

MARKETING CAMPAIGN OPTIMIZATION: PROFIT MODELING

BY SEMINAR APPLIED PREDICTIVE MODELING (SS19) – ASMIR MUMINOVIC, LUKAS KOLBE | AUGUST 12, 2019

Marketing Campaign Optimization

Causal Inference in Profit Uplift Modeling #### Authors: Asmir Muminovic, Lukas Kolbe

Motivation

The global spending on advertising amounts to more than 540 billion US dollars for 2018 only, and the spending for 2019 is predicted to reach over 560 billion US dollars. The marketing spendings have continuously increased since 2010 [17], and this trend does not seem to brittle in the near future. Although marketing is such an integral part of most businesses, marketers often fail to maximize the profitability of their marketing efforts. Traditionally, this issue was tackled via utilization of so-called “Response” models [18], which targets the most likely to buy customers. This model has a big downside, since it disregard causality. We effectively do not know if the targeted customers would have bought anyways, is annoyed by the targeting, or does not react at all. Targeting these customers would result in unnecessary costs. We should therefore select customers who will buy because of the campaign, that is, those who are likely to buy if targeted, but unlikely to buy otherwise [1]. This could be achieved with uplift modeling, since it measures the treatment effect. Additionally, customers have different spendings, therefore revenue uplift modeling seems more viable than standard conversion modeling. But, since targeting customers can come with a price in form of vouchers which effectively reduce the profits, Profit Uplift Modeling is what truly optimizes the profitability of online marketing campaigns.

Datasets For this project, we used four different datasets. All of them contain data from marketing campaigns in online shops, each having a treatment and control group and both conversion and revenue/spend as target variables. All our models focus on modeling the treatment effect as additional revenue gained by the treatment, thus focusing on the continuous spending variable as the target. The binary conversion variable is used for stratification of the data.

All datasets show an inherent treatment effect, i.e. the spending of customers in the treatment group is higher than spending of customers in the control group. This treatment effect is what our analysis tries to isolate and assign to single individuals/sessions in our data.

All datasets show an imbalance in the spending class, meaning that the cases where the spending is >0 are quite rare. This is an important caveat which might influence model learning, since the cases where people actually spend money are both the most interesting and the most rare.

Kevin Hillstrom Data The first dataset was the well known Hillström dataset published by Kevin Hillström in 2008. It contains well cleaned data which revolves around an email campaign that was sent out to a group of male and female customers of an online shop. As taken from the Blog description, the dataset contains 64,000 customers who last purchased within twelve months. The customers were involved in an e-mail test.

- 1/3 were randomly chosen to receive an e-mail campaign featuring Mens merchandise.
- 1/3 were randomly chosen to receive an e-mail campaign featuring Womens merchandise.
- 1/3 were randomly chosen to not receive an e-mail campaign.

In our case, we do *not* differentiate between the two treatments. Instead, we try to optimize the targeting decision within the population supplied, regardless of mens or womens treatment.

During a period of two weeks following the e-mail campaign, results were tracked. Hillstrom suggests that the user tries to find out whether the mens or the womens campaign was more successful.

The data and its full description can be found here: [https://blog.minethatdata.com/2008/03/minethatdata-e-mail-analytics-and-data.html]

The Code for reproduction of the results can be found in our GitHub Repo:
[https://github.com/lukekolbe/hu_apa/tree/master/R%20Code/hillstrom]

Fashion A, Fashion B, Books&Toys The chair of Information Systems at the Humboldt University of Berlin provided us with three real-world e-commerce datasets from a cooperating partner. These datasets, namely FashionA, FashionB and Books&Toys contain 1.000.000, 700.000 and 150.000 customer sessions, respectively. The data was collected over a one year period. All datasets are similar in structure, since they all possess the same variables.

All three sets share the same 93 features, which can be grouped into campaign related variables, time counting variables, session status variables (interaction counter, time variables, the target and treatment variables) [.....]

While the data has the same inherent features, the context of the campaign might be different each time. Also, the actual nature of the campaigns showed to be quite different:

The treatments were defined by their Campaign Unit (Currency or Percent), the Campaign Value (either a fixed discount amount or a discount percentage) and the Campaign Minimum Order Value (MOV). The latter defined whether the user had to reach a certain shopping cart value before becoming eligible for treatment. This is very interesting, as it means that being in the treatment group does not necessarily mean everyone treated gets a discount. Instead, the discount is conditional on reaching the MOV. For any cost and profit analysis, this is very important.

In **Fashion A**, a user had to reach 125€ before getting to claim the 20€ discount proposed by the campaign. The goal here seems to have been an upselling effect: the high MOV reduces the number of people who become eligible for the discount. For everyone else, the assumption is that showing them the option of a discount motivates them to shop. In many cases, they might find items they like, but not for more than the MOV. Yet still they might commit to the cart items they added and check out. In such cases, there might be a positive effect of the campaign on the amount spent, without causing any cost. Some ~64.000 rows in the Data had a differing campaign: 50€ MOV and a 5€ campaign value. These lines were removed because they had a lower average treatment effect than those in the majority campaign and were vastly outnumbered.

```
mean(f_a$checkoutAmount[f_a$controlGroup==0&f_a$campaignValue==500]) -  
mean(f_a$checkoutAmount[f_a$controlGroup==1&f_a$campaignValue==500])
```

0.250630

```
mean(f_a$checkoutAmount[f_a$controlGroup==0&f_a$campaignValue==2000]) -  
mean(f_a$checkoutAmount[f_a$controlGroup==1&f_a$campaignValue==2000])
```

0.4990753

This left us with a completely homogeneous campaign structure in Fashion A:

- Campaign Value of 20€
- Campaign Minimum Order Value of 125€

In **Fashion B**, the treatment was similar, but with a lower coupon value (5€) and lower MOV (50 and 30€). In **Books&Toys**, there was a mix of fixed discounts and percentage on final cart sum.

In all of these datasets, the target variable *checkoutAmount* (the final amount paid) has the cost of the treatment already internalized. This is relevant for the calculation of profits later on.

Because of the differences between the datasets, we opted to treat every dataset individually regarding feature selection and cleaning.

Data Cleaning Since the Hillstrom Data is in a very clean state as-is, no major transformation of the data was done. Only factor variables were dummified in order to make them compatible with the causalBoosting algorithm. Also, the treatment vector, a factor with 3 levels (No E-Mail, Womens E-Mail, Mens E-Mail) was transformed into a binary treatment vector.

The cleaning process for Fashion A, Fashion B and Books&Toys followed the same steps, respectively. The primary goal of our thorough feature reduction was computational performance: as we had to train a set of 5 models on each of the datasets, we could not afford to run models for extended periods of time. This is why we chose to narrow down the set of features while retaining the maximum possible information value.

An outlier treatment was not done

Feature creation Firstly, we transformed the unusual controlGroup variable into a treatment variable, effectively reversing the scale. This is needed as most of our models rely on a binary treatment vector.

Secondly, we created an “eligibility” variable, indicating whether a person i’s checkoutAmount plus campaignValue puts them over the Minimum Order Value (MOV): $(\text{checkoutAmount}_i + \text{campaignValue}_i | \text{treatment}_i = 1) \geq \text{campaignMov}_i$

After calculating the eligibility, we computed the ExpectedDiscount, which indicates how much we expect each person’s treatment (given eligibility) to cost:

$(\text{ExpectedDiscount}_i | \text{eligibility} = 1 \ \& \ \text{campaignUnit} = \text{“CURRENCY”}) = \text{campaignValue}_i$
 $(\text{ExpectedDiscount}_i | \text{eligibility} = 1 \ \& \ \text{campaignUnit} = \text{“PERCENT”}) = (\text{checkoutAmount}_i / (1 - \text{campaignValue})) - \text{checkoutAmount}$

1. Remove unnecessary lines Some features such as trackerKey, campaignId, campaignTags were not useful for analysis. These features were deleted. A variable called label was also not needed, since it contained a transformed binary target variable (see Gubela 2017) that was likely intended in order to do machine learning with conventional methods. As we model treatment effects directly, this was not necessary (see above).

Some other deleted features include factors with only one level, variables with only NA values and variables with an NA ratio of more than 85%.

2. NA treatment After deleting the features with highest NA ratios, we undertook a median imputation on time variables (as setting time counters such as timeSinceLastVisit to 0 would imply a very recent visit while the opposite is likely the case).

For other counters or binary variables, we set NA values to 0.

3. Correlated features In order to further reduce the features, we conducted a correlation analysis and removed features with high correlation. The threshold was set to 0.75. This is rather low, the default value of the used findCorrelation function of the ‘caret’ package is 0.9. But since we needed to reduce features for the costly models to be deployed, we set the threshold to a lower level. For all datasets (barring Hillstrom), this removed between 11 and 12 features, leaving about 60 features for possible training plus targets and campaign specific features.

```
1 #identifying pairs of highly correlated variables and removing one of each pair
2 #build a correlation matrix without necessary variables (otherwise the method will kick "treatment and others")
3
4 exclude.vars <- c("converted", "checkoutAmount", "treatment",
5                 "eligibility", "ExpectedDiscount", "aborted",
6                 "confirmed", "campaignMov", "campaignValue", "campaignUnit")
7
8 correlationMatrix <- cor(f_a[, -which(names(f_a) %in% c(exclude.vars))])
9 highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75, names=TRUE, verbose=TRUE, exact=TRUE)
10
11 f_a <- f_a[, -which(names(f_a) %in% c(highlyCorrelated))]
```

corr_f_a.R view raw (https://gist.github.com/lukekolbe/497a9c4a6d2851028c06230292825ffd/raw/3044e57b6d44dda28058021c5251676fa17171/corr_f_a.R)
(https://gist.github.com/lukekolbe/497a9c4a6d2851028c06230292825ffd#file-corr_f_a-r) hosted with ❤ by GitHub (<https://github.com>)

Exemplary correlation handling

Feature Selection: Adjusted Net Information Value (NIV) To further reduce the amount of features, we deployed the NIV function from Leo Guelmann's 'Uplift' Package [8].

