Comparing Different versions of Random Forests

TODOs

1. Update everything with the recent runs on SCF.
2. Try to find out where and why we are doing poorly.
3. The next batch of observations will have several seeds per run, aggregate the data over those runs to become robust against randomization errors

Robustness

How often does our algorithm fail, when other algorithms don't fail?

```
##                       did_not_run
##                        FALSE TRUE
##   hte_adaptive_nomsp  1349   22
##   hte_adaptive_wmsp   1349   22
##   hte_honest_wmsp     1222  149
##   ranger              1357   14
```

We can see that our algorithm fails in less than 2% of all cases, but more often than ranger. A comparison with randomForest is impossible, since it fails in rougly 30% of all cases, since it does not work for big data sets.

```
##                       did_not_run
##                        FALSE TRUE
##   hte_adaptive_nomsp  1349   22
##   hte_adaptive_wmsp   1349   22
##   hte_honest_wmsp     1222  149
##   ranger              1357   14
```

Why does it sometimes fail?

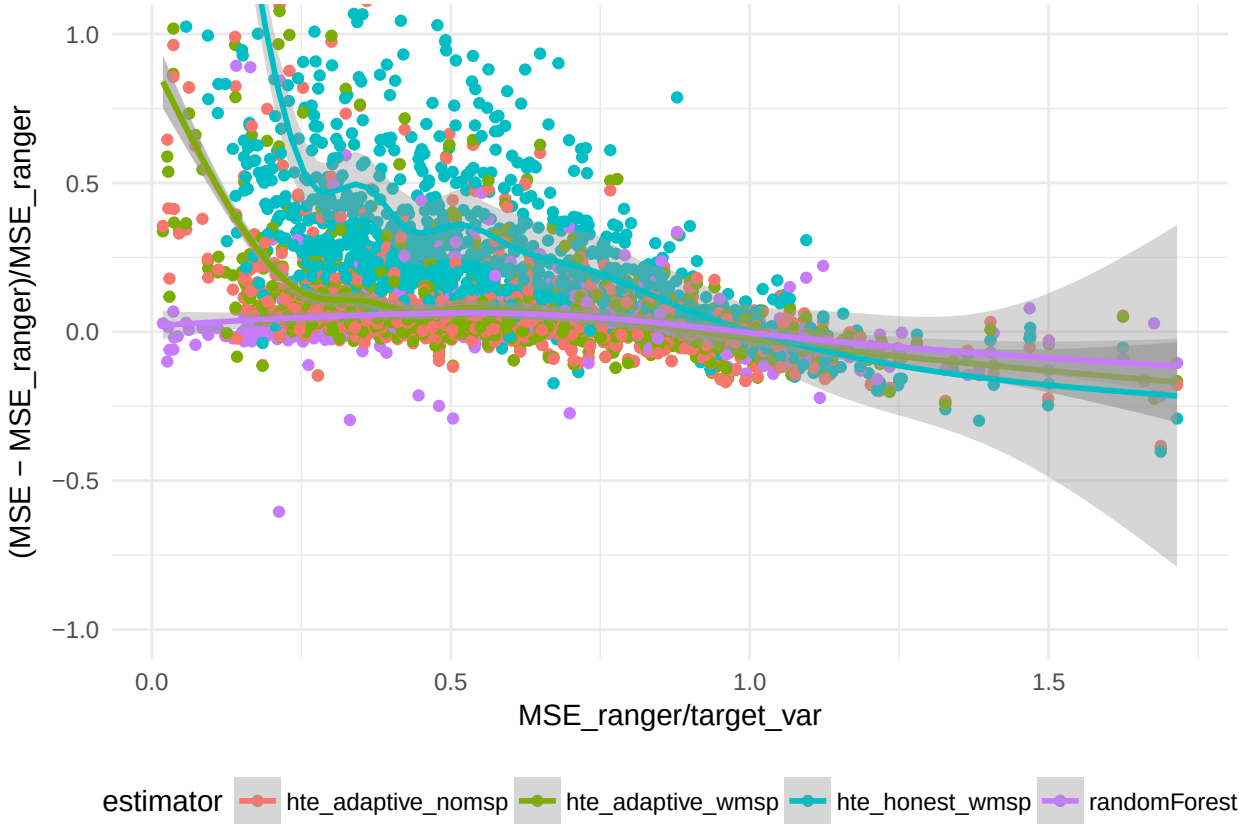
```
##          more_features_than_obs
## hte_fails FALSE TRUE
##     FALSE   381  968
##     TRUE      1   21
```

Conclusion:

