**Rapleaf Hackathon**

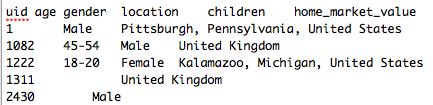
<http://www.kaggle.com/c/RapLeafHackAThon/Data>

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:

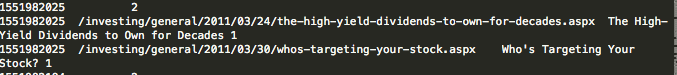
Macintosh HD:Users:dc:Desktop:Screen Shot 2011-09-15 at 12.23.22 PM.png

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

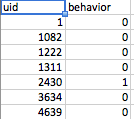
and



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

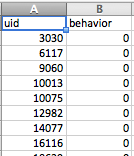
**training.csv**

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.



**Using R**

An intro into using R. I haven’t touched this in about 10 years and the command line syntax for how commas, vectors and minus signs isn’t common or intuitive with everything else out there.

To read in data from a file into a data frame:

***training<-read.csv(“/Users/dc/training.csv”, headers=TRUE)***

To make sure the data is read in correctly you should do head(training) and see distinct column names. If not fix the read.csv files by adding separators.

To access one column:

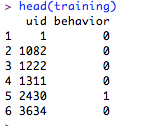
training$behavior

You can also use attach(training) and have behavior available without the training$.

R has a specific notation for Vectors, and Matricies. Vectors are denoted with the c() function for column. A vector is a single row with columns. c(1,2,3,4) refers to a column object with 4 columns with values 1,2,3,4. Depending on context, c(1,2,3,4) can refer to a real vector with 1 row of values 1,2,3,4 or it can refer to 4 rows in a matrix with all the rows in the data.frame as in training[,c(1,2,3,4)]. Multiple meanings for the same object.

Be careful of how R displays data. R is WYSIWIG. If R prints out data in a row it doesn’t mean the data is represented internally as a row.

head(training) returns the first 6 rows:



If I want the first column of training, head(training[,c(1)]) or head(training[,1]). Looks like a row, but is really a column in memory.

Macintosh HD:Users:dc:Desktop:Screen Shot 2011-09-15 at 12.52.17 PM.png

**The operations on a data frame:**

Listing column names:***names(data.frame)***

Listing top 10 or so rows: ***head(data.frame)***

Simple stats on columns and num of rows: ***summary(data.frame)***

number of rows: ***nrow(data.frame)***

listing cells: ***training[1,2]*** where 1 is the row number and 2 is the column number.

***is.data.frame(data.frame)***

listing specific columns in all the rows, make sure you use brackets to indicate this is a matrix: ***training[,4:10]***  lists all the rows for columns 4-10

listing specific rows in all the columns: ***training[2:100, ]*** lists all the columns for rows 2-100

***training[,c(1,2,3,4) ]*** refers to all rows in columns 1,2,3,4 in data frame training.

listing specific rows in specific columns: ***training[2:100, 1:4 ]*** lists rows 2-100 columns 1-4.

To display the range of the rows and range of columns use colons.

***traindemo[1:2,1:10]***

To select inside a column use the subset function and use the equality string == operator for strings and ints

>testdata=subset(heading,uid==1)

> nrow(testdata)

[1] 1

To get help in R:

>***?subset***

Help is activated using the question mark operator.

The algorithms which tend to perform the best are ensemble methods. We will look at Random Forests as am example. One of the advantages of Random Forests is the ability to distribute this algorithm over a cluster in map reduce fashion and the resistance to overfitting.

There are multiple libraries for Random Forests, randomForest, rpart, and party. The best one to start with may be randomForest and how to fit an ROC curve to evaluate tree fitting performance.

Adding a library function or set of library functions to your personal installation of R on your local development machines requires 2 steps, a package installation to download the files and a library command to load the commands into memory.

To load the randomForest library go to the command line and type:

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

then to load the library functions into the R runtime,

***>library(randomForest).***

A very common beginner mistake is to run the install.packages command and forgetting to run the library command.

Data in R is represented as either a vector or data frame. A data frame can be visualized as a matrix of data in the context of this document.

To access columns of a data frame use the $ sign. data$nameofeach name in names above. training$uid refers to the uid column, this is clearer and less error prone than training[1,]

library(randomForest)

library(ROCR)

install.packages(“session”)

library(session)

**To save a session session.save() so when R crashes and you have to sudo kill – on an R process, you can get the commands back.**

**Using the Churn Data:**

This is the ROC curve for the churn dataset using rpart and our own ROC plotting function.