The net information value is derived from the well known Information Value (IV) and optimized for purposed of uplift modeling. It is computed as

$$NIV = 100 \times \sum_{i=1}^G (P(x = i | y = 1)^T \times P(x = i | y = 0)^C - P(x = i | y = 0)^T \times P(x = i | y = 1)^C) \times NWOE_i$$

where

$$NWOE_i = WOE_i^T - WOE_i^C$$

The adjusted net information value is computed as follows:

1. Take B bootstrap samples and compute the NIV for each variable on each sample
2. Compute the mean of the NIV

(*NIVmean*)

and standard deviation of the NIV

(*NIVsd*)

for each variable over all the B bootstraps

3. The adjusted NIV for a given variable is computed by adding a penalty term to the mean NIV:

$$NIVmean - \frac{NIVsd}{\sqrt{B}}$$

```
1 n <- names(f_a)
2 f_niv_fa <- as.formula(paste("converted ~", paste("trt(treatment) +"),
3                           paste(n[!n %in% exclude.vars], collapse = " + ")))
4
5 fa_niv <- niv(f_niv_fa, f_a, subset=NULL, na.action = na.pass, B = 10, direction = 1,
6             nbins = 10, continuous = 4, plotit = TRUE)
7
8 f_a_niv <- fa_niv$niv
```

niv_gist.R view raw (https://gist.github.com/lukekolbe/9b5a09776fc547b6b235700ac61aa890/raw/9161f4ee16bbc8441c9742d3ce58bc71d1bd8c83/niv_gist.R) (https://gist.github.com/lukekolbe/9b5a09776fc547b6b235700ac61aa890#file-niv_gist-r) hosted with ❤ by GitHub (<https://github.com>)

From the results of the NIV, the top 25 features were selected for training.

Stratification Since the datasets at hand were quite different in size, we opted to stratify them in a way that leaves us with training (and test) sets of similar size. The challenge was to find a way to keep the balance of the data: each set should have a ratio of treatment and control group members very close to the initial dataset. At the same time, we wanted to make sure that the ratio of conversions (actually buying something) was also kept similar.

The package ‘splitstackshape’ allows for such stratification across multiple variables and was used to receive our training and test sets. The ratios by which to split the data depended on the size of the dataset.

```
1 # Splitting into train & test set -----
2 set.seed(101010, kind = "Mersenne-Twister", normal.kind = "Inversion", sample.kind = "Rounding")
3
4 strat_split <- stratified(f_a, c("treatment", "converted"), 0.667, bothSets=TRUE)
5 f_a.train <- as.data.frame(strat_split[[1]])
6 f_a.validate <- as.data.frame(strat_split[[2]])
7
8 strat_validate_split <- stratified(f_a.validate, c("treatment", "converted"), 0.7, bothSets=TRUE)
9 f_a.test <- as.data.frame(strat_validate_split[[2]]) #the actual test data
10
11 # second part of stratification (TRAINING DATA) -----
12 strat_trainsplit_small <- stratified(f_a.train, c("treatment", "converted"), 0.85, bothSets=TRUE)
13 f_a.train_small <- as.data.frame(strat_trainsplit_small[[2]]) #the actual training data
```

view raw
(https://gist.github.com/lukekolbe/dd9bf80fdbf74e0f9844d0e6c40a70de/raw/a08814355253bc635edb6d2a64305dbb76d01f04/sample_stratification_f_a.R)
sample_stratification_f_a.R (https://gist.github.com/lukekolbe/dd9bf80fdbf74e0f9844d0e6c40a70de#file-sample_stratification_f_a-r) hosted with ❤ by GitHub (<https://github.com>)

Model Selection

Approaches to Uplift modeling There are three main approaches in the uplift literature, the Class Transformation approach, the Two-Model approach and the direct modeling approach[7]. The Class Transformation method is popular but yields mixed results, while some argue that the performance of this method is typically outperformed by the Two-Model approach [6], others say the exact opposite[7]. Since, there is no clear opinion on this matter, we decided to use the Two-Model approach as a baseline model, which finds wide application as such.

Two-Model approach The Two-Model approach, how the name suggests, uses two predictive models, one using the treatment group

data and the other using the control group data. This makes it possible to use a variety of machine learning models, which is the biggest advantage of this approach. The prediction of these two models get subtracted from each other to generate the uplift predictions. However, this could be a major drawback, for two reasons. First, since we employ two models, we end up having two error terms and secondly and more important both models focus on predicting the outcome isolated from each other, instead of focusing on the differences between the two groups, which could lead to missing the "weak" uplift signal [7].

Direct Model approach There are various approaches to model uplift directly, for example Logistic Regression, K-Nearest-Neighbors, Support Vector Machines. But the by far most frequently used ones are in the family of tree-based models, such as Decision Trees [10], Random Forest [11], Boosting[12] and Bayesian Additive Regression Trees [13]. According to Gubela et. al tree-based models are among the top performers [14].

Model Selection As mentioned above, tree-based models are usually outperforming other models. This is why we opted to use those mentioned in the Direct Model approach section along with a Ridge Regression Two-Model approach as our baseline model.

Honest Causal Tree Since the Causal Tree is the base learner of most of our utilized models, we will quickly summarize its functionality. Causal Trees do not differ much from traditional Decision Trees, the only difference is instead of estimating a mean outcome in each leaf it estimates the average treatment effect. This is achieved by using a modified Mean Squared Error criterion for splitting. At each leaf it partitions the data in such way that it maximizes the heterogeneity in treatment effects between groups and penalizes a partition that creates variance in leaf estimates [9]. The heterogeneity is usually measured with the Kullback-Leibler divergence [4].

$$KL(P|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

But how does the tree make sure, that the heterogeneity is not due to noise in the data. The solution is to make the Causal Tree **"honest"** by splitting our training data into two subsamples, where one is functioning as a subsample to find splits and the other is used to estimate the effects employing the splits found in the splitting subsample [9]. Each case of the estimation subsample is dropped down the tree until it falls into a leaf. This procedure is basically mimicing Cross-Validation. Cross-Validation per se cannot be applied in uplift settings, because of the fundamental problem of Causal Inference. For every individual, only one outcome is observed, never both at the same time.

Honest Causal Forest The Honest Causal Forest is an just like any other Random Forest, an ensemble of Honest Causal Trees. The idea is the same, instead of fitting a single Honest Causal Tree, which can be noisy we fit multiple Causal Honest Trees at the same time, effectively The "Random Forest" part functions the same way as a standard Random Forest (e.g., resampling, considering a subset of predictors, averaging across many trees, etc.)[5][11].

"Cross Validated" Honest Causal Boosting Causal Boosting is also an ensemble algorithm which uses Causal Trees as a "weak" base learner. This change results in the learned function to be able to find treatment effect heterogeneities [12]. This Causal Boosting algorithm is validated the same way as the Causal Tree, but the creators of this algorithm still referred to it as "Cross Validated", although this is rather an "Honest" Causal Boosting algorithm.

Causal Bayesian Additive Regression Trees (BART) Causal BART is very similar to Causal Boosting, both use sequential weak learners. But instead of applying a learning rate to the sequentials, the Bayesian approach uses a set of priors. By using a prior, which provide regularization in a way that no single regression tree can dominate the total fit, and likelihood to get a posterior distribution of our predictions[13].

Model Deployment and Training We put these up against the two-model approach which functions as our benchmark model. In this regard we want to find out if these specially constructed algorithms outperform the traditional and rather fast implementations, here the ridge regression two model approach.

Baseline: Two-Model with Ridge Regression In order to judge whether the direct modeling of treatment effects is more efficient than the conventional two-model approach, we built such a model as baseline. We picked Ridge Regression since it penalizes on coefficient size without removing coefficients entirely, as LASSO does.

```
1  # RIDGE TWO MODEL -----
2
3  data <- f_a.training
4
5  n <- names(data)
6  f <- as.formula(paste("checkoutAmount ~", paste(n[!n %in% exclude.vars], collapse = " + ")))
7
8  ridge_model.matrix_t <- model.matrix(f,data[data$treatment==1,]),[-1] #preparing data for training
9  ridge_model.matrix_c <- model.matrix(f,data[data$treatment==0,]),[-1]
10
11 yt <- data[data$treatment==1,which(names(data) %in% c("checkoutAmount"))] #extracting the target variable as vector
12 yc <- data[data$treatment==0,which(names(data) %in% c("checkoutAmount"))]
13
14 lambdas <- 10^seq(3, -2, by = -.1) # defining the range of possible lambdas from which to pick the best
```

```

15
16 set.seed(101010, kind = "Mersenne-Twister", normal.kind = "Inversion", sample.kind = "Rounding")
17 ridge_f_a_t <- cv.glmnet(x=ridge_model.matrix_t, y=yt, alpha = 0, lambda = lambdas) #training on treated
18 ridge_f_a_c <- cv.glmnet(x=ridge_model.matrix_c, y=yc, alpha = 0, lambda = lambdas) #training on control
19
20 opt_lambda_t <- ridge_f_a_t$lambda.min #choosing best lambda
21 opt_lambda_c <- ridge_f_a_c$lambda.min
22
23 ridge_prediction.matrix <- model.matrix(f,f_a.test)[-1] #preparing data for prediction
24
25 ridge_f_a_t_predict <- predict(ridge_f_a_t, s = opt_lambda_t, newx = ridge_prediction.matrix) #predicting on treated
26 ridge_f_a_c_predict <- predict(ridge_f_a_c, s = opt_lambda_c, newx = ridge_prediction.matrix) #predicting on control
27
28 ridge_f_a_uplift <- ridge_f_a_t_predict-ridge_f_a_c_predict #computing uplift as difference between the groups

```

2- view raw (https://gist.github.com/lukekolbe/317c7c868df54edeab45c0cd376dd9c9/raw/9e8d759526f1dc7f79bbf1f5f9fea1f6e2d04ff4/2-Model_f_a.R) Model_f_a.R (https://gist.github.com/lukekolbe/317c7c868df54edeab45c0cd376dd9c9#file-2-model_f_a-r) hosted with ❤ by GitHub (<https://github.com>)

CausalTree As it is a popular and well established method for treatment effect modeling and the foundation of causal_forest() and causalBoosting() models, we naturally fit a causalTree (package 'causalTree'), a derivative of Leo Breiman's Decision Trees geared toward modeling causal inference.

