**Rapleaf Hackathon Working Document**

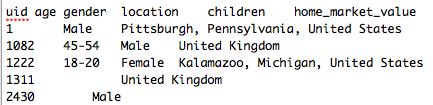
<http://www.kaggle.com/c/RapLeafHackAThon/Data>

The dataset consists of 4 files.

1. demographics.tsv
2. headlines.tsv
3. training.csv
4. example\_entry.csv

**demographics.tsv**

This file is provided by Rapleaf. The first couple lines of this file looks like:



The columns in the file are:

***uid age gender location children home\_market\_value home\_owner\_status home\_property\_type household\_income length\_of\_residence marital\_status***

There are more fields than these 11 available from the RapLeaf API, some of which you have to pay for. The data in demographics.tsv has to be formatted because R can’t understand ranges of values like Age=45-54. You have a choice to format the data ranges into distinct categories which can be accepted by s/w such as R or change to a numeric range which isn’t limited.

* getting rid of the k and mm values where house values are specified as 1mm+ or 500k-1mm.
* getting rid of the + signs in the years column, 20+ years to a numerical value vs. categorical value.

**headlines.tsv**

This file is provided by the financial website which is confidential. But doing a web search on the urls reveals this is motley fool at fool.com. This file contains the urls visited by the users. The columns for this file are:

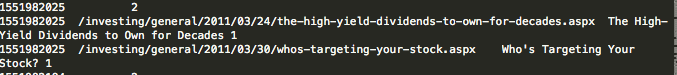
Macintosh HD:Users:dc:Desktop:Screen Shot 2011-09-15 at 12.23.22 PM.png

where uid represents the user id, same as in demographics.tsv, url represents the url with the front part of fool.com removed, headline lists the headline of the html page and repetitions is the number of times the user went to the page.

There are 2 types of data in headlines.tsv.

Macintosh HD:Users:dc:Desktop:Screen Shot 2011-09-15 at 12.22.49 PM.png

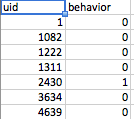
and



The counts for page vies are listed with the specific urls listed. These counts don’t always match up but should be close enough.

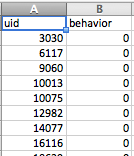
**training.csv**

Training file with columns uid and behavior. Behavior=0 indicates the user is not a paying customer, behavior=1 indicates the user is a paying customer. Below user 2430 is a premium subscriber.



**example\_entry.csv**

This is the file to submit to kaggle.com after you find your probabilities. This is a list of user ids. Fill in your results under behavior. We aren’t going to discuss how to do this in this document.



**Here is an example of how to submit to the competition using the Random Forest**

**> training<-read.csv("/Users/dc/Desktop/HeadlineTraining.txt", header=TRUE, sep="\t")**

**> head(training)**

**uid beh pageV uid.1 age sex loc child home. own prop income**

**1 1 0 2450 1 Male Pittsburgh, Pennsylvania, United States**

**2 1082 0 324 1082 45-54 Male United Kingdom**

**3 1222 0 10 1222 18-20 Female Kalamazoo, Michigan, United States 50k-75k**

**4 1311 0 48 1311 United Kingdom**

**5 2430 1 41 2430 Male**

**6 3634 0 16 3634 Male**

**residlen marital**

**1**

**2**

**3**

**4**

**5**

**6**

**//convert the pageV to numeric from categorical.**

**//convert beh from numeric to categorical**

**> data=training**

**> nrow(training)**

**[1] 201398**

**> training2=data.frame(beh=data$beh, pageV=as.numeric(data$pageV), age=data$age, sex=data$sex, child=data$child, home=data$home., own=data$own, prop=data$prop, income=data$income, residlen=data$residlen, marital=data$marital)**

**> nrow(training2)**

**[1] 201398**

**> sampledtrainingindex<-sample(1:nrow(training), nrow(training)/3)**

**> length(sampledtrainingindex)**

**[1] 67132**

**> rapleaftest=training2[sampledtrainingindex,]**

**> rapleaftrain=training2[-sampledtrainingindex,]**

**> nrow(rapleaftest)**

**[1] 67132**

**> nrow(rapleaftrain)**

**[1] 134266**

**>rapleaf.rf<-randomForest(rapleaftrain[,c(2:11)], rapleaftrain$beh, do.trace=TRUE, ntree=100, importance=TRUE, , forest=TRUE)**

**To clean the NA terms for randomForests,**

**>trainingr<-na.roughfix(training)**

**When running randomForests, set the first try to ntree to 20-30. You don’t need a 100-500 trees and leave out all the predictor variables. In the line above we have c(2:11), rn with c(2,3,4) as an example to see the error rates move around.**

