Predicting the critical temperature for superconductivity-Code

Fatma Mahfoudh & Sonia Aloui

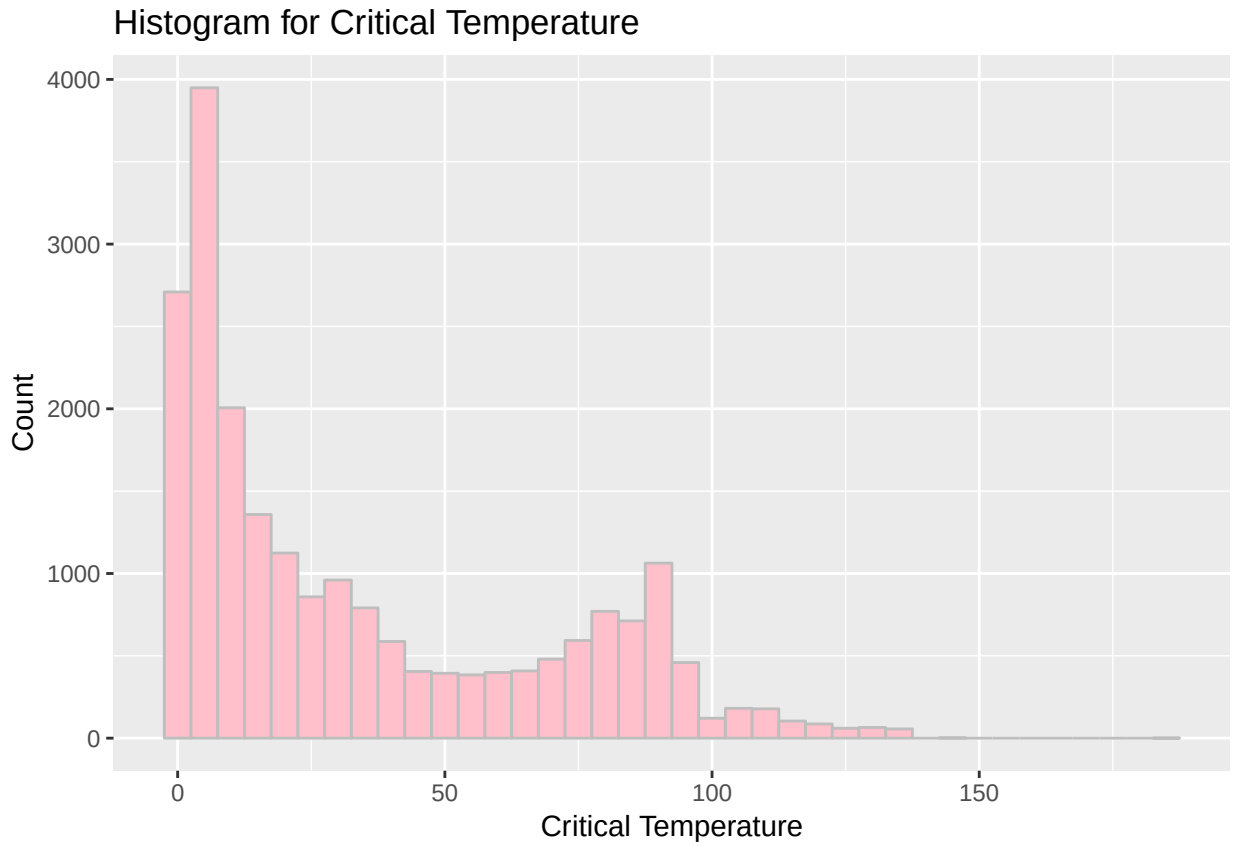
January 6, 2019

Data loading

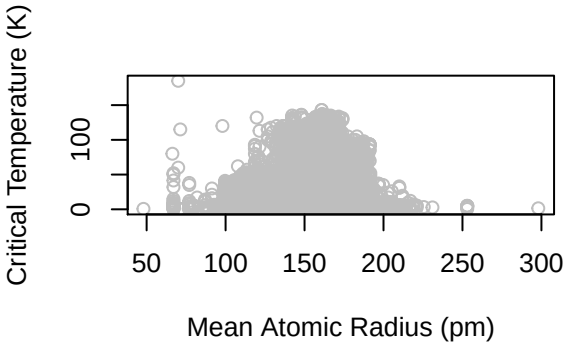
```
cond <- read.csv('train.csv',encoding='utf-8')
dim(cond)
```

