1 Loss Function and its Derivatives

In this Section, we write down the math. We will use the symbol $\bigoplus$ to denote the log-space summation operator: $\bigoplus_{i=1}^{N} \log(a_i) = \log\left(\sum_{i=1}^{N} a_i\right)$. Writing expressions using this notation will allow us to obtain numerically stable computational expressions.

Let $m \in \{1, \ldots, M\}$ index component predictive models and $i \in \{1, \ldots, N\}$ index prediction cases in the training data. For example, in a seasonal time series prediction context i may index times at which we make predictions for seasonal quantities, or combination of a time at which we make a prediction and the prediction horizon for predictions at individual weeks. Let $f_m(y_i|\mathbf{x}_i)$ denote the predictive density from model m for the value of the random variable Y_i. The $\mathbf{x}_i$ is a vector observed covariates which may be used by any of the component models as conditioning variables in forming the predictive distribution, and is also used in calculating the component model weights. Note that the component models and computation of the component model weights may use a proper subset of the variables in $\mathbf{x}_i$.

The combined predictive density for case i is

$$f(y_i|\mathbf{x}_i) = \sum_{m=1}^{M} \pi_m(\mathbf{x}_i) f_m(y_i|\mathbf{x}_i), \text{ where} \tag{1}$$

$$\pi_m(\mathbf{x}_i) = \frac{\exp\{\rho_m(\mathbf{x}_i)\}}{\sum_{m'=1}^{M} \exp\{\rho_{m'}(\mathbf{x}_i)\}} \tag{2}$$

In Equation (1) the $\pi_m(\mathbf{x}_i)$ are the model weights, which we regard as functions of $\mathbf{x}_i$. These weights must be non-negative and sum to 1 across m. We ensure that these constraints are met by parameterizing the $\pi_m(\mathbf{x}_i)$ in terms of the softmax transformation of real-valued functions $\rho_m(\mathbf{x}_i)$ in Equation (2). For notational brevity, we will suppress the expression of these quantities as functions of $\mathbf{x}_i$ and write $\rho_m(\mathbf{x}_i) = \rho_{mi}$ with $\boldsymbol{\rho} = (\rho_{11}, \ldots, \rho_{MN})$, and $\pi_m(\mathbf{x}_i) = \pi_{mi}$ with $\boldsymbol{\pi} = (\pi_{11}, \ldots, \pi_{MN})$.

Our goal is to estimate the functions ρ_{mi}. To do this, we require as inputs cross-validated estimates of $\log\{f_m(y_i|\mathbf{x}_i)\}$ for each component model m and case i in the training data; we denote these values by $\log\{f_m^{cv}(y_i|\mathbf{x}_i)\}$. We will focus on optimization of the log-score of the combined predictive distribution for now; we may consider other loss functions in the future. Considered as a function of the vector of values ρ_{mi} for each combination of m and i, this loss function is given by

$$L(\boldsymbol{\rho}) = \sum_{i=1}^{N} \log\{f(y_i|\mathbf{x}_i)\} \tag{3}$$

$$= \sum_{i=1}^{N} \log\left\{\sum_{m=1}^{M} \pi_{mi} f_m(y_i|\mathbf{x}_i)\right\} \tag{4}$$

$$= \sum_{i=1}^{N} \log\left\{\sum_{m=1}^{M} \frac{\exp(\rho_{mi})}{\sum_{m'=1}^{M} \exp(\rho_{m'i})} f_m(y_i|\mathbf{x}_i)\right\} \tag{5}$$

We must find the first and second order partial derivatives of L with respect to each $\rho_{m^*i^*}$.