**There are 2 data files,**

1. **training.txt which contains the converted and cleaned demographic and headline data. All of the categorical data is converted to numeric.**
2. **HeadineEntry.txt and HeadlineTraining.txt which contain the raw data processed where pageV is added. There is no other conversion from categorical to numerical values.**

**We have to perform selective joins on the 2 files to determine which is the best mix for our data set.**

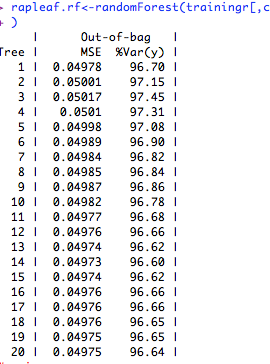
**Read in data:**

**>training<-read.csv(file="/Users/dc/Desktop/training.txt", header=TRUE, sep="\t")**

**> head(training)**

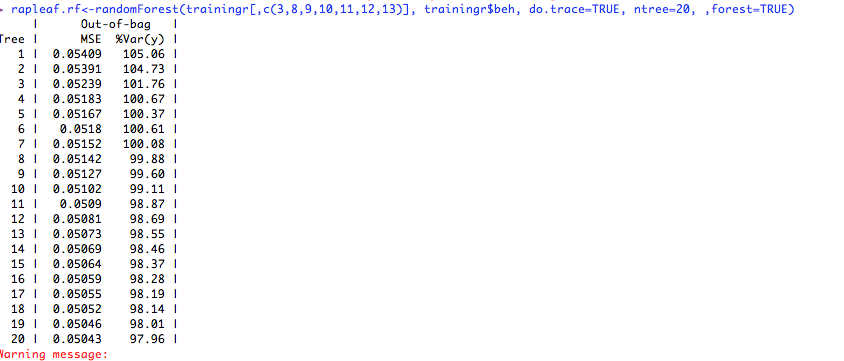
**>** **headlineTraining<-read.csv(file="/Users/dc/Desktop/HeadlineTraining.txt", header=TRUE ,sep="\t")**

**> training2<-data.frame(uid=training$uid,beh=training$beh, rep=training$rep, bag=training$bag, bigram=training$bigram, age=as.factor(training$age), sex=as.factor(training$sex), loc=as.double(training$loc), child=as.factor(training$child), homeval=training$homeval, owner=training$owner, proptype=as.factor(training$prop\_type), houseincome=training$houseincome, lenresid=as.factor(training$len\_resid), marital=as.factor(training$marital))**

****

**In the screen shot above you can see the error rate rise to .050l then decrease after that. This is good.**

**Add more predictor variables..**

****

You can see from the screen shot above the MSE error is decreasing and not stabilized, this is a good place to add more trees.

The first screen shot shows a Var of 96.64, add more trees in the second screen shot to get to this variance.

**>rapleaf.rf<-randomForest(,c(3,8,9,10,11,12,13), training$beh, do.trace=TRUE, , forest=TRUE)**

Get some statistics on the new data set

**> training2Index<-sample(1:nrow(training2), round(nrow(training2)\*.75))**

**> training2Sampled<-training2[training2Index,]**

**> training2SampledTest<-training2[-training2Index,]**

Importance: <http://www.stanford.edu/~stephsus/R-randomforest-guide.pdf> once you have all the variables in the model and can get it to run, time to figure out where to invest your time. You wont be able to clean all the data to what or where you want it. We can determine the relative contribution of each predictor variable to the result.

**> install.packages("Design")**

**> install.packages("languageR")**

**> install.packages("party",dependencies=TRUE)**

To load the packages into the runtime:

**>library(party)**

**>library(languageR)**

**>library(Design)**

The problem after loading in data is the missing values denoted by <NA> if the column is a factor and NA if the column is numeric. Make sure you don’t confuse the string “NA” with the R NA entry which means there is nothing there.

**> training<-read.csv(file="/Users/dc/Desktop/training.txt", header=TRUE, sep="\t")**

//clean the training values, create factors for columns which are numeric. This doesn’t do anyting to fix the NA values.

**> training2<-data.frame(uid=training$uid,beh=training$beh, rep=training$rep, bag=training$bag, bigram=training$bigram, age=as.factor(training$age), sex=as.factor(training$sex), loc=as.double(training$loc), child=as.factor(training$child), homeval=training$homeval, owner=training$owner, proptype=as.factor(training$prop\_type), houseincome=training$houseincome, lenresid=as.factor(training$len\_resid), marital=as.factor(training$marital))**

**> names(training2)**

This puts averages in the columns whih are numeric. This is good?