**Divide the Churn data set into a training and test set:**

***> churnTrainIndex<-sample(1:nrow(churn),round(nrow(churn)\*.75))***

***> churnTraining<-churn[churnTrainIndex,]***

***> churnTest<-churn[-churnTrainIndex,]***

***> nrow(churnTest)***

***[1] 833***

***> nrow(churnTraining)***

***[1] 2500***

**SVM model:**

***> svm.model<-svm(churn~ ., data=churnTraining, cost=100, gamma=1)***

***> svm.model***

***Call:***

***svm(formula = churn ~ ., data = churnTraining, cost = 100, gamma = 1)***

***Parameters:***

***SVM-Type: C-classification***

***SVM-Kernel: radial***

***cost: 100***

***gamma: 1***

***Number of Support Vectors: 2491***

***> svm.pred<-predict(svm.model, churnTest[,-1])***

***> svmt=table(pred=svm.pred, true=churnTest[,1])***

***> svmt***

***true***

***pred No Yes***

***No 736 97***

***Yes 0 0***

**To get the probabilities rerun the svm.model with probabilies=TRUE**

**> svm.model<-svm(churn~ ., data=churnTraining, cost=100, gamma=1,probability=TRUE)**

**> svm.prob=predict(svm.model, churnTest[,-1],probability=TRUE)**

**> py=attr(svm.prob, "probabilities")[,2]**

**py gives the yes probabilities.**

**From the probabilities we can print confusion tables for different alpha:**

***> table(makePrediction(py,0.4),churnTest$churn, dnn=c("pred","actual"))***

***actual***

***pred No Yes***

***No 723 88***

***Yes 13 9***

***> table(makePrediction(py,0.5),churnTest$churn, dnn=c("pred","actual"))***

***actual***

***pred No Yes***

***No 731 91***

***Yes 5 6***

**Plot the ROC curve using fp and tp from the probs and set of confusion tables above:**

**Logistic regression using lm and using the RORC plotting functions**

Assuming we have a training and test set built.

***> churn.lm<-glm(churn~. , data=train,family=binomial())***

***> churn.lm$score<-predict(churn.lm , type='response',test)***

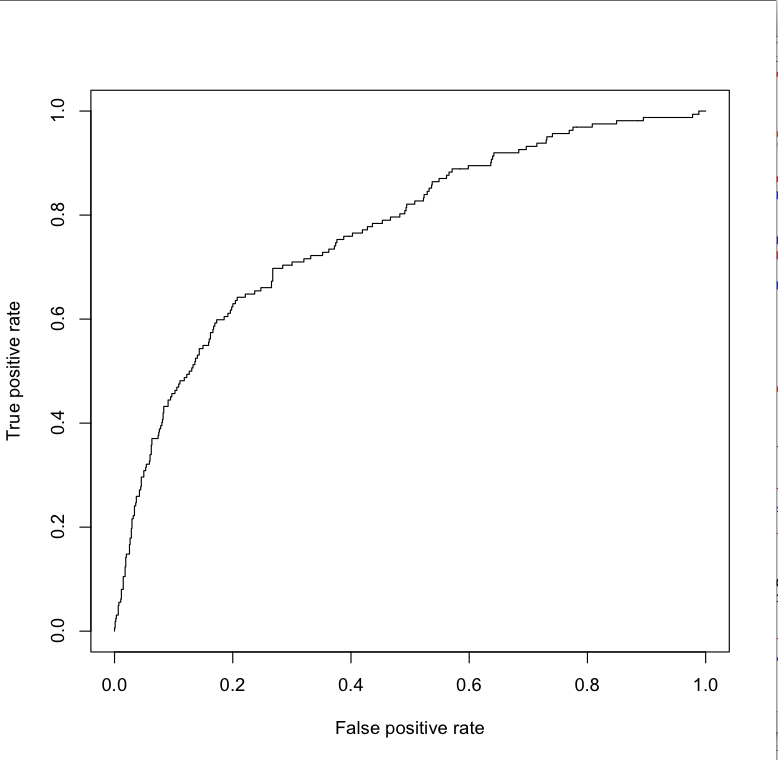
***> pred<-prediction(churn.lm$score, test$churn)***