$$\frac{\partial}{\partial \rho_{m^*i^*}} L(\boldsymbol{\rho}) = \frac{\partial}{\partial \rho_{m^*i^*}} \sum_{i=1}^{N} \log\left\{\sum_{m=1}^{M} \frac{\exp(\rho_{mi})}{\sum_{m'=1}^{M} \exp(\rho_{m'i})} f_m(y_i|\mathbf{x}_i)\right\} \tag{6}$$

$$= \left\{\frac{1}{\sum_{m=1}^{M} \frac{\exp(\rho_{mi^*})}{\sum_{m'=1}^{M} \exp(\rho_{m'i^*})} f_m(y_{i^*}|\mathbf{x}_{i^*})}\right\} \times \frac{\partial}{\partial \rho_{m^*i^*}} \sum_{m=1}^{M} \frac{\exp(\rho_{mi^*})}{\sum_{m'=1}^{M} \exp(\rho_{m'i^*})} f_m(y_{i^*}|\mathbf{x}_{i^*}) \tag{7}$$

Now note that for $m^* = m$,

$$\begin{aligned}
\frac{\partial}{\partial \rho_{m^* i^*}} \pi_{m i^*} &= \frac{\partial}{\partial \rho_{m^* i^*}} \frac{\exp(\rho_{m i^*})}{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})} \\
&= \frac{\exp(\rho_{m i^*}) \left\{ \sum_{m'=1}^{M} \exp(\rho_{m' i^*}) \right\} - \exp(\rho_{m i^*})^2}{\left\{ \sum_{m'=1}^{M} \exp(\rho_{m' i^*}) \right\}^2} \\
&= \pi_{m i^*} - \pi_{m i^*}^2 .
\end{aligned}$$

For $m^* \neq m$,

$$\begin{aligned}
\frac{\partial}{\partial \rho_{m^* i^*}} \pi_{m i^*} &= \frac{\partial}{\partial \rho_{m^* i^*}} \frac{\exp(\rho_{m i^*})}{\sum_{m'=1}^{M} \exp(\rho_{m' i^*})} \\
&= \frac{-\exp(\rho_{m i^*}) \exp(\rho_{m^* i^*})}{\left\{ \sum_{m'=1}^{M} \exp(\rho_{m' i^*}) \right\}^2} \\
&= -\pi_{m i^*} \pi_{m^* i^*} .
\end{aligned}$$

Substituting these results into Equation (7), we obtain

$$\frac{\partial}{\partial \rho_{m^* i^*}} L(\boldsymbol{\rho}) = \left\{ \frac{1}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \right\} \left\{ \pi_{m^* i^*} \left(f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) - \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*}) \right) \right\} \tag{8}$$

$$= \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} - \pi_{m^* i^*} \tag{9}$$

Now, we calculate the second order derivative as

$$\frac{\partial^2}{\partial \rho_{m^* i^*}^2} L(\boldsymbol{\rho}) = \frac{\partial}{\partial \rho_{m^* i^*}} \left[\frac{\partial}{\partial \rho_{m^* i^*}} L(\boldsymbol{\rho}) \right] \tag{10}$$

$$= \frac{\partial}{\partial \rho_{m^* i^*}} \left[\frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} - \pi_{m^* i^*} \right] \tag{11}$$

$$= \frac{\left(\pi_{m^* i^*} - \pi_{m^* i^*}^2 \right) f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})}{\left\{ \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*}) \right\}^2} \tag{12}$$

$$- \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) \left\{ \pi_{m^* i^*} \left(f_{m^*}(y_{i^*} | \mathbf{x}_{i^*}) - \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*}) \right) \right\}}{\left\{ \sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*}) \right\}^2} \tag{13}$$

$$- \left(\pi_{m^* i^*} - \pi_{m^* i^*}^2 \right) \tag{14}$$

$$= (1 - \pi_{m^* i^*}) \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \tag{15}$$

$$- \left[\frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \right]^2 + \pi_{m^* i^*} \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \tag{16}$$