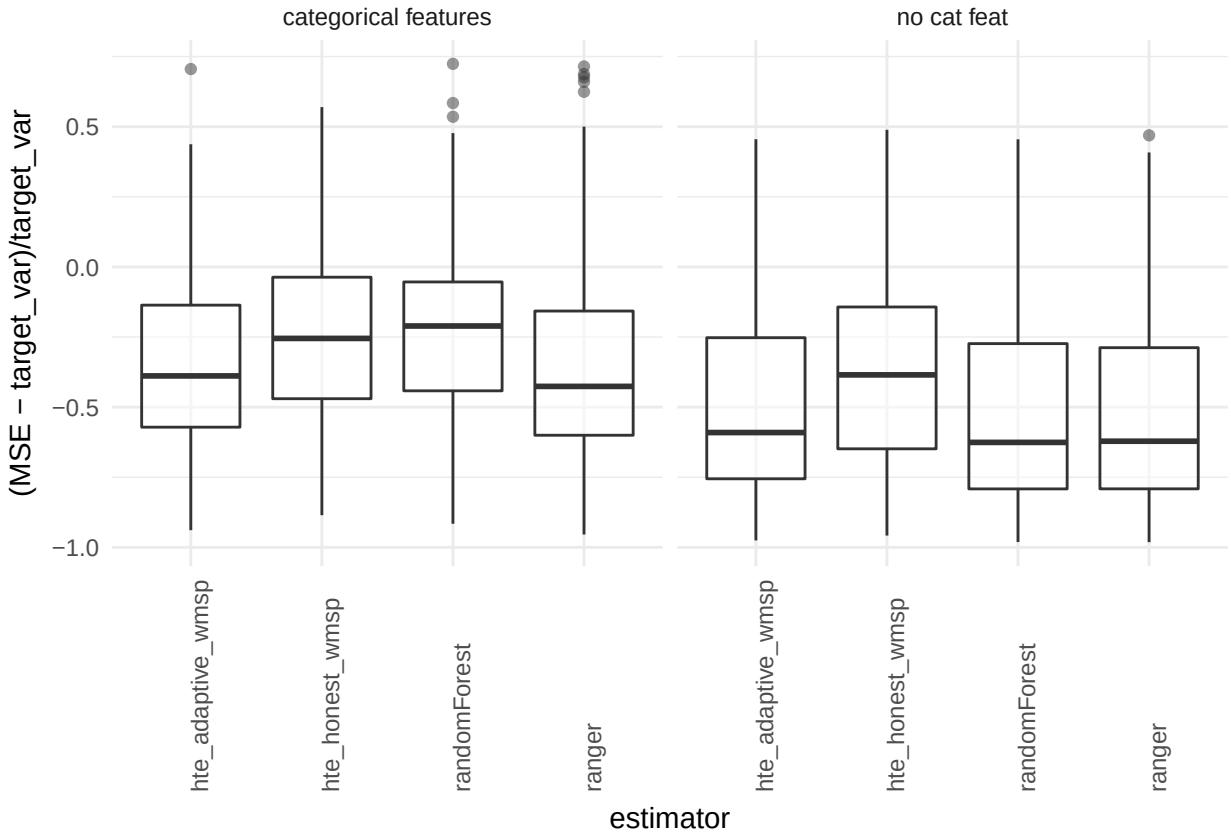
- Out of the 22 data sets where our algorithm fails, 21 have more features than observations. And all data sets ranger fails we fail as well.
- We are still much better than randomForest
- We should reset parameters automatically such that when the parameters are choosen too big or too small, that we readjust them such that no error occurs. E.g. we often fail because the leaf size is choosen too big. In such a case, I think it would be better to set the leaf size to the biggest possible value, which does not cause an error and throw a warning, or to return just the mean of all y values.

MSE Performance

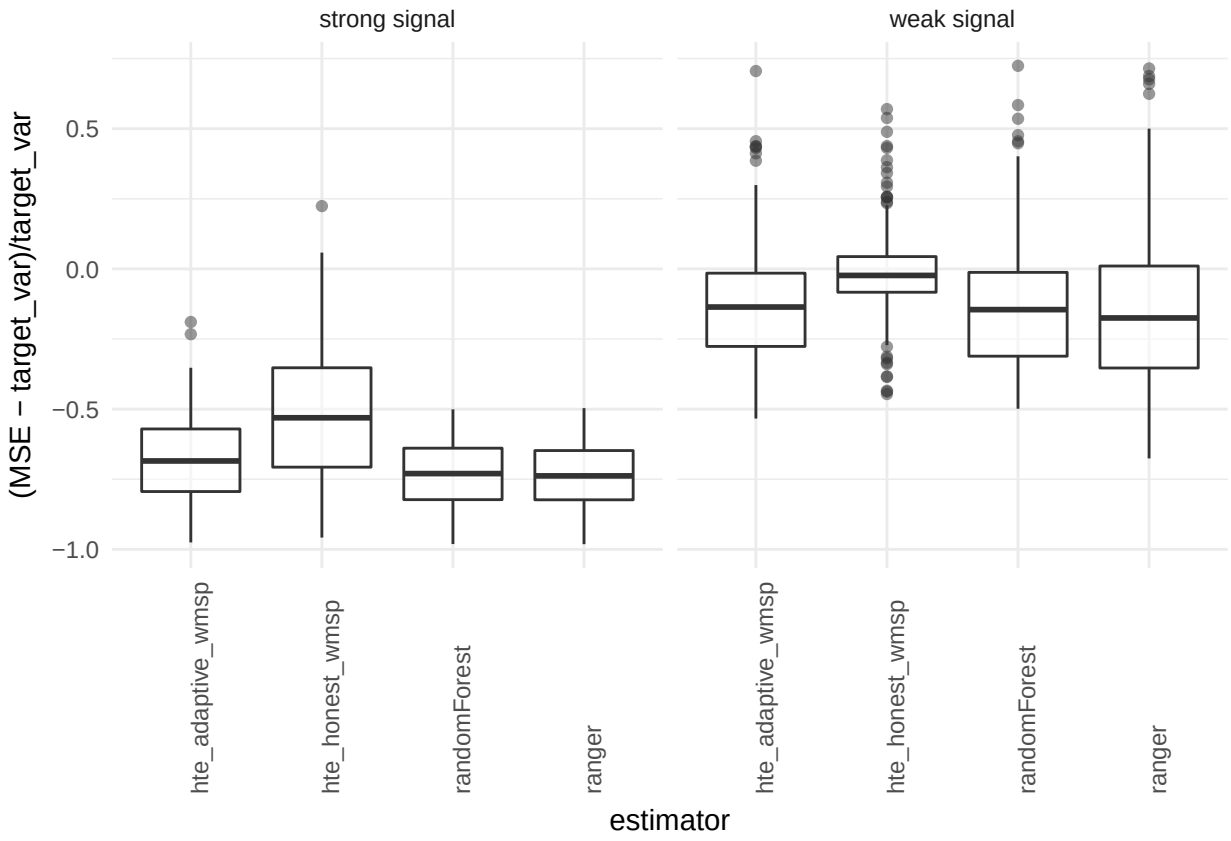
Performance relative to the explained variance