The tree was configured as giving an 'honest' estimation, i.e. the tree was fit on the training data and the nodes were estimated using the (separate) estimation set. Likewise, splitting criteria and cross-validation parameters are also optimized for honest estimation.

Computation-wise, the model is surprisingly costly. The CP parameter was crucial for model predictive power as well as computation time. Setting it too high and the resulting tree only has one node. Setting it too low and the tree becomes overly complex and costly to compute. The difference between $cp=0.0000001$ ($1e^{-7}$) and $cp=0.00000001$ ($1e^{-8}$) amounted to 10-fold increase in computation time

```

1 ct_model.frame <- model.frame(f,data)
2
3 system.time(ct_f_a <- honest.causalTree(formula=ct_model.frame,
4                                         data=data,
5                                         treatment = data$treatment,
6                                         est_data = f_b.estimation, #specifying the dataset used for estimating the leaf nodes
7                                         est_treatment = f_b.estimation$treatment, #treatment var in the estimation data
8                                         HonestSampleSize = nrow(f_b.estimation),
9                                         cp = 0.0000001, #setting cp value influences complexity of the tree and computation ti
10                                        split.Rule = "CT",
11                                        split.Honest = T,
12                                        minsize=10, #more is uicker
13                                        cv.option = "CT", #option to calculate the cross validation error
14                                        cv.Honest = T, #apply honest risk evaluation function in cross validation
15                                        xval=10)) #10-fold cross validation
16
17 opcp <- ct_f_a$cptable[,1][which.min(ct_f_a$cptable[,4])]
18 opTree <- prune(ct_f_a, opcp)
19 rpart.plot(opTree)
20
21 f_a_ct.hon.pred_6_0_1_28 <- predict(ct_f_a, f_a.test[, -which(names(f_a.test) %in% exclude.vars)])

```

view raw (https://gist.github.com/lukekolbe/27513b368226dcc1ee1096be6421d1c9/raw/66d27a96795f7a74a7ce76af757f774b303cf2ec/causalTree_f_a.R) causalTree_f_a.R (https://gist.github.com/lukekolbe/27513b368226dcc1ee1096be6421d1c9#file-causaltree_f_a-r) hosted with ❤ by GitHub (<https://github.com>)

CausalForest The Causal Forest (package 'grf') was fitted using a parallelized approach, building four forests with 1000 trees each and

then combining them using the designated combine function of the 'grf' package. Parallelization was deployed using the 'doParallel' package. Performance this way in terms of computation time was good.

The Forests were again configured toward honest estimation. All NULL variables were automatically tuned.

```
1 registerDoParallel(cores=4) #set up 4 R backends for parallelization
2
3 system.time(cf_f_a_dopar <- foreach(ntree=rep(1000,4), #parallelized training of 4000 trees
4                                     .combine=function(a,b,c,d)grf::merge_forests(list(a,b,c,d)), #combining the 4 trees into o
5                                     .multicombine=TRUE,.packages='grf') %dopar% {
6                                     causal_forest(
7                                         X = data[,-which(names(data) %in% exclude.vars)], #removing checkoutAmount and treatme
8                                         Y = data$checkoutAmount, #target vector
9                                         W = data$treatment, #treatment vector
10                                        num.trees = ntree, #1000 trees each
11                                        honesty = TRUE, #use honest splitting
12                                        honesty.fraction = NULL, #use half the data to determine splits
13                                        tune.parameters = TRUE, #Tune all parameters set to NULL via cross validation
14                                        seed = 1839 #fixed C++ RNG
15                                    )
16                                }
17 )
18 stopImplicitCluster() #stop and remove parallel backends
19
20 cf_f_a.preds <- predict(object = cf_f_a,
21                        newdata=f_a.test[, -which(names(data) %in% exclude.vars)], # removing, checkoutAmount, converted, confi
22                        estimate.variance = TRUE)
```

view raw

(https://gist.github.com/lukekolbe/be6a0ded7db88dab6b3cce96124d6d33/raw/366a0a253a2a0671d4ecaeb950e82ec4820eac95/causal_forest_f_a.R)
causal_forest_f_a.R (https://gist.github.com/lukekolbe/be6a0ded7db88dab6b3cce96124d6d33#file-causal_forest_f_a-r) hosted with ❤ by GitHub
(<https://github.com>)

CausalBoosting In theory a powerful model, the Causal Boosting (package 'causalLearning') model suffered from severe performance issues in our application. The model in its base configuration took over a week to compute (it was stopped after 7 days) and was thus not feasible for implementation. Instead, we de-tuned it by reducing the number of trees to 50 (default = 500), setting the learning rate (eps) to a rather weak 0.3 and reducing the cross validation folds to 5. Still it took up to 36 hours on designated servers to compute. It is entirely possible that this low-cost approach has negative influence on performance, but computational cost is a factor to consider when choosing models.

```
1 # CausalBoosting -----
2
3 system.time(cv.cb_f_a <- cv.causalBoosting(data[, -which(names(data) %in% exclude.vars)], # more data causes significant increa
4                                             tx=data$treatment,
5                                             y=data$checkoutAmount,
6                                             num.trees=50, # instead of 500
7                                             splitSpread = 0.1,
8                                             maxleaves = 8,
9                                             eps=0.3, #instead of 0.1
10                                            nfolds=5)) #instead of 10
11
12
13 cb_f_a.pred <- predict(cv.cb_f_a,
14                      newx = f_a.test[, -which(names(data) %in% exclude.vars)],
15                      newtx = f_a.test$treatment,
16                      type = "treatment.effect",
17                      num.trees = 50)
```

view raw (https://gist.github.com/lukekolbe/2bc00088b843ae785915df5575f674c1/raw/f4b04c27473b02dc9a9589b31cf5b35815f6ca68/causalBoost_f_a.R)
causalBoost_f_a.R (https://gist.github.com/lukekolbe/2bc00088b843ae785915df5575f674c1#file-causalboost_f_a-r) hosted with ❤ by GitHub
(<https://github.com>)

Causal Bart Implementation of Causal Bart (package 'bartCause') was rather straightforward. Computation was rather quick, even with 1000 trees and elaborated sampling. An interesting feature of the bartc() function is the parameter group.by, which lets the model calculate the treatment effects for different groups. This might be useful when there isn't just one treatment, but in fact different treatments within a population (see description of Books&Toys dataset).

```
1  # BART -----
2
3  data <- f_a.train_small
4  conf<-as.matrix(data[, -which(names(data) %in% exclude.vars)])
5
6  x <- conf
7  y <- data$checkoutAmount
8  z <- data$treatment
9
10 x.new <- f_a.test[, -which(names(f_a.test) %in% exclude.vars)]
11
12 system.time(fit <- bartc(y, z, x,
13                          method.trt = "bart",
14                          method.rsp = "bart",
15                          estimand="att", #average treatment effect on the treated
16                          n.samples = 20L,
17                          n.chains = 8L, #Integer specifying how many independent tree sets and fits should be calculated.
18                          n.burn = 10L,
19                          n.threads = 4L, #Integer specifying how many threads to use for parallelization
20                          n.trees = 1000L,
21                          keepTrees = TRUE, #necessary for prediction!
22                          verbose = FALSE))
23
24 y1_att <- predict(fit, x.new, value = "y1", combineChains = TRUE)
```

view raw (https://gist.github.com/lukekolbe/455e31d86fb7ce96ad08993cd7e81cf3/raw/6aad3969ca252f8582cab56dead3fc046843b2ff/causalbart_f_a.R)
causalbart_f_a.R (https://gist.github.com/lukekolbe/455e31d86fb7ce96ad08993cd7e81cf3#file-causalbart_f_a-r) hosted with ❤ by GitHub
(<https://github.com>)

Implications towards Profit, Revenue and Costs.

We have two different kind of costs. Cost-type one are costs which result from using a voucher, effectively reducing the revenue a customer generates for the amount of the campaign Value. The other cost-type stems from not treating customers, which would have generated an additional revenue, if they were treated. This could be seen as some form of opportunity costs.

The following definitions should help to understand the complexity of incurring costs:

(1) $\text{discount}_i := \text{Campaign Value}_i$; if $\text{basketValue}_i \geq \text{MOV}$
0; else

(2) $\text{Profit}_i := \text{basketValue}_i - \text{discount}_i := \text{checkoutAmount}_i$

One should bear in mind, that the cost-type one, costs from discounts are already internalized in the checkout amount variable (1). The discount can either be equal to the campaign value or zero, following the logic of (1). As, stated in (2) the checkout amount is equal to the profit.