**>training2r<-na.roughfix(training2)**

#I actually ran the below because it took forever, started leaving fields off

**> rapleaf.rf<-randomForest(rapleaftrain[,c(2:7)], rapleaftrain$beh, do.trace=TRUE, ntree=100, importance=TRUE, , forest=TRUE)**

**> pred<-predict(rapleaf.rf, rapleaftest[,c(2:7)])**

**this will take a while…. ½ hour or more….**

**> predict(rapleaf.rf, rapleaftest[,c(2:7)])**

**> pred<-predict(rapleaf.rf, rapleaftest[,c(2:7)])**

**> rapleaf.pred<-prediction(pred,rapleaftest$beh)**

**Error: could not find function "prediction"**

**> library(ROCR)**

**> rapleaf.pred<-prediction(pred,rapleaftest$beh)**

**> rapleaf.perf<-performance(rapleaf.pred, "tpr","fpr")**

**> performance(rapleaf.pred, "auc")@y.values[[1]]**

**[1] 0.7064355**

**Create output data, I didn’t train on the test data because running this took so damned long.**

**> entry<-read.csv("/Users/dc/Desktop/HeadlineEntry.txt", header=TRUE, sep="\t")**

**//note the use of as.character() to convert 052 factor into a string then to a //numeric. If you don’t do this you get factor levels instead of numbers.**

**> entry2<-data.frame(pageV=as.double(as.character(entry$beh)), age=entry$uid.1, sex=entry$age, child=entry$loc, home=entry$child, own=entry$home., prop=entry$own, income=entry$prop, residlen=entry$income, marital=entry$residlen)**

**> head(entry2)**

**pageV age sex child home own prop income residlen marital**

**1 52 Male**

**2 11 55-64 Male Yes 150k-200k Own Single Family Dwelling 250k+ 11-15 years Married**

**3 39 55-64 Male**

**4 287**

**5 7 65+ Male No 25k-50k Own Single Family Dwelling 75k-100k 16-19 years Married**

**6 28 45-54 Female Yes 100k-150k Own 50k-75k 11-15 years Married**

**To save memory by deleting objects:**

**> headings<-NULL**

**Boosted Regression Trees:**

The distro package implementing gradient boosted trees are much easier to use and less work than the randomForest libraries.

rodica@amazon.com

> headlineTraining2<=data.frame(beh=headlineTraining$bet, pageV=headlineTraining$pageV, age=headlineTraining$age, sex=headlineTraining$sex, child=headlineTraining$child,home=headlineTraining$home. , own=headlineTraining$own, prop=headlineTraining$prop, income=headlineTraining$income, residlen=headlineTraining$residlen, marital=headlineTraining$marital)

**Test predictive power of demographich data minus location, age, sex. 4%.**

[**https://gist.github.com/1288633**](https://gist.github.com/1288633)

Test predictive power of pageViews from headline.tsv data minus location:

73%

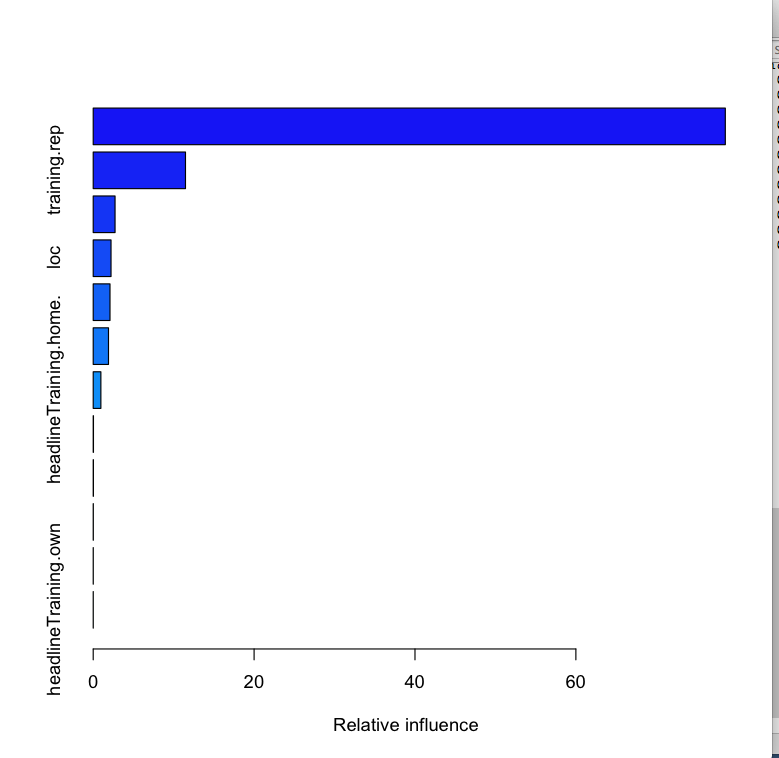
<https://gist.github.com/1288643>

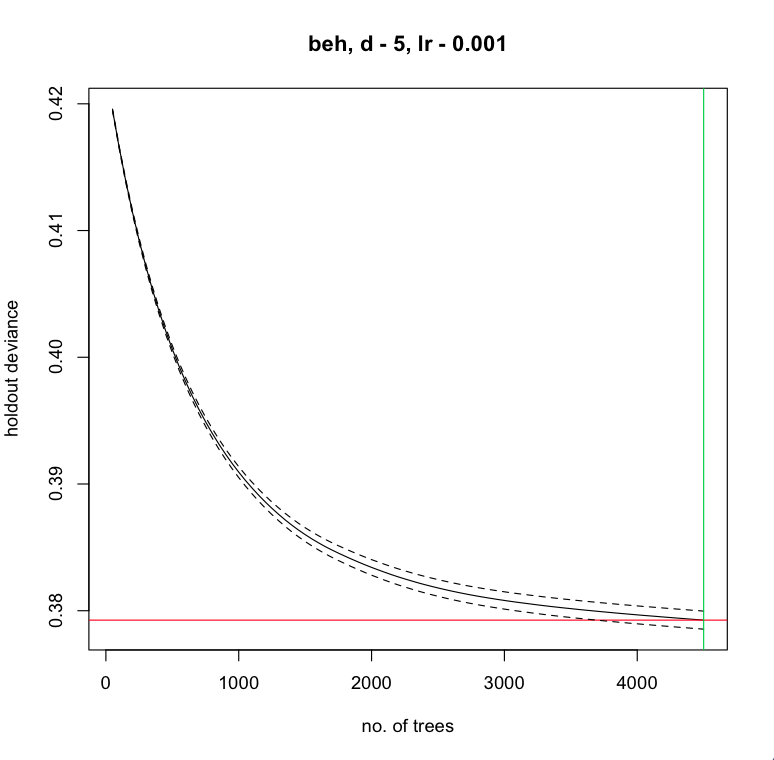
The most important contribution seems to be from page views from the headlines.tsv file. This isn’t surprising since this is the only field which isn’t sparse, we have data for all the users.

Test predictive power of repetitions repeated with summary data from headlines and demographics data without location. 75% ROC

<https://gist.github.com/b4e8134ff780c12a4f9e>

Here is a graph of the difference in features, with location being the most important demographic feature because we converted this to a numeric attribute. This probably indicates we need to convert the other attributes into a numeric attribute and then measure the change.