$$- \left(\pi_{m^* i^*} - \pi_{m^* i^*}^2 \right) \tag{17}$$

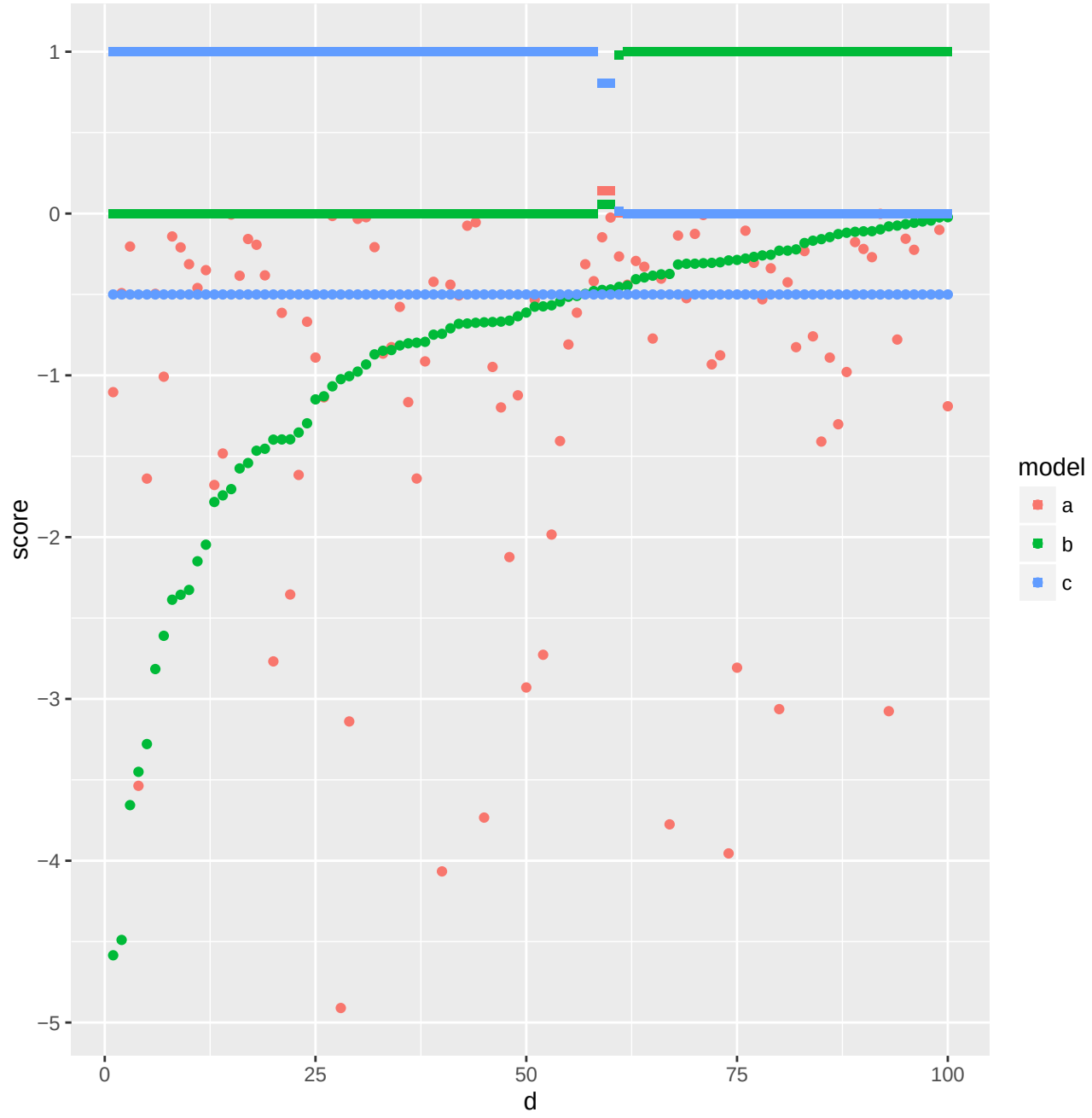
$$= \frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} - \left[\frac{\pi_{m^* i^*} f_{m^*}(y_{i^*} | \mathbf{x}_{i^*})}{\sum_{m=1}^{M} \pi_{m i^*} f_m(y_{i^*} | \mathbf{x}_{i^*})} \right]^2 - \left(\pi_{m^* i^*} - \pi_{m^* i^*}^2 \right) \tag{18}$$

