STAT 685: Dr. Suojin Wang's Group

Modeling Seoul Bike Sharing Demand

Nam Tran, Bai Zou

9/15/2020

Summary

On a high level, we're working to predict the amount of rental bikes, in Seoul, Korea, needed for the next hour given several covariates. A use case is by knowing ho wmuch bike demand there will be, the company providing the rental bikes can allocate or deallocate as necessary, both improving customer satisfaction, e.g., not needing to wait for bikes, as well as lowering costs by properly anticipating demand.

We were tasked with building a predictor, with our supervisor (y) being count data and our features (X) being eight quantitative features and four qualitative feature.

We partitioned our data (`dat`) into a learn (`xl`) and test (`xt`) portion, using 11 months of consecutive data as the learn portion and using the following, remaining one month as the test portion.

After exploratory data anlysis, imputation of missing data, and normalization of the features, we tried a multitude of linear and non-linear estimators. We also created a equal-weighted average ensemble of the estimators we built, for which the ensemble performed best with respect to test error.

We submitted our predictions using the ensemble estimator.

Features and Supervisor Observations

Our observations:

- Features
  - We have eight quantatitive features and four qualitative features.
  - Presence of large pairwise multicollinearity amongst the quantatitive features, i.e., largest off-diagonal element from correlation matrix is around 0.91.
  - Performed PCA on quantatitive features and noted that approximately 68% of the variance was concentrated in the first three principal components, out of eight.
  - From first two PC's, there were clusters of potential outliers but may just have been a function of the non-normality of the covariates.
  - Quantative features are, for the most part, distributed normally, per Shapiro-Wilk tests.
- Supervisor
  - From a QQ-plot relative to a normal distribution, it suggests the supervisor is not normally distributed.

Missing Data

Out observations and actions:

- Missing Data
  - There was no data missing (observations or features) and as such, no imputation was needed.

Data Processing

Our actions:

- We left the supervisor alone, i.e., "raw."
- We centered and scaled the quantitative features.
- We created a total of 36 indicators ($levels - 1$) for the qualitative features and didnt't center nor scale them.

Building the Estimators

We tried linear and non-linear estimators. We specified an appropriate `tuneGrid` for each instance and for cases where we saw that our optimum (`bestTune`) was at the edge, we would modify our `tuneGrid` appropriately.

We used 10-Fold CV to generate a risk estimate (the average of the 10 left-out folds' ***testError***) for each permutation of hyperparameters, and chose the set of hyperparameters with the minimal 10-Fold CV risk estimate.

Test error was the average squared error loss, i.e., $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

The following are estimators considered and any "interesting" notes in parenthesis.

Estimators Considered:

- Linear Estimators
  - *Linear Regression*
  - *Elastic Net* ($\hat{\alpha} = 0.05$ {did not "like" L1 reguarlization}; chose all 36 features)
  - *Refitted Lasso* (effectively linear regression, since a linear regression model was estimated using all 36 features)
- Non-Linear Estimators
  - *MARS* (`degree` $= 3$; `nprune` $= 50$)
  - *Simple Decision Tree*
  - *Random Forest*
  - *Boosting* ($\hat{\eta} = 0.01$; `colsample_bytree` $= 0.5$, `nrounds` $= 750$, `max_depth` $= 8$)

Even though some of the the pairwise correlations of the estimators were high, we still considered an estimator that's a simple average (ensemble) of all the aforementioned estimators, called *Ensemble*, and another variant called *Ensemble2*.

Evaluation of Estimators

Based on a risk estimate, i.e., test error, w.r.t. squared error loss, i.e., $\frac{1}{n}\sum_{i=1}^{n}(y_i\hat{y}_i)^2$, we evaluated the estimators on our test data, i.e., `xt`. Further, although, mathematically related, we also evaluated the estimators on an out-of-sample R^2, i.e., R^2_{OOS}.

Further, we evaluated the estimators w.r.t. the notion of *lift curve.*

More details in the appendix, but on a high-level:

- Risk Estimate
  - *Boosting* performed the best, followed by *Rf*, i.e., Random Forest.
  - All the non-linear methods out-performed the linear methods (*Linear Regression*, *ElasticNet*, *Refitted Lasso*).
- Lift Curve
  - All estimators had sane lift.

Appendix

Packages and Helper Function

```
require(tidyverse)
require(dplyr)
require(caret)
require(glmnet)
require(doParallel)
require(earth)
require(vip)
require(ranger)
require(xgboost)
require(RANN)
require(lubridate)
require(flipTime)
require(e1071)

theme_update(plot.title = element_text(hjust = 0.5))
```

Loading the Data

```
set.seed(1)

# Loading the Data
datPath = paste0(dirname(getwd()), "/data/SeoulBikeData.csv")
colNames = c("Date", "RentedBikeCount", "Hour", "Temp", "Humidity",
             "WindSpeed", "Visibility", "DewPointTemp", "SolarRadiation",
             "Rainfall", "Snowfall", "Seasons", "Holiday", "FunctionalDay")
dat = read_csv(datPath, col_names = colNames, skip=1)

# Setting up Factors
dat$Seasons = as_factor(dat$Seasons)
dat$Holiday = as_factor(dat$Holiday)
dat$FunctionalDay = as_factor(dat$FunctionalDay)

# Creating DateTime that incorporates both Date and Hours and dropping Hour
dat$Date = AsDateTime(dat$Date) + hours(dat$Hour)

# We consider our "train" dataset all that we have, i.e., the total, for which we'll split later.
Xtot = dat %>% select(-RentedBikeCount, -Date)
Ytot = dat %>% pull(RentedBikeCount)
```

Exploratory Data Analysis

```
anchorDate = "2018/11/01"

learnNdx = dat$Date < anchorDate

yl = Ytot[learnNdx]
yt = Ytot[-learnNdx]
xl = Xtot[learnNdx,]
xt = Xtot[-learnNdx, ]
```

Identifying Qualitative Features

```
featureLevels = sapply(xl, function(x) { length(unique(x)) })
head(sort(featureLevels), 5)
```

```
##       Holiday FunctionalDay       Seasons          Hour      Snowfall
##             2             2             4            24            45
```

In this case, it seems that `Holiday`, `FunctionalDay`, `Seasons`, and potentially `Hour` are qualitative features and should be treated as such.

We say potentially `Hour`, since if we enter `Hour` as a quantitative feature, then we're entering `Hour` as a linear effect, which may not be the case.

Considering Missing Data

```
featureNA   = head(sort(sapply(xl, function(x) { sum(is.na(x) )}), decreasing = TRUE))
obsNA       = head(sort(apply(xl, 1, function(x) { sum(is.na(x) )}), decreasing = TRUE))
naSummaryDf = data.frame(featureNA=featureNA, obsNA=obsNA, row.names = 1:length(obsNA))
naSummaryDf
```

```
##   featureNA obsNA
## 1         0     0
## 2         0     0
## 3         0     0
## 4         0     0
## 5         0     0
## 6         0     0
```

From this, it doesn't seem we have any "degenerate" columns that warrant feature deletion and additionally we don't have any specific observations where a lot of the `NA`'s exist. In short, imputtation isn't needed here.

