Midterm

Aaron Politsky

October 30, 2015

1

Let's source our dependencies and load our libraries.

```
library(caret)
library(data.table)
library(doParallel)
library(plyr)
source("EvaluationMetrics.R")
```

And set up parallelization:

```
cl <- makeCluster(detectCores())    # I don't mind using all of my cores
clusterEvalQ(cl, library(foreach))
registerDoParallel(cl)   # register this cluster
```

Now, import our Accident data:

```
set.seed(99) # Gretzky.  Why not?

accidents <- as.data.table(read.table(
  file.path("data", 'Accidents.csv'),
  header=TRUE, sep=',', stringsAsFactors=T))
```

1.1 Choosing a Variable to Predict

MAX_SEV_IR indicates the maximum severity of the accident, and has three levels:

- 0: no injury
- 1: non-fatal injury
- 2: fatal injury

Let's assume we are interested in predicting if there is a severe injury that requires medical attention. One might argue that responders cannot help a dead-on-arrival person, and that fatal injury status should be disregarded. But since MAX_SEV_IR indicates the maximum severity of the accident, it is possible that even if there is a fatality, we do not know if there are non-fatal, serious injuries that require care. (Assume that even with a MAX_SEV_IR:2 accident in which we know there was only one person involved, we are still better safe than sorry and should respond as if all is not lost.) So, let's create another variable called severe.injury which is "no" if MAX_SEV_IR indicates no injury and "yes" if otherwise.

```r
accidents$severe.injury <- accidents$MAX_SEV_IR != 0
accidents[, severe.injury := factor(severe.injury,
                                    levels=c(FALSE, TRUE), labels=c("No", "Yes"))]

# and because we may need to do the same with alcohol
accidents[, ALCHL_I := factor(ALCHL_I,
                                    levels=c(2,1), labels=c("NoAlcohol", "Alcohol"))]

table(accidents$severe.injury, accidents$MAX_SEV_IR)
```

```
##
##            0     1     2
##   No   20721     0     0
##   Yes      0 20996   466
```

1.2 Developing a predictive model

Choosing input variables

At the time we decide to respond to the accident, it is likely that we know the following:

- HOUR_I_R: rush hour or not
- INT_HWY Interstate?
- MAN_COL_I collision, head on, or other?
- PED_ACC_R pedestrian/cyclist involved?
- REL_JCT_I_R accident at intersection/interchange or not?
- REL_RWY_R accident on roadway or not?
- LGTCON_I_R Lighting conditions {day, dark, lighted, etc.}
- SPD_LIM Speed limit, miles per hour
- TRAF_CON_R presence and type of Traffic control device
- TRAF_WAY two-way traffic, divided hwy, one-way road
- VEH_INVL Number of vehicles involved
- WEATHER_R adverse weather presence/type

We may know the following:

- WRK_ZONE construction zone
- NO_INJ_I Number of injuries (may be unclear at time of reporting)
- INJURY_CRASH 1=yes, 0= no (we may not be sure at time of reporting, and this data is ex post)
- PRPTYDMG_CRASH 1=property damage, 2=no property damage (only truly known ex post)
- FATALITIES 1= yes, 0= no

We likely do not know the following, or only know it for certain ex post:

- ALCOHOL_I Alcohol involved = 1, not involved = 2
- ALIGN_I 1 = straight, 2 = curve
- STRATUM_R 1= NASS Crashes Involving At Least One Passenger Vehicle, i.e., A Passenger Car, Sport Utility Vehicle, Pickup Truck Or Van) Towed Due To Damage From The Crash Scene And No Medium Or Heavy Trucks Are Involved. 0=not

- PROFIL_I_R 1= level, 0=other
- SUR_CON Surface conditions (1=dry, 2=wet, 3=snow/slush, 4=ice, 5=sand/dirt/oil, 8=other, 9=unknown). Some of these categories are hyper-local (e.g. sand/dirt/oil)
- MAX_SEV_IR which we profess to not know by construction of this problem.

Let's choose the definitely known ones:

```
predictor.names <-
  c("HOUR_I_R",
    "INT_HWY",
    "MANCOL_I_R",
    "PED_ACC_R",
    "RELJCT_I_R",
    "REL_RWY_R",
    "LGTCON_I_R",
    "SPD_LIM",
    "TRAF_CON_R",
    "TRAF_WAY",
    "VEH_INVL",
    "WEATHER_R"
)
```

Lets convert to factors:

```
for(colname in names(accidents)) {
  accidents[[colname]] <- as.factor(accidents[[colname]])
}
```

Just to sanity-check, the classes of the variables are:

```
sapply(accidents, class)
```

```
##       HOUR_I_R        ALCHL_I        ALIGN_I      STRATUM_R       WRK_ZONE
##       "factor"       "factor"       "factor"       "factor"       "factor"
##       WKDY_I_R        INT_HWY     LGTCON_I_R     MANCOL_I_R      PED_ACC_R
##       "factor"       "factor"       "factor"       "factor"       "factor"
##     RELJCT_I_R      REL_RWY_R     PROFIL_I_R        SPD_LIM       SUR_COND
##       "factor"       "factor"       "factor"       "factor"       "factor"
##     TRAF_CON_R       TRAF_WAY       VEH_INVL      WEATHER_R   INJURY_CRASH
##       "factor"       "factor"       "factor"       "factor"       "factor"
##       NO_INJ_I PRPTYDMG_CRASH     FATALITIES     MAX_SEV_IR severe.injury
##       "factor"       "factor"       "factor"       "factor"       "factor"
```

```
num.samples <- nrow(accidents)
```

Out of the **42,183** samples, the incidence of a severe injury accident is **50.88%**.