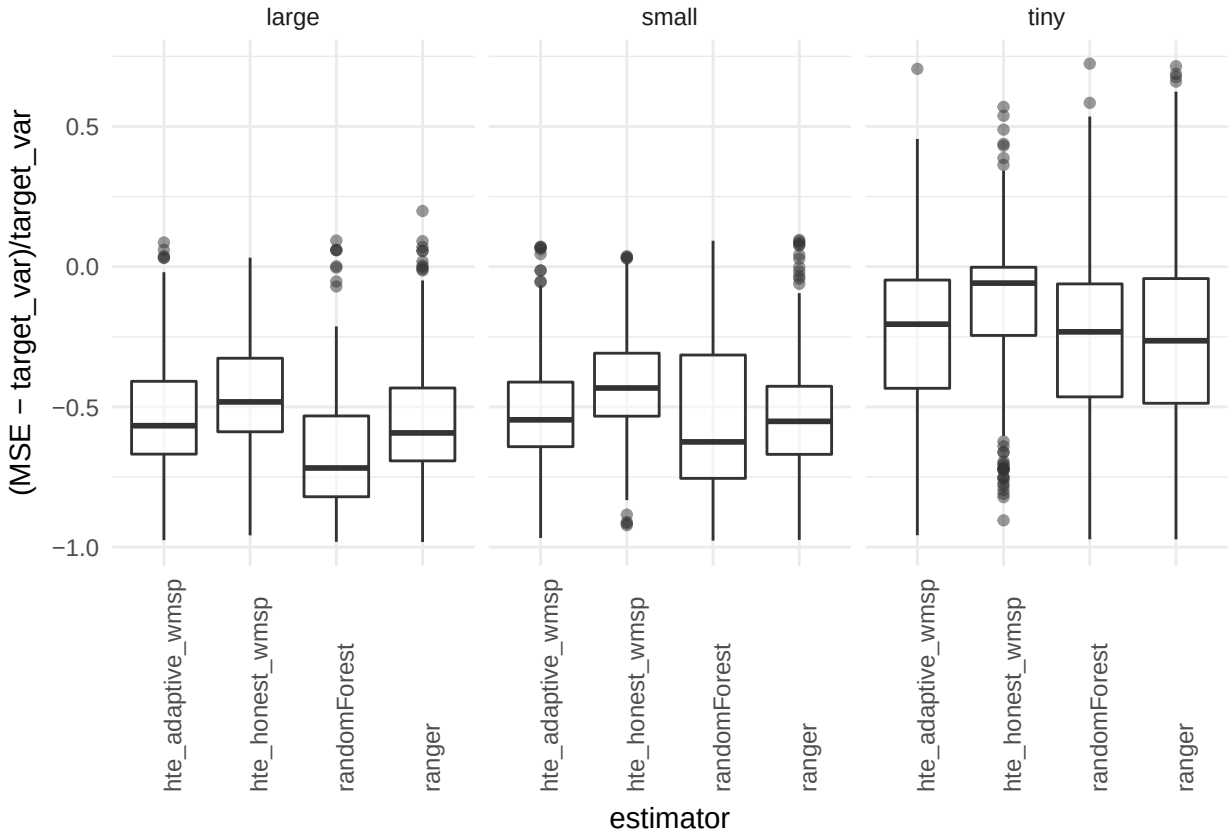
Performance for data set with and without categorical variables



