

Predicting the Critical Temperature of a Superconductor

Chandra Vamshi Dasyam(885931)

Muhammad Ahmad Falak(886100)

Index

- Goal
- Data source
- Data set features description
- Fitting two models
- Results
- Comparing two models

Goal

- Create two statistical models to predict T_c from elemental properties.
- Statistical model = Machine learning model
- Compare the two models

Data

- Data source: The UCI Machine Learning Repository is a collection of databases, domain theories, and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms.

- WebSite:
<https://archive.ics.uci.edu/ml/datasets/Superconductivity+Data>
- Has no Errors
- Has no NA data

Data set Description

We have data with 21,263 rows and 82 columns: 81 columns corresponding to the features extracted and 1 column of the observed Tc values.

Before discussing these 81 variable we need to look at the following tables

Variable	Units	Description
Atomic Mass	atomic mass units (AMU)	average of the 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)	distance from the nucleus to the outermost electron
Density	kilograms per meters 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/(m × K))	thermal conductivity coefficient κ
Valence	no units	typical number of chemical bonds formed by the element

Table 1: This table shows the properties of an element which are used for creating features to predict Tc.

Feature & Description	Formula	Sample Value
Mean	$= \mu = (t_1 + t_2)/2$	35.5
Weighted mean	$= \nu = (p_1 t_1) + (p_2 t_2)$	44.43
Geometric mean	$= (t_1 t_2)^{1/2}$	33.23
Weighted geometric mean	$= (t_1)^{p_1} (t_2)^{p_2}$	43.21
Entropy	$= -w_1 \ln(w_1) - w_2 \ln(w_2)$	0.63
Weighted entropy	$= -A \ln(A) - B \ln(B)$	0.26
Range	$= t_1 - t_2 \ (t_1 > t_2)$	25
Weighted range	$= p_1 t_1 - p_2 t_2$	37.86
Standard deviation	$= [(1/2)((t_1 - \mu)^2 + (t_2 - \mu)^2)]^{1/2}$	12.5
Weighted standard deviation	$= [p_1(t_1 - \nu)^2 + p_2(t_2 - \nu)^2]^{1/2}$	8.75

Table 2: This table summarizes the procedure for how feature extraction is made from material's chemical formula.

Here I will describe the feature extraction process through a detailed example:

Consider Re₇Zr₁ with T_c = 6.7 K, and focus on the features extracted based on thermal conductivity.

Rhenium and Zirconium thermal conductivity coefficients are $t_1 = 48$ and $t_2 = 23$ W/(m×K) respectively.

The ratios of the elements in the material are used to define features:

$$p_1 = 6/6+1=7, \quad p_2 = 1/6+1=7. \quad (1)$$

The fractions of total thermal conductivities are used as well:

$$w_1 = t_1/t_1+t_2 = 48/48 + 23$$

$$w_2 = t_2/t_1+t_2 = 23/48+23 \quad (2)$$

Now we need a couple of intermediate values based on equations (1) and (2):

$$A = p_1w_1/p_1w_1+p_2w_2 \approx 0.926,$$

$$B = p_2w_2/p_1w_1+p_2w_2 \approx 0.074.$$

Once we have obtained the values p_1, p_2, w_1, w_2, A , and B , we can extract 10 features from Rhenium and Zirconium thermal conductivities as shown in table (2).

We repeat the same process above with the 8 variables listed in table (1).

For example, for features based on atomic mass, just replace t_1 and t_2 with the atomic masses of Rhenium and Zirconium respectively, then carry on with the calculations of p_1, p_2, w_1, w_2, A, B , and finally calculate the 10 features defined in table (2). This gives us $8 \times 10 = 80$ features.

One additional features, a numeric variable counting the number of elements in the superconductor, is also extracted. We end up with 81 features in total.

In summary: We have data with 21,263 rows and 82 columns: 81 columns corresponding to the features extracted and 1 column of the observed Tc values.

One more example:

Example: Nb_{0.8} Pd_{0.2} with Tc = 1.98 K

Use the thermal conductivities of niobium and palladium to define a new “feature”: Mean thermal conductivity = $(54 + 71)/2 = 62.5 \text{ W}/(\text{m}\times\text{K})$

10	Feature & Description	
	Mean	
	Weighted mean	
	Geometric mean	
	Weighted geometric mean	
	Entropy	
	Weighted entropy	
	Range	
	Weighted range	
	Standard deviation	
	Weighted standard deviation	

8	Variable	
	Atomic Mass	
	First Ionization Energy	
	Atomic Radius	
	Density	
	Electron Affinity	
	Fusion Heat	
	Thermal Conductivity	
	Valence	