We don't have a missing data problem with this data set:

```
sapply(accidents, function(col) sum(is.na(col)))
```

```
##        HOUR_I_R        ALCHL_I        ALIGN_I      STRATUM_R       WRK_ZONE
##               0              0              0              0              0
##        WKDY_I_R        INT_HWY     LGTCON_I_R     MANCOL_I_R      PED_ACC_R
##               0              0              0              0              0
##      RELJCT_I_R      REL_RWY_R     PROFIL_I_R        SPD_LIM       SUR_COND
##               0              0              0              0              0
##      TRAF_CON_R       TRAF_WAY       VEH_INVL      WEATHER_R   INJURY_CRASH
##               0              0              0              0              0
##        NO_INJ_I PRPTYDMG_CRASH     FATALITIES     MAX_SEV_IR  severe.injury
##               0              0              0              0              0
```

Let's split a Validation set out from the Training data, for use in estimating OOS performance:

```
valid_proportion <- 1 / 3
valid_indices <- createDataPartition(
  y=accidents$severe.injury,
  p=valid_proportion,
  list=FALSE)

accidents.valid <- accidents[valid_indices, ]
accidents <- accidents[-valid_indices, ]
```

Just to sanity-check that the data sets have been split representatively by **caret**: the Yes incidences in the Training and Validation sets are **50.88** and **50.88**, respectively.

What about level-counts?

```
sapply(predictor.names, function(colname) {
  table(accidents[[colname]])
})
```

```
## $HOUR_I_R
##
##     0     1
## 16032 12090
##
## $INT_HWY
##
##     0     1     9
## 24032  4073    17
##
## $MANCOL_I_R
##
##     0     1     2
##  9073   616 18433
##
## $PED_ACC_R
##
##     0     1
## 26975  1147
##
## $RELJCT_I_R
##
```

```
##     0     1
## 12451 15671
##
## $REL_RWY_R
##
##     0     1
##  6634 21488
##
## $LGTCON_I_R
##
##     1     2     3
## 19558  3291  5273
##
## $SPD_LIM
##
##    5   10   15   20   25   30   35   40   45   50   55   60   65   70   75
##    2   15  121  175 2823 2451 5701 2859 4360 1113 4413 1115 1822  962  190
##
## $TRAF_CON_R
##
##     0     1     2
## 18023  5760  4339
##
## $TRAF_WAY
##
##     1     2     3
## 15974 10840  1308
##
## $VEH_INVL
##
##     1     2     3     4     5     6     7     8     9    10    23
##  8606 16842  2178   390    75    15     9     5     1     0     1
##
## $WEATHER_R
##
##     1     2
## 24133  3989
```

Our training data has no 10-vehicle crashes and few observations with greater than 6 cars. Let's collapse all levels above 6 into "GreaterThan6" for crashes with vehicles greater than 6.

```r
accidents[VEH_INVL %in%
            levels(accidents$VEH_INVL)[as.integer(levels(accidents$VEH_INVL) ) > 6],
          "VEH_INVL" := "GreaterThan6"]
accidents <- droplevels(accidents)

accidents.valid[
  VEH_INVL %in%
    levels(accidents.valid$VEH_INVL)[as.integer(levels(accidents.valid$VEH_INVL)) > 6],
  "VEH_INVL" := "GreaterThan6"]
accidents.valid <- droplevels(accidents.valid)

# check
```

```r
sapply(predictor.names, function(colname) {
  any(table(accidents[[colname]])==0)
})
```

```
##   HOUR_I_R    INT_HWY MANCOL_I_R  PED_ACC_R RELJCT_I_R  REL_RWY_R
##      FALSE      FALSE      FALSE      FALSE      FALSE      FALSE
## LGTCON_I_R    SPD_LIM TRAF_CON_R   TRAF_WAY   VEH_INVL  WEATHER_R
##      FALSE      FALSE      FALSE      FALSE      FALSE      FALSE
```

```r
sapply(predictor.names, function(colname) {
  any(table(accidents.valid[[colname]])==0)
})
```

```
##   HOUR_I_R    INT_HWY MANCOL_I_R  PED_ACC_R RELJCT_I_R  REL_RWY_R
##      FALSE      FALSE      FALSE      FALSE      FALSE      FALSE
## LGTCON_I_R    SPD_LIM TRAF_CON_R   TRAF_WAY   VEH_INVL  WEATHER_R
##      FALSE      FALSE      FALSE      FALSE      FALSE      FALSE
```

Classification Models

Let's train 3 types of classification models: a Random Forest, a Boosted Trees model and a Logistic Regression.

```r
caret_optimized_metric <- 'logLoss'   # equivalent to 1 / 2 of Deviance

caret_train_control <- trainControl(
  classProbs=TRUE,             # compute class probabilities
  summaryFunction=mnLogLoss,   # equivalent to 1 / 2 of Deviance
  method='repeatedcv',         # repeated Cross Validation
  number=5,                    # 5 folds
  repeats=2,                   # repeats
  allowParallel=TRUE)
```

```r
B <- 500

system.time(
  rf_model <- train(
    x=accidents[, predictor.names, with=FALSE],
    y=accidents$severe.injury,
    method='parRF',     # parallel Random Forest
    metric=caret_optimized_metric,
    ntree=B,            # number of trees in the Random Forest
    nodesize=30,        # minimum node size set small enough to allow for complex trees,
    # but not so small as to require too large B to eliminate high variance
    importance=TRUE,    # evaluate importance of predictors
    keep.inbag=FALSE,
    trControl=caret_train_control,
    tuneGrid=NULL)
)
```

```r
B <- 1000

system.time(
  boost_model <- train(
    x=accidents[, predictor.names, with=FALSE],
    y=accidents$severe.injury,
    method='gbm',       # Generalized Boosted Models
    metric=caret_optimized_metric,
    verbose=T,
    trControl=caret_train_control,
    tuneGrid=expand.grid(
      n.trees=B,              # number of trees
      interaction.depth=5,   # max tree depth,
      n.minobsinnode=100,     # minimum node size
      shrinkage=0.01))        # shrinkage parameter, a.k.a. "learning rate"
)
```

```r
log_reg_model <-
  glm(severe.injury ~ .,
      data=data.frame(accidents[, c(predictor.names, "severe.injury"), with=F]),
      family = binomial()
)
```

Justification Based on Out of Sample Performance

We'll now evaluate the OOS performances of these 3 models on the Validation set to select a model we think is best:

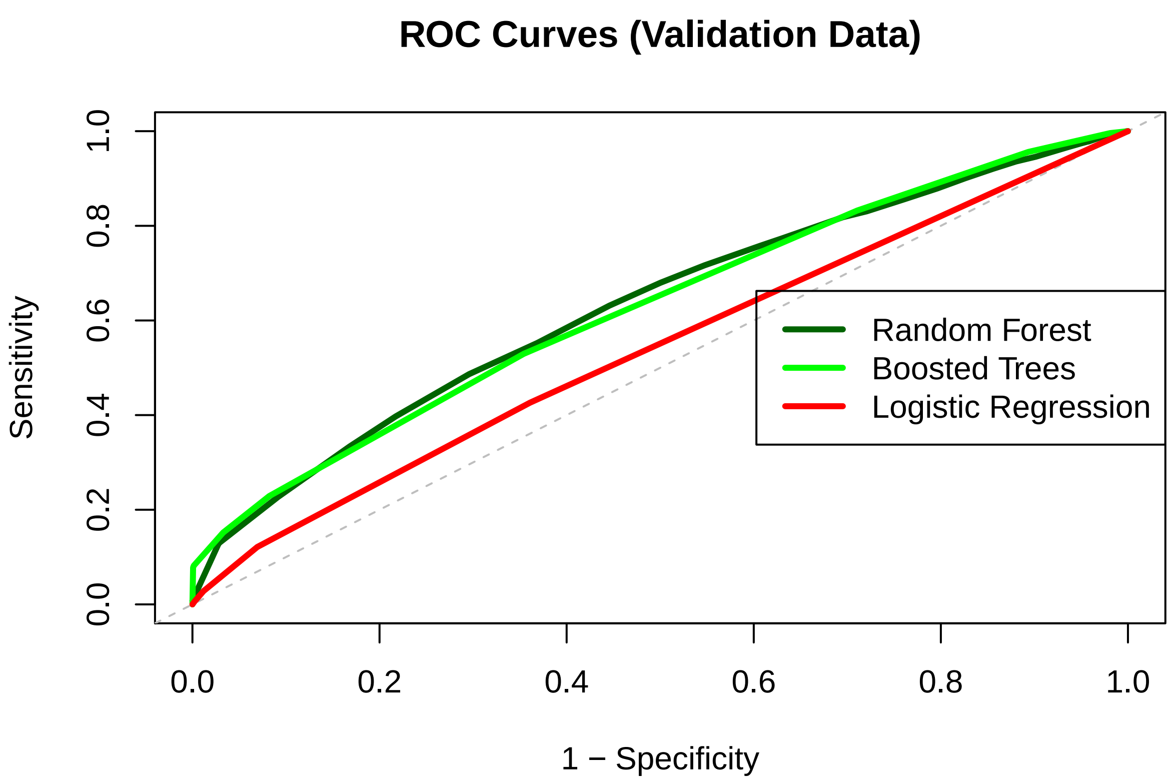
```r
# Check for missing Validation levels:
sapply(predictor.names, function(colname) {
    all(levels(accidents.valid[[colname]]) %in% levels(accidents[[colname]]))
})
```

```
##   HOUR_I_R    INT_HWY MANCOL_I_R  PED_ACC_R RELJCT_I_R  REL_RWY_R
##       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
## LGTCON_I_R    SPD_LIM TRAF_CON_R   TRAF_WAY   VEH_INVL  WEATHER_R
##       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
```

```r
low_prob <- 1e-06
high_prob <- 1 - low_prob
log_low_prob <- log(low_prob)
log_high_prob <- log(high_prob)
log_prob_thresholds <- seq(from = log_low_prob, to = log_high_prob, length.out = 100)
prob_thresholds <- exp(log_prob_thresholds)

rf_pred_probs <- predict(rf_model, newdata = accidents.valid[, predictor.names,
    with = FALSE], type = "prob")
rf_oos_performance <- bin_classif_eval(rf_pred_probs$Yes, accidents.valid$severe.injury,
    thresholds = prob_thresholds)

boost_pred_probs <- predict(boost_model, newdata = accidents.valid[, predictor.names,
    with = FALSE], type = "prob")
```

```r
boost_oos_performance <- bin_classif_eval(boost_pred_probs$Yes, accidents.valid$severe.injury,
    thresholds = prob_thresholds)

log_reg_pred_probs <- predict(log_reg_model, newdata = data.frame(accidents.valid[,
    c(predictor.names, "severe.injury"), with = FALSE]), type = "response")

log_reg_oos_performance <- bin_classif_eval(log_reg_pred_probs, accidents.valid$severe.injury,
    thresholds = prob_thresholds)
```

```r
plot(x=1 - rf_oos_performance$specificity,
     y=rf_oos_performance$sensitivity,
     type = "l", col='darkgreen', lwd=3,
     xlim = c(0., 1.), ylim = c(0., 1.),
     main = "ROC Curves (Validation Data)",
     xlab = "1 - Specificity", ylab = "Sensitivity")
abline(a=0,b=1,lty=2,col=8)
lines(x=1 - boost_oos_performance$specificity,
      y=boost_oos_performance$sensitivity,
      col='green', lwd=3)
lines(x=1 - log_reg_oos_performance$specificity,
      y=log_reg_oos_performance$sensitivity,
      col='red', lwd=3)
legend('right', c('Random Forest', 'Boosted Trees', 'Logistic Regression'),
   lty=1, col=c('darkgreen', 'green', 'red'), lwd=3, cex=1.)
```



Boosting and RandomForest look great, but Logistic didn't do so well.

1.3: Measuring the effect of alcohol on the severity of the accident.

Simple Logit

We could estimate a logit model regressing severity on alcohol, controlling for all other predictors so as to isolate the ceteris-paribus effect. The coefficient on the alcohol involvement dummy will represent an estimate of the difference in log of probability that the accident is severe given alcohol involvement.

(Here, using a no-intercept model, because we haven't defined the base category for any of our other predictors.)

```
logit.alcohol.effect.model <- glm(severe.injury ~ . - 1, data = accidents[,
    c(predictor.names, "ALCHL_I", "severe.injury"), with = F], family = binomial())

exp(logit.alcohol.effect.model$coefficients["ALCHL_IAlcohol"])
```

```
## ALCHL_IAlcohol
##       1.422888
```

Alcohol involvement increases the probability of a severe injury accident by **42.29%**, controlling for our other predictors.

(Note: Having taken a quarter of Program Evaluation at Harris, I feel the need to mention that if we really wanted to measure the effect of alcohol involvement on severity of injuries, we should consider a potential outcomes framework. I would probably use a propensity score matching technique: weighing the non-alcohol group by p/(1-p), then calculate the difference in their severity means. One problem with propensity scores is that you have to get the functional forms right if you're using a regression, but now we have tree methods, which can sort out those nasty, nonlinear functional forms and get a great p-score prediction. Exciting, but I'm moving on to number 2.)