2 Simulated Application

```
library(tidyr)
library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##    filter, lag
## The following objects are masked from 'package:base':
##
##    intersect, setdiff, setequal, union

library(xgboost)

##
## Attaching package:  'xgboost'
## The following object is masked from 'package:dplyr':
##
##    slice

library(xgbstack)
library(ggplot2)

### For now, let's just make up some data for the purposes of method development
### this will need to go into test code too
set.seed(9873)
loso_pred_res <- data.frame(
  model = paste0("log_score_", rep(letters[1:3], each = 100)),
  d = rep(1:100, times = 3),
  loso_log_score = c(
    log(runif(100, 0, 1)), # model a's performance not related to d
    sort(log(runif(100, 0, 1))), # model b's performance increasing in d
    rep(-0.5, 100))  # model c's performance constant
) %>%
  spread(model, loso_log_score)

# ggplot(loso_pred_res) + geom_point(aes(x = d, y = loso_log_score, colour = model, group = model))

## convert training data to format used in xgboost
## refactor this into part of fit method eventually
component_models <- letters[1:3]
predictive_vars <- "d"
dtrain <- xgb.DMatrix(
  data = as.matrix(loso_pred_res[, predictive_vars, drop = FALSE]) %>%
    `storage.mode<-`("double"))

### call xbg.train to do estimation.
### refactor this to be part of fit method eventually.
obj_deriv_fn <- get_obj_deriv_fn(component_model_log_scores = as.matrix(
      loso_pred_res[, paste0("log_score_", component_models), drop = FALSE]
    ) %>%
    `storage.mode<-`("double"))
```

```r
# fit_1round <- xgb.train(
#   params = list(
#     booster = "gbtree", # change to gblinear to fit lines
#     subsample = 1, # use half of the observations in each boosting iteration
#     colsample_bytree = 1, # use all of the columns to do prediction (for now, only one column...)
#     max_depth = 10,
#     gamma = 0,
#     num_class = length(component_models)
#   ),
#   data = dtrain,
#   nrounds = 1,
#   obj = obj_deriv_fn,
#   verbose = 2
# )
#
# predictions_1round <- predict(fit_1round, newdata = dtrain)
#
# preds_1round <- preds_to_matrix(preds = predictions_1round, num_models = length(component_models))
#
#
# debug(obj_deriv_fn)
fit <- xgb.train(
  params = list(
    booster = "gbtree", # change to gblinear to fit lines
    subsample = 1, # use half of the observations in each boosting iteration
    colsample_bytree = 1, # use all of the columns to do prediction (for now, only one column...)
    max_depth = 10,
    min_child_weight = 0,
    gamma = 0,
    num_class = length(component_models)
  ),
  data = dtrain,
  nrounds = 1000,
  obj = obj_deriv_fn,
  verbose = 0
)

#xgb.plot.tree("d", model = fit)

predictions <- predict(fit, newdata = dtrain)

preds <- preds_to_matrix(preds = predictions, num_models = length(component_models))
#dim(preds) <- rev(dim(preds))
#preds <- t(preds)

component_model_scores_df <- as.data.frame(as.matrix(
      loso_pred_res[, paste0("log_score_", component_models), drop = FALSE]
    ) %>%
    `storage.mode<-`("double")) %>%
  `colnames<-`(letters[1:3]) %>%
  gather_("model", "score", letters[1:3]) %>%
  mutate(d = rep(1:100, 3))

log_denom <- logspace_sum_matrix_rows(preds)
```

```
component_model_weights_df <- as.data.frame(exp(sweep(preds, 1, log_denom, `-`))) %>%
  `colnames<-`(letters[1:3])

component_model_weights_df <- component_model_weights_df %>%
  gather_("model", "weight", letters[1:3]) %>%
  mutate(d = rep(1:100, 3))

ggplot() +
  geom_point(aes(x = d, y = score, colour = model), data = component_model_scores_df) +