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

***> plot(perf)***

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

***[1] 0.769504***



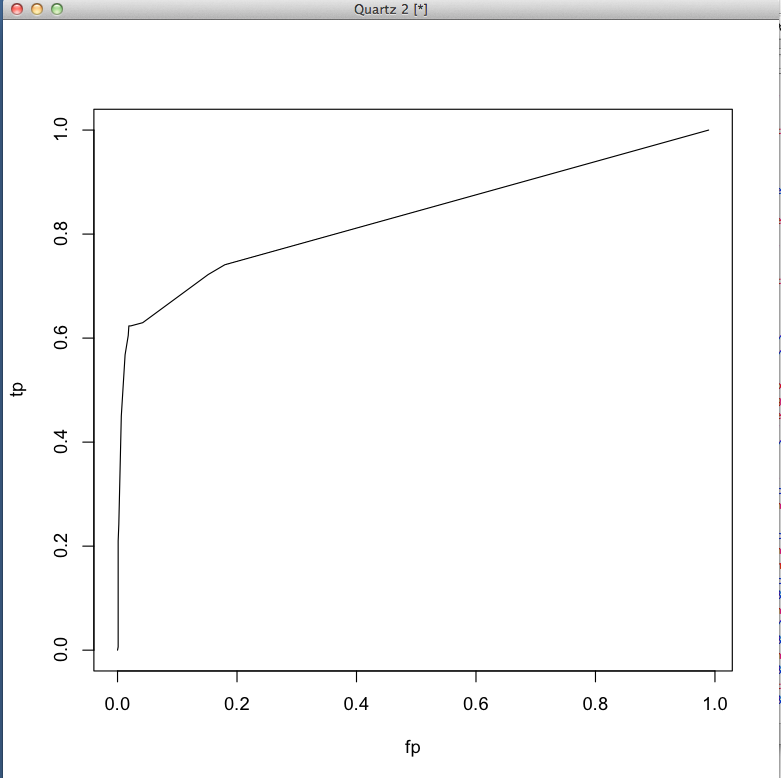
**ROC curves for a rpart tree:**

***>test\_rows=sample.int(nrow(churn),nrow(churn)/3)***

***> test=churn[test\_rows,]***

***>churn.fit3=rpart(churn~., data=train, parms=list(split="gini"))***

***> drawROC(churn.fit3, test)***



RORC code:

***> probs=predict(churn.fit3, test)[,2]***

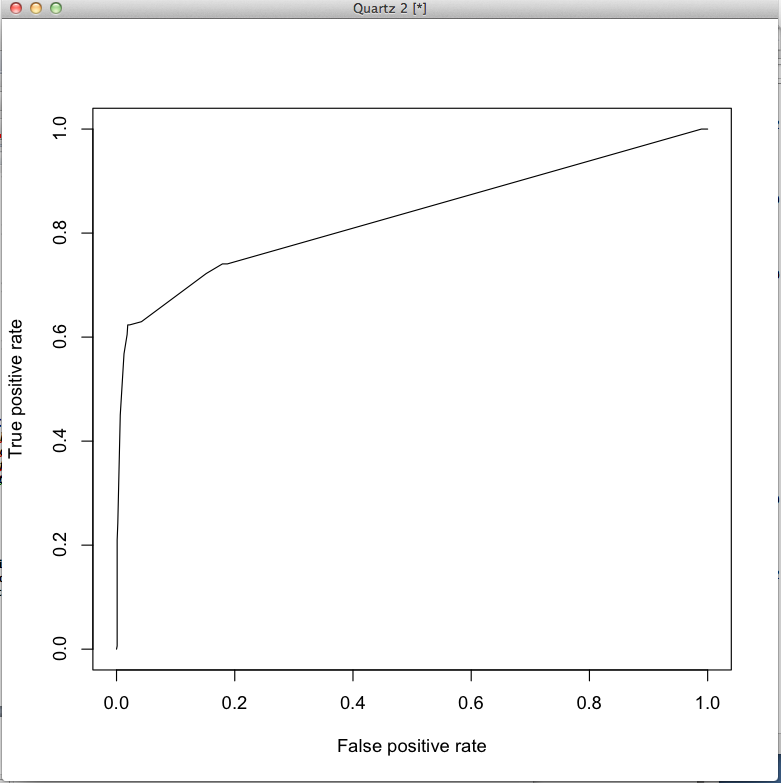
***> pred.rpart<-prediction(probs, test$churn)***

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

***> plot(perf.rpart)***

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

***[1] 0.8320129***



**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)**

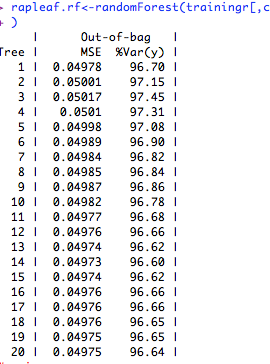
**When running randomForests, set the first try to ntree to 20-30. You don’t need a 100-500 trees and leave out all the predictor variables. In the line above we have c(2:11), rn with c(2,3,4) as an example to see the error rates move around.**

