Insights into the Air Quality in Beijing

Jiali Luan

6/14/2017

Abstract:

In order to facilitate our understanding of the severe air pollution in Beijing, I established several models based on the data of PM 2.5 Index and meteorological records in Beijing from 2010-2014 to predict future PM 2.5. Four classification methods, namely, KNN, decision tree, random forest and logistic regression, were adopted separately. As the result, the random forest and the KNN methods give the least test and train errors when predicting PM 2.5 for this particular dataset, presumably suggesting that these two methods generate more accurate prediction for future PM 2.5 Index.

Introduction:

Beijing's air pollution poses a big threat to the health of local residents. The PM 2.5 Index has been known as an important indication of the air quality. Exploring an effective way of predicting PM 2.5 Index will be an important step towards addressing the pollution. However, various factors such as dew point, temperature, pressure, etc. may influence PM 2.5 and the relationship between them remains unclear. Thus, it is challenging to accurately predict PM 2.5. To address this problem, statistical modeling including KNN, decision tree, random forest, and logistic regression can be adpoted since they are particularly useful for classifying response variable (ie. air quality) based on the numerous explanatory variables (meteorological records).From UCI Machine Learning Repository, I selected the hourly dataset PM 2.5 Index with the corresponding meteorological records in Beijing from 2010-2014 to predict the air quality is either healthy or unhealthy. Based on the result of each modeling method, the KNN and random forest output the lowest error rate in both testing and training cases. However, for the KNN method,the testing error and training error have a slightly bigger difference, presumbaly due to overfitting.Besides, since binary class of the response variable such as "healthy" and "unhealthy" is not informative, I further detailed the classes(Good,Moderate,Unhealthy for Sensitive Group,Unhealthy,Very Unhealthy,Hazardous).The increased variety compromises the accuarcy of prediction though.

Data Description:

The PM 2.5 Index dataset includes 43824 observations and 12 attributes. Among these attributes, 11 are numeric and 1 is categorical. Here is a list of the atrributes' information: year: year of data in this row, month: month of data in this row, day: day of data in this row, hour: hour of data in this row, pm2.5: PM2.5 concentration (ug/m^3), DEWP: Dew Point (Celcius),TEMP: Temperature (Celcius), PRES: Pressure (hPa),cbwd: Combined wind direction (cv(calm and variable),NW(northwest, covering E,ESE,SE,SSE and S), NE(northeast,covering NNE, NE and ENE),SE(southeast,covering E, ESE, SE, SSE and S)), Iws: Cumulated wind speed (m/s), Is: Cumulated hours of snow, Ir: Cumulated hours of rain.

```
library(tree)
library(plyr)
library(dplyr)
library(randomForest)
library(class)
library(magrittr)
library(ggplot2)
library(corrplot)
library(cowplot)
```

```
library(gmum.r)
library(kknn)
library(nnet)
library(factoextra)
```

Data Process:

I created two empty tables to record the test error and validation error for each modeling methods that I used in binaray class case and multiple class case.

```
records.bin = matrix(NA, nrow=4, ncol=2)
colnames(records.bin) <- c("train.error(bin.)","test.error(bin.)")
rownames(records.bin) <- c("tree","knn","logistic","random forest")
records.mul = matrix(NA, nrow=4, ncol=2)
colnames(records.mul) <- c("train.error(mul.)","test.error(mul.)")
rownames(records.mul) <- c("tree","knn","random forest","N/A")
```

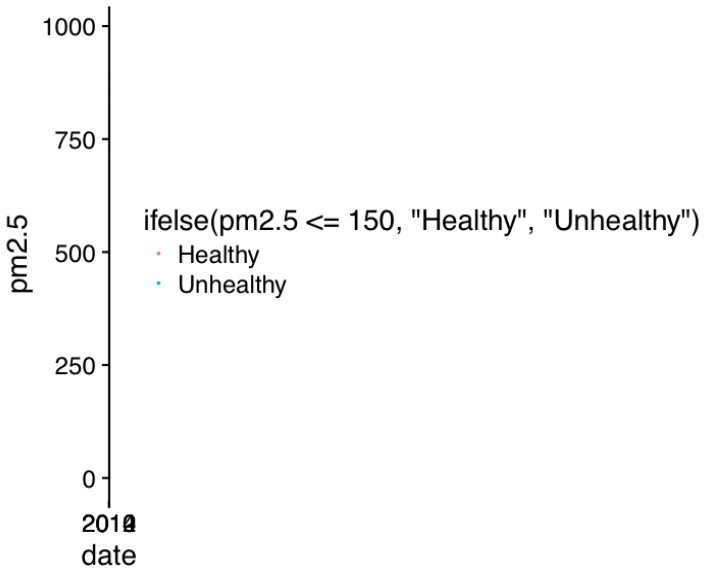
```
erate<-function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}
```