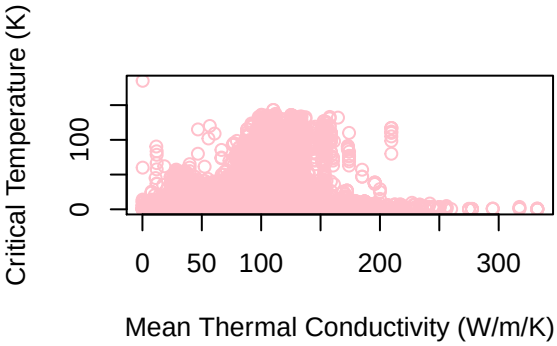
```
## [1] 21263    82
```

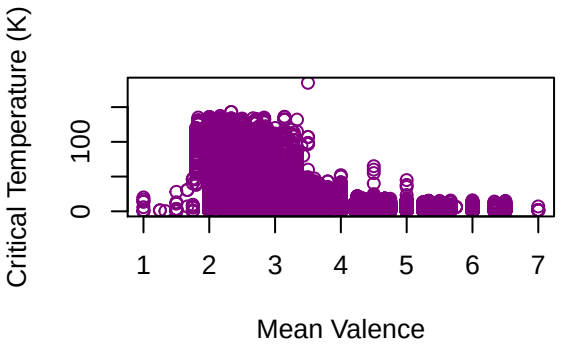
```
library(ggplot2)
ggplot(cond, aes(cond$critical_temp)) + geom_histogram(col="gray",fill="pink",binwidth=5) +
  labs(title="Histogram for Critical Temperature") +
  labs (x="Critical Temperature", y="Count")
```

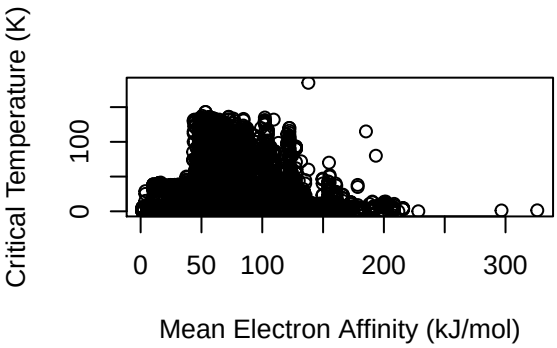



```
par(mfrow=c(2,2))
plot(cond$mean_atomic_radius,cond$critical_temp,xlab="Mean Atomic Radius (pm)",ylab="Critical Temperatu
plot(cond$mean_ThermalConductivity,cond$critical_temp,xlab="Mean Thermal Conductivity (W/m/K)",ylab="Cr
plot(cond$mean_Valence,cond$critical_temp,xlab="Mean Valence",ylab="Critical Temperature (K)",col="#800
plot(cond$mean_ElectronAffinity,cond$critical_temp,xlab="Mean Electron Affinity (kJ/mol)",ylab="Critical
```










```
sum(is.na(cond))
```

```
## [1] 0
```

There are no missing values in our dataset. We will move on directly to building the machine learning model since the data is already preprocessed.