Hence, we do profit modeling when utilizing the *checkoutAmount* variable as the target variable in our models.

(3) $\text{basketValue}_{tr} := \text{basketValue}_{ct} + \text{Uplift}_{tr}$ The basket value of a treated person is equal to the basket value of a person in the control group, plus the uplift in revenue which was generated by the treatment.

(4) $\text{basketValue}_{ct} := \text{checkoutAmount}_{ct}$ The basket value of a person in the control group is equal to its checkout amount, since there are no additional factors, such as discount or uplifts, since this person was not treated.

Model Evaluation Since there is no ground truth to evaluate our estimates on, finding a measure for the model performance in Uplift modeling is a core issue.

Most practitioners resort to measures such as uplift curves [16]. In the following example, we are going to show you how to compute these step-by-step.

We are starting with ranking our model predictions for both treated and control data points, and rank them from high to low.

```
1 # We rank the uplift scores from high to low and add this information to a dataframe
2 mm_pred = cbind(
3   uplift = p,
4   target = t.d$checkoutAmount,
5   treatment = t.d$treatment,
6   uplift_rank = length(p) + 1 - rank(p, ties.method = "random")
7 )
```

view raw (https://gist.github.com/lukekolbe/18eb7c3330a42bb390f6db54c2954372/raw/62ae6bd2bcd3a060a46c614a094018b52fa8089d/eval-1_mm_pred.R)
eval-1_mm_pred.R (https://gist.github.com/lukekolbe/18eb7c3330a42bb390f6db54c2954372#file-eval-1_mm_pred-r) hosted with ❤ by GitHub (<https://github.com>)

We continue binning the individuals accordingly to their rankings into deciles. Consequently the individuals with the highest uplift scores will populate the first group.

```
1 # We define deciles (groups), which get used in the literature to evaluate our model performance across 10 equally sized sub p
2 groups = 10
3 bk_pred = unique(quantile(mm_pred[, 4], probs = seq(0, 1, 1 / groups)))
4
5 if ((length(bk_pred) - 1) != groups) {
6   warning (
7     "uplift: due to many ties in uplift predictions, the ties will be dealt with randomly",
8     groups
9   )
10 }
11 # We add the deciles to our model matrix
12 mm_pred = cbind(mm_pred,
13   decile = cut(
14     mm_pred[, 4],
15     breaks = bk_pred,
16     labels = NULL,
17     include.lowest = T
18   ))
```

view raw (https://gist.github.com/lukekolbe/bc5c40f0f0ac8d9844daae7f4c045cb1/raw/e0477e8f57fa8e82bfc84800b1c815fc53bfb89f/mm_deciles.R)
mm_deciles.R (https://gist.github.com/lukekolbe/bc5c40f0f0ac8d9844daae7f4c045cb1#file-mm_deciles-r) hosted with ❤ by GitHub (<https://github.com>)

Next, we are interested in the actual values, especially the following:

- Number of individuals in the treatment group per decile
- Number of individuals in the control group per decile
- The spending of both groups per decile
- The spending per capita of both groups per decile
- The achieved uplift per decile

```

1 # We compute the following variables from our model matrix
2 pred_n.y1_ct0 <- tapply(mm_pred[mm_pred[, 3] == 0,][, 2], mm_pred[mm_pred[, 3] == 0,][, 5], sum) # Sum of revenue of people
3 pred_n.y1_ct1 <- tapply(mm_pred[mm_pred[, 3] == 1,][, 2], mm_pred[mm_pred[, 3] == 1,][, 5], sum) # Sum of revenue of people
4 pred_r.y1_ct0 <- tapply(mm_pred[mm_pred[, 3] == 0,][, 2], mm_pred[mm_pred[, 3] == 0,][, 5], mean) # Average revenue of people
5 pred_r.y1_ct1 <- tapply(mm_pred[mm_pred[, 3] == 1,][, 2], mm_pred[mm_pred[, 3] == 1,][, 5], mean) # Average revenue of people
6 pred_n.ct0 <- tapply(mm_pred[mm_pred[, 3] == 0,][, 2], mm_pred[mm_pred[, 3] == 0,][, 5], length) # Sum of people not having rec
7 pred_n.ct1 <- tapply(mm_pred[mm_pred[, 3] == 1,][, 2], mm_pred[mm_pred[, 3] == 1,][, 5], length) # Sum of people having rec
8
9 # In rare situations the ratio of a group can be non-existing because there are no people in the treatment or control group.
10 # We set these to 0.
11 pred_r.y1_ct0 <- ifelse(is.na(pred_r.y1_ct0), 0, pred_r.y1_ct0)
12 pred_r.y1_ct1 <- ifelse(is.na(pred_r.y1_ct1), 0, pred_r.y1_ct1)
13 # The uplift is the difference of the sum of revenues between our treatment and control group
14 uplift = pred_r.y1_ct1 - pred_r.y1_ct0
15 # We bind our previously computed variables together to a performance matrix
16 perf_pred <- cbind(group = c(1:10),
17                   n.ct1 = pred_n.ct1,
18                   n.ct0 = pred_n.ct0,
19                   n.y1_ct1 = pred_n.y1_ct1,
20                   n.y1_ct0 = pred_n.y1_ct0,
21                   r.y1_ct1 = pred_r.y1_ct1,
22                   r.y1_ct0 = pred_r.y1_ct0,
23                   uplift = uplift)

```

view raw (https://gist.github.com/lukekolbe/a2df2f2cc1c15f799f5ac24cd8804a0b/raw/10af769687a27afb6f667638fe227f5c4b2f891bd/perf_matrix.R)
perf_matrix.R (https://gist.github.com/lukekolbe/a2df2f2cc1c15f799f5ac24cd8804a0b#file-perf_matrix-r) hosted with ❤ by GitHub (<https://github.com>)

Based on these values we can create some plots, visualizing the performance matrix. Ideally, the plot should have high average checkout amount for the treatment group and low average checkout amounts for the control group in the first deciles and vice versa in the last deciles.

By subtracting the average checkout amount for the control group from the average checkout amount of the treatment group we achieve the uplift per capita per decile as pictured in the following plot.

Now using our beforehand calculated performance matrix we are able to generate uplift curves. The procedure is as follows:

1. We calculate the cumulative sum of the revenue of the treated and the control group while controlling for the total population in each group of the specified decile.
2. We calculate the percentage of the total amount of people (both treatment and control) present in each decile.

```

1 groups = 10
2 pred_r.cumul.y1_ct1 <- cumsum(perf_pred[, "n.y1_ct1"]) / cumsum(perf_pred[, "n.ct1"]) # Cumulative revenue per treated person
3 pred_r.cumul.y1_ct0 <- cumsum(perf_pred[, "n.y1_ct0"]) / cumsum(perf_pred[, "n.ct0"]) # Cumulative revenue per controlled person
4 deciles <- seq(1 / groups, 1, 1 / groups)
5 pred_r.cumul.y1_ct1[is.na(pred_r.cumul.y1_ct1)] <- 0 # If they are missing set them to zero
6 pred_r.cumul.y1_ct0[is.na(pred_r.cumul.y1_ct0)] <- 0

```

view raw (https://gist.github.com/lukekolbe/42c33b759d727fac25ebbcfe9f7bad1e/raw/639fe2436d7f2d451c9c23dae3814e87ccaea00/pred_cumul.R)

Afterwards, we are able to calculate the incremental gains for the model performance. This represents our uplift curve. The overall incremental gains represent the random curve. This is corresponding to the global effect of the treatment.

```
1 pred_inc.gains = c(0.0, (pred_r.cumul.y1_ct1 - pred_r.cumul.y1_ct0) * deciles) # The incremental gains
2 pred_overall.inc.gains <- sum(perf_pred[, "n.y1_ct1"]) / sum(perf_pred[, "n.ct1"]) - sum(perf_pred[, "n.y1_ct0"]) / sum(perf_
3 pred_random.inc.gains <- c(0, cumsum(rep(pred_overall.inc.gains / groups, groups))) # A cumulation of equal random gains whic
4
```

eval-5.R view raw (<https://gist.github.com/lukekolbe/c3324c449bb15d33ba06e464ff89207f/raw/f0bd196763d04ce80a78023df08fc9b7dbb6ce03/eval-5.R>) (<https://gist.github.com/lukekolbe/c3324c449bb15d33ba06e464ff89207f#file-eval-5-r>) hosted with ❤ by GitHub (<https://github.com>)

In our case the positive slope of the random curve confirms the overall positive effect of the randomized campaign. Finally, the shape of the curves shows the strong positive and negative uplifts. Contrary, the closer an uplift curve is to the random line, the weaker are the uplifts achieved.