First, I loaded the dataset and removed the observations that have missing values. There are 2067 rows with missing values. Deleting the rows that have imcomplete information is an important step and imporve the accuracy. For the binary classfication, I assigned "healthy" label to the PM 2.5 that is less than or equal to 150. For anything number above 150, they are classified as "unhelthy". I made a plot to give a general idea about how bad the airquality is in Beijing. One can see that the maximum PM 2.5 is almost 1000. In the standard scale provided by the government, the maximum is 500. There are 8991 hours in Beijing that the PM 2.5 exceeds the limit on the standard scale. In the next step, I sclaed all the numeric attributes except the "year","month","day",and "hour". Scaling makes more sense when we have multiple variables that have different order of magnitude.

```
AQ<-read.csv("Beijing.csv")
suppressPackageStartupMessages(attach(AQ))
sum(is.na(AQ))
```

```
## [1] 2067
```

```
AQ=na.omit(AQ)
AQ=AQ%>%
  filter(complete.cases((.)))
AQ$date=as.Date(paste(AQ$year, AQ$month, AQ$day, sep='-'))
AQ$date_time=as.POSIXct(paste(AQ$date, AQ$hour), format="%Y-%m-%d %H")
general_plot <- ggplot(AQ,aes(date, pm2.5,color=ifelse(pm2.5<=150,"Healthy","Unhealthy")))+
  geom_point(size=0.2)
general_plot
```



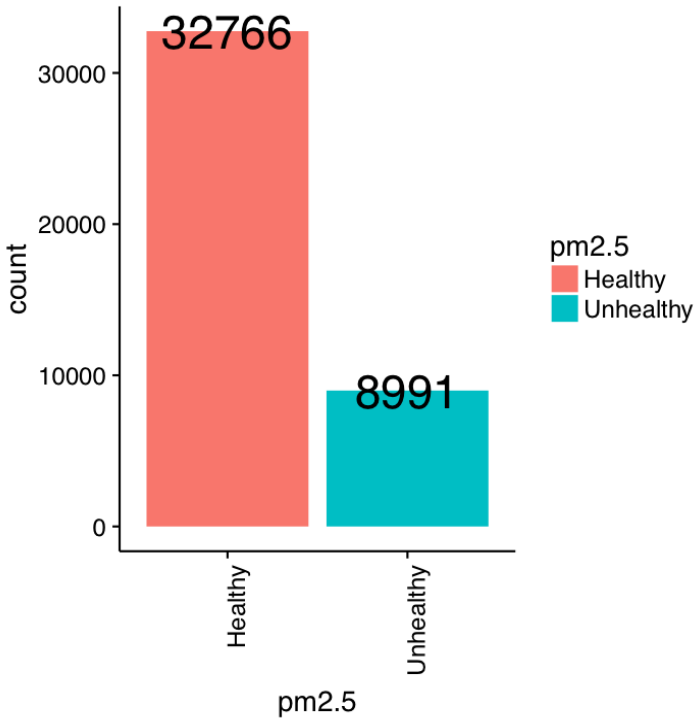

```
nrow(filter(AQ,pm2.5>150))
```

```
## [1] 8991
```

```
AQ=AQ%>%
  mutate(pm2.5=as.factor(ifelse(pm2.5<=150,"Healthy","Unhealthy")))
AQ=AQ[-c(1)]
AQ[c(6:8,10:12)]=scale(AQ[c(6:8,10:12)])
```

This bar graph visually shows the imbalance of this data set.

```
set.seed(1)
count_plot <-
  ggplot(AQ,aes(x=pm2.5,fill=pm2.5)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=8,stat ="count", aes(label = ..count.., y = ..count..))
count_plot
```



I previously mentioned about the four attributes "year", "month", "day", and "hour" that are related to time. I choose to remove the factor "year" since it lacks the significance of comparability. However, the rest three factors should also be included in modeling because they can answer questions such as "How seasons can affect air quality?", "Does air pollution tend to be low at night?", "Do we tend to have better air quality during holidays?(Many Chinese national holidays are in the beginning of each month)" Therefore, I converted the integers into classes for each attributes. For the "day" factor, I divided them into "Early Month", "Mid Month", and "Late Month". For the "month" factor, I labeled them with four seasons. Lastly, for the "hour" factor, I put them into "Wee", "Morning","Afternoon" and "Night".