Data Splitting

We split the data into training set (2/3) and testing set (1/3).

```
library(lattice)
library(caret)
set.seed(45)
train_index <- createDataPartition(cond$critical_temp, p =2/3, list = FALSE)
train <- cond[ train_index,]
test  <- cond[-train_index,]

X_train <- train[names(train)!="critical_temp"]
y_train <- train$critical_temp

X_test <- test[names(test)!="critical_temp"]
y_test <- test$critical_temp
```

Training models

We are going to build our models with the training set using K-fold cross validation (K=5). The machine learning models compared are: multiple linear regression, regression trees, bagging, random forest and gradient boosting. The function below takes time to give results.

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------- tidyverse 1.2.
```

```
## v tibble  1.4.2     v purrr   0.2.5
## v tidyr   0.8.2     v dplyr   0.7.8
## v readr   1.3.1     v stringr 1.3.1
## v tibble  1.4.2     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------------------------- tidyverse_conflicts()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

```
library(rsample)
```

```
##
## Attaching package: 'rsample'
```

```
## The following object is masked from 'package:tidyr':
##
##     fill
```

```
library(tree)
library(ipred)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(tidyverse)
library(caret)
library(xgboost)
```

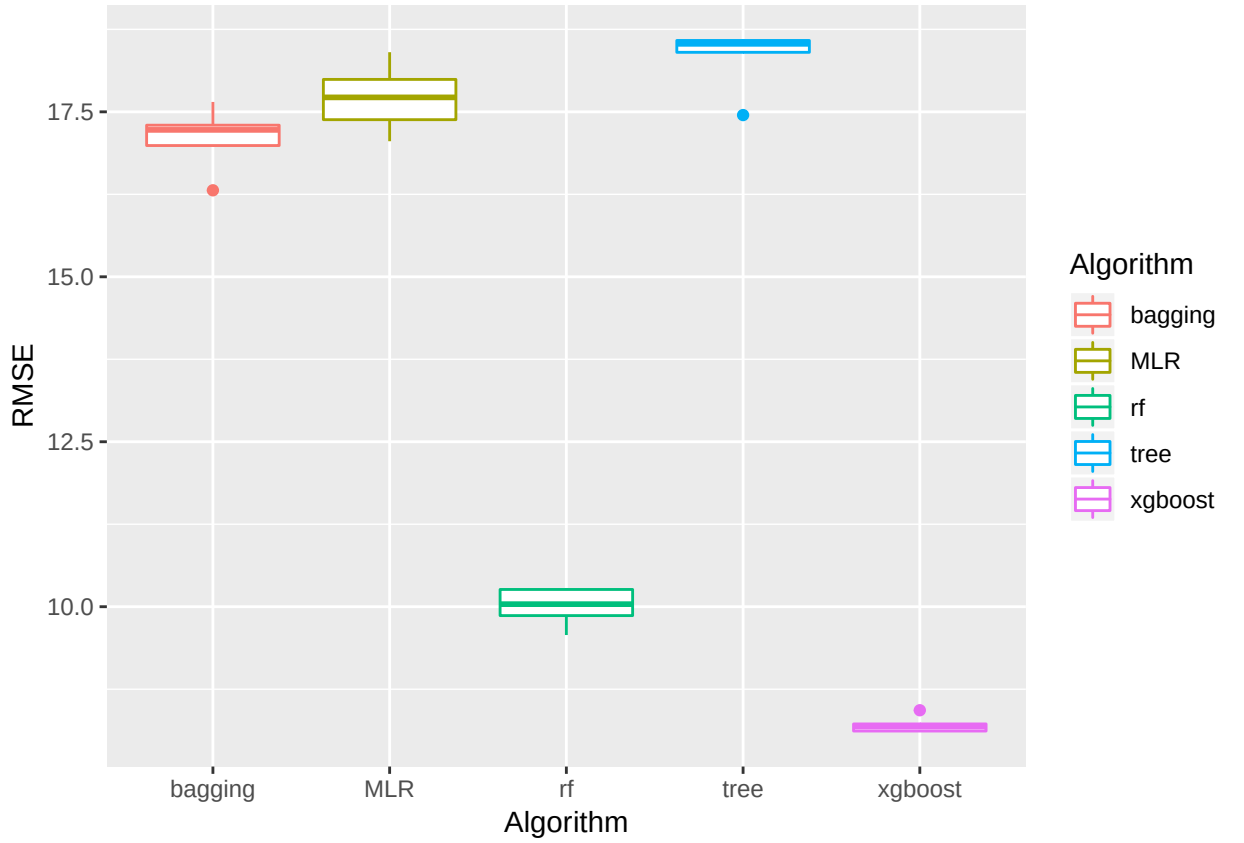
```
##
```

```
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
set.seed(45)
fold <- vfold_cv(train, v=5)
performance <- map_dfr(fold$splits,function(x)
  {
  x_test <- as_tibble(x,data="assessment")
  x_train <- as_tibble(x,data="analysis")

  #lm
  y_pred_lm <-predict(lm(critical_temp ~.,data=x_train),newdata=x_test)
  rmse_lm <-sqrt(mean((x_test$critical_temp-y_pred_lm)^2))

  #tree
  y_pred_tree <- predict(tree(critical_temp~.,data=x_train),newdata=x_test)
  rmse_tree <- sqrt(mean((x_test$critical_temp-y_pred_tree)^2))

  #bagging
  y_pred_bag <- predict(bagging(critical_temp~.,data=x_train,coob=T),newdata=x_test)
  rmse_bag <-sqrt(mean((x_test$critical_temp-y_pred_bag)^2))

  #random forest
  y_pred_rf <- predict(randomForest(critical_temp~.,data=x_train,coob = T),newdata=x_test)
  rmse_rf <- sqrt(mean((x_test$critical_temp-y_pred_rf)^2))

  #gradient boosting
  y_pred_xg <- predict(xgboost(data=as.matrix(x_train),label=x_train$critical_temp,verbose=0,
                               nrounds=5,objective="reg:linear"),
                       newdata=as.matrix(x_test))
  rmse_xg <- sqrt(mean((x_test$critical_temp-y_pred_xg)^2))

  tibble(MLR=rmse_lm,tree=rmse_tree,bagging=rmse_bag,rf=rmse_rf,xgboost=rmse_xg)
  }
  )
performance <- gather(performance,key = "Algorithm",value = "RMSE")
```

```
ggplot(performance,aes(x=Algorithm,y=RMSE,col=Algorithm))+geom_boxplot()
```