We proceed with computing the area of both of these curves.

```
1 pred_x <- c(0.0, seq(1 / groups, 1, 1 / groups))
2 pred_y <- pred_inc.gains
3 pred_auc <- 0
4 pred_auc.rand <- 0
5 # calculating the Qini curve
6 for (i in 2:length(pred_x)) {
7   pred_auc <-
8     pred_auc + 0.5 * (pred_x[i] - pred_x[i - 1]) * (pred_y[i] + pred_y[i-1])
9 }
10 # calculating the random curve
11 pred_y.rand <- pred_random.inc.gains
12 for (i in 2:length(pred_x)) {
13   pred_auc.rand <-
14     pred_auc.rand + 0.5 * (pred_x[i] - pred_x[i - 1]) * (pred_y.rand[i] + pred_y.rand[i -
15                                                         1])
16 }
```

eval-6_qini.R view raw (https://gist.github.com/lukekolbe/490216b804451790f4a88dfa3b2a6caa/raw/9279917b3617a86563a2a1ab79d30ebd50ac5909/eval-6_qini.R) (https://gist.github.com/lukekolbe/490216b804451790f4a88dfa3b2a6caa#file-eval-6_qini-r) hosted with ❤ by GitHub (<https://github.com>)

Next up we subtract the area under to random curve from the area of our uplift curve.

```
1 # calculating the Qini
2 qini.matrix[t, which(names(qini.matrix) %in% paste(n))] <-
3   pred_auc - pred_auc.rand
4   return(qini.matrix)
5 }
```

view raw (https://gist.github.com/lukekolbe/c6d024d8c1a880e76d13b51157b4e865/raw/9c9024fc5466955c4d832c27308a4e0e15039dfc/eval-7_qini-calc.R) eval-7_qini-calc.R (https://gist.github.com/lukekolbe/c6d024d8c1a880e76d13b51157b4e865#file-eval-7_qini-calc-r) hosted with ❤ by GitHub (<https://github.com>)

The area between the Qini curve and the Random curve is called the Qini-coefficient.

The code was adapted from Guelman's Uplift package. <https://cran.r-project.org/web/packages/uplift/index.html> (<https://cran.r-project.org/web/packages/uplift/index.html>)

Model Profitability

Since we now want to know how profitable a campaign would be which is based on our model predictions, we need to know the revenue and cost streams which would occur and compare these to the realized campaign from the random experiment. Thus, we need to calculate the additional revenue, cost, and finally profit we generate.

From here on we refer to the campaign based on the model as the *model campaign* and for the realized campaign from the random experiment as the *campaign*.

Assumption We assume, that every individual who is eligible to use a voucher, indeed uses it. This is rather a conservative point of view. But we have no indication of how many people actually used the discount code for every dataset.

In the first step we need these variables, two of these, namely *eligibility* and *ExpectedDiscount* should be familiar to this point:

- *eligibility*
- *model eligibility*
- *ExpecedDiscount*
- *model ExpectedDiscount*

So what are the two other variables referring to our model?

The *model eligibility* ("m.eligibility") variable is an adaption of the *eligibility* variable we spoke about in our **Feature Creation** section. The difference is that *model eligibility* is applied to all individuals, internalizes the uplift (treatment effect), and is not accounting for the campaign value (discount). The reason for the lastest is, that you cannot use a voucher before qualifying to use it. Effectively, in this case the checkout amount is equal to the revenue and is not internalizing the discount.

This function demonstrates how the *model eligibility* variable is constructed:

(p + checkoutAmount_i) >= campaignMov_i

After calculating the *model eligibility*, we computed the *model ExpectedDiscount*, which indicates how much we expect each person's treatment (given *model eligibility*) to cost. Notice that individuals receiving a discount in percent are calculated slightly different than those with an absolut discount value:

(ExpectedDiscount_i | eligibility = 1 & campaignUnit = "CURRENCY") = campaignValue_i
(ExpectedDiscount_i | eligibility = 1 & campaignUnit = "PERCENT") = (checkoutAmount_i / (1 - campaignValue)) - checkoutAmount Costs

These new variables are added to our existing model matrix.

```
1   t.d$uplift <- p
2   #the eligibility rule for those who are newly eligible as per our model
3   t.d$m.eligibility[t.d$eligibility==0] <- ifelse(((t.d$uplift[t.d$eligibility==0] +
4   t.d$checkoutAmount[t.d$eligibility==0]) >= t.d$campaignMov[t.d$eligibili
5
6   t.d$m.eligibility[t.d$eligibility==1] <- ifelse(((t.d$uplift[t.d$eligibility==1] +
7   t.d$checkoutAmount[t.d$eligibility==1] +
8   t.d$ExpectedDiscount[t.d$eligibility==1]) >= t.d$campaignMov[t.d$eligibi
9
10  #the eligibility rule for those who are eligible as per our model but were also eligible before
11  t.d$m.ExpectedDiscount <- numeric(nrow(t.d))
12  t.d$m.ExpectedDiscount[t.d$campaignUnit=="CURRENCY"&t.d$m.eligibility==1] <-
13    t.d$campaignValue[t.d$campaignUnit=="CURRENCY"&t.d$m.eligibility==1]/100
14
15  # for those newly eligible we take the checkoutAmount (which is revenue),
16  # add the treatment effect, and compute the discount they receive as percentage of their revenue
17  t.d$m.ExpectedDiscount[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==0] <-
18    ((t.d$checkoutAmount[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==0] + # (checkoutAmount + uplift (pr
19    t.d$uplift[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==0]))*
20    (t.d$campaignValue[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==0]/10000))
21
22  # for those already eligible we set the expected Discount as per our Model differently:
```

```

23 # since the checkoutAmount for those sessions has already internalized treatment cost,
24 # we add the estimated ExpectedDiscount to the checkoutAmount as well as the Uplift as predicted by the model
25 # This is not entirely correct, but otherwise we would compute a discount on top of a discount,
26 # since people who were eligible in the original campaign have cost internalized.
27 t.d$m.ExpectedDiscount[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==1] <-
28   ((t.d$checkoutAmount[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==1]
29     + t.d$ExpectedDiscount[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==1]
30     + t.d$uplift[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==1])
31   * (t.d$campaignValue[t.d$campaignUnit=="PERCENT"&t.d$m.eligibility==1&t.d$eligibility==1]/10000))

```

view raw

https://gist.github.com/AsmirMumin/d178be6d9d00d824e834d046bc870194/raw/b2c381c0fa46f84a33a920031e629a5a3d8e19c1/eligibility_model_campaign.R
 eligibility_model_campaign.R (https://gist.github.com/AsmirMumin/d178be6d9d00d824e834d046bc870194#file-eligibility_model_campaign-r) hosted with
 ❤ by GitHub (<https://github.com>)

We proceed with calculating the number of *eligible* and *model eligible* customers per decile to be able to put these in our profitability matrix later on. The difference of these measures is the number of additional eligible customers, which we called *delta.eligible*.

```

1 pred_n.ct1.eligible <- tapply(mm_pred[mm_pred[, 6] == 1,][, 6],
2                               mm_pred[mm_pred[, 6] == 1,][, 5], length) # no of eligible customers per decile in the initia
3
4 # model features
5 pred_n.m.eligible = tapply(mm_pred[mm_pred[, 8] == 1,][, 8],
6                             mm_pred[mm_pred[, 8] == 1,][, 5], length) # number of eligible customers in the model campaign (m
7
8 #helper function that adds columns in case last deciles are left empty
9 if(length(pred_n.m.eligible)<10){
10   length <- (length(pred_n.m.eligible)+1)
11   for(i in c(length:10)){
12     pred_n.m.eligible[i] <- 0
13   }
14 }
15
16 delta.eligible <- pred_n.m.eligible - pred_n.ct1.eligible

```

view raw

https://gist.github.com/AsmirMumin/29de48f8a851911f8494080c4b4fe522/raw/48950b3d9cf97f499c9a3d626a3bfcbe116caa76/delta_eligible_new.R
 delta_eligible_new.R (https://gist.github.com/AsmirMumin/29de48f8a851911f8494080c4b4fe522#file-delta_eligible_new-r) hosted with ❤ by GitHub
 (<https://github.com>)

The number of additionally eligible individuals is core to compute the additional costs stemming from our *model campaign*, since only these individuals can use the discount voucher, which effectively reduces our profits and thus adds costs. The model costs are generated by individuals using their vouchers multiplied by the campaign value (discount) of these vouchers. Same applies for the campaign costs of the *campaign*. The difference of these costs are the additional costs caused by our *model campaign*, and we are only interested in these.

```

1 m.cost = tapply(mm_pred[mm_pred[, 8] == 1,][, 9], #give me all rows of column 9 (m.expectedDiscount) where model.eligible ==
2                 mm_pred[mm_pred[, 8] == 1,][, 5], sum) #group them by decile and compute the sum for each decile
3
4 #helper function that adds columns in case last deciles are left empty
5 if(length(m.cost)<10){
6   length <- (length(m.cost)+1)
7   for(i in c(length:10)){
8     m.cost[i] <- 0
9   }
10 }
11

```

```

12   campaign.cost <- tapply(mm_pred[mm_pred[, 6] == 1,][, 7], #give me all rows of column 7 (expectedDiscount) where model.eligi
13                               mm_pred[mm_pred[, 6] == 1,][, 5], sum)
14
15   if(length(campaign.cost)<10){
16     length <- (length(campaign.cost)+1)
17     for(i in c(length:10)){
18       campaign.cost[i] <- 0
19     }
20   }
21   delta.cost <- m.cost - campaign.cost # additional cost

```

view raw

(https://gist.github.com/AsmirMumin/73edf863c8b93efae170d77ee2dd29ad/raw/9a2312053d26937b7b44871a78338f38cfb324ff/delta_model_cost.R)
 delta_model_cost.R (https://gist.github.com/AsmirMumin/73edf863c8b93efae170d77ee2dd29ad#file-delta_model_cost-r) hosted with ❤ by GitHub
 (<https://github.com>)

The additional revenue is basically the number of additionally treated individuals multiplied by the uplift achieved on a decile level. Now, we are able to calculate the additional profit, which is generated by our *model campaign* if we treated every customer for each decile.

```

1   delta.revenue <- pred_n.ct0 * uplift #additional money earned by treating everyone in each decile
2   delta.profit = delta.revenue - delta.cost # profit generated by our model camapign if we treated everyone in each decile
3
4   # function to assess profit according to the cumulated no. of deciles treated from our model
5   profit_decile = function(x) {
6     if (x <= 9) {
7       sum(delta.profit[1:x]) + sum(delta.profit[(x + 1):10] * (-1))
8     } else{
9       sum(delta.profit[1:x])
10    }
11  }

```

view raw

(https://gist.github.com/AsmirMumin/f0a08e6a42fb2d6f15f10443e1bbab28/raw/42c0182c7c6451c4a9b7e3c793f3e01efde4c584/delta_revenue_delta_profit.R)
 delta_revenue_delta_profit.R (https://gist.github.com/AsmirMumin/f0a08e6a42fb2d6f15f10443e1bbab28#file-delta_revenue_delta_profit-r) hosted with ❤ by GitHub
 (<https://github.com>)

But since it is not optimal to treat everyone. We need to compute the additional profits generated cumulatively for each treated decile to detect how many deciles we should treat.