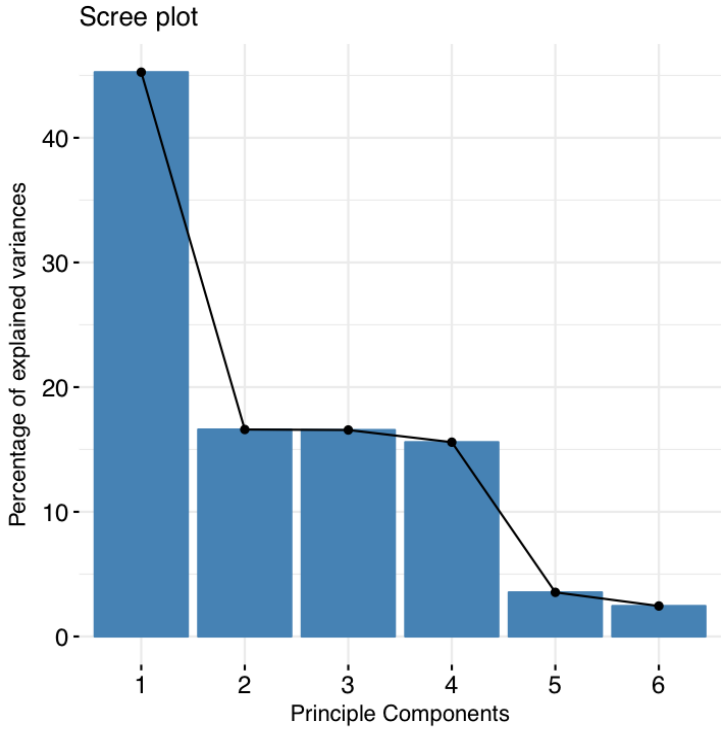
```
AQ$day<- cut(AQ$day, breaks=c(0,10,20,31), labels=c("Early Month","Mid Month","Late Month"))
AQ$month<- cut(AQ$month, breaks=c(0,3,6,9,12), labels=c("Spring","Summer","Fall","Winter"))
AQ$hour<- cut(AQ$hour, breaks=c(-1,6,12,18,23), labels=c("Wee","Morning","Afternoon","Night"))
AQ=AQ[-c(1,13,14)]
```

I conducted a summary of the Principal Component Analysis. It shows that four principal components includes 94% information about the dataset. The attributes "TEMP","DEWP","PRES" are the important variables of PC1 and the attributes "ls" and "lr" are the important variables of PC2.

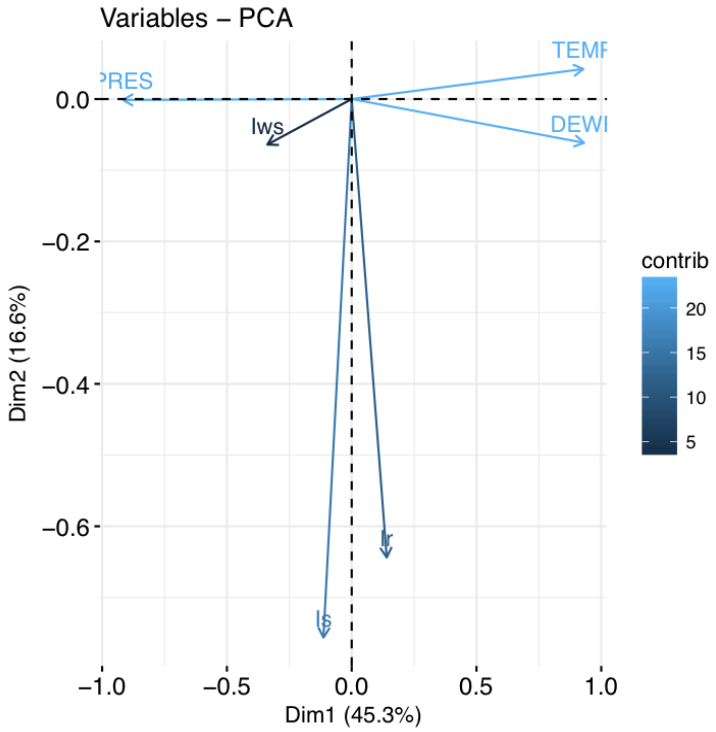
```
pca<-prcomp(AQ[-c(1,2,3,4,8)])
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6
## Standard deviation     1.6479 0.9981 0.9969 0.9670 0.46092 0.38310
## Proportion of Variance 0.4526 0.1660 0.1656 0.1558 0.03541 0.02446
## Cumulative Proportion  0.4526 0.6187 0.7843 0.9401 0.97554 1.00000
```

```
fviz_screeplot(pca, ncp=10,xlab="Principle Components")
```




```
fviz_pca_var(pca, col.var="contrib")
```

