***PREDICTING THE CRITICAL TEMPERATURE OF SUPERCONDUCTORS***

**ENGINEERING DATA ANALYSIS**

Submitted in partial fulfillment of the

Requirement for the assigned work for the course

**SUBMITTED BY:**

Ketan Sethi

**DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING**

**SPRING 2019**

**SUBMITTED TO:**

Dr. Satish Bukkapatnam



# ***EXECUTIVE SUMMARY***

Objective: The primary goal of this project lies in being able to predict the **Critical Temperature** of a superconductor, given a certain list of its chemical properties. In doing so, a statistical modelling approach supported by actual data has been adapted rather than a theoretical basis for correlation of individual properties to superconductivity. It is intended to improvise on the prediction accuracy results currently available in publications (*references listed in the reference section*) using the same set of data and features extracted from any given material’s chemical formula. With the proposed model, a superconductor can be positively assumed to start behaving as one within a close range of absolute zero temperature. An attempt to bring down the number of features being actually used for the prediction, thereby providing a better interpretability with respect to some of the most important chemical properties or their levels that are necessary for a material to possess to behave as a superconductor at that temperature.

The immediate beneficiaries of this work would be the researchers working on superconductivity and other scientific enthusiasts who base most of their work on empirical and practical experimentation to corroborate their work. As a result of the prediction model summarized in this report, one can assume that it may lead to a reduction in the experimental setup time as with a prior estimate of the critical temperatures, and know the vicinity of temperatures that needs to be checked. This would in turn translate to substantial cost reductions via feasibility studies based on a prior knowledge of the critical temperatures so as to decide on whether or not a possible large-scale application can be thought of with a particular superconductor.

Literature Review: The inspiration for this work draws from the recent publication by Kam Hamidieh, University of Pennsylvania – Wharton, Statistics Department (2018). This work has made use of 80 predictors extracted from a small set of material properties of the superconductors. The author has attempted to run a complete large model to help in the development of a general prediction model for the critical temperatures of the superconductors. At the same time, made a benchmark out of the multiple linear regression model also with all the predictors. Apart from this, other notable works are by Valentin et al. (2017), who present a combined classification and prediction model to predict the critical temperatures for the Cuprates and Iron (Fe) based superconductors and not a general model. Previously, Owolabi et al. (2014) and Owolabi and Olatunji (2015) focus on predicting Tc for Fe and MgB2 based superconductors respectively. Thus, as was evident, none of the works yet provide a good metric to identify at least one differentiating property of the superconductor that would be in support to categorically state its existence or level of existence led to the superconductivity in that material.

In the approach proposed in this report, for one; aim to achieve a better prediction of the temperature for a broad class of superconductors listed in the same data. In addition to that, attempt has been made to achieve those results with a reduced number of features. Statistical inferences were drawn for the selection of those features.

Methods Used: As this is a typical problem of large-scale prediction of a quantitative variable based on many predictors, commencement was done by employing a *multiple linear regression* with the critical temperature as the response variable. Subsequently, improvisation of the multiple linear regression was done using *Ridge and Lasso regression*.

Other more complex decision-making methods namely *Decision Trees, Random Forests, Bagging, Support Vector Regression, Partial Least Squares, Neural Networks and finally Extreme Gradient Boosting algorithm* have also been implemented.

The number of predictors to be handled for each iteration proved out to be a major hurdle. This led to substantial run time for the algorithms to run on the dataset. Furthermore, the results obtained by utilizing 80 predictors did not proved out to be helpful in identifying a possible dependence of the response variable on some critical material properties. Documenting the interpretability of the obtained results was an additional hurdle.

Key results and Highlights: The following key results can be highlighted from all our implemented models.

# ***INTRODUCTION***

Superconductors are those elemental compounds or materials that begin conducting electrical current through them at or below a specific temperature, better known as the *Critical Temperature* of the material. Under ideal conditions, this temperature is absolute zero temperature, i.e. 0K. At this temperature, the material begins to display zero resistivity properties along with other uniquely notable effects such as the Meissner and Josephson effects. Lately, the superconductors have found a growing number of existing and potential applications that make use of them. The closest that an ordinary person has come of using them is in the MRI imaging machines for medical diagnosis as well in fast transportation systems like the MagLev (magnetic levitation) trains.

However, as exciting as the concept sounds to a layman; the mystery of as to what elemental properties in a material are responsible for such a behavior has still not been solved. In 1911, Heike Kamerlingh Onnes accidentally discovered superconductivity; since then the theory behind finding out the exact demarcating properties of the material that contribute towards such an effect has remained an open problem to be solved by the scientific community till date. Moreover, the use of such materials has remained largely in research labs, is due to fact that these materials have to be cooled down to unnatural low temperatures before they begin displaying superconductivity; many a times to the boiling temp of Nitrogen (77K) of many known superconductors. Thus, it is of great importance to be able to correctly know this temperature beforehand as closely as possible to be able to design applications or experiments with the materials.

Our reference work accomplished an out of sample root mean – squared error of ±9.5K and we aim to achieve better than this or achieve a high R2 value on the test set.

The Data: The original data for the materials/compounds comes from the Japan’s National Institute for Material Science (NIMS); the SuperCon data repository. This repository houses the elemental formula along with the critical temperature for superconductivity for all the known superconductors.

Initially, eight key variables or elemental properties were shortlisted from which the rest of features were extracted making use of the molecular formula of the compounds. These key properties are listed below in table 1.

|  |  |  |
| --- | --- | --- |
| Variable | Units | Description |
| Atomic Mass | atomic mass units (AMU) | Total proton and neutron rest masses |
| First Ionization Energy | kilo-joules per mole (kJ/mol) | Energy required to remove a valence electron |
| Atomic Radius | picometer (pm) | calculated atomic radius |
| Density | kilograms per meter cubed (kg/m^3) | Density at standard temperature and pressure |
| Electron Affinity | kilo-joules per mole (kJ/mol) | Energy required to add an electron to a neutral atom |
| Fusion Heat | kilo-joules per mole (kJ/mol) | Energy to change from solid to liquid without temperature change |
| Thermal Conductivity | watts per meter Kelvin (W/mK) | Thermal conductivity coefficient K |
| Valence | no units | typical number of chemical bonds formed by the element |

Table . Key Variables

The formulas for extracting other features are listed in table 2. This method of feature extraction from raw data not only helps in understanding the possible behavior of the response variable on the elemental properties, but also introduces plausible unnecessary correlations among the predictors as they are all derived from some common formula.



The following table lists all the predictors and the response variable in the dataset.



Table 3 Data - Predictors

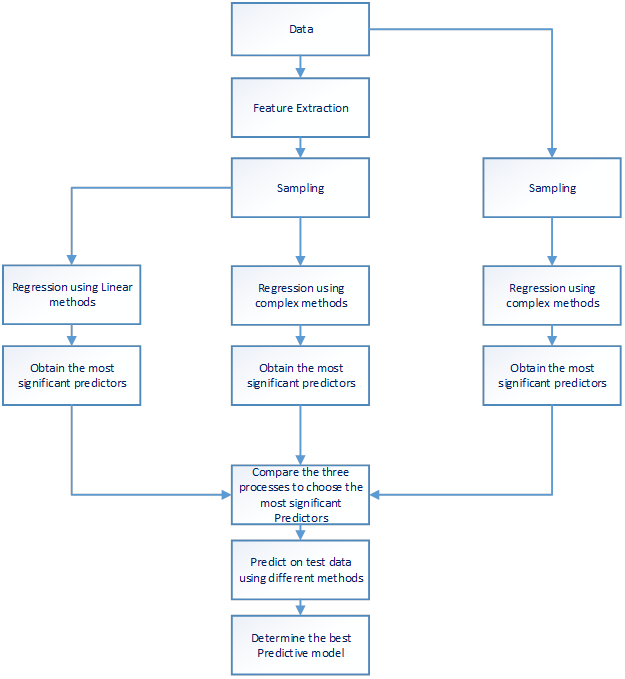


# ***PROJECT APPROACH***

Process Flow: Here, an outline of the project flow via a flow diagram has been presented. Entire project has been split in two main approaches. While one focused on finding the best possible result with the multiple linear regression and other linear methods; the other approach is entirely dedicated to applying several complex methods like decision trees and get better prediction accuracy with possibly a better interpretation of the model.

The linear methods were however not applied directly on the entire raw data set. As mentioned previously in the [introduction](#_INTRODUCTION); high correlation amongst the predictors was suspected and thus, feature selection was done. Upon finalizing the dataset with the reduced features, the data was sampled randomly to divide into a training dataset and a test dataset. The training data to testing data was sampled with a ration of 80:20.