Final Data

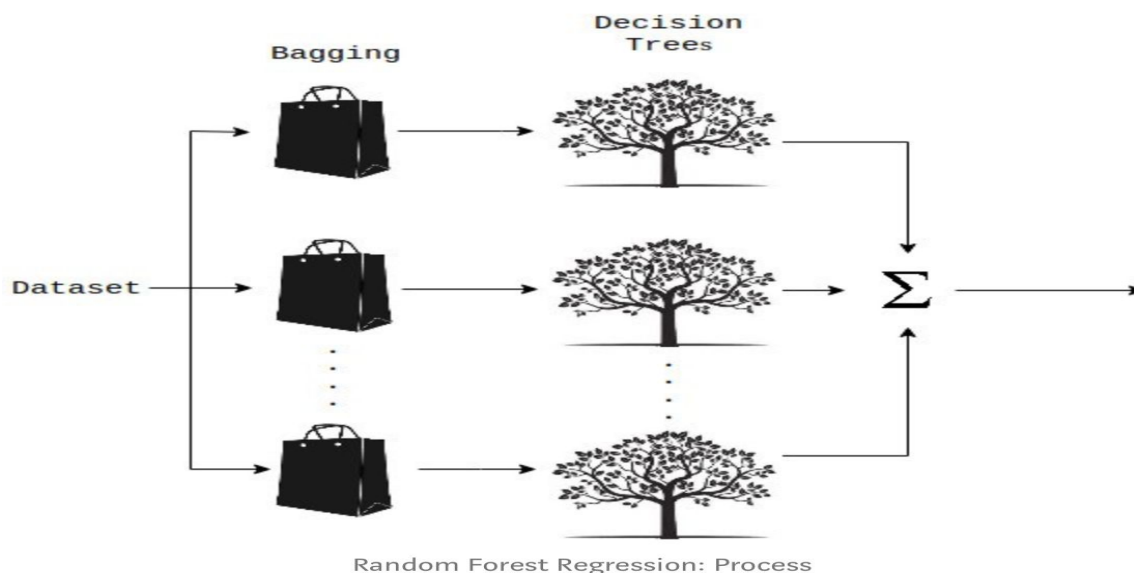
- Data size = 21263 rows and 82 columns
- 82 = 81 features extracted for each superconductor(10 features * 8 elemental properties plus 1 feature for total number of elements)+ T_c

Models

- Random Forest Tree Regression
- Convolution Neural Networks

Random Forest Tree Regression:

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called **Bootstrap Aggregation**, commonly known as **bagging**.



Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

The training algorithm for random forests applies the general technique of bagging, to tree learners.

Given a training set $X = x_1, \dots, x_n$ with responses $Y = y_1, \dots, y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples: For $b = 1, \dots, B$:

1. Sample, with replacement, n training examples from X, Y ; call these X_b, Y_b .
2. Train a classification or regression tree f_b on X_b, Y_b .
3. After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x' :

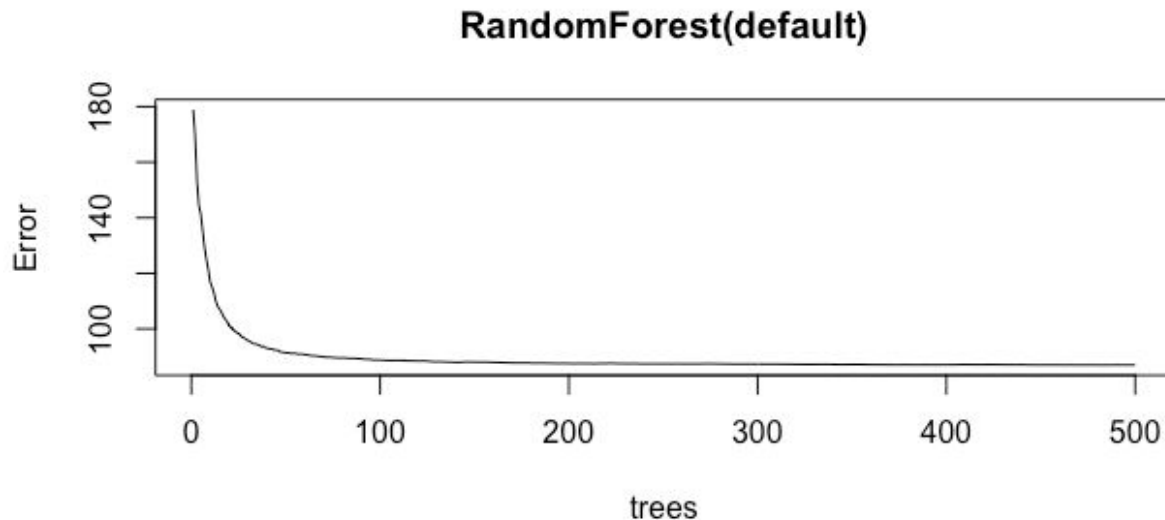
$$\hat{f} = \frac{1}{B} \sum_{b=1}^B f_b(x')$$

Prediction using random forest:

1. Loaded the data
2. Splitted the data into 80% training and 20%testing
3. Scaled the data

4. Default rf model is fitted on the train data($m=p/3$) and trees=500

5. Plot of the fitted model is shown below



From the above graph it is clear that after 200 trees there is no decrease in error rate.