Performance for data set with high and low signal to noise ratio

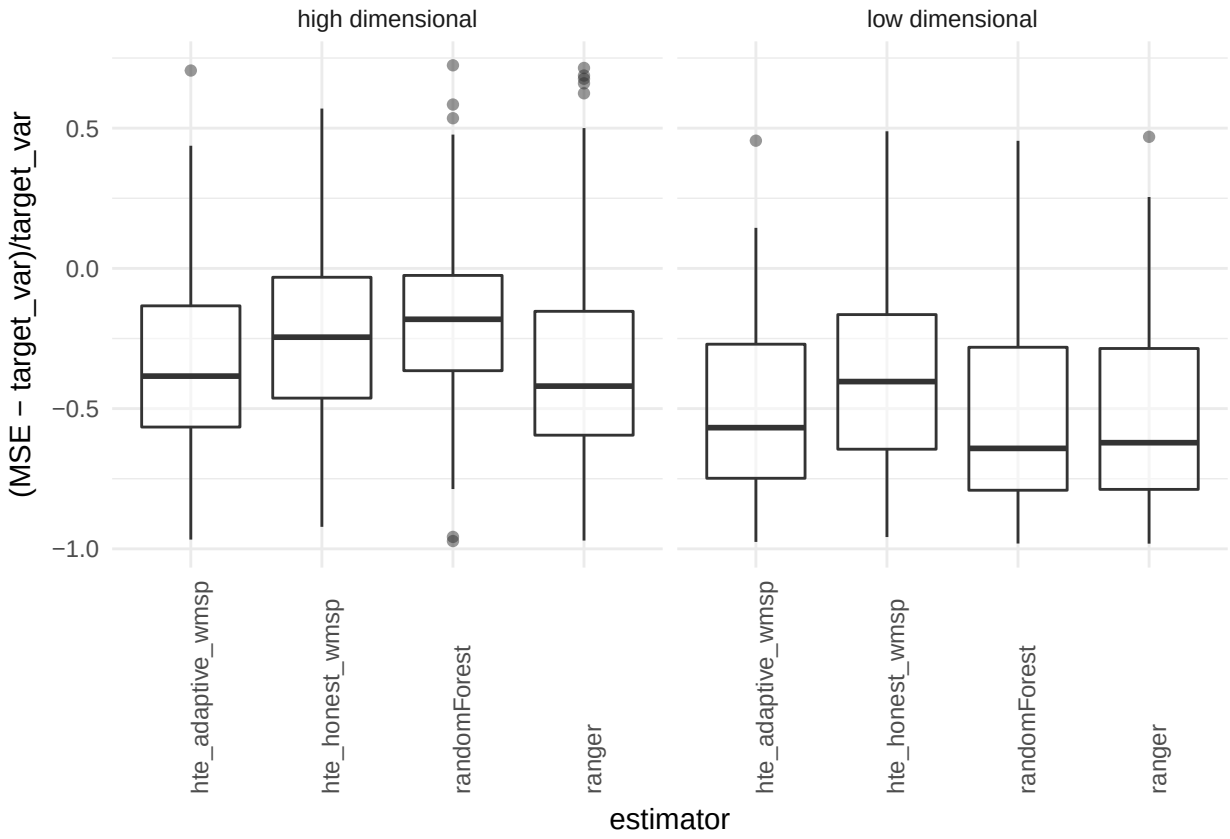


Performance in terms of the size of the data sets



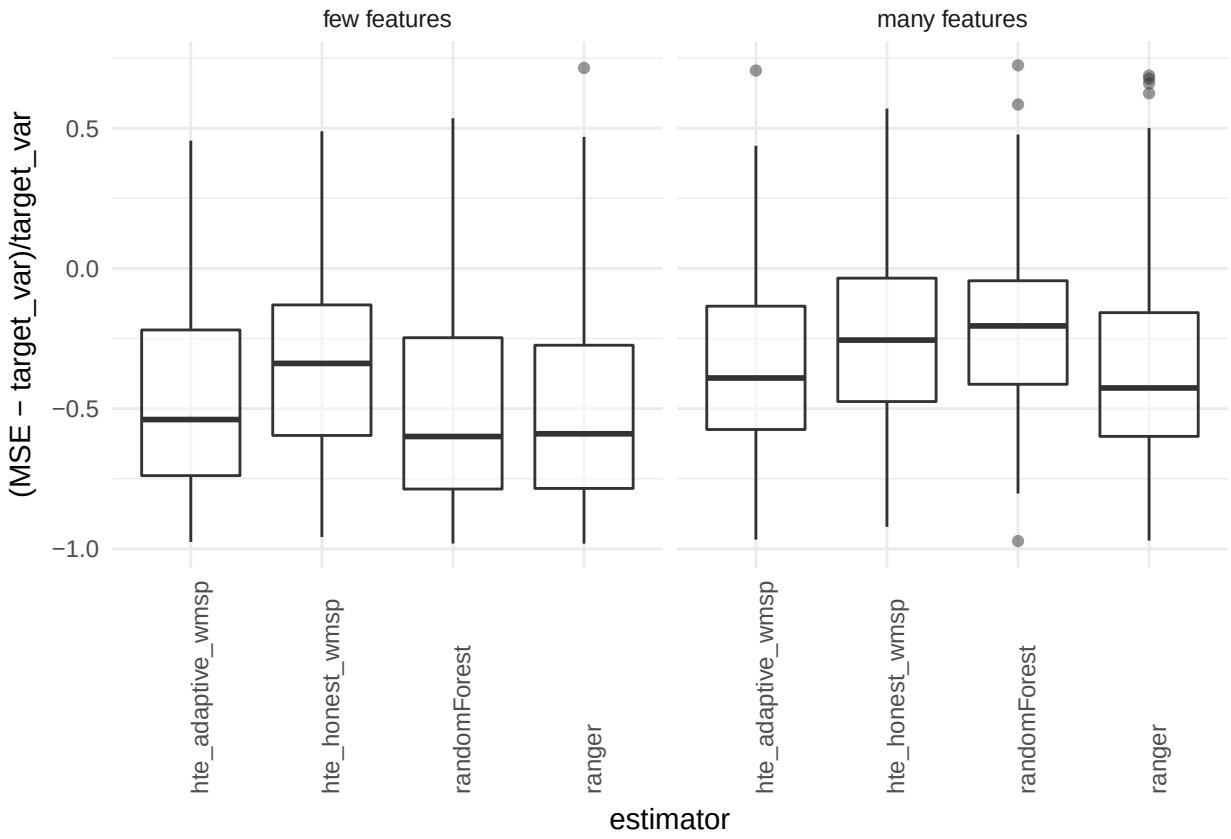
Performance in terms of the high and low dimensional

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0010  0.6418  2.6764  8.4412 12.8125 82.8953
```



Performance in terms of many and few features

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0  1025.0  1025.0   819.6  1025.0  7129.0
```