  geom_point(aes(x = d, y = weight, colour = model), shape = 15, data = component_model_weights_df)
```



```
xgb.plot.tree <-
function (feature_names = NULL, filename_dump = NULL, model = NULL,
    n_first_tree = NULL, CSSstyle = NULL, width = NULL, height = NULL)
{
    if (!(class(CSSstyle) %in% c("character", "NULL") && length(CSSstyle) <=
        1)) {
        stop("style: Has to be a character vector of size 1.")
    }
    if (!class(model) %in% c("xgb.Booster", "NULL")) {
        stop("model: Has to be an object of class xgb.Booster model generaged by the xgb.train function.")
    }
    if (!requireNamespace("DiagrammeR", quietly = TRUE)) {
        stop("DiagrammeR package is required for xgb.plot.tree",
            call. = FALSE)
    }
    if (is.null(model)) {
        allTrees <- xgb.model.dt.tree(feature_names = feature_names,
            filename_dump = filename_dump, n_first_tree = n_first_tree)
    }
    else {
        allTrees <- xgb.model.dt.tree(feature_names = feature_names,
            model = model, n_first_tree = n_first_tree)
    }
    allTrees[Feature != "Leaf", `:=`(yesPath, paste(ID, "(",
        Feature, "<br/>Cover: ", Cover, "<br/>Gain: ", Quality,
        ")-->|< ", Split, "|", Yes, ">", Yes.Feature, "]", sep = ""))]
    allTrees[Feature != "Leaf", `:=`(noPath, paste(ID, "(", Feature,
        ")-->|>= ", Split, "|", No, ">", No.Feature, "]", sep = ""))]
    if (is.null(CSSstyle)) {
        CSSstyle <- "classDef greenNode fill:#A2EB86, stroke:#04C4AB, stroke-width:2px;classDef redNode fi
    }
    yes <- allTrees[Feature != "Leaf", c(Yes)] %>% paste(collapse = ",") %>%
        paste("class ", ., " greenNode", sep = "")
    no <- allTrees[Feature != "Leaf", c(No)] %>% paste(collapse = ",") %>%
        paste("class ", ., " redNode", sep = "")
    path <- allTrees[Feature != "Leaf", c(yesPath, noPath)] %>%
        .[order(.)] %>% paste(sep = "", collapse = ";") %>% paste("graph LR",
        ., collapse = "", sep = ";") %>% paste(CSSstyle, yes,
        no, sep = ";")
    DiagrammeR::mermaid(path, width, height)
}

xgb.model.dt.tree <-
function (feature_names = NULL, filename_dump = NULL, model = NULL,
    text = NULL, n_first_tree = NULL)
{
    if (!class(feature_names) %in% c("character", "NULL")) {
        stop("feature_names: Has to be a vector of character or NULL if the model dump already contains fe
    }
    if (!(class(filename_dump) %in% c("character", "NULL") &&
        length(filename_dump) <= 1)) {
        stop("filename_dump: Has to be a character vector of size 1 representing the path to the model dum
    }
    else if (!is.null(filename_dump) && !file.exists(filename_dump)) {
```

```
        stop("filename_dump: path to the model doesn't exist.")
    }
    else if (is.null(filename_dump) && is.null(model) && is.null(text)) {
        stop("filename_dump & model & text: no path to dump model, no model, no text dump, have been provi
    }
    if (!class(model) %in% c("xgb.Booster", "NULL")) {
        stop("model: Has to be an object of class xgb.Booster model generaged by the xgb.train function.")
    }
    if (!class(text) %in% c("character", "NULL")) {
        stop("text: Has to be a vector of character or NULL if a path to the model dump has already been p
    }
    if (!class(n_first_tree) %in% c("numeric", "NULL") | length(n_first_tree) >
        1) {
        stop("n_first_tree: Has to be a numeric vector of size 1.")
    }
    if (!is.null(model)) {
        text = xgb.dump(model = model, with.stats = T)
    }
    else if (!is.null(filename_dump)) {