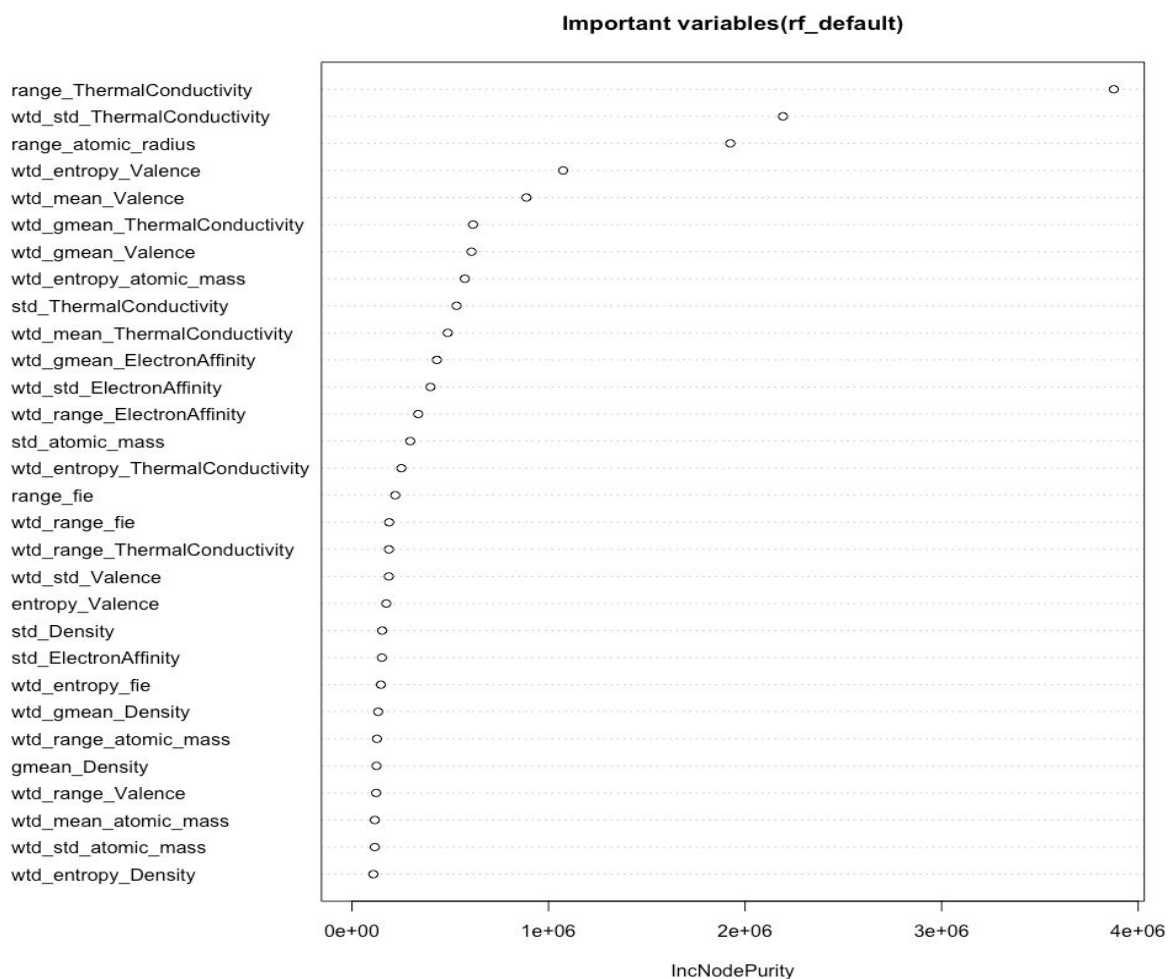
Let's look at the first six original values of the test dataset and predicted values of the test dataset