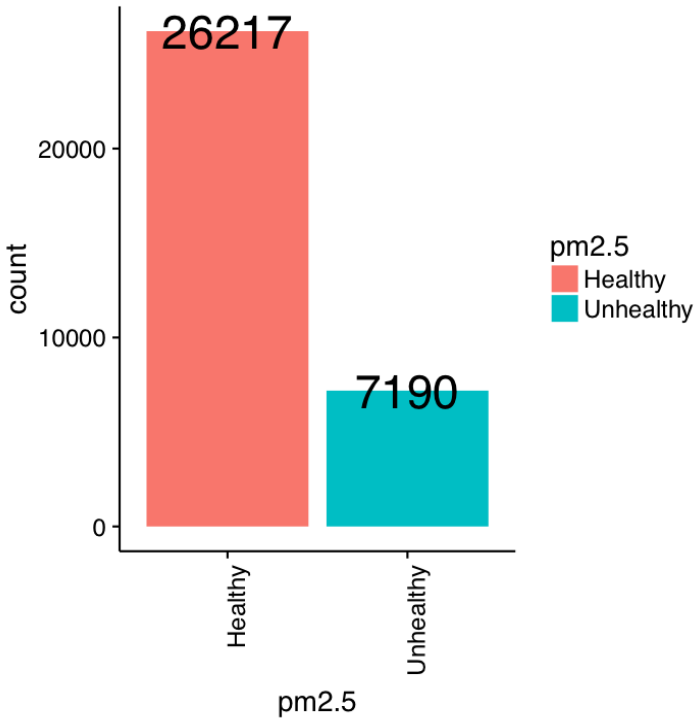


In data mining, it is important to make testing and training set. In this case, I allocated 80% of the data to be the training set and 20% of the data to be the testing set.

```
set.seed(1)
test=sample(1:nrow(AQ),8350)
AQ.test=AQ[test,]
AQ.train=AQ[-test,]
```

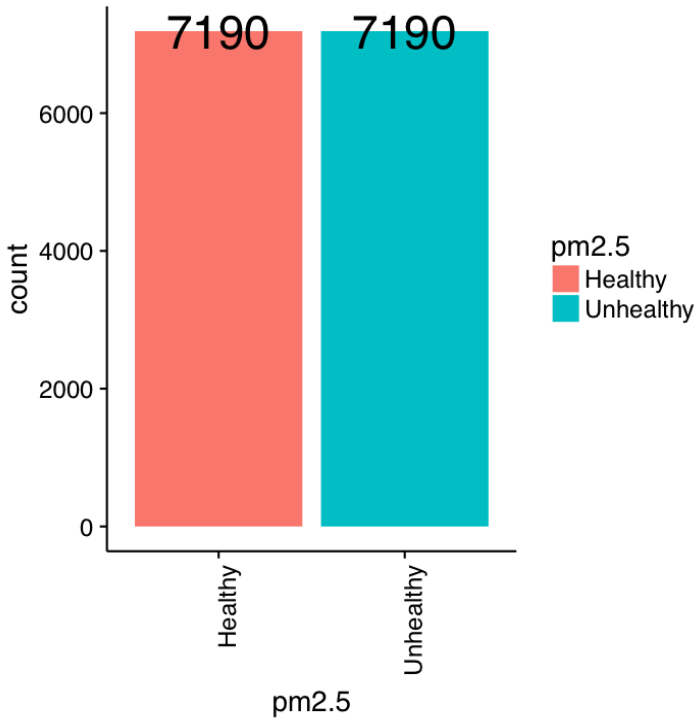
However, the training set is significantly imblanced. Observation with label "healthy" almost quadruples the observations with label "unhealthy".

```
train_count_plot <-
  ggplot(AQ.train,aes(x=pm2.5,fill=pm2.5)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=8,stat ="count", aes(label = ..count.., y = ..count..))
train_count_plot
```