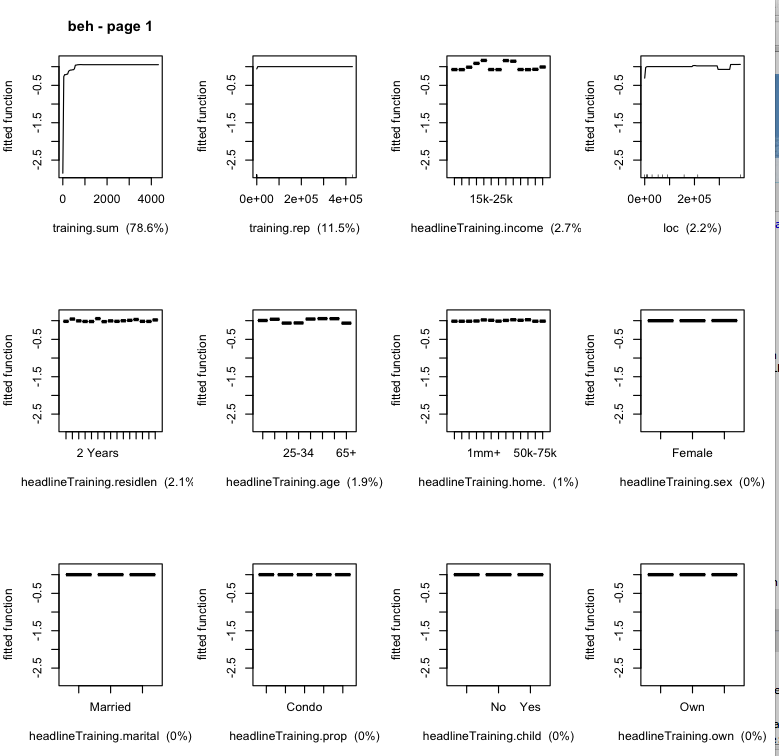




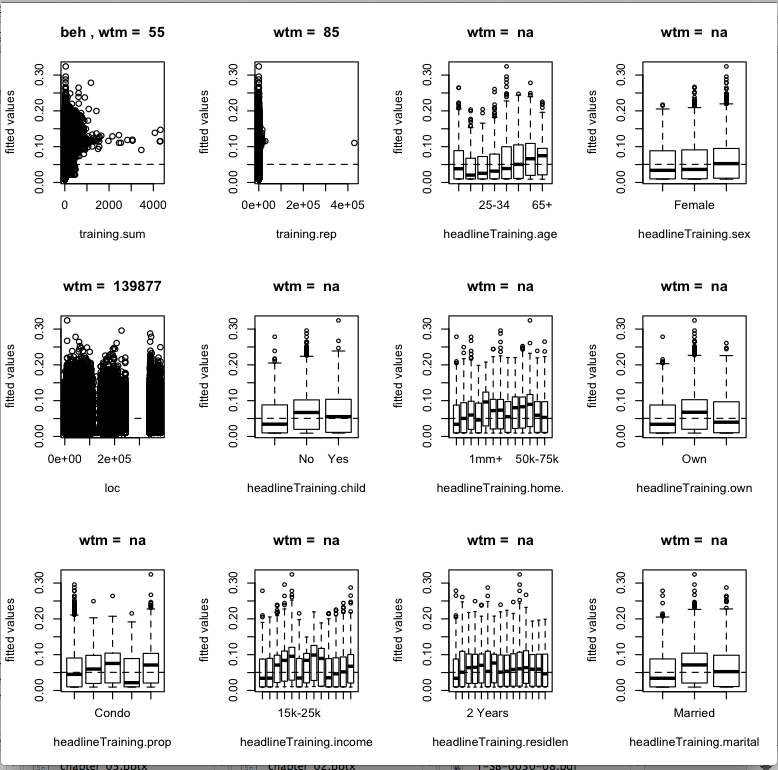
The intersection of the red and green lines indicate where the minimum is.

The graph above is bad in the sense we are looking for the holdout deviance to start rising again. If we see that then we know roughly how to pick the number of trees to run in our production model. We can see th holdout deviance decreasing with the number of trees still. This is also a sign we need to spend some time with cleaning the features given the amount the holdout deviance is decreasing as the number of trees increase. This is also an indication we have correlated features.

> gbm.plot(gbm2, nplots=12)



> gbm.plot.fits(gbm2)



**Find the Interactions between the predictor vars.**

> find.int<-gbm.interactions(gbm2)

gbm.interactions - version 2.9

Cross tabulating interactions for gbm model with 12 predictors

1 2 3 4 5 6 7 8 9 10 11

> find.int$interactions

training.sum training.rep headlineTraining.age headlineTraining.sex loc headlineTraining.child headlineTraining.home.

training.sum 0 1.78 2.26 0.01 1.66 0 1.64

training.rep 0 0.00 0.02 0.00 0.05 0 0.01

headlineTraining.age 0 0.00 0.00 0.00 0.11 0 0.00

headlineTraining.sex 0 0.00 0.00 0.00 0.00 0 0.00

loc 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.child 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.home. 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.own 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.prop 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.income 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.residlen 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.marital 0 0.00 0.00 0.00 0.00 0 0.00

headlineTraining.own headlineTraining.prop headlineTraining.income headlineTraining.residlen headlineTraining.marital

training.sum 0 0 0.38 0.34 0

training.rep 0 0 0.18 0.02 0

headlineTraining.age 0 0 0.05 0.01 0

headlineTraining.sex 0 0 0.00 0.00 0

loc 0 0 0.04 0.01 0

headlineTraining.child 0 0 0.00 0.00 0

headlineTraining.home. 0 0 0.00 0.01 0

headlineTraining.own 0 0 0.00 0.00 0

headlineTraining.prop 0 0 0.00 0.00 0

headlineTraining.income 0 0 0.00 0.07 0

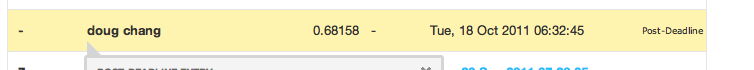
headlineTraining.residlen 0 0 0.00 0.00 0

headlineTraining.marital 0 0 0.00 0.00 0

>

**Running a prediction with and without demographics for a test on the final results as a contrast to importance:**

**Bigrams without demographics: ROC score=.68158**

****

**Code for bigrams without demographics:**

**> head(trainingbigram)**

**beh bag bigram age sex loc child homeval owner prop\_type houseincome len\_resid marital**

**1 0 28082 257018 NA 1000 107657 NA NA NA NA NA NA NA**

**2 0 0 0 49 1000 140 NA NA NA NA NA NA NA**

**3 0 0 0 19 2000 76244 NA NA NA NA 62500 NA NA**

**4 0 0 0 NA NA 140 NA NA NA NA NA NA NA**

**5 1 143 162 NA 1000 NA NA NA NA NA NA NA NA**

**6 0 0 0 NA 1000 NA NA NA NA NA NA NA NA**

**> gbmNoDemo<-gbm.step(data=trainingbigram, gbm.x=2:3 ,gbm.y=1, fold.vector=NULL, tree.complexity=4, learning.rate=0.01)**

