.model, direction='forward', scope=biggest)
print(summary(fwd.model))


# 2. Lasso and Ridge Regularization
X <- model.matrix(factor(clicked) ~ ., brv.train)
y <- brv.train$clicked
grid=10^seq(2, -2, length = 100) # choosing lambda in the range of -2
   to 10

ridge = glmnet(X, y, alpha=0, lambda=grid, family="binomial") #
   alpha=0 -> Ridge
plot(ridge, xvar="lambda", col=1:dim(coef(ridge))[1]) # Get the plot
   of coefficients w.r.t. lambda
lbs_fun('topright', ridge)

lasso = glmnet(X, y, alpha=1, lambda=grid, family="binomial") #
   alpha=1 -> Lasso
plot(lasso, xvar="lambda", col=1:dim(coef(lasso))[1], xlim = c(-4.6,
   -3.5)) # Get the plot of coefficients w.r.t. lambda
lbs_fun('topright', lasso)

# - Forward stepwise selection finds that the model with the lowest
   AIC is one that
#contains platform, source_id, publisher_id, display_id, ad_id,
   category_confidence_level,
#traffic_source, and several interaction terms.
# - Backwards stepwise selection immediately quits and returns clicked
   ~ 1.
# - Ridge Regularization shows that platformmobile,
   topic_confidence_level,
#category_confidence_level, traffic_sourcesearch, and
   traffic_sourcesocial are
#the most relevant covariates.
# - Lasso Regularization immediately pushes all coefficients to 0,
   except for
#the coefficient for platformmobile.

#Based on the results above, we will go with the covariates returned by
```

```
#forward stepwise selection, because this model returns the lowest
   AIC, and
#it contains covariates that make sense intuitively.

#----------- Threshold for Logistic Regression: More ROC ------------
#Since we could achieve a zero-one loss of .2 simply by predicting 0
   (no click),
#we believe it is important to examine confusion matricies for
   different thresholds.

#Since the goal is to predict when an Ad will be clicked on, it is not
   a big deal if
#we have false positives, as long as we have true positives to go
   along with them.
#Therefore, our goal is to optimize for precision, rather than
   zero-one loss

glmROCThresh <- function() {
  glm.m1 <- fwd.model
  rawPre <- predict(glm.m1, brv.train, type="response")
  glm.roc <- data.frame()
  for(i in seq(0, 1, 0.001)){
    pre1 <- predictThreshGlm(rawPre, K=i)
    cm <- as.matrix(ConfusionMatrix(y_pred = pre1, y_true =
       brv.train$clicked))
    if(!("0" %in% colnames(cm))) cm <- cbind(c(0,0),cm)
    else if(!("1" %in% colnames(cm))) cm <- cbind(cm, c(0,0))
    print(cm)
    tn <- cm[1]
    fn <- cm[2]
    fp <- cm[3]
    tp <- cm[4]
    tpr <- tp/(tp+fn)
    fpr <- fp/(fp+tn)
    zeroOne <- ZeroOneLoss(pre1, brv.train$clicked)
    newRow <- data.frame(thresh=i, tn=tn, fp=fp, fn=fn, tp=tp, tpr=tpr,
       fpr=fpr, zo=zeroOne)
    glm.roc <- bind_rows(glm.roc, newRow)
  }
  loss.plot <- ggplot(glm.roc, aes(thresh, zo)) +
    geom_point(aes(color=glm.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    labs(color='Threshold')
  print(loss.plot)
  roc.plot <- ggplot(glm.roc, aes(fpr, tpr)) +
    geom_point(aes(color=glm.roc$thresh)) +
    xlim(0,1) + ylim(0,1) +
    geom_abline(slope=1, intercept=0, color="blue") +
    labs(color='Threshold')
  print(roc.plot)
  return(glm.roc)
```

```
}
glm_final.roc <- glmROCThresh()

glm_final.roc <- mutate(glm_final.roc, precision = tp/(tp+fp))

precision.plot <- ggplot(glm_final.roc, aes(thresh, precision)) +
  geom_point()
print(precision.plot)
sensitivity.plot <- ggplot(glm_final.roc, aes(thresh, tpr)) +
  geom_point()
print(sensitivity.plot)

#We can achieve a precision of 0.5, for certain thresholds around
   0.29, but this still results
#in very few positives total. Our chart of sensativity shows that our
   true positive rate
#is well below 0.1.

#In addition, we worry that a threshold that achieves a precision of
   0.5 is subject to overfitting.
#Therefore, we will cross validate these results on our hold out set
glmROCHoldout <- function() {
  glm.m1 <- fwd.model
  rawPre <- predict(glm.m1, brv.holdout, type="response")
  glm.roc <- data.frame()
  for(i in seq(0.25, 0.3, 0.0001)){
    pre1 <- predictThreshGlm(rawPre, K=i)
    cm <- as.matrix(ConfusionMatrix(y_pred = pre1, y_true =
       brv.train$clicked))
    if(!("0" %in% colnames(cm))) cm <- cbind(c(0,0),cm)
    else if(!("1" %in% colnames(cm))) cm <- cbind(cm, c(0,0))
    print(cm)
    tn <- cm[1]
    fn <- cm[2]
    fp <- cm[3]
    tp <- cm[4]
    tpr <- tp/(tp+fn)
    fpr <- fp/(fp+tn)
    zeroOne <- ZeroOneLoss(pre1, brv.train$clicked)
    newRow <- data.frame(thresh=i, tn=tn, fp=fp, fn=fn, tp=tp, tpr=tpr,
       fpr=fpr, zo=zeroOne)
    glm.roc <- bind_rows(glm.roc, newRow)
  }
  return(glm.roc)
}
glm_holdout.roc <- glmROCThresh()
glm_holdout.roc <- mutate(glm_holdout.roc, precision = tp/(tp+fp))

holdout.precision.plot <- ggplot(glm_holdout.roc, aes(thresh,
   precision)) +
  geom_point()
```

```
print(holdout.precision.plot)
holdout.sensitivity.plot <- ggplot(glm_holdout.roc, aes(thresh, tpr)) +
  geom_point() + xlim(.25,.3) + ylim(0,.1)
print(holdout.sensitivity.plot)
holdout.zo.plot <- ggplot(glm_holdout.roc, aes(thresh, zo)) +
  geom_point()
print(holdout.zo.plot)
```