y	19	23	11	36	27	82
<input type="checkbox"/> y	23	19	56	27	24	67

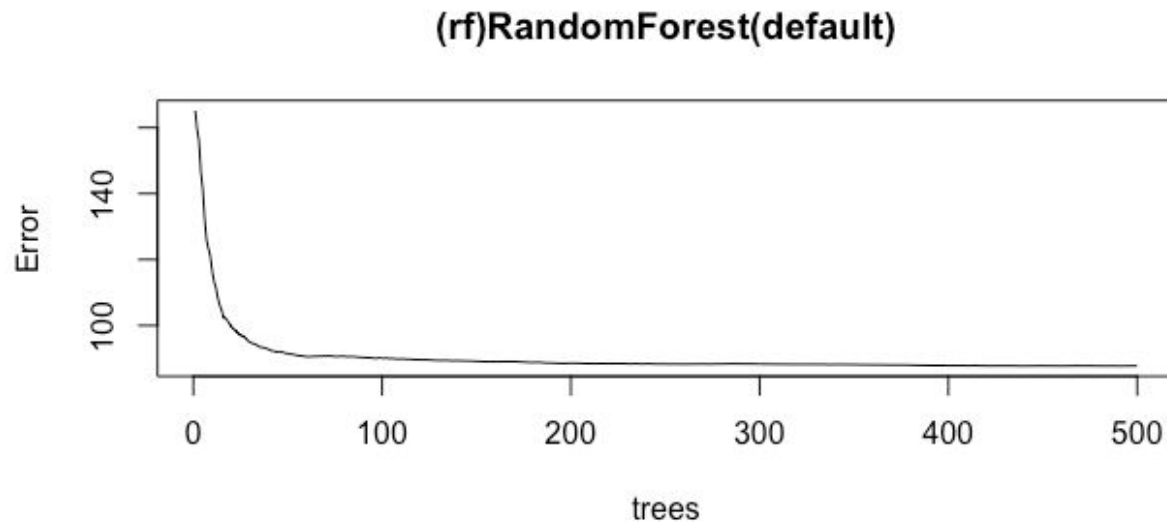
As our dataset has 81 variable which is huge so we decided to use variable importance and use only 30 most important variables from the dataset.

Variable importance is measured by recording the decrease in MSE each time a variable is used as a node split in a tree. The remaining error left in predictive accuracy after a node split is known as node impurity and a variable that reduces this impurity is considered more important than those variables that do not. Consequently, we accumulate the reduction in MSE for each variable across all the trees and the variable with the greatest accumulated impact is considered the more important, or impactful.

The below plot shows the most important variables



Now we will use only these variables from our dataset for fitting



Now Let's look at the first six original values of the train dataset and predicted values of the train dataset

y	19	23	11	36	27	82
<input type="checkbox"/> y	25	20	54	30	26	65

Out of bag error (OOB)

The one of the important features of random forest is out of bag error. In simple words oob is suppose there are five samples but tree is trained on four samples now this trained tree is used to predict the value of that fifth sample. It is like validation.

If there are N rows in the training data set. Then, the probability of not picking a row in a random draw is

$$\frac{N-1}{N}$$

Using sampling-with-replacement the probability of not picking N rows in random draws is

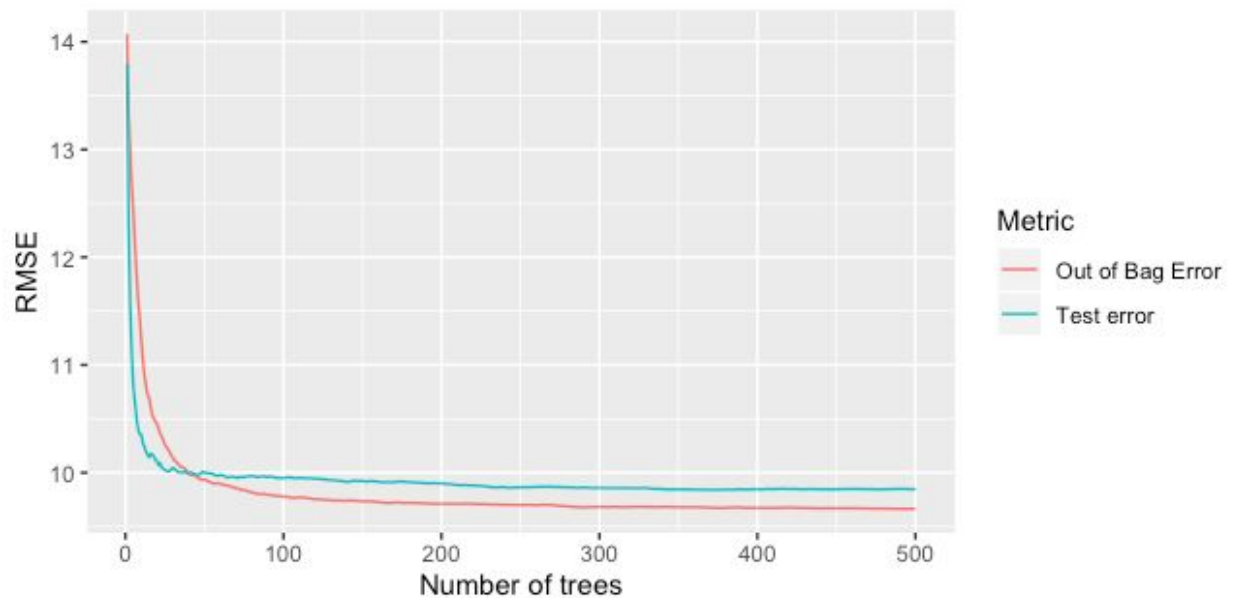
$$\left(\frac{N-1}{N}\right)^N$$

which in the limit of large N becomes equal to

$$\lim_{N \rightarrow \infty} \left(1 - \frac{1}{N}\right)^N = e^{-1} = 0.368$$

Therefore, about 36.8 % of total training data are available as OOB sample for each tree and hence it can be used for evaluating or validating the random forest model.