Building Post-Imputation-Normalization X

```
# ---------------------------------------
# Identifying qualitative features
# ---------------------------------------
qualFeatures = which(featureLevels < 30)

xlQuant = xl[,-qualFeatures]
xlQual  = xl[, qualFeatures, drop = FALSE] %>% sapply(as.factor)
xtQuant = xt[,-qualFeatures]
xtQual  = xt[, qualFeatures, drop = FALSE] %>% sapply(as.factor)

# ---------------------------------------
# Center/Scale of Quantitative Features
# ---------------------------------------
ppStandardization = preProcess(xlQuant, method=c("center", "scale"))
xlQuantPost = predict(ppStandardization, xlQuant)
xtQuantPost = predict(ppStandardization, xtQuant)

# ---------------------------------------
# Creating Dummary Variables
```

```
# ----------------------------------------
ppDummy = dummyVars(~ ., data = xlQual, fullRank = TRUE)

xlQualPost = predict(ppDummy, xlQual)
xtQualPost = predict(ppDummy, xtQual)

# ----------------------------------------
# Getting a singular matrix now
# ----------------------------------------
xlFull = cbind(xlQuantPost, xlQualPost)
xtFull = cbind(xtQuantPost, xtQualPost)

xlFullMat = as.matrix(xlFull)
xtFullMat = as.matrix(xtFull)
```

Correlation Amongst the Features

Let's look at the correlation amongst the features themselves.

```
corMat = cor(xlFull)
absCorMat = abs(corMat)[upper.tri(corMat)]

round(quantile(absCorMat, c(0.5, 0.75, 0.9, 0.95, 0.99, 1)), 3)
```

```
##   50%   75%   90%   95%   99%  100%
## 0.043 0.044 0.110 0.161 0.515 0.918
```

The features aren't highly correlated w.r.t. each other (this does suggest we have a bunch of weak estimators that might lend themselves to boosting).

Let's look at the correlation amongst the supervisor and the quantitative features

```
absCorSup = sapply(xlFull, function(x) { abs(cor(yl, x)) })
round(quantile(sapply(xlFull, function(x) { abs(cor(yl, x)) }), c(.5, .75, .9, .95, .99, 1)), 3)
```

```
##   50%   75%   90%   95%   99%  100%
## 0.122 0.182 0.285 0.410 0.515 0.551
```

The fact we see a large correlation around 41% at 95^{th} and then a "drop-off" of correlation around 29% at the 90^{th} suggests that we have a potentially dominating feature that's an outlier in correlation with the rest of the features, which might be a problem in say, bagging, for which we constantly choose that feature and induce correlation amongst our trees.

Potential ways to overcome this include, but aren't limited to,

- Random Forest which is similar to bagging but for each split, we only consider a random subset of the features, allowing for a tree to not include the dominating feature and thus be less correlated.
- Boosting, where we can both control the learning rate (η), thus we don't take a full step in the negative of the gradient, as well as the number of features we consider per-tree/per-split as well.

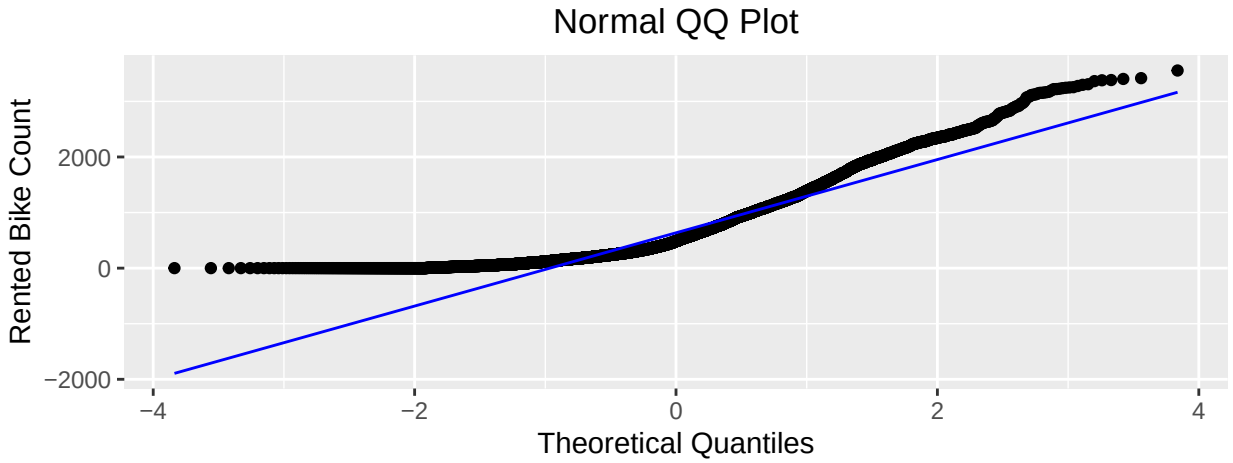
Normality (or lack of) of the Features

TBD

Normality (or lack thereof) of the Supervisor

```
ggplot() +
  aes(sample = yl) +
```

```
  geom_qq(distribution = qnorm) +
  geom_qq_line(line.p = c(0.25, 0.75), col = "blue") +
  labs(y="Rented Bike Count", x="Theoretical Quantiles",
       title="Normal QQ Plot")
```



From this, it's abundantly clear that the supervisor isn't normal. This is useful, since the typical squared error loss used in standard linear regression, $\ell(y, \hat{y}) = (y - \hat{y})^2$ is only optimal for normal data. While noting prevents us from using such squared error loss, we might be leaving things on the table, and as such, we might consider a different loss and objective function to optimize against.