        text <- readLines(filename_dump) %>% str_trim(side = "both")
    }
    position <- str_match(text, "booster") %>% is.na %>% not %>%
        which %>% c(length(text) + 1)
    extract <- function(x, pattern) str_extract(x, pattern) %>%
        str_split("=") %>% lapply(function(x) x[2] %>% as.numeric) %>%
        unlist
    n_round <- min(length(position) - 1, n_first_tree)
    addTreeId <- function(x, i) paste(i, x, sep = "-")
    allTrees <- data.table()
    anynumber_regex <- "[-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?"
    for (i in 1:n_round) {
        tree <- text[(position[i] + 1):(position[i + 1] - 1)]
        if (length(tree) < 2)
            next
        treeID <- i - 1
        notLeaf <- str_match(tree, "leaf") %>% is.na
        leaf <- notLeaf %>% not %>% tree[.]
        branch <- notLeaf %>% tree[.]
        idBranch <- str_extract(branch, "\\d*:") %>% str_replace(":",
            "") %>% addTreeId(treeID)
        idLeaf <- str_extract(leaf, "\\d*:") %>% str_replace(":",
            "") %>% addTreeId(treeID)
        featureBranch <- str_extract(branch, "f\\d*<") %>% str_replace("<",
            "") %>% str_replace("f", "") %>% as.numeric
        if (!is.null(feature_names)) {
            featureBranch <- feature_names[featureBranch + 1]
        }
        featureLeaf <- rep("Leaf", length(leaf))
        splitBranch <- str_extract(branch, paste0("<", anynumber_regex,
            "\\]")) %>% str_replace("<", "") %>% str_replace("\\]",
            "")
        splitLeaf <- rep(NA, length(leaf))
        yesBranch <- extract(branch, "yes=\\d*") %>% addTreeId(treeID)
        yesLeaf <- rep(NA, length(leaf))
```

```
        noBranch <- extract(branch, "no=\\d*") %>% addTreeId(treeID)
        noLeaf <- rep(NA, length(leaf))
        missingBranch <- extract(branch, "missing=\\d+") %>%
            addTreeId(treeID)
        missingLeaf <- rep(NA, length(leaf))
        qualityBranch <- extract(branch, paste0("gain=", anynumber_regex))
        qualityLeaf <- extract(leaf, paste0("leaf=", anynumber_regex))
        coverBranch <- extract(branch, "cover=\\d*\\.*\\d*")
        coverLeaf <- extract(leaf, "cover=\\d*\\.*\\d*")
        dt <- data.table(ID = c(idBranch, idLeaf), Feature = c(featureBranch,
            featureLeaf), Split = c(splitBranch, splitLeaf),
            Yes = c(yesBranch, yesLeaf), No = c(noBranch, noLeaf),
            Missing = c(missingBranch, missingLeaf), Quality = c(qualityBranch,
                qualityLeaf), Cover = c(coverBranch, coverLeaf))[order(ID)][,
            `:=`(Tree, treeID)]
        allTrees <- rbindlist(list(allTrees, dt), use.names = T,
            fill = F)
    }
    yes <- allTrees[!is.na(dt$Yes), dt$Yes]
    set(allTrees, i = which(allTrees[, Feature] != "Leaf"), j = "Yes.Feature",
        value = allTrees[ID %in% yes, Feature])
    set(allTrees, i = which(allTrees[, Feature] != "Leaf"), j = "Yes.Cover",
        value = allTrees[ID %in% yes, Cover])
    set(allTrees, i = which(allTrees[, Feature] != "Leaf"), j = "Yes.Quality",
        value = allTrees[ID %in% yes, Quality])
    no <- allTrees[!is.na(No), No]
    set(allTrees, i = which(allTrees[, Feature] != "Leaf"), j = "No.Feature",
        value = allTrees[ID %in% no, Feature])
    set(allTrees, i = which(allTrees[, Feature] != "Leaf"), j = "No.Cover",
        value = allTrees[ID %in% no, Cover])
    set(allTrees, i = which(allTrees[, Feature] != "Leaf"), j = "No.Quality",
        value = allTrees[ID %in% no, Quality])
    allTrees
}

xgb.plot.tree("d", model = fit)

## Error in eval(expr, envir, enclos):  could not find function "str_match"
```