Random forest also allows us to use a validation set to measure predictive accuracy if we did not want to use the OOB samples. Here we split our training set further to create a training and validation set. We then supply the validation data in the `xtest` and `ytest` arguments.



From the above plot it is clear that using oob is better to reduce the error of the predictions.

Tuning the model

Random forests are fairly easy to tune since there are only a handful of tuning parameters(hyperparameters).

Eg: ntree, mtry, sampsize, nodesize, maxnodes.

So let's try to find the best mtry parameter.

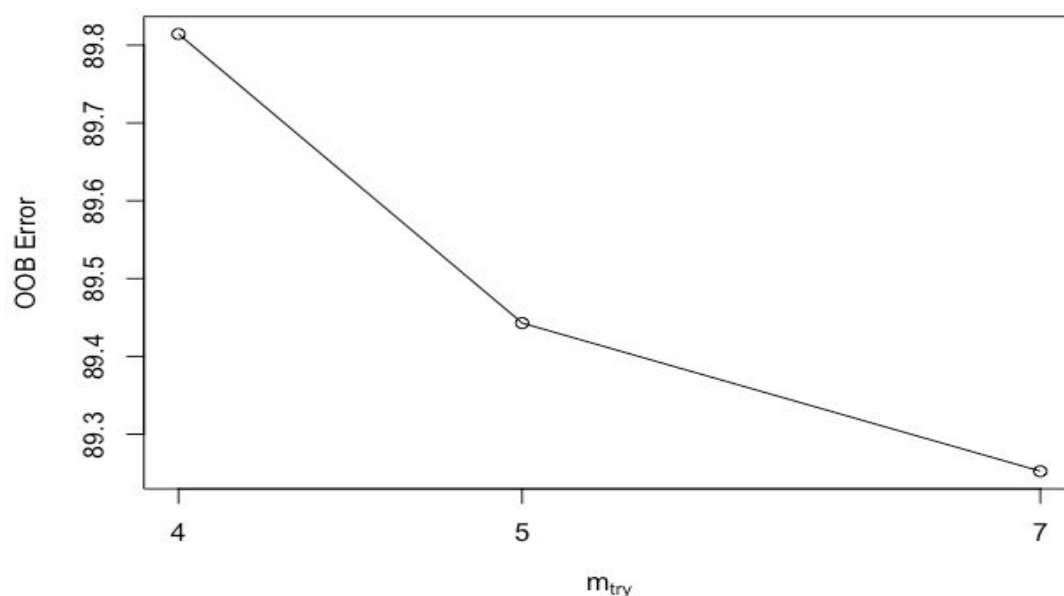
So for tuning mtry parameter random forest provides **tuneRf** for a quick assessment of mtry.

mtry: The number of variables to randomly sampled as candidates at each split.

Sampsize: The number of samples to train on. The default value is 63.25% of the training set.

Node size: minimum number of samples within the terminal nodes.

Lets starts with $m_{try} = 5$ and increases by a factor of 1.5 until the OOB error stops improving by 0.4%.



The best value for hyperparameter m_{try} is 7 because if we have lowest oob error if m_{try} is 7

Full Grid Search with Ranger

Ranger is a fast implementation of random forests, particularly suited for high dimensional data.

Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary.

Grid-searching does NOT only apply to one model type.

Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model. It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run. Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination.

Grid-searching

Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary.

Grid-searching does NOT only apply to one model type.

Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model. It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run. Grid-Search will build a model on each parameter combination possible. It iterates through every


parameter combination and stores a model for each combination.

After performing grid search operation on various mtry , node size, sample size we got the following values as optimized values for this hyper parameters.