Gradient boosting is the best performing model with a rmse ~ 8K. However, the model was trained using the default parameters. We can have a lower rmse using appropriate parameters. To find them out, we will perform, next, hyperparameter tuning.

Hyperparameter Tuning

We create a grid using different values for η, column subsampling (colsample_bytree), subsample ratio (subsample), maximum depth of a tree (max_depth) and minimum child weight (min_child_weight). These parameters are defined in the report. After that, we apply xgboost using the combinations of parameters and measure rmse. Those giving the lowest rmse are selected for our model. This task takes quite a long time.

```
args <- expand.grid(eta=c(0.01,0.015,0.02),colsample_bytree=c(0.25,0.5,0.75),subsample=0.5,min_child_wei

xgb <- function(eta,cs,ss,cw,max){

  y_pred <- predict(xgboost(data=as.matrix(train),
                            label=train$critical_temp,                    params=list(eta=eta,colsamp
                            verbose=0,nrounds=300),
                    newdata=as.matrix(test))
  rmse_val <- sqrt(mean((test$critical_temp-y_pred)^2))
  return(c(rmse_val,eta,cs,ss,cw,max))
}

rmse_mat <-mapply(FUN=xgb,eta=args$eta,cs=args$colsample_bytree,ss=args$subsample,cw=args$min_child_weig

rmse_mat <- t(rmse_mat)
```

```
rmse_mat <- as.data.frame(rmse_mat)
names(rmse_mat) <- c('rmse','eta','colsample_bytree','subsample','min_child_weight','max_depth')
best_param <- rmse_mat[rmse_mat$rmse==min(rmse_mat$rmse),]
best_param
```

```
##         rmse  eta colsample_bytree subsample min_child_weight max_depth
## 126 0.945834 0.02             0.75       0.5               10        21
```

```
best_xgb <- xgboost(data=as.matrix(train),label=train$critical_temp,params=list(eta=best_param$eta,colsa
```