This is done, by cumulating profits for each treated decile and accounting for the missed profits of deciles which are not treated. We did this for every possible number of deciles and finally found the maximum. This is telling us how much profit we gain and at how many deciles we should treat.

```

1   profit_per_deciles_treated = numeric(10)
2
3   for (i in seq_along(1:10)) {
4     profit_per_deciles_treated[i] <-
5       profit_decile(i) # saving the cumulated profits for deciless treated
6   }
7   profit_per_deciles_treated <- cbind(c(1:10), profit_per_deciles_treated)
8
9
10  max.profit.model[t, which(names(max.profit.model) %in% paste(n)) = as.character(paste(
11    "Decile:",profit_per_deciles_treated[,1][which.max(profit_per_deciles_treated[,2])),
12    "|","Profit:", format(round(max(
13      profit_per_deciles_treated[,2]),2),

```

```
14      nsmall = 2), sep = " ")

```

view raw
https://gist.github.com/AsmirMumin/e58e83c4fd9e00f4f71691634bb0a3ee/raw/945b7d63e3ba36bd4ffe5fe4734627945227e34/Profit_per_deciles_max_profit.R
 Profit_per_deciles_max_profit.R (https://gist.github.com/AsmirMumin/e58e83c4fd9e00f4f71691634bb0a3ee#file-profit_per_deciles_max_profit-r) hosted with ❤ by GitHub (<https://github.com>)

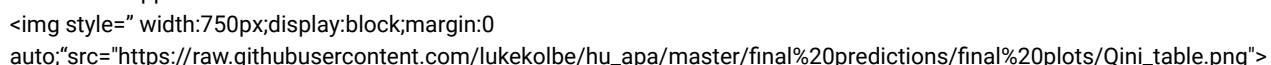
Since, it is not always possible to treat as many as suggested when maximizing the profit, due to budget constraints. We opted to calculate the Return on Investment (ROI) as well and use it as our decision criterion. This enables marketers to gain the maximal profitability per euro spend. We save these metrics to a profitability matrix.

```
1  #ROI per deciles treated
2  ROI_per_deciles_treated = cbind(c(1:10), profit_per_deciles_treated[,2]/cumsum(m.cost))
3  ROI_per_deciles_treated
4
5  #decile to treat
6  treatment.decision[t, which(names(treatment.decision) %in% paste(n))] <- as.character(paste(
7    "Decile:", ROI_per_deciles_treated[,1][which.max(ROI_per_deciles_treated[,2])],
8    "|", "ROI:", format(round(max(
9      ROI_per_deciles_treated[,2]), 2),
10     nsmall = 2), sep = " "))
11
12
13  profitability <- cbind(
14    group      = 1:10,
15    n.ct1      = pred_n.ct1,
16    n.ct0      = pred_n.ct0,
17    uplift     = uplift,
18    eligible   = pred_n.ct1.eligible,
19    delta.eligible = delta.eligible,
20    delta.Cost = delta.cost,
21    delta.Rev  = delta.revenue,
22    delta.Profit = delta.profit,
23    delta.cum_Profit_tr = profit_per_deciles_treated[,2],
24    delta.cum_ROI_tr = ROI_per_deciles_treated[,2]
25  )
26
27  return(list(max.profit.model, treatment.decision))
28 }
```

view raw
https://gist.github.com/AsmirMumin/16706b352d3fa586926ce0a457a2890d/raw/828516a663f4f0b6c1646d3c98d9af4a4ae334b5/ROI_Profitability_Matirx.R
 ROI_Profitability_Matirx.R (https://gist.github.com/AsmirMumin/16706b352d3fa586926ce0a457a2890d#file-roi_profitability_matirx-r) hosted with ❤ by GitHub (<https://github.com>)

The plot shows the profitability matrix, which is a table with columns for group, n.ct1, n.ct0, uplift, eligible, delta.eligible, delta.Cost, delta.Rev, delta.Profit, delta.cum_Profit_tr, and delta.cum_ROI_tr. The rows represent different deciles (1 to 10).

Results So how did our more sophisticated models, which are directly measuring the uplift in comparison to our baseline model, the Two-Model approach?

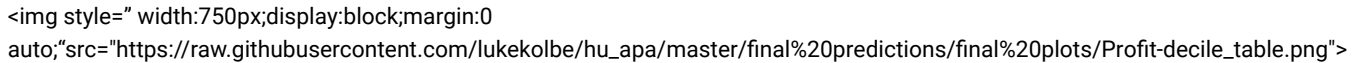
The plot shows the Qini table, which is a table with columns for group, n.ct1, n.ct0, uplift, eligible, delta.eligible, delta.Cost, delta.Rev, delta.Profit, delta.cum_Profit_tr, and delta.cum_ROI_tr. The rows represent different deciles (1 to 10).

The tree based models dominated the Two-Model approach measured by the Qini-coefficient in our FashionA and FashionB dataset. In the BooksToys dataset, the Two-Model approach placed second to Causal BART and in the Hillstrom dataset it placed best, leaving out the ensemble model.

We could not confirm that having more features generally better the model performance.

Which models are especially mention-worthy? The Causal Bart generally performed pretty well across all datasets, but FashionB. It was only beaten by our ensemble model, which averages the predictions of the other models did well in every dataset, and is considered the overall winner when measuring the Qini-coefficient.

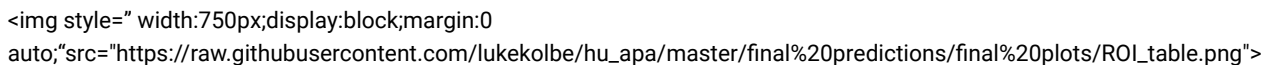
Although the Qini-coefficient is a viable measure for the model performance, it does not account for costs which are occurring from the treatment. Thus, these results are only fully valid, when we have low or no costs from additional treatments. Since costs are occurring in some of the datasets, the of models can change.

A line plot showing the absolute profits gained by different models across various deciles. The x-axis represents deciles and the y-axis represents profits. The plot shows that the ensemble model is the best performer, followed by Causal Forest and Causal Boosting.

Considering the absolute profits gained by the models, the ensemble model no longer is the best performer. We also do not have a clear winner here. In the FashionA dataset, the Causal Forest is surprisingly the winner, with 13k profits generated when targeting 9 deciles. In FashionB causal Boosting is generating the most profits, with 17k while treating 8 deciles. In BooyToys we achieved the highest profit gain of 50k, when treating only the first decile, which makes it also the most profitable one in terms of Return on Investment (ROI). In the Hillstrom dataset, Causal Boosting won again with 4k profits while treating all customers.

Causal Boosting was able to place first in two of our four datasets, making it the best performing in terms of absolute profits.

If budget is restricted, the ROI would be the measure to decide on which deciles to treat.

A line plot showing the Return on Investment (ROI) for different models across various deciles. The x-axis represents deciles and the y-axis represents ROI. The plot shows that the ensemble model is the best performer, followed by Causal Forest and Causal Boosting.

You might ask yourself, why are one of the latest deciles the most profitable ones, this is due to the campaign design. We explain this in the next section.

Key Findings We see that the way the campaign is constructed/designed is heavily impactful on the uplifts achieved and the general profitability of our campaign. For example, in FashionA, the marketers have used upselling, a technique employed in marketing to incentivize customers to spend a certain amount of money. In this case, the treatment group received a 20€ voucher, but those vouchers could be only used if the customers had a minimum order value of 125€. The data shows that not everybody was willing to spend this amount of money, since a lot of treated customers didn't reach the necessary spending, but still we estimated a lot of customers to respond positively on the treatment. This means, that solely giving the customers a chance of saving 20€ is effective as a treatment, which means it incentivizes customers to shop. The data suggests that some customers would browse the shopping website for products they like, finally ending up at a lower cart value than the minimum order value requires, and still buy those articles. One could assume that they were invested too much to this point, be it timewise or emotionally to churn now. The profit analysis for FashionA shows that the most profitable deciles for every model were always the last ones, which indicates that this upselling technique allows to treat most of the customers, since the cost which come with the 20€ voucher would only occur for a minority of the treated. The same logic applies to dataset FashionB, where the minimum order value is between 30€ and 50€. Similarly, for the Hillstrom data, where treatment is generally very cheap, it might still be beneficial to treat many people – only avoiding the sleeping dogs (customers who have negative treatment effects).

Contrarily, if we don't restrict the treatment, every additional treatment equates to additional cost. The profitability table above shows that in Books&Toys, the lower deciles are most profitable. This is likely because many rows in the data have no MOV assigned, meaning that everyone who is treated receives a discount. In these cases the treatment cost amounts to substantial amounts when treating all or most people. Hence, the profit as we model it does not only reflect the uplift gained, but also the money *not* spend by not treating later deciles.