As with the complex methods; majority of the algorithms were run on the original data set without the reduced number of features. This was due to the robustness of the methods being used and their ability to produce interpretable results taking highly correlated predictors into consideration as well.



New Technique Description: We have chosen to learn and implement the Extreme Gradient Boosting algorithm to the selected dataset. This new technique is based on the classical Boosting algorithm but tweaked to perform much better than the former.

Boosting: Boosting builds models from individual “weak learners” in an iterative way. The basic principle is to make our model learn from previous mistakes. This is achieved by putting more weight on instances where the loss function has higher value.

Gradient Boosting: The gradient is used to minimize loss function similar to neural nets. In each repetition a model is built and its outcome is compared to the actual outcome as per expectation. The loss function is used to calculate the gradient which is the partial derivative of the loss function. The weights for the model are calculated based on the gradient obtained, which also helps to determine the direction in which model parameters needs to be changed to achieve the solution.

## Parameters Include:

* Number of iterations (Trees to be constructed)
* Tree complexity and depth
* Proportion of samples
* Proportion of features to train on

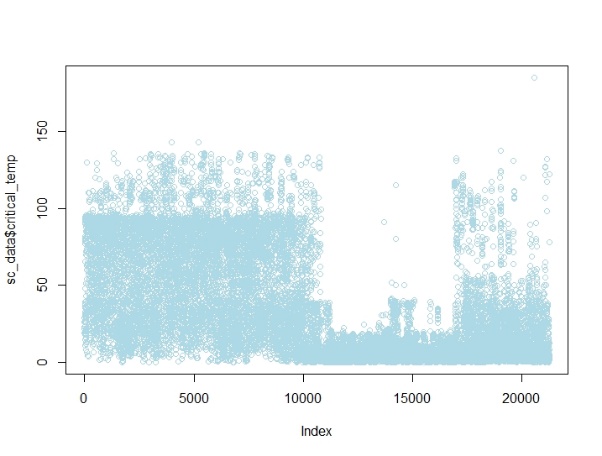
Extreme Gradient Boosting: The ‘xGBoost’ alternative on the former algorithm is as follows.

* xGBoost uses second order partial derivatives i.e. second order gradients which provide more information regarding the direction and how to achieve solution
* xGBoost can penalize based on L1 and L2 regularizations
* For faster computing xGBoost can make use of multiple cores of CPU. This is possible due to the block structure in its system design. Data is sorted and stored in memory units called blocks
* This enables the data layout to be reused by subsequent iterations instead of computing it again

# ***IMPLEMENTATION***

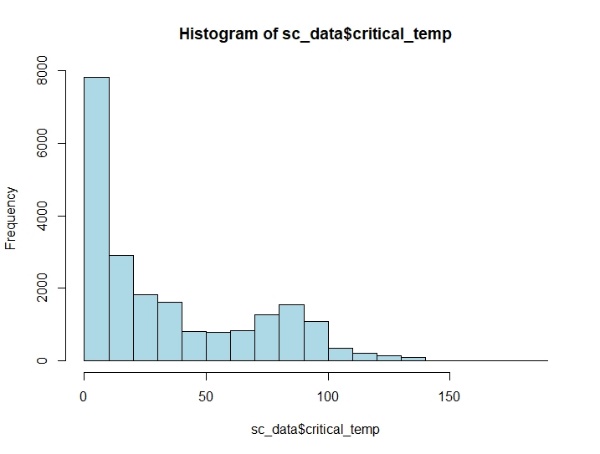
## Exploratory analysis:

We first wished to see the distribution of the critical temperatures of all the 21263 elements listed to gain some insight into the data. Thus, we plotted the scatter plot of all the critical temperatures.



The scatter plot here shows that the data has a majority of the low temperature superconductors i.e., superconductors having a critical temperature below 35K.

Fig. 1 Scatter Plot - Critical Temperatures



This was further corroborated when the histogram of the compounds for their critical temperatures was plotted.

Fig. 2 Histogram of Critical Temperatures

On having a first glance at the data and the source that it came from, it indicated that the features may have strong correlations among themselves. This was due to the fact that majority of the variables have been extracted by a mathematical formula, the operands of which are common to all.

As a result, first step towards the project was finding all the correlations among the predictors and consequently remove them from the data to start modelling.

Correlation Plots: As the feature extraction was done on the eight key variables listed in table 1 above; a decision was made to check for the correlations in groups of ten feature; each derived from one of those key variables. Presented below are the plots from the analysis.

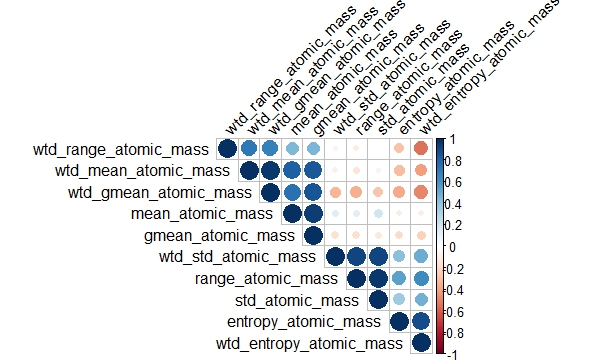
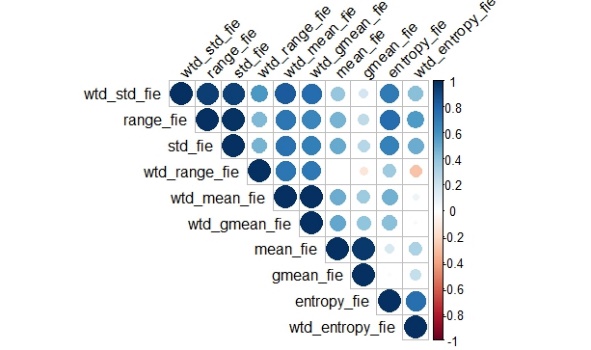


Fig. 3 CorrPlot - Atomic Radius

Fig. 4 CorrPlot - Density

Fig. 1 CorrPLot - Atomic Mass

Fig. 3 CorrPlot - Atomic Mass

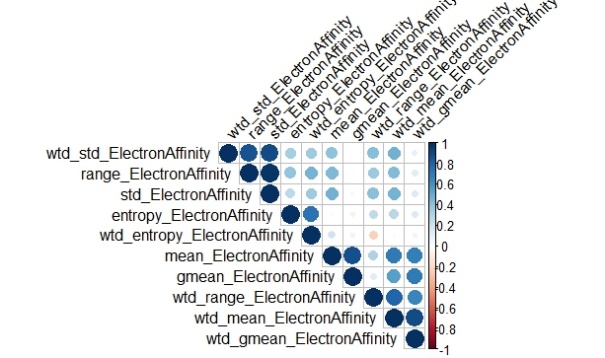
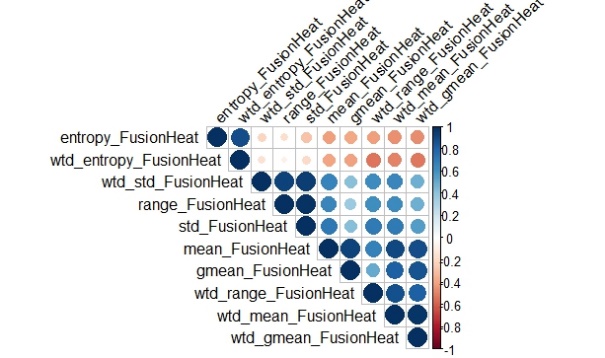
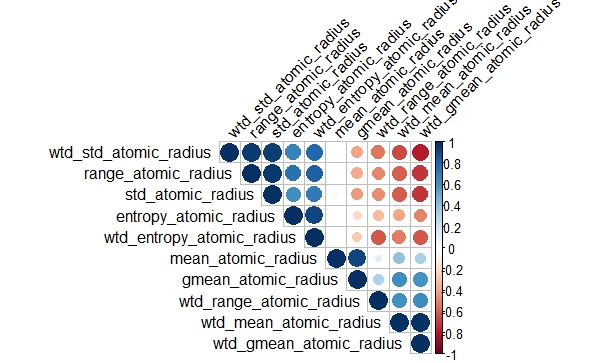
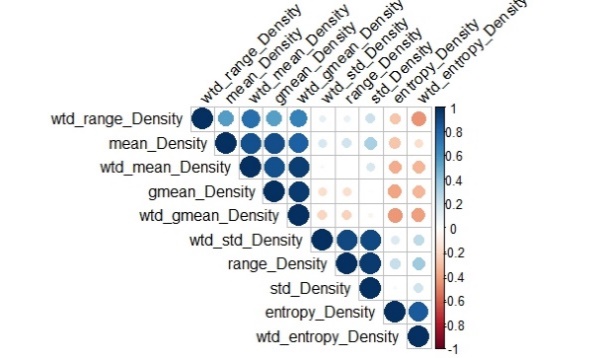


