Data Science and Strategic Pricing

Instructor: Jacob LaRiviere, Director of Economics and Data Science at Amazon

Email: [jlarivi1@uw.edu](mailto:jlarivi1@uw.edu), [lghhager@uw.edu](mailto:lghhager@uw.edu) (TA)

# Course Assignments & Reading

Course assignments should be knitted Rmarkdown file and turned in at the start of class unless otherwise noted. Feel free to work in groups but everyone is required to turn in their own work with answers written in your own words. In both calculations and complex ideas, write down each step of logic used in reaching your conclusion. Keep in mind that in most cases a good answer is one precise sentence; quality is heavily favored over quantity. This will be graded on a full credit, half credit and no credit basis. All work must be typed

Discussion questions do not need be written out ahead of time. Students will be called on, potentially at random, to add their insight. This part of class will contribute heavily to your course participation grade.

**Week 6, due Nov 10**

**Assignment to be turned in.** Please turn in your Rmarkdown file (as HTML) with answers embedded.

In this assignment we’re going to use a regression tree to break stores into bins or types. This will facilitate pricing differently for each store type. Our target variable will be sales weighted price.

1. Create a sales weighted price for orange juice by store.
   1. You’ll first need to create actual sales (call it “Q”) instead of log sales for the weighting and put it into your dataframe.
   2. You can use the weighted.mean() function for each store-week combination in the dplyr library. If works like this:

Df1 <- ddply(dataframe, c('var1','var2'),function(x) c(weighted\_mean = weighted.mean(x$price,x$Q)))

Here 'var1','var2' are the two identifiers for the variables to create a weighted average by (store and week in our case), the function takes as an input “x” (which is the dataframe specified beforehand) then creates weighted\_mean of x$price weighted by x$Q. You’ll then need to merge this back in to the original dataframe.

You can also calculate the weighted average manually.

1. Now use oj$weighted\_price as the LHS variable in a regression tree to predict differences in sales weight prices with store demographics as RHS variables. Note that you’ll only need to do for a single brand since weighted price and sociodemographic variables are identical across brands within a store.
   1. There are a couple libraries you’ll need which you’ll see in the lecture notes (rpart, maptree, etc.)
   2. There are two main pieces of code:

dataToPass<-oj[,c("weighted\_mean","AGE60","EDUC","ETHNIC","INCOME","HHLARGE","WORKWOM","HVAL150","SSTRDIST","SSTRVOL","CPDIST5","CPWVOL5")]

#The above creates a dataframe from the existing one (with weighted mean merged back in) which will then be passed into rpart (tree partitioning algorithm).

fit<-rpart(as.formula(weighted\_mean ~ .),data=dataToPass,method="anova",cp=0.007)

#This is the code which will fit the tree.

* 1. Play around with a couple different complexity parameters to get a feel for the data

draw.tree(fit) #This draws the tree

* 1. Choose three different leaves to group stores into based upon what explains sales weighted price.
     1. Assign each store to one of these leaves (we used this code previously).

dataToPass$leaf = fit$where #This assigns leaves to observations.

1. Estimate the own price elasticities for each one of the store buckets/leaves using the preferred specification:

reg\_int <- glm(logmove~log(price)\*brand\*feat, data=oj\_leaf\_L)

* 1. Now estimate cross price elasticities jointly with own price elasticities. This means you must create a dataframe which has the prices of all types of OJ at the store. (e.g., you should be able to use the Trop\_Cross code you’ve used previously.
  2. You’ll also have to run 3 separate regressions for each leaf for a total of nine regressions.

reg\_int <- glm(logmove\_D~log(price\_D)\*feat + log(price\_T)\*feat + log(price\_MM)\*feat, data=oj\_leaf\_L\_D)

In this example, we are investigating the own and cross price elasticities for Dominick’s brand (D) within leaf L.

* + 1. Save the coefficients for each leaf in a 3x3 matrix. The diagonals will be own price elasticities and the off diagonals will be cross price elasticities.
    2. There will be a unique 3x3 matrix for each leaf.
    3. The 3x3 matrices WON’T be upper triangular because we’re estimating three unique regressions for each leaf.
  1. Comment on any differences between own and cross price elasticities by leaf.

1. Now let’s use the elasticities to think about pricing differentials.
   1. In the leaf with the highest own-price elasticities, what should the markups be relative to the other leafs?
   2. How do cross-price elasticities vary with the highest versus lowest own price elasticity leafs?
      1. What does this imply about differences in markups within high versus low elasticity stores across brands?
      2. Can you say anything about what this means for the timing of sales? Should they occur at the same or different times across stores?
2. We’re going to do some basic exploration with xgboost.
   1. Install the package xgboost and library it.
   2. Divide the data into a training set (80% of the data) and a hold-out set (20% of the data).
   3. We’re going to train a model to predict logmove. To do this, we’re going to create a training and testing matrix that we can give to the package to do cross validation on.
      1. Use the xgb.DMatrix function to create a train and test matrix. This function takes arguments “data” (must be a matrix) and “label” (the outcome, logmove in our case).
      2. Use the xgb.cv function to do 5-fold cross-validation on our training data. We’ll just use the defaults for most of the hyperparameters. A few useful arguments:
         1. nfold: number of folds for cross-validation
         2. nrounds: number of training rounds (generally, we want this to be a very large number since we don’t want to be artificially stopped short of achieving a minimum)
         3. early\_stopping\_rounds: if this argument is set, XGBoost will stop training if the testing error does not improve in whatever number the user puts here. This should be our stopping criterion (as opposed to hitting nrounds)
         4. print\_every\_n: if you set this to, say, 100, XGBoost will report its progress every 100 iterations, instead of each iteration.
         5. Important note: we’re not actually cross-validating or setting any of the hyperparameters that make XGBoost a powerful algorithm. If you’re curious about what other parameters you can set, inspect the documentation for this function or for the function xgboost.
      3. Report the training RMSE (root mean squared error) and testing RMSE from the best model. How does this compare to previous models that we’ve used (remember that you should square this to get MSE)?
      4. Use the xgboost function to train a model on the full training data using our one cross-validated hyperparameter (the number of training iterations). To do this, find the best iteration of the cross validated model and set that as nrounds for the xgboost function.
      5. Use the predict command (the same way that we do in regression) and your testing xgb.DMatrix to assess the fit of the model on the test data. How does the MSE compare to the MSE from cross-validation? How does it compare to prior models?