Conclusion

In Conclusion, we were able to increase our model profits with every model. Our results suggest that uplift modeling is a viable tool to optimize marketing campaign profits. But one should bear in mind, the cost of treatments if the number of the effectively treated is high or has no restriction. The campaigns with a higher minimum order value were most profitable.

Our models mostly performed better than random. But it is hard to declare a clear winner, since it is highly depended what our campaign looks like.

Recommendations We recommend to use various models to find the best performing ones for a given scenario, regarding budget restrictions and campaign design.

It is also important to account for the computational costs: Causal Boosting, which was computationally very heavy, might not be a sensible choice for treatment effect modeling when time is a factor or model adaption and re-training cycles are frequent.

Deciding for a model should ideally consider both the performance as measures by the Qini score, as well as the (predicted) profitability. Building ensembles might be useful!

Also regarding the modeling, it is much simpler to assess the profitability when you are not dealing with a variable which has internalized costs. This proved to be quite troubling when finding the right assessment models, as for some lines the checkoutAmount is actual revenue, while for others it is revenue minus treatment cost.

Literature [1] M. Soltys and S. Jaroszewicz. Boosting algorithms for uplift modeling, 2018.

[2] B. Hansotia and B. Rukstales. Incremental value modeling. Journal of Interactive Marketing, 16(3):35–46, 2002.

[3] K. Hillstrom. The MineThatData e-mail analytics and data mining challenge. MineThatData blog, <http://blog.minethatdata.com/2008/03/> (<http://blog.minethatdata.com/2008/03/>) minethatdata-e-mail-analytics-and-data.html, 2008. Retrieved on 06.10.2014.

[4] I. Csiszar and P. Shields. Information theory and statistics: A tutorial. Foundations and Trends in Communications and Information Theory, 1(4):417–528, 2004.

[5] L. Guelman, M. Guill'en, and A.M. P'erez-Mar'in. Random forests for uplift modeling: An insurance customer retention case. In Modeling and Simulation in Engineering, Economics and Management, volume 115 of Lecture Notes in Business Information Processing (LNBIP), pages 123–133. Springer, 2012.

[6] M. Ja'skowski and S. Jaroszewicz. Uplift modeling for clinical trial data. In ICML 2012 Workshop on Machine Learning for Clinical Data Analysis, Edinburgh, Scotland, June 2012.

[7] P. Gutierrez and J.-Y. G'erdard. Causal Inference and Uplift Modeling. A review of the literature, 2016.

[8] L. Guelman, M. Guillen, A. M. P'erez-Mar'in, et al. Optimal personalized treatment rules for marketing interventions: A review of methods, a new proposal, and an insurance case study. Technical report, 2014.

[9] S. Athey and G. W. Imbens. Recursive partitioning for heterogeneous causal effects. arXiv preprint arXiv:1504.01132, 2015a.

[10] S. Athey and G. W. Imbens. Machine learning methods for estimating heterogeneous causal effects. stat, 1050:5, 2015b.

[11] S. Athey and S. Wager. Estimation and inference of heterogeneous treatment effects using random forests. arXiv preprint arXiv:1510.04342, 2015.

[12] S. Powers et. al. Some methods for heterogeneous treatment effect estimation in high dimensions. arXiv preprint arXiv:1707.00102, 2017.

[13] P. R. Hahn et. al. Bayesian Regression tree models for causal inference: Regularization, Confounding, and Heterogeneous Effects. arXiv preprint arXiv:1706.09523, 2019.

[14] Gubela et. al. Revenue Uplift Modeling, 2007.

[15] J. L. Hill. Bayesian Nonparametric Modeling for Causal Inference, Journal of Computational and Graphical Statistics, 20:1, 217-240. DOI: 10.1198/jcgs.2010.08162, 2011.

[16] M. Soltys, S. Jaroszewicz, and P. Rzepakowski. Ensemble methods for uplift modeling. Data mining and knowledge discovery, 29(6):1531–1559, 2015.

[17] <https://www.statista.com/statistics/236943/global-advertising-spending/> (<https://www.statista.com/statistics/236943/global-advertising-spending/>)

CATEGORIES

course-projects (37) (<https://humboldt-wi.github.io/blog/categories/course-projects>)

instruction (2) (<https://humboldt-wi.github.io/blog/categories/instruction>)

TAGS

🔖 **A/B-TESTING** ([HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/A/B-TESTING](https://humboldt-wi.github.io/blog/tags/a/b-testing))

🔖 **ALBERT** ([HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ALBERT](https://humboldt-wi.github.io/blog/tags/albert))

🔖 **ATTENTION** ([HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ATTENTION](https://humboldt-wi.github.io/blog/tags/attention))

🔖 **AWD-LSTM** ([HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/AWD-LSTM](https://humboldt-wi.github.io/blog/tags/awd-lstm))

- 🔗 [BAYESIAN-DEEP-LEARNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BAYESIAN-DEEP-LEARNING\)](https://humboldt-wi.github.io/blog/tags/bayesian-deep-learning)
- 🔗 [BAYESIAN-TOPIC-MODELLING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BAYESIAN-TOPIC-MODELLING\)](https://humboldt-wi.github.io/blog/tags/bayesian-topic-modelling)
- 🔗 [BERT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BERT\)](https://humboldt-wi.github.io/blog/tags/bert)
- 🔗 [BILM \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BILM\)](https://humboldt-wi.github.io/blog/tags/bilm)
- 🔗 [BINARY \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BINARY\)](https://humboldt-wi.github.io/blog/tags/binary)
- 🔗 [BLACK-BOX \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BLACK-BOX\)](https://humboldt-wi.github.io/blog/tags/black-box)
- 🔗 [BLOCKCHAIN \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/BLOCKCHAIN\)](https://humboldt-wi.github.io/blog/tags/blockchain)
- 🔗 [CAUSAL-INFERENCE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CAUSAL-INFERENCE\)](https://humboldt-wi.github.io/blog/tags/causal-inference)
- 🔗 [CLASS17/18 \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CLASS17/18\)](https://humboldt-wi.github.io/blog/tags/class17/18)
- 🔗 [CLASS18/19 \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CLASS18/19\)](https://humboldt-wi.github.io/blog/tags/class18/19)
- 🔗 [CLASS19 \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CLASS19\)](https://humboldt-wi.github.io/blog/tags/class19)
- 🔗 [CLASS19/20 \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CLASS19/20\)](https://humboldt-wi.github.io/blog/tags/class19/20)
- 🔗 [CLASSIFICATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CLASSIFICATION\)](https://humboldt-wi.github.io/blog/tags/classification)
- 🔗 [CNN \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CNN\)](https://humboldt-wi.github.io/blog/tags/cnn)
- 🔗 [COARSENEDED-EXACT-MATCHING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/COARSENEDED-EXACT-MATCHING\)](https://humboldt-wi.github.io/blog/tags/coarsened-exact-matching)
- 🔗 [CONVERSION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CONVERSION\)](https://humboldt-wi.github.io/blog/tags/conversion)
- 🔗 [CONVOLUTIONAL-NEURAL-NETWORKS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CONVOLUTIONAL-NEURAL-NETWORKS\)](https://humboldt-wi.github.io/blog/tags/convolutional-neural-networks)
- 🔗 [CREDIT-RISK \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/CREDIT-RISK\)](https://humboldt-wi.github.io/blog/tags/credit-risk)
- 🔗 [DATA-SIMULATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DATA-SIMULATION\)](https://humboldt-wi.github.io/blog/tags/data-simulation)
- 🔗 [DEEP-LEARNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DEEP-LEARNING\)](https://humboldt-wi.github.io/blog/tags/deep-learning)
- 🔗 [DEEPLARNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DEEPLARNING\)](https://humboldt-wi.github.io/blog/tags/deeplearning)
- 🔗 [DISTANT-TRANSFER-LEARNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DISTANT-TRANSFER-LEARNING\)](https://humboldt-wi.github.io/blog/tags/distant-transfer-learning)
- 🔗 [DML \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DML\)](https://humboldt-wi.github.io/blog/tags/dml)
- 🔗 [DOC2VEC \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DOC2VEC\)](https://humboldt-wi.github.io/blog/tags/doc2vec)
- 🔗 [DOCUMENT-EMBEDDINGS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/DOCUMENT-EMBEDDINGS\)](https://humboldt-wi.github.io/blog/tags/document-embeddings)
- 🔗 [ECONOMICUNCERTAINTY \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ECONOMICUNCERTAINTY\)](https://humboldt-wi.github.io/blog/tags/economicuncertainty)
- 🔗 [ELMO \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ELMO\)](https://humboldt-wi.github.io/blog/tags/elmo)
- 🔗 [EMBEDDINGS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/EMBEDDINGS\)](https://humboldt-wi.github.io/blog/tags/embeddings)
- 🔗 [EXPLANATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/EXPLANATION\)](https://humboldt-wi.github.io/blog/tags/explanation)
- 🔗 [FASTTEXT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/FASTTEXT\)](https://humboldt-wi.github.io/blog/tags/fasttext)
- 🔗 [FINE-TUNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/FINE-TUNING\)](https://humboldt-wi.github.io/blog/tags/fine-tuning)

🔗 [GENETIC-MATCHING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/GENETIC-MATCHING\)](https://humboldt-wi.github.io/blog/tags/genetic-matching)

🔗 [GLOVE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/GLOVE\)](https://humboldt-wi.github.io/blog/tags/glove)

🔗 [GPT-2 \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/GPT-2\)](https://humboldt-wi.github.io/blog/tags/gpt-2)

🔗 [GRU \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/GRU\)](https://humboldt-wi.github.io/blog/tags/gru)