Fig. 2 CorrPlot - First Ionization Energy

Fig. 6 CorrPlot - Fusion Heat

Fig. 5 CorrPlot - Electron Affinity

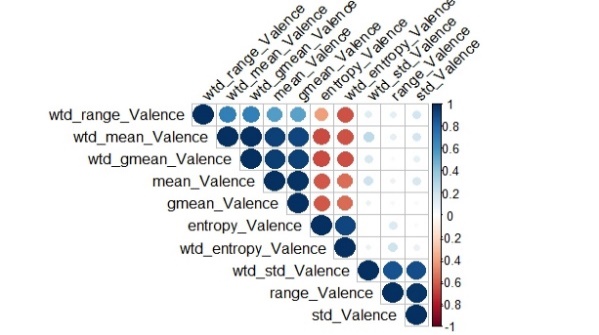
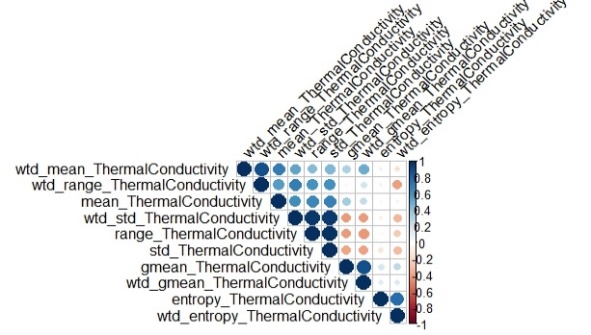


Fig. 7 CorrPlot - Thermal Conductivity

Fig. 8 CorrPlot - Valence

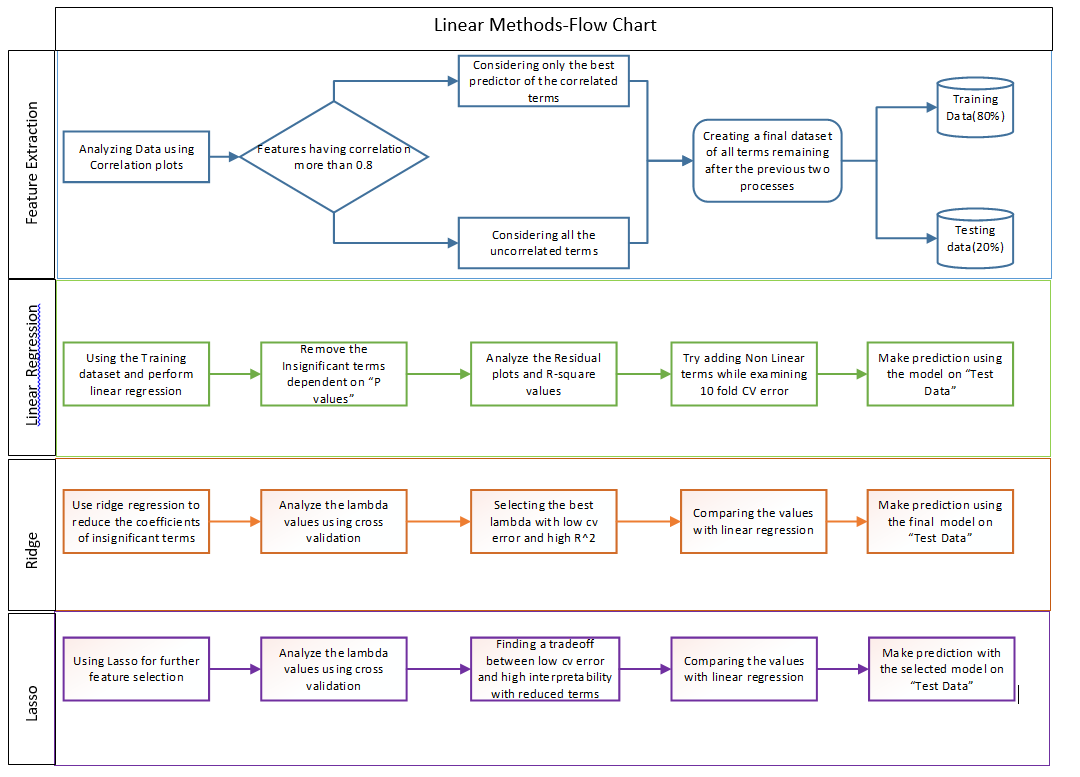
As suspected, there were high correlations observed amongst the groups that were chosen. Majority of the derived predictors were highly positively correlated to the each other while a few showed high negative correlations. This is seen in the Blue and Red regions in the plots respectively. Out of each group of ten predictors, ones which could more or less explain the entire group’s behavior were chosen. Also, the few un-correlated predictors were kept intact in the data. This feature selection brought down the number of predictors significantly from 80 to 36; which is a decrease of 55%.

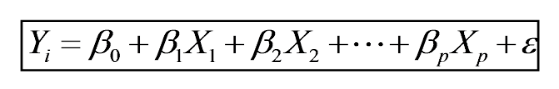


# ***METHODS***

## Linear Methods:

Process Flow: It is known that there are very few linear methods which help in Feature Extraction and hence while considering linear methods, one starts with a reduced dataset which was achieved in the previous section. Analysis was commenced with “Linear Regression” which is set as a benchmark to compare other methods with. This will be succeeded by Ridge Regression to achieve better results and then to Lasso to achieve better interpretability. All the models were trained on “Train Data”, which is 80% of the actual dataset and the remaining 20% of the data is used for prediction.



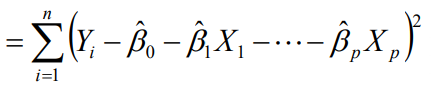
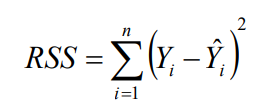
Linear Regression: Linear regression is a linear approach of modelling the relationship between a quantitative response and one or more explanatory variables.

General Equation .

Here

* β0 is the intercept that is the average value of Y if all the X’s are zero
* βj is the average increase in Y when Xj is increased by one and all other X’s are held constant

In Linear Regression, coefficients are estimated “Ordinary Least Squares” that is by minimizing

R\_square: This statistic will be used to compare the model by determining its value on the

“test data” by using where RSS is same as above and TSS is .

Output:

* On Reduced data set with insignificant terms removed

attach(sc\_data1\_Train)

sc\_linear=lm(critical\_temp~.-wtd\_gmean\_fie-range\_fie-wtd\_range\_fie-wtd\_entropy\_atomic\_radius-wtd\_std\_ElectronAffinity-mean\_ThermalConductivity-wtd\_range\_ThermalConductivity-entropy\_Valence,data = sc\_data1\_Train)

summary(sc\_linear)

