Overview: Nelder Mead Algo

What it is:

1. An unconstrained heuristic optimization algorithm
2. Can be used to find optimal model parameters with respect to an objective function
   - For example, the optimal parameters for an xgboost algorithm with respect to the AUC value.
3. Or simply to find coordinates of closed-form functions that returns the function's optimum value

What it means:

1. Unconstrained means special treatment must be made for optimizing on non-negative model parameters.
2. Heuristic optimization means that more often than not, the algorithm is likely to converge to a local optimum

Whys:

1. Nelder Mead (NM) R package for optimizing closed-form functions exists but none for problems like optimizing on the parameters of the xgboost are available
2. While likely to converge to local optima, the run time is much less than running an exhaustive search. This would be especially useful when optimizing on numerous parameters.
3. Easy to code and helpful for Kaggle competitions. At least for me ;)

Reference: https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method

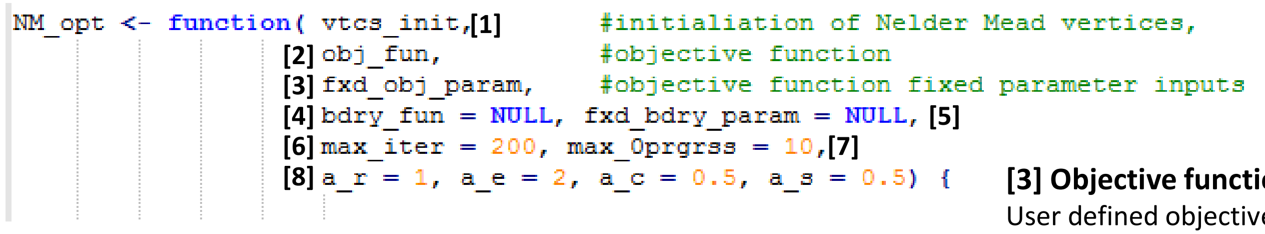
Understanding the function: NM_opt() inputs

[1] Initialization inputs:
NM is a simplex optimization algo, therefore when optimizing on an N dimensional problem, we need to define N+1
vertices to form a simplex.

- NM_opt() therefore will only take in (N+1) x N matrices (*N+1 inputs and N parameters*)

[2] Objective function:
User defined objective function to be passed. For optimizing on model parameters, like xgboost, we need to create a function wrapper that

1. wraps the model's function
2. that will subsequently pass the results to the objective function to evaluate and output the objective value.



```
NM_opt <- function( vtcs_init,[1]       #initialiation of Nelder Mead vertices,
                [2] obj_fun,            #objective function
                [3] fxd_obj_param,      #objective function fixed parameter inputs
                [4] bdry_fun = NULL, fxd_bdry_param = NULL, [5]
                [6] max_iter = 200, max_0prgrss = 10,[7]
                [8] a_r = 1, a_e = 2, a_c = 0.5, a_s = 0.5) {
```

[3] Objective function fixed inputs:
User defined objective function to be passed. For optimizing on model parameters, like xgboost, we need to create a function wrapper that

1. wraps the model's function
2. that will subsequently pass the results to the objective function to evaluate and output the objective value.

Hence, the wrapper effectively evaluates the objective function based on the choice of input parameters.

[4] "boundary" function:
As NM is an unconstrained optimization algorithm , the search for the optima will therefore have to be executed on the entire real number line.
However, many models' have parameters that are constrained to the positive numbers. This means that their inputs to "*vtcs_init*" will have to be mapped to the real number. This also means that there needs to be an internal inverse mapping function to map back the vertices to the model's parameter constraints for the model's ingestion. This function serves that purpose.

[5] "boundary" function fixed inputs:
Any fixed inputs to the user defined "boundary" function that may be required.

[6] max_iter:
The maximum number of iterations the algorithm will run before stopping.

[7] max_0prgrss:
A primitive stopping criterion that sets the number of iterations the algorithm will make without any improvements to the objective function before stopping.

[8] Nelder Mead Constants (optional):
The default Nelder Mead constants that dictate how much the simplex will deform in the search of an optimum objective value

a_r -> reflection constant
a_e -> expansion constant
a_c -> contraction constant
a_s -> shrinkage (or similarly, reduction) constant

Please refer to: https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method

An usage example: NM_opt()

[1] Initialization inputs:
(N+1) x N matrix. Where the rows, *N+1,* are the inputs that define the vertices and the columns*, N,* are parameters

- Here we have a wrapper for the xgboost function
- The xgb parameter inputs however are transformed.
    - log(x) , logarithmic transform for [0-inf) inputs and –log|(1/x) -1| or the inverse logistic transform, for [0,1] inputs.

