Introduction

Uplift models (or heterogeneous treatment effect models) is a branch of machine learning with the goal of maximizing some objective function by assigning an optimal treatment to each individual. In practice it is often ambiguous objective function should be. For instance, when applied to a marketing campaign the objective function might be to increase user behavior (of some sort) while maintaining some level of costs. The trade-offs between an increase in user behavior and increase in costs are often not defined a-priori.

The mr\_uplift package in python builds and evaluates trade-offs among several different response variables of interest. It estimates something akin to a [Production Possibility Frontier](https://en.wikipedia.org/wiki/Production%E2%80%93possibility_frontier) for uplift models. Using these estimated PPF curves make it easier for stakeholders to determine the trade-offs associated with using the model.

In this post I use the GOTV dataset to construct a hypothetical problem where we are interested in increasing voter turnout while being cognizant of the costs involved. I will highlight some functionality of the mr\_uplift package that I have found to be helpful in practice. In particular:

* Encoding treatment variables that share common features
* Evaluating trade-offs among different response variables
* Ability to use only a subset of the treatment should the need arise
* Variable Importance

Finally, in the appendix I will compare a loss function designed to maximize these trade-off curves directly with a mean squared error approach.

GOTV Data and Preprocessing

The GOTV dataset is a randomized experiment where ~45% of individuals received one of four letters (treatments) in the mail designed to increase voter turnout in the subsequent election. The remaining 55% receive no letter and are designated the control.

Using individual level data (demographic features and previous voting behavior) we are tasked with building an uplift model to increase voter turnout by assigning one of the 5 treatments (I consider the control to be a treatment in addition to four possible letters) to each individual.

For this example I include an additional assumption where sending one of the four letters has a cost of 1 unit (I will discuss the interpretability of this below). Assuming a constant cost for mailing is reasonable in this particular case but in other settings the cost may vary with the treatment and/ or the user.

Below I discuss pre-processing the data into the three necessary variable groupings: the treatment variables, the response variables, and the explanatory variables.

Encoding the Treatment Variables

In most uplift models treatments are assigned a dummied out and transformed into a series of columns an indicator for each treatment. However, this ignores the information that can be shared between the treatments.

The GOTV treatments have a nested structure where a subsequent treatments includes a previous treatments attributes and includes another one. For instance; The base mailing letter is called ‘civic duty’ and notifies recipient to “DO YOUR CIVIC DUTY AND VOTE!”. The subsequent ‘hawthorne’ mailing letter includes the ‘civic duty’ information in addition to a note that his voting is public information. Similarly the ‘self’ letter builds off the ‘hawthorne’ letter and the ‘neighbors’ letter builds off the ‘self’ letter.

We can include this nested information of treatments as shown below. Note that encoding the treatment information this way may or may not be helpful in this example. It is meant to be demonstrative of the capabilities of mr\_uplift package. In practice I have found this way encoding to be very helpful where there are several ordered choices for treatments.

#py\_install('mr\_uplift')  
**library**(hete)  
**library**(dplyr)  
**library**(reticulate)  
  
data(gotv)  
gotv = data.frame(gotv)  
  
t = data.frame(matrix(0,nrow(gotv),4))  
colnames(t) = c("civic\_duty", "hawthorne","self","neighbors")  
gotv$treatment = as.character(gotv$treatment)  
  
t[(gotv$treatment!='Control'), c("civic\_duty")] = 1  
t[which(gotv$treatment %**in**% c("Hawthorne","Self","Neighbors") ), c( "hawthorne")] = 1  
t[which(gotv$treatment %**in**% c("Self","Neighbors") ), c( "self")] = 1  
t[which(gotv$treatment %**in**% c("Neighbors") ), c( "neighbors")] = 1  
  
*#no-letter treatment is considered a row of zeros while the neighbors treatment is a row of ones*  
print(unique(t))

## civic\_duty hawthorne self neighbors  
## 1 1 0 0 0  
## 3 1 1 0 0  
## 6 0 0 0 0  
## 17 1 1 1 0  
## 20 1 1 1 1

Encoding the Response Variables

Here we have two responses: whether someone voted in subsequent election and whether they had a cost or not. The cost is defined to be one if an individual received one of the 4 letters.

y = data.frame( voted = (gotv[,c('voted')]=='Yes')\*1, cost = t[,1])  
print(head(y))