Run a linear regression to gain intuition for interactions

```
summary(
    lm(
      I(MSE > MSE_ranger) ~ size +
        size:(hasCategoricalFeat +
              number.of.features  +
              signal_strength +
                feat_obs_ratio),
      data = benchmark_aggr %>% filter(estimator == 'hte_adaptive_wmsp') %>%
        mutate(
          hasCategoricalFeat = number.of.features != number.of.numeric.features,
          signal_strength = 1 - MSE_randomForest / target_var,
          feat_obs_ratio = n_features / n_obs,
          size = ifelse(
            number.of.instances <= 209,
            'tiny',
            ifelse(
              number.of.instances < 500,
              'small',
              'large'
              )))))
```

```
##
## Call:
## lm(formula = I(MSE > MSE_ranger) ~ size + size:(hasCategoricalFeat +
```

```
##     number.of.features + signal_strength + feat_obs_ratio), data = benchmark_aggr %>%
##     filter(estimator == "hte_adaptive_wmsp") %>% mutate(hasCategoricalFeat = number.of.features !=
##     number.of.numeric.features, signal_strength = 1 - MSE_randomForest/target_var,
##     feat_obs_ratio = n_features/n_obs, size = ifelse(number.of.instances <=
##         209, "tiny", ifelse(number.of.instances < 500, "small",
##         "large"))))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98662 -0.25471  0.02492  0.27201  0.91011
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       1.525e-01  1.128e-01   1.352   0.1772
## sizesmall                         2.106e-01  1.641e-01   1.283   0.2003
## sizetiny                          2.077e-01  1.256e-01   1.653   0.0991
## sizelarge:hasCategoricalFeatTRUE -3.303e-03  1.321e-01  -0.025   0.9801
## sizesmall:hasCategoricalFeatTRUE -2.579e-01  1.109e-01  -2.325   0.0206
## sizetiny:hasCategoricalFeatTRUE  -2.898e-02  5.955e-02  -0.487   0.6268
## sizelarge:number.of.features      1.468e-03  4.193e-03   0.350   0.7265
## sizesmall:number.of.features     -3.833e-03  1.185e-02  -0.323   0.7465
## sizetiny:number.of.features      -2.610e-05  6.364e-05  -0.410   0.6820
## sizelarge:signal_strength         1.028e+00  1.553e-01   6.623 1.19e-10
## sizesmall:signal_strength         7.718e-01  1.658e-01   4.655 4.48e-06
## sizetiny:signal_strength          8.473e-01  7.845e-02  10.801  < 2e-16
## sizelarge:feat_obs_ratio          3.638e-01  2.715e+00   0.134   0.8935
## sizesmall:feat_obs_ratio          1.170e+00  3.241e+00   0.361   0.7183
## sizetiny:feat_obs_ratio           5.551e-03  2.459e-03   2.257   0.0246
##
## (Intercept)
## sizesmall
## sizetiny                          .
## sizelarge:hasCategoricalFeatTRUE
## sizesmall:hasCategoricalFeatTRUE *
## sizetiny:hasCategoricalFeatTRUE
## sizelarge:number.of.features
## sizesmall:number.of.features
## sizetiny:number.of.features
## sizelarge:signal_strength        ***
## sizesmall:signal_strength        ***
## sizetiny:signal_strength         ***
## sizelarge:feat_obs_ratio
## sizesmall:feat_obs_ratio
## sizetiny:feat_obs_ratio          *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.374 on 384 degrees of freedom
##   (823 observations deleted due to missingness)
## Multiple R-squared:  0.3686, Adjusted R-squared:  0.3456
## F-statistic: 16.01 on 14 and 384 DF,  p-value: < 2.2e-16
```

```
# summary(
#     lm(
```

```
#       I(MSE > MSE_ranger) ~ I(number.of.features == number.of.numeric.features) +
#         number.of.features +
#         I(MSE_randomForest / target_var),
#       data = benchmark_aggr %>% filter(estimator == 'hte_honest_wmsp')
#     )
#   )

table(benchmark_aggr$MSE[benchmark_aggr$estimator == 'hte_adaptive_wmsp'] >
      benchmark_aggr$MSE_ranger[benchmark_aggr$estimator == 'hte_adaptive_wmsp']) /
  sum(benchmark_aggr$estimator == 'hte_adaptive_wmsp')
```

```
##
##    FALSE     TRUE
## 0.289689 0.710311
```

```
table(benchmark_aggr$MSE[benchmark_aggr$estimator == 'hte_adaptive_wmsp'] >
      benchmark_aggr$MSE_randomForest[benchmark_aggr$estimator == 'hte_adaptive_wmsp']) /
  sum(benchmark_aggr$estimator == 'hte_adaptive_wmsp')
```

```
##
##     FALSE      TRUE
## 0.1096563 0.2168576
```

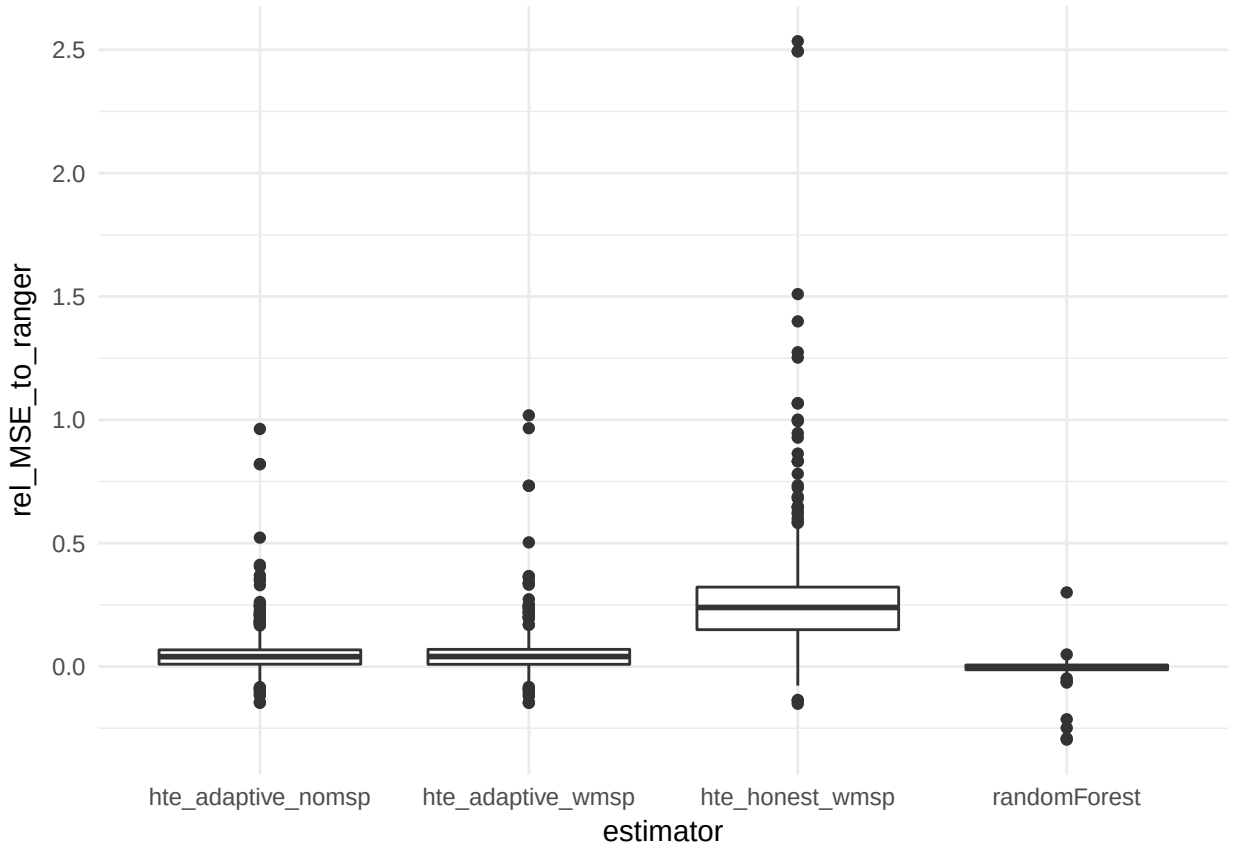
Conclusion for our adaptive RF:

- For the data sets which ran so far, we suffered only marginally, but other impelementations did outperform us in 30% of all cases. The difference for the MSE however, was rather marginal.
- It seems to be the case that we are not loosing anything in the way we handle categorical features. In fact, it seems to be the case that in data sets which have categorical features, we are doing slightly better.
- We are also doing better in settings with a weak signal
- We somehow loose a lot for small data sets, while we are doing pretty good on big data sets

Analyze examples which are particularily bad

```
benchmark_aggr %>% filter(estimator != "ranger") %>%
  filter(n_features < 3 * n_obs) %>% # only look at those with a lot of observations
  filter(n_obs > 300) %>%
  mutate(rel_MSE_to_ranger = (MSE - MSE_ranger) / MSE_ranger) -> benchmark_ranger_vs_hte

benchmark_ranger_vs_hte %>%
  ggplot(aes(x = estimator, y = rel_MSE_to_ranger)) +
  geom_boxplot() +
  theme_minimal()
```

```
## Warning: Removed 379 rows containing non-finite values (stat_boxplot).
```



Analyzing the worst task 2287:

```
benchmark_ranger_vs_hte %>% filter(estimator =='hte_adaptive_nomsp',
                                   rel_MSE_to_ranger > .5)
# Let's analyze the worst case:
data.id <- 196
target.feature <- "class"

# ------------------------------------------------------------------------------
data_set <-  getOMLDataSet(data.id = data.id)
non_missing_rows <- apply(!is.na(data_set$data),1, all) # only take rows which
# which don't have missing values

features <- data_set$data[non_missing_rows, colnames(data_set$data) != data_set$target.features]
target <- data_set$data[non_missing_rows,  colnames(data_set$data) == data_set$target.features]


# 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model', 'origin'
features <- features[, c( 'weight', 'acceleration', 'model')]
# features$model <- as.numeric(features$model)

features$model <- factor(sample(1:100)[as.numeric(features$model)])
# split the data into training and test set
n_smp <- length(target)
# set.seed(7684240)
idx_1 <- sample(1:n_smp, round(n_smp / 2))
```

```
idx_2 <- (1:n_smp)[-idx_1]
features_1 <- features[idx_1, ]
features_2 <- features[idx_2, ]
y_1 <- target[idx_1]
y_2  <- target[idx_2]

library(ranger)
library(hte)
ntree = 500
mtry = function(features) max(round(ncol(features) / 3), 1)
nodesize = 5
replace = TRUE
sampsize = function(target) length(target)

ranger_e <- ranger(
              y ~ .,
              data = data.frame(features_1, y = y_1),
              num.trees = ntree,
              mtry = mtry(features_1),
              min.node.size = nodesize,
              replace = replace,
              sample.fraction = 1
              )
(MSE_1 <- mean((y_2 - predict(ranger_e, features_2)$predictions) ^ 2))

hrf_e <- honestRF(
            x = features_1,
            y = y_1,
            ntree = ntree,
            replace = replace,
            sampsize = sampsize(target),
            mtry = mtry(features_1),
            nodesizeSpl = nodesize,
            nodesizeAvg = nodesize,
            splitratio = 1,
            middleSplit = FALSE
          )
(MSE_1 <- mean((y_2 - predict(hrf_e, features_2)) ^ 2))


# ------------------------------------------------------------------------------
```

For task 2287, I discovered a very strange behavior:

- the feature model is a factor containingnumbrs 70, 71, ... 80. In the standard case ranger is doing much better than we do on average 9.0 and we have 14.0
- If I transform the factor to a numeric (now it has ordering), then we and ranger do simlarily well and we both roughly get an MSE of 9.0
- If I make the factor random letters i.e. 70 becomse 'i', 71 becomes 'f', then both methods do similarily poorly and achieve roughly
- If I randomly reorder the model numbers, but keep them as a factor, i.e. 70 -> 76, 71 -> 73, ..., then both models seem to achieve roughly around 14.0

Conclusion:

I think in this data set ranger is automatically converting a factor of numerics to a numeric. It is thereby introducing an ordering which can be useful and exploited. However, that type of ordering could also be wrong information and lead to poorer performance. However, given that in most data sets numerics seem to have an ordering, this can be a smar behavior.