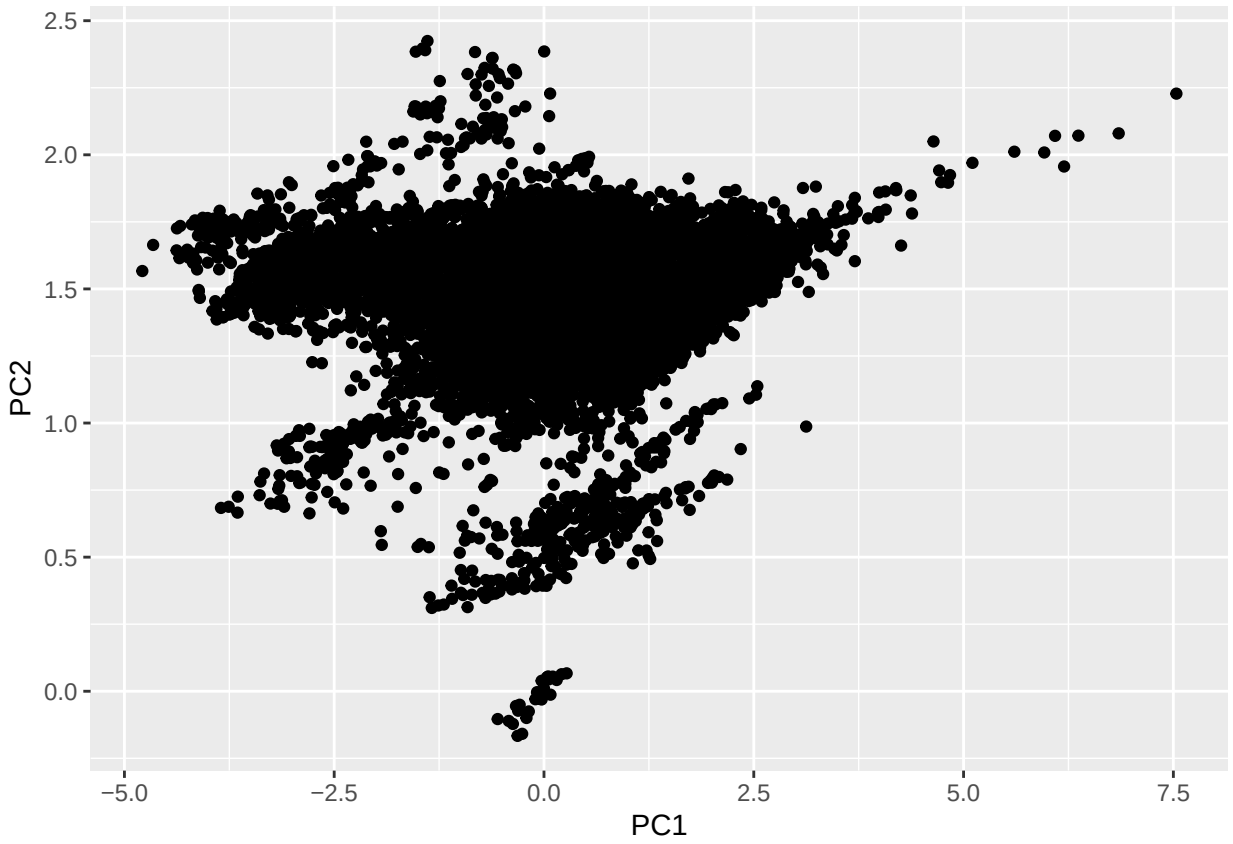
Further, this isn't surprising since the rented bike data count is "count" data which is typically modeled by a Poisson distribution.

In practice, we'll attempt to use the squared error loss and when possible, we'll see if we get better results using a more appropriate objective function.

Identifying Outliers

Here, we can look at the first two principle components on a scatter plot and see if there are any pernicious outliers.

```
pcaOut = prcomp(xlFull, center=FALSE, scale=FALSE)

pcaOut$x[,1:2] %>% as.data.frame %>% ggplot() +
  geom_point(aes(x=PC1, y=PC2))
```



Ideally we'd see an ellipsoid, but here we see some "clusters" of points. The concern might be the data point at approximately $(7.5, 2.25)$, but it's not that concerning given there's a trail leading up to that point. Note that PCA, underneath the hood, implicitly assumes normality of our data. And, in so much it's not normal, we might get plots that are non-sensical.

That being said, we'll keep an eye out for the "trail points" as we proceed along with our research.

Cross-Validation + Parallelism

We're intending to use 6 cores (please don't melt down computer) as well as using a 10-fold CV as our risk-estimate.

```
cl = makeCluster(6)
registerDoParallel(cl)

trControl = trainControl(method = "cv", number = 10)
```

Linear Estimators

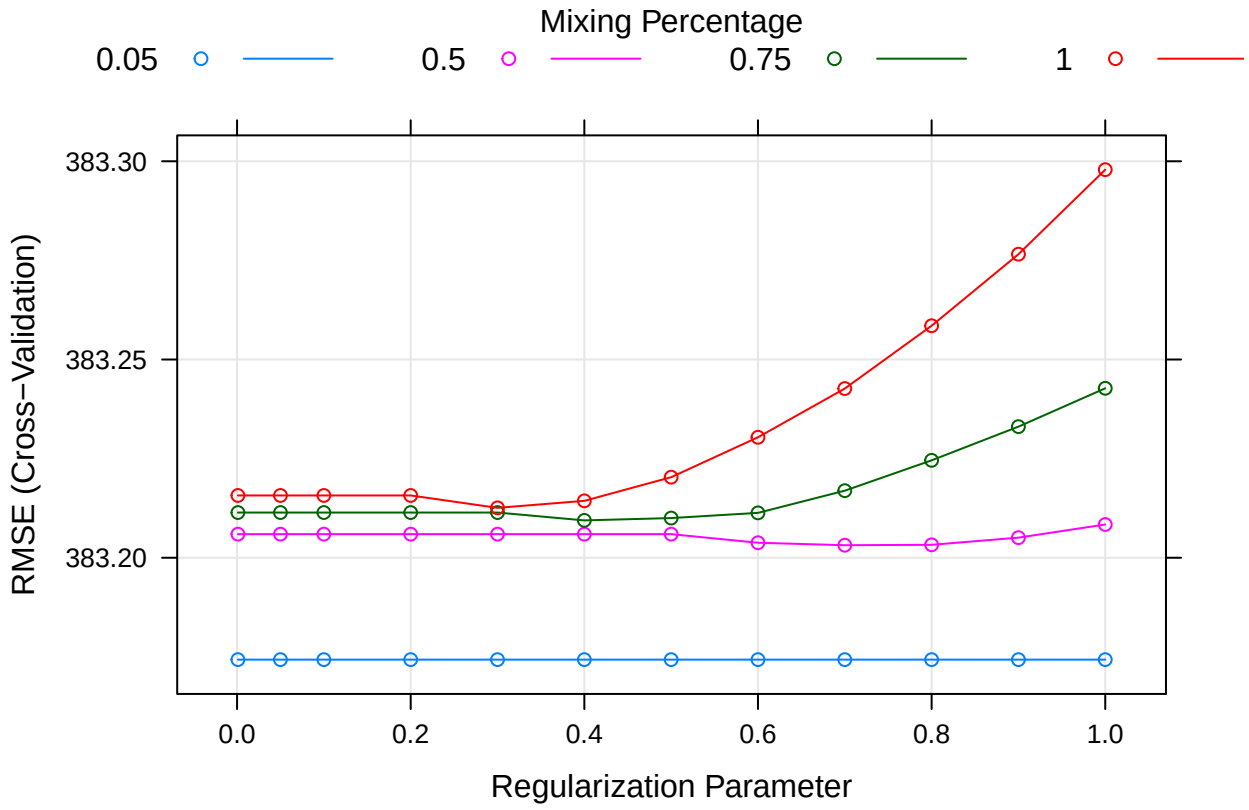
Linear Estimators We're Considering:

- Linear Regression
- Elastic Net
- Refitted Lasso

```
set.seed(1)
linearOut = train(x = xlFullMat, y = yl,
                  method = "lm",
                  trControl = trControl)
```

```
yhatLinear = predict(linearOut$finalModel, xlFull)
```

```
set.seed(1)
lassoGrid = expand.grid(
  lambda = c(0.001, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0),
  alpha = c(0.05, 0.5, 0.75, 1))
elasticOut = train(x = xlFullMat, y = yl,
                   method="glmnet",
                   tuneGrid = lassoGrid,
                   trControl = trControl)
plot(elasticOut)
```




```
# Refitting with 'core' glmnet
glmnetOut     = glmnet(x = xlFullMat, y = yl, alpha = elasticOut$bestTune$alpha)
betaHatGlmnet = coef(glmnetOut, s = elasticOut$bestTune$lambda)
yhatGlmnet    = predict(glmnetOut, xlFullMat, s=elasticOut$bestTune$lambda)
```

```
set.seed(1)

# Fitting with glmnet in general with alpha = 1.0
glmnetOut     = cv.glmnet(x = xlFullMat, y = yl, alpha = 1.0)

# We use lambda.min since lambda.1se will result in an empty active set!
betaHatGlmnet = coef(glmnetOut, s = "lambda.min")
refittedSIdx = which(as.vector(abs(betaHatGlmnet) > 1e-6)[-1])

# Fitting a linear model
refittedLassoOut = train(x = xlFullMat[,refittedSIdx], y = yl,
```

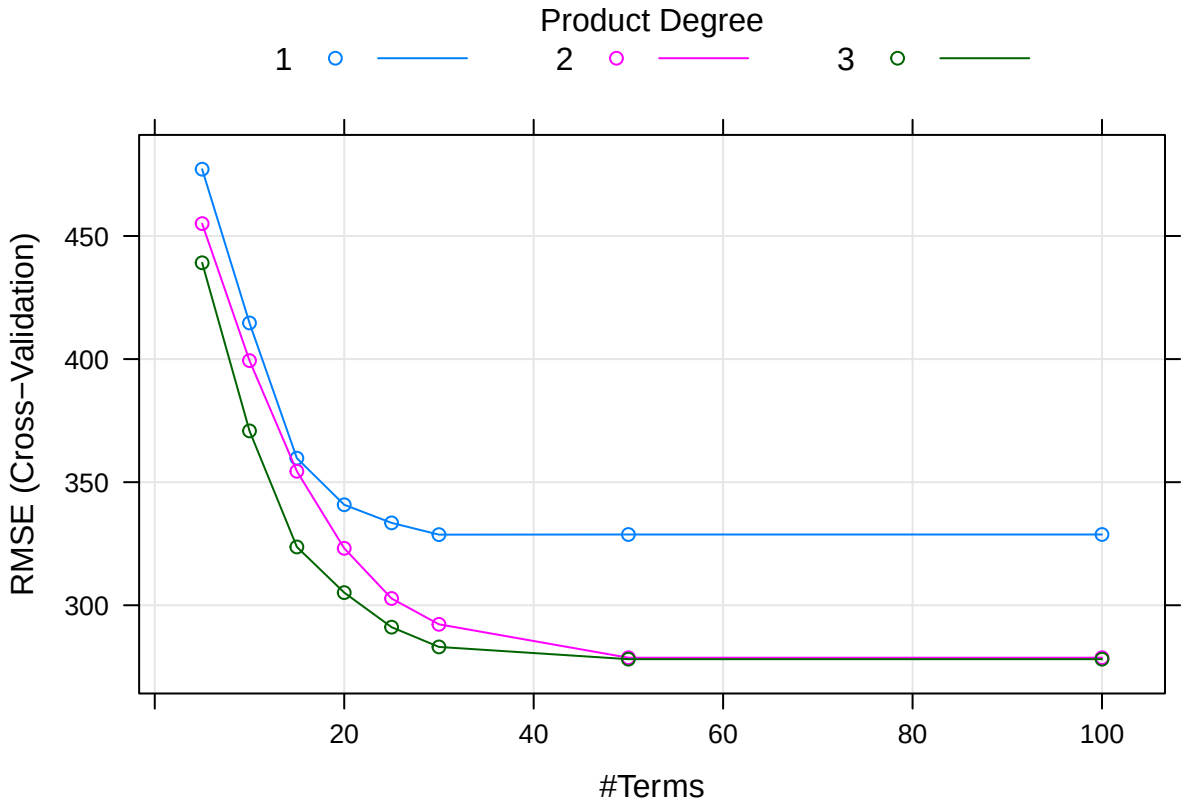
```
                          method = "lm",
                          trControl = trControl)
yhatRefitted = predict(refittedLassoOut, xlFullMat[,refittedSIdx])
```

Non-Linear Estimators

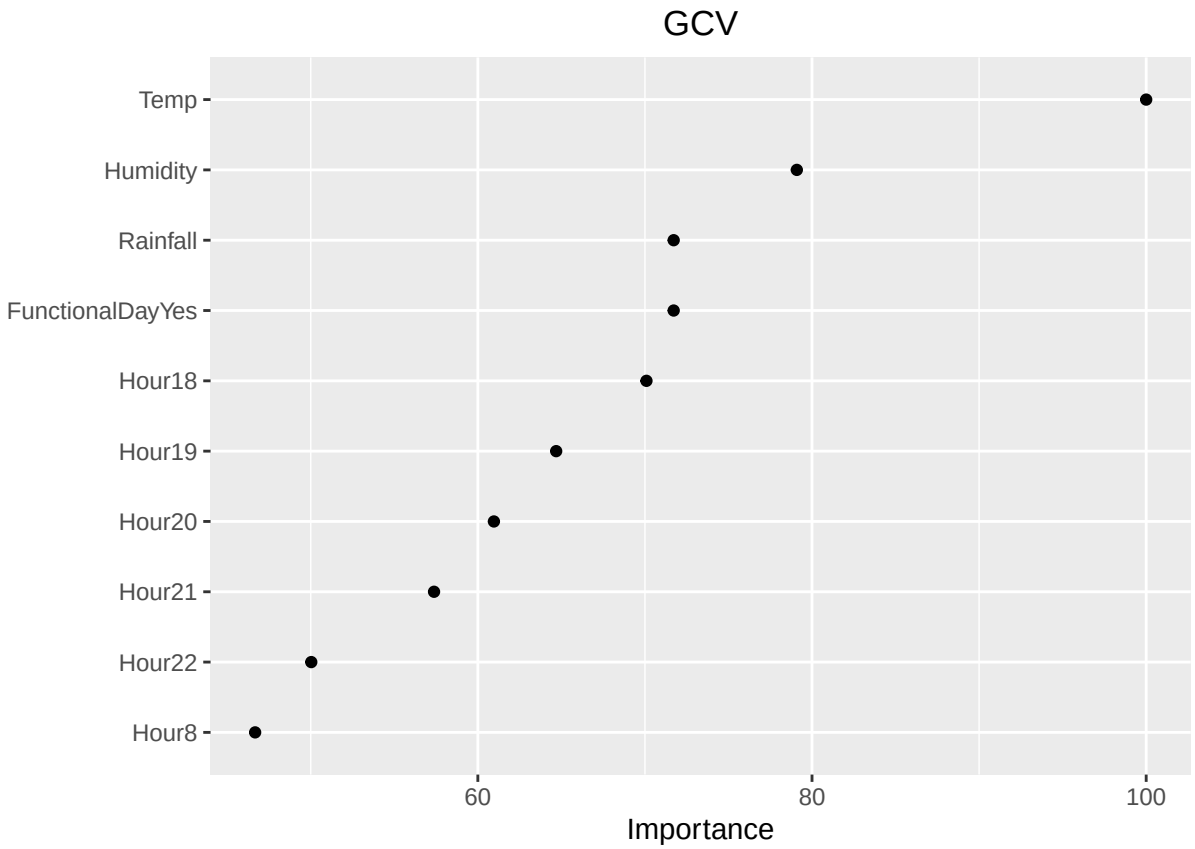
Non-Linear Estimators We're Considering:

- MARS
- Simple Decision Tree
- Random Forest
- Boosting

```
set.seed(1)
tuneGrid = expand.grid(
  degree=1:3,                              # number of interactions
  nprune = c(5,10,15,20,25,30, 50, 100)) # number of knots

marsOut = train(x = xlFullMat, y = yl,
                method="earth",
                tuneGrid = tuneGrid,
                trControl = trControl)
plot(marsOut)
```

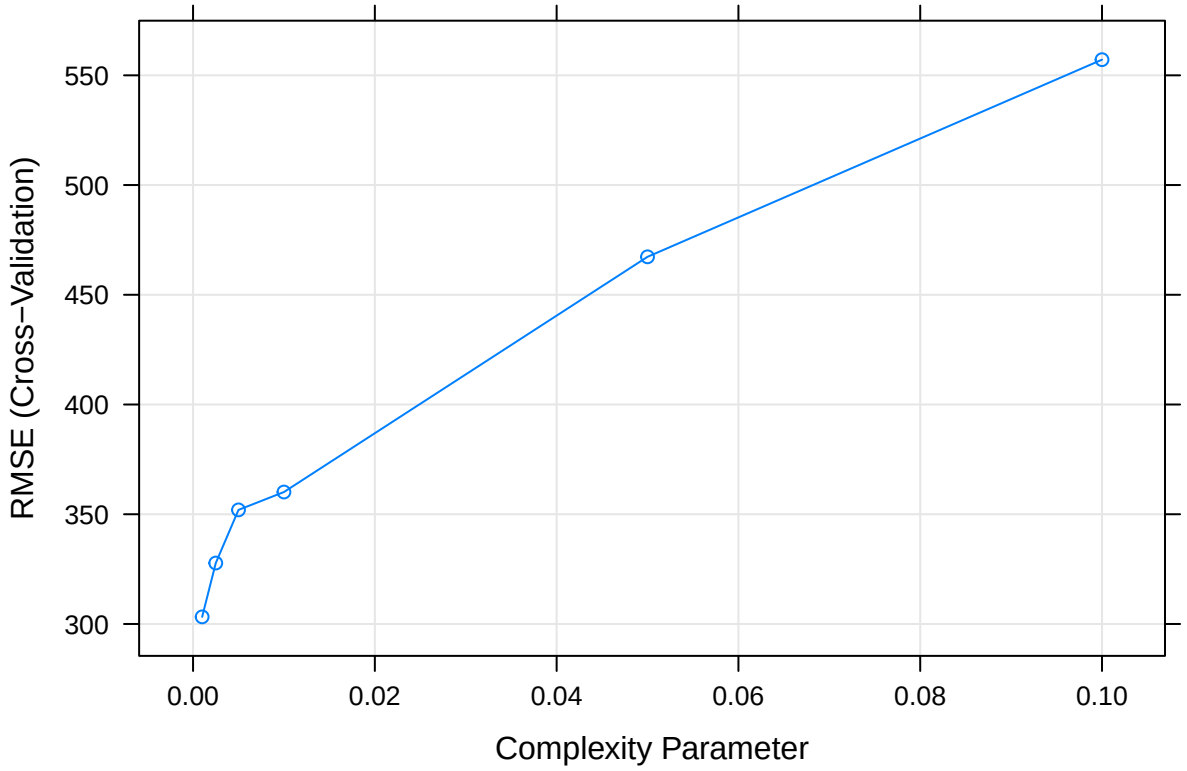



```
vip(marsOut, num_features = 10, geom="point", value="gcv") + ggtitle("GCV")
```




```
yhatMARS = predict(marsOut$finalModel, xlFullMat)
```

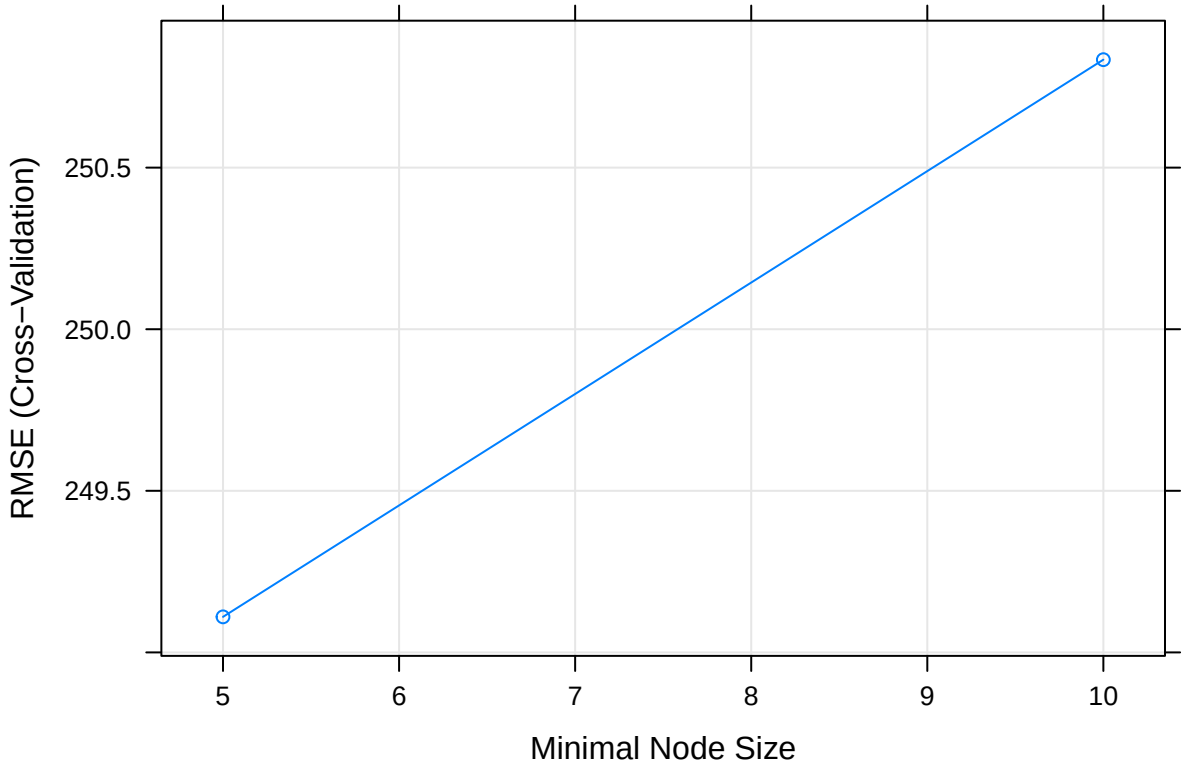
```
set.seed(1)
tuneGrid = data.frame(cp=c(0.001, 0.0025, 0.005, 0.01, 0.05, 0.1))
rpartOut = train(x = xlFull, y = yl,
                 method = "rpart",
                 tuneGrid = tuneGrid,
                 trControl = trControl)
plot(rpartOut)
```




```
yhatDT = predict(rpartOut$finalModel, xlFull)
```

As we can see here, the "best" model for the decission tree is to simply predict the mean of our target, in this case, $\hat{y} = \bar{y} \approx 0.597$.

```
set.seed(1)
tuneGrid = data.frame(splitrule="variance",
                      mtry = round(sqrt(ncol(xlFull))),
                      min.node.size = c(5, 10))
rangerOut = train(x = xlFullMat, y = yl,
                  method = "ranger",
                  tuneGrid = tuneGrid,
                  num.trees = 500,
                  trControl = trControl)
plot(rangerOut)
```



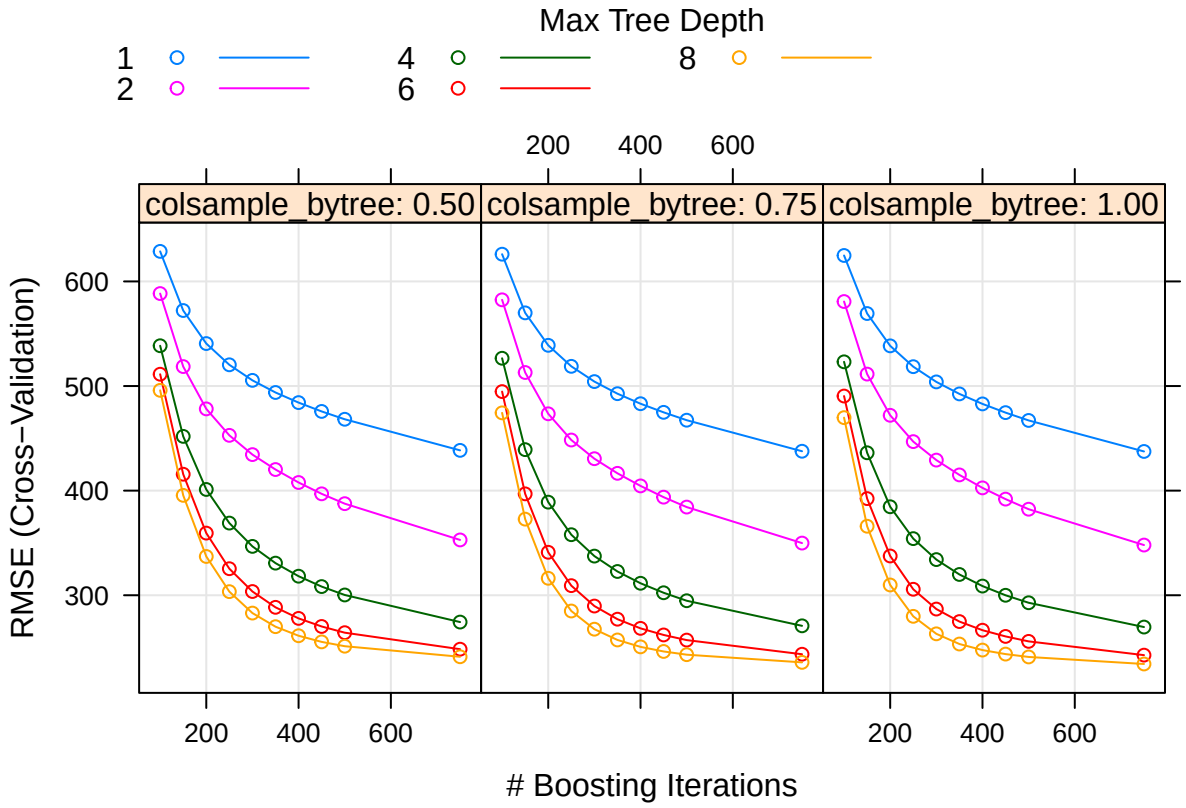

```
yhatRf = predict(rangerOut, xlFull)
```

Not the most ideal residual plot but may be symptomatic of it being non-linear and being in-sample.

```
set.seed(1)
tuneGrid = expand.grid("nrounds" = c(100, 150, 200, 250, 300, 350, 400, 450, 500, 750),
                       "max_depth" = c(1, 2, 4, 6, 8),
                       "eta" = c(0.01),
                       "gamma" = 0,
                       "colsample_bytree" = c(.5, .75, 1),
                       "min_child_weight" = 0,
                       "subsample" = 0.5)

boostOutObjSe = train(x=xlFullMat, y = yl,
                      method="xgbTree",
                      objective="reg:squarederror",
                      tuneGrid = tuneGrid,
                      trControl = trControl)
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##  objective
##   Only the last value for each of them will be used.
```

```
plot(boostOutObjSe)
```




```
yhatBoost = predict(boostOutObjSe, xlFullMat)
```

Similar comments as we mentioned for the random forest residual plot.

Judging Performance of Estimators on "Test" Data Set

Looking at Test Error of the Estimators

Now, we can look at `xt` which is our nominal test data set (for which we have a `yt`).

Some things to keep in mind:

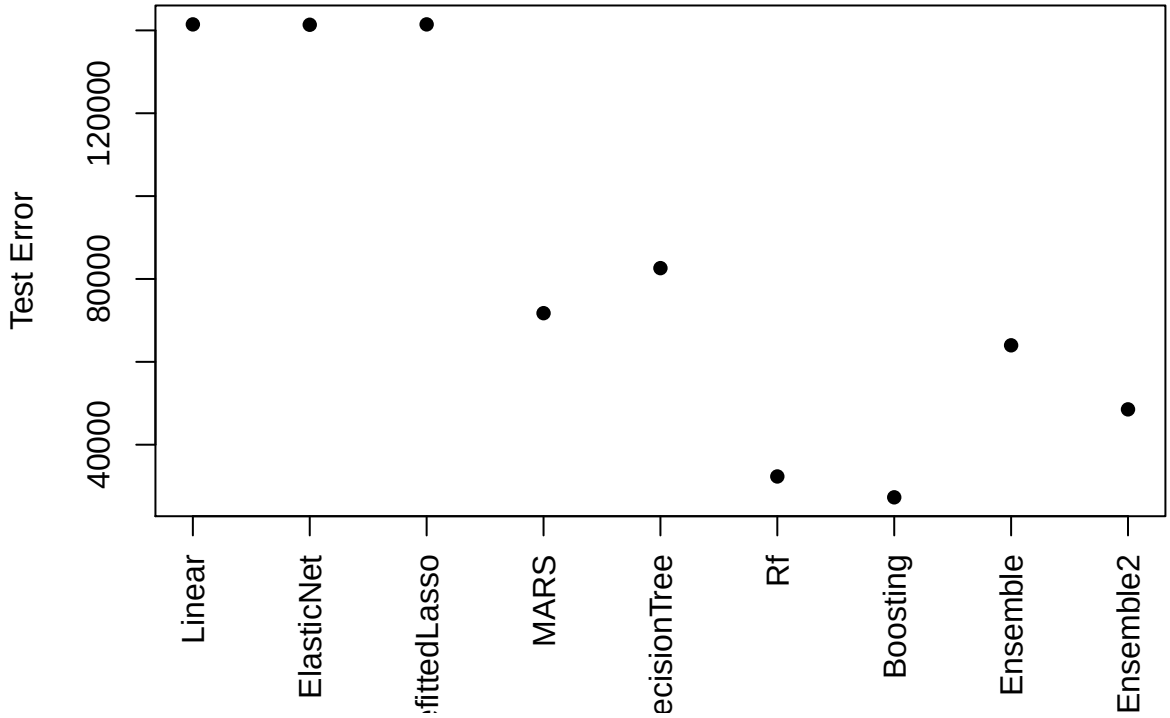
- I'm going to create a simple ensemble that does an equal weight averaging subsets of the $\hat{y}_{i,est}$'s.

```
# getting yhats for all the estimators
makeYhats = function(datDf) {
  resultsTest = data.frame(
    yhatLinear        = predict(linearOut, datDf),
    yhatElasticNet    = predict(elasticOut, datDf),
    yhatRefittedLasso = predict(refittedLassoOut, datDf[, refittedSIdx]),
    yhatMARS          = predict(marsOut, datDf)[,1],
    yhatDecisionTree  = predict(rpartOut, datDf),
    yhatRf            = predict(rangerOut, datDf),
    yhatBoosting      = predict(boostOutObjSe, datDf)
  )

  resultsTest$yhatEnsemble  = apply(resultsTest, 1, mean)
  resultsTest$yhatEnsemble2 = apply(resultsTest %>% select(yhatRefittedLasso:yhatBoosting), 1, mean)
  resultsTest
}
```

```
resultsTest = makeYhats(xtFull)
```

One way to judge the various estimators is to simply look at the test error for the various estimators.

```
testErrors = sapply(resultsTest, function(yhatEst) {
  mean((yhatEst - yt)^2)
})

plot(1:length(testErrors), testErrors, xlab='',xaxt='n', pch = 16, ylab='Test Error')
axis(1,labels = str_replace(names(testErrors), "yhat", ""), at =1:length(testErrors), las = 3)
```



We can observe:

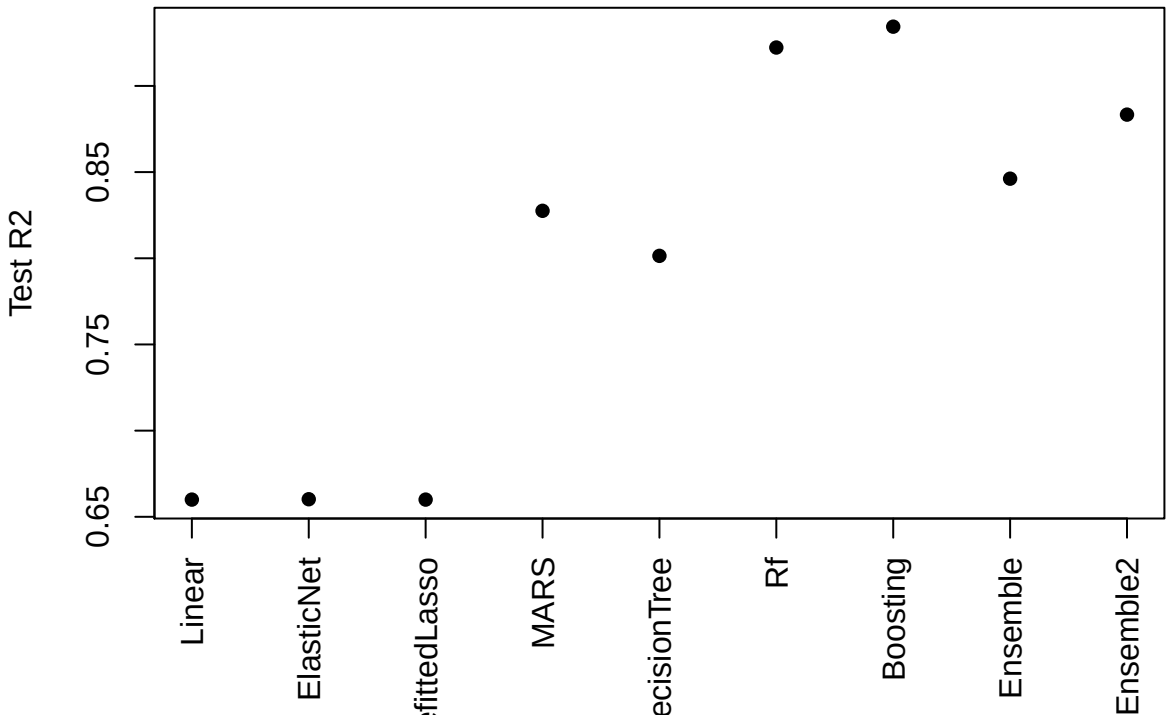
- The linear methodologies (`Linear`, `ElasticNet`, and `refittedLasso`) aren't as performant, in general, i.e., there are non-linearities that it can't pick up.
- `Boosting` and `Rf` are very similar.
- The "best" single estimator is `Boosting`.

Looking at Test R^2 of the Estimators

We can use the test data and construct an out-of-sample R^2, i.e.,

$$R^2_{OOS} = 1 - \frac{\sum_{i=1}^{n}(y - \hat{y})^2}{\sum_{i=1}^{n}(y - \bar{y})^2}$$

```
testR2 = sapply(resultsTest, function(yhatEst) {
  1 - sum((yhatEst - yt)^2) / sum((yt - mean(yt))^2)
})

plot(1:length(testR2), testR2, xlab='',xaxt='n', pch = 16, ylab='Test R2')
axis(1,labels = str_replace(names(testR2), "yhat", ""), at =1:length(testR2), las = 3)
```

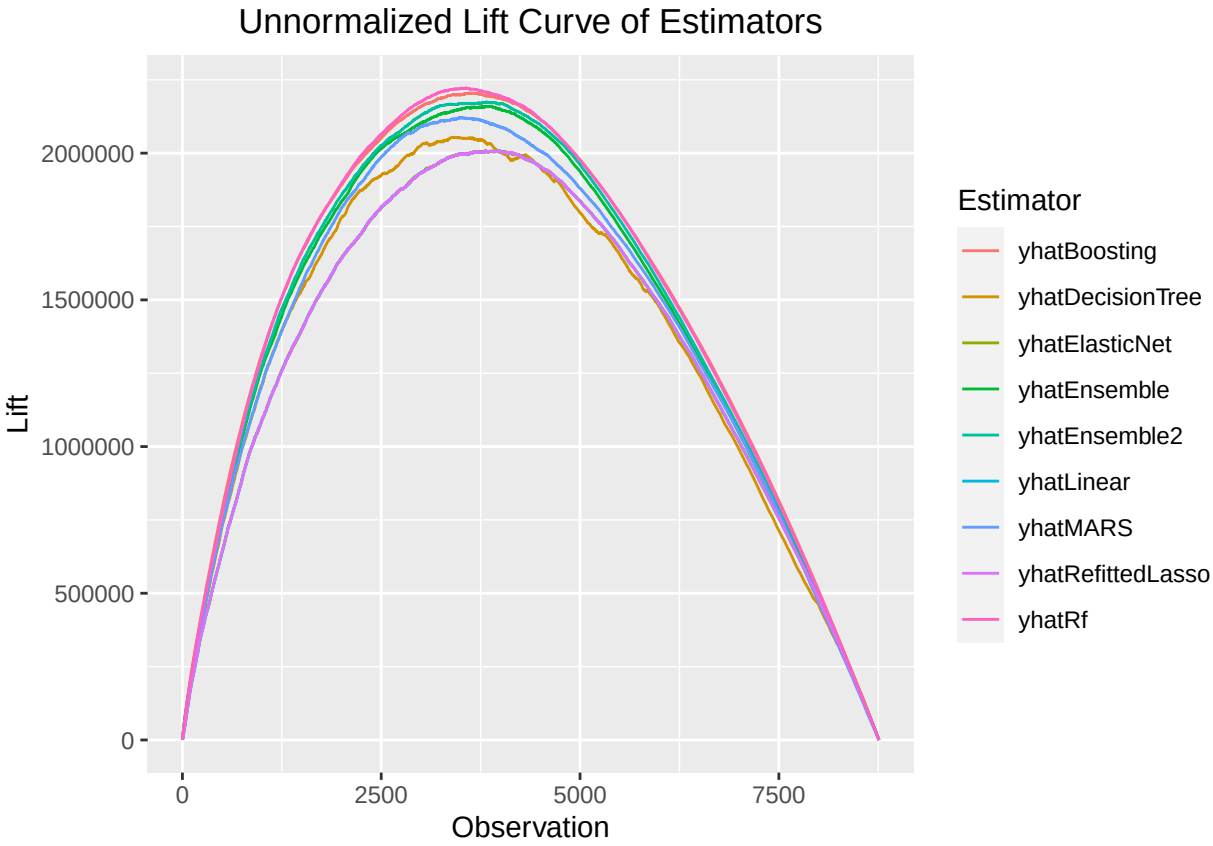


Looking at Lift Curves of Estimators

While `test error` and `test R2` is a great metric, it loses some nuance (e.g., does a particular predictor do better in the middle of the distribution of y as compared to the tail of y). Doing lift curves can help regain some of that information. The general idea is that given $(y_i, \hat{y}_i)$, we sort based on $\hat{y}_i$ in descending order (largest to smallest) and then we take the cumulative sum of the now sorted y_i's. The "ideal" lift curve would be a parabolic shape.

This also might prove helpful in that given the squared error loss, we primarily care in getting the "edges" right.

```
# generating unnormalized lift curves
stackedLcs = do.call("rbind", lapply(names(resultsTest), function(est) {
  yhat      = resultsTest[,est]
  yhatOrder = order(yhat*-1)
  yLift     = cumsum(yt[yhatOrder] - mean(yt))

  data.frame(x=1:length(yhat), y=yLift, Estimator=est)
}))

stackedLcs %>% ggplot(aes(color=Estimator)) +
  geom_line(aes(x=x, y=y)) +
  labs( title= "Unnormalized Lift Curve of Estimators", y="Lift", x = "Observation") +
  theme(plot.title = element_text(hjust = 0.5))
```



After all is said and done, the simple average ensemble seems reasonable.

Further Sanity Check of Ensemble

We can look at the correlation of the estimators with each other. If there's low correlation amongst the different estimators, then that can aid the ensemble in having enough diversity (very hand-wavy statement).

```
cor((resultsTest %>% select(-c(yhatDecisionTree, yhatEnsemble)))) %>% round(3)
```

```
##                   yhatLinear yhatElasticNet yhatRefittedLasso yhatMARS yhatRf
## yhatLinear             1.000          1.000             1.000    0.875  0.868
## yhatElasticNet         1.000          1.000             1.000    0.875  0.868
## yhatRefittedLasso      1.000          1.000             1.000    0.875  0.868
## yhatMARS               0.875          0.875             0.875    1.000  0.963
## yhatRf                 0.868          0.868             0.868    0.963  1.000
## yhatBoosting           0.854          0.855             0.854    0.961  0.993
## yhatEnsemble2          0.908          0.908             0.908    0.980  0.990
##                   yhatBoosting yhatEnsemble2
## yhatLinear               0.854         0.908
## yhatElasticNet           0.855         0.908
## yhatRefittedLasso        0.854         0.908
## yhatMARS                 0.961         0.980
## yhatRf                   0.993         0.990
## yhatBoosting             1.000         0.987
## yhatEnsemble2            0.987         1.000
```

From this, there's at times large correlation amongst the different estimators. As such, there's not a lot of diversity and consequently we shouldn't expect an ensemble predictor to do well here.