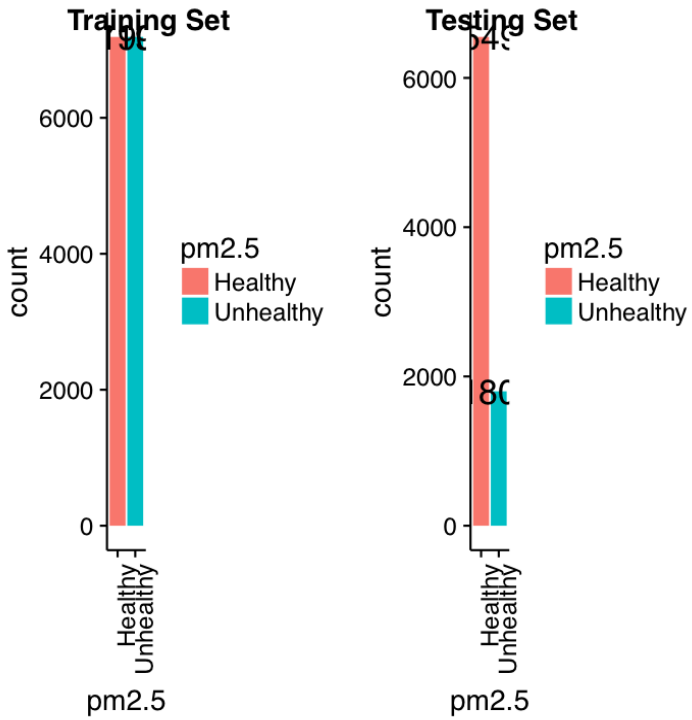


There for I chose undersample method to balanced the training data set. I randomly sampled 7190 observations from the data with "healthy" label. Now, each of the class would have equal number of observations.

```
healthy=AQ.train%>%
  filter(pm2.5=="Healthy")
unhealthy=AQ.train%>%
  filter(pm2.5=="Unhealthy")
healthy=healthy[sample(nrow(healthy),7190),]
AQ.train=rbind(healthy,unhealthy)
new_count_plot<-ggplot(AQ.train,aes(x=pm2.5,fill=pm2.5)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=8,stat ="count", aes(label = ..count.., y = ..count..))
new_count_plot
```




```
p.train <- ggplot(AQ.train,aes(x=pm2.5,fill=pm2.5)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=6,stat ="count", aes(label = ..count.., y = ..count..))
p.test<- ggplot(AQ.test,aes(x=pm2.5,fill=pm2.5)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=6,stat ="count", aes(label = ..count.., y = ..count..))
plot_grid(p.train, p.test, labels=c("Training Set", "Testing Set"), ncol = 2, nrow = 1,scale=1)
```



I used 10-fold cross validation for my training set. All the data are divided into 10 equal folds. Each fold would be a testing set for once and the rest would be treated as training set. Then we calculate the error for each fold and take the mean. We also learned other cross resampling method such as hold out and bootstrap. However, hold out method would not be an ideal choice because I have a large dataset, so simply divide them into two parts probably will not give an accurate result. Bootstrap is not as good as k-fold cross validation because with replacement, not all of my data in the training set will be used. In addition, bootstrap would be a good choice for smaller data.

```
nfold = 10
set.seed(1)
folds = seq.int(nrow(AQ.train)) %>%       ## sequential obs ids
    cut(breaks = nfold, labels=FALSE) %>%   ## sequential fold ids
    sample()                                 ## random fold ids
```

The first model I used is KNN (non-parametric). The concept behind this model is labeling an object based on the majority vote of its neighbors. In defining the object's neighbor, we calculate the distance between them. For KNN alogrithm, it requires the attributes to be either numeric or categorical. In my case, we have a mixed data. One might suggest to convert the categorical ones by assigning 1,2,3,4. However, it is not useful because the distance metrics for the categorical predictors are not defined.

```
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){

  train = (folddef!=chunkid)

  Xtr = Xdat[train,]
  Ytr = Ydat[train]
```

```
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]


  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)

  data.frame(train.error = erate(predYtr, Ytr),
             val.error = erate(predYvl, Yvl))
}
```

Because the dataset is large, I chose the model to have 15 nearest neighbors. The 15 neighbors would vote on the label of a certain object.

```
errors = NULL
set.seed(1)
for(k in 1:15){
  tmp = ldply(1:nfold, do.chunk,
              folddef=folds,
              Xdat=dplyr::select(AQ.train,-pm2.5,-day,-month,-hour,-cbwd),
              Ydat=AQ.train$pm2.5, k = k)    %>%  ## compute fold-wise errors
        summarise_all(funs(mean)) %>%           ## mean training/test errors
        mutate(neighbors = k)
  errors = rbind(errors, tmp)
}
```

The output shows that 7 neighbors would give the lowest validation error.