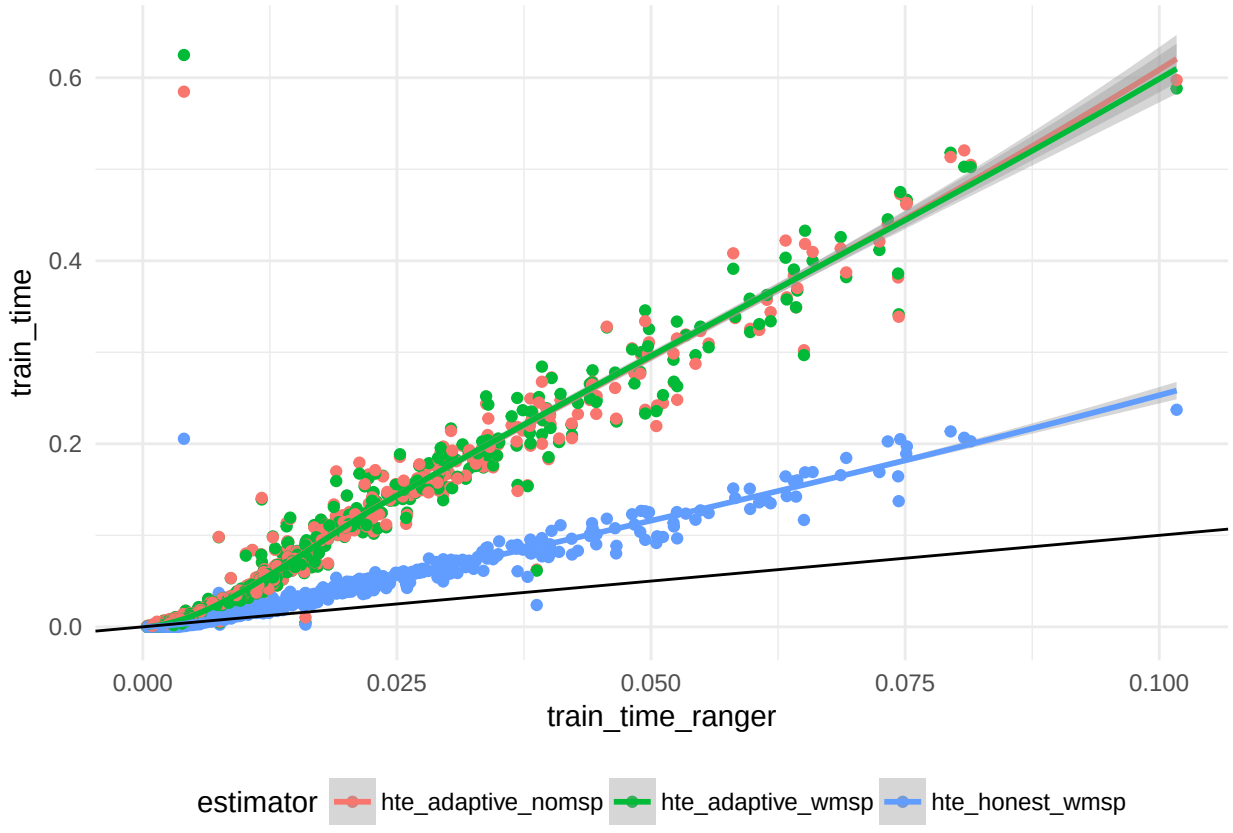
Analyzing the worst task 4840:

```
benchmark_ranger_vs_hte %>% filter(estimator =='hte_adaptive_nomsp',
                                   rel_MSE_to_ranger > .4)
# Let's analyze the worst case:
data.id <- 196
target.feature <- "velocity"

# ------------------------------------------------------------------------------
data_set <-  getOMLDataSet(data.id = data.id)
non_missing_rows <- apply(!is.na(data_set$data),1, all) # only take rows which
# which don't have missing values

features <- data_set$data[non_missing_rows, colnames(data_set$data) != data_set$target.features]
target <- data_set$data[non_missing_rows,  colnames(data_set$data) == data_set$target.features]

# features$group <- factor(sample(c(letters, LETTERS))[as.numeric(features$group)])

features <- features%>%select( - angle)
YaleToolkit::whatis(data_set$data)

# split the data into training and test set
n_smp <- length(target)
# set.seed(7684240)
idx_1 <- sample(1:n_smp, round(n_smp / 2))
idx_2 <- (1:n_smp)[-idx_1]
features_1 <- features[idx_1, ]
features_2 <- features[idx_2, ]
y_1 <- target[idx_1]
y_2  <- target[idx_2]

library(ranger)
library(hte)
ntree = 500
mtry = function(features) max(round(ncol(features) / 3), 1)
nodesize = 5
replace = TRUE
sampsize = function(target) length(target)

ranger_e <- ranger(
              y ~ .,
              data = data.frame(features_1, y = y_1),
              num.trees = ntree,
              mtry = mtry(features_1),
              min.node.size = nodesize,
              replace = replace,
              sample.fraction = 1
              )
(MSE_1 <- mean((y_2 - predict(ranger_e, features_2)$predictions) ^ 2))
```