## voted cost  
## 1 0 1  
## 2 0 1  
## 3 1 1  
## 4 1 1  
## 5 1 1  
## 6 0 0

Encoding the Explanatory Variables

Here we just need to make sure we convert the categorical features into numeric types. Nothing too interesting here.

gotv[,'p2004'] = tolower(gotv[,'p2004'])  
x = as.data.frame((gotv[,c('g2000','g2002','p2000','p2002','p2004')]=='yes')\*1)  
x[,'sex'] = (gotv[,'sex'] == 'female')\*1  
x = cbind(x, gotv[,c('hh\_size','age','n\_general','n\_primary')])  
print(head(x))

## g2000 g2002 p2000 p2002 p2004 sex hh\_size age n\_general n\_primary  
## 1 1 1 0 1 0 0 2 65 3 1  
## 2 1 1 0 1 0 1 2 59 3 1  
## 3 1 1 0 1 0 0 3 55 3 1  
## 4 1 1 0 1 0 1 3 56 3 1  
## 5 1 1 0 1 0 1 3 24 3 1  
## 6 0 0 0 0 0 0 3 25 1 0

Building a MRUplift Model

With pre-processing of the responses, treatments, and explanatory variables finished we can now use the reticulate package to pass the data into python and built a MRUplift model. The MRUplift code automatically grid-searches and makes train/test split to build trade-off curves.

Here I use the ‘optimized\_loss’ functionality that attempts to maximize the PPF curve directly instead of using an MSE loss. In the appendix I compare using this loss with an MSE loss.

After the model is built we can create ‘erupt\_curves’ on the test dataset to see how the model performs and the trade-offs between costs and voter turnout are. A matrix of ‘objective\_weights’ is inputed into this function determining the relative weights of response variables to maximize. Here I set the ‘cost’ weight to be -1 while varying the ‘voting’ weight between 0 and 30 (30 was chosen arbitarily). For each of these ‘objective\_weights’ the package calculates the treatment that maximizes the expected weighted sum of response variables.

**import** numpy **as** np  
**import** pandas **as** pd  
**from** mr\_uplift.mr\_uplift **import** MRUplift

t = r.t  
y = r.y  
x = r.x  
  
param\_grid\_mse = dict(num\_nodes=[8,32],   
 dropout=[ .1,.2,.3],   
 activation=['relu'],   
 num\_layers=[1,2,3],   
 epochs=[25],   
 batch\_size=[512],  
 copy\_several\_times = [10,1],  
 alpha = [.99,.75,.5,0])  
  
uplift\_model\_optim = MRUplift()  
  
*#Using the 'optimized\_loss' functionality*   
*#uplift\_model\_optim.fit(x, y, t, param\_grid = param\_grid\_optim, n\_jobs = 1, optimized\_loss = True)*  
*#uplift\_model\_optim.save('~/Github/mr\_upliftR/models/mr\_uplift\_gotv\_mult\_tmt2')*  
uplift\_model\_optim.load('~/Github/mr\_upliftR/models/mr\_uplift\_gotv\_mult\_tmt2')

objective\_weights = np.concatenate([np.array([(x) for x in range(30)]).reshape(-1,1), -np.ones(30).reshape(-1,1)], axis = 1)

erupt\_curves\_optim, dists\_optim = uplift\_model\_optim.get\_erupt\_curves(objective\_weights = objective\_weights)

Initial Model Results

We can now plot the results of the MRUplift package in R using ggplot2. Below we can see two types of charts.

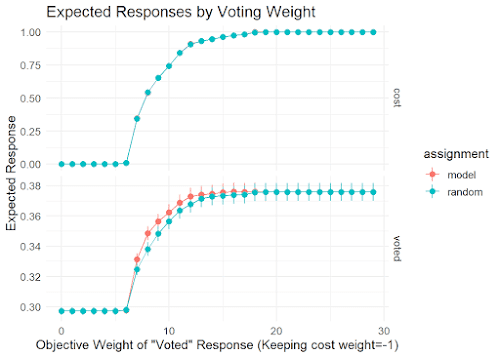
This first chart shows the expected responses (along with 95% CI) for a given set of objective weights signified by the model assignment. As the weight on voting increases from zero to 30 we see an increase in both voting activity and and increase in costs. There is also a ‘random’ assignment ERUPT curve – for each objective weights this ‘shuffles’ the treatment assignment. The difference between the model vs random assignment shows how well the model is learning the heterogeneous treatment effect (HETE). Since there is no HETE effects in the costs by construction these two curves will be equal for that response variable.