2 Tabloid, Revisited

Let's get ready to fit our models.

```
library(caret)
library(data.table)
library(doParallel)
library(plyr)
source("EvaluationMetrics.R")
```

```
cl <- makeCluster(detectCores())   # I don't mind using all of my cores
clusterEvalQ(cl, library(foreach))
registerDoParallel(cl)   # register this cluster
```

```
# download data and read data into data.table format
y_var_name <- 'purchase'
y_classes <- c('not_responsive', 'responsive')

X_var_names <- c(
  'nTab',
  'moCbook',
  'iRecMer1',
  'llDol',
```

```r
  'propSpec',
  'recW4',
  'moShoe',
  'nWoApp',
  'nMen'
  )
column_classes <- c(
  purchase='integer',
  nTab='numeric',
  moCbook='numeric',
  iRecMer1='numeric',
  llDol='numeric',
  propSpec='numeric',
  recW4='numeric',
  moShoe='numeric',
  nWoApp='numeric',
  nMen='numeric'
  )


tabloid <- fread(
  file.path("data", 'tabdat9n20.csv'),
  colClasses=column_classes)
tabloid[ , purchase := factor(purchase,
                              levels=c(0, 1), labels=y_classes)]

num.samples <- nrow(tabloid)
```

Tidying

```r
sapply(tabloid, function(col) sum(is.na(col)))
```

```
## purchase     nTab  moCbook iRecMer1 propSpec    recW4   moShoe   nWoApp
##        0        0        0        0        0        0        0        0
##     nMen    llDol
##        0        0
```

No missing data.

Out of the **20,000** samples, the incidence of marketing-responsive purchase is **2.46%**. Note that this creates a "**skewed classes**" problem: one of the classes of cases (here the "responsive" class) is significantly rarer than the other.

2.1

Since our data is likely in good shape, but we almost certainly don't have an idea of functional form, I bet tree methods would work well. And, since question 2.2 asks about importance, we can use random forests and boosting, and good old logit to assess variable importance. Let's also fit a lasso, while we're at it.

First, split up our train and validation data:

```r
set.seed(99) # I should probably watch Gretzky on youtube this December.

valid_proportion <- 1 / 3
valid_indices <- createDataPartition(
  y=tabloid$purchase,
  p=valid_proportion,
  list=FALSE)

tabloid_valid <- tabloid[valid_indices, ]
tabloid_train <- tabloid[-valid_indices, ]
```

Just to sanity-check that the data sets have been split representatively by **`caret`**: the responsive incidences in the Training and Validation sets are **2.45** and **2.46**, respectively.

Fitting our Models

Let's train 3 types of classification models: a Random Forest, a Boosted Trees model, our all-X logistic regression, and a Lasso.

```r
caret_optimized_metric <- 'logLoss'   # equivalent to 1 / 2 of Deviance

caret_train_control <- trainControl(
  classProbs=TRUE,             # compute class probabilities
  summaryFunction=mnLogLoss,   # equivalent to 1 / 2 of Deviance
  method='repeatedcv',         # repeated Cross Validation
  number=5,                    # 5 folds
  repeats=3,                   # repeats
  allowParallel=TRUE)
```

```r
B <- 500

rf_model <- train(
  x=tabloid_train[, X_var_names, with=FALSE],
  y=tabloid_train$purchase,
  method='parRF',     # parallel Random Forest
  metric=caret_optimized_metric,
  verbose=TRUE,
  ntree=B,            # number of trees in the Random Forest
  nodesize=30,        # minimum node size set small enough to allow for complex trees,
                      # but not so small as to require too large B to eliminate high variance
  importance=TRUE,    # evaluate importance of predictors
  keep.inbag=TRUE,
  trControl=caret_train_control,
  tuneGrid=NULL)
```

```r
B <- 1000

system.time(
boost_model <- train(
  x=tabloid_train[, X_var_names, with=FALSE],
  y=tabloid_train$purchase,
  method='gbm',       # Generalized Boosted Models
```

```r
  metric=caret_optimized_metric,
  verbose=TRUE,
  trControl=caret_train_control,
  tuneGrid=expand.grid(
    n.trees=B,              # number of trees
    interaction.depth=c(4,10),    # max tree depth,
    n.minobsinnode=100,     # minimum node size
    shrinkage=c(0.2,0.01)))        # shrinkage parameter, a.k.a. "learning rate"
)
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
## Loaded glmnet 2.0-2
```

```r
x.train <- as.matrix(tabloid_train[,-("purchase"), with=F])
x.valid <- as.matrix(tabloid_valid[,-("purchase"), with=F])
y.train <- tabloid_train$purchase
```