**GBM STEP - version 2.9**

**Performing cross-validation optimisation of a boosted regression tree model**

**for beh with dataframe trainingbigram and using a family of bernoulli**

**Using 201398 observations and 2 predictors**

**creating 10 initial models of 50 trees**

**folds are stratified by prevalence**

**total mean deviance = 0.4228**

**tolerance is fixed at 4e-04**

**ntrees resid. dev.**

**50 0.4075**

**now adding trees...**

**100 0.4015**

**150 0.399**

**200 0.398**

**250 0.3975**

**300 0.3973**

**350 0.3972**

**400 0.3971**

**450 0.3971**

**500 0.397**

**550 0.397**

**600 0.397**

**650 0.397**

**700 0.397**

**750 0.3969**

**800 0.3969**

**850 0.3969**

**900 0.3969**

**950 0.3969**

**1000 0.3969**

**1050 0.3969**

**1100 0.3969**

**1150 0.3969**

**fitting final gbm model with a fixed number of 1150 trees for beh**

**mean total deviance = 0.423**

**mean residual deviance = 0.395**

**estimated cv deviance = 0.397 ; se = 0.001**

**training data correlation = 0.17**

**cv correlation = 0.16 ; se = 0.002**

**training data ROC score = 0.706**

**cv ROC score = 0.697 ; se = 0.002**

**elapsed time - 35.84 minutes**

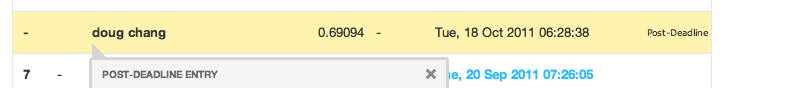
**> summary(gbmNoDemo)**

**var rel.inf**

**1 bag 82.33316**

**2 bigram 17.66684**

**Bigrams with demographics: ROC score=.6904**

****

**Code for bigrams with demographics:**

**> gbmbigram<-gbm.step(data=trainingbigram, gbm.x=2:13 ,gbm.y=1, fold.vector=NULL, tree.complexity=4, learning.rate=0.01)**