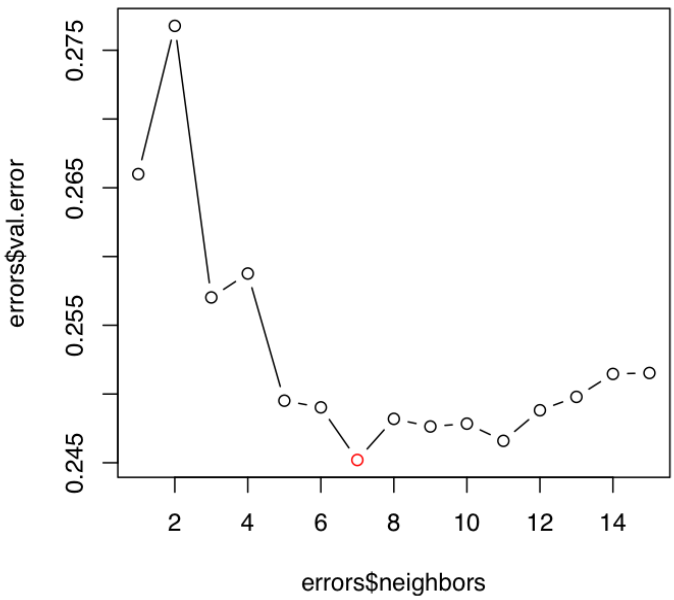
```
errors[which.min(errors$val.error),]
```

```
##   train.error val.error neighbors
## 7   0.1847628 0.2452017         7
```

```
best.k<-(errors[which.min(errors$val.error),]$neighbors)
best.k
```

```
## [1] 7
```

```
plot(errors$neighbors, errors$val.error,type="b",col=ifelse(errors$neighbors==7,"red","black"))
```




```
set.seed(4)
Xtrain_knn = dplyr::select(AQ.train,-pm2.5,-month,-day,-hour,-cbwd)
Ytrain_knn = AQ.train$pm2.5
Xtest_knn  = dplyr::select(AQ.test,-pm2.5,-month,-day,-hour,-cbwd)
Ytest_knn = AQ.test$pm2.5
```

So I choose 7 as the ideal number of neighbors and use the knn function to get the prediction of the testing set and training set. Compare them with the true label for each set, I got 18.12% training error and 27.81% testing error. So after modeling, we get a better result thant the set that was untouched. The big difference between the two errors could also be caused by overfitting.

```
pred.Ytrain = knn(train=Xtrain_knn,test=Xtrain_knn,cl=Ytrain_knn,k=best.k)

train.error.rate = erate(pred.Ytrain,Ytrain_knn)
records.bin[2,1]<-train.error.rate

pred.Ytest = knn(train=Xtrain_knn, test=Xtest_knn, cl=Ytrain_knn, k=best.k)
test.error.rate = erate(pred.Ytest,Ytest_knn)
records.bin[2,2]<-test.error.rate
records.bin
```

```
##               train.error(bin.) test.error(bin.)
## tree                         NA              NA
## knn                   0.1812239       0.2778443
## logistic                     NA              NA
## random forest                NA              NA
```

The next modeling method I used is Classfication Tree(non-parametric). From the tree diagram, it shows that for any "DEWP" <-1.19514 (after scaling) we can calssify that observation as "healthy" disregard of other predictors. Months in summer and fall can also affect the air quality. There tend to be more "unhealthy" labels under the branch of fall.

```
tree_control<-tree.control(nobs=nrow(AQ.train), mincut = 5, minsize = 15, mindev = 0.002)
tree.train=tree(pm2.5~.,data=AQ.train, control = tree_control)
summary(tree.train)
```

```
##
## Classification tree:
## tree(formula = pm2.5 ~ ., data = AQ.train, control = tree_control)
## Variables actually used in tree construction:
## [1] "DEWP"  "Iws"   "TEMP"  "PRES"  "month" "day"
## Number of terminal nodes:  31
## Residual mean deviance:  0.9946 = 14270 / 14350
## Misclassification error rate: 0.2592 = 3727 / 14380
```

```
plot(tree.train)
text(tree.train,pretty=0,cex=0.8,srt=35,col="red")
title("Classification Tree of Size 31")
```