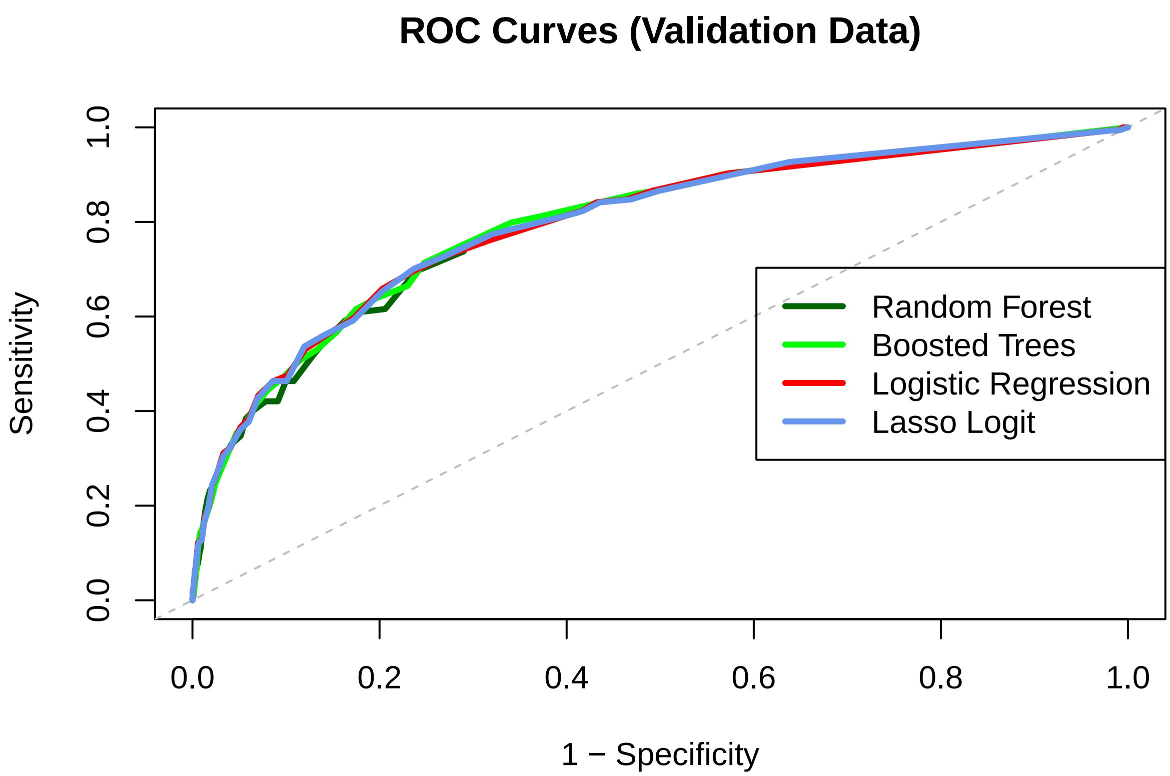
```r
lasso_model <- cv.glmnet(x.train, y.train, nfolds = 5, parallel = T, family="binomial", alpha=1)
coefs <- coef(lasso_model$glmnet.fit, s=lasso_model$lambda.min)
```

```r
log_reg_model <- train(
  x=tabloid_train[, X_var_names, with=FALSE],
  y=tabloid_train$purchase,
  preProcess=c('center', 'scale'),
  method='plr',    # Penalized Logistic Regression
  metric=caret_optimized_metric,
  trControl=caret_train_control,
  tuneGrid=expand.grid(
    lambda=0,      # weight penalty parameter
    cp='aic'))     # complexity parameter (AIC / BIC)
```

Let's evaluate them on the validation data:

```r
low_prob <- 1e-6
high_prob <- 1 - low_prob
log_low_prob <- log(low_prob)
log_high_prob <- log(high_prob)
log_prob_thresholds <- seq(from=log_low_prob, to=log_high_prob, length.out=100)
prob_thresholds <- exp(log_prob_thresholds)

rf_pred_probs <- predict(
  rf_model, newdata=tabloid_valid[ , X_var_names, with=FALSE], type='prob')
rf_oos_performance <- bin_classif_eval(
  rf_pred_probs$responsive, tabloid_valid$purchase, thresholds=prob_thresholds)

boost_pred_probs <- predict(
  boost_model, newdata=tabloid_valid[ , X_var_names, with=FALSE], type='prob')
boost_oos_performance <- bin_classif_eval(
  boost_pred_probs$responsive, tabloid_valid$purchase, thresholds=prob_thresholds)
```

```r
log_reg_pred_probs <- predict(
  log_reg_model, newdata=tabloid_valid[, X_var_names, with=FALSE], type='prob')
log_reg_oos_performance <- bin_classif_eval(
  log_reg_pred_probs$responsive, tabloid_valid$purchase, thresholds=prob_thresholds)

lasso_pred_probs <- predict(
  lasso_model$glmnet.fit, x.valid,
  type="response", s=lasso_model$lambda.min
)
lasso_oos_perf <- bin_classif_eval(
  lasso_pred_probs, tabloid_valid$purchase, thresholds = prob_thresholds)


plot(x=1 - rf_oos_performance$specificity,
     y=rf_oos_performance$sensitivity,
     type = "l", col='darkgreen', lwd=3,
     xlim = c(0., 1.), ylim = c(0., 1.),
     main = "ROC Curves (Validation Data)",
     xlab = "1 - Specificity", ylab = "Sensitivity")
abline(a=0,b=1,lty=2,col=8)
lines(x=1 - boost_oos_performance$specificity,
      y=boost_oos_performance$sensitivity,
      col='green', lwd=3)
lines(x=1 - log_reg_oos_performance$specificity,
      y=log_reg_oos_performance$sensitivity,
      col='red', lwd=3)
lines(x=1 - lasso_oos_perf$specificity,
      y=lasso_oos_perf$sensitivity,
      col='cornflowerblue', lwd=3)
legend('right', c('Random Forest', 'Boosted Trees', 'Logistic Regression', 'Lasso Logit'),
   lty=1, col=c('darkgreen', 'green', 'red', 'cornflowerblue'), lwd=3, cex=1.)
```



They all are pretty close. Hard to say what to choose.