🔗 [HIERARCHICAL-NETWORK \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/HIERARCHICAL-NETWORK\)](https://humboldt-wi.github.io/blog/tags/hierarchical-network)

🔗 [ICE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ICE\)](https://humboldt-wi.github.io/blog/tags/ice)

🔗 [IMAGE-ANALYSIS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/IMAGE-ANALYSIS\)](https://humboldt-wi.github.io/blog/tags/image-analysis)

🔗 [IMAGE-CAPTIONING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/IMAGE-CAPTIONING\)](https://humboldt-wi.github.io/blog/tags/image-captioning)

🔗 [IMBALANCED-DATA \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/IMBALANCED-DATA\)](https://humboldt-wi.github.io/blog/tags/imbalanced-data)

🔗 [INFERENCE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/INFERENCE\)](https://humboldt-wi.github.io/blog/tags/inference)

🔗 [ITE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ITE\)](https://humboldt-wi.github.io/blog/tags/ite)

🔗 [KERAS-IMDB-DATASET \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/KERAS-IMDB-DATASET\)](https://humboldt-wi.github.io/blog/tags/keras-imdb-dataset)

🔗 [KNN-ALGORITHM \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/KNN-ALGORITHM\)](https://humboldt-wi.github.io/blog/tags/knn-algorithm)

🔗 [LANGUAGE-MODEL \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LANGUAGE-MODEL\)](https://humboldt-wi.github.io/blog/tags/language-model)

🔗 [LANGUAGE-MODELING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LANGUAGE-MODELING\)](https://humboldt-wi.github.io/blog/tags/language-modeling)

🔗 [LANGUAGE-MODELLING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LANGUAGE-MODELLING\)](https://humboldt-wi.github.io/blog/tags/language-modelling)

🔗 [LDA \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LDA\)](https://humboldt-wi.github.io/blog/tags/lda)

🔗 [LIME \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LIME\)](https://humboldt-wi.github.io/blog/tags/lime)

🔗 [LONG-SHORT-TERM-MEMORY \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LONG-SHORT-TERM-MEMORY\)](https://humboldt-wi.github.io/blog/tags/long-short-term-memory)

🔗 [LSTM \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/LSTM\)](https://humboldt-wi.github.io/blog/tags/lstm)

🔗 [MACHINE-LEARNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/MACHINE-LEARNING\)](https://humboldt-wi.github.io/blog/tags/machine-learning)

🔗 [MATCHING-METHODS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/MATCHING-METHODS\)](https://humboldt-wi.github.io/blog/tags/matching-methods)

🔗 [MATCHIT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/MATCHIT\)](https://humboldt-wi.github.io/blog/tags/matchit)

🔗 [MONTE-CARLO-DROPOUT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/MONTE-CARLO-DROPOUT\)](https://humboldt-wi.github.io/blog/tags/monte-carlo-dropout)

🔗 [MOVIE-REVIEWS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/MOVIE-REVIEWS\)](https://humboldt-wi.github.io/blog/tags/movie-reviews)

🔗 [NEAREST-NEIGHBOR \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/NEAREST-NEIGHBOR\)](https://humboldt-wi.github.io/blog/tags/nearest-neighbor)

🔗 [NEURAL-NETWORK \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/NEURAL-NETWORK\)](https://humboldt-wi.github.io/blog/tags/neural-network)

🔗 [NEURAL-NETWORKS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/NEURAL-NETWORKS\)](https://humboldt-wi.github.io/blog/tags/neural-networks)

🔗 [NLP \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/NLP\)](https://humboldt-wi.github.io/blog/tags/nlp)

🔗 [NN \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/NN\)](https://humboldt-wi.github.io/blog/tags/nn)

🔗 [OPTIMAL-MATCHING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/OPTIMAL-MATCHING\)](https://humboldt-wi.github.io/blog/tags/optimal-matching)

🔗 [OVERSAMPLING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/OVERSAMPLING\)](https://humboldt-wi.github.io/blog/tags/oversampling)

🔗 [PDP \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/PDP\)](https://humboldt-wi.github.io/blog/tags/pdp)

🔗 [PRETRAINING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/PRETRAINING\)](https://humboldt-wi.github.io/blog/tags/pretraining)

🔗 [PROPENSITY-SCORE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/PROPENSITY-SCORE\)](https://humboldt-wi.github.io/blog/tags/propensity-score)

🔗 [PROPENSITY-SCORE-WEIGHTING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/PROPENSITY-SCORE-WEIGHTING\)](https://humboldt-wi.github.io/blog/tags/propensity-score-weighting)

🔗 [RECOMMENDATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/RECOMMENDATION\)](https://humboldt-wi.github.io/blog/tags/recommendation)

🔗 [RECOMMENDER-SYSTEM \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/RECOMMENDER-SYSTEM\)](https://humboldt-wi.github.io/blog/tags/recommender-system)

🔗 [RECOMMENDER-SYSTEMS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/RECOMMENDER-SYSTEMS\)](https://humboldt-wi.github.io/blog/tags/recommender-systems)

🔗 [RNN \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/RNN\)](https://humboldt-wi.github.io/blog/tags/rnn)

🔗 [ROBERTA \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ROBERTA\)](https://humboldt-wi.github.io/blog/tags/roberta)

🔗 [RS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/RS\)](https://humboldt-wi.github.io/blog/tags/rs)

🔗 [SENTIMENT-ANALYSIS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SENTIMENT-ANALYSIS\)](https://humboldt-wi.github.io/blog/tags/sentiment-analysis)

🔗 [SENTIMENT-CLASSIFICATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SENTIMENT-CLASSIFICATION\)](https://humboldt-wi.github.io/blog/tags/sentiment-classification)

🔗 [SEQ2SEQ \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SEQ2SEQ\)](https://humboldt-wi.github.io/blog/tags/seq2seq)

🔗 [SHARE-PRICE-PREDICTION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SHARE-PRICE-PREDICTION\)](https://humboldt-wi.github.io/blog/tags/share-price-prediction)

🔗 [SIMPLETRANSFORMERS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SIMPLETRANSFORMERS\)](https://humboldt-wi.github.io/blog/tags/simpletransformers)

🔗 [SIMULATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SIMULATION\)](https://humboldt-wi.github.io/blog/tags/simulation)

🔗 [SURVIVAL-ANALYSIS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/SURVIVAL-ANALYSIS\)](https://humboldt-wi.github.io/blog/tags/survival-analysis)

🔗 [TEXT-ANALYSIS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TEXT-ANALYSIS\)](https://humboldt-wi.github.io/blog/tags/text-analysis)

🔗 [TEXT-CLASSIFICATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TEXT-CLASSIFICATION\)](https://humboldt-wi.github.io/blog/tags/text-classification)

🔗 [TEXT-GENERATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TEXT-GENERATION\)](https://humboldt-wi.github.io/blog/tags/text-generation)

🔗 [TEXT-MINING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TEXT-MINING\)](https://humboldt-wi.github.io/blog/tags/text-mining)

🔗 [TEXT-SUMMARIZATION \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TEXT-SUMMARIZATION\)](https://humboldt-wi.github.io/blog/tags/text-summarization)

🔗 [TIME-SERIES \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TIME-SERIES\)](https://humboldt-wi.github.io/blog/tags/time-series)

🔗 [TIME-SERIES-FORECASTING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TIME-SERIES-FORECASTING\)](https://humboldt-wi.github.io/blog/tags/time-series-forecasting)

🔗 [TOXIC-COMMENTS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TOXIC-COMMENTS\)](https://humboldt-wi.github.io/blog/tags/toxic-comments)

🔗 [TRANSFER-LEARNING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TRANSFER-LEARNING\)](https://humboldt-wi.github.io/blog/tags/transfer-learning)

🔗 [TRANSFORMERS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TRANSFORMERS\)](https://humboldt-wi.github.io/blog/tags/transformers)

🔗 [TREATMENT-EFFECT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TREATMENT-EFFECT\)](https://humboldt-wi.github.io/blog/tags/treatment-effect)

🔗 [TWITTER \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/TWITTER\)](https://humboldt-wi.github.io/blog/tags/twitter)

🔗 [ULMFIT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/ULMFIT\)](https://humboldt-wi.github.io/blog/tags/ulmfit)

🔗 [UNCERTAINTY \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/UNCERTAINTY\)](https://humboldt-wi.github.io/blog/tags/uncertainty)

🔗 [UPLIFT \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/UPLIFT\)](https://humboldt-wi.github.io/blog/tags/uplift)

🔗 [UPLIFT-MODELING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/UPLIFT-MODELING\)](https://humboldt-wi.github.io/blog/tags/uplift-modeling)

🔗 [UPLIFT-MODELLING \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/UPLIFT-MODELLING\)](https://humboldt-wi.github.io/blog/tags/uplift-modelling)

🔗 [VARIATIONAL-INFERENCE \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/VARIATIONAL-INFERENCE\)](https://humboldt-wi.github.io/blog/tags/variational-inference)

🔗 [WIKITEXT-103 \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/WIKITEXT-103\)](https://humboldt-wi.github.io/blog/tags/wikitext-103)

🔗 [WORD-EMBEDDINGS \(HTTPS://HUMBOLDT-WI.GITHUB.IO/BLOG/TAGS/WORD-EMBEDDINGS\)](https://humboldt-wi.github.io/blog/tags/word-embeddings)

Copyright (c) 2017, Chair of Information System at HU-Berlin; all rights reserved.

Template by Bootstrapious (<http://bootstrapious.com/free-templates>). Ported to Hugo by DevCows (<https://github.com/devcows/hugo-universal-theme>)