Predicting Transparent Conductors

Samuel A, Mikolaj W

April 3, 2020

# 

Table of Contents

[Background: 3](#_Toc37275732)

[Data 4](#_Toc37275733)

[Preparation Work and Packages 5](#_Toc37275734)

[Libraries 5](#_Toc37275735)

[Data Preparation 5](#_Toc37275736)

[Exploring structure within predictors with PCA 26](#_Toc37275737)

[Correlation 28](#_Toc37275738)

[Principal Component Analysis 29](#_Toc37275739)

[Model Building 34](#_Toc37275740)

[MARS 34](#_Toc37275741)

[Model 1: formation energy 34](#_Toc37275742)

[Model 2: bandgap energy 46](#_Toc37275743)

[Final Metrics 55](#_Toc37275744)

[Treed Regression 55](#_Toc37275745)

[Model 1: formation energy 55](#_Toc37275746)

[Final Metrics 61](#_Toc37275747)

[Model 2: bandgap energy 62](#_Toc37275748)

[Final Metrics 67](#_Toc37275749)

[XGboost 67](#_Toc37275750)

[Model 1: formation energy 67](#_Toc37275751)

[Final Metrics 68](#_Toc37275752)

[Model 2: bandgap energy 70](#_Toc37275753)

[Final Metrics 71](#_Toc37275754)

[Shrinkage methods: Ridge, Lasso, Elastic Net 72](#_Toc37275755)

[Model 1: formation energy 73](#_Toc37275756)

[Elastic Net 74](#_Toc37275757)

[Ridge 75](#_Toc37275758)

[Lasso 75](#_Toc37275759)

[Final Metrics 75](#_Toc37275760)

[Model 2: bandgap energy 76](#_Toc37275761)

[Elastic Net 76](#_Toc37275762)

[Ridge 77](#_Toc37275763)

[Lasso 78](#_Toc37275764)

[Final Metrics 78](#_Toc37275765)

[Random Forest 79](#_Toc37275766)

[Model 1: formation energy 80](#_Toc37275767)

[Final Metrics 83](#_Toc37275768)

[Model 2: bandgap energy 84](#_Toc37275769)

[Final Metrics 87](#_Toc37275770)

[Neural Networks 88](#_Toc37275771)

[Model 1: formation energy 90](#_Toc37275772)

[Final Metrics 91](#_Toc37275773)

[Model 2: bandgap energy 91](#_Toc37275774)

[Final Metrics 93](#_Toc37275775)

[Keras Regressor: neural network 93](#_Toc37275776)

[Final Models: 95](#_Toc37275777)

[Predicting Formation of Energy: Random Forest 96](#_Toc37275778)

[Final Results 98](#_Toc37275779)

[Variable Importance 99](#_Toc37275780)

[Predicting Bandgap Energy: XGboost 101](#_Toc37275781)

[Final Results 101](#_Toc37275782)

[Variable Importance 102](#_Toc37275783)

# Background:

Innovative materials design is needed to tackle some of the most important health, environmental, energy, social, and economic challenges of this century. In particular, improving the properties of materials that are intrinsically connected to the generation and utilization of energy is crucial if we are to mitigate environmental damage due to a growing global demand. [Transparent conductors](https://en.wikipedia.org/wiki/Transparent_conducting_film) are an important class of compounds that are both electrically conductive and have a low absorption in the visible range, which are typically competing properties. A combination of both of these characteristics is key for the operation of a variety of technological devices such as photovoltaic cells, light-emitting diodes for flat-panel displays, transistors, sensors, touch screens, and lasers. However, only a small number of compounds are currently known to display both transparency and conductivity suitable enough to be used as transparent conducting materials.

[Aluminum](https://en.wikipedia.org/wiki/Aluminium_oxide) (Al), [gallium](https://en.wikipedia.org/wiki/Gallium(III)_oxide) (Ga), [indium](https://en.wikipedia.org/wiki/Indium(III)_oxide) (In) sesquioxides are some of the most promising transparent conductors because of a combination of both large [bandgap](https://en.wikipedia.org/wiki/Band_gap#Photovoltaic_cells) energies, which leads to optical transparency over the visible range, and high [conductivities](https://en.wikipedia.org/wiki/Electrical_resistivity_and_conductivity). These materials are also chemically stable and relatively inexpensive to produce. Alloying of these binary compounds in ternary or quaternary mixtures could enable the design of a new material at a specific composition with improved properties over what is current possible. These alloys are described by the formula where , , and can vary but are limited by the constraint . The total number of atoms in the [unit cell](https://en.wikipedia.org/wiki/Crystal_structure),  (where is an integer), is typically between 5 and 100. However, the main limitation in the design of compounds is that identification and discovery of novel materials for targeted applications requires an examination of enormous compositional and configurational degrees of freedom (i.e., many combinations of , , and ). To avoid costly and inefficient trial-and-error of synthetic routes, computational data-driven methods can be used to guide the discovery of potentially more efficient materials to aid in the development of advanced (or totally new) technologies. In computational material science, the standard tool for computing these properties is the quantum-mechanical method known as density-functional theory (DFT). However, DFT calculations are expensive, requiring hundreds or thousands of CPU hours on supercomputers for large systems, which prohibits the modeling of a sizable number of possible compositions and configurations. As a result, potential  materials remain relatively unexplored. Data-driven models offer an alternative approach to efficiently search for new possible compounds in targeted applications but at a significantly reduced computational cost.

This problem aims to accomplish this goal by having you or your team develop models for the prediction to two target properties/responses: the formation of energy (which is an indication of the stability of a new material) and the bandgap energy (which is an indication of the potential for transparency over the visible range) to facilitate the discovery of new transparent conductors and allow for advancements in the above-mentioned technologies.

# Data

There are two datafiles for this midterm project/exam:

**Conductors (train).csv** – 1,567 material formulations where both responses are known.

**Conductors (test).csv** – 833 material formulations where both responses are unknown.

You will obviously be building a model using the training data to predict both response values for the test cases. The predictive accuracy of your models will be judged using the following criterion, averaged over both responses:

This essentially the RMSEP where the responses have been natural log-transformed. The is being used to deal with the fact that both responses have values that are zero. The responses are NOT given to you in the log-scale so you will need to do that (but DO NOT FORGET TO ADD 1!!). As the metric being used to measure the accuracy of your predictions is already in the scale, you will not need to back-transform your final predictions or worry about back-transforming response values within any CV functions you use.

Build models using any of the methods we have examined for both responses and submit your predictions in scale.

# Preparation Work and Packages

## Libraries

tidyr, dplyr, ggplot2, nnet, car ISLR , glmnet, pls, plsr, corrplot, stringr, gridExtra, ipred, randomForest, GGally, maptree, xgboost, earth, keras, plotmo, factoextra, FactoMineR, rpart, Cubist

## Data Preparation

Format the response in the training set to ln(y + 1).

require(dplyr)  
cond\_train2 = cond\_train  
  
cond\_train %>%  
 mutate(formation\_energy\_ev\_natom\_log\_y\_plus\_1 = log(formation\_energy\_ev\_natom + 1),   
 bandgap\_energy\_ev\_log\_y\_plus\_1 = log(bandgap\_energy\_ev + 1)) -> cond\_train2  
  
drop.cols <- c('formation\_energy\_ev\_natom', 'bandgap\_energy\_ev')  
  
cond\_train2 %>%  
 select(-drop.cols) -> cond\_train2

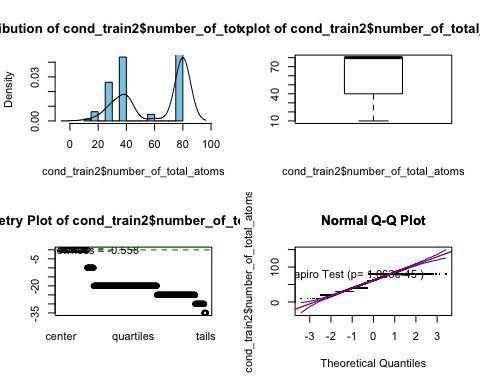
Starting out, we’ll try and see what transformations would be optimal for existing variables. There are only 11 in total by default, so going through every variable an individual basis is more than doable. We also will follow a doctrine of optimization being greater than explainability for this approach, as one of our chief goals is quality of predictions.

cond\_train2\_trans = cond\_train2  
  
#checks for optimal transformations  
myBC = function(y)  
{  
 BCtran(y)  
 results = powerTransform(y)  
 summary(results)  
}

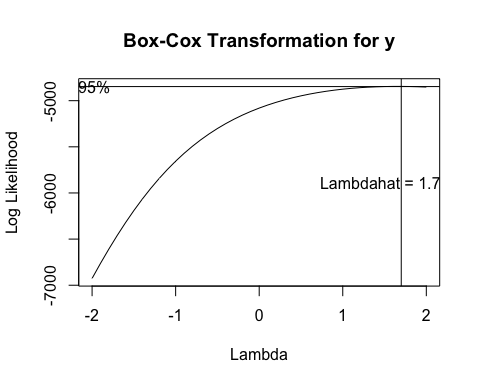
Space group itself is an interesting variable, as while it seems ordinal at first, space groups themselves are a bit complex and a higher group number is not a consitant difference. This will be accounted for later.

Starting with Total Atoms

Statplot(cond\_train2$number\_of\_total\_atoms)

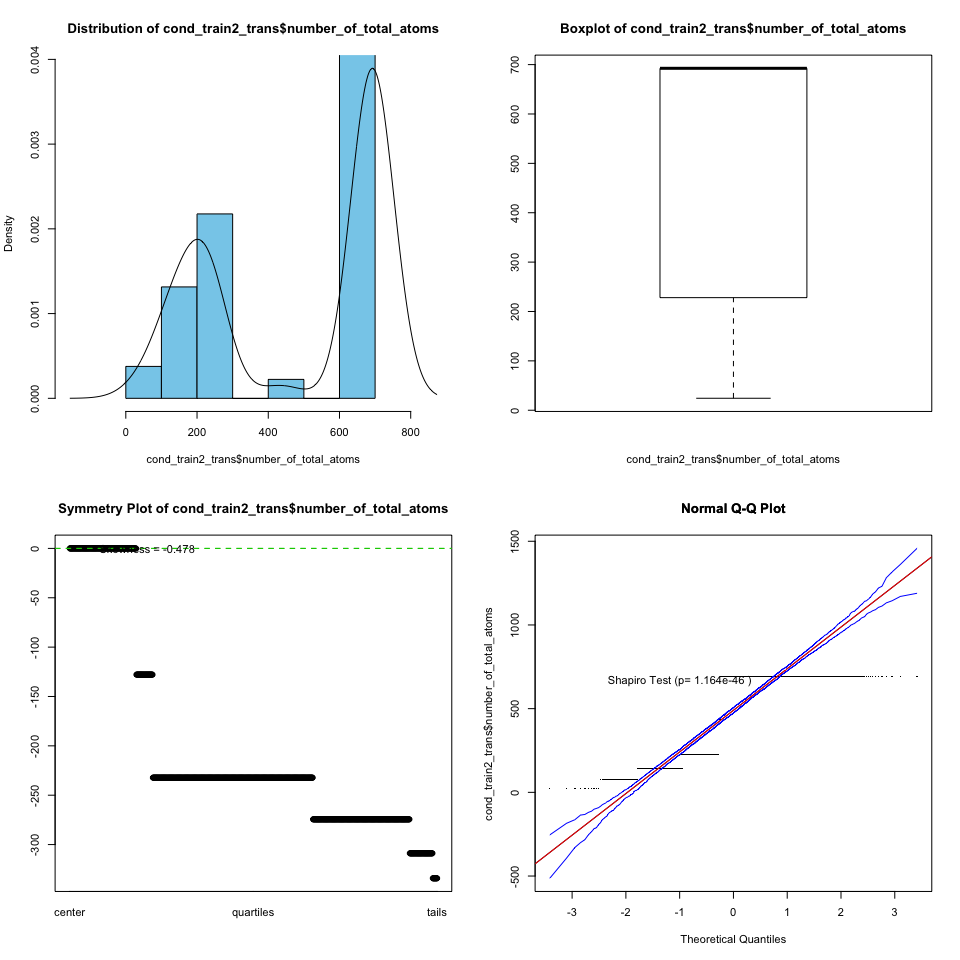


myBC(cond\_train2$number\_of\_total\_atoms)



## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y 1.6584 1.66 1.4833 1.8336  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 470.167 1 < 2.22e-16  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 60.21255 1 8.5487e-15

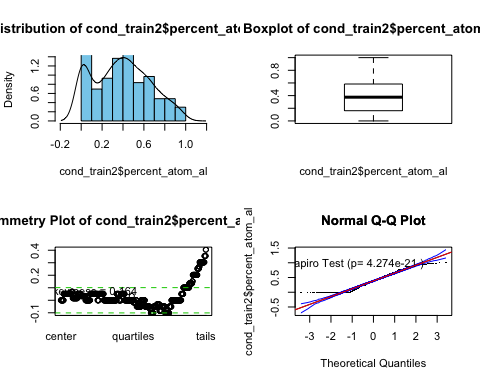
cond\_train2\_trans$number\_of\_total\_atoms = bcPower(cond\_train2$number\_of\_total\_atoms, 1.6)  
  
Statplot(cond\_train2\_trans$number\_of\_total\_atoms)



These next three variables measure the makeup of the elements Galluim, Aluminum, and Indium as percentages. All three are of course related in that regard, and certain makeups may include some or all of each of these. This may make it difficult to view each as an individual in terms of model utilization, but there are a few things we can try. First, we can simply transform each variable.

Percent\_atom\_al is next

Statplot(cond\_train2$percent\_atom\_al)

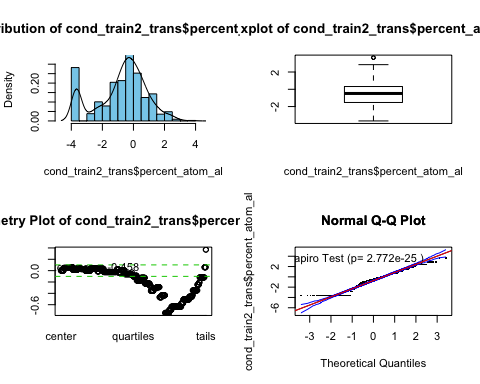


Since we are working with percentage data, we thought it would be best to use a logit transformation here.

cond\_train2\_trans$percent\_atom\_al = logit(cond\_train2$percent\_atom\_al)

## Warning in logit(cond\_train2$percent\_atom\_al): proportions remapped to  
## (0.025, 0.975)

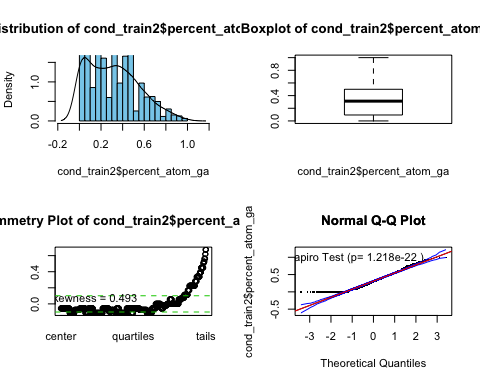
Statplot(cond\_train2\_trans$percent\_atom\_al)



The improvement is noticeable, although the zeros still tend to have a negative impact.

Next was percent\_atom\_ga

Statplot(cond\_train2$percent\_atom\_ga)

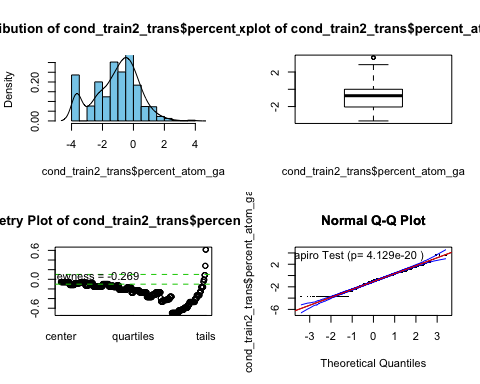


This variable similarly was best handled with logit.

cond\_train2\_trans$percent\_atom\_ga = logit(cond\_train2$percent\_atom\_ga)

## Warning in logit(cond\_train2$percent\_atom\_ga): proportions remapped to  
## (0.025, 0.975)

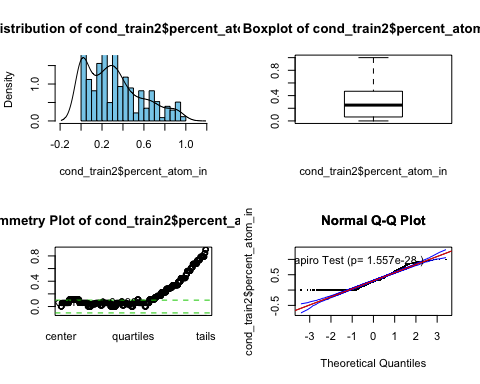
Statplot(cond\_train2\_trans$percent\_atom\_ga)



We still have a problem with zeros, although in every other regard we have improved.

percent\_atom\_in is the final variable in this series, and likewise was handled similarly.

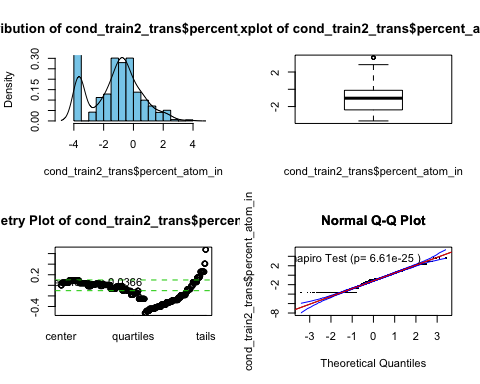
Statplot(cond\_train2$percent\_atom\_in)



cond\_train2\_trans$percent\_atom\_in = logit(cond\_train2$percent\_atom\_in)

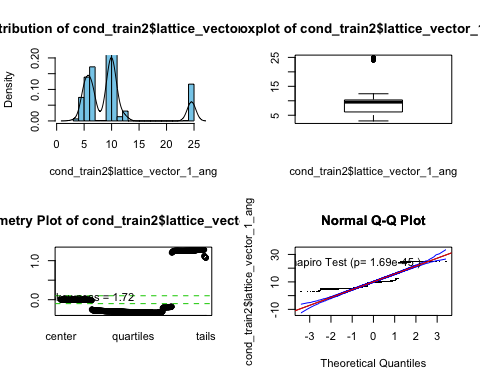
## Warning in logit(cond\_train2$percent\_atom\_in): proportions remapped to  
## (0.025, 0.975)

Statplot(cond\_train2\_trans$percent\_atom\_in)



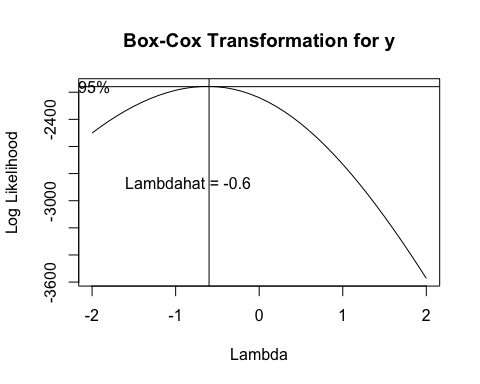
Next, we have a series of lattice vector variables that should be simple enough to box cox transform.

Statplot(cond\_train2$lattice\_vector\_1\_ang)



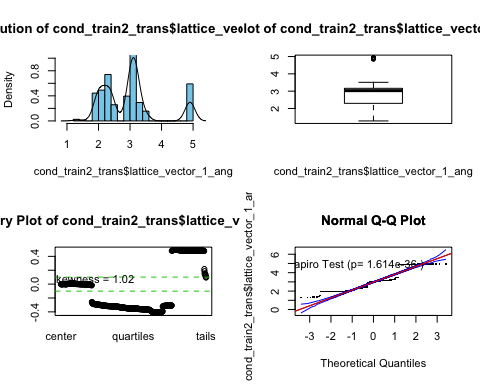
Our myBC function tends to indicate using a power of 0.6, however, better results through testing were found in the for of a 4th root transformation. While the result is still not ideal, the improvement is still massive and should not be discounted. This may prove to be the beginning of a pattern in terms of how the lattice vectors all hold.

myBC(cond\_train2$lattice\_vector\_1\_ang)



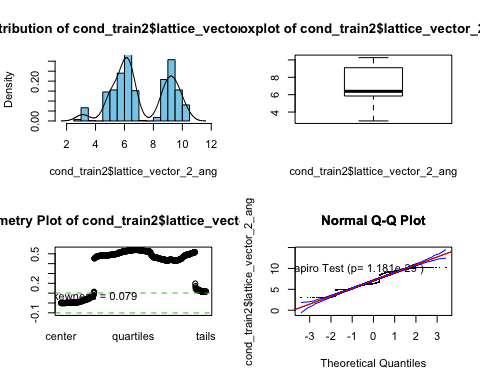
## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y -0.6207 -0.62 -0.7171 -0.5243  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 166.5686 1 < 2.22e-16  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 1148.252 1 < 2.22e-16

cond\_train2\_trans$lattice\_vector\_1\_ang = bcPower(cond\_train2$lattice\_vector\_1\_ang, 0.25)  
  
Statplot(cond\_train2\_trans$lattice\_vector\_1\_ang)



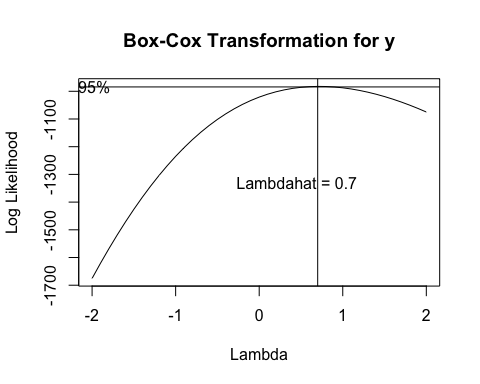
The next vector angle has similar shortcomings but uses a different solution.

Statplot(cond\_train2$lattice\_vector\_2\_ang)



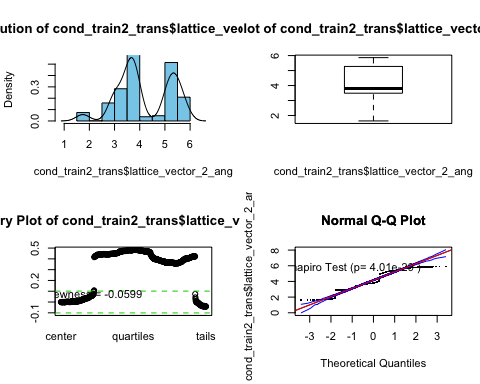
This time, the bcPower function was more on target, with the power transformation of 0.7 being the best. Unfortunately, the best in this case is not a very desirable variable structure, but once again still is an improvement.

myBC(cond\_train2$lattice\_vector\_2\_ang)

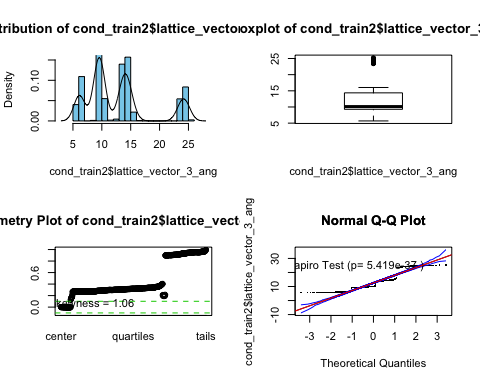


## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y 0.7188 0.72 0.5499 0.8877  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 77.27689 1 < 2.22e-16  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 10.21984 1 0.0013894

cond\_train2\_trans$lattice\_vector\_2\_ang = bcPower(cond\_train2$lattice\_vector\_2\_ang, 0.7)  
  
Statplot(cond\_train2\_trans$lattice\_vector\_2\_ang)

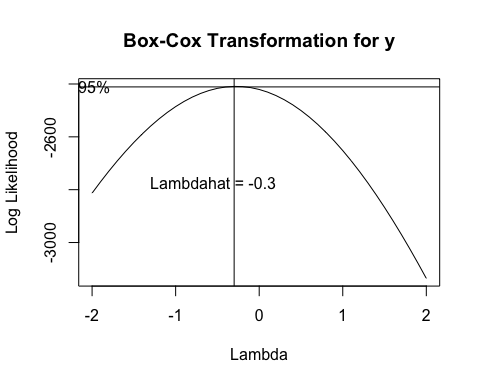
 For the last vector angle, we have a similar story.

Statplot(cond\_train2$lattice\_vector\_3\_ang)



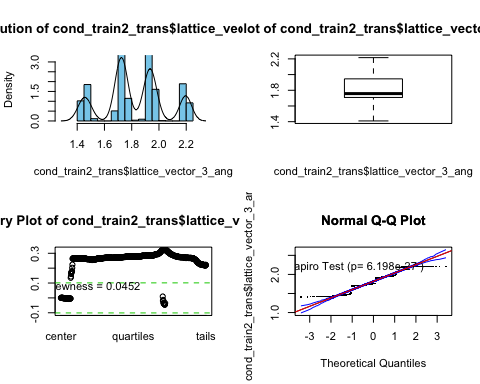
The box cox gave the best result again with a negative 5th root, although much like the former our results still have shortcomings despite the imrpovements.

myBC(cond\_train2$lattice\_vector\_3\_ang)



## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y -0.2777 -0.33 -0.3904 -0.1649  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 23.37957 1 1.3298e-06  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 485.8501 1 < 2.22e-16

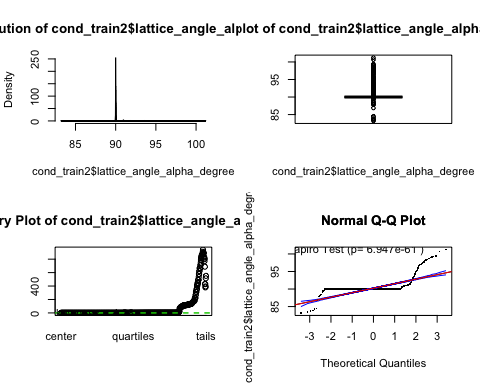
cond\_train2\_trans$lattice\_vector\_3\_ang = bcPower(cond\_train2$lattice\_vector\_3\_ang, -0.25)  
  
Statplot(cond\_train2\_trans$lattice\_vector\_3\_ang)



The next three predictors are angle degrees, which all seem to hover around 90 degrees with occasional variation coming off of each. This will hopefull scale easily as a result.

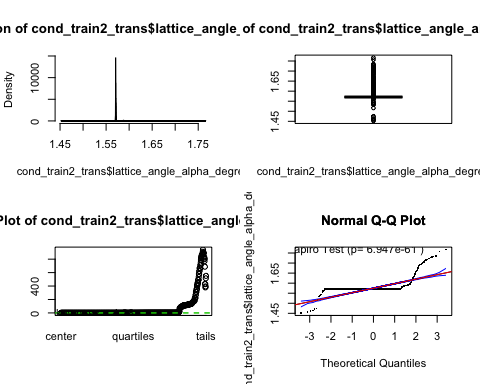
Starting with lattice angle alpha degree

Statplot(cond\_train2$lattice\_angle\_alpha\_degree)

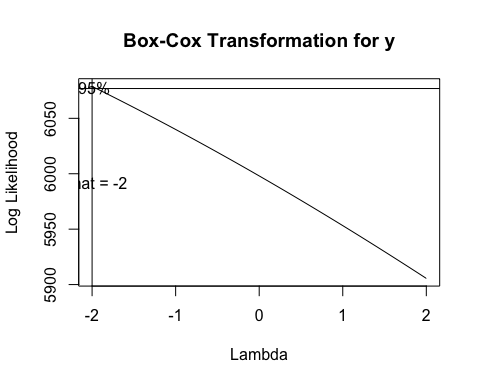


We can see some unique problems due to the nature of degree measures here, so we will try converting to radians in order to assist in this.

radians\_to\_degree <- function(rad) {  
 (rad \* 180) / (pi)  
}  
  
degree\_to\_rad <- function(deg) {  
 (deg \* pi) / (180)  
}  
  
  
cond\_train2\_trans$lattice\_angle\_alpha\_degree = degree\_to\_rad(cond\_train2$lattice\_angle\_alpha\_degree)  
  
Statplot(cond\_train2\_trans$lattice\_angle\_alpha\_degree)

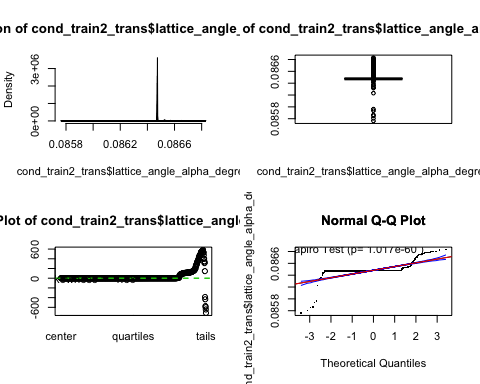
 This should hopefully allow for easier transformation.

myBC(cond\_train2\_trans$lattice\_angle\_alpha\_degree)



## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y -11.4851 -11.49 -12.4091 -10.561  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 535.1435 1 < 2.22e-16  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 624.8217 1 < 2.22e-16

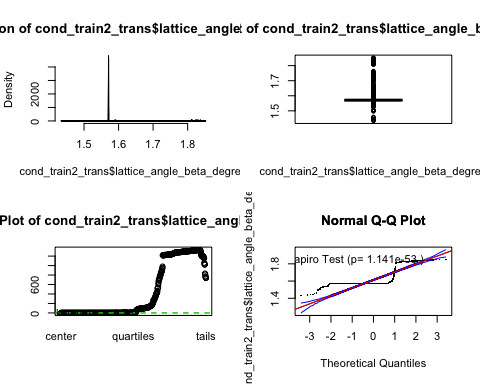
cond\_train2\_trans$lattice\_angle\_alpha\_degree = bcPower(cond\_train2\_trans$lattice\_angle\_alpha\_degree, -11.5 )  
  
Statplot(cond\_train2\_trans$lattice\_angle\_alpha\_degree)



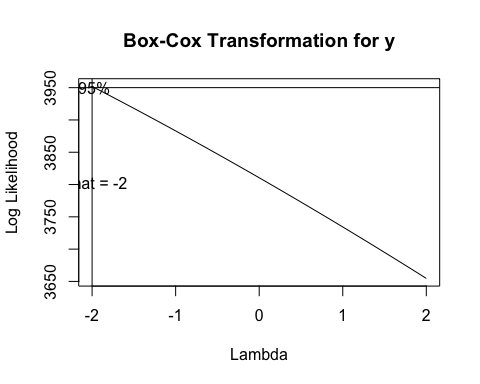
However, while it allows us to at least transform the data, it still does not allow us to truly fix it. We can box cox transform it with an abhorrent -11.5, but the massive amount of observations that close in around 90 simply leave us with odd results.

We see similar issues with the next to angle variables.

cond\_train2\_trans$lattice\_angle\_beta\_degree = degree\_to\_rad(cond\_train2$lattice\_angle\_beta\_degree)  
  
Statplot(cond\_train2\_trans$lattice\_angle\_beta\_degree)

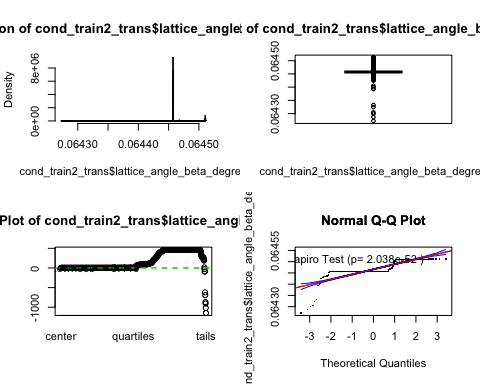


myBC(cond\_train2\_trans$lattice\_angle\_beta\_degree)

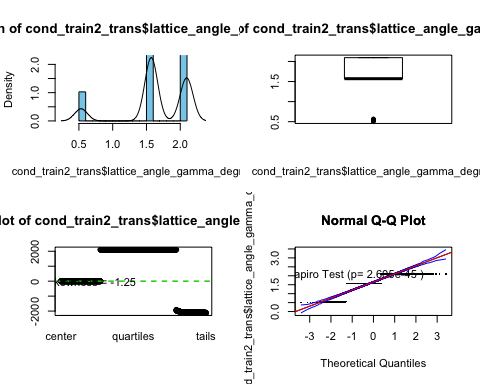


## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y -15.6096 -15.61 -16.3701 -14.849  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 1285.861 1 < 2.22e-16  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 1438.16 1 < 2.22e-16

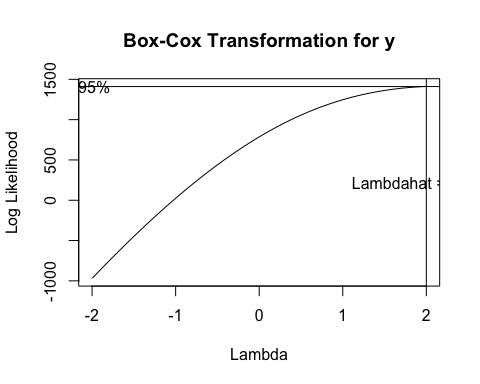
cond\_train2\_trans$lattice\_angle\_beta\_degree = bcPower(cond\_train2\_trans$lattice\_angle\_beta\_degree, -15.5 )  
  
Statplot(cond\_train2\_trans$lattice\_angle\_beta\_degree)



cond\_train2\_trans$lattice\_angle\_gamma\_degree = degree\_to\_rad(cond\_train2$lattice\_angle\_gamma\_degree)  
  
Statplot(cond\_train2\_trans$lattice\_angle\_gamma\_degree)

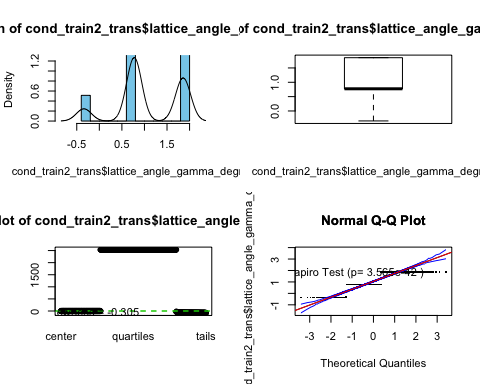


myBC(cond\_train2\_trans$lattice\_angle\_gamma\_degree)



## bcPower Transformation to Normality   
## Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd  
## y 2.2239 2.22 2.0791 2.3686  
##   
## Likelihood ratio test that transformation parameter is equal to 0  
## (log transformation)  
## LRT df pval  
## LR test, lambda = (0) 1262.534 1 < 2.22e-16  
##   
## Likelihood ratio test that no transformation is needed  
## LRT df pval  
## LR test, lambda = (1) 340.3614 1 < 2.22e-16

cond\_train2\_trans$lattice\_angle\_gamma\_degree = bcPower(cond\_train2\_trans$lattice\_angle\_gamma\_degree, 2.2)  
  
Statplot(cond\_train2\_trans$lattice\_angle\_gamma\_degree)



Ultimately, what this process shows us is that in their current states, even transformed, many of these variables are not likely to be of great use to us. However, with a little enginuity we might be able to stretch these a little farther.

Going back to Space groups, we can create categorical variables for each group as a way to focus in on that variables, as only 6 of the possible 230 existing space groups are actually used.

require(sjmisc)

## Loading required package: sjmisc

##   
## Attaching package: 'sjmisc'

## The following object is masked from 'package:tidyr':  
##   
## replace\_na

cond\_train2\_trans %>%   
 to\_dummy(spacegroup, suffix = "label" ) %>%   
 bind\_cols(cond\_train2\_trans) %>%   
 select(ID, spacegroup, everything()) -> cond\_train3  
  
cond\_train3 %>%  
 rename(Spacegroup\_12 = V1, Spacegroup\_33 = V2, Spacegroup\_167 = V3, Spacegroup\_194 = V4, Spacegroup\_206 = V5, Spacegroup\_227 = V6) -> cond\_train3

Number of total atoms, while numeric, only has 6 unique values across 1,567 rows. While the consistancy is nice, it does make our ability to transform it into a normal scale fairly difficuly as seen pervisouly. To circumvent this problem, we have elected to turn it into categorical data as well, as we feel that the benefit of being able to focus in on a category of atoms outweights the loss in ordinal power. However, certain models still will be fed one or the other based on their preference.

cond\_train3 %>%   
 to\_dummy(number\_of\_total\_atoms, suffix = "label" ) %>%   
 bind\_cols(cond\_train3) %>%   
 select(ID, number\_of\_total\_atoms, everything()) -> cond\_train3  
  
 cond\_train3 %>%  
 rename( atoms\_10 = V1, atoms\_20 = V2, atoms\_30 = V3, atoms\_40 = V4, atoms\_60 = V5, atoms\_80 = V6) -> cond\_train3

We also want to make dummy variables based on whether or not certain elements are present within a potential superconductor, as this can help mitigate colineation problems in situations where these percentage varaibles would otherwise be related, and additionally they may prove useful on their own.

cond\_train3 %>%  
 mutate(Aluminum = ifelse(percent\_atom\_al > 0, 1, 0), Gallium = ifelse(percent\_atom\_ga > 0, 1, 0), Indium = ifelse(percent\_atom\_in > 0, 1, 0)) -> cond\_train3

Now, we will go back and do all of this to the test set

library(car)  
cond\_test\_ready = cond\_teat  
  
cond\_test\_ready$number\_of\_total\_atoms = bcPower(cond\_teat$number\_of\_total\_atoms, 1.6)  
  
cond\_test\_ready$percent\_atom\_al = logit(cond\_teat$percent\_atom\_al)

## Warning in logit(cond\_teat$percent\_atom\_al): proportions remapped to  
## (0.025, 0.975)

cond\_test\_ready$percent\_atom\_ga = logit(cond\_teat$percent\_atom\_ga)

## Warning in logit(cond\_teat$percent\_atom\_ga): proportions remapped to  
## (0.025, 0.975)

cond\_test\_ready$percent\_atom\_in = logit(cond\_teat$percent\_atom\_in)

## Warning in logit(cond\_teat$percent\_atom\_in): proportions remapped to  
## (0.025, 0.975)

cond\_test\_ready$lattice\_vector\_1\_ang = bcPower(cond\_teat$lattice\_vector\_1\_ang, 0.25)  
  
cond\_test\_ready$lattice\_vector\_2\_ang = bcPower(cond\_teat$lattice\_vector\_2\_ang, 0.7)  
  
cond\_test\_ready$lattice\_vector\_3\_ang = bcPower(cond\_teat$lattice\_vector\_3\_ang, -0.25)  
  
cond\_test\_ready$lattice\_angle\_alpha\_degree = degree\_to\_rad(cond\_teat$lattice\_angle\_alpha\_degree)  
  
cond\_test\_ready$lattice\_angle\_beta\_degree = degree\_to\_rad(cond\_teat$lattice\_angle\_beta\_degree)  
  
cond\_test\_ready$lattice\_angle\_gamma\_degree = degree\_to\_rad(cond\_teat$lattice\_angle\_gamma\_degree)  
  
cond\_test\_ready$lattice\_angle\_alpha\_degree = bcPower(cond\_teat$lattice\_angle\_alpha\_degree, -11.5 )  
  
cond\_test\_ready$lattice\_angle\_beta\_degree = bcPower(cond\_teat$lattice\_angle\_beta\_degree, -15.5 )  
  
cond\_test\_ready$lattice\_angle\_gamma\_degree = bcPower(cond\_teat$lattice\_angle\_gamma\_degree, 2.2)  
  
#spacegroup  
cond\_test\_ready %>%   
 to\_dummy(spacegroup, suffix = "label" ) %>%   
 bind\_cols(cond\_test\_ready) %>%   
 select(ID, spacegroup, everything()) -> cond\_test\_ready  
  
cond\_test\_ready %>%  
 rename(Spacegroup\_12 = V1, Spacegroup\_33 = V2, Spacegroup\_167 = V3, Spacegroup\_194 = V4, Spacegroup\_206 = V5, Spacegroup\_227 = V6) -> cond\_test\_ready  
  
  
#total atoms  
cond\_test\_ready %>%   
 to\_dummy(number\_of\_total\_atoms, suffix = "label" ) %>%   
 bind\_cols(cond\_test\_ready) %>%   
 select(ID, number\_of\_total\_atoms, everything()) -> cond\_test\_ready  
  
 cond\_test\_ready %>%  
 rename( atoms\_10 = V1, atoms\_20 = V2, atoms\_30 = V3, atoms\_40 = V4, atoms\_60 = V5, atoms\_80 = V6) -> cond\_test\_ready  
  
   
 cond\_test\_ready %>%  
 mutate(Aluminum = ifelse(percent\_atom\_al > 0, 1, 0), Gallium = ifelse(percent\_atom\_ga > 0, 1, 0), Indium = ifelse(percent\_atom\_in > 0, 1, 0)) -> cond\_test\_ready  
  
   
   
   
cond\_test\_ready %>%  
 rename( formation\_energy\_ev\_natom\_log\_y\_plus\_1 = formation\_energy\_ev\_natom, bandgap\_energy\_ev\_log\_y\_plus\_1 = bandgap\_energy\_ev) -> cond\_test\_ready

Writing both .csv files for future use

write.csv(cond\_train3, "cond\_train\_new\_variables.csv", row.names = FALSE)  
  
write.csv(cond\_test\_ready, "cond\_test\_new\_variables.csv", row.names = FALSE)

# Exploring structure within predictors with PCA

Now that we have decided on our predictors, we thought it may prove useful to conduct a simple principle component analysis on the variables to see if we could find any patterns, as this may help decide certain metrics latter on and help with interpretations.

Load data and prep:

###### Load and prepare data ######  
#Load data:  
{#setwd("~/OneDrive - MNSCU/myGithub/Supervised\_Learning/Multiple\_Linear\_Regression/MachineLearning-SupervisedLearning/Conductor Formulations")  
 #Train set  
 cond\_train = read.csv("Conductors (train).csv")  
 #Test set  
 cond\_test = read.csv("Conductors (test).csv")  
   
 ### New training sets  
 cond\_train\_new\_vars = read.csv("cond\_train\_new\_variables.csv")  
 #No\_box\_cox  
 #angles measures are still in degrees; no transformaitons on predictors  
 cond\_train\_new\_vars\_no\_box\_cox = read.csv("cond\_train\_new\_variables\_no\_box\_cox.csv")  
   
 ####### New Variables #######  
   
 #Resposne vectors   
 y\_train\_form\_eng\_trans = cond\_train\_new\_vars$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
 y\_train\_bandgap\_eng\_trans = cond\_train\_new\_vars$bandgap\_energy\_ev\_log\_y\_plus\_1  
   
 #Resposne vectors for no box\_cox  
 #y\_train\_form\_eng\_trans = cond\_train\_new\_vars\_no\_box\_cox$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
 #y\_train\_bandgap\_eng\_trans = cond\_train\_new\_vars\_no\_box\_cox$bandgap\_energy\_ev\_log\_y\_plus\_1  
   
 #Df of all x's  
 x\_train = cond\_train\_new\_vars[,-c(1,25,26)]  
 #x\_train =cond\_train\_new\_vars\_no\_box\_cox[,-c(1,2,26,27)]  
}  
#Rearrange the data for two models: Model 1: formation energy as response; Model 2: bandgap energy as response  
{#detach("package:Ecfun", unload = TRUE)  
 cond\_train\_new\_vars %>%  
 select(25,2:24,27:29) -> train\_model1  
   
 cond\_train\_new\_vars%>%  
 select(26, 2:24,27:29) -> train\_model2  
}   
#Rearrange the data for two models for no box\_cox  
{  
 #detach("package:Ecfun", unload = TRUE)  
 cond\_train\_new\_vars\_no\_box\_cox %>%  
 select(26,3:25,28:30) -> train\_model1\_no\_box\_cox  
   
 cond\_train\_new\_vars\_no\_box\_cox%>%  
 select(27, 3:25,28:30) -> train\_model2\_no\_box\_cox  
}

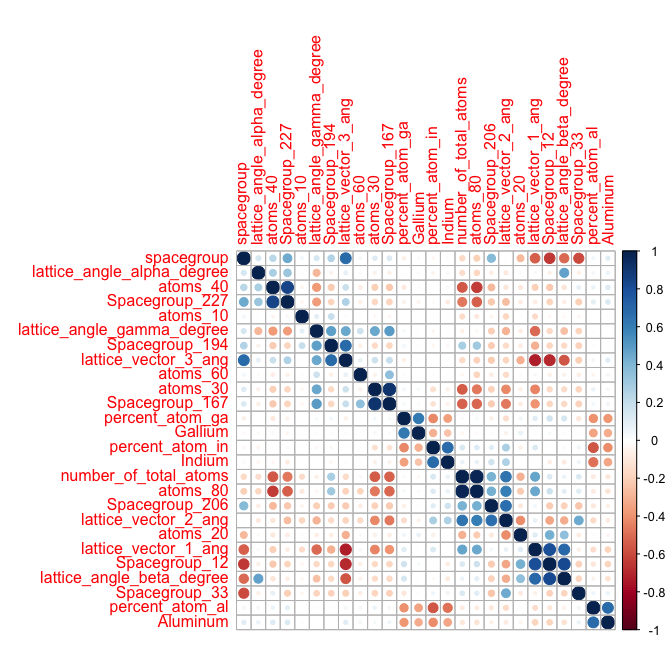
We are going to be working with ‘x\_train’ data frame. It only includes our final (added and transformed) independent variables.

x\_train.mat = as.matrix(x\_train)

## Correlation

Let’s first explore the correlation of our predictors by building a correlation plot:

{  
corr.x\_train = cor(x\_train.mat)  
#jpeg("corr.plot\_predictors.png", width = 500, height = 500)  
corrplot(corr.x\_train, order = 'hclust')  
#dev.off()  
}



Looking at a quick corrplot, we do not see a ton of correlation in our predictors, so we can expect that most will end up having at least something to add to the more advanced models unless they end up having only random correlation with the responses.

## Principal Component Analysis

Now, we are going to conduct principal component analysis using FactorMineR and factoextra libraries:

summary(cond.PCA)

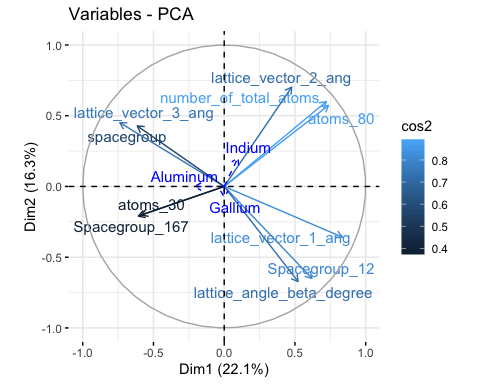
##   
## Call:  
## PCA(X = x\_train.scale, quanti.sup = c(24, 25, 26))   
##   
##   
## Eigenvalues  
## Dim.1 Dim.2 Dim.3 Dim.4 Dim.5 Dim.6  
## Variance 5.079 3.755 3.031 2.028 1.700 1.512  
## % of var. 22.085 16.326 13.177 8.817 7.389 6.576  
## Cumulative % of var. 22.085 38.410 51.587 60.404 67.793 74.369  
## Dim.7 Dim.8 Dim.9 Dim.10 Dim.11 Dim.12  
## Variance 1.271 1.173 1.056 0.978 0.829 0.327  
## % of var. 5.526 5.101 4.589 4.253 3.602 1.420  
## Cumulative % of var. 79.895 84.996 89.585 93.838 97.441 98.860  
## Dim.13 Dim.14 Dim.15 Dim.16 Dim.17 Dim.18  
## Variance 0.182 0.058 0.020 0.001 0.001 0.000  
## % of var. 0.793 0.253 0.085 0.006 0.002 0.000  
## Cumulative % of var. 99.653 99.906 99.991 99.998 100.000 100.000  
## Dim.19 Dim.20 Dim.21 Dim.22 Dim.23  
## Variance 0.000 0.000 0.000 0.000 0.000  
## % of var. 0.000 0.000 0.000 0.000 0.000  
## Cumulative % of var. 100.000 100.000 100.000 100.000 100.000  
##   
## Individuals (the 10 first)  
## Dist Dim.1 ctr cos2 Dim.2 ctr  
## 1 | 3.666 | 1.571 0.031 0.184 | 0.555 0.005  
## 2 | 4.791 | -1.820 0.042 0.144 | -1.750 0.052  
## 3 | 5.095 | -3.508 0.155 0.474 | -0.840 0.012  
## 4 | 4.355 | -0.621 0.005 0.020 | 2.186 0.081  
## 5 | 4.698 | -1.721 0.037 0.134 | -1.820 0.056  
## 6 | 4.187 | -1.018 0.013 0.059 | 1.854 0.058  
## 7 | 7.643 | -2.325 0.068 0.093 | -0.959 0.016  
## 8 | 4.146 | -1.080 0.015 0.068 | 1.870 0.059  
## 9 | 4.508 | -0.662 0.006 0.022 | -0.707 0.008  
## 10 | 3.993 | 1.252 0.020 0.098 | 1.609 0.044  
## cos2 Dim.3 ctr cos2   
## 1 0.023 | 0.087 0.000 0.001 |  
## 2 0.133 | -3.415 0.246 0.508 |  
## 3 0.027 | 2.019 0.086 0.157 |  
## 4 0.252 | 1.292 0.035 0.088 |  
## 5 0.150 | -3.321 0.232 0.500 |  
## 6 0.196 | 1.378 0.040 0.108 |  
## 7 0.016 | 2.368 0.118 0.096 |  
## 8 0.204 | 1.303 0.036 0.099 |  
## 9 0.025 | -1.473 0.046 0.107 |  
## 10 0.162 | -0.019 0.000 0.000 |  
##   
## Variables (the 10 first)  
## Dim.1 ctr cos2 Dim.2 ctr cos2   
## number\_of\_total\_atoms | 0.722 10.250 0.521 | 0.600 9.584 0.360 |  
## atoms\_10 | -0.121 0.290 0.015 | -0.008 0.002 0.000 |  
## atoms\_20 | 0.127 0.315 0.016 | -0.525 7.353 0.276 |  
## atoms\_30 | -0.587 6.777 0.344 | -0.201 1.078 0.040 |  
## atoms\_40 | -0.378 2.806 0.143 | -0.276 2.030 0.076 |  
## atoms\_60 | -0.141 0.392 0.020 | -0.052 0.072 0.003 |  
## atoms\_80 | 0.737 10.702 0.544 | 0.573 8.740 0.328 |  
## spacegroup | -0.615 7.443 0.378 | 0.428 4.876 0.183 |  
## Spacegroup\_12 | 0.620 7.567 0.384 | -0.650 11.250 0.422 |  
## Spacegroup\_33 | 0.233 1.071 0.054 | 0.116 0.359 0.013 |  
## Dim.3 ctr cos2   
## number\_of\_total\_atoms 0.127 0.532 0.016 |  
## atoms\_10 0.076 0.189 0.006 |  
## atoms\_20 0.150 0.746 0.023 |  
## atoms\_30 0.519 8.880 0.269 |  
## atoms\_40 -0.809 21.618 0.655 |  
## atoms\_60 0.190 1.190 0.036 |  
## atoms\_80 0.200 1.322 0.040 |  
## spacegroup -0.278 2.546 0.077 |  
## Spacegroup\_12 0.262 2.263 0.069 |  
## Spacegroup\_33 -0.147 0.716 0.022 |  
##   
## Supplementary continuous variables  
## Dim.1 cos2 Dim.2 cos2 Dim.3 cos2  
## Aluminum | -0.209 0.044 | -0.001 0.000 | -0.110 0.012  
## Gallium | 0.002 0.000 | -0.076 0.006 | 0.116 0.014  
## Indium | 0.097 0.009 | 0.203 0.041 | -0.072 0.005  
##   
## Aluminum |  
## Gallium |  
## Indium |

Below, we see the significant variables contribution to the 1st and 2nd dimensions:

#Significant correlation of variables to the first 2 dimensions  
dimdesc(cond.PCA, axes=c(1,2))

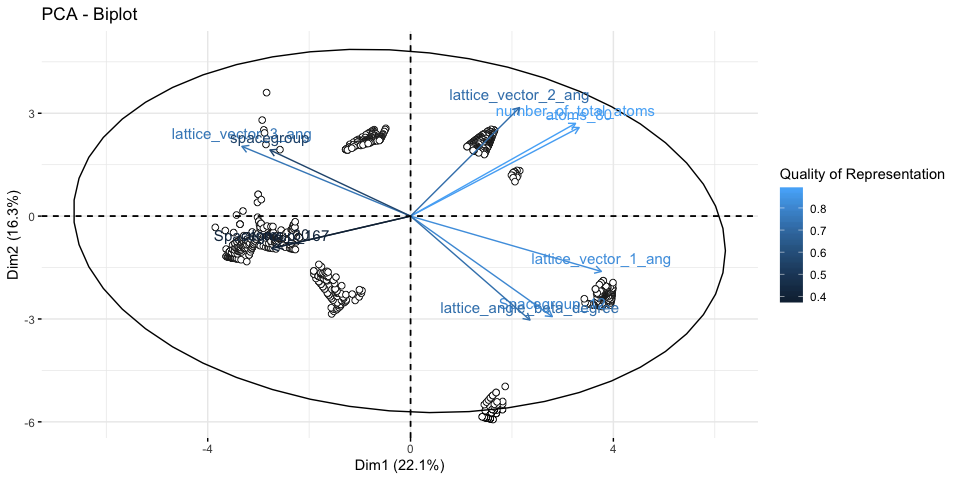
## $Dim.1  
## $Dim.1$quanti  
## correlation p.value  
## lattice\_vector\_1\_ang 0.83462990 0.000000e+00  
## atoms\_80 0.73730378 7.280116e-269  
## number\_of\_total\_atoms 0.72156455 3.572875e-252  
## Spacegroup\_12 0.61997838 4.495244e-167  
## lattice\_angle\_beta\_degree 0.52223894 2.297369e-110  
## lattice\_vector\_2\_ang 0.47666536 1.080533e-89  
## Spacegroup\_206 0.29766804 1.986333e-33  
## Spacegroup\_33 0.23326355 8.296522e-21  
## percent\_atom\_in 0.18247823 3.365616e-13  
## atoms\_20 0.12655088 5.013870e-07  
## Indium 0.09732012 1.140669e-04  
## percent\_atom\_ga 0.06881065 6.431119e-03  
## lattice\_angle\_alpha\_degree -0.06785986 7.205150e-03  
## atoms\_10 -0.12135515 1.450997e-06  
## atoms\_60 -0.14104360 2.060364e-08  
## Spacegroup\_194 -0.18175007 4.187555e-13  
## percent\_atom\_al -0.18258939 3.254941e-13  
## Aluminum -0.20918582 5.919516e-17  
## atoms\_40 -0.37752302 2.967277e-54  
## Spacegroup\_227 -0.39675347 3.148030e-60  
## lattice\_angle\_gamma\_degree -0.44330679 1.992468e-76  
## atoms\_30 -0.58670008 1.414162e-145  
## Spacegroup\_167 -0.60731357 1.365747e-158  
## spacegroup -0.61487765 1.305042e-163  
## lattice\_vector\_3\_ang -0.73756534 3.754980e-269  
##   
##   
## $Dim.2  
## $Dim.2$quanti  
## correlation p.value  
## lattice\_vector\_2\_ang 0.70214645 4.170317e-233  
## number\_of\_total\_atoms 0.59988915 8.560551e-154  
## atoms\_80 0.57287144 2.347207e-137  
## Spacegroup\_206 0.52754914 5.520081e-113  
## lattice\_vector\_3\_ang 0.45095405 2.438887e-79  
## Spacegroup\_194 0.43124392 5.549538e-72  
## spacegroup 0.42790574 8.766327e-71  
## lattice\_angle\_gamma\_degree 0.23331218 8.140530e-21  
## Indium 0.20340211 4.276807e-16  
## percent\_atom\_in 0.19612853 4.732546e-15  
## Spacegroup\_33 0.11614633 4.033813e-06  
## atoms\_60 -0.05193735 3.981095e-02  
## percent\_atom\_al -0.06266820 1.309401e-02  
## Gallium -0.07632333 2.500463e-03  
## percent\_atom\_ga -0.16044652 1.684123e-10  
## atoms\_30 -0.20120565 8.924841e-16  
## Spacegroup\_167 -0.20973614 4.889176e-17  
## lattice\_angle\_alpha\_degree -0.25961567 1.489635e-25  
## atoms\_40 -0.27606362 8.336187e-29  
## Spacegroup\_227 -0.27929181 1.802362e-29  
## lattice\_vector\_1\_ang -0.35788168 1.486795e-48  
## atoms\_20 -0.52543300 6.184791e-112  
## Spacegroup\_12 -0.64994805 8.821162e-189  
## lattice\_angle\_beta\_degree -0.67198967 2.183665e-206

#Selecting only 8 features whose quality of representaiton was the highest:  
fviz\_pca\_var(cond.PCA, repel = T, select.var = list(cos2 = 10), col.var = "cos2")



When adding supplementary categorical variables to a plot of highly quality of representation, we can see that each of the three categories is pulled in the same are as three distinct groups. Each group seems to be compromised of its own singular vector angle, “Greek” letter degree, and one other unique numeric variable, which means we can likely expect Aluminum, Gallium, and Indium to be some of the biggest players in our model ahead, with each tending to bring along a gallery of associated variables.

fviz\_pca\_biplot(cond.PCA, repel = F,  
 #Variables  
 select.var = list(contrib=10),  
 #alpha.var = "contrib",   
 col.var = "cos2",  
 #gradient.cols = "RdYlBu",  
 #Individuals  
 select.ind = list(contrib = 1000),  
 geom.ind = "point",  
 #fill.ind = cond\_train\_new\_vars$spacegroup, col.ind = "black",  
 pointshape = 21, pointsize = 2,  
 palette = "jco",  
 addEllipses = TRUE,  
 #col.var = "contrib",   
 legend.title = list(fill = "spacegroup", color = "Quality of Representation",  
 alpha = "Cos2"),  
 cex=.7,   
 ggtheme = theme\_minimal())



We definitely can see some clusters in our training data, although not all of them line up perfectly in the major groups of directions as indicated by our variables. This is likely were the difficulty in predicting our responses will come from, as we will have to be able to strike a balance between disassociated yet strong predictors. This will probably result in our more complex methods performing better, as they can better brute force their way through this problem.

# Model Building

## MARS

### Model 1: formation energy

To start off, we will attempt a Mars Model using our new variables for just formation energy.

drop.cols <- c('bandgap\_energy\_ev\_log\_y\_plus\_1','ID')  
  
cond\_train3 %>%  
 select(-drop.cols) -> cond\_train3\_form  
  
  
  
require(earth)

## Loading required package: earth

## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

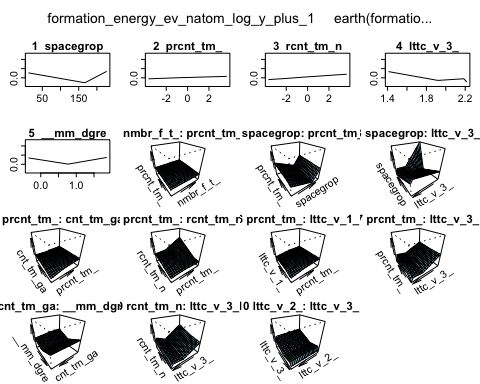
## Loading required package: TeachingDemos

form.mars = earth(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,degree = 2, data = cond\_train3\_form )  
  
summary(form.mars)

## Call: earth(formula=formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,  
## data=cond\_train3\_form, degree=2)  
##   
## coefficients  
## (Intercept) -0.0703477  
## h(167-spacegroup) 0.0033655  
## h(spacegroup-167) 0.0020583  
## h(-0.885013-percent\_atom\_al) -0.0309927  
## h(percent\_atom\_al- -0.885013) -0.0074377  
## h(-0.745168-percent\_atom\_in) -0.0366204  
## h(percent\_atom\_in- -0.745168) 0.0106559  
## h(2.18957-lattice\_vector\_3\_ang) -0.4742156  
## h(lattice\_vector\_3\_ang-2.18957) -2.9433049  
## h(0.772827-lattice\_angle\_gamma\_degree) 0.3014618  
## h(lattice\_angle\_gamma\_degree-0.772827) 0.3344752  
## h(number\_of\_total\_atoms-74.8022) \* h(-0.885013-percent\_atom\_al) -0.0000209  
## h(167-spacegroup) \* h(percent\_atom\_al- -2.8508) 0.0000445  
## h(167-spacegroup) \* h(-2.8508-percent\_atom\_al) -0.0004919  
## h(spacegroup-167) \* h(lattice\_vector\_3\_ang-1.91764) -0.0044545  
## h(spacegroup-167) \* h(1.91764-lattice\_vector\_3\_ang) 0.0516293  
## h(-0.885013-percent\_atom\_al) \* h(percent\_atom\_ga- -1.78449) 0.0065019  
## h(-0.885013-percent\_atom\_al) \* h(-1.78449-percent\_atom\_ga) -0.0168457  
## h(percent\_atom\_al- -0.885013) \* h(percent\_atom\_in- -2.04943) 0.0235761  
## h(-0.885013-percent\_atom\_al) \* h(lattice\_vector\_1\_ang-2.37221) -0.0095672  
## h(-0.885013-percent\_atom\_al) \* h(2.37221-lattice\_vector\_1\_ang) -0.0284676  
## h(-2.38434-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.18957) -8.0030636  
## h(percent\_atom\_al- -2.38434) \* h(lattice\_vector\_3\_ang-2.18957) -2.1554418  
## h(-0.885013-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.20375) 4.4252831  
## h(-0.885013-percent\_atom\_al) \* h(2.20375-lattice\_vector\_3\_ang) 0.0507585  
## h(-0.484246-percent\_atom\_ga) \* h(0.772827-lattice\_angle\_gamma\_degree) 0.0072462  
## h(percent\_atom\_ga- -0.484246) \* h(0.772827-lattice\_angle\_gamma\_degree) -0.0247753  
## h(percent\_atom\_in- -0.745168) \* h(lattice\_vector\_3\_ang-1.95538) 0.1709035  
## h(percent\_atom\_in- -0.745168) \* h(1.95538-lattice\_vector\_3\_ang) 0.0972542  
## h(5.62647-lattice\_vector\_2\_ang) \* h(2.18957-lattice\_vector\_3\_ang) 0.0126369  
## h(lattice\_vector\_2\_ang-5.62647) \* h(2.18957-lattice\_vector\_3\_ang) -0.8341012  
##   
## Selected 31 of 33 terms, and 9 of 26 predictors  
## Termination condition: RSq changed by less than 0.001 at 33 terms  
## Importance: percent\_atom\_in, lattice\_vector\_3\_ang, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 10 20  
## GCV 0.001095939 RSS 1.554793 GRSq 0.8549565 RSq 0.8685168

plotmo(form.mars)

## plotmo grid: number\_of\_total\_atoms atoms\_10 atoms\_20 atoms\_30 atoms\_40  
## 692.5198 0 0 0 0  
## atoms\_60 atoms\_80 spacegroup Spacegroup\_12 Spacegroup\_33 Spacegroup\_167  
## 0 1 194 0 0 0  
## Spacegroup\_194 Spacegroup\_206 Spacegroup\_227 percent\_atom\_al  
## 0 0 0 -0.484246  
## percent\_atom\_ga percent\_atom\_in lattice\_vector\_1\_ang lattice\_vector\_2\_ang  
## -0.7451684 -1.033015 3.029535 3.802559  
## lattice\_vector\_3\_ang lattice\_angle\_alpha\_degree lattice\_angle\_beta\_degree  
## 1.757565 0.08647366 0.06445731  
## lattice\_angle\_gamma\_degree Aluminum Gallium Indium  
## 0.7730218 0 0 0



Just starting out, we have a generalized R^2 of roughly 0.85% before we start going crazing with parameterization, which is promising as far as the maximum potential goes. It also only picked 9 of the 26 predictors. Next, we will try to play around with a few settings to improve the model.

form.mars = earth(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,degree = 1, data = cond\_train3\_form )  
  
summary(form.mars)

## Call: earth(formula=formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,  
## data=cond\_train3\_form, degree=1)  
##   
## coefficients  
## (Intercept) 2.40686  
## h(228.028-number\_of\_total\_atoms) 0.00234  
## h(number\_of\_total\_atoms-228.028) -0.00069  
## h(167-spacegroup) 0.00035  
## h(spacegroup-167) 0.00255  
## h(percent\_atom\_al- -2.8508) -0.03249  
## h(-0.885013-percent\_atom\_al) -0.05171  
## h(percent\_atom\_al- -0.885013) 0.06095  
## h(percent\_atom\_al-0.360287) -0.03412  
## h(1.03302-percent\_atom\_ga) 0.00967  
## h(percent\_atom\_ga-1.03302) 0.02837  
## h(-0.745168-percent\_atom\_in) -0.02464  
## h(percent\_atom\_in- -0.745168) -0.02337  
## h(1.93706-lattice\_vector\_1\_ang) 0.13228  
## h(lattice\_vector\_1\_ang-1.93706) 1.49651  
## h(lattice\_vector\_1\_ang-1.97456) -1.21432  
## h(lattice\_vector\_1\_ang-4.91885) -8.27536  
## h(lattice\_vector\_1\_ang-4.92558) 12.07464  
## h(lattice\_vector\_2\_ang-1.83805) -0.42473  
## h(lattice\_vector\_2\_ang-5.33558) -0.17947  
## h(5.62647-lattice\_vector\_2\_ang) -0.52843  
## h(2.18957-lattice\_vector\_3\_ang) -1.39581  
## h(lattice\_vector\_3\_ang-2.18957) -2.52037  
## h(lattice\_angle\_alpha\_degree-0.0865225) -679.55683  
## h(lattice\_angle\_alpha\_degree-0.0865886) 780.90244  
## h(0.772827-lattice\_angle\_gamma\_degree) -0.41873  
## h(lattice\_angle\_gamma\_degree-0.772827) -0.21762  
## h(lattice\_angle\_gamma\_degree-1.85646) 129.54914  
## h(lattice\_angle\_gamma\_degree-1.85677) -148.26987  
##   
## Selected 29 of 31 terms, and 10 of 26 predictors  
## Termination condition: RSq changed by less than 0.001 at 31 terms  
## Importance: percent\_atom\_in, lattice\_vector\_3\_ang, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 28 (additive model)  
## GCV 0.001288637 RSS 1.875061 GRSq 0.8294536 RSq 0.841433

Having interaction definitely is needed as shown above.

form.mars = earth(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,degree = 2, nk = 27, data = cond\_train3\_form )  
  
summary(form.mars)

## Call: earth(formula=formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,  
## data=cond\_train3\_form, degree=2, nk=27)  
##   
## coefficients  
## (Intercept) -0.0202186  
## h(167-spacegroup) 0.0032270  
## h(spacegroup-167) 0.0021104  
## h(-0.885013-percent\_atom\_al) -0.0236991  
## h(percent\_atom\_al- -0.885013) -0.0085880  
## h(-0.745168-percent\_atom\_in) -0.0376131  
## h(2.18957-lattice\_vector\_3\_ang) -0.5290584  
## h(lattice\_vector\_3\_ang-2.18957) -6.0080629  
## h(0.772827-lattice\_angle\_gamma\_degree) 0.2649207  
## h(lattice\_angle\_gamma\_degree-0.772827) 0.3010992  
## h(167-spacegroup) \* h(percent\_atom\_al- -2.8508) 0.0000405  
## h(167-spacegroup) \* h(-2.8508-percent\_atom\_al) -0.0004262  
## h(spacegroup-167) \* h(lattice\_vector\_3\_ang-1.91764) -0.0056773  
## h(spacegroup-167) \* h(1.91764-lattice\_vector\_3\_ang) 0.0488271  
## h(-0.885013-percent\_atom\_al) \* h(percent\_atom\_ga- -1.78449) 0.0040829  
## h(-0.885013-percent\_atom\_al) \* h(-1.78449-percent\_atom\_ga) -0.0126242  
## h(percent\_atom\_al- -0.885013) \* h(percent\_atom\_in- -2.04943) 0.0225318  
## h(-2.38434-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.18957) -7.3551671  
## h(percent\_atom\_al- -2.38434) \* h(lattice\_vector\_3\_ang-2.18957) -1.8221276  
## h(-0.484246-percent\_atom\_ga) \* h(0.772827-lattice\_angle\_gamma\_degree) 0.0082020  
## h(percent\_atom\_ga- -0.484246) \* h(0.772827-lattice\_angle\_gamma\_degree) -0.0232313  
## h(percent\_atom\_in- -0.745168) \* h(lattice\_vector\_3\_ang-1.95538) 0.3578530  
## h(percent\_atom\_in- -0.745168) \* h(1.95538-lattice\_vector\_3\_ang) 0.1136444  
## h(5.62647-lattice\_vector\_2\_ang) \* h(2.18957-lattice\_vector\_3\_ang) 0.0210238  
## h(lattice\_vector\_2\_ang-5.62647) \* h(2.18957-lattice\_vector\_3\_ang) -0.9065361  
##   
## Selected 25 of 27 terms, and 7 of 26 predictors  
## Termination condition: Reached nk 27  
## Importance: percent\_atom\_in, lattice\_vector\_3\_ang, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 9 15  
## GCV 0.001120209 RSS 1.621362 GRSq 0.8517444 RSq 0.8628874

After about 27 parameters, we stop gaining anything of note in terms of GRsq.

form.mars = earth(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,degree = 2, nk = 50, data = cond\_train3\_form, nprune = 20, pmethod = "exhaustive")

## Exhaustive pruning: number of subsets 7.9e+09 bx sing val ratio 1.1e-06

summary(form.mars)

## Call: earth(formula=formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,  
## data=cond\_train3\_form, pmethod="exhaustive", degree=2,  
## nprune=20, nk=50)  
##   
## coefficients  
## (Intercept) -0.0269222  
## h(167-spacegroup) 0.0025997  
## h(spacegroup-167) 0.0019556  
## h(-0.885013-percent\_atom\_al) -0.0201464  
## h(-0.745168-percent\_atom\_in) -0.0382737  
## h(2.18957-lattice\_vector\_3\_ang) -0.3167459  
## h(lattice\_vector\_3\_ang-2.18957) -5.6255477  
## h(0.772827-lattice\_angle\_gamma\_degree) 0.2358555  
## h(lattice\_angle\_gamma\_degree-0.772827) 0.2600089  
## h(167-spacegroup) \* h(-2.8508-percent\_atom\_al) -0.0004477  
## h(spacegroup-167) \* h(1.91764-lattice\_vector\_3\_ang) 0.0372149  
## h(-0.885013-percent\_atom\_al) \* h(percent\_atom\_ga- -1.78449) 0.0056276  
## h(percent\_atom\_al- -0.885013) \* h(percent\_atom\_in- -2.04943) 0.0210273  
## h(-0.885013-percent\_atom\_al) \* h(lattice\_vector\_1\_ang-2.37221) -0.0071044  
## h(-2.38434-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.18957) -9.2013837  
## h(-0.885013-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.20375) 6.1948447  
## h(percent\_atom\_ga- -0.484246) \* h(0.772827-lattice\_angle\_gamma\_degree) -0.0268502  
## h(percent\_atom\_in- -0.745168) \* h(1.95538-lattice\_vector\_3\_ang) 0.1153999  
## h(percent\_atom\_in- -0.745168) \* h(lattice\_vector\_3\_ang-1.95538) 0.1999221  
## h(lattice\_vector\_2\_ang-5.62647) \* h(2.18957-lattice\_vector\_3\_ang) -1.1825177  
##   
## Selected 20 of 33 terms, and 8 of 26 predictors (pmethod="exhaustive")  
## Termination condition: RSq changed by less than 0.001 at 33 terms  
## Importance: percent\_atom\_al, percent\_atom\_in, lattice\_vector\_3\_ang, ...  
## Number of terms at each degree of interaction: 1 8 11  
## GCV 0.001132773 RSS 1.666877 GRSq 0.8500816 RSq 0.8590383

Utilizing exhaustive pruning and cutting the model down to 20 variables, we can get a comparable R^2 with only 20 terms, which should help prevent us from overfitting.

form.mars = earth(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,degree = 2, nk = 50, data = cond\_train3\_form, nfold = 20, nprune = 20, pmethod = "exhaustive" )

## Exhaustive pruning: number of subsets 7.9e+09 bx sing val ratio 1.1e-06  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 2.8e-06  
## Exhaustive pruning: number of subsets 1.7e+10 bx sing val ratio 1.1e-06  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 3.5e-06  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 6e-10  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 3.3e-06  
## Exhaustive pruning: number of subsets 8.6e+09 bx sing val ratio 1e-06  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 1.7e-06  
## Exhaustive pruning: number of subsets 1.4e+11 bx sing val ratio 8e-09  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 2.9e-06  
## Exhaustive pruning: number of subsets 1.1e+09 bx sing val ratio 3e-06  
## Exhaustive pruning: number of subsets 8.6e+09 bx sing val ratio 2.8e-06  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 3.8e-06  
## Exhaustive pruning: number of subsets 1.7e+10 bx sing val ratio 1.1e-08  
## Exhaustive pruning: number of subsets 1.7e+10 bx sing val ratio 9.3e-07  
## Exhaustive pruning: number of subsets 8.6e+09 bx sing val ratio 1.1e-06  
## Exhaustive pruning: number of subsets 1.1e+09 bx sing val ratio 3.8e-10  
## Exhaustive pruning: number of subsets 3.4e+10 bx sing val ratio 1e-08

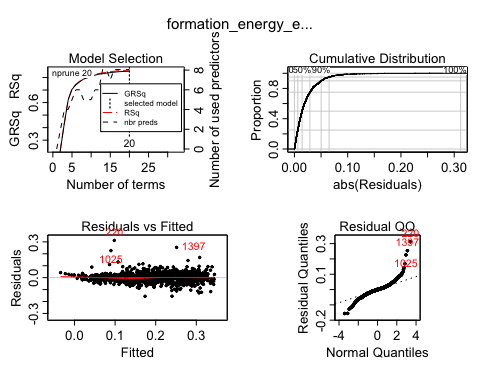
summary(form.mars)

## Call: earth(formula=formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,  
## data=cond\_train3\_form, pmethod="exhaustive", degree=2,  
## nprune=20, nfold=20, nk=50)  
##   
## coefficients  
## (Intercept) -0.0269222  
## h(167-spacegroup) 0.0025997  
## h(spacegroup-167) 0.0019556  
## h(-0.885013-percent\_atom\_al) -0.0201464  
## h(-0.745168-percent\_atom\_in) -0.0382737  
## h(2.18957-lattice\_vector\_3\_ang) -0.3167459  
## h(lattice\_vector\_3\_ang-2.18957) -5.6255477  
## h(0.772827-lattice\_angle\_gamma\_degree) 0.2358555  
## h(lattice\_angle\_gamma\_degree-0.772827) 0.2600089  
## h(167-spacegroup) \* h(-2.8508-percent\_atom\_al) -0.0004477  
## h(spacegroup-167) \* h(1.91764-lattice\_vector\_3\_ang) 0.0372149  
## h(-0.885013-percent\_atom\_al) \* h(percent\_atom\_ga- -1.78449) 0.0056276  
## h(percent\_atom\_al- -0.885013) \* h(percent\_atom\_in- -2.04943) 0.0210273  
## h(-0.885013-percent\_atom\_al) \* h(lattice\_vector\_1\_ang-2.37221) -0.0071044  
## h(-2.38434-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.18957) -9.2013837  
## h(-0.885013-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-2.20375) 6.1948447  
## h(percent\_atom\_ga- -0.484246) \* h(0.772827-lattice\_angle\_gamma\_degree) -0.0268502  
## h(percent\_atom\_in- -0.745168) \* h(1.95538-lattice\_vector\_3\_ang) 0.1153999  
## h(percent\_atom\_in- -0.745168) \* h(lattice\_vector\_3\_ang-1.95538) 0.1999221  
## h(lattice\_vector\_2\_ang-5.62647) \* h(2.18957-lattice\_vector\_3\_ang) -1.1825177  
##   
## Selected 20 of 33 terms, and 8 of 26 predictors (pmethod="exhaustive")  
## Termination condition: RSq changed by less than 0.001 at 33 terms  
## Importance: percent\_atom\_al, percent\_atom\_in, lattice\_vector\_3\_ang, ...  
## Number of terms at each degree of interaction: 1 8 11  
## GCV 0.001132773 RSS 1.666877 GRSq 0.8500816 RSq 0.8590383 CVRSq 0.8435465  
##   
## Note: the cross-validation sd's below are standard deviations across folds  
##   
## Cross validation: nterms 26.85 sd 3.17 nvars 8.55 sd 1.19  
##   
## CVRSq sd MaxErr sd  
## 0.844 0.065 0.315 0.156

Internal CV using 20 folds shows that we can consistantly reach this level, at least just on this set.

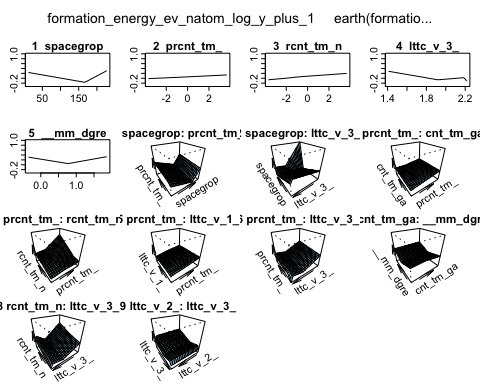
We will now use the evimp command to look at variable importance for this model.

plot(form.mars)



plotmo(form.mars)

## plotmo grid: number\_of\_total\_atoms atoms\_10 atoms\_20 atoms\_30 atoms\_40  
## 692.5198 0 0 0 0  
## atoms\_60 atoms\_80 spacegroup Spacegroup\_12 Spacegroup\_33 Spacegroup\_167  
## 0 1 194 0 0 0  
## Spacegroup\_194 Spacegroup\_206 Spacegroup\_227 percent\_atom\_al  
## 0 0 0 -0.484246  
## percent\_atom\_ga percent\_atom\_in lattice\_vector\_1\_ang lattice\_vector\_2\_ang  
## -0.7451684 -1.033015 3.029535 3.802559  
## lattice\_vector\_3\_ang lattice\_angle\_alpha\_degree lattice\_angle\_beta\_degree  
## 1.757565 0.08647366 0.06445731  
## lattice\_angle\_gamma\_degree Aluminum Gallium Indium  
## 0.7730218 0 0 0



evimp(form.mars)

## nsubsets gcv rss  
## percent\_atom\_al 19 100.0 100.0  
## percent\_atom\_in 19 100.0 100.0  
## lattice\_vector\_3\_ang 18 85.4 85.4  
## spacegroup 17 71.5 71.6  
## lattice\_vector\_2\_ang 15 41.2 41.7  
## number\_of\_total\_atoms-unused 12 65.3> 65.3>  
## lattice\_angle\_gamma\_degree 9 19.2 20.0  
## percent\_atom\_ga 7 15.4 16.2  
## lattice\_vector\_1\_ang 1 4.5 5.0

Looking here, we can see that percent atoms for Aluminum and Indium are the most important, with lattice vector 3 ang, 2 ang, spacegroup, and total atoms also pulling weight among a few others. None of our new categorical variables made the cut, but that makes sense given that Mars greatly prefers numeric data.

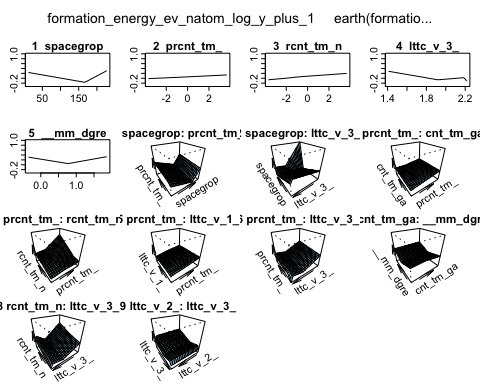
evimp(form.mars, trim = FALSE)

## nsubsets gcv rss  
## percent\_atom\_al 19 100.0 100.0  
## percent\_atom\_in 19 100.0 100.0  
## lattice\_vector\_3\_ang 18 85.4 85.4  
## spacegroup 17 71.5 71.6  
## lattice\_vector\_2\_ang 15 41.2 41.7  
## number\_of\_total\_atoms-unused 12 65.3> 65.3>  
## lattice\_angle\_gamma\_degree 9 19.2 20.0  
## percent\_atom\_ga 7 15.4 16.2  
## lattice\_vector\_1\_ang 1 4.5 5.0  
## atoms\_10-unused 0 0.0 0.0  
## atoms\_20-unused 0 0.0 0.0  
## atoms\_30-unused 0 0.0 0.0  
## atoms\_40-unused 0 0.0 0.0  
## atoms\_60-unused 0 0.0 0.0  
## atoms\_80-unused 0 0.0 0.0  
## Spacegroup\_12-unused 0 0.0 0.0  
## Spacegroup\_33-unused 0 0.0 0.0  
## Spacegroup\_167-unused 0 0.0 0.0  
## Spacegroup\_194-unused 0 0.0 0.0  
## Spacegroup\_206-unused 0 0.0 0.0  
## Spacegroup\_227-unused 0 0.0 0.0  
## lattice\_angle\_alpha\_degree-unused 0 0.0 0.0  
## lattice\_angle\_beta\_degree-unused 0 0.0 0.0  
## Aluminum-unused 0 0.0 0.0  
## Gallium-unused 0 0.0 0.0  
## Indium-unused 0 0.0 0.0

Looking at all of the variables, are current model uses almost all of contributing ones in some capacity other then number\_of\_total\_atoms used. However, because that number would be correlated with the amount of other elements used (due to the nature of those elements) it was likely disbarred for collineation reasons by the model.

plotmo(form.mars)

## plotmo grid: number\_of\_total\_atoms atoms\_10 atoms\_20 atoms\_30 atoms\_40  
## 692.5198 0 0 0 0  
## atoms\_60 atoms\_80 spacegroup Spacegroup\_12 Spacegroup\_33 Spacegroup\_167  
## 0 1 194 0 0 0  
## Spacegroup\_194 Spacegroup\_206 Spacegroup\_227 percent\_atom\_al  
## 0 0 0 -0.484246  
## percent\_atom\_ga percent\_atom\_in lattice\_vector\_1\_ang lattice\_vector\_2\_ang  
## -0.7451684 -1.033015 3.029535 3.802559  
## lattice\_vector\_3\_ang lattice\_angle\_alpha\_degree lattice\_angle\_beta\_degree  
## 1.757565 0.08647366 0.06445731  
## lattice\_angle\_gamma\_degree Aluminum Gallium Indium  
## 0.7730218 0 0 0



For form energy predictions using Mars, here we have our best.

### Model 2: bandgap energy

Next, we will devise a model for bandgap energy

drop.cols <- c('formation\_energy\_ev\_natom\_log\_y\_plus\_1','ID')  
  
cond\_train3 %>%  
 select(-drop.cols) -> cond\_train3\_band  
  
  
  
  
band.mars = earth(bandgap\_energy\_ev\_log\_y\_plus\_1~.,degree = 2, data = cond\_train3\_band )  
  
summary(band.mars)

## Call: earth(formula=bandgap\_energy\_ev\_log\_y\_plus\_1~.,  
## data=cond\_train3\_band, degree=2)  
##   
## coefficients  
## (Intercept) 1.2008  
## h(206-spacegroup) -0.0003  
## h(spacegroup-206) -0.0261  
## h(-1.78449-percent\_atom\_al) -0.0374  
## h(percent\_atom\_al- -1.78449) 0.1265  
## h(-2.1521-percent\_atom\_in) 0.0431  
## h(percent\_atom\_in- -2.1521) -0.1461  
## h(percent\_atom\_in-0.745168) 0.0991  
## h(1.99367-lattice\_vector\_3\_ang) -0.1132  
## h(lattice\_vector\_3\_ang-1.99367) -1.8514  
## h(0.772827-lattice\_angle\_gamma\_degree) 0.1126  
## h(lattice\_angle\_gamma\_degree-0.772827) 0.0659  
## h(-1.78449-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-1.9216) 0.5349  
## h(-1.78449-percent\_atom\_al) \* h(1.9216-lattice\_vector\_3\_ang) 0.1116  
## h(-2.8508-percent\_atom\_in) \* h(lattice\_vector\_3\_ang-1.99367) 0.7100  
## h(1.95023-lattice\_vector\_1\_ang) \* h(1.99367-lattice\_vector\_3\_ang) -3.1687  
## h(3.06842-lattice\_vector\_1\_ang) \* h(0.772827-lattice\_angle\_gamma\_degree) -12.6867  
## h(1.99367-lattice\_vector\_3\_ang) \* h(lattice\_angle\_alpha\_degree-0.0864747) 17410.5232  
##   
## Selected 18 of 21 terms, and 7 of 26 predictors  
## Termination condition: RSq changed by less than 0.001 at 21 terms  
## Importance: percent\_atom\_in, spacegroup, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 11 6  
## GCV 0.008203795 RSS 12.15152 GRSq 0.9248203 RSq 0.9288456

Starting off basic with intercation, we already have 92% GRSq. We some tweaking, we may be able to create a potentially powerful model here.

band.mars = earth(bandgap\_energy\_ev\_log\_y\_plus\_1~.,degree = 1, data = cond\_train3\_band )  
  
summary(band.mars)

## Call: earth(formula=bandgap\_energy\_ev\_log\_y\_plus\_1~.,  
## data=cond\_train3\_band, degree=1)  
##   
## coefficients  
## (Intercept) 1.234  
## Spacegroup\_167 0.229  
## h(206-spacegroup) 0.000  
## h(spacegroup-206) -0.011  
## h(percent\_atom\_al- -1.78449) 0.130  
## h(-2.1521-percent\_atom\_in) 0.056  
## h(percent\_atom\_in- -2.1521) -0.145  
## h(percent\_atom\_in-0.885013) 0.098  
## h(1.99367-lattice\_vector\_3\_ang) -0.114  
## h(lattice\_angle\_alpha\_degree-0.0864731) -80130.727  
## h(lattice\_angle\_alpha\_degree-0.0864746) 82429.062  
## h(lattice\_angle\_alpha\_degree-0.0865886) -2446.335  
## h(0.773052-lattice\_angle\_gamma\_degree) -0.184  
## h(lattice\_angle\_gamma\_degree-0.773052) -0.187  
##   
## Selected 14 of 16 terms, and 7 of 26 predictors  
## Termination condition: RSq changed by less than 0.001 at 16 terms  
## Importance: percent\_atom\_in, spacegroup, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 13 (additive model)  
## GCV 0.008687642 RSS 13.14844 GRSq 0.9203863 RSq 0.923008

From the run above, we can see that interaction does not gain us a whole lot, so we may wish to remove it to keep the model simple.

band.mars = earth(bandgap\_energy\_ev\_log\_y\_plus\_1~.,degree = 1, data = cond\_train3\_band, nk = 16 )  
  
summary(band.mars)

## Call: earth(formula=bandgap\_energy\_ev\_log\_y\_plus\_1~.,  
## data=cond\_train3\_band, degree=1, nk=16)  
##   
## coefficients  
## (Intercept) 1.15109  
## Spacegroup\_167 0.09428  
## h(206-spacegroup) -0.00030  
## h(spacegroup-206) -0.01962  
## h(percent\_atom\_al- -1.78449) 0.12782  
## h(-2.1521-percent\_atom\_in) 0.05470  
## h(percent\_atom\_in- -2.1521) -0.14313  
## h(percent\_atom\_in-0.885013) 0.09748  
## h(lattice\_vector\_3\_ang-1.99367) -1.02474  
## h(lattice\_angle\_alpha\_degree-0.0864746) 682.73487  
##   
## Selected 10 of 12 terms, and 6 of 26 predictors  
## Termination condition: Reached nk 16  
## Importance: percent\_atom\_in, spacegroup, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 9 (additive model)  
## GCV 0.009415481 RSS 14.39844 GRSq 0.9137164 RSq 0.9156885

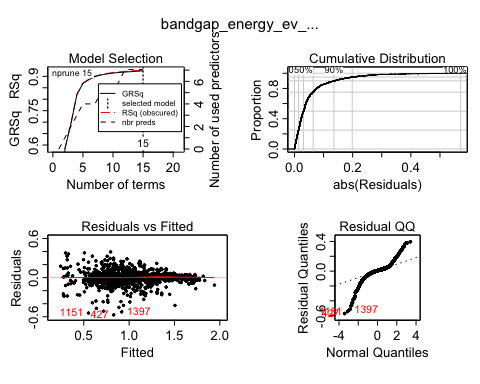
Unlike form energy, being able to zoom in on an exact amount is more difficult, so we will instead hit it harder with pruning combinations to try and simplfy or improve the model.

band.mars = earth(bandgap\_energy\_ev\_log\_y\_plus\_1~.,degree = 2, data = cond\_train3\_band, nk = 30, nfold = 20, nprune = 15, pmethod = "exhaustive" )  
  
summary(band.mars)

## Call: earth(formula=bandgap\_energy\_ev\_log\_y\_plus\_1~.,  
## data=cond\_train3\_band, pmethod="exhaustive", degree=2,  
## nprune=15, nfold=20, nk=30)  
##   
## coefficients  
## (Intercept) 1.1577  
## h(206-spacegroup) -0.0003  
## h(spacegroup-206) -0.0259  
## h(percent\_atom\_al- -1.78449) 0.1317  
## h(-2.1521-percent\_atom\_in) 0.0432  
## h(percent\_atom\_in- -2.1521) -0.1452  
## h(percent\_atom\_in-0.745168) 0.0913  
## h(lattice\_vector\_3\_ang-1.99367) -1.7495  
## h(0.772827-lattice\_angle\_gamma\_degree) 0.1281  
## h(lattice\_angle\_gamma\_degree-0.772827) 0.0814  
## h(-1.78449-percent\_atom\_al) \* h(lattice\_vector\_3\_ang-1.9216) 0.3935  
## h(-2.8508-percent\_atom\_in) \* h(lattice\_vector\_3\_ang-1.99367) 0.6751  
## h(1.95023-lattice\_vector\_1\_ang) \* h(1.99367-lattice\_vector\_3\_ang) -3.2471  
## h(3.06842-lattice\_vector\_1\_ang) \* h(0.772827-lattice\_angle\_gamma\_degree) -13.0959  
## h(1.99367-lattice\_vector\_3\_ang) \* h(lattice\_angle\_alpha\_degree-0.0864747) 18016.3312  
##   
## Selected 15 of 21 terms, and 7 of 26 predictors (pmethod="exhaustive")  
## Termination condition: RSq changed by less than 0.001 at 21 terms  
## Importance: percent\_atom\_in, spacegroup, percent\_atom\_al, ...  
## Number of terms at each degree of interaction: 1 9 5  
## GCV 0.008317691 RSS 12.44183 GRSq 0.9237766 RSq 0.9271457 CVRSq 0.9206003  
##   
## Note: the cross-validation sd's below are standard deviations across folds  
##   
## Cross validation: nterms 18.85 sd 1.31 nvars 7.40 sd 0.75  
##   
## CVRSq sd MaxErr sd  
## 0.921 0.031 -0.585 0.349

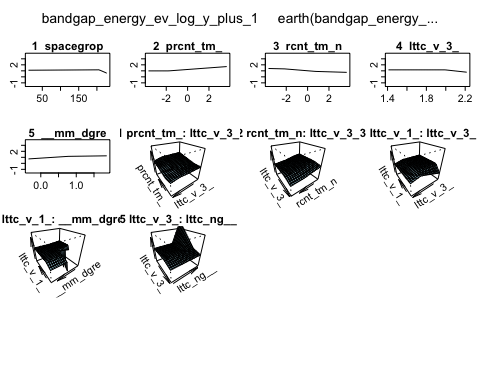
After 20 exhaustive k-fold method internal cross-validation, we found that an R^2 of aroud 0.92 was very consistant. Assuming this translates to actual prediction output, we have potentitally a very accurate model using fairly simple methodology.

plot(band.mars)



plotmo(band.mars)

## plotmo grid: number\_of\_total\_atoms atoms\_10 atoms\_20 atoms\_30 atoms\_40  
## 692.5198 0 0 0 0  
## atoms\_60 atoms\_80 spacegroup Spacegroup\_12 Spacegroup\_33 Spacegroup\_167  
## 0 1 194 0 0 0  
## Spacegroup\_194 Spacegroup\_206 Spacegroup\_227 percent\_atom\_al  
## 0 0 0 -0.484246  
## percent\_atom\_ga percent\_atom\_in lattice\_vector\_1\_ang lattice\_vector\_2\_ang  
## -0.7451684 -1.033015 3.029535 3.802559  
## lattice\_vector\_3\_ang lattice\_angle\_alpha\_degree lattice\_angle\_beta\_degree  
## 1.757565 0.08647366 0.06445731  
## lattice\_angle\_gamma\_degree Aluminum Gallium Indium  
## 0.7730218 0 0 0



evimp(band.mars)

## nsubsets gcv rss  
## percent\_atom\_in 14 100.0 100.0  
## spacegroup 13 60.8 60.8  
## percent\_atom\_al 12 50.0 50.1  
## lattice\_vector\_3\_ang 11 33.5 33.7  
## lattice\_angle\_alpha\_degree 8 16.4 16.8  
## lattice\_vector\_1\_ang 5 10.2 10.7  
## lattice\_angle\_gamma\_degree 4 8.7 9.2

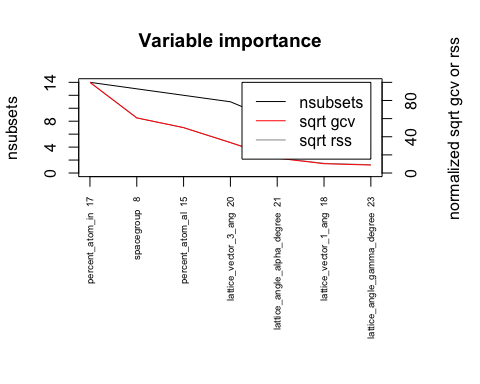
For predicting bandgap energy, a smaller number of variables are important. With spacegroup, percent\_atom\_in, vectors 1 and 3 and angles gamma and alpha being used. Percent atom in the structure though was by far the most important, being 40% of the total models power in our case.

evimp(band.mars, trim = FALSE)

## nsubsets gcv rss  
## percent\_atom\_in 14 100.0 100.0  
## spacegroup 13 60.8 60.8  
## percent\_atom\_al 12 50.0 50.1  
## lattice\_vector\_3\_ang 11 33.5 33.7  
## lattice\_angle\_alpha\_degree 8 16.4 16.8  
## lattice\_vector\_1\_ang 5 10.2 10.7  
## lattice\_angle\_gamma\_degree 4 8.7 9.2  
## number\_of\_total\_atoms-unused 0 0.0 0.0  
## atoms\_10-unused 0 0.0 0.0  
## atoms\_20-unused 0 0.0 0.0  
## atoms\_30-unused 0 0.0 0.0  
## atoms\_40-unused 0 0.0 0.0  
## atoms\_60-unused 0 0.0 0.0  
## atoms\_80-unused 0 0.0 0.0  
## Spacegroup\_12-unused 0 0.0 0.0  
## Spacegroup\_33-unused 0 0.0 0.0  
## Spacegroup\_167-unused 0 0.0 0.0  
## Spacegroup\_194-unused 0 0.0 0.0  
## Spacegroup\_206-unused 0 0.0 0.0  
## Spacegroup\_227-unused 0 0.0 0.0  
## percent\_atom\_ga-unused 0 0.0 0.0  
## lattice\_vector\_2\_ang-unused 0 0.0 0.0  
## lattice\_angle\_beta\_degree-unused 0 0.0 0.0  
## Aluminum-unused 0 0.0 0.0  
## Gallium-unused 0 0.0 0.0  
## Indium-unused 0 0.0 0.0

Compared to formation energy, way less variables were important for the Mars Model, with variables related to the core spatial structure of the observation being far more important than the chemnical makeup. In a sense, it is a bit of the inverse of formation energy as far as predicting.

band.evimp <- evimp(band.mars)  
  
plot(band.evimp, cex.var = 0.6)



#cross validation function  
MARS.cv = function(response,data, degree, nk, nprune, pmethod, p=.667,B=10) {  
 n <- length(response)  
 MSE <- rep(0,B)  
 MAE = rep(0,B)  
 MAPE = rep(0,B)  
 MSLE = rep(0,B)  
 for (i in 1:B) {  
 ss <- floor(n\*p)  
 sam <- sample(1:n,ss,replace=F)  
   
 fit2 <- earth(response[sam]~.,degree = degree, data = data[sam,], nk = nk, nfold = 1, nprune = nprune, pmethod = pmethod )  
   
 ynew <- predict(fit2,newdata=data[-sam,])  
 MSE[i] = mean((response[-sam]-ynew)^2)  
 MSLE[i] = mean((log(ynew+1) - log(response[-sam]+1))^2)  
 MAE[i] = mean(abs(response[-sam]-ynew))  
 MAPE[i] = mean(abs(response[-sam] - ynew)/response[-sam])  
 }  
 cat("RMSLEP =",sqrt(mean(MSLE)),"RMSEP =",sqrt(mean(MSE))," MAEP=",mean(MAE)," MAPEP=",mean(MAPE))  
 cv = return(data.frame(RMSLEP = sqrt(MSLE),RMSEP=sqrt(MSE),MAEP=MAE,MAPEP=MAPE))  
}  
  
  
MARS.cv(response = cond\_train3\_band$bandgap\_energy\_ev\_log\_y\_plus\_1, data = cond\_train3\_band[,-24], nk = 30, nprune = 15, pmethod = "exhaustive", degree = 2, B = 20, p=0.667)

## RMSLEP = 0.05604325 RMSEP = 0.09631577 MAEP= 0.05784702 MAPEP= 1.645897

## RMSLEP RMSEP MAEP MAPEP  
## 1 0.05426660 0.09479437 0.05718396 0.15013864  
## 2 0.05788186 0.10087911 0.06006838 0.11177777  
## 3 0.05539998 0.09537922 0.05562238 3.96677348  
## 4 0.04461826 0.07894663 0.04923898 0.07669893  
## 5 0.05780650 0.10072886 0.05998580 0.08207571  
## 6 0.06038841 0.10202315 0.05990641 0.21369045  
## 7 0.06056574 0.10165381 0.05928595 0.27163995  
## 8 0.05203075 0.09277966 0.05714706 0.07162583  
## 9 0.05288974 0.09298896 0.05672253 0.21061825  
## 10 0.04880358 0.08491354 0.05319712 0.08786679  
## 11 0.06256204 0.10239650 0.06144542 8.36353767  
## 12 0.05696759 0.09515652 0.05782640 0.27499763  
## 13 0.05162679 0.09046884 0.05556503 0.17817114  
## 14 0.05769218 0.09621332 0.05604885 0.31206769  
## 15 0.06326405 0.10781100 0.06211085 5.78182853  
## 16 0.05112877 0.09226358 0.05903679 4.58593804  
## 17 0.05916515 0.09716714 0.05699882 7.45553037  
## 18 0.05791335 0.10248384 0.06174122 0.25374161  
## 19 0.05104167 0.09068458 0.05770611 0.21939291  
## 20 0.06074279 0.10199546 0.06010238 0.24983727

#RMSLEP = 0.05490808 RMSEP = 0.09487381 MAEP= 0.05763223 MAPEP= 1.313522  
  
MARS.cv(response = cond\_train3\_form$formation\_energy\_ev\_natom\_log\_y\_plus\_1, data = cond\_train3\_form[,-24], nk = 50, nprune = 20, pmethod = "exhaustive", degree = 2, B = 20, p=0.667)

## Exhaustive pruning: number of subsets 4.1e+09 bx sing val ratio 1.7e-07  
## Exhaustive pruning: number of subsets 7.9e+09 bx sing val ratio 9.8e-09  
## Exhaustive pruning: number of subsets 2.9e+10 bx sing val ratio 3.6e-06  
## Exhaustive pruning: number of subsets 7.9e+09 bx sing val ratio 1.6e-10  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 3.5e-06  
## Exhaustive pruning: number of subsets 1.5e+10 bx sing val ratio 9.4e-09  
## Exhaustive pruning: number of subsets 1.5e+10 bx sing val ratio 1.7e-10  
## Exhaustive pruning: number of subsets 1.1e+09 bx sing val ratio 3.8e-07  
## Exhaustive pruning: number of subsets 1.1e+09 bx sing val ratio 6.1e-07  
## Exhaustive pruning: number of subsets 4.1e+09 bx sing val ratio 2.2e-07  
## Exhaustive pruning: number of subsets 1.5e+10 bx sing val ratio 5.1e-06  
## Exhaustive pruning: number of subsets 2.1e+09 bx sing val ratio 7.5e-08  
## RMSLEP = 0.03258738 RMSEP = 0.04228082 MAEP= 0.02402845 MAPEP= 0.2894962

## RMSLEP RMSEP MAEP MAPEP  
## 1 0.04352172 0.06225227 0.02604315 0.2404242  
## 2 0.03423618 0.04224687 0.02600427 0.2509599  
## 3 0.02538132 0.03006241 0.02131360 0.2194071  
## 4 0.02815692 0.03383979 0.02276735 0.2372492  
## 5 0.04498651 0.06451169 0.02729540 0.2495545  
## 6 0.02790435 0.03358343 0.02423582 0.2253161  
## 7 0.02966181 0.03609511 0.02394756 0.2227588  
## 8 0.03311689 0.04034985 0.02550137 0.2073614  
## 9 0.02986991 0.03573962 0.02464988 0.2547696  
## 10 0.02839031 0.03451762 0.02197766 0.1744979  
## 11 0.02815246 0.03368847 0.02313588 0.2384204  
## 12 0.02867039 0.03542648 0.02274080 0.2647282  
## 13 0.02548546 0.03075604 0.02097246 0.3443874  
## 14 0.03645910 0.04975936 0.02424492 0.4799169  
## 15 0.02842447 0.03378154 0.02311635 0.4184162  
## 16 0.02908683 0.03462580 0.02333284 0.3920579  
## 17 0.03212950 0.03909462 0.02475082 0.2204810  
## 18 0.04671434 0.07098803 0.02732002 0.5523494  
## 19 0.03211932 0.03945283 0.02479489 0.2183965  
## 20 0.02765410 0.03284420 0.02242404 0.3784722

#### Final Metrics

#RMSLEP = 0.03042171 RMSEP = 0.03754247 MAEP= 0.02394626 MAPEP= 0.2698358

While the Mars results look promising, certain variables were not fully utilized, particularly are categorical ones. Treed regression may benefit more from this, and also may pick on more direct patterns the Mars model missed, so it was tried as well.

## Treed Regression

### Model 1: formation energy

require(rpart)

## Loading required package: rpart

require(maptree)

## Loading required package: maptree

## Loading required package: cluster

require(Cubist)

## Loading required package: Cubist

## Loading required package: lattice

form\_test\_x = cond\_train3\_form[,-24]  
  
form\_test\_y = cond\_train3\_form[,24]  
  
cond\_cub = cubist(x = form\_test\_x, y = form\_test\_y, committees = 1)  
  
#summary(cond\_cub)

The summary output was shortened to just show evaluation results.

Evaluation on training data (1567 cases):

Average |error| 0.0236536  
Relative |error| 0.33  
Correlation coefficient 0.90  
  
  
Attribute usage:  
 Conds Model  
  
 84% 86% percent\_atom\_in  
 57% 87% lattice\_vector\_2\_ang  
 46% 95% percent\_atom\_al  
 41% 100% lattice\_vector\_3\_ang  
 39% 45% spacegroup  
 26% 54% percent\_atom\_ga  
 11% 36% lattice\_angle\_gamma\_degree  
 11% lattice\_angle\_alpha\_degree  
 10% 48% lattice\_vector\_1\_ang  
 8% lattice\_angle\_beta\_degree

Time: 0.1 secs

From here, we can see that with no boosting a basic treed model still gets a correlation coefficient of 0.90, which is impressive given how easy the implementation is. However, playing around with nearest neighbor adjustments, we may be able to push this further.

#cross validation function  
cubist.cv = function(x,y,p=.667,B=10,committees=1,neighbors=0) {  
 n <- length(y)  
 MSE <- rep(0,B)  
 MAE = rep(0,B)  
 MAPE = rep(0,B)  
 MSLE = rep(0,B)  
 for (i in 1:B) {  
 ss <- floor(n\*p)  
 sam <- sample(1:n,ss,replace=F)  
 fit2 <- cubist(x[sam,],y[sam],committees=committees,neighbors=neighbors)  
 ynew <- predict(fit2,newdata=x[-sam,],neighbors=neighbors)  
 MSE[i] = mean((y[-sam]-ynew)^2)  
 MSLE[i] = mean((log(ynew+1) - log(y[-sam]+1))^2)  
 MAE[i] = mean(abs(y[-sam]-ynew))  
 MAPE[i] = mean(abs(y[-sam] - ynew)/y[-sam])  
 }  
 cat("RMSLEP =",sqrt(mean(MSLE)),"RMSEP =",sqrt(mean(MSE))," MAEP=",mean(MAE)," MAPEP=",mean(MAPE))  
 cv = return(data.frame(RMSLEP = sqrt(MSLE),RMSEP=sqrt(MSE),MAEP=MAE,MAPEP=MAPE))  
}  
  
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 0)  
# RMSLEP = 0.03224218 RMSEP = 0.03927782 MAEP= 0.02463013 MAPEP= 0.2780565  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 1)  
# RMSLEP = 0.03555901 RMSEP = 0.04348276 MAEP= 0.02616518 MAPEP= 0.2537234  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 2)  
# RMSLEP = 0.03314788 RMSEP = 0.04045804 MAEP= 0.02460608 MAPEP= 0.2357997  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 3)  
# RMSLEP = 0.03148721 RMSEP = 0.03828172 MAEP= 0.02371599 MAPEP= 0.2223311  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 4)  
# RMSLEP = 0.03325151 RMSEP = 0.04083543 MAEP= 0.02441508 MAPEP= 0.2283243  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 5)  
# RMSLEP = 0.03205617 RMSEP = 0.03912071 MAEP= 0.02406652 MAPEP= 0.2333697  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 6)  
# RMSLEP = 0.03177122 RMSEP = 0.03893749 MAEP= 0.02381878 MAPEP= 0.219773  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 7)  
# RMSLEP = 0.03295414 RMSEP = 0.0406539 MAEP= 0.02417382 MAPEP= 0.239655  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 8)  
# RMSLEP = 0.03240087 RMSEP = 0.0398106 MAEP= 0.02403365 MAPEP= 0.2549661  
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 9)  
# RMSLEP = 0.03144095 RMSEP = 0.0382648 MAEP= 0.02347479 MAPEP= 0.2271004

Trying all possible values of neighbor, we can decisively say that using a setting of 9 bodes the best for our model. We will now use a similar process for boosting.

# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 1, neighbors = 9)  
# RMSLEP = 0.03237573 RMSEP = 0.03984964 MAEP= 0.02395975 MAPEP= 0.2510261  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 5, neighbors = 9)  
# RMSLEP = 0.02893934 RMSEP = 0.03519775 MAEP= 0.02235095 MAPEP= 0.2099355  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 10, neighbors = 9)  
# RMSLEP = 0.02937464 RMSEP = 0.03593288 MAEP= 0.02234299 MAPEP= 0.2145308  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 15, neighbors = 9)  
# RMSLEP = 0.02851337 RMSEP = 0.03484903 MAEP= 0.0219714 MAPEP= 0.2110919  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 20, neighbors = 9)  
# RMSLEP = 0.02828576 RMSEP = 0.03436924 MAEP= 0.02208001 MAPEP= 0.2271556  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 25, neighbors = 9)  
# RMSLEP = 0.02847509 RMSEP = 0.03471625 MAEP= 0.02216734 MAPEP= 0.229742  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 30, neighbors = 9)  
# RMSLEP = 0.0283022 RMSEP = 0.03460824 MAEP= 0.02184816 MAPEP= 0.2198067  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 35, neighbors = 9)  
# RMSLEP = 0.02717477 RMSEP = 0.0330505 MAEP= 0.02148992 MAPEP= 0.2043025  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 40, neighbors = 9)  
# RMSLEP = 0.02864012 RMSEP = 0.03495877 MAEP= 0.02190446 MAPEP= 0.2211497  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 45, neighbors = 9)  
# RMSLEP = 0.02848043 RMSEP = 0.03473952 MAEP= 0.021976 MAPEP= 0.2266264  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 50, neighbors = 9)  
# RMSLEP = 0.02855454 RMSEP = 0.034921 MAEP= 0.02209456 MAPEP= 0.2184926  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 55, neighbors = 9)  
# RMSLEP = 0.02790271 RMSEP = 0.03400922 MAEP= 0.02181917 MAPEP= 0.2154055  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 60, neighbors = 9)  
# RMSLEP = 0.02833183 RMSEP = 0.03463292 MAEP= 0.02193248 MAPEP= 0.2111712  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 65, neighbors = 9)  
# RMSLEP = 0.02879634 RMSEP = 0.03522894 MAEP= 0.02180916 MAPEP= 0.2033285  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 70, neighbors = 9)  
# RMSLEP = 0.02880328 RMSEP = 0.03512528 MAEP= 0.02213365 MAPEP= 0.2345095  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 75, neighbors = 9)  
# RMSLEP = 0.02778491 RMSEP = 0.03381687 MAEP= 0.0215106 MAPEP= 0.2213219  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 80, neighbors = 9)  
# RMSLEP = 0.02846212 RMSEP = 0.03467116 MAEP= 0.02176291 MAPEP= 0.2295154  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 85, neighbors = 9)  
# RMSLEP = 0.02895798 RMSEP = 0.03540791 MAEP= 0.02195497 MAPEP= 0.2213394  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 90, neighbors = 9)  
# RMSLEP = 0.02873599 RMSEP = 0.03510079 MAEP= 0.02202193 MAPEP= 0.2078274  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 95, neighbors = 9)  
# RMSLEP = 0.02885052 RMSEP = 0.03518716 MAEP= 0.02212498 MAPEP= 0.2482671  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 100, neighbors = 9)  
# RMSLEP = 0.02803035 RMSEP = 0.03412854 MAEP= 0.02171941 MAPEP= 0.2296888  
#

A boosting value of 35 narrowly produces the best value, leaving us with our final Treed model for Formation energy.

cubist\_form\_final = cubist(form\_test\_x,form\_test\_y, p = 0.667, B = 20, committees = 35, neighbors = 9)  
  
#summary(cubist\_form\_final)

Evaluation on training data (1567 cases):

Average |error| 0.0232752  
Relative |error| 0.32  
Correlation coefficient 0.90  
  
  
Attribute usage:  
 Conds Model  
  
 70% 91% percent\_atom\_in  
 57% 89% lattice\_vector\_3\_ang  
 45% 84% lattice\_vector\_2\_ang  
 39% 87% percent\_atom\_al  
 36% 39% spacegroup  
 11% 59% lattice\_vector\_1\_ang  
 9% 65% percent\_atom\_ga  
 5% lattice\_angle\_alpha\_degree  
 4% 60% lattice\_angle\_gamma\_degree  
 2% lattice\_angle\_beta\_degree  
 13% number\_of\_total\_atoms  
 11% atoms\_40  
 2% atoms\_30

#### Final Metrics

RMSLEP = 0.02717477 RMSEP = 0.0330505 MAEP= 0.02148992 MAPEP= 0.2043025

R^2 = 0.81

While our improvement over the base model were small, the reduction in error still is something to be considered and does still make it worth it to consider playing around with boosting and nearest neighbors.

### Model 2: bandgap energy

Now we will try bandgap as a response.

form\_test\_x2 = cond\_train3\_band[,-24]  
  
form\_test\_y2 = cond\_train3\_band[,24]  
  
cond\_cub2 = cubist(x = form\_test\_x2, y = form\_test\_y2, committees = 1)  
  
#summary(cond\_cub2)

Evaluation on training data (1567 cases):

Average |error| 0.0605275  
Relative |error| 0.22  
Correlation coefficient 0.94  
  
  
Attribute usage:  
 Conds Model  
  
 92% 95% lattice\_vector\_3\_ang  
 89% 80% percent\_atom\_in  
 81% 82% percent\_atom\_al  
 63% 65% spacegroup  
 37% 42% lattice\_vector\_1\_ang  
 13% 16% lattice\_angle\_gamma\_degree  
 10% 57% percent\_atom\_ga  
 4% 89% lattice\_vector\_2\_ang  
 11% atoms\_40

While an impressive start, having low error scores and a coefficient of 0.94, it still may prove useful to optimize the parameters to maximize our yields and reduce overfitting.

# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 0)  
# RMSLEP = 0.06537628 RMSEP = 0.1152551 MAEP= 0.06144679 MAPEP= 2.249768  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 1)  
# RMSLEP = 0.07195803 RMSEP = 0.1242237 MAEP= 0.06637847 MAPEP= 0.755666  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 2)  
# RMSLEP = 0.06775683 RMSEP = 0.1185319 MAEP= 0.06252493 MAPEP= 2.151726  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 3)  
# RMSLEP = 0.06736106 RMSEP = 0.1188582 MAEP= 0.06178664 MAPEP= 1.14858  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 4)  
# RMSLEP = 0.06934879 RMSEP = 0.1224337 MAEP= 0.06336363 MAPEP= 2.084187  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 5)  
# RMSLEP = 0.06341158 RMSEP = 0.1137942 MAEP= 0.05969285 MAPEP= 2.539457  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 6)  
# RMSLEP = 0.06681612 RMSEP = 0.1179883 MAEP= 0.06238921 MAPEP= 1.382312  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 7)  
# RMSLEP = 0.06543542 RMSEP = 0.1143767 MAEP= 0.06111267 MAPEP= 1.301673  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 8)  
# RMSLEP = 0.06572626 RMSEP = 0.1153895 MAEP= 0.06001655 MAPEP= 1.629123  
#   
#   
#   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 9)  
# RMSLEP = 0.06377824 RMSEP = 0.1139196 MAEP= 0.06060954 MAPEP= 2.925799

The optimal value for neighbors is 5. Now onto boosting.

# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 1, neighbors = 5)  
# RMSLEP = 0.06568027 RMSEP = 0.1131044 MAEP= 0.06050445 MAPEP= 2.551471  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 5, neighbors = 5)  
# RMSLEP = 0.05646231 RMSEP = 0.09855742 MAEP= 0.05619269 MAPEP= 2.47747  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 10, neighbors = 5)  
# RMSLEP = 0.0557989 RMSEP = 0.09769368 MAEP= 0.05582741 MAPEP= 1.76166  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 15, neighbors = 5)  
# RMSLEP = 0.05732295 RMSEP = 0.09926732 MAEP= 0.05657491 MAPEP= 1.373725  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 20, neighbors = 5)  
# RMSLEP = 0.0581008 RMSEP = 0.1000819 MAEP= 0.05658944 MAPEP= 2.448342  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 25, neighbors = 5)  
# RMSLEP = 0.05757621 RMSEP = 0.1004281 MAEP= 0.05637488 MAPEP= 2.383588  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 30, neighbors = 5)  
# RMSLEP = 0.05798935 RMSEP = 0.1000757 MAEP= 0.05589889 MAPEP= 2.954169  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 35, neighbors = 5)  
# RMSLEP = 0.05785085 RMSEP = 0.1006174 MAEP= 0.05638628 MAPEP= 1.453625  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 40, neighbors = 5)  
# RMSLEP = 0.05616211 RMSEP = 0.09804356 MAEP= 0.05557444 MAPEP= 1.534716  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 45, neighbors = 5)  
# RMSLEP = 0.05548687 RMSEP = 0.09659392 MAEP= 0.05467256 MAPEP= 2.219598  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 50, neighbors = 5)  
# RMSLEP = 0.05736875 RMSEP = 0.0986013 MAEP= 0.05579035 MAPEP= 2.864756  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 55, neighbors = 5)  
# RMSLEP = 0.05785987 RMSEP = 0.1010301 MAEP= 0.05638847 MAPEP= 2.534446  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 60, neighbors = 5)  
# RMSLEP = 0.05546727 RMSEP = 0.09642195 MAEP= 0.05523967 MAPEP= 2.568528  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 65, neighbors = 5)  
# RMSLEP = 0.05788636 RMSEP = 0.1001062 MAEP= 0.05632725 MAPEP= 1.455314  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 70, neighbors = 5)  
# RMSLEP = 0.05740603 RMSEP = 0.1005015 MAEP= 0.05679506 MAPEP= 2.144069  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 75, neighbors = 5)  
# RMSLEP = 0.0563129 RMSEP = 0.09802249 MAEP= 0.05560001 MAPEP= 1.443422  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 80, neighbors = 5)  
# RMSLEP = 0.05729028 RMSEP = 0.09899598 MAEP= 0.05590552 MAPEP= 2.46016  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 85, neighbors = 5)  
# RMSLEP = 0.05538537 RMSEP = 0.09533345 MAEP= 0.05419751 MAPEP= 1.591038  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 90, neighbors = 5)  
# RMSLEP = 0.05892291 RMSEP = 0.1018736 MAEP= 0.05710835 MAPEP= 2.26647  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 95, neighbors = 5)  
# RMSLEP = 0.05499022 RMSEP = 0.09668556 MAEP= 0.05557619 MAPEP= 2.159504  
# >   
# >   
# >   
# > cubist.cv(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 100, neighbors = 5)  
# RMSLEP = 0.05631326 RMSEP = 0.09833671 MAEP= 0.0559818 MAPEP= 2.106057

The best model form bandgap energy uses 85 boosts according to our work above, meaning our best model via treed is shown below.

cubist\_band\_final = cubist(form\_test\_x2,form\_test\_y2, p = 0.667, B = 20, committees = 85, neighbors = 5)  
  
#summary(cubist\_band\_final)

Evaluation on training data (1567 cases):

Average |error| 0.0594649  
Relative |error| 0.22  
Correlation coefficient 0.94  
  
  
Attribute usage:  
 Conds Model  
  
 83% 91% lattice\_vector\_3\_ang  
 74% 93% percent\_atom\_in  
 59% 82% percent\_atom\_al  
 28% 47% spacegroup  
 8% 67% lattice\_angle\_gamma\_degree  
 7% lattice\_angle\_alpha\_degree  
 6% 84% lattice\_vector\_2\_ang  
 5% 66% lattice\_vector\_1\_ang  
 4% 75% percent\_atom\_ga  
 2% lattice\_angle\_beta\_degree  
 31% number\_of\_total\_atoms  
 29% Spacegroup\_167  
 23% atoms\_40  
 11% atoms\_30  
 3% atoms\_60

#### Final Metrics

RMSLEP = 0.05538537 RMSEP = 0.09533345 MAEP= 0.05419751 MAPEP= 1.591038

R^2 = 0.8836

Much like our model for formation energy, our direct metrics based such as error have only improved slightly, and our correlation coefficient did not change much. But these improvements still are measurable and therefore should be taken into account.

## XGboost

Another tree-based method tried was XGboost, as we thought that it was possible that the regression-based approach of the treed model could potentially be holding it back given the difficult nature of our predictors.

### Model 1: formation energy

Like before, we will start with formation energy as our response.

require(xgboost)  
   
   
form\_test\_y\_list = as.list(form\_test\_y)  
   
form\_test\_x\_mat = as.matrix(form\_test\_x)  
   
basic\_boost\_form = xgboost(data = form\_test\_x\_mat, label = form\_test\_y\_list, nrounds = 30, nthread = 4, verbose = T, objective = "trilinear", eta = 0.9, max.depth = 6 )

While these initial results look good with just basic parameters, we still need to try and optimize the model by changing them. We will us K-folds cross-validation to get those results.

params = list(booster="gbtree",objective="reg:squaredlogerror",eta=0.9,gamma=0,  
max\_depth=5,min\_child\_weight=80,subsample=.66,colsample\_bytree=1,  
lambda=0.80,alpha=0.55)  
   
   
from\_xg\_cv\_results = xgb.cv(params = params, data = form\_test\_x\_mat, nrounds = 500, print\_every\_n = 10, nthreads = 4, nfold = 20, stratified = F, early\_stopping\_rounds = 30, maximize = 0, label = form\_test\_y\_list)

[1] train-rmsle:0.145127+0.002620 test-rmsle:0.144935+0.009187 Multiple eval metrics are present. Will use test\_rmsle for early stopping. Will train until test\_rmsle hasn’t improved in 30 rounds.

[11] train-rmsle:0.033935+0.000620 test-rmsle:0.035315+0.004063 [21] train-rmsle:0.030586+0.000534 test-rmsle:0.032836+0.004206 [31] train-rmsle:0.029062+0.000438 test-rmsle:0.031696+0.004325 [41] train-rmsle:0.028140+0.000436 test-rmsle:0.030978+0.004436 [51] train-rmsle:0.027481+0.000403 test-rmsle:0.030678+0.004610 [61] train-rmsle:0.027002+0.000363 test-rmsle:0.030468+0.004622 [71] train-rmsle:0.026560+0.000380 test-rmsle:0.030222+0.004620 [81] train-rmsle:0.026210+0.000363 test-rmsle:0.030137+0.004666 [91] train-rmsle:0.025918+0.000362 test-rmsle:0.029991+0.004657 [101] train-rmsle:0.025651+0.000355 test-rmsle:0.029905+0.004694 [111] train-rmsle:0.025428+0.000356 test-rmsle:0.029781+0.004759 [121] train-rmsle:0.025224+0.000343 test-rmsle:0.029680+0.004743 [131] train-rmsle:0.025017+0.000350 test-rmsle:0.029600+0.004719 [141] train-rmsle:0.024851+0.000344 test-rmsle:0.029529+0.004716 [151] train-rmsle:0.024693+0.000349 test-rmsle:0.029572+0.004690 [161] train-rmsle:0.024544+0.000351 test-rmsle:0.029538+0.004799 [171] train-rmsle:0.024389+0.000348 test-rmsle:0.029471+0.004873 [181] train-rmsle:0.024262+0.000356 test-rmsle:0.029406+0.004858 [191] train-rmsle:0.024139+0.000345 test-rmsle:0.029332+0.004830 [201] train-rmsle:0.024022+0.000341 test-rmsle:0.029341+0.004875 [211] train-rmsle:0.023910+0.000328 test-rmsle:0.029309+0.004926 [221] train-rmsle:0.023802+0.000323 test-rmsle:0.029305+0.004913 [231] train-rmsle:0.023687+0.000335 test-rmsle:0.029237+0.004862 [241] train-rmsle:0.023578+0.000320 test-rmsle:0.029244+0.004969 [251] train-rmsle:0.023477+0.000306 test-rmsle:0.029220+0.004972 [261] train-rmsle:0.023398+0.000303 test-rmsle:0.029251+0.004904 [271] train-rmsle:0.023301+0.000306 test-rmsle:0.029276+0.004927 Stopping. Best iteration: [246] train-rmsle:0.023528+0.000314 test-rmsle:0.029176+0.004953

#### Final Metrics

After attempting to modify parameters to beat the Treed regression model, the closest that could be gotten consistently was the one above with 256 rounds for and RMSLE of 0.029176.

Other Metrics:

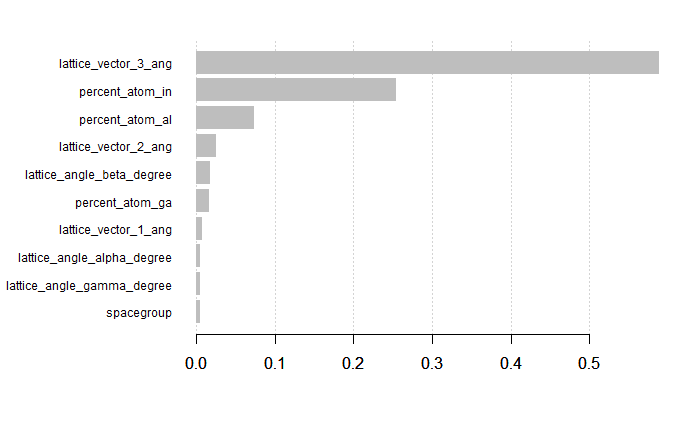
RMSE: 0.036691

MAE: 0.024237

Final Model:

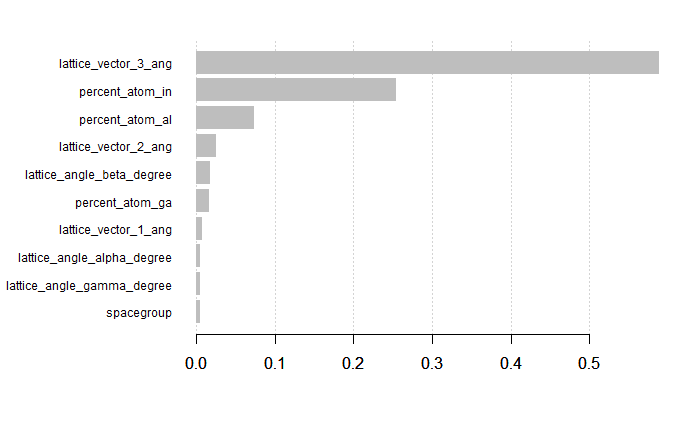
Looking at variable importance for top ten predictors:

mat = xgb.importance(feature\_names=colnames(form\_test\_x\_mat),model=XG\_form)  
xgb.plot.importance(importance\_matrix=mat[1:10])



For all predictors:

mat = xgb.importance(feature\_names=colnames(form\_test\_x\_mat),model=XG\_form)  
xgb.plot.importance(importance\_matrix=mat[1:10])



### Model 2: bandgap energy

Next, we will work similarly on bandgap.

form\_test\_y2\_list = as.list(form\_test\_y2)  
 form\_test\_x2\_mat = as.matrix(form\_test\_x2)  
   
basic\_boost\_form = xgboost(data = form\_test\_x\_mat, label = form\_test\_y\_list, nrounds = 30, nthread = 4, verbose = T, objective = "reg:linear", eta = 0.9, max.depth = 6 )

While even better than our initial run through from before, we still will go through the same process.

params = list(booster="gbtree",objective="reg:squaredlogerror",eta=0.4,gamma=0,  
 max\_depth=1,min\_child\_weight=15,subsample=.66,colsample\_bytree=0.75,  
lambda=0.5,alpha=0)  
   
from\_xg\_cv\_results2 = xgb.cv(params = params, data = form\_test\_x2\_mat, nrounds = 500, print\_every\_n = 10, nthreads = 7, nfold = 20, stratified = F, early\_stopping\_rounds = 30, maximize = 0, label = form\_test\_y2\_list)

[1] train-rmsle:0.267417+0.000682 test-rmsle:0.267147+0.012896 Multiple eval metrics are present. Will use test\_rmsle for early stopping. Will train until test\_rmsle hasn’t improved in 30 rounds.

[11] train-rmsle:0.075467+0.000847 test-rmsle:0.075456+0.012180 [21] train-rmsle:0.061307+0.000904 test-rmsle:0.061674+0.011922 [31] train-rmsle:0.057481+0.001031 test-rmsle:0.058701+0.011041 [41] train-rmsle:0.055696+0.000976 test-rmsle:0.057283+0.011041 [51] train-rmsle:0.054432+0.000859 test-rmsle:0.056481+0.011143 [61] train-rmsle:0.053474+0.000824 test-rmsle:0.055702+0.011280 [71] train-rmsle:0.052698+0.000756 test-rmsle:0.055063+0.011169 [81] train-rmsle:0.052084+0.000742 test-rmsle:0.054496+0.011229 [91] train-rmsle:0.051502+0.000704 test-rmsle:0.054383+0.011125 [101] train-rmsle:0.051016+0.000681 test-rmsle:0.054262+0.011255 [111] train-rmsle:0.050615+0.000703 test-rmsle:0.054092+0.011103 [121] train-rmsle:0.050237+0.000691 test-rmsle:0.053993+0.011396 [131] train-rmsle:0.049917+0.000685 test-rmsle:0.053875+0.011476 [141] train-rmsle:0.049621+0.000691 test-rmsle:0.053708+0.011328 [151] train-rmsle:0.049317+0.000675 test-rmsle:0.053769+0.011501 [161] train-rmsle:0.049029+0.000642 test-rmsle:0.053776+0.011665 [171] train-rmsle:0.048782+0.000629 test-rmsle:0.053591+0.011670 [181] train-rmsle:0.048542+0.000614 test-rmsle:0.053580+0.011512 [191] train-rmsle:0.048309+0.000612 test-rmsle:0.053621+0.011569 [201] train-rmsle:0.048122+0.000604 test-rmsle:0.053479+0.011579 [211] train-rmsle:0.047936+0.000596 test-rmsle:0.053238+0.011461 [221] train-rmsle:0.047741+0.000586 test-rmsle:0.053430+0.011315 [231] train-rmsle:0.047560+0.000564 test-rmsle:0.053464+0.011292 Stopping. Best iteration: [210] train-rmsle:0.047958+0.000605 test-rmsle:0.053097+0.011444

#### Final Metrics

After many iterative cycles of modification, the above model won out in cross-validation having gone 210 rounds for and RMSLE of 0.053097.

Other Metrics:

RMSE: 0.04795765

MAE: 0.062172

XG\_band = xgboost(data = form\_test\_x2\_mat, label = form\_test\_y2\_list, nrounds = 210, nthread = 7, verbose = T, objective = "reg:squaredlogerror", eta = 0.4, max.depth = 1, lambda = 0.5, alpha = 0, stratified = F, gamma = 0, colsample\_bytree=1, maximize = 0, min\_child\_weight=15)

Variable importance for this model is discussed in the section “Predicting Bandgap Energy: XGboost.”

## Shrinkage methods: Ridge, Lasso, Elastic Net

###### Monte Carlo Cross-Validation of Elastic Net, Ridge and Lasso Regression ####  
{glmnet.ssmc = function(X,y,p=.667,M=100,alpha=1,lambda=1) {  
 RMSEP = rep(0,M)  
 RMSLEP = rep(0,M)  
 MAEP = rep(0,M)  
 MAPEP = rep(0,M)  
 n = nrow(X)  
 for (i in 1:M) {  
 ss = floor(n\*p)  
 sam = sample(1:n,ss,replace=F)  
 fit = glmnet(X[sam,],y[sam],lambda=lambda,alpha=alpha)  
 ypred = predict(fit,newx=X[-sam,])  
 RMSEP[i] = sqrt(mean((y[-sam]-ypred)^2))  
 RMSLEP[i] = sqrt(mean((log(ypred +1) - log(y[-sam] +1))^2))   
 MAEP[i] = mean(abs(y[-sam]-ypred))  
 yp = ypred[y[-sam]!=0]  
 ya = y[-sam][y[-sam]!=0]  
 MAPEP[i]=mean(abs(yp-ya)/ya)  
 }  
 cat("RMSEP=",mean(RMSEP),  
 "RMSLEP=",mean(RMSLEP),  
 "MAEP=",mean(MAEP),  
 "MAPEP=",mean(MAPEP))  
 cv = return(data.frame(RMSEP=RMSEP, RMSLEP = RMSLEP, MAEP=MAEP,MAPEP=MAPEP))   
}  
}

###### Load and prepare data ######  
#Load data:  
{#setwd("~/OneDrive - MNSCU/myGithub/Supervised\_Learning/Multiple\_Linear\_Regression/MachineLearning-SupervisedLearning/Conductor Formulations")  
 #Train set  
 cond\_train = read.csv("Conductors (train).csv")  
 #Test set  
 cond\_test = read.csv("Conductors (test).csv")  
 cond\_test\_final = read.csv("cond\_test\_new\_variables.csv")  
 ### New training sets  
 cond\_train\_new\_vars = read.csv("cond\_train\_new\_variables.csv")  
 #No\_box\_cox  
 #angles measures are still in degrees; no transformaitons on predictors  
 cond\_train\_new\_vars\_no\_box\_cox = read.csv("cond\_train\_new\_variables\_no\_box\_cox.csv")  
   
 ####### New Variables #######  
   
 #Resposne vectors   
 y\_train\_form\_eng\_trans = cond\_train\_new\_vars$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
 y\_train\_bandgap\_eng\_trans = cond\_train\_new\_vars$bandgap\_energy\_ev\_log\_y\_plus\_1  
   
 #Resposne vectors for no box\_cox  
 #y\_train\_form\_eng\_trans = cond\_train\_new\_vars\_no\_box\_cox$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
 #y\_train\_bandgap\_eng\_trans = cond\_train\_new\_vars\_no\_box\_cox$bandgap\_energy\_ev\_log\_y\_plus\_1  
   
 #Df of all x's  
 x\_train = cond\_train\_new\_vars[,-c(1,25,26)]  
 #x\_train =cond\_train\_new\_vars\_no\_box\_cox[,-c(1,2,26,27)]  
}  
#Rearrange the data for two models: Model 1: formation energy as response; Model 2: bandgap energy as response  
{#detach("package:Ecfun", unload = TRUE)  
 cond\_train\_new\_vars %>%  
 select(25,2:24,27:29) -> train\_model1  
   
 cond\_train\_new\_vars%>%  
 select(26, 2:24,27:29) -> train\_model2  
}   
#Rearrange the data for two models for no box\_cox  
{  
 #detach("package:Ecfun", unload = TRUE)  
 cond\_train\_new\_vars\_no\_box\_cox %>%  
 select(26,3:25,28:30) -> train\_model1\_no\_box\_cox  
   
 cond\_train\_new\_vars\_no\_box\_cox%>%  
 select(27, 3:25,28:30) -> train\_model2\_no\_box\_cox  
}

### Model 1: formation energy

######## Model 1:response is formation energy ########  
####### Set up #########  
{  
 X = scale(model.matrix(formation\_energy\_ev\_natom\_log\_y\_plus\_1~., data = train\_model1)[,-1])  
 y = train\_model1$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
}

#### Elastic Net

Below we go through the process of finding the optimal alpha value for Elastic Net:

*###### For Elastic Net: finding lambda and fitting the model ########*  
*#Find best lambda*  
{   
 *# par(mfrow=c(4,4))*  
 *# #Change alpha*  
 *# alpha = 0.8*  
 *# cv.en = cv.glmnet(X,y,alpha=alpha)*  
 *# bestlam.en = cv.en$lambda.min*  
 *#plot(cv.en)*  
 *#title(main = paste("Best log(lambda) for Elastic Net", (round(log(bestlam.en),digits = 2))), sub = paste("Best lambda:", round(bestlam.en,5)))*  
}

*#Fit with optimal lambda*  
{  
 *# en.mod = glmnet(X,y,alpha=alpha, lambda =bestlam.en)*  
   
 *#y and yhat correlation*  
 *# en.cor = cor(y, predict(en.mod, newx = X))*  
   
 *#y and yhat correlation^2 = R^2*  
 *# en.rsqaured = cor(y, predict(en.mod, newx = X))^2*  
 *# plot(y,predict(en.mod,newx=X),xlab="Actual Age (y-values)",ylab="Predicted Age (yhat-values)", main = paste("Elastic Net Model:", "   ", "Correlation:", round(en.cor,4), "   ", "R^2:", round(en.rsqaured,4)),*  
 *#      sub = paste("alpha:", " ", alpha))*  
}

Now, we are going to fit Elastic Net models with alphas of 0.95, 0.5, 0.05.

###### Elastic net #########  
{alpha1 = 0.95  
cv.en = cv.glmnet(X,y,alpha=alpha1)  
bestlam.en = cv.en$lambda.min  
  
alpha2 = 0.5  
cv.en = cv.glmnet(X,y,alpha=alpha2)  
bestlam.en = cv.en$lambda.min  
  
alpha3 = 0.05  
cv.en = cv.glmnet(X,y,alpha=alpha3)  
bestlam.en = cv.en$lambda.min  
#plot(cv.en)  
#title(main = paste("Best log(lambda) for Elastic Net", (round(log(bestlam.en),digits = 2))), sub = paste("Best lambda:", round(bestlam.en,5)))  
en.results = glmnet.ssmc(X,y,p=.75, M=1000,alpha=alpha1,lambda=bestlam.en)  
en.results2 = glmnet.ssmc(X,y,p=.75, M=1000,alpha=alpha2,lambda=bestlam.en)  
en.results3 = glmnet.ssmc(X,y,p=.75, M=1000,alpha=alpha3,lambda=bestlam.en)  
}

## RMSEP= 0.05021873 RMSLEP= 0.04295012 MAEP= 0.0373822 MAPEP= 0.5040031RMSEP= 0.04990377 RMSLEP= 0.04264238 MAEP= 0.03713411 MAPEP= 0.495656RMSEP= 0.04993664 RMSLEP= 0.04267904 MAEP= 0.0371632 MAPEP= 0.5040992

#### Ridge

Now, we are fitting ridge regression model:

###### Ridge regression##########  
{cv.ridge= cv.glmnet(X, y, alpha = 0)  
bestlam.ridge = cv.ridge$lambda.min  
#plot(cv.ridge)  
#title(main = paste("Best log(lambda) for Ridge", (round(log(bestlam.ridge),digits = 2))), sub = paste("Best lambda:", round(bestlam.ridge)))  
ridge.results = glmnet.ssmc(X,y, p=.75, M=1000,alpha=0,lambda=bestlam.ridge)  
}

## RMSEP= 0.05190884 RMSLEP= 0.04445466 MAEP= 0.03903798 MAPEP= 0.5613401

#### Lasso

Lastly, we are fitting lasso regression model:

###### Lasso regression##############  
{cv.lasso = cv.glmnet(X, y, alpha = 1)  
bestlam.lasso = cv.lasso$lambda.min  
#plot(cv.lasso)  
#title(main = paste("Best log(lambda) for Lasso", (round(log(bestlam.lasso),digits = 4))), sub = paste("Best lambda:", round(bestlam.lasso, 4)))  
lasso.results = glmnet.ssmc(X,y,p=.75, M=1000,alpha=1,lambda=bestlam.lasso)  
}

## RMSEP= 0.04834215 RMSLEP= 0.04121558 MAEP= 0.03594626 MAPEP= 0.4895475

#### Final Metrics

Now, let’s compare the three different shrinkage methods we used:

###### Compare the three methods ###########  
{names = c("Ridge", "Lasso", "Elastic Net, alpha = 0.95", "Elastic Net, alpha = 0.5", "Elastic Net, alpha = 0.0.005")  
metrics =cbind(((names)), rbind(  
 do.call(cbind, lapply(ridge.results,mean)),   
 do.call(cbind, lapply(lasso.results, mean)),  
 do.call(cbind, lapply(en.results, mean)),  
 do.call(cbind, lapply(en.results2, mean)),  
 do.call(cbind, lapply(en.results3, mean))))  
  
df.metrics = as.data.frame(metrics)  
#write.csv(df.metrics, file = "Model\_1\_shrinkage\_regression\_results.csv", row.names = FALSE)  
df.metrics[order(df.metrics$RMSLEP),]  
}

## V1 RMSEP RMSLEP  
## 2 Lasso 0.048342152775784 0.0412155794223863  
## 4 Elastic Net, alpha = 0.5 0.0499037661228458 0.0426423813029742  
## 5 Elastic Net, alpha = 0.0.005 0.0499366411548562 0.0426790397055859  
## 3 Elastic Net, alpha = 0.95 0.0502187267468045 0.0429501187613037  
## 1 Ridge 0.0519088403813339 0.0444546596416351  
## MAEP MAPEP  
## 2 0.0359462598254077 0.489547495696644  
## 4 0.0371341060185126 0.495655979947912  
## 5 0.0371632000383486 0.50409922785998  
## 3 0.037382195110951 0.5040031189895  
## 1 0.0390379825762234 0.561340106011516

The winner amongst the three shrinkage methods for Model 1 (response: formation energy) is Lasso (RMSLEP=0.0412)

### Model 2: bandgap energy

###### Model 2:response is bandgap energy ########  
###### Set up #########  
{  
X = scale(model.matrix(bandgap\_energy\_ev\_log\_y\_plus\_1~., data = train\_model2)[,-1])  
y = train\_model2$bandgap\_energy\_ev\_log\_y\_plus\_1  
}

#### Elastic Net

Below we go through the process of finding the optimal alpha value for Elastic Net:

###### For Elastic Net: finding lambda and fitting the model ########  
#Find best lambda  
{   
 # par(mfrow=c(4,4))  
 #Change alpha  
 # alpha = 0.8  
 # cv.en = cv.glmnet(X,y,alpha=alpha)  
 # bestlam.en = cv.en$lambda.min  
 #plot(cv.en)  
 #title(main = paste("Best log(lambda) for Elastic Net", (round(log(bestlam.en),digits = 2))), sub = paste("Best lambda:", round(bestlam.en,5)))  
 #en.results = glmnet.ssmc(X,y,p=.75, M=1000,alpha=0.1,lambda=bestlam.en)  
}

## NULL

#Fit with optimal lambda  
{  
 # en.mod = glmnet(X,y,alpha=alpha, lambda =bestlam.en)  
 #   
 # #y and yhat correlation  
 # en.cor = cor(y, predict(en.mod, newx = X))  
 #   
 # #y and yhat correlation^2 = R^2  
 # en.rsqaured = cor(y, predict(en.mod, newx = X))^2  
 # plot(y,predict(en.mod,newx=X),xlab="Actual Age (y-values)",ylab="Predicted Age (yhat-values)", main = paste("Elastic Net Model:", " ", "Correlation:", round(en.cor,4), " ", "R^2:", round(en.rsqaured,4)),  
 # sub = paste("alpha:", " ", alpha))  
}

## NULL

Now, we are going to fit Elastic Net models with alphas of 0.5, 0.4, 0.8.

###### Elastic net #########  
{alpha1 = 0.8  
cv.en = cv.glmnet(X,y,alpha=alpha1)  
bestlam.en = cv.en$lambda.min  
  
alpha2 = 0.5  
cv.en = cv.glmnet(X,y,alpha=alpha2)  
bestlam.en = cv.en$lambda.min  
  
alpha3 = 0.4  
cv.en = cv.glmnet(X,y,alpha=alpha3)  
bestlam.en = cv.en$lambda.min  
#plot(cv.en)  
#title(main = paste("Best log(lambda) for Elastic Net", (round(log(bestlam.en),digits = 2))), sub = paste("Best lambda:", round(bestlam.en,5)))  
en.results = glmnet.ssmc(X,y,p=.75, M=1000,alpha=alpha1,lambda=bestlam.en)  
en.results2 = glmnet.ssmc(X,y,p=.75, M=1000,alpha=alpha2,lambda=bestlam.en)  
en.results3 = glmnet.ssmc(X,y,p=.75, M=1000,alpha=alpha3,lambda=bestlam.en)  
}

## RMSEP= 0.1011245 RMSLEP= 0.05745398 MAEP= 0.06824558 MAPEP= 1.635079RMSEP= 0.1004352 RMSLEP= 0.05699335 MAEP= 0.0679065 MAPEP= 1.67269RMSEP= 0.1010488 RMSLEP= 0.05741274 MAEP= 0.06816403 MAPEP= 1.748579

#### Ridge

Now, we are fitting ridge regression model:

###### Ridge regression##########  
{cv.ridge= cv.glmnet(X, y, alpha = 0)  
bestlam.ridge = cv.ridge$lambda.min  
#plot(cv.ridge)  
#title(main = paste("Best log(lambda) for Ridge", (round(log(bestlam.ridge),digits = 2))), sub = paste("Best lambda:", round(bestlam.ridge)))  
ridge.results = glmnet.ssmc(X,y, p=.75, M=1000,alpha=0,lambda=bestlam.ridge)  
}

## RMSEP= 0.1076936 RMSLEP= 0.06082086 MAEP= 0.07444468 MAPEP= 1.967728

#### Lasso

Lastly, we are fitting lasso regression model:

###### Lasso regression##############  
{cv.lasso = cv.glmnet(X, y, alpha = 1)  
bestlam.lasso = cv.lasso$lambda.min  
#plot(cv.lasso)  
#title(main = paste("Best log(lambda) for Lasso", (round(log(bestlam.lasso),digits = 4))), sub = paste("Best lambda:", round(bestlam.lasso, 4)))  
lasso.results = glmnet.ssmc(X,y,p=.75, M=1000,alpha=1,lambda=bestlam.lasso)  
}

## RMSEP= 0.1012675 RMSLEP= 0.05759733 MAEP= 0.06818431 MAPEP= 1.598254

#### Final Metrics

Now, let’s compare the three different shrinkage methods we used:

###### Compare the three methods ###########  
{names = c("Ridge", "Lasso", "Elastic Net, alpha = 0.8", "Elastic Net, alpha = 0.5", "Elastic Net, alpha = 0.4")  
metrics =cbind(((names)), rbind(  
 do.call(cbind, lapply(ridge.results,mean)),   
 do.call(cbind, lapply(lasso.results, mean)),  
 do.call(cbind, lapply(en.results, mean)),  
 do.call(cbind, lapply(en.results2, mean)),  
 do.call(cbind, lapply(en.results3, mean))))  
  
df.metrics = as.data.frame(metrics)  
#write.csv(df.metrics, file = "Model\_2\_shrinkage\_regression\_results.csv", row.names = FALSE)  
df.metrics[order(df.metrics$RMSLEP),]  
}

## V1 RMSEP RMSLEP  
## 4 Elastic Net, alpha = 0.5 0.100435161196804 0.0569933504773648  
## 5 Elastic Net, alpha = 0.4 0.1010488168194 0.057412744190101  
## 3 Elastic Net, alpha = 0.8 0.101124519881328 0.0574539794859449  
## 2 Lasso 0.101267501006356 0.0575973275643823  
## 1 Ridge 0.107693572620408 0.0608208591649753  
## MAEP MAPEP  
## 4 0.0679064987872233 1.67268965068565  
## 5 0.0681640334149643 1.74857899749765  
## 3 0.0682455833642216 1.63507938704905  
## 2 0.0681843134046107 1.59825433620615  
## 1 0.0744446806602136 1.96772807716676

The winner amongst the three shrinkage methods for Model 2 (reposne: bandgap energy) is, as well, Lasso (RMSLEP=0.0571)

The shrinkage methods did not perform better than MARS, Treed Regression or XGboosting for Model 1 (resposne=formation energy); for Model 2 (response=bandgap energy) Lasso and Elastic Net (alpha=0.5) did not do that bad. We will revisit all the models’ RMSLE metrics at the end.

We still must consider Random Forest.

## Random Forest

###### Functions #####  
#detach("package:Ecfun", unload = TRUE)  
#rf.sscv Monte-Carlo CV function  
{rf.sscv = function(fit,data,p=.667,B=100,mtry=fit$mtry,ntree=fit$ntree) {  
 RMSEP = rep(0,B)  
 RMSLEP = rep(0,B)  
 MAEP = rep(0,B)  
 MAPEP = rep(0,B)  
 y = fit$y  
 n = nrow(data)  
 ss <- floor(n\*p)  
 for (i in 1:B) {  
 sam = sample(1:n,ss,replace=F)  
 fit2 = randomForest(formula(fit),data=data[sam,],mtry=mtry,ntree=ntree)  
 ynew = predict(fit2,newdata=data[-sam,])  
 RMSEP[i] = sqrt(mean((y[-sam]-ynew)^2))  
 RMSLEP[i] = sqrt(mean((log(ynew +1) - log(y[-sam] +1))^2))  
 MAEP[i] = mean(abs(y[-sam]-ynew))  
 MAPEP[i] = mean((abs(y[-sam]-ynew)/y[-sam]))  
   
 }  
 RMSEP = mean(RMSEP)  
 RMSLEP = mean(RMSLEP)  
 MAEP = mean(MAEP)  
 MAPEP = mean(MAPEP)  
 cat("RMSEP\n")  
 cat("===============\n")  
 cat(RMSEP,"\n\n")  
 cat("RMSLEP\n")  
 cat("===============\n")  
 cat(RMSLEP,"\n\n")  
 cat("MAEP\n")  
 cat("===============\n")  
 cat(MAEP,"\n\n")  
 cat("MAPEP\n")  
 cat("===============\n")  
 cat(MAPEP,"\n\n")  
 temp = data.frame(RMSEP=RMSEP, RMSLEP=RMSLEP, MAEP=MAEP,MAPEP=MAPEP)  
 return(temp)  
}  
  
}  
#Predict Accuracy function  
{PredAcc = function(y, ypred){  
 RMSEP = sqrt(mean((y-ypred)^2))  
 RMSLEP = sqrt(mean((log(ypred +1) - log(y+1))^2))  
 MAE = mean(abs(y-ypred))  
 MAPE = mean(abs(y-ypred)/y)\*100  
 cat("RMSEP\n")  
 cat("================\n")  
 cat(RMSEP, "\n\n")  
 cat("RMSLEP\n")  
 cat("================\n")  
 cat(RMSLEP, "\n\n")  
 cat("MAE\n")  
 cat("================\n")  
 cat(MAE, "\n\n")  
 cat("MAPE\n")  
 cat("================\n")  
 cat(MAPE, "\n\n")  
 return(data.frame(RMSEP = RMSEP, RMSLEP = RMSLEP, MAE = MAE, MAPE = MAPE))  
}  
}  
#Plotting variable importance  
{rfimp = function(rffit, horiz=T) {barplot(sort(rffit$importance[,1]),horiz=horiz,  
 xlab="Mean Decreasing in Accuracy",main="Variable Importance")  
}  
}

The data was loaded and prepared in the Shirnkage Methods section. We are going to continue working with train\_model1 and train\_model2 data frames.

### Model 1: formation energy

###### Model 1 #####  
{y = train\_model1$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
X = train\_model1  
cond.rf = randomForest(y~.,data=X,importance=F)  
}

We are splitting our training set into training and validation sets:

####### Create validation set #######  
#trainmodel1 is X  
#response is y  
{n = nrow(X)  
set.seed(2019)  
sam = sample(1:n,size=floor(n\*.6667),replace=F)  
X.train = X[sam,] #form training dataset  
X.valid = X[-sam,] #form validation dataset  
y.train = X.train$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
y.valid = X.valid$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
  
#cond.rf = randomForest(y.train~.,data=X.train)  
#plot(cond.rf)  
#abline(h=0.000128,col="red",lwd=2)  
}

Now we are going to go through the process of finding optimal number of trees (ntree) and number of variables to randomly pick at each stage (mtry)

#1st: choose optimal mtry's  
  
# results = rf.sscv(cond.rf,X,mtry=3)  
# results = rf.sscv(cond.rf,X,mtry=4)  
# results = rf.sscv(cond.rf,X,mtry=5)  
# results = rf.sscv(cond.rf,X,mtry=6)  
  
#Compare mtry's with errorest function  
{#m=4  
# m=4  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
# #m=5  
# m=5  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
# #m=6  
# m=6  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
#   
# #m=7  
# m=7  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
}

## NULL

#2nd: Increasing the number of trees beyonf the default ntree=500  
#BEST = based on best mtry from above  
#BEST = 5  
  
# results = rf.sscv(cond.rf,X,mtry=BEST,ntree=500,B=250)  
# results = rf.sscv(cond.rf,X,mtry=BEST,ntree=750,B=250)  
# results = rf.sscv(cond.rf,X,mtry=BEST,ntree=1000,B=250)

###### Compare best mtry and ntree combos #######   
#ntree of 50, 80, 100, 350  
  
# cond.rf = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,ntree=50, mtry=9)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,ntree=80)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,ntree=100)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,ntree=350)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,ntree=150, mtry =12)  
# results = rf.sscv(cond.rf,X.train)  
  
#From best to least: ntree = 350, then 80, then 50, then 100  
  
#mtry  
  
# cond.rf = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,mtry = 6,ntree=1000)  
# results = rf.sscv(cond.rf,X.train)  
# cond.final = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,mtry=5,ntree=750)  
# results = rf.sscv(cond.rf,X.train)  
  
#mtry=5, ntree=350  
#mtry=5, ntree=750  
#mtry=6, ntree=1000  
#mtry=6, ntree=350

Our thorough process of exploring different number of bootstrap trees and number of variables to randomly pick at each stage gave us the following optimal hyperparameters: Best combo is: mtry=5 and ntree=350

#### Final Metrics

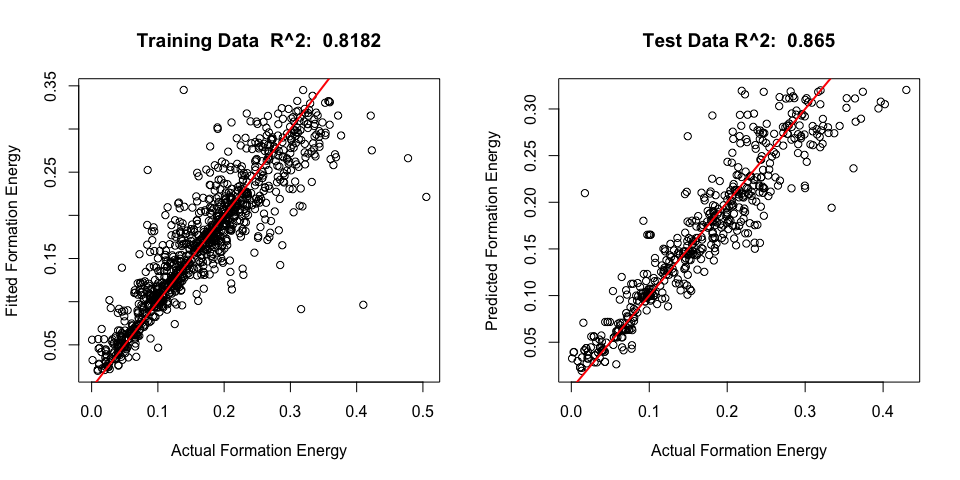
This gives us our final model:

####### Final randomForest model #######  
{cond.final = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,mtry=5,ntree=350, importance = T)  
ypred = predict(cond.final,newdata=X.valid)  
prediciton.rf = PredAcc(y.valid,ypred)  
}

## RMSEP  
## ================  
## 0.03233925   
##   
## RMSLEP  
## ================  
## 0.02672702   
##   
## MAE  
## ================  
## 0.02222394   
##   
## MAPE  
## ================  
## 29.88186

Let’s see how our final random forest model predicts. We are plotting the actual formation energy values vs. predicted for both the training set data (figure on the left) and validation set data (figure on the right)

par(mfrow=c(1,2))  
plot(y.train,predict(cond.final),xlab="Actual Formation Energy",ylab="Fitted Formation Energy",  
 main= paste("Training Data", " R^2: ",   
 round((cor(y.train,   
 predict(cond.final))^2),4)))  
abline(0,1,col="red",lwd=2)  
  
plot(y.valid, ypred, xlab="Actual Formation Energy",ylab="Predicted Formation Energy",  
 main=paste("Test Data", "R^2: ",  
 round((cor(y.valid,   
 ypred)^2),4)))  
abline(0,1,col="red",lwd=2)



#par(mfrow=c(1,1))

Our final random forest model predicts with accuracy of RMSLEP=0.0268 and gives us 86.35% rsquared value. It is by far the best model for predicting formation energy.

##### Predicting on Test Set

Prediction on Test Set

randomforest.final = cond.final

cond\_test\_final %>%  
 select (-c(bandgap\_energy\_ev\_log\_y\_plus\_1,formation\_energy\_ev\_natom\_log\_y\_plus\_1,ID)) -> Modified\_Xs

Formation\_Predictions = predict(randomforest.final, as.matrix(Modified\_Xs))  
write.csv(Formation\_Predictions, file = "formation\_predictions\_RandomForest.csv")

### Model 2: bandgap energy

############ Model 2 #################  
{y = train\_model2\_no\_box\_cox$bandgap\_energy\_ev\_log\_y\_plus\_1  
X = train\_model2\_no\_box\_cox  
cond.rf = randomForest(y~.,data=X,importance=T)  
}

We are splitting our training set into training and validation sets:

####### Create validation set #######  
#trainmodel2 is X  
#response is y  
{n = nrow(X)  
set.seed(2019)  
sam = sample(1:n,size=floor(n\*.6667),replace=F)  
X.train = X[sam,] #form training dataset  
X.valid = X[-sam,] #form validation dataset  
y.train = X.train$bandgap\_energy\_ev\_log\_y\_plus\_1  
y.valid = X.valid$bandgap\_energy\_ev\_log\_y\_plus\_1  
  
# par(mfrow=c(1,1))  
# cond.rf = randomForest(y.train~.,data=X.train)  
# plot(cond.rf)  
# abline(h=0.000128,col="red",lwd=2)  
}

Now we are going to go through the process of finding optimal number of trees (ntree) and number of variables to randomly pick at each stage (mtry)

#1st: choose optimal mtry's  
  
# results = rf.sscv(cond.rf,X,mtry=3)  
# results = rf.sscv(cond.rf,X,mtry=4)  
# results = rf.sscv(cond.rf,X,mtry=5)  
# results = rf.sscv(cond.rf,X,mtry=6)  
  
{#m=4  
# m=4  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
# #m=5  
# m=5  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
# #m=6  
# m=6  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
#   
# #m=7  
# m=7  
# myforest = function(formula,data){randomForest(formula,data,mtry=m)}  
# error.RF = numeric(10)  
# for (i in 1:10) error.RF[i] = errorest(y~.,  
# data=X,model=myforest)$error  
# mean(error.RF)  
# summary(error.RF)  
}

## NULL

#2nd: Increasing the number of trees beyonf the default ntree=500  
#BEST = based on best mtry from above  
#BEST = 4  
  
# results = rf.sscv(cond.rf,X,mtry=BEST,ntree=500,B=250)  
# results = rf.sscv(cond.rf,X,mtry=BEST,ntree=750,B=250)  
# results = rf.sscv(cond.rf,X,mtry=BEST,ntree=1000,B=250)  
  
#mtry=4,ntree=1000

###### Compare best mtry and ntree combos #######   
#ntree of 50, 80, 100, 350  
  
# cond.rf = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,ntree=40)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,ntree=80)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,ntree=750)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,ntree=350)  
# results = rf.sscv(cond.rf,X.train)  
# cond.rf = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,ntree=150, mtry = 6)  
# results = rf.sscv(cond.rf,X.train)  
  
# #mtry  
  
# cond.rf = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,mtry = 4,ntree=1000)  
# results = rf.sscv(cond.rf,X.train)  
# cond.final = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,mtry=5,ntree=750)  
# results = rf.sscv(cond.rf,X.train)  
  
#mtry=4, ntree=1000  
#mtry=5, ntree=750  
  
#Best combo is: mtry=4 and ntree=1000

Our thorough process of exploring different number of bootstrap trees and number of variables to randomly pick at each stage gave us the following optimal hyperparameters: Best combo is: mtry=4 and ntree=1000

#### Final Metrics

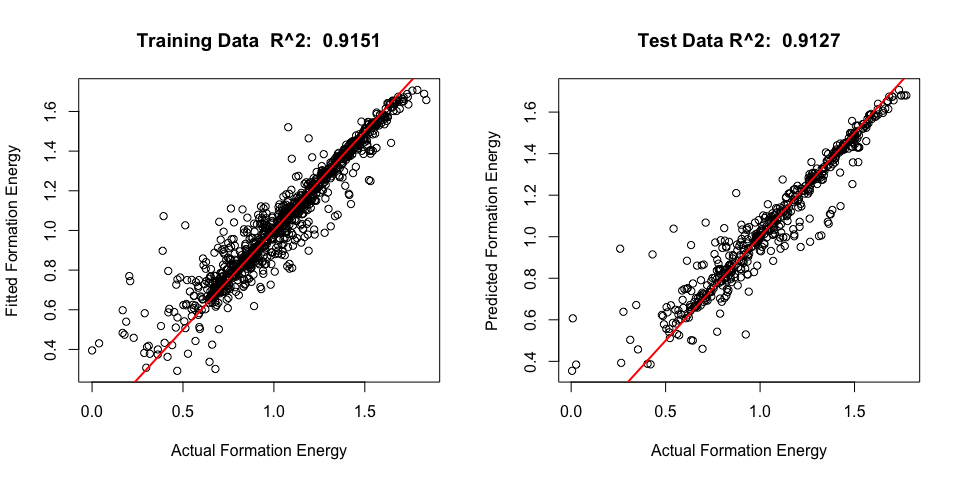
This gives us our final model:

####### Final randomForest model #######  
{cond.final = randomForest(bandgap\_energy\_ev\_log\_y\_plus\_1~.,data=X.train,mtry=4,ntree=1000, importance = T)  
ypred = predict(cond.final,newdata=X.valid)  
prediction = PredAcc(y.valid, ypred)  
}

## RMSEP  
## ================  
## 0.1004869   
##   
## RMSLEP  
## ================  
## 0.05864491   
##   
## MAE  
## ================  
## 0.05741837   
##   
## MAPE  
## ================  
## 36.56483

Let’s see how our final random forest model predicts. We are plotting the actual formation energy values vs. predicted for both the training set data (figure on the left) and validation set data (figure on the right)

{par(mfrow=c(1,2))  
plot(y.train,predict(cond.final),xlab="Actual Formation Energy",ylab="Fitted Formation Energy",  
 main= paste("Training Data", " R^2: ",   
 round((cor(y.train,   
 predict(cond.final))^2),4)))  
abline(0,1,col="red",lwd=2)  
  
plot(y.valid, ypred, xlab="Actual Formation Energy",ylab="Predicted Formation Energy",  
 main=paste("Test Data", "R^2: ",  
 round((cor(y.valid,   
 ypred)^2),4)))  
abline(0,1,col="red",lwd=2)  
#par(mfrow=c(1,1))  
}



Our final random forest model predicts with accuracy of RMSLEP=0.05864 and gives us 91.27% rsquared value. It is not a good model to predict bandgap energy. Lasso, Elastic Net (alpha=0.5), Treed Regression, MARS, and XGboosting are performing better than Random Forest for Model 2.

Our last statistical model to consider are Neural Networks

## Neural Networks

Functions:

###### Functions #####  
#detach("package:Ecfun", unload = TRUE)  
#rf.sscv Monte-Carlo CV function  
{rf.sscv = function(fit,data,p=.667,B=100,mtry=fit$mtry,ntree=fit$ntree) {  
 RMSEP = rep(0,B)  
 RMSLEP = rep(0,B)  
 MAEP = rep(0,B)  
 MAPEP = rep(0,B)  
 y = fit$y  
 n = nrow(data)  
 ss <- floor(n\*p)  
 for (i in 1:B) {  
 sam = sample(1:n,ss,replace=F)  
 fit2 = randomForest(formula(fit),data=data[sam,],mtry=mtry,ntree=ntree)  
 ynew = predict(fit2,newdata=data[-sam,])  
 RMSEP[i] = sqrt(mean((y[-sam]-ynew)^2))  
 RMSLEP[i] = sqrt(mean((log(ynew +1) - log(y[-sam] +1))^2))  
 MAEP[i] = mean(abs(y[-sam]-ynew))  
 MAPEP[i] = mean((abs(y[-sam]-ynew)/y[-sam]))  
   
 }  
 RMSEP = mean(RMSEP)  
 RMSLEP = mean(RMSLEP)  
 MAEP = mean(MAEP)  
 MAPEP = mean(MAPEP)  
 cat("RMSEP\n")  
 cat("===============\n")  
 cat(RMSEP,"\n\n")  
 cat("RMSLEP\n")  
 cat("===============\n")  
 cat(RMSLEP,"\n\n")  
 cat("MAEP\n")  
 cat("===============\n")  
 cat(MAEP,"\n\n")  
 cat("MAPEP\n")  
 cat("===============\n")  
 cat(MAPEP,"\n\n")  
 temp = data.frame(RMSEP=RMSEP, RMSLEP=RMSLEP, MAEP=MAEP,MAPEP=MAPEP)  
 return(temp)  
}  
  
}  
#Predict Accuracy function  
{PredAcc = function(y, ypred){  
 RMSEP = sqrt(mean((y-ypred)^2))  
 RMSLEP = sqrt(mean((log(ypred +1) - log(y+1))^2))  
 MAE = mean(abs(y-ypred))  
 MAPE = mean(abs(y-ypred)/y)\*100  
 cat("RMSEP\n")  
 cat("================\n")  
 cat(RMSEP, "\n\n")  
 cat("RMSLEP\n")  
 cat("================\n")  
 cat(RMSLEP, "\n\n")  
 cat("MAE\n")  
 cat("================\n")  
 cat(MAE, "\n\n")  
 cat("MAPE\n")  
 cat("================\n")  
 cat(MAPE, "\n\n")  
 return(data.frame(RMSEP = RMSEP, RMSLEP = RMSLEP, MAE = MAE, MAPE = MAPE))  
}  
}  
  
#Plotting variable importance  
{rfimp = function(rffit, horiz=T) {barplot(sort(rffit$importance[,1]),horiz=horiz,  
 xlab="Mean Decreasing in Accuracy",main="Variable Importance")  
}  
}

The data was loaded and prepared in the Shirnkage Methods section. We are going to continue working with train\_model1 and train\_model2 data frames.

### Model 1: formation energy

plot(X$formation\_energy\_ev\_natom\_log\_y\_plus\_1, fitted(mod1))  
abline(0,1, lwd=2, col="blue")

We are now splitting the training set into the training and validation sets

###### Create train, dev, and test sets #######  
  
{  
n = nrow(X)  
set.seed(1111)  
sam = sample(1:n,size=floor(n\*.6667),replace=F)  
X.train = X[sam,] #form training dataset  
X.valid = X[-sam,] #form validation dataset  
y.train = X.train$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
y.valid = X.valid$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
}

Fit three different Neural Net models with different hyperparameters:

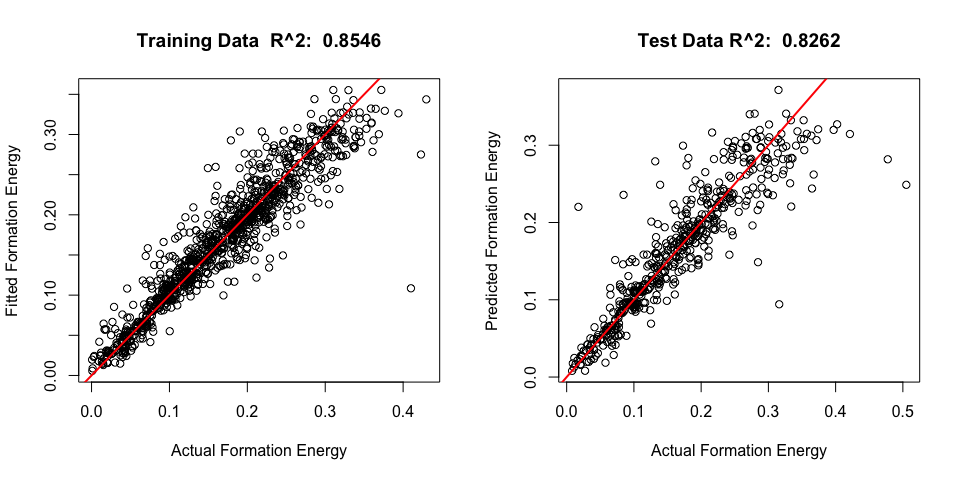
Assign the Neural Net model into ‘nn’ below to check how it predicts and fits our data:

#best nn2 so far  
nn = nn2  
  
{ypred = predict(nn, newdata = X.valid)  
prediction = PredAcc(y.valid, ypred)  
}

## RMSEP  
## ================  
## 0.03736334   
##   
## RMSLEP  
## ================  
## 0.03040962   
##   
## MAE  
## ================  
## 0.02331236   
##   
## MAPE  
## ================  
## 17.51146

#### Final Metrics

{par(mfrow=c(1,2))  
plot(y.train,predict(nn),xlab="Actual Formation Energy",ylab="Fitted Formation Energy",  
 main= paste("Training Data", " R^2: ",   
 round((cor(y.train,   
 predict(nn1))^2),4)))  
abline(0,1,col="red",lwd=2)  
  
plot(y.valid, ypred, xlab="Actual Formation Energy",ylab="Predicted Formation Energy",  
 main=paste("Test Data", "R^2: ",  
 round((cor(y.valid,   
 ypred)^2),4)))  
abline(0,1,col="red",lwd=2)  
par(mfrow=c(1,1))  
}



Our final Neural Network model predicts with accuracy of RMSLEP=0.03075 and gives us 82.39% rsquared value. It is not a good model to predict formation energy.

### Model 2: bandgap energy

plot(X$bandgap\_energy\_ev\_log\_y\_plus\_1, fitted(mod1))  
abline(0,1, lwd=2, col="blue")

We are now splitting the training set into the training and validation sets

###### Create train, dev, and test sets #######  
  
{n = nrow(X)  
set.seed(1111)  
sam = sample(1:n,size=floor(n\*.6667),replace=F)  
X.train = X[sam,] #form training dataset  
X.valid = X[-sam,] #form validation dataset  
y.train = X.train$bandgap\_energy\_ev\_log\_y\_plus\_1  
y.valid = X.valid$bandgap\_energy\_ev\_log\_y\_plus\_1  
  
#cond.rf = randomForest(y.train~.,data=X.train)  
#plot(cond.rf)  
#abline(h=0.000128,col="red",lwd=2)  
}

Fit three different Neural Net models with different hyperparameters:

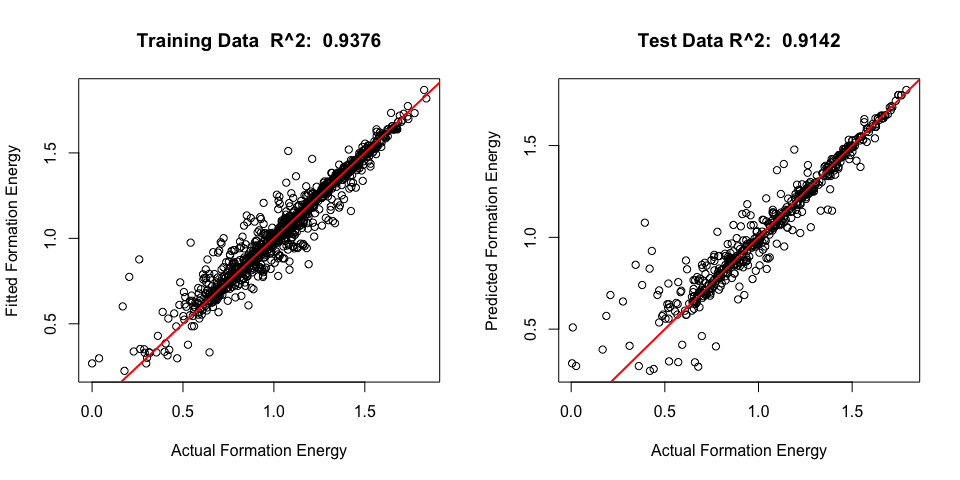
Assign the Neural Net model into ‘nn’ below to check how it predicts and fits our data:

#best nn3 so far  
nn = nn3  
  
{ypred = predict(nn, newdata = X.valid)  
prediction = PredAcc(y.valid, ypred)  
}

## RMSEP  
## ================  
## 0.1011948   
##   
## RMSLEP  
## ================  
## 0.06182588   
##   
## MAE  
## ================  
## 0.05602597   
##   
## MAPE  
## ================  
## 33.24002

#### Final Metrics

par(mfrow=c(1,2))  
 plot(y.train,predict(nn1),xlab="Actual Formation Energy",ylab="Fitted Formation Energy",  
 main= paste("Training Data", " R^2: ",   
 round((cor(y.train,   
 predict(nn1))^2),4)))  
 abline(0,1,col="red",lwd=2)  
   
 plot(y.valid, ypred, xlab="Actual Formation Energy",ylab="Predicted Formation Energy",  
 main=paste("Test Data", "R^2: ",  
 round((cor(y.valid,   
 ypred)^2),4)))  
 abline(0,1,col="red",lwd=2)  
 par(mfrow=c(1,1))



Our final Neural Network model predicts with accuracy of RMSLEP=0.0618 and gives us 91.42% rsquared value. It is not a good model to predict bandgap energy.

## Keras Regressor: neural network

####### KERAS ########  
require(keras)

## Loading required package: keras

{y = train\_model1$formation\_energy\_ev\_natom\_log\_y\_plus\_1  
train\_data = as.matrix(scale(train\_model1))  
}

build\_model <- function() {  
   
 model <- keras\_model\_sequential() %>%  
 layer\_dense(units = 64, activation = "relu",  
 input\_shape = dim(train\_data)[2]) %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 32, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 24, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 1, activation = "linear")  
   
 model %>% compile(  
 loss = "mse",  
 optimizer = optimizer\_rmsprop(lr = 0.002),  
 metrics = list("mean\_squared\_logarithmic\_error")  
 )  
   
 return(model)  
}

model <- build\_model()  
model %>% summary()

## Model: "sequential"  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## Layer (type) Output Shape Param #   
## ===========================================================================  
## dense (Dense) (None, 64) 1792   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dropout (Dropout) (None, 64) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dense\_1 (Dense) (None, 32) 2080   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dropout\_1 (Dropout) (None, 32) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dense\_2 (Dense) (None, 24) 792   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dropout\_2 (Dropout) (None, 24) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dense\_3 (Dense) (None, 1) 25   
## ===========================================================================  
## Total params: 4,689  
## Trainable params: 4,689  
## Non-trainable params: 0  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Display training progress by printing a single dot for each completed epoch.  
print\_dot\_callback <- callback\_lambda(  
 on\_epoch\_end = function(epoch, logs) {  
 if (epoch %% 80 == 0) cat("\n")  
 cat(".")  
 }  
)   
epochs <- 100

# Fit the model and store training stats  
history <- model %>% fit(  
 train\_data,  
 y,  
 epochs = epochs,  
 validation\_split = 0.2,  
 verbose = 0,  
 callbacks = list(print\_dot\_callback)  
)

##   
## ................................................................................  
## ....................

sqrt(mean(history$metrics$mean\_squared\_logarithmic\_error))

## [1] 0.06884552

Not performing well…

## Final Models:

Our Random Forest model performed the best in predicting the formation of energy, whereas the XGboost Model performed the best in predicting the bandgap energy

Below, is the comparison of all the models we ran, in predicting formation energy and bandgap energy:

#Sorted for RMSLEP for formation energy  
models.all = read.csv("model\_comparison.csv")  
models.all

## Model RMSLEP.for.formation RMSLEP.for.bandgap  
## 1 Random Forest 0.0267 0.0586  
## 2 Treed Regression 0.0272 0.0554  
## 3 Xgboosting 0.0292 0.0531  
## 4 Neural Net 0.0304 0.0617  
## 5 MARS 0.0304 0.0549  
## 6 Lasso 0.0411 0.0574  
## 7 Elastic Net, alpha = 0.5 0.0427 0.0573  
## 8 Ridge 0.0446 0.0604

#Sorted for RMSLEP for bandgap energy  
models.all[order(models.all$RMSLEP.for.bandgap),]

## Model RMSLEP.for.formation RMSLEP.for.bandgap  
## 3 Xgboosting 0.0292 0.0531  
## 5 MARS 0.0304 0.0549  
## 2 Treed Regression 0.0272 0.0554  
## 7 Elastic Net, alpha = 0.5 0.0427 0.0573  
## 6 Lasso 0.0411 0.0574  
## 1 Random Forest 0.0267 0.0586  
## 8 Ridge 0.0446 0.0604  
## 4 Neural Net 0.0304 0.0617

### 

### Predicting Formation of Energy: Random Forest

Revisiting the best model and going over the variable importance:

Rename the predictors’ names for plotmo and refit the randomforest model:

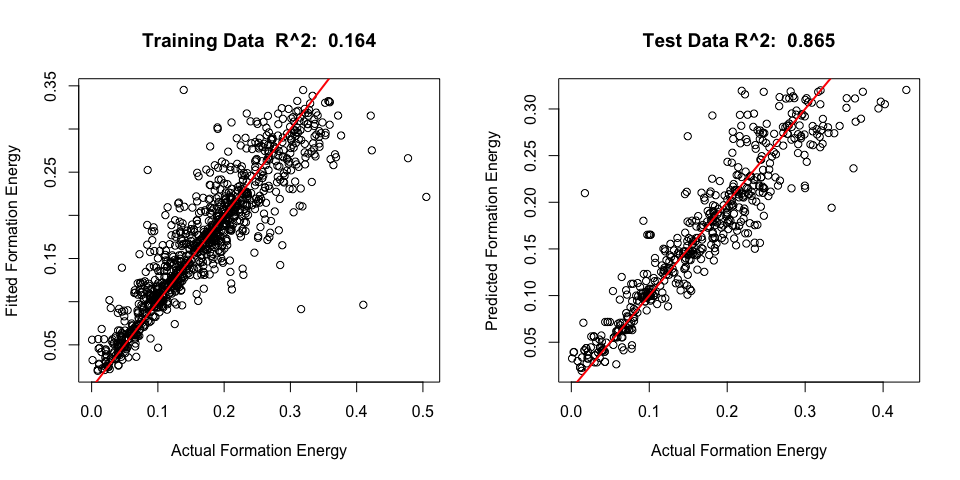
train\_model1%>%  
 rename(tot\_atoms = number\_of\_total\_atoms,  
 atm10 = atoms\_10,  
 atm20 = atoms\_20,  
 atm30 = atoms\_30,  
 atm40 = atoms\_40,  
 atm60 = atoms\_60,  
 atm80 = atoms\_80,  
 spgr = spacegroup,  
 spgr12 = Spacegroup\_12,  
 spgr33 = Spacegroup\_33,  
 spgr167 = Spacegroup\_167,  
 spgr194 = Spacegroup\_194,  
 spgr206 = Spacegroup\_206,  
 spgr227 = Spacegroup\_227,  
 pctatmAL = percent\_atom\_al,  
 pctatmGA = percent\_atom\_ga,  
 pctatmIN = percent\_atom\_in,  
 ltc1ang = lattice\_vector\_1\_ang,  
 ltc2ang = lattice\_vector\_2\_ang,  
 ltc3ang = lattice\_vector\_3\_ang,  
 ltcalph = lattice\_angle\_alpha\_degree,  
 ltcbeta = lattice\_angle\_beta\_degree,  
 ltcgam = lattice\_angle\_gamma\_degree,  
 alum = Aluminum,  
 gall = Gallium,  
 ind = Indium) -> train\_model1\_renamed

randomforest.final = randomForest(formation\_energy\_ev\_natom\_log\_y\_plus\_1~.,data=X.train,mtry=5,ntree=350, importance = T)

ypred = predict(randomforest.final,newdata=X.valid)  
prediciton.rf = PredAcc(y.valid,ypred)

## RMSEP  
## ================  
## 0.03233925   
##   
## RMSLEP  
## ================  
## 0.02672702   
##   
## MAE  
## ================  
## 0.02222394   
##   
## MAPE  
## ================  
## 29.88186

par(mfrow=c(1,2))  
plot(y.train,predict(randomforest.final),xlab="Actual Formation Energy",ylab="Fitted Formation Energy",  
 main= paste("Training Data", " R^2: ",   
 round((cor(y.train,   
 predict(cond.final))^2),4)))  
abline(0,1,col="red",lwd=2)  
  
plot(y.valid, ypred, xlab="Actual Formation Energy",ylab="Predicted Formation Energy",  
 main=paste("Test Data", "R^2: ",  
 round((cor(y.valid,   
 ypred)^2),4)))  
abline(0,1,col="red",lwd=2)



rfimp = function(rffit, horiz=T) {  
 barplot(sort(rffit$importance[,1]),horiz=horiz,  
 sub="Mean Decreasing in Accuracy",main="Variable Importance",   
 cex.names = 0.6,  
 las = 2,  
 xpd = FALSE,  
 col="#69b3a2",  
 )  
}

par(mfrow=c(1,1))  
par(mar=c(8,8,8,6))  
rfimp(randomforest.final, horiz = T)

#### Final Results

Our final random forest model predicts with accuracy of RMSLE=0.0267 and gives us 86.5% R^2 value. It is the best model for predicting formation energy. The optimal combination of hyperparameters of our random forest model is ntree=350 and mtry=5. After Random Forest, Treed Regression and XGboosting performed the best. All other models had a RMSLE > 0.03.

#Comparing to other models  
models.all[,c(1,2)]

## Model RMSLEP.for.formation  
## 1 Random Forest 0.0267  
## 2 Treed Regression 0.0272  
## 3 Xgboosting 0.0292  
## 4 Neural Net 0.0304  
## 5 MARS 0.0304  
## 6 Lasso 0.0411  
## 7 Elastic Net, alpha = 0.5 0.0427  
## 8 Ridge 0.0446

cond\_test\_final %>%  
 select (-c(bandgap\_energy\_ev\_log\_y\_plus\_1,formation\_energy\_ev\_natom\_log\_y\_plus\_1,ID)) -> Modified\_Xs

#Formation\_Predictions = predict(randomforest.final, as.matrix(Modified\_Xs))  
#write.csv(Formation\_Predictions, file = "formation\_predictions\_RandomForest.csv")

#### Variable Importance

A screenshot of a cell phone

Description automatically generated

For our randomforest, many different variables were used to great effect to aide in our predictions. The amount of Indium and lattice 3 angle were together the strongest, with vector alpha, the numerically scale spacegroups, lattice angle 2 and the make-up of the other two elements following suit as far as high scoring contributions. Although within the context of this model in specific, most variables were contributing something worthwhile, with only the atom categories (barring 40) and element indicators falling short. The wide variety makes sense given the nature of the method, but it does lead to an analysis of importance alone being perhaps less useful. Looking at effects on response will likely prove more fruitful.

**par**(mfrow=**c**(1,1))

**require**(plotmo)  
plotmo(randomforest.final, caption = NULL)

*## plotmo grid: tot\_atoms atm10 atm20 atm30 atm40 atm60 atm80 spgr spgr12*  
*## 692.5198 0 0 0 0 0 1 194 0*  
*## spgr33 spgr167 spgr194 spgr206 spgr227 pctatmAL pctatmGA pctatmIN*  
*## 0 0 0 0 0 -0.484246 -0.8850132 -1.033015*  
*## ltc1ang ltc2ang ltc3ang ltcalph ltcbeta ltcgam alum gall ind*  
*## 3.028466 3.802559 1.756726 0.08647367 0.0644573 0.7730203 0 0 0*

A picture containing table, sitting, computer, white

Description automatically generated

If one wanted to maximize formation energy, based on this model we could give them some directions to go off. First, stick to spacegroup 227 if you can, as it tends to produce the highest compared to other groups by notable margin. Try to strike a balance between Iniduim and Aluminum for the core molecular structure, both contribute the most when kept at moderate levels. Though if you do add indium, make sure you commit to it, as small amounts off it tend to result in much lower scores. Gallium can be strong in smaller amounts, but do not make a major stakeholder in your formulation. Lattice angle alpha in conjunction with a small focus on lattice vector 3 should also produce better results. Finally, do not put too much stress on lattice vector 2, as in its higher stages it tends to drop off in returns on the response.

### Predicting Bandgap Energy: XGboost

cond\_test\_ready %>%  
select (-c(bandgap\_energy\_ev\_log\_y\_plus\_1,formation\_energy\_ev\_natom\_log\_y\_plus\_1,ID)) -> Modified\_Xs  
   
Bandgap\_Predictions = predict(XG\_band, as.matrix(Modified\_Xs))  
  
write.csv(Bandgap\_Predictions, file = "bandgap\_predictions\_XGboost.csv")

As a reminder, using this model:

# XG\_band = xgboost(data = form\_test\_x2\_mat, label = form\_test\_y2\_list, nrounds = 210, nthread = 7, verbose = T, objective = "reg:squaredlogerror", eta = 0.4, max.depth = 1, lambda = 0.5, alpha = 0, stratified = F, gamma = 0, colsample\_bytree=1, maximize = 0, min\_child\_weight=15)

#### Final Results

#Comparing to other models  
models.all[order(models.all$RMSLEP.for.bandgap),c(1,3)]

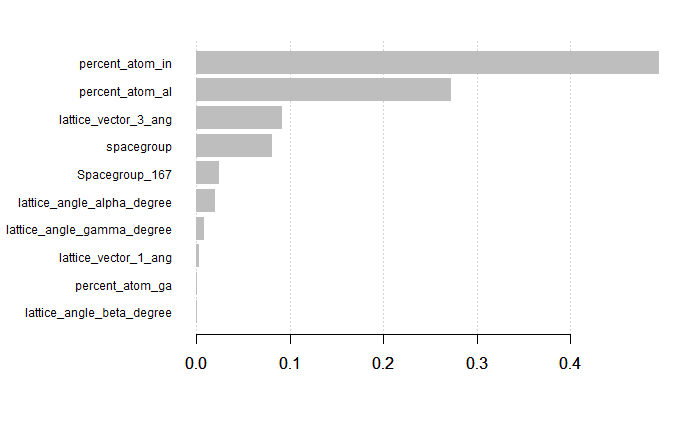
## Model RMSLEP.for.bandgap  
## 3 Xgboosting 0.0531  
## 5 MARS 0.0549  
## 2 Treed Regression 0.0554  
## 7 Elastic Net, alpha = 0.5 0.0573  
## 6 Lasso 0.0574  
## 1 Random Forest 0.0586  
## 8 Ridge 0.0604  
## 4 Neural Net 0.0617

For predicting bandgap energy using RMSLE as our metric, our XGBoost won out over every other model we tried (0.029176). Digging deeper into it, we found that the algorithm performed best going around 210 rounds with a maximum of 7 threads for any given boost. A ridge regression component did glean some improvement, with going about halfway to maximum potential being implemented. Oddly, LASSO components were found to only detract from predictive power and were thus not included. Each child node needed to have at least 15 instances of it being used to be created, and the most layers any single tree could have is 6. The number of variables supplied to the trees was maximized. The learning rate was low at 0.4, which seems low, but makes more sense when one looks at what the structure of the top trees looks like.

#### Variable Importance

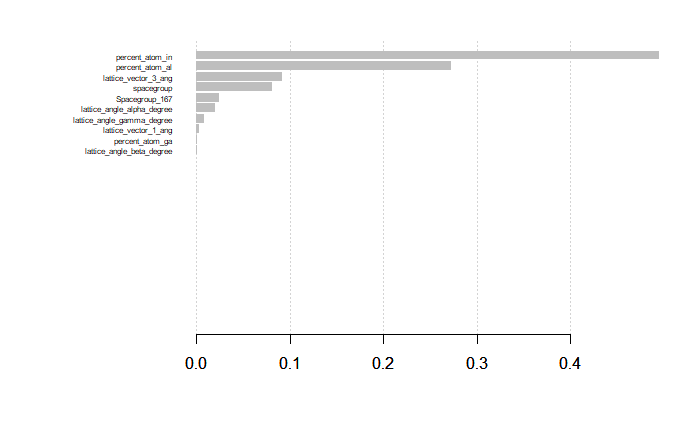
Variable importance for final model for bandgap energy using top 10 predictors.

mat = xgb.importance(feature\_names=colnames(form\_test\_x2\_mat),model=XG\_band)  
xgb.plot.importance(importance\_matrix=mat[1:10])



Looking at all predictors

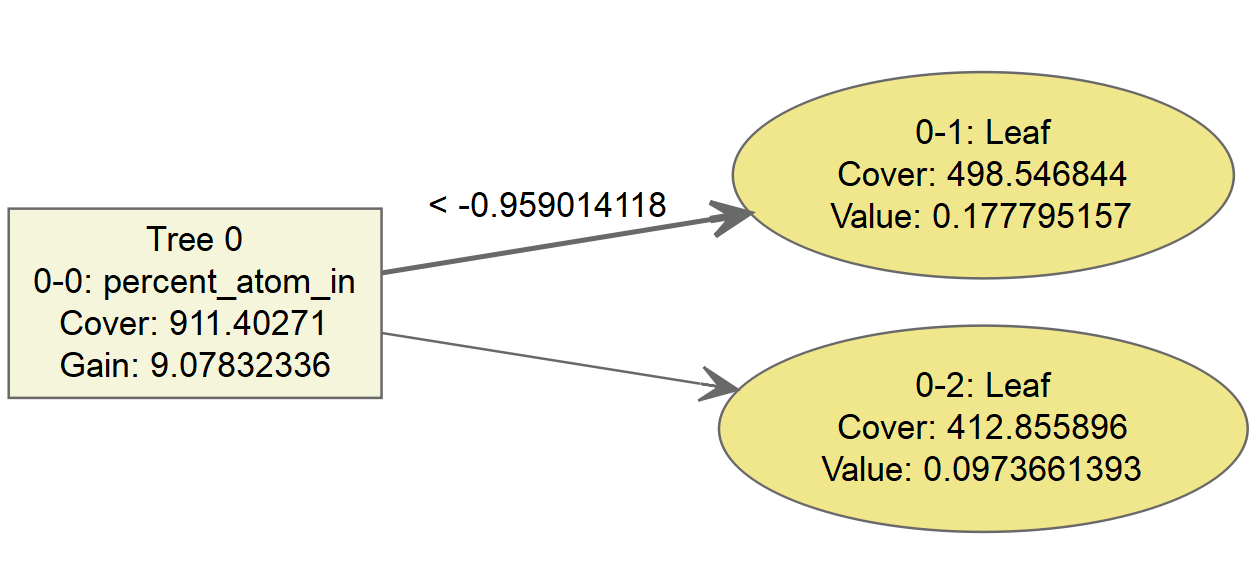
mat = xgb.importance(feature\_names=colnames(form\_test\_x2\_mat),model=XG\_band)  
xgb.plot.importance(importance\_matrix=mat[1:26])



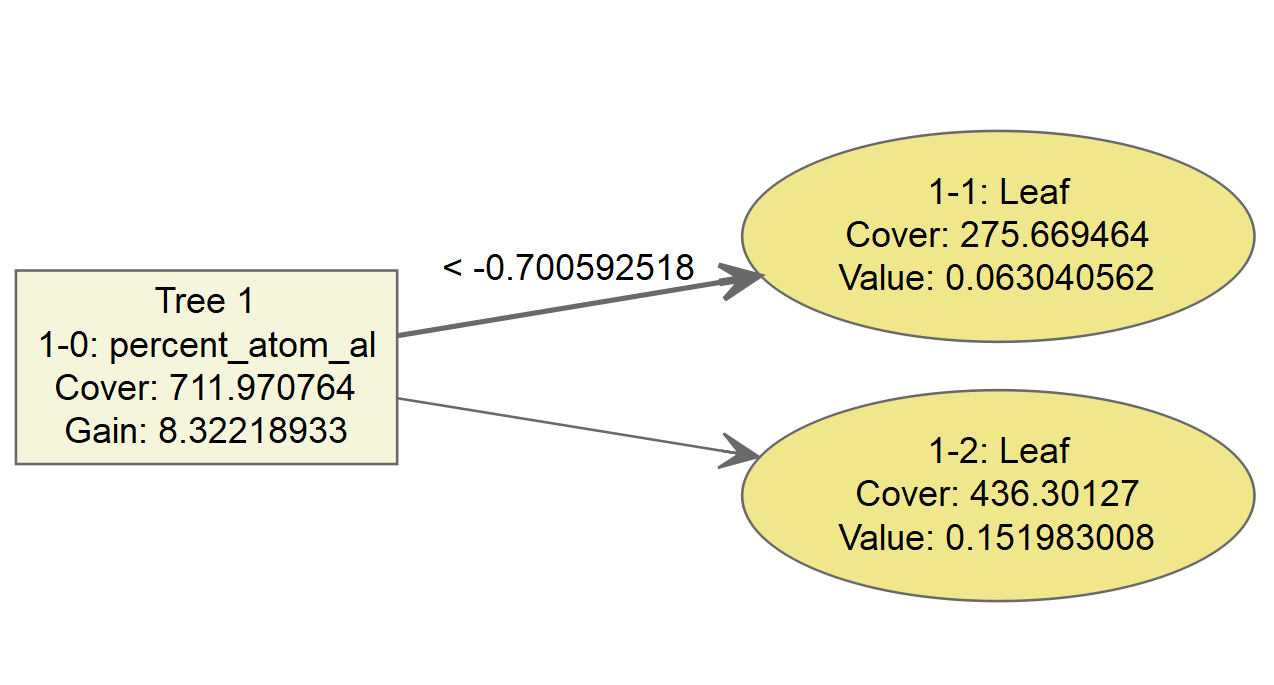
We can see from the charts above that the most important variables by far were what percent of Indium and Aluminum were in each conductor, with the two of the being responsible for almost 70% of the result. Additionally, only a small portion of our variable were actually used, with both existing metrics and new ones we created being used. Spacegroups themselves numerically were very important, as was the category specifically calling out spacegroup 167, which in itself was distinct enough to warrant a little over 2% importance. Lattice vector angles 1 and 3, and angle degrees gamma and alpha were also used, with and beta and vector 2 being absent, likely as geometrically the information can be considered redundant. Beyond that, most variables were ignored, probably due to either again being redundant or simply not relevant.

Looking at the all the tree plots is not practical with XGboost, especially considering we would have over 200 0f them due to how the trees are structured. Below is shown the first few trees in the boosted chain just to give an idea as to what each boost does.

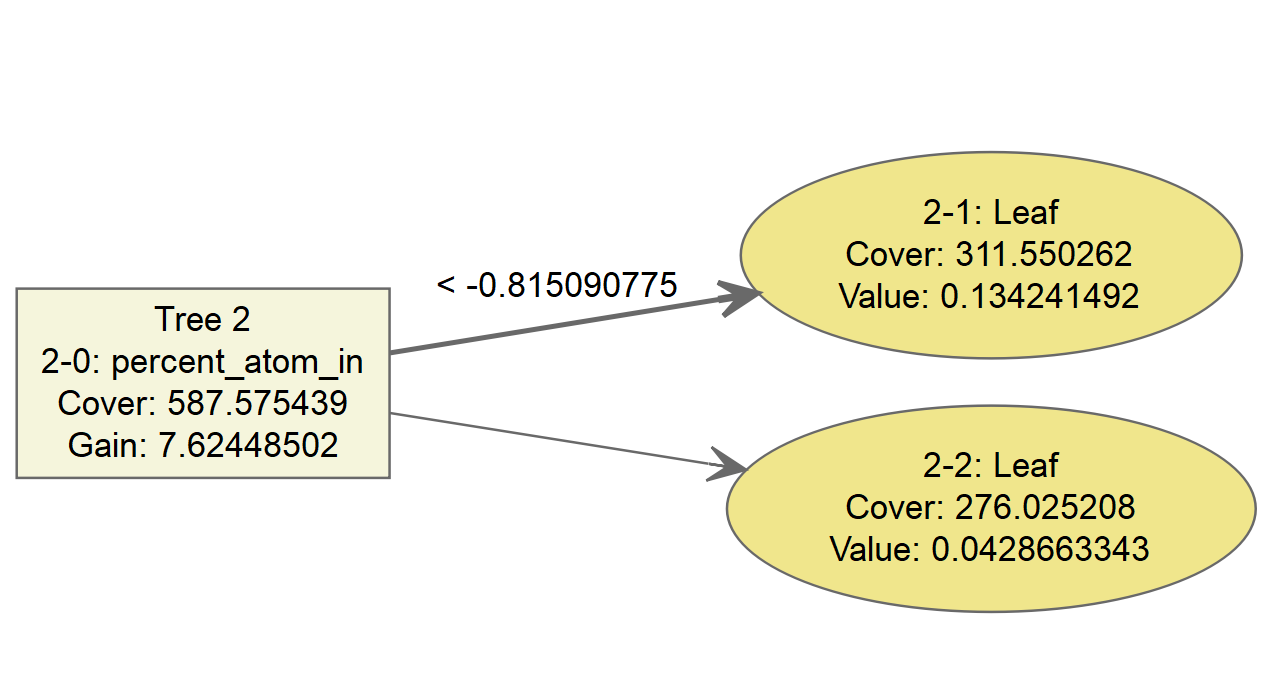
require(DiagrammeR)  
xgb.plot.tree(model = XG\_band, trees = 0, show\_node\_id = T)



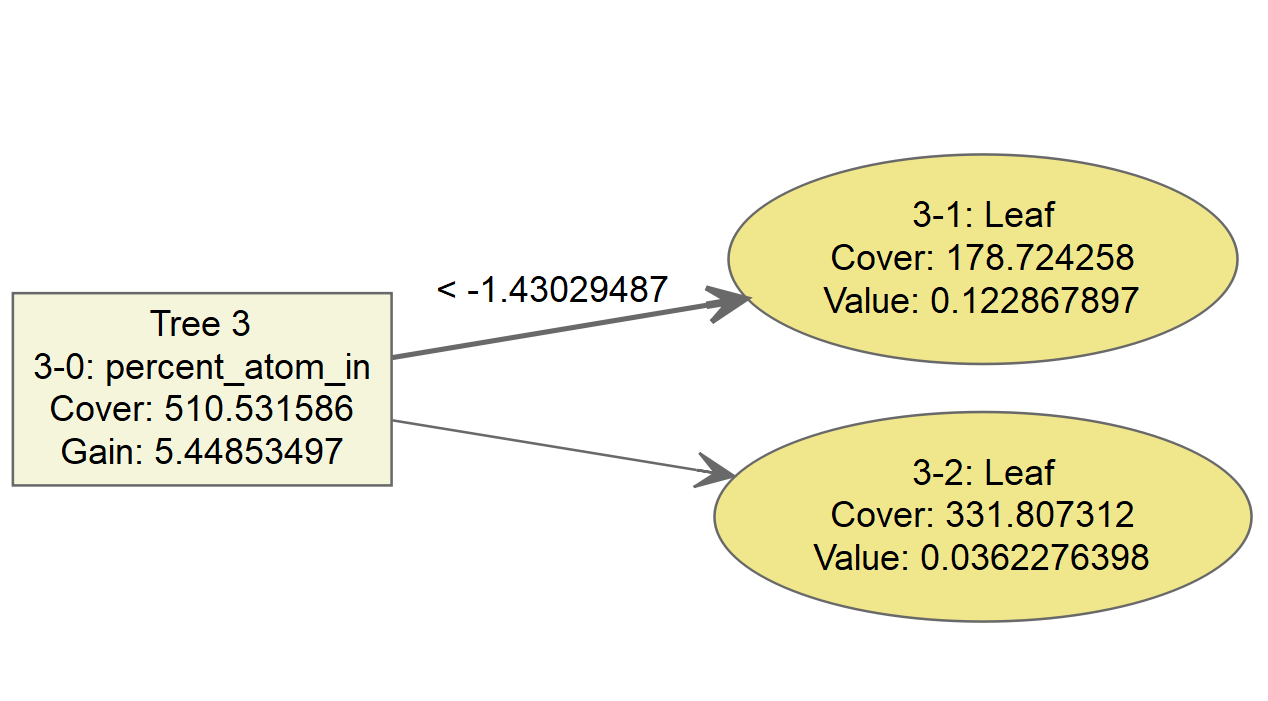
xgb.plot.tree(model = XG\_band, trees = 1, show\_node\_id = T)



xgb.plot.tree(model = XG\_band, trees = 2, show\_node\_id = T)



xgb.plot.tree(model = XG\_band, trees = 3, show\_node\_id = T)



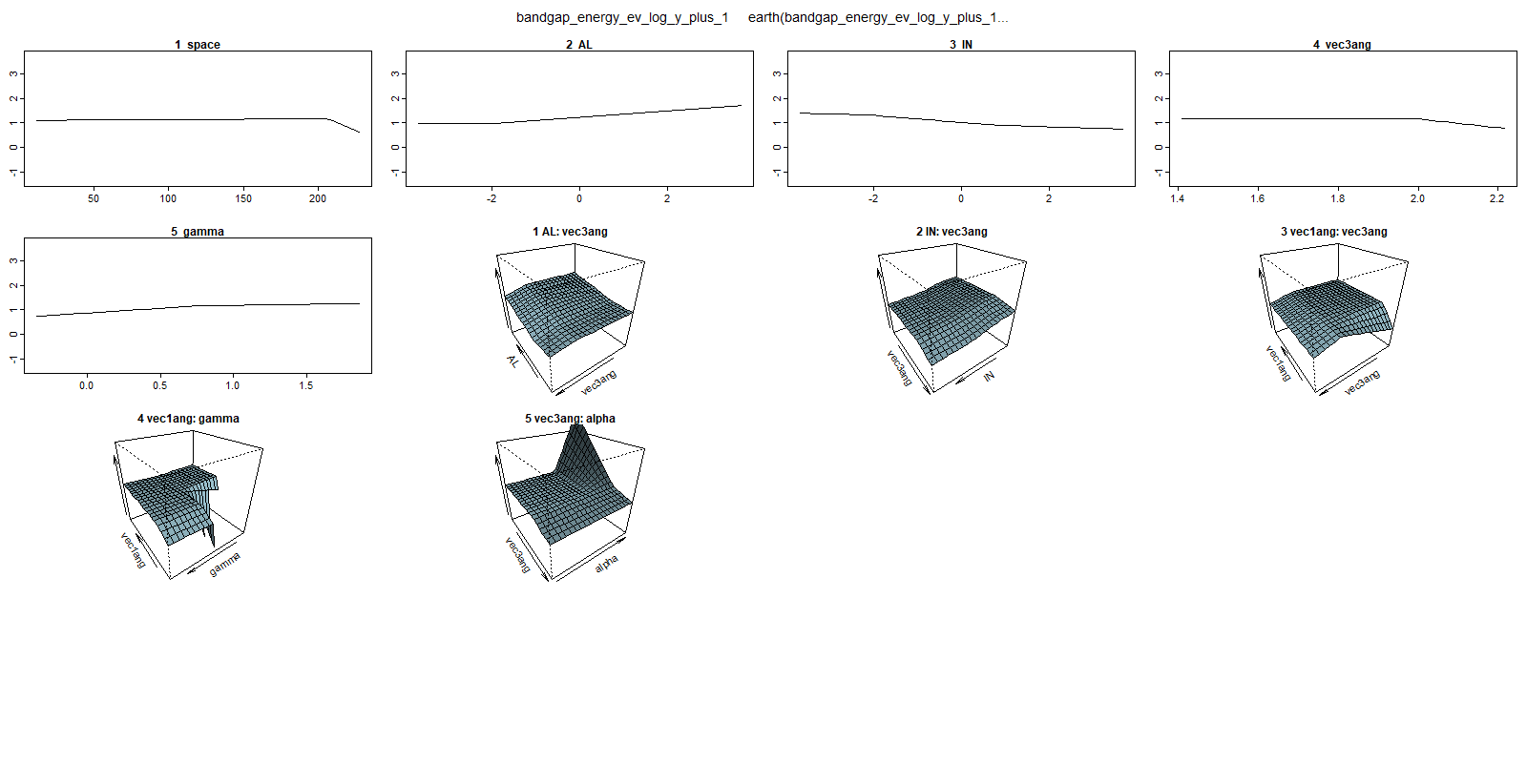
Note how the first couple of trees with the largest impact are all very simple. Given the small amount of variables and our decision to lower the learning rate along with having a few key predictors found to be useful, it makes sense that our extreme boosting mostly covered simple trees that by themselves were still fairly competent, which lines up with the results from our MARS and Shrinkage Models, which tended to follow similar patterns in terms of variable importance.

XGboosts implementation does not a good way to tell specifically how the variables effect the data. However, we can use our other models that performed almost as well to get at least a similar understanding as to how specific predictors affect the results. Our MARS model, being the second best for our result, only being about 0.02 RMSLE off, should work nicely.

cond\_train3\_band\_pifs = read.csv("cond\_train3\_band\_stup.csv")  
band.mars2 = earth(bandgap\_energy\_ev\_log\_y\_plus\_1~.,degree = 2, data = cond\_train3\_band\_pifs, nk = 30, nfold = 20, nprune = 15, pmethod = "exhaustive" )

plotmo(band.mars2, caption = NULL )

## plotmo grid: X total\_atoms X10 X20 X30 X40 X60 X80 space S12 S33 S167  
## 784 692.5198 0 0 0 0 0 1 194 0 0 0  
## S194 S206 S227 AL GA IN vec1ang vec2ang vec3ang  
## 0 0 0 -0.484246 -0.7451684 -1.033015 3.029535 3.802559 1.757565  
## alpha beta gamma Al\_C GA\_C IN\_C  
## 0.08647366 0.06445731 0.7730218 0 0 0



Variable names had to be changed in order to prevent plotmo from failing to provide readable plots, as shown in the new read in.

From these plots generated using our model, we can tell that if one wanted to maximize bandgap energy they should add a higher percentage of Aluminum atoms, have a combination of less vector degree 3 and less lattice angle alpha, and reduce the total amount of indium present in the formulation, and crank up lattice angle gamma. However, if one is to increase angle gamma, also add in a little bit of vector angle 1 to get the maximal return. For spacegroup, go with specifically spacegroup 167 if you can, as it is really the only spacegroup to meaningfully increase this response.