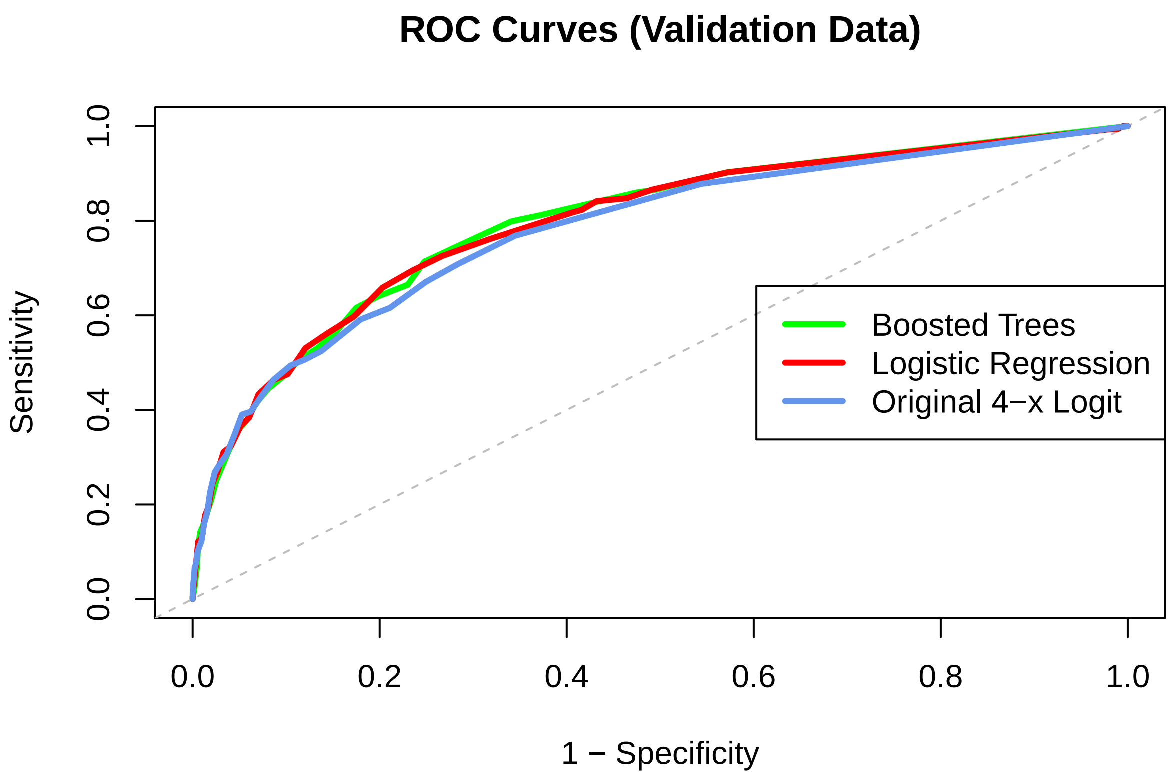
Let's fit our original, 4-predictor model:

```
limited.log_reg_model <- train(
  x=tabloid_train[, X_var_names[1:4], with=FALSE],
  y=tabloid_train$purchase,
  preProcess=c('center', 'scale'),
  method='plr',     # Penalized Logistic Regression
  metric=caret_optimized_metric,
  trControl=caret_train_control,
  tuneGrid=expand.grid(
    lambda=0,       # weight penalty parameter
    cp='aic'))      # complexity parameter (AIC / BIC)
```

How does it compare to the 9-predictor model, and our boosted model?

```
limited.log_reg_pred_probs <- predict(
  limited.log_reg_model, newdata=tabloid_valid[, X_var_names[1:4], with=FALSE], type='prob')
limited.log_reg_oos_performance <- bin_classif_eval(
  limited.log_reg_pred_probs$responsive, tabloid_valid$purchase, thresholds=prob_thresholds)

plot(x=1 - boost_oos_performance$specificity,
     y=boost_oos_performance$sensitivity,
     type = "l", col='green', lwd=3,
     xlim = c(0., 1.), ylim = c(0., 1.),
     main = "ROC Curves (Validation Data)",
     xlab = "1 - Specificity", ylab = "Sensitivity")
abline(a=0,b=1,lty=2,col=8)
lines(x=1 - log_reg_oos_performance$specificity,
```

```
      y=log_reg_oos_performance$sensitivity,
      col='red', lwd=3)
lines(x=1 - limited.log_reg_oos_performance$specificity,
      y=limited.log_reg_oos_performance$sensitivity,
      col='cornflowerblue', lwd=3)
legend('right', c('Boosted Trees', 'Logistic Regression', 'Original 4-x Logit'),
   lty=1, col=c('green', 'red', 'cornflowerblue'), lwd=3, cex=1.)
```



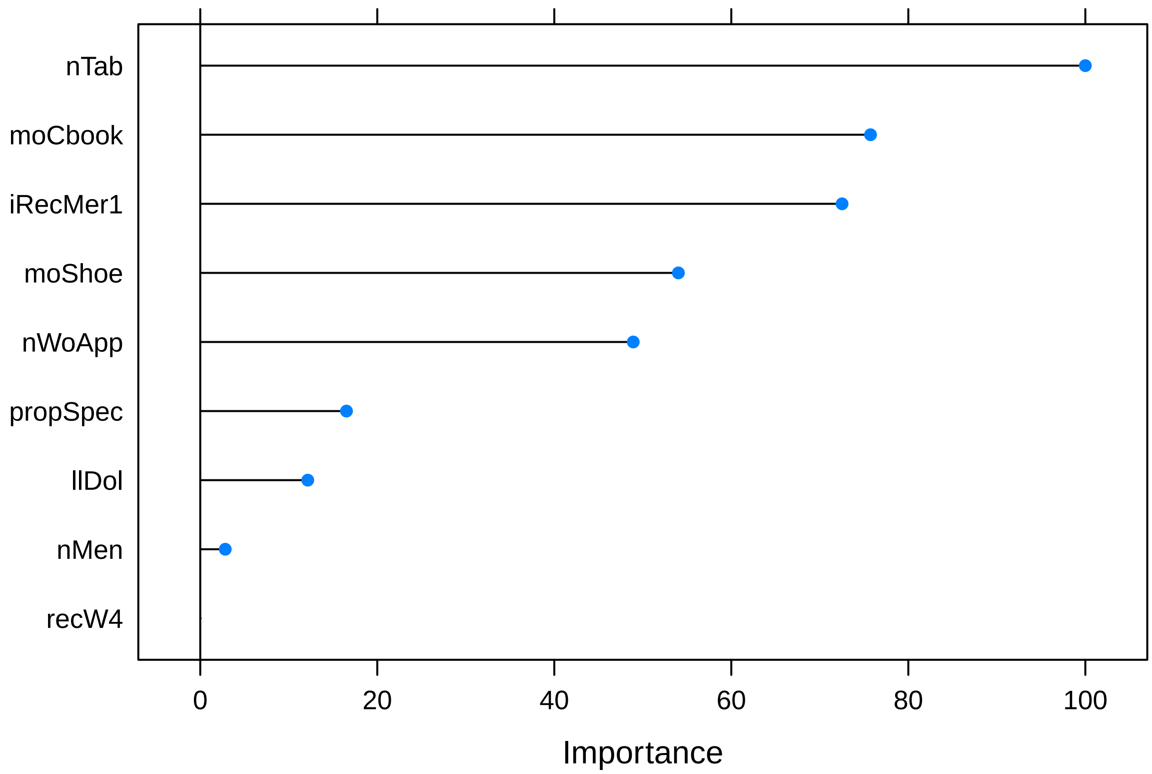
2.2

Based on the plots, the old logit with the 4 predictors doesn't seem much worse than the new logit, and it's also close in performance to the boosted trees model. Sure, the new model is a little better, but I would conclude that the 9-predictor model doesn't add much over the 4-predictor model, so I'm not sure the 5 new predictors are that useful.