**GBM STEP - version 2.9**

**Performing cross-validation optimisation of a boosted regression tree model**

**for beh with dataframe trainingbigram and using a family of bernoulli**

**Using 201398 observations and 12 predictors**

**creating 10 initial models of 50 trees**

**folds are stratified by prevalence**

**total mean deviance = 0.4228**

**tolerance is fixed at 4e-04**

**ntrees resid. dev.**

**50 0.4075**

**now adding trees...**

**100 0.4012**

**150 0.3984**

**200 0.3969**

**250 0.3961**

**300 0.3956**

**350 0.3952**

**400 0.395**

**450 0.3948**

**500 0.3947**

**550 0.3945**

**600 0.3944**

**650 0.3943**

**700 0.3942**

**750 0.3941**

**800 0.394**

**850 0.3939**

**900 0.3938**

**950 0.3937**

**1000 0.3936**

**1050 0.3935**

**1100 0.3934**

**1150 0.3934**

**1200 0.3933**

**1250 0.3932**

**1300 0.3931**

**1350 0.3931**

**1400 0.393**

**1450 0.3929**

**1500 0.3929**

**1550 0.3928**

**1600 0.3928**

**1650 0.3927**

**1700 0.3926**

**1750 0.3926**

**1800 0.3925**

**1850 0.3925**

**1900 0.3924**

**1950 0.3924**

**2000 0.3924**

**2050 0.3923**

**2100 0.3923**

**2150 0.3922**

**2200 0.3922**

**2250 0.3921**

**2300 0.3921**

**2350 0.3921**

**2400 0.392**

**2450 0.392**

**2500 0.3919**

**2550 0.3919**

**2600 0.3919**

**fitting final gbm model with a fixed number of 2600 trees for beh**

**mean total deviance = 0.423**

**mean residual deviance = 0.386**

**estimated cv deviance = 0.392 ; se = 0**

**training data correlation = 0.206**

**cv correlation = 0.176 ; se = 0.002**

**training data ROC score = 0.735**

**cv ROC score = 0.717 ; se = 0.001**

**elapsed time - 388.02 minutes**

**> summary(gbmbigram)**

**var rel.inf**

**1 bag 57.3785672**

**2 bigram 14.0975092**

**3 loc 12.4332417**

**4 houseincome 4.7689077**

**5 age 3.3130331**

**6 len\_resid 2.8410079**

**7 homeval 2.2359683**

**8 sex 1.3978125**

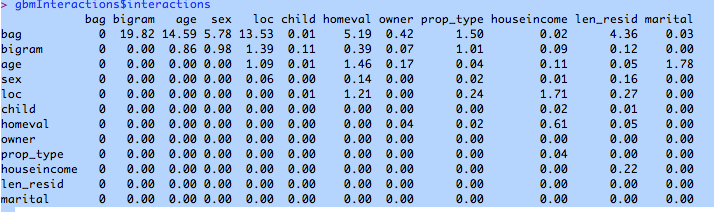
**9 marital 0.6258613**

**10 prop\_type 0.3917826**

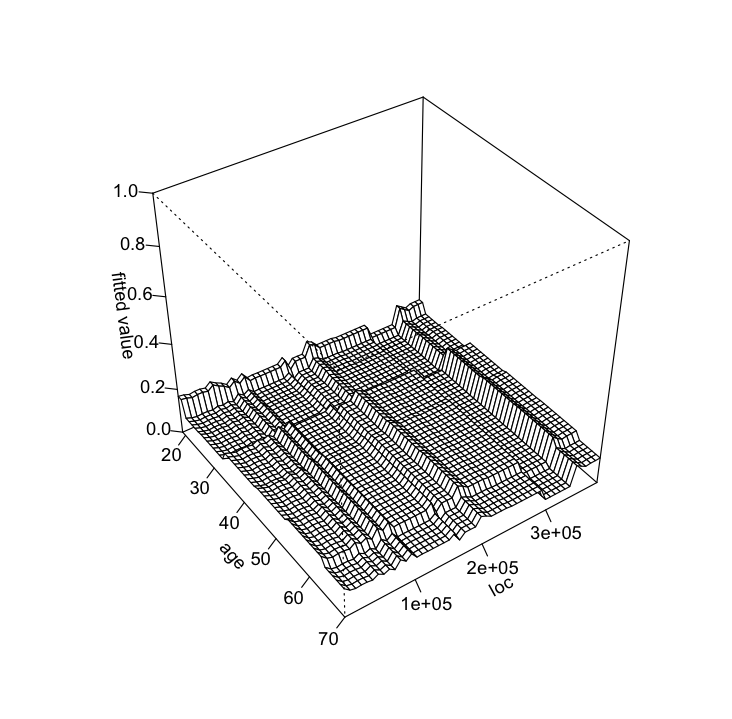
**11 child 0.3006566**

**12 owner 0.2156519**

**Interactions:**

****

**See age/location are important. Look at pairwise interactions:**

****

**Prediction:**

**> predictEntry<-data.frame(bag=entry$bag, bigram=entry$bigram, age=entry$age, sex=entry$sex, loc=entry$loc, child=entry$child, homeval=entry$homeval, owner=entry$owner, prop\_type=entry$prop\_type, houseincome=entry$houseincome, len\_resid=entry$len\_resid, marital=entry$marital)**

**> predict<-predict.gbm(gbmbigram, predictEntry, n.trees=gbmbigram$gbm.call$best.trees, type="response")**

**> write.csv(predict, file="/Users/dc/Desktop/demobigram.csv")**

**Then using excel cut and paste the UID column from the spreadsheet template into the generated demobigram.csv file.**

**Applying NLP/Text Analytics to the Rapleaf Hackathon Dataset**

RapLeaf hosted a Data Mining competition on www.kaggle.com, which ran for 3 weeks and I am happy that the 100 - 120 hours that I spent on it with a total of 62 submissions were well worth having won the competition. This was the second data mining competition that I participated on kaggle.com and won. The first one was a Wine Price prediction competition([http://blog.kaggle.com/2010/12/13/how-we-did-it-jie-and-neeral-on-winning-the-first-kaggle-in-class-competition-at-stanford/](http://blog.kaggle.com/2010/12/13/)) hosted by Stanford University that I participated and won along with Jie Yang, a Research Engineer from Yahoo Labs.

The competition data consisted of user demographics information along with the URLs and headlines that each user visited. A training set was provided to help train the algorithm to correctly predict user behavior. The goal of the competition was to predict a consumer behavior(i.e. sign up for newsletter, read another article, etc.) on a Personal Finance Site([http://www.fool.com](http://www.fool.com/)) for the users that are not in the training set. The competition submission file needed to a csv file with 100,000 rows and 2 columns(uid, behavior). The competition was constrained to maximum 5 submissions per day.