Testing the model

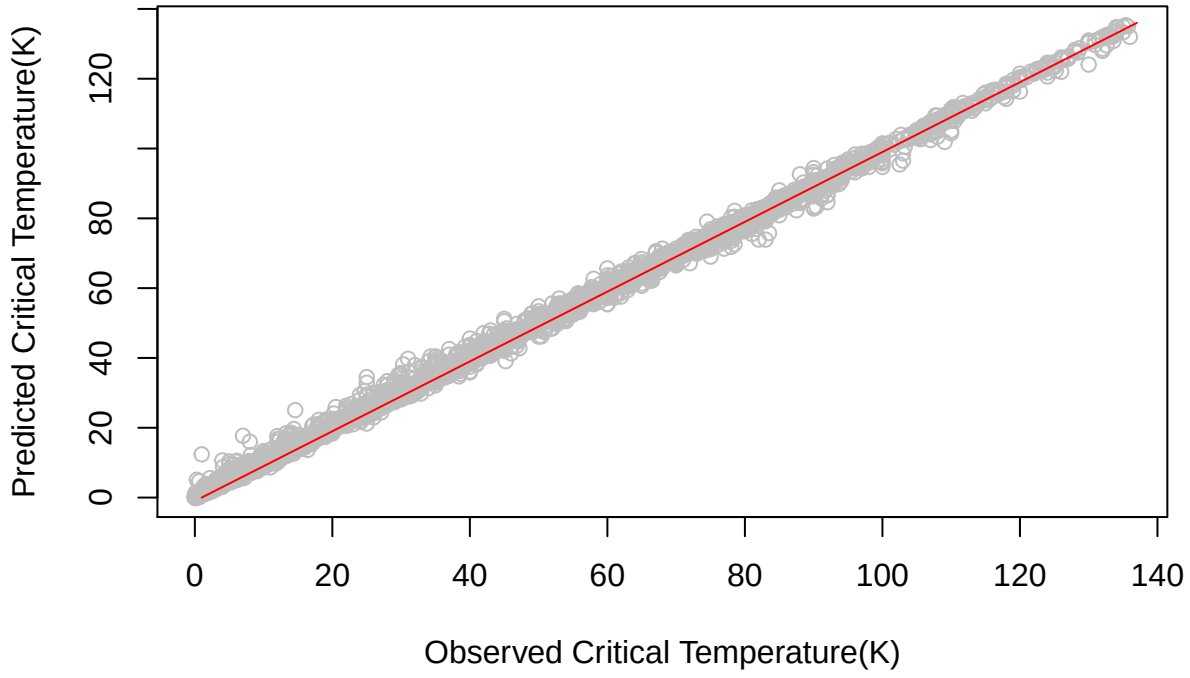
Let's apply our new model on the test set and measure the rmse.

```
y_pred <- predict(best_xgb,newdata=as.matrix(test))
sqrt(mean((y_pred-test$critical_temp)^2))
```

```
## [1] 1.133743
```

The result is clearly better than the one obtained with default parameters. And, to conclude, we compare all values of predicted and observed critical temperature in the figure below (test set).

```
plot(test$critical_temp,y_pred,xlab='Observed Critical Temperature(K)',ylab='Predicted Critical Tempera
lines(0:max(test$critical_temp),col='red')
```



The predicted and observed values ranging between 0K and 50K are very close. Beyond 50K, we see clearly the differences which are due, maybe, to the fact that there are not enough samples with $T_c > 50K$ in the training set.