Still, which model is best will depend on what threshold we choose in our business application.

The important variables, according to Random Forest, are:

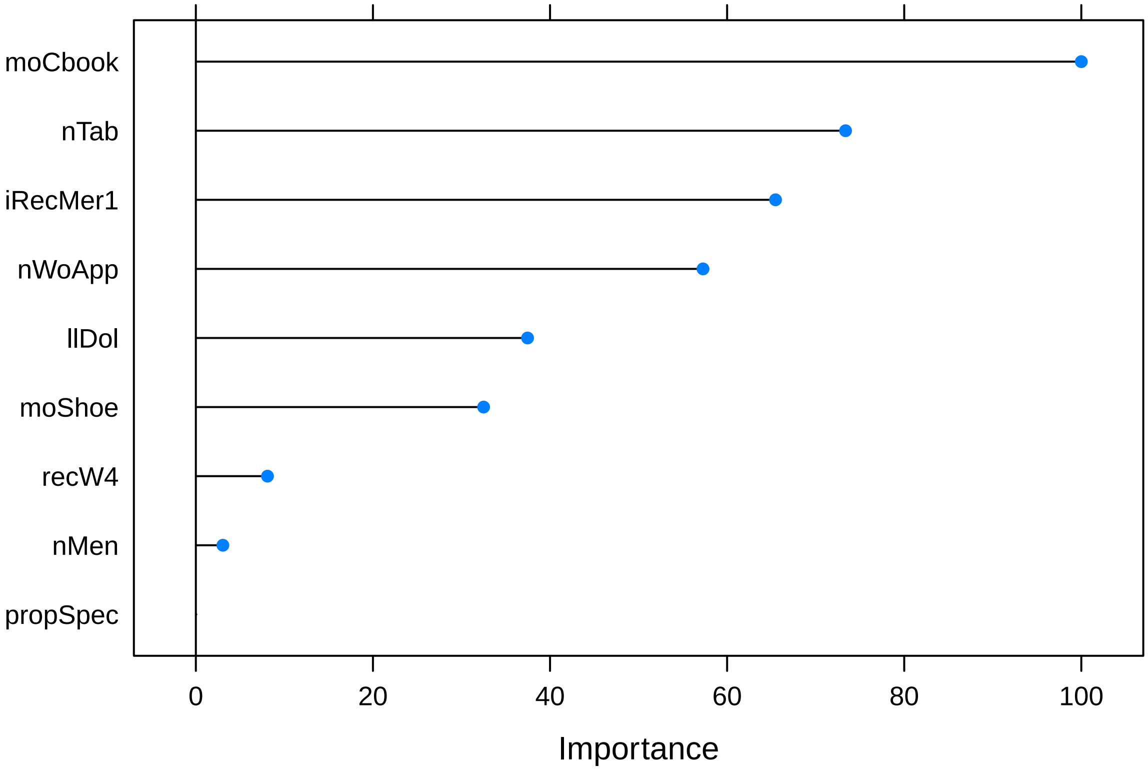
```
rf.imp <- varImp(rf_model)
plot(rf.imp)
```



Three of the original are still on top, but moShoe, nWoApp, and propSpec jump ahead of llDol.

According to Boosting:

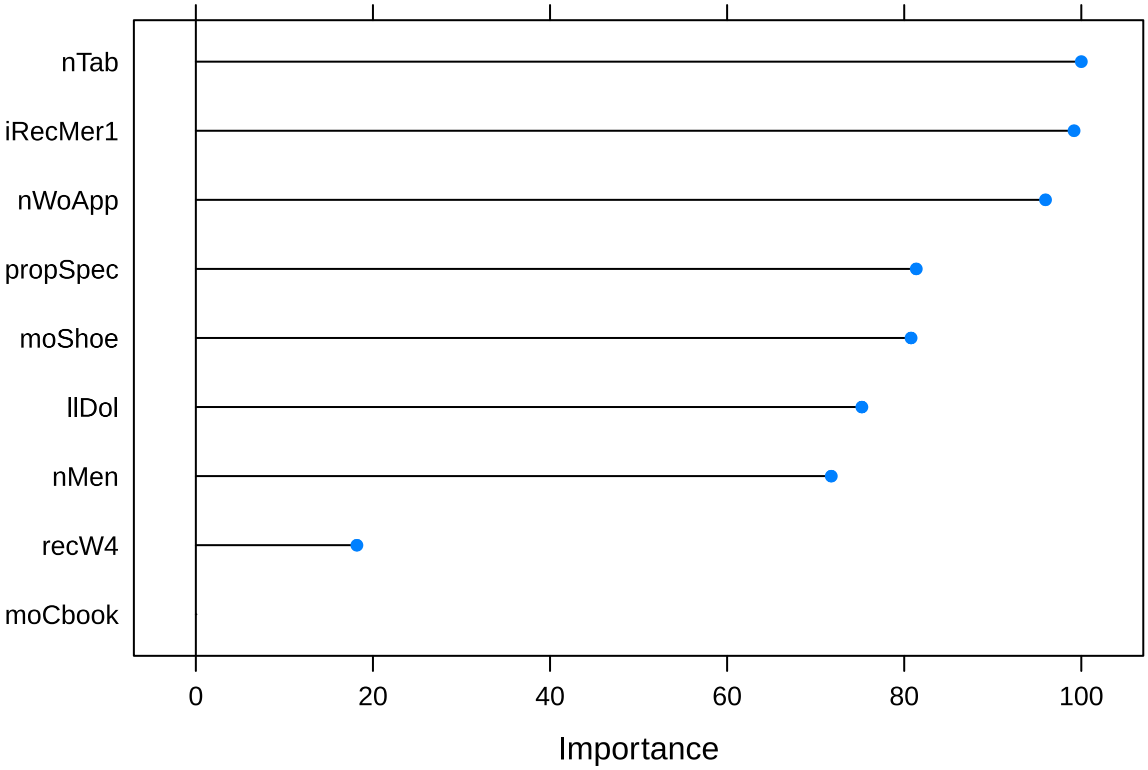
```
boost.imp <- varImp(boost_model)
plot(boost.imp)
```



Here, MoCbook rules. The original 3 are still on top, but nWoApp leaps llDol.

According to our 9-predictor logit:

```
logit.imp <- varImp(log_reg_model)
plot(logit.imp)
```



Here, many are seemingly important. However, moCbook falls to the bottom, and llDol drops a bit.