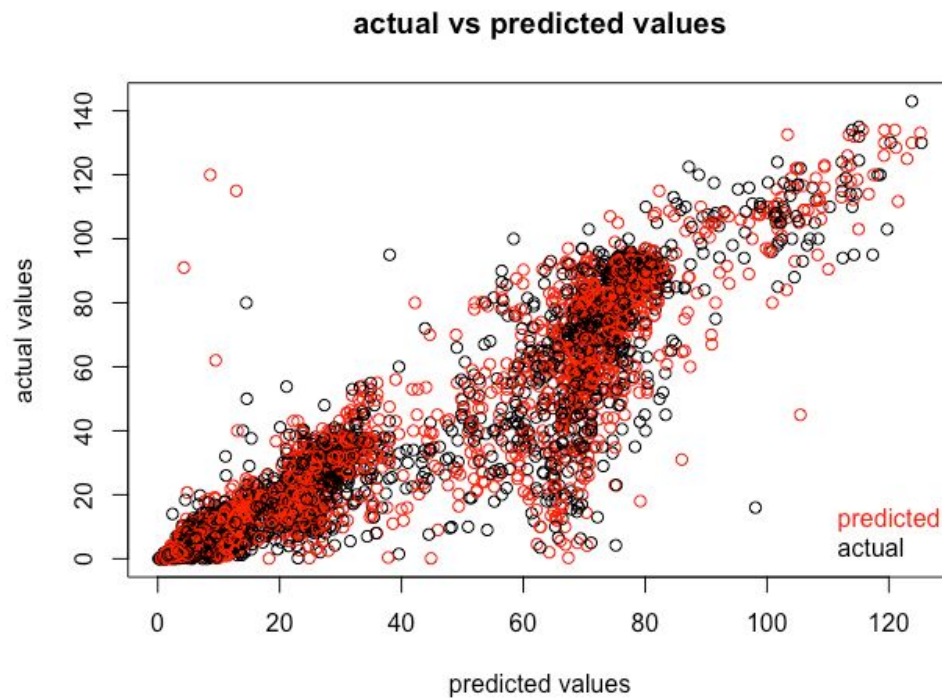
Mtry :19, node_size: 3, sampe_size: 0.8(80%)

Now fit the random forest by using above optimal hyperparameters.