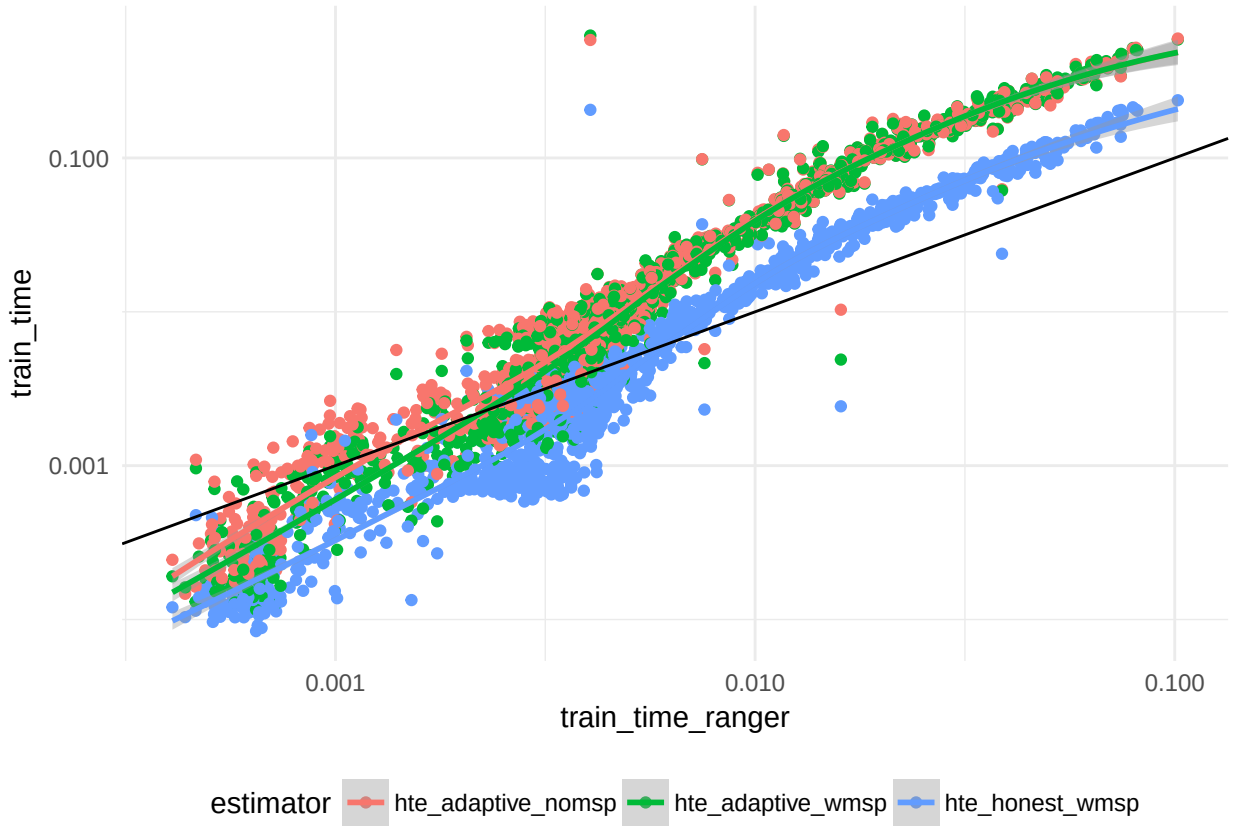
```
hrf_e <- honestRF(
            x = features_1,
            y = y_1,
            ntree = ntree,
            replace = replace,
            sampsize = sampsize(target),
            mtry = mtry(features_1),
            nodesizeSpl = nodesize,
            nodesizeAvg = nodesize,
            splitratio = 1,
            middleSplit = FALSE
          )
(MSE_1 <- mean((y_2 - predict(hrf_e, features_2)) ^ 2))


# ------------------------------------------------------------------------------
```

Speed comparison:

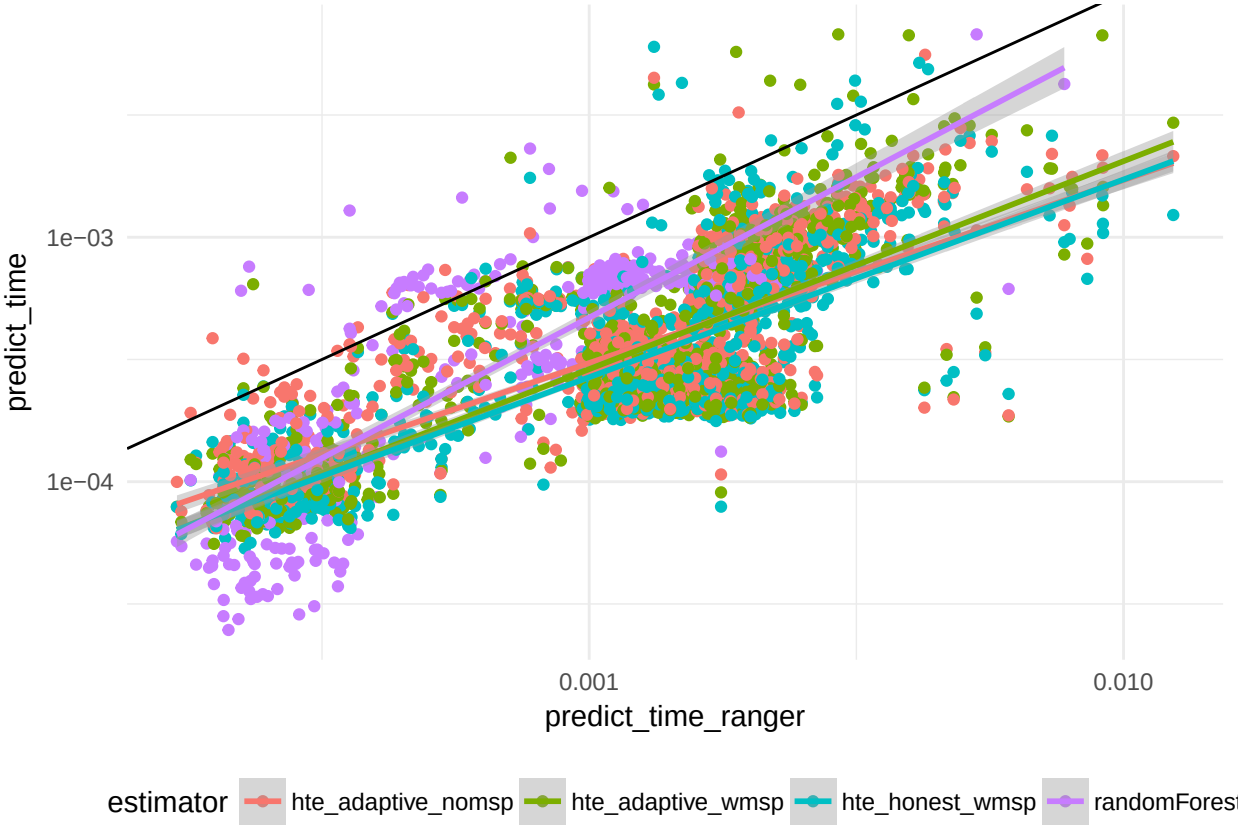
Training time:



Training time without randomForest



Prediciton time:



Conclusion:

- randomForest is much slower than ranger and our implementation
- It is not clear whether we are slower or faster than ranger.

Final Conclusion:

1. It seems to be worth while studying how to automatically exploit missing data. Most data sets here have missing data
2. It would be nice if our version of rf never fails.
3. We want to get rid of the warning: “used as adaptive random forest”
4. We seem to be loosing a lot on data sets where some categories are only presented in the test data set.