##   
## Call:  
## lm(formula = critical\_temp ~ . - wtd\_gmean\_fie - range\_fie -   
## wtd\_range\_fie - wtd\_entropy\_atomic\_radius - wtd\_std\_ElectronAffinity -   
## mean\_ThermalConductivity - wtd\_range\_ThermalConductivity -   
## entropy\_Valence, data = sc\_data1\_Train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -78.939 -11.725 0.074 12.463 119.011   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -3.597e+01 4.704e+00 -7.646 2.19e-14 \*\*\*  
## wtd\_gmean\_atomic\_mass 8.484e-02 1.317e-02 6.443 1.21e-10 \*\*\*  
## wtd\_entropy\_atomic\_mass 2.625e+01 1.605e+00 16.359 < 2e-16 \*\*\*  
## range\_atomic\_mass 1.495e-01 5.646e-03 26.474 < 2e-16 \*\*\*  
## wtd\_range\_atomic\_mass -1.369e-01 1.898e-02 -7.215 5.61e-13 \*\*\*  
## mean\_fie 4.428e-02 3.541e-03 12.506 < 2e-16 \*\*\*  
## mean\_atomic\_radius -1.096e-01 1.904e-02 -5.756 8.79e-09 \*\*\*  
## wtd\_gmean\_atomic\_radius 2.271e-01 1.535e-02 14.787 < 2e-16 \*\*\*  
## range\_atomic\_radius 1.308e-01 7.773e-03 16.828 < 2e-16 \*\*\*  
## wtd\_range\_atomic\_radius -6.694e-02 1.149e-02 -5.826 5.79e-09 \*\*\*  
## gmean\_Density -1.431e-03 1.458e-04 -9.817 < 2e-16 \*\*\*  
## entropy\_Density -1.586e+01 1.538e+00 -10.312 < 2e-16 \*\*\*  
## range\_Density -1.277e-03 6.619e-05 -19.295 < 2e-16 \*\*\*  
## wtd\_range\_Density 1.608e-03 1.501e-04 10.715 < 2e-16 \*\*\*  
## wtd\_mean\_ElectronAffinity -1.727e-01 1.120e-02 -15.415 < 2e-16 \*\*\*  
## gmean\_ElectronAffinity 1.290e-01 1.375e-02 9.383 < 2e-16 \*\*\*  
## entropy\_ElectronAffinity -1.734e+01 1.617e+00 -10.724 < 2e-16 \*\*\*  
## wtd\_entropy\_ElectronAffinity -2.830e+01 1.354e+00 -20.899 < 2e-16 \*\*\*  
## wtd\_gmean\_FusionHeat 9.648e-02 2.216e-02 4.355 1.34e-05 \*\*\*  
## wtd\_entropy\_FusionHeat 1.165e+01 1.220e+00 9.547 < 2e-16 \*\*\*  
## std\_FusionHeat -2.479e-01 2.711e-02 -9.146 < 2e-16 \*\*\*  
## gmean\_ThermalConductivity -6.802e-02 7.964e-03 -8.542 < 2e-16 \*\*\*  
## entropy\_ThermalConductivity 1.904e+01 1.131e+00 16.839 < 2e-16 \*\*\*  
## wtd\_entropy\_ThermalConductivity 6.389e+00 1.157e+00 5.520 3.43e-08 \*\*\*  
## wtd\_std\_ThermalConductivity 2.812e-01 6.259e-03 44.929 < 2e-16 \*\*\*  
## wtd\_mean\_Valence -1.992e+00 3.977e-01 -5.009 5.53e-07 \*\*\*  
## wtd\_range\_Valence 4.309e+00 3.960e-01 10.882 < 2e-16 \*\*\*  
## wtd\_std\_Valence -1.272e+01 4.680e-01 -27.181 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 19.37 on 16509 degrees of freedom  
## Multiple R-squared: 0.6805, Adjusted R-squared: 0.68   
## F-statistic: 1302 on 27 and 16509 DF, p-value: < 2.2e-16

Prediction

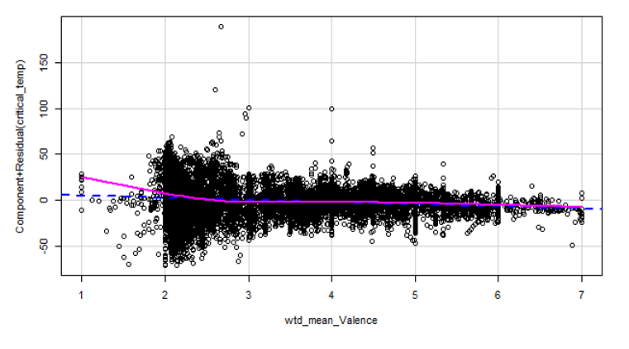
* On Test Data

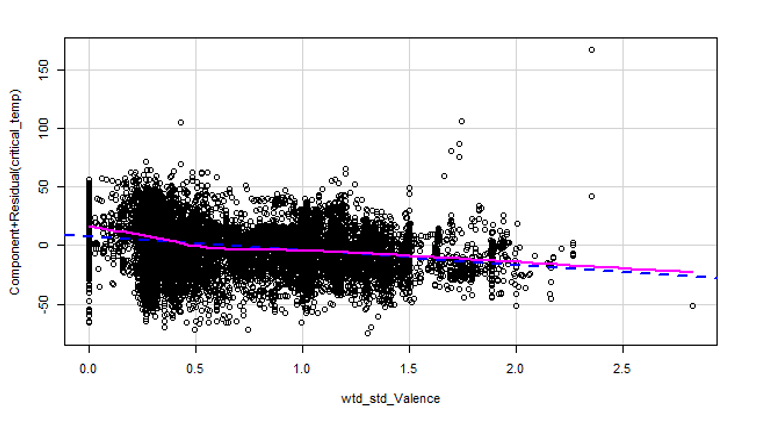
pred\_linear=predict(sc\_linear,sc\_data1\_Test)  
#R^2  
RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_linear)^2))  
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))  
R\_square=1-(RSS/TSS)  
print(R\_square)

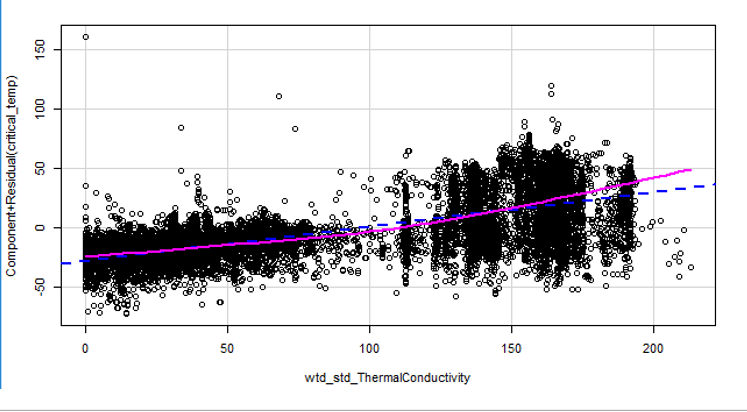
## [1] 0.6795226

Observation

* The residuals with each variable is plotted and a pattern is obtained only in three variables.
* Different Non-Linear terms of these variables were iterated to randomize the residuals and obtain better results in terms of R\_Squared values.







Linear Regression with Non Linear Terms

Output

sc\_linear=lm(critical\_temp~.-wtd\_gmean\_fie-range\_fie-wtd\_entropy\_atomic\_radius-wtd\_std\_ElectronAffinity-mean\_ThermalConductivity-wtd\_range\_ThermalConductivity-entropy\_Valence+I(wtd\_mean\_Valence^2)+I(wtd\_std\_ThermalConductivity^2)-wtd\_std\_ThermalConductivity-wtd\_range\_atomic\_radius-wtd\_range\_fie,data = sc\_data1\_Train)

summary(sc\_linear)

## Residual standard error: 19.15 on 16512 degrees of freedom

## Multiple R-squared: 0.6904, Adjusted R-squared: 0.6899

## F-statistic: 1364 on 27 and 16512 DF, p-value: < 2.2e-16

pred\_linear=predict(sc\_linear,sc\_data1\_Test)

#R^2

RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_linear)^2))

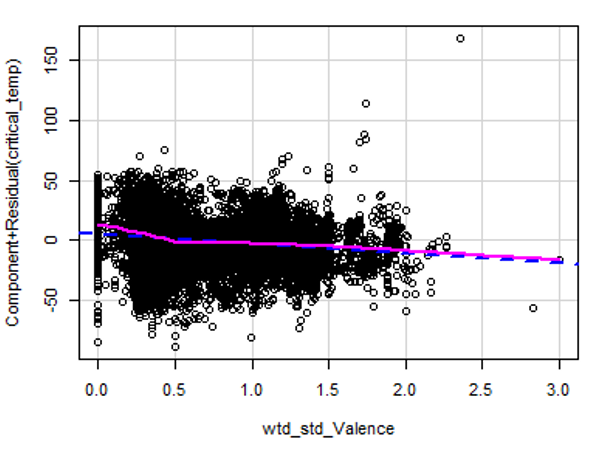
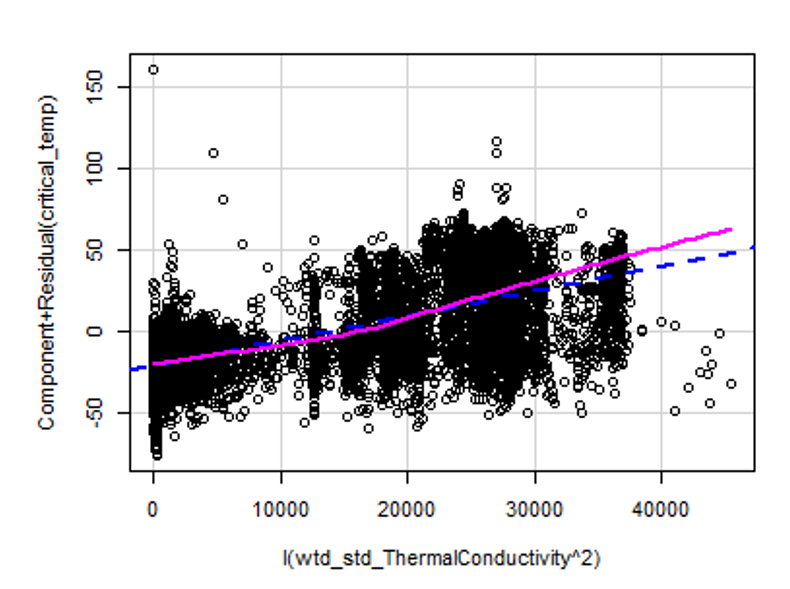
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))

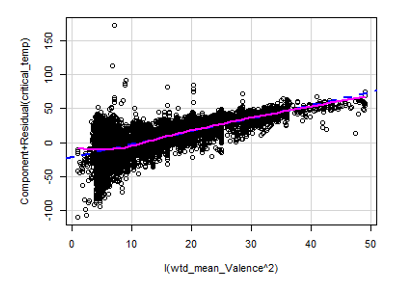
R\_square=1-(RSS/TSS)

print(R\_square)

**[1] 0.6751566**

Observation

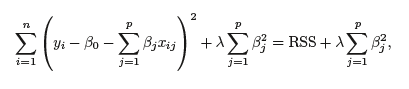


* The residuals are plotted with each variable and a pattern is obtained only in three variables
* It can be observed that the pattern is getting removed from the residuals.
* The model explains 69% of the variance which is more than what previous iteration explained but it performs almost similar in respect to test data.