[2] Objective function:
User defined function that runs the xgboost cross validation function and evaluates the corresponding AUC. Here, the wrapper is predefined to optimize on 8 variables.

[3] Objective function fixed inputs:
These are the fixed inputs to the xgboost wrapper function that are not optimized on for passing to user defined objective function.

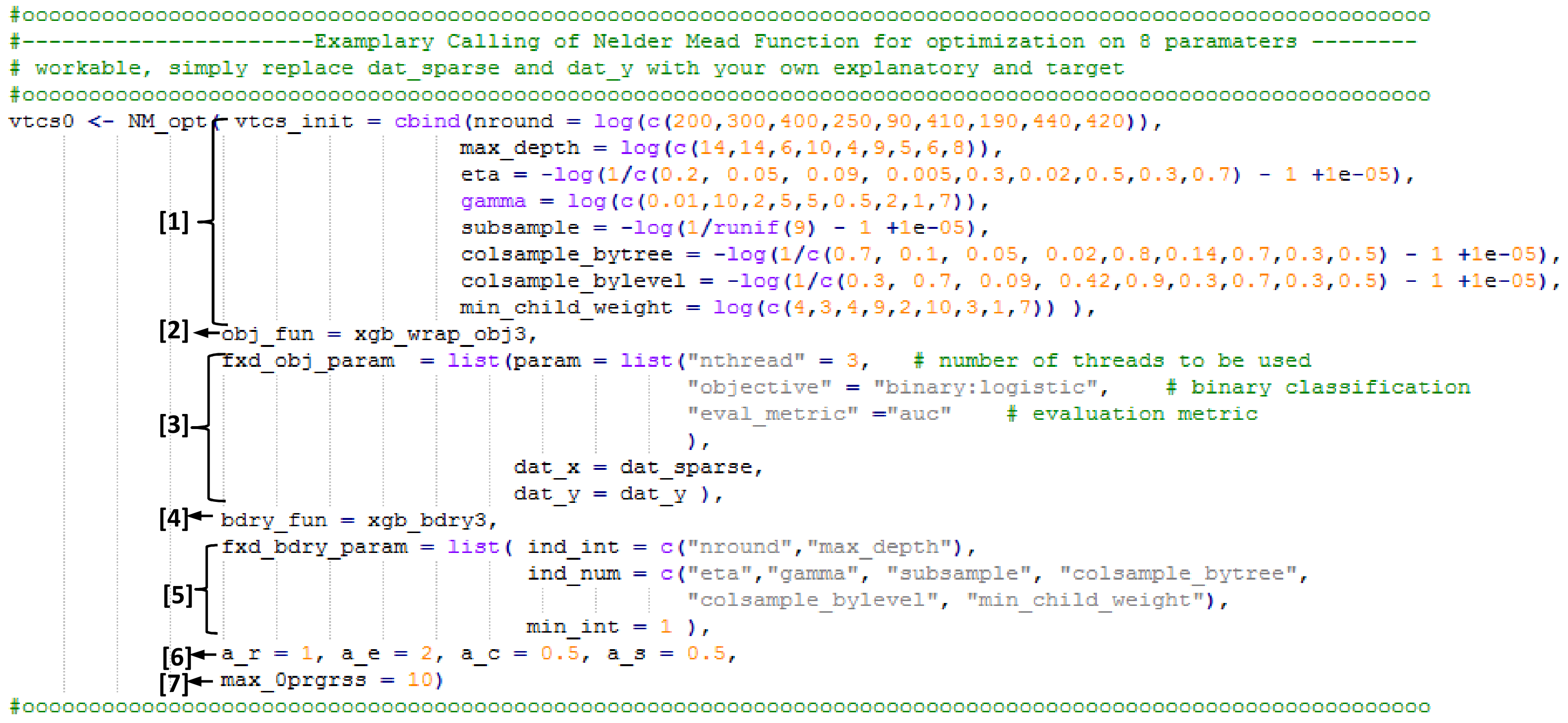
```
#ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
#-----------------------Examplary Calling of Nelder Mead Function for optimization on 8 paramaters --------
# workable, simply replace dat_sparse and dat_y with your own explanatory and target
#ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
vtcs0 <- NM_opt( vtcs_init = cbind(nround = log(c(200,300,400,250,90,410,190,440,420)),
                                  max_depth = log(c(14,14,6,10,4,9,5,6,8)),
                                  eta = -log(1/c(0.2, 0.05, 0.09, 0.005,0.3,0.02,0.5,0.3,0.7) - 1 +1e-05),
                                  gamma = log(c(0.01,10,2,5,5,0.5,2,1,7)),
            [1]                   subsample = -log(1/runif(9) - 1 +1e-05),
                                  colsample_bytree = -log(1/c(0.7, 0.1, 0.05, 0.02,0.8,0.14,0.7,0.3,0.5) - 1 +1e-05),
                                  colsample_bylevel = -log(1/c(0.3, 0.7, 0.09, 0.42,0.9,0.3,0.7,0.3,0.5) - 1 +1e-05),
                                  min_child_weight = log(c(4,3,4,9,2,10,3,1,7)) ),
            [2] obj_fun = xgb_wrap_obj3,
                fxd_obj_param  = list(param = list("nthread" = 3,   # number of threads to be used
                                                   "objective" = "binary:logistic",    # binary classification
            [3]                                    "eval_metric" ="auc"    # evaluation metric
                                                   ),
                                      dat_x = dat_sparse,
                                      dat_y = dat_y ),
            [4] bdry_fun = xgb_bdry3,
                fxd_bdry_param = list( ind_int = c("nround","max_depth"),
            [5]                        ind_num = c("eta","gamma", "subsample", "colsample_bytree",
                                                   "colsample_bylevel", "min_child_weight"),
                                       min_int = 1 ),
            [6] a_r = 1, a_e = 2, a_c = 0.5, a_s = 0.5,
            [7] max_0prgrss = 10)
#ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

An usage example: NM_opt()

[4] boundary function:
User defined functions, for mapping back “unconstrained values” to function parameter boundaries.
- Here the function serves to execute an inverse mapping of the transformed unconstrained parameters for xgboost’s ingestion.

[5] boundary function inputs:
While the boundary function is meant to ingest and transform data, the b

[6] Objective function fixed inputs:
These are the fixed inputs or inputs that are not optimized on for passing to user defined objective function.