**Data Provided**  
The data files were as follows:

* demographics: One record per user along with Rapleaf data about each one
* headlines: Each row contains one URL accessed by one user, along with the title of that page, and the number of times that user accessed that URL
* training: For those users in the training set, a binary value indicating whether or not they subscribed
* example\_entry: A sample entry showing the users in the test set, and a constant value. For your entries, you should replace the constant value with a probability of subscription, based on your model

Training Data size : 201398 users

Test Data size : 100000 users

Headlines data size : 6M

I would say about 90% of the effort that I spent was on feature engineering.

**Feature Engineering**

Here is an example URL from the headlines data.

/investing/beginning/2006/12/19/credit-check-countdown.aspx

I created new features with the first word in the URL string delimited by ‘\’, here, investing and also features concatenating first and second word, here, investing.beginning. The value that I assigned to these features were the repetitions (i.e. the number of times the user viewed URLs with the word. The features were then row-normalized to take out the influence of hyper-active users.

Also, based on the training data, I figured out the distinct headline words(after removing stop-words) for behavior = 0(words.0) and behavior = 1(words.1). For each user, I then created 2 features that captured the Jaccard Similarity based on the all headline words for the user with words.0 and words.1 respectively.

Based on my experience from a Web Mining Hackathon that I participated a few months, I used diffbot API to crawl the date a particular blog/article at the URL was written. I also retrieved the Author name and the article text for the blog.

I created one feature for one author having the value as the total number of repetitions per user, per author. The author features were row-normalized. Two other author related Jaccard Similarity features were added based on auth.0 and auth.1 (i.e. auth.0 = authors who wrote articles read by users with behavior = 0, auth.1 similarly for behavior =1).

From the Wikipedia page for Motley Fool, I discovered that the paid subscription was started in April 2002. Based on this knowledge, I engineered the following date related features(date was the when the article/blog was posted by the author)

* Average date difference for each user w.r.to April 01, 2002
* 1 feature per year. i.e. year.1999, year.2000, etc. having value as the total repetitions per user, per year row-normalized
* 2 features relating to proportions of dates that were on or before April 01, 2002 and after.

Finally, I performed Latent Dirichlet Allocation Topic Modeling on both the article text(10 topics) and headline words(10 topics) to add topic proportion features for both articles and headlines. Stopwords were removed before performing LDA.

The rest were all demographics features. The was a lot of data cleaning and processing involved since there were a lot of blank lines in the headlines data and also the crawled data had noisy dates that needed to be fixed. Also, the author names were not consistent, in the sense for some blogs an author say "Jerome Seinfeld" found it cool to put his name as "Jerome 'the comedian' Seinfeld". There were a lot of such cases which need to be mapped back to the original name using regular expressions in R.

With all the above features, I had about 500 features. I performed a dry run of Gradient Boosted Machines to identify the important features and threw away the rest of them. Relative Influence returned by the summary function from gbm R package helped me filter the important features. After the filtering, I was left with 120 features.

**Modeling**

I used RandomForest, Gradient Boosted Machines(GBM), Linear Model and Robust Linear Model to form an ensemble for prediction. For the RandomForest, I had to settle for 200 trees due to time constraints, if time permitted, I would have settled for atleast the default which is 500 trees. For GBM, I chose to go for shrinkage = 0.001, 5000 trees and 5-fold cross-validation. The number of trees for GBM was decided based on best performance on CV error. For each of the models, I trained separately on pool of users who had just demographics information and users who had both demographics and headlines data available. Through experimentation, I found that this method proved to return a better AUC. Also, I would have liked to experiment with Mahout implementation of randomForest to see if I could have got a faster turn-around.

I ended up with an AUC of 0.80457 on the public leaderboard and an AUC of  0.80224 on the final test set.

<http://www.kaggle.com/c/RapLeafHackAThon/Leaderboard>

Team : Seeker

**Software**

I primarily used R for modeling purpose and Java for crawling the URLs with the help of diffbot API.

R packages used :

* lda
* lsa
* gbm
* randomForest
* lm
* MASS
* kernlab
* e1071
* som

How to measure the statistical significance of signals. Do an trial run over a sample data set with the new signal. How do you predict which signal will be more valuable before doing this work?

**LDA;** One of the more recent methods applied to text analytics more efficient than the use of tf\*idf on documents is the use of algorithms to summarize the content of a document and some way to compute similarity. For multiattribute values we use

**//does LDA increase the performance score? Easy way to implement this?**

**//if not sparse what is the contribution? Can you predict based on the number of entries which have demographics? The 100k entries, only some have demographics. How many?**

**How many of the 200k have demographics? When we split do we train on only 75%? Yes, we want to make this train on all 100%. How do you test the error if you split and test on 100%?**