Result

Hence, a model with only linear terms is the best model for “Linear Regression”.

Ridge Regression: Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity. In particular, the ridge regression coefficient estimates βR are the values that minimize.



Where λ ≥ 0 is a tuning parameter, determined through cross validation.

Even though ridge regression is being used, it is known that this method can only reduce the value of coefficient and not reduce it to zero as a result of which, it is not used for variable selection. Ridge regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region. Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero.

Code:

x=model.matrix(critical\_temp~.,sc\_data1\_Train)[,-1]

y=sc\_data1\_Train$critical\_temp

x\_test=model.matrix(critical\_temp~.,sc\_data1\_Test)[,-1]

y\_test=sc\_data1\_Test$critical\_temp

grid=10^seq(5,-5,length=100)

sc\_ridge=glmnet(x,y,alpha = 0,lambda = grid)

set.seed(1)

cv\_out=cv.glmnet(x,y,alpha=0)

plot(cv\_out)

bestlam=cv\_out$lambda.min

pred\_ridge=predict(sc\_ridge,s=bestlam,newx = x\_test)

#R^2

RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_ridge)^2))

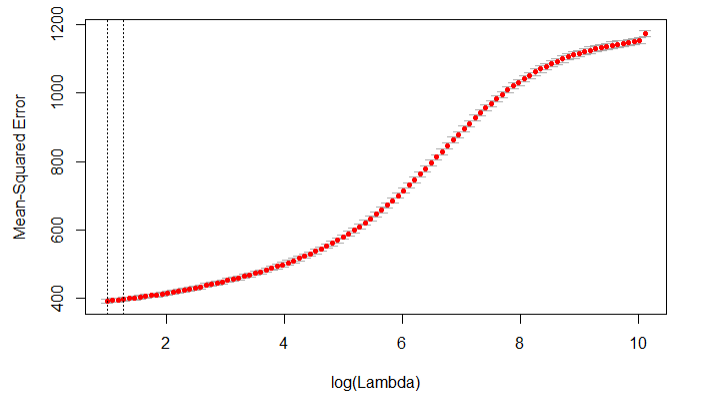
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))

R\_square=1-(RSS/TSS)

print(R\_square)

[1] 0.6589719

Output



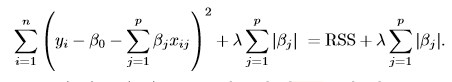
Observation

* As seen in the figure above, increasing the value of “lambda” increases the error
* This indicates that our model behaves better with the Linear Regression coefficient values

Results

* Ridge is only able to explain 65.89% of the variance present in the dataset
* This prediction is made with a low “Lambda value” indicating a model similar to “Linear Regression”

Lasso: Main purpose to use ridge regression is for variable selection which was missing in Ridge regression. In lasso, constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero.

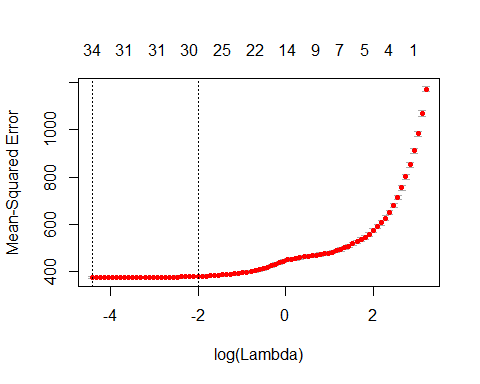


Where λ ≥ 0 is a tuning parameter, determined through cross validation.

Code:

x=model.matrix(critical\_temp~.,sc\_data1\_Train)[,-1]  
y=sc\_data1\_Train$critical\_temp  
x\_test=model.matrix(critical\_temp~.,sc\_data1\_Test)[,-1]  
y\_test=sc\_data1\_Test$critical\_temp  
  
grid=10^seq(10,-2,length=100)  
  
sc\_lasso=glmnet(x,y,alpha = 1,lambda = grid)  
  
set.seed(1)  
cv\_out=cv.glmnet(x,y,alpha=1)

Output



#R^2  
RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_lasso)^2))  
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))  
R\_square=1-(RSS/TSS)  
print(R\_square)

## [1] 0.6794531

Observation

* As seen in the figure above initially increasing the value of “lambda” does not increase error and helps reduce predictors.
* We will select a point close to the “knee point “ to achieve variable selection.

bestlam=cv\_out$lambda.1se  
bestlam

## [1] 0.1347874

lasso\_coef=predict(sc\_lasso,s=bestlam,type ="coefficients")  
  
  
pred\_lasso=predict(sc\_lasso,s=bestlam,newx = x\_test)  
result=data.frame(sc\_data1\_Test$critical\_temp,pred\_lasso)  
lasso\_coef

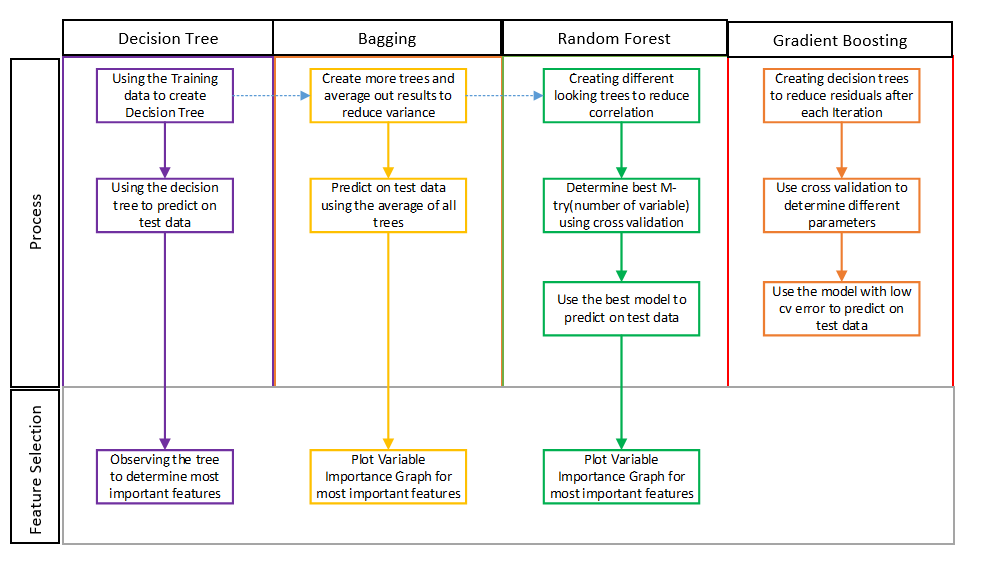
## wtd\_gmean\_atomic\_mass 6.513623e-02  
## wtd\_entropy\_atomic\_mass 2.402785e+01  
## range\_atomic\_mass 1.239755e-01  
## wtd\_range\_atomic\_mass -9.305235e-02  
## mean\_fie 4.450569e-02  
## wtd\_gmean\_fie -6.655131e-04  
## range\_fie .   
## wtd\_range\_fie -6.653570e-03  
## mean\_atomic\_radius -2.827440e-02  
## wtd\_gmean\_atomic\_radius 1.335092e-01  
## wtd\_entropy\_atomic\_radius .   
## range\_atomic\_radius 1.156585e-01  
## wtd\_range\_atomic\_radius -1.337774e-02  
## gmean\_Density -1.246592e-03  
## entropy\_Density -1.339041e+01  
## range\_Density -1.051065e-03  
## wtd\_range\_Density 1.117871e-03  
## wtd\_mean\_ElectronAffinity -1.192676e-01  
## gmean\_ElectronAffinity 4.064981e-02  
## entropy\_ElectronAffinity -1.313213e+01  
## wtd\_entropy\_ElectronAffinity -2.476591e+01  
## wtd\_std\_ElectronAffinity .   
## wtd\_gmean\_FusionHeat 4.994021e-02  
## wtd\_entropy\_FusionHeat 9.618637e+00  
## std\_FusionHeat -2.127475e-01  
## mean\_ThermalConductivity .   
## gmean\_ThermalConductivity -5.046068e-02  
## entropy\_ThermalConductivity 1.443472e+01  
## wtd\_entropy\_ThermalConductivity 7.271038e+00  
## wtd\_range\_ThermalConductivity 1.389179e-02  
## wtd\_std\_ThermalConductivity 2.648282e-01  
## wtd\_mean\_Valence -5.840390e-01  
## entropy\_Valence .   
## wtd\_range\_Valence 2.658098e+00  
## wtd\_std\_Valence -1.187172e+01