```
#ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
#-----------------------Examplary Calling of Nelder Mead Function for optimization on 8 paramaters --------
# workable, simply replace dat_sparse and dat_y with your own explanatory and target
#ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
vtcs0 <- NM_opt( vtcs_init = cbind(nround = log(c(200,300,400,250,90,410,190,440,420)),
                                  max_depth = log(c(14,14,6,10,4,9,5,6,8)),
                                  eta = -log(1/c(0.2, 0.05, 0.09, 0.005,0.3,0.02,0.5,0.3,0.7) - 1 +1e-05),
                                  gamma = log(c(0.01,10,2,5,5,0.5,2,1,7)),
                                  subsample = -log(1/runif(9) - 1 +1e-05),
                                  colsample_bytree = -log(1/c(0.7, 0.1, 0.05, 0.02,0.8,0.14,0.7,0.3,0.5) - 1 +1e-05),
                                  colsample_bylevel = -log(1/c(0.3, 0.7, 0.09, 0.42,0.9,0.3,0.7,0.3,0.5) - 1 +1e-05),
                                  min_child_weight = log(c(4,3,4,9,2,10,3,1,7)) ),
                obj_fun = xgb_wrap_obj3,
                fxd_obj_param  = list(param = list("nthread" = 3,   # number of threads to be used
                                                  "objective" = "binary:logistic",    # binary classification
                                                  "eval_metric" ="auc"    # evaluation metric
                                                  ),
                                      dat_x = dat_sparse,
                                      dat_y = dat_y ),
                bdry_fun = xgb_bdry3,
                fxd_bdry_param = list( ind_int = c("nround","max_depth"),
                                       ind_num = c("eta","gamma", "subsample", "colsample_bytree",
                                                  "colsample_bylevel", "min_child_weight"),
                                       min_int = 1 ),
                a_r = 1, a_e = 2, a_c = 0.5, a_s = 0.5,
                max_0prgrss = 10)
#ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```