**Read in data:**

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

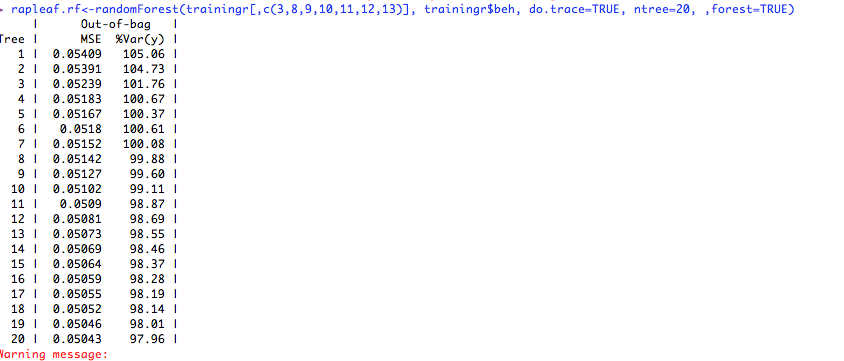
**> head(training)**

**> 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**

**Gradient Boosted Tree:**

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

>headlineTraining<-read.csv(file=”headline.txt”, header=TRUE, sep=”\t”)

> names(headlineTraining)

[1] "uid" "bet" "pageV" "uid.1" "age" "sex" "loc" "child" "home." "own" "prop" "income" "residlen" "marital"

Run a gbm tree on the columns after loc since we didn’t normalize this to categories. We can later test to see how good loc does in making this better or worse.

> gbm1<-gbm.step(data=headlineTraining, gbm.x=8:14 ,gbm.y=2, fold.vector=NULL, tree.complexity=1, learning.rate=0.001)

GBM STEP - version 2.9

Performing cross-validation optimisation of a boosted regression tree model

for bet with dataframe headlineTraining and using a family of bernoulli

Using 201398 observations and 7 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 0.4226

now adding trees...

100 0.4224

150 0.4223

200 0.4221

250 0.422

300 0.4219

350 0.4218

400 0.4217

450 0.4216

500 0.4215

550 0.4214

600 0.4213

650 0.4213

700 0.4212

750 0.4212

800 0.4211

850 0.4211

900 0.421

950 0.421

1000 0.4209

1050 0.4209

1100 0.4209

1150 0.4208

1200 0.4208

1250 0.4208

1300 0.4208

1350 0.4207

1400 0.4207

fitting final gbm model with a fixed number of 1400 trees for bet

mean total deviance = 0.423

mean residual deviance = 0.421

estimated cv deviance = 0.421 ; se = 0

training data correlation = 0.052

cv correlation = 0.051 ; se = 0.003

training data ROC score = 0.548

cv ROC score = 0.546 ; se = 0.003

elapsed time - 56.32 minutes

> 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)

**> gbm1<-gbm.step(data=headlineTraining2, gbm.x=2:11 ,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 headlineTraining2 and using a family of bernoulli

Using 201398 observations and 10 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 0.4013

now adding trees...

100 0.3921

150 0.3877

200 0.3855

250 0.3843

300 0.3837

350 0.3834

400 0.3832

450 0.3831

500 0.383

550 0.3829

600 0.3829

650 0.3829

700 0.3828

750 0.3828

800 0.3828

850 0.3828

900 0.3828

950 0.3828

1000 0.3828

1050 0.3828

1100 0.3827

1150 0.3827

1200 0.3827

fitting final gbm model with a fixed number of 1200 trees for beh

mean total deviance = 0.423

mean residual deviance = 0.38

estimated cv deviance = 0.383 ; se = 0.001

training data correlation = 0.205

cv correlation = 0.19 ; se = 0.002

training data ROC score = 0.745

cv ROC score = 0.737 ; se = 0.002

elapsed time - 131.36 minutes

**Applying NLP to the Rapleaf Hackathon Dataset**

**There are 10965 subscribed customers**

**When you use bag of words you get 34 words in paying customers not in nonsubscribed customers.**

**Word model Stats:**

**Bag of words: max frequency, 83031**

**Bigrams:max frequency 11977**

**Bigrams in subscriber urls = 46538**

**Bigrams in nonsubscriber urls:136935**

**Bigrams after removal of stopwords(bigrams with >1k frequency): 46534.**

**Trigrams:**

**Using bigrams you get 231 bigrams in subscribed bag of words which are not in nonsubscribed bag of words. Question is how many of these 10965 customers can be distinguished by these 231 words?**

**If you do trigrams do you get a better signal?**