Results

* Lasso helps reduce features and does not reduce the R\_Square value with a significant amount
* Lasso helps achieve better interpretability even though its R^2 value is similar to Linear Regression model
* We can achieve better interpretability by increasing the value of Lambda

## Complex Methods:

Process Flow: Complex methods have a crucial advantage over linear methods, which is automated feature selection. The methods are designed to determine the best features and predict the data based on it. While this may appear a better bet but in case of linear decision boundaries, these complex methods under-fit and perform worse than Linear regression methods. While using these methods, both dataset for predictions are used (one is the reduced dataset with non-correlated terms and other is the raw dataset with all the terms). By doing this, it is aimed to achieve similar significant terms which will prove the correctness of feature selection.

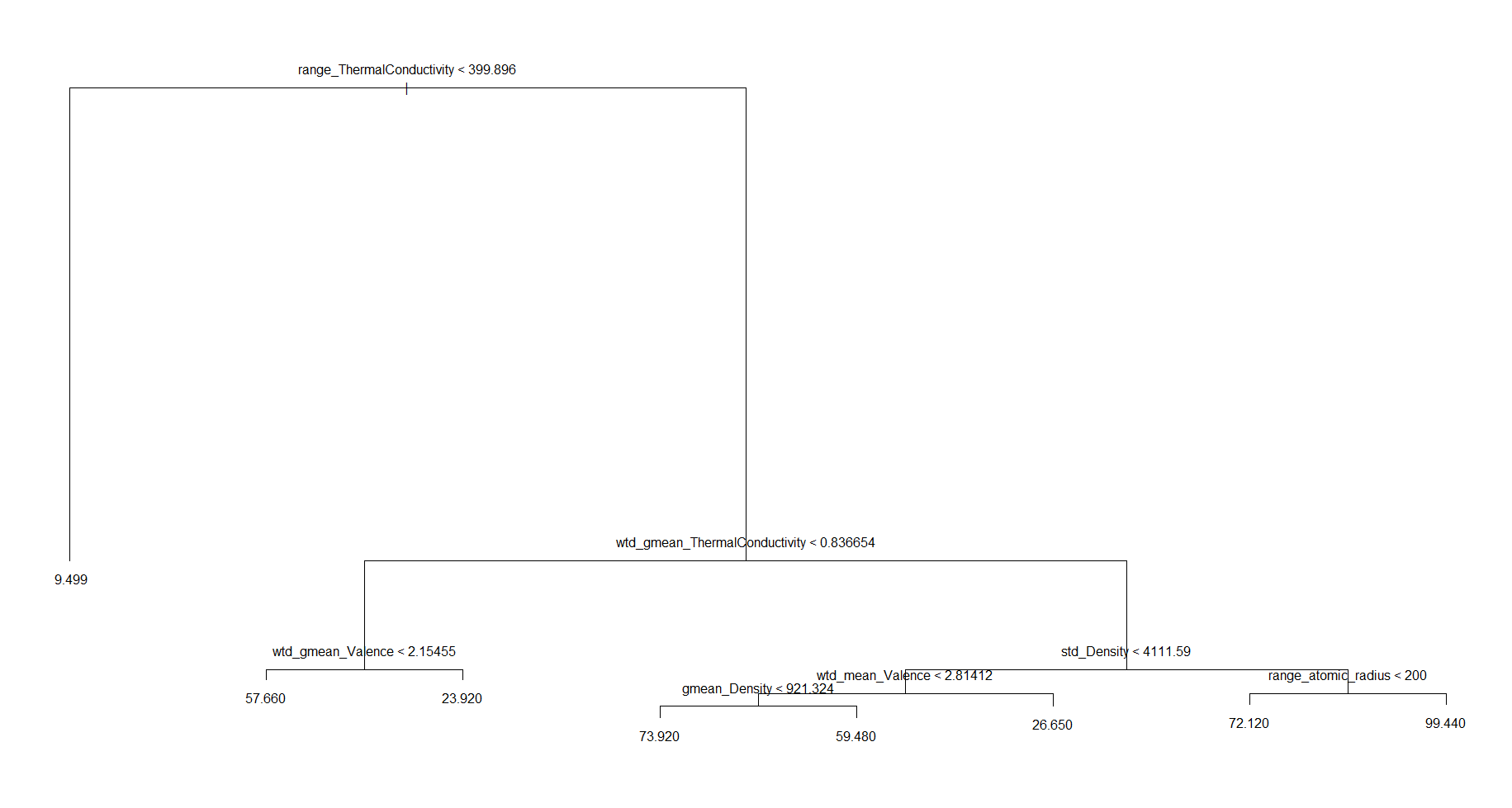


Decision Tree: **Decision Tree** - **Regression**. **Decision tree** builds **regression** or classification models in the form of a **tree** structure. Leaf node (e.g., Hours Played) represents a **decision** on the numerical target. The topmost **decision** node in a **tree** which corresponds to the best predictor called root node.

In regression as well as classification the decision trees segments the feature space into areas with mean value of points as the label for the area in regression and majority label of points in case of classification , The feature space is divided into areas based on the dependence of data set on the predictors

Code

#Loading the Data  
setwd('C:\\Users\\ACER\\Desktop\\Superconductivity')   
sc\_data=read.csv ("Superconductor.csv")  
split=sample.split(sc\_data,SplitRatio = 0.8)  
sc\_data1\_Train=subset(sc\_data,split=="TRUE")  
sc\_data1\_Test=subset(sc\_data,split=="FALSE")  
  
#Decision Tree  
sc\_tree=tree(sc\_data1\_Train$critical\_temp~.,data=sc\_data1\_Train)  
plot(sc\_tree)  
text(sc\_tree,pretty = 0)

Output

pred\_tree=predict(sc\_tree,sc\_data1\_Test)  
  
  
#R^2  
RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_tree)^2))  
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))  
R\_square=1-(RSS/TSS)  
print(R\_square)

## [1] 0.7443315

Observation:

* Decision Tree explains 74.43% of variance hence indicating that model cannot be fully explained by a linear model.
* We will move on to Bagging and Random Forest to test our above hypothesis.

Result:

The R2 predicted on test data is 74.43%, which is better than all the linear methods tested so far.

Bagging: The Decision trees obtain a method with high variance and low bias as it fits a complex decision boundary. Thus to reduce the variance we use Bootstrap Aggregation abbreviated as Bagging which samples the dataset by creating a new dataset with selecting n points with replacement , The bagged tree is tested on the points that are not picked.