Prediction on the test dataset

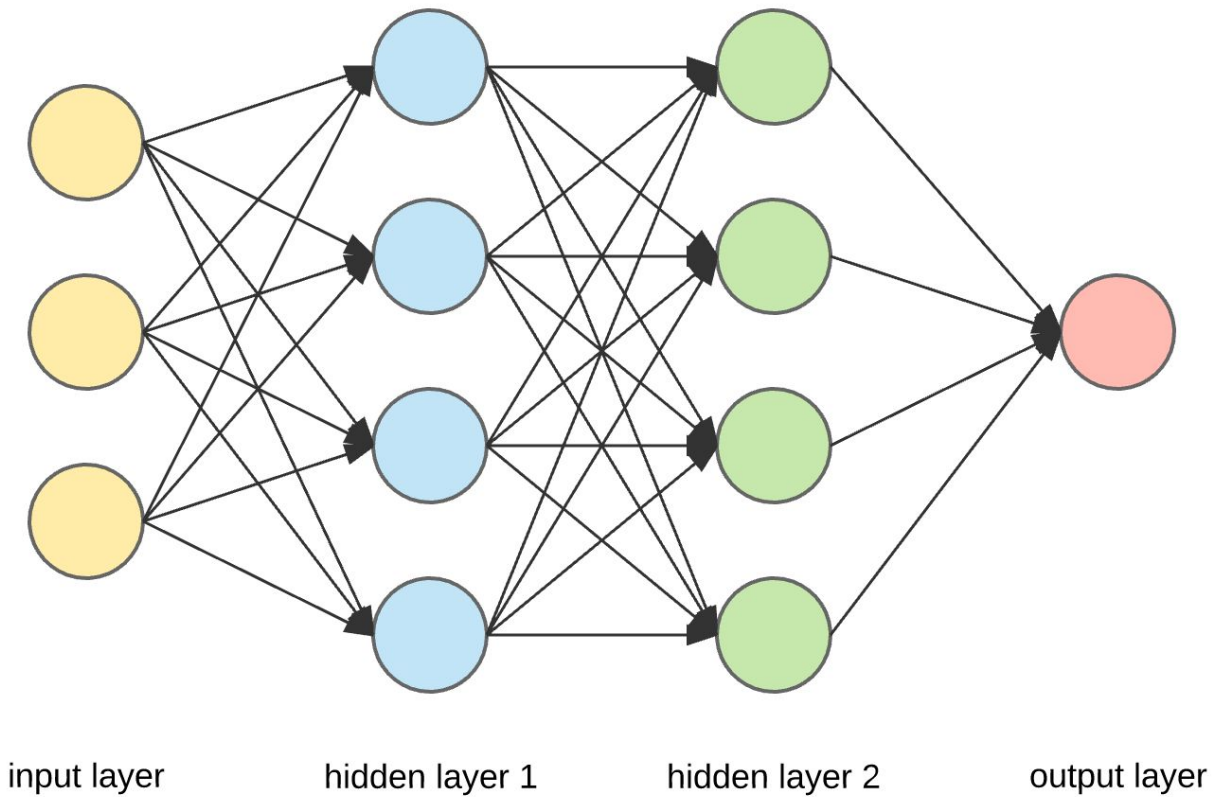
y	26	82	30	23	85
 y	40	73	29	26	78

mse:164



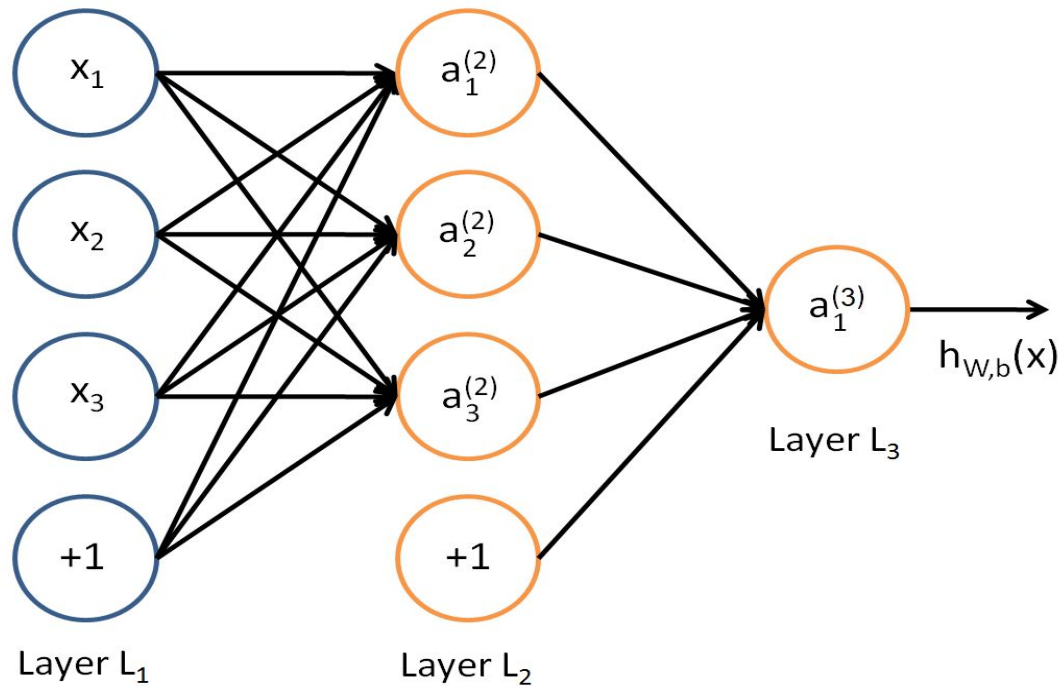
Neural Network:

Artificial Neural Network is computing system inspired by biological neural network that constitute animal brain. Such systems “learn” to perform tasks by considering examples, generally without being programmed with any task-specific rules.



The Neural Network is constructed from 3 types of layers:

- Input layer — initial data for the neural network.
- Hidden layers — intermediate layer between input and output layer and place where all the computation is done.
- Output layer — produce the result for given inputs.



The above model have 4 input nodes (3 + 1 “bias”). One hidden layer with 4 nodes (3 + 1 “bias”) and one output node.

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

X is the input layer an A is the hidden layer

Now we will look at the weight matrix which is usually denoted by θ

$$\theta^{(1)} = \begin{bmatrix} \theta_{10} & \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{20} & \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{30} & \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix}$$

Next, we will compute the “activation” nodes for the hidden layer. In order to do that we need to multiply the input vector X and weights matrix θ^1 for the first layer ($X * \theta^1$) and then apply the activation function g . Then the result is

$$\begin{aligned}a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)\end{aligned}$$

And by multiplying hidden layer vector with weights matrix θ for the second layer ($A * \theta$) we get output for the hypothesis function:

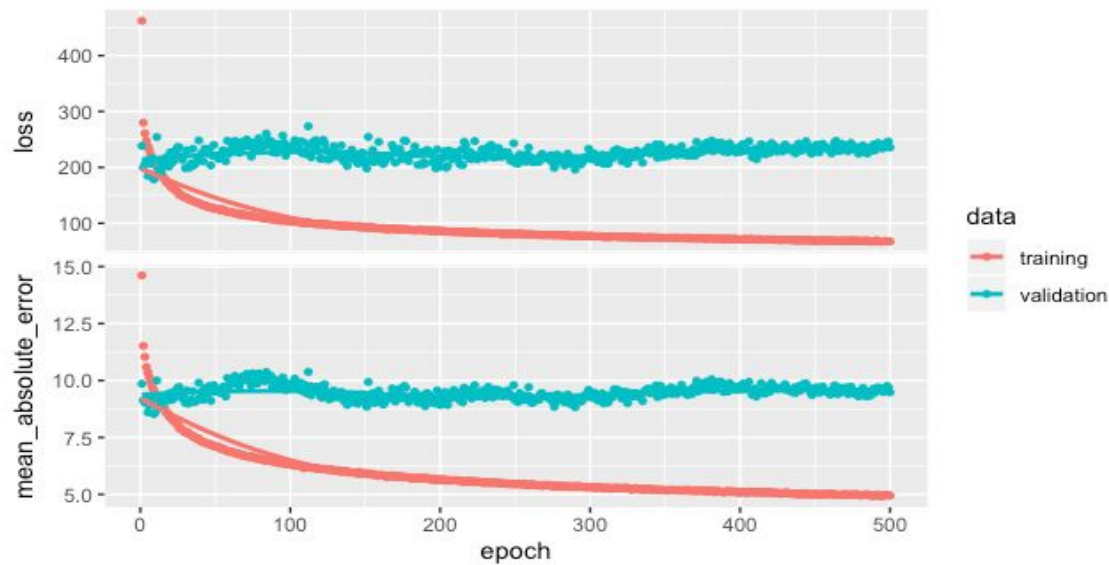
$$h_{\theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

Prediction using Neural network:

- Create the instance of keras model
- Set the layers and units
- Fit the model

Fitted the model with 3 hidden layers with 128 units, 64 units, 16 units, one input layer and one output layer. Batch size of 32 and epoch = 500. Used loss = mse and relu activation function because we are predicting discrete value.

Plots for the above model

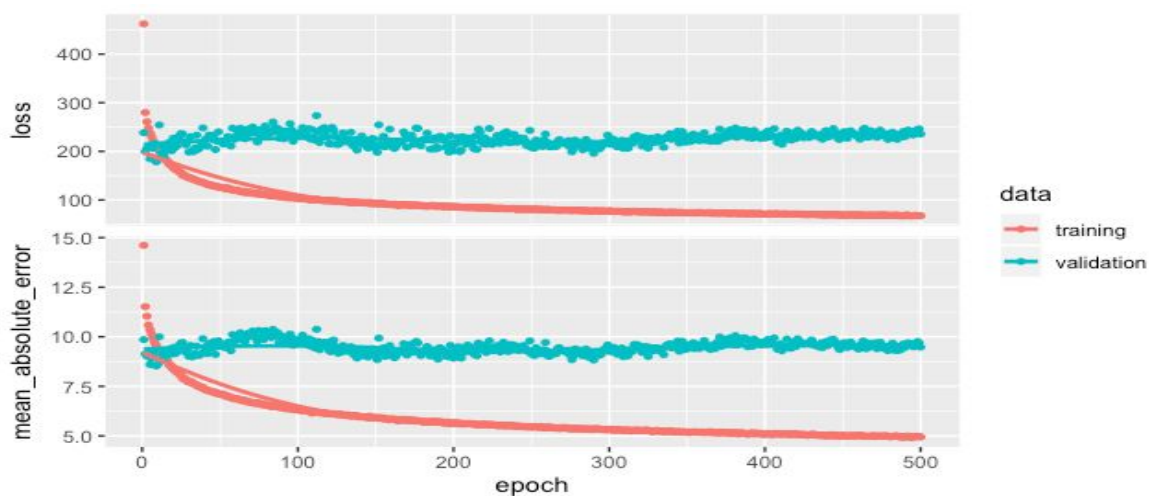


The above plot is loss and mean absolute error on training and validation set. Loss is decreasing as no.of epoch increasing and mae is at constant position.

The **mse** of the model on test is around 161.

So increased one more layer with 32 units and fitted model.

This time the model was improved and the **mse** of the model on the test is 147, improved a lot.



Predictions on the test dataset using neural network

y	26	82	30	23	85
^[?] y	28	80	26	28	98

mse:147

Comparison of random forest and neural network:

Both models are good at prediction but neural network gave better predictions than random forest.

And also the mean square error of neural network is lower than random forest.

y	26	82	30	23	85
^[?] y(random forest)	40	73	29	26	78
^[?] y(neural network)	28	80	26	28	98

	Random forest	Neural network
mse(test data set)	164	147
rmse	12.8	12.1

Prediction using material name

```
predict_tc_rf("Sr0.1La1.9Cu1O4")
```

```
predict_tc_nn("Sr0.1La1.9Cu1O4")
```

Material name	Actual temperature	Random forest	Neural network
Sr0.1La1.9Cu1O4	33	54	30

Neural network gives the best results.

Comments

- Features extracted based on thermal conductivity, atomic radius, valence, electron affinity, and atomic mass contribute the most to the model's predictive accuracy.
- Many potential variables are missing: pressure, crystal structure, manufacturing method, etc.