Overall, the important variables seem to be pretty consistent with the original 4-x model's important variables, with some slight disagreement/shifting.

Let's average their relative importance across these three models:

```
importance <- c()
for(var in rownames(rf.imp$importance)) {
  importance[[var]] <- (rf.imp$importance[var,"responsive"] +
                        boost.imp$importance[var,] +
                        logit.imp$importance[var,"responsive"])
}
importance[order(importance, decreasing = T)]
```

```
##      nTab  iRecMer1    nWoApp   moCbook    moShoe     llDol  propSpec
## 273.37662 237.16604 202.15556 175.73220 167.29407 124.82565  97.88189
##      nMen     recW4
##  77.64250  26.27899
```

I would say that the following are the most important.

```
names(importance[order(importance, decreasing = T)])[1:6]
```

```
## [1] "nTab"     "iRecMer1" "nWoApp"   "moCbook"  "moShoe"   "llDol"
```

Note: We could also examine the lasso results, but we'd have to do some pre-scaling, and then tell lasso not to scale. Then the coefficients would indicate their effects, relative to other predictors. Another time, perhaps.

2.3

We need to decide whom to target based on their probability of responding and a probability cutoff s, then send promotions costing $0.80 to these people. If they respond, we get $40 (or $39.20, net).

Economic theory suggests that we lower our threshold until the expected marginal profit equals the marginal cost for a given threshold. This is exactly the same condition for when expected utility equals 0.

$$E(U) = -0.80(1 - s) + 39.20s = 0$$

Implying $s = .8/40 = 0.02$

Let's calculate the profit we get from each model given s = 0.02.

```
s <- 0.02 # Threshold

profit <- function(probs) {
  dotarget <- probs > s
  print(confusion <- table(dotarget, tabloid_valid$purchase))
  revenue <- 40*confusion["TRUE","responsive"]
  costs <- 0.8*sum(confusion["TRUE",])
  profit <- revenue - costs
}

profit.list <- sapply(list(rf_pred_probs$responsive,
                           boost_pred_probs$responsive,
                           log_reg_pred_probs$responsive,
                           lasso_pred_probs), function(probs) profit(probs))
```

```
##
## dotarget not_responsive responsive
##    FALSE           5799         88
##    TRUE             704         76
##
## dotarget not_responsive responsive
##    FALSE           4889         47
##    TRUE            1614        117
##
## dotarget not_responsive responsive
##    FALSE           4758         45
##    TRUE            1745        119
##
## dotarget not_responsive responsive
##    FALSE           4753         45
##    TRUE            1750        119
```

```
names(profit.list) <- c("rf", "boost", "logit reg", "lasso")

profit.list
```

```
##        rf     boost logit reg     lasso
##    2416.0    3295.2    3268.8    3264.8
```

Using our most profitable model (Boosting) our profit is $**3,295.20**.

3

Prompt:

We would like to target some subset of the huge number of visitors to our main retail web page with a new special offer. Instead of the normal early May special offer of a discounted flower bouquet for Mom, we've decided to offer select customers a 30% discount on any electric razor purchase from our stock.

3.1

Intro

For either offer, we can consider the costs and the benefits.

The costs to display an offer are effectively zero, so let's consider that portion 0. This implies that we will show every visitor one of the offers (unless we predict that showing them an offer will drive them away from the site, which we believe to be unlikely). The cost of the visitor taking the special offer (vs. buying at retial price) is the discount amount of the offer, times, of course, the quantity they bought at that discount. (Let's assume the quantity for either offer is capped at 1, to keep things simple.)

The benefit of a visitor taking an offer has two components: the profit (or loss) due to the sale of the offered good (at the discounted price), plus the profit we can earn from add-on or future sales. The former could be positive, zero, or negative, in the case that we discount such that we expect the offer to be a loss-leader. The latter, let's assume, is positive.

Generally speaking, we want to choose the offer that has the highest expected net benefit, be it from them purchasing the offered product, or from this motivating them to purchase other goods, or both.

For either offer, we will predict several probabilities: the probability that the person takes the offer at the discounted price, and the probability that the person makes *other purchases given* that the visitor has been offered a bouquet.

For simplicity, let's assume our profit from selling the discounted, offered product to be its net profit. Let's also assume we can estimate the expected profit of other purchases (below, $pi_{other purchases}) which would be another, netsted expected value framework of its own, conditional on all sorts of things.

Our expected value calculation for offer O would look like:

$$E(O) = P(Takes_O) \times \pi_{taken,O} + P(Other\ Purchases \mid offered_O) \times \pi_{other\ purchases}$$

We would base our decision on whether to show a given visitor the Bouquet offer or the Razor offer based on whichever offer's Expected value is higher, and, greater than zero (in the case that loss-leading is expected to fail to generate other purchases. In this case, we show no offer.)

3.2

To solve this problem, we need to build models to predict two probabilities:

- the probability of a given visitor to take the offer
- the probability of a given visitor to make other purchases given that they have been shown an offer.

As I mentioned above, we would also need to construct a model for the Expected profit of the other purchases. This could be a function of the type of offer shown.

3.3

The following assumes we can identify a visitor (or household) using cookies or some IP-related information. In other words, it's not as if every visit is anonymous.

We likely have data on visitors who have visited in the past. Since we offer this every May, we can see how prior offers (bouquets) may have influenced their behavior. But since the razor offer is new, we don't have prior offer history data.

What we can think about are whether the person is likely to take either offer. Which product interests them more? Prior purchase and visiting data (what products they purchased, or merely viewed), may serve as predictors for these. We should feel more confident about predicting behavior of heavy-history visitors, but for new or light-history visitors, we could build models matching their behavior (propensity score matching?) to better-known customers, and then predicting based on these similarities.

This should give us a sense of whether they'd bite either offer, how they respond having been shown an offer, and how much we can expect them to buy given either offer.

We would want to routinely update our training set with new behavior as we gain experience with our visitors.

Some concerns: it may be tricky to assess whether the offer was what motivated them to act. They may have bought the offered good, regardless, and even at a higher price (always-takers). On the other hand, they may never have taken the offer (never-takers). We want to focus on the effects of those who were moved on the margin ("compliers"). To help learn, we could run random experiments to try to measure the efficacy of an offer, but I digress.