The method reduces variance as follows:







Code

library(randomForest)

model\_bag=randomForest(sc\_data1\_Train$critical\_temp~.,data=sc\_data1\_Train,mtry=81,ntree=500)

varImpPlot(model\_bag)

pred\_y=predict(model\_bag,sc\_data1\_Test)

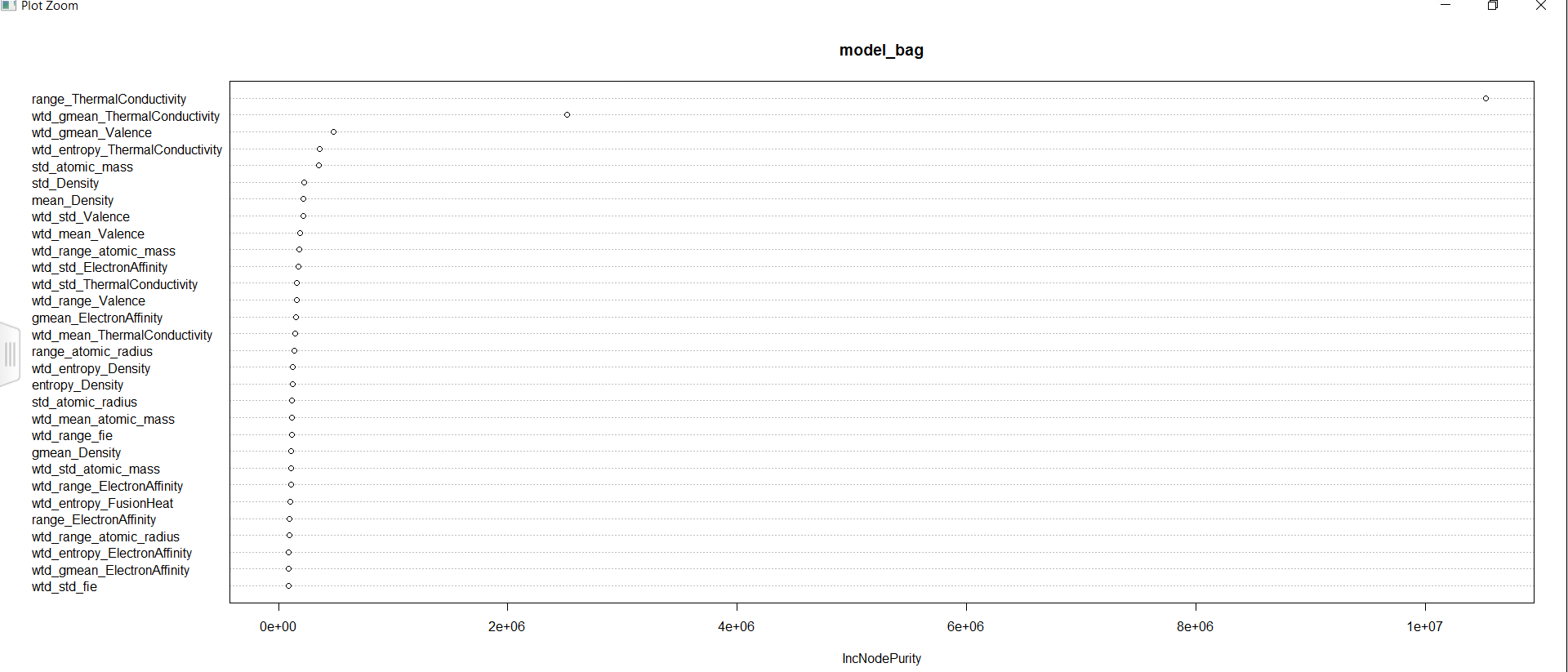
#R^2

RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_y)^2))

TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))

R\_square=1-(RSS/TSS)

print(R\_square)

Observation

* As expected in Bagging, the variable importance plots show first two terms most to be the most important ones, and rest very less. This happens because of the nature of method to create trees with similar nodes.

Result

The Bagging model explains 92.24% of the data showing a significant improvement from the linear models.

Random Forests: Bagging is a special case of Random Forest in which all the variables are considered. In the case of bagging, since all the predictors are considered while creating each tree, most of the trees will have similar nodes, and, consequently, the trees will be correlated. As shown above, the variance reduces while averaging, but this formula is limited because the terms should be independent. Hence, Bagging may not reduce variance.

To overcome this, Random Forest is used, in which only a few variables are considered for each tree, leading to uncorrelated tree and reduced variance.

Code

split=sample.split(sc\_data,SplitRatio = 0.8)

sc\_data2\_Train=subset(sc\_data,split=="TRUE")

sc\_data2\_Test=subset(sc\_data,split=="FALSE")

library(randomForest)

model\_rf=randomForest(sc\_data2\_Train$critical\_temp~.,data=sc\_data2\_Train,mtry=13,ntree=500)

varImpPlot(model\_rf)

pred\_y=predict(model\_rf,sc\_data1\_Test)

#R^2

RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_y)^2))

TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))

R\_square=1-(RSS/TSS)

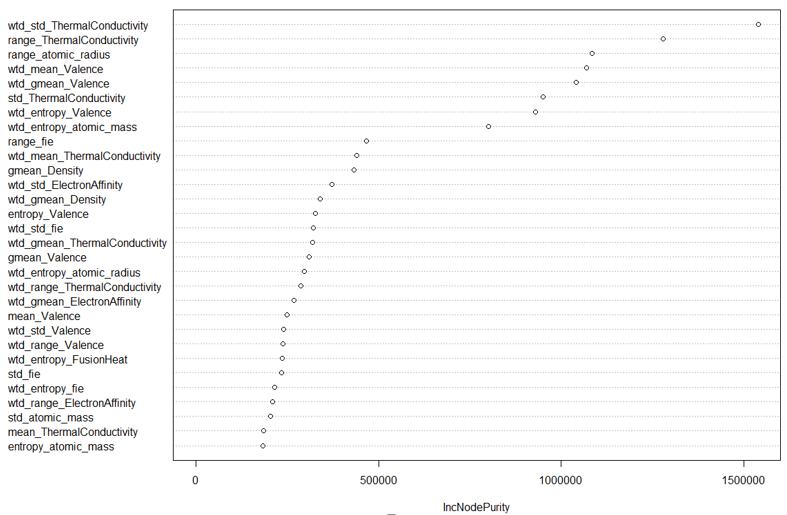
print(R\_square)

[1] 0.9246421

Observation

* There is a significant improvement in R square value
* Due to the creation on uncorrelated trees the most important feature changes to “wtd\_std\_Thermal\_conductivity”
* We created the tree as mentioned in the flow chart on the whole dataset, there is a significant reduce in the time taken by the algorithm from Bagging

Output



Result

* The model explains 92.4% of the variance in the dataset which shows significant improvement from linear methods and also minor improvement from Bagging

Random Forest on reduced dataset

Code

library(randomForest)

model\_bag=randomForest(sc\_data1\_Train$critical\_temp~.,data=sc\_data1\_Train,mtry=6,ntree=500)

varImpPlot(model\_bag)

pred\_y=predict(model\_bag,sc\_data1\_Test)

#R^2

RSS=(mean((sc\_data1\_Test$critical\_temp-pred\_y)^2))

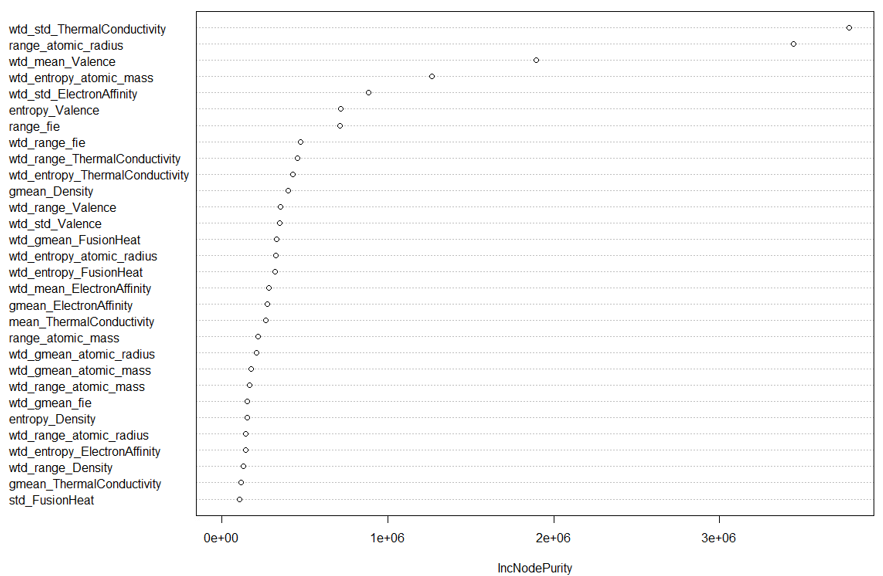
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))

R\_square=1-(RSS/TSS)

print(R\_square)

[1] 0.9241643

Output



Observation:

* We run the Random forest algorithm on the data set with all the features and on the dataset on which we performed “Feature Extraction”
* we achieve two almost identical R square values on test data as well as two similar variable importance plots , hence confirm the process we followed for feature selection

## New Methods

Gradient Boosting :

Code

library(xgboost)

setwd('C:\\Users\\ACER\\Desktop\\Superconductivity')

sc\_data=read.csv ("Superconductor1.csv")

split=sample.split(sc\_data1,SplitRatio = 0.8)

sc\_data1\_Train=subset(sc\_data,split=="TRUE")

sc\_data1\_Test=subset(sc\_data,split=="FALSE")

sc\_data2\_Train=sc\_data1\_Train[,-82]

sc\_data2\_Test=sc\_data1\_Test[,-82]

sc\_train\_matrix=xgb.DMatrix(data=as.matrix(sc\_data2\_Train),label=sc\_data1\_Train$critical\_temp)

sc\_test\_matrix=xgb.DMatrix(data=as.matrix(sc\_data2\_Test),label=sc\_data1\_Test$critical\_temp)

nc=length(sc\_data1\_Train$critical\_temp)

grid

xgb\_param=list("objective"="reg:linear",

"eval\_metric"="rmse")

watchlist=list(train=sc\_train\_matrix,test=sc\_test\_matrix)