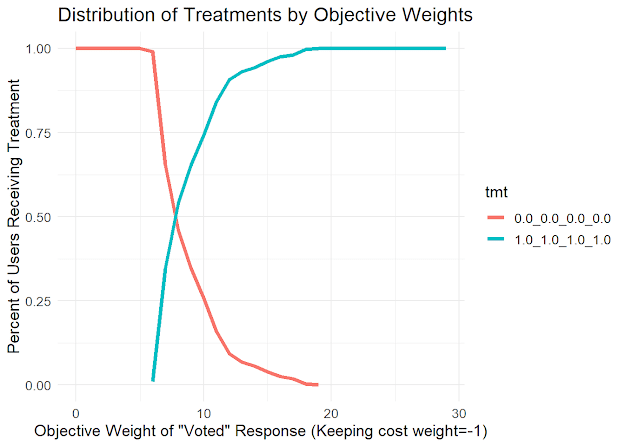
The second chart shows the distribution of treatments for each objective weights. Note that users receive no mail when we set the objective function to be 0 for voting and -1 or costs. As the relative weights changes more users receive the treatment vector (1,1,1,1) which corresponds to the ‘neighbors’ treatment.

**library**(ggplot2)  
erupt\_curves = py$erupt\_curves\_optim   
dists = py$dists\_optim

erupt\_curves[,'weights'] = as.numeric(gsub(',-1.0','',erupt\_curves[,'weights']))  
dists[,'weights'] = as.numeric(gsub(',-1.0','',dists[,'weights']))  
  
ggplot(erupt\_curves, aes(x = weights, y = mean, color = assignment,group = assignment))+geom\_line()+facet\_grid(response\_var\_names~., scales = "free\_y")+  
 geom\_pointrange(aes(ymin=mean-2\*std, ymax=mean+2\*std))+theme\_minimal()+  
 xlab('Objective Weight of "Voted" Response (Keeping cost weight=-1)')+  
 ylab('Expected Response')+  
 ggtitle('Expected Responses by Voting Weight')+  
 theme(text = element\_text(size=13))

ggplot(dists, aes(x = weights , y = percent\_tmt, group = as.factor(tmt), colour =tmt))+geom\_line(size = 1.5)+theme\_minimal()+  
 xlab('Objective Weight of "Voted" Response (Keeping cost weight=-1)')+  
 ylab('Percent of Users Receiving Treatment')+  
 ggtitle('Distribution of Treatments by Objective Weights')+  
 theme(text = element\_text(size=13))

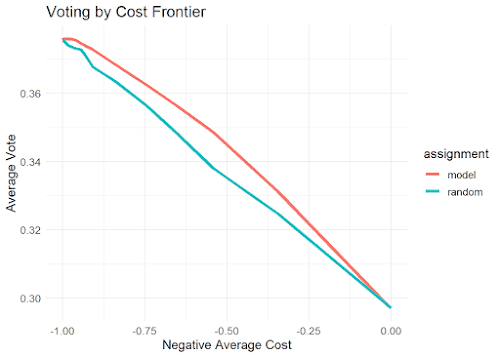




In order to see the trade-offs more clearly we can plot the first set of charts against each other. This shows a costs vs voting curve. Below, I have set cost to be negative conforming to traditional PPF curves that say we want to be up and to the right.

erupt\_curves\_cost = subset(erupt\_curves, response\_var\_names == 'cost')  
erupt\_curves\_voted = subset(erupt\_curves, response\_var\_names == 'voted')  
  
colnames(erupt\_curves\_voted)[1:2] = paste0('voted\_',colnames(erupt\_curves\_voted)[1:2])  
colnames(erupt\_curves\_cost)[1:2] = paste0('cost\_',colnames(erupt\_curves\_cost)[1:2])  
  
ppf = merge(erupt\_curves\_voted, erupt\_curves\_cost, by = c('weights','assignment'))  
  
ggplot(ppf, aes(x = -cost\_mean, y = voted\_mean, group = assignment, colour = assignment, label = weights))+geom\_line(size = 1.5)+theme\_minimal()+  
 xlab('Negative Average Cost')+  
 ylab('Average Vote')+  
 ggtitle('Voting by Cost Frontier')+  
 theme(text = element\_text(size=13))

Using these charts we can decide where we want to be on these charts in a few ways. One way would be to determine that the benefit of 1 additional vote is worth 10 units of cost. This corresponds to an increase in costs of .75 unis while an increase in voting by 0.065 units.



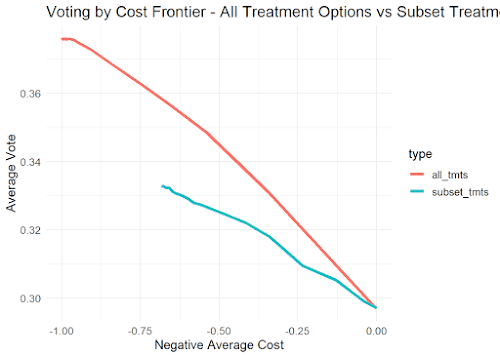
Alternatively, if we had a predetermined budget of .75 per user we can determine that the optimal set of weights correspond to a weight of 10 for voting.

Hey can you not use that treatment?

After presenting initial results a stakeholder might be hesitant to use the neighbor treatment due to the strong wording in the letter. What if we didn’t use that treatment? We can specify which treatments to use in the get\_erupt\_curves functionality shown below.

temp\_tmts = np.array([[0.0,0.0,0.0,0.0],  
 [1.0,0.0,0.0,0.0],  
 [1.0,1.0,0.0,0.0],  
 [1.0,1.0,1.0,0.0]])  
  
erupt\_curves\_optim\_4\_tmts, dists\_optim\_4\_tmts = uplift\_model\_optim.get\_erupt\_curves(objective\_weights = objective\_weights, treatments = temp\_tmts)

We can compare the trade-offs of using the model with all treatments or subsetted treatments. A graph showing the trade-offs is shown below but with code removed for brevity.



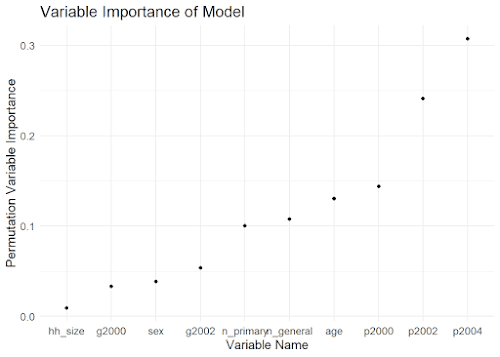
It appears the next strongest option ‘self’ option. However, using this instead of the ‘neighbors’ treatment shows dramatically decreased model performance. Whether adverse effects of using that treatment are outweighed by the measured benefits is something the stakeholder will have to decide.

What Variables are Important?

After the model is built and if we are ok using all treatments we can now look into what are important features for the model. One use the permutation\_varimp functionality shown below.

varimp = uplift\_model\_optim.permutation\_varimp(weights = objective\_weights[10])

varimp = py$varimp  
varimp[,'var\_names'] = factor(varimp[,'var\_names'], varimp[,'var\_names'][order(varimp[,'permuation\_varimp\_metric'])])  
ggplot(varimp, aes(x = var\_names, y = permuation\_varimp\_metric))+geom\_point()+theme\_minimal()+  
 xlab('Variable Name')+  
 ylab('Permutation Variable Importance')+  
 ggtitle('Variable Importance of Model')+  
 theme(text = element\_text(size=13))



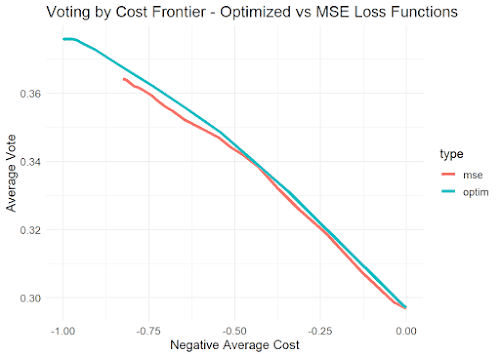
The above chart suggests that previous primary voting behavior is most important while household size is not.

Conclusions

This post went over a hypothetical uplift model problem with the GOTV dataset and the mr\_uplift package. It went over a few unique features of the mr\_uplift package that I have found to be useful in practice. Please check out the package and feel free to contribute!

Appendix Comparing MSE vs Optimized Loss

Fitting uplift models are generally hard because we are interested in estimating the interaction between the treatment(s) and other explanatory variables. Below is a short comparison between using this loss vs a standard MSE error model.



One can see that the frontier of the optimized model is up and to the right of the MSE frontier. This means that the optimized loss function is ‘better’ in the sense that it achieves a higher voter rate for a given cost or a lower cost for a given voter rate.