XGB\_model=xgb.train(params=xgb\_param,data=sc\_train\_matrix,eta=0.05,max\_depth=16,nrounds=300,subsample=1,watchlist = watchlist)

e=data.frame(XGB\_model$evaluation\_log)

plot(e$iter,e$train\_rmse)

lines(e$iter,e$test\_rmse,col="red")

pred\_q=predict(XGB\_model,sc\_test\_matrix)

pred\_q=data.frame(pred\_q)

result=data.frame(sc\_data1\_Test$critical\_temp,pred\_q)

#R^2

RSS=(mean((result$sc\_data1\_Test.critical\_temp - result$pred\_q)^2))

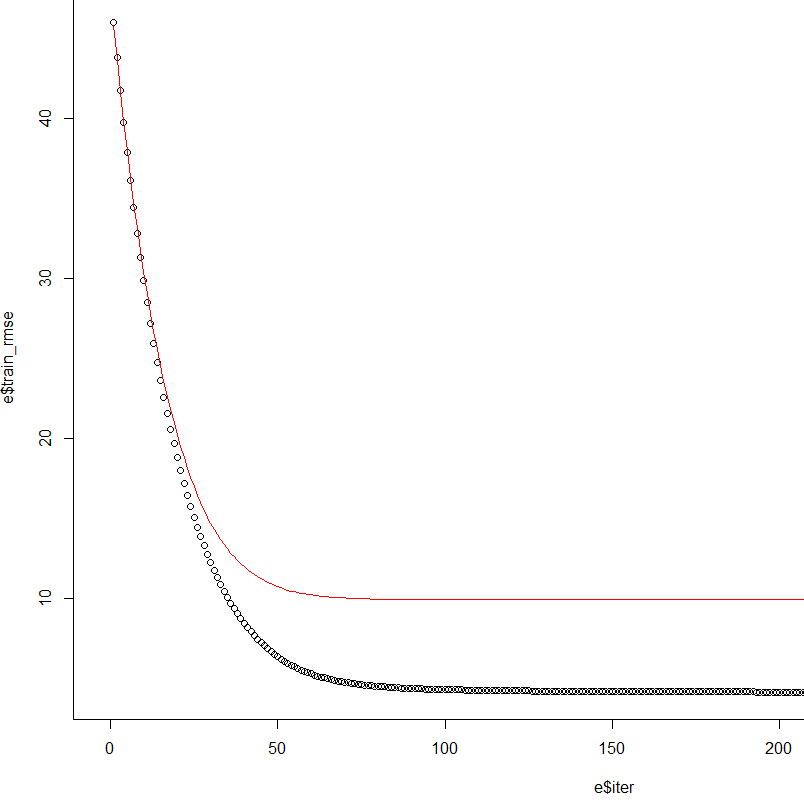
TSS=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))

R\_square=1-(RSS/TSS)

print(R\_square)

[1] 0.915577

Output



Observation

* As expected the training error reduces faster than Test error
* We also see that after a certain point the test error will not reduce any further and if we take a higher iterative value we could have over-fitting

Result

We selected the iteration as “134” through 10 fold cross validation , and were able to achieve an R square value of 91.54%

Partial Least Squares

It is a supervised alternative to PCR(Principal Component Regression) ,It is a dimension reduction method it Identifies new set of features (Zi) which are linear combinations of original features ,but unlike PCR ,PLS achieve these new set of features in a supervised way that is considering response

Hence, in computing Z1, PLS places the highest weight on the variables that are most strongly related to the response.

sc\_pls=**plsr**(critical\_temp**~**.,data=sc\_data1\_Train,scale=TRUE,validation="CV")  
**summary**(sc\_pls)

Output: The selection of number of components to be used for prediction is done using cross validation

pls\_pred=**predict**(sc\_pls,sc\_data1\_Test,ncomp=16)  
  
*#R^2*  
RSS=(**mean**((sc\_data1\_Test**$**critical\_temp**-**pls\_pred)**^**2))  
TSS=(**mean**((sc\_data1\_Test**$**critical\_temp**-**(**mean**(sc\_data1\_Test**$**critical\_temp)))**^**2))  
R\_square=1**-**(RSS**/**TSS)  
**print**(R\_square)

## Result

## ## [1] 0.6731818

## Support Vector Regression:

## It is difficult to predict a value which has infinite response values through decision boundary like the maximum margin hyper-planes(SVM) but still support vector machines can be extended to predict numerical values . In case of regression , a margin of tolerance (epsilon) is set in approximation to the SVM. However, the main idea is always the same: to minimize error, individualizing the hyper-plane which maximizes the margin, keeping in mind that part of the error is tolerated. We introduce slack variables ξn and ξ\*n for each point. This approach is similar to the “soft margin” concept in SVM classification, because the slack variables allow regression errors to exist up to the value of ξn and ξ\*n, yet still satisfy the required conditions.

## Code

# Specify variables to use  
sc\_data1 = sc\_data[,-c(1,2,3,4,6,10,11,13,14,16,17,20,21,23,24,26,30,31,32,33,35,37,40,41,42,45,48,49,50,52,53,54,56,58,59,61,63,65,68,70,72,74,75,77,78,80)]

# Split into training and testing data  
set.seed(1)  
split=sample.split(sc\_data1,SplitRatio = 0.8)  
split2 = sample.split(sc\_data1, SplitRatio = 0.5)  
sc\_data1\_Train=subset(sc\_data1,split=="TRUE")  
sc\_data1\_Test=subset(sc\_data1,split=="FALSE")  
sc\_tunedata\_Train=subset(sc\_data1,split2 == "TRUE")  
sc\_tunedata\_Test=subset(sc\_data1,split2=="FALSE")

# Tune the model on 50% of data  
OptModel = tune(svm, critical\_temp~., data = sc\_tunedata\_Train, ranges = list(epsilon=seq(0,1,0.1), cost = 2^(2:7)))  
  
# Plot RMSE vs epsilon and cost values  
plot(OptModel)

# Print the optimal model's maximum allowable error and cost parameters  
print(OptModel$best.parameters)

## epsilon cost  
## 46 0.1 64

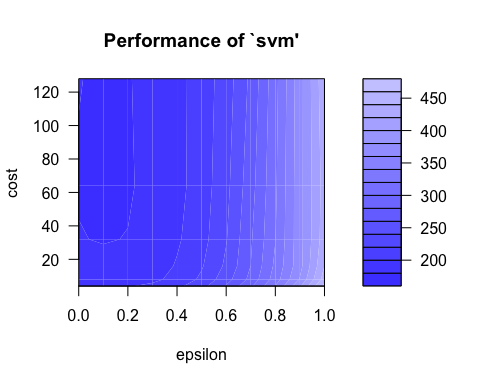
# Build model on 80% of data using optimal parameters found during tuning  
modelsvm2 = svm(critical\_temp~., data = sc\_data1\_Train, cost = 64, epsilon = 0.1)

# Make predictions on testing data (20% of data)  
predYsvm2 = predict(modelsvm2, sc\_data1\_Test)  
RSS2=(mean((sc\_data1\_Test$critical\_temp-predYsvm2)^2))  
TSS2=(mean((sc\_data1\_Test$critical\_temp-(mean(sc\_data1\_Test$critical\_temp)))^2))  
  
# Calculate R-squared of optimal model on testing data  
R\_square2=1-(RSS2/TSS2)  
print(R\_square2)

## [1] 0.8654372

* The tuning function builds a model, using the training data, for each combination of epsilon and cost values, cross-validates each model using 10-fold cross-validation, and chooses the model with the lowest RMSE as the optimal model
* The tuning was limited to 50% of the data because increasing the amount of data used for tuning increase the processing time exponentially and the machines used for running the codes were not capable enough to handle data of this magnitude.

## Output



* The darker shades indicate a lower RMSE, as indicated by the legend on the right-hand side of the plot

## Observation

* Based on the graph, the optimal epsilon lies around 0.1, and the cost value does not have as significant of an effect on the RMSE as does the epsilon value.

## Result

* The R2 predicted on the test data is 86.54%, a better result than all of the linear methods which were tested.
* The optimal epsilon value was found to be 0.1, meaning that the model has a very low margin of tolerance for errors.
* The optimal cost value was found to be 64, creating a relatively medium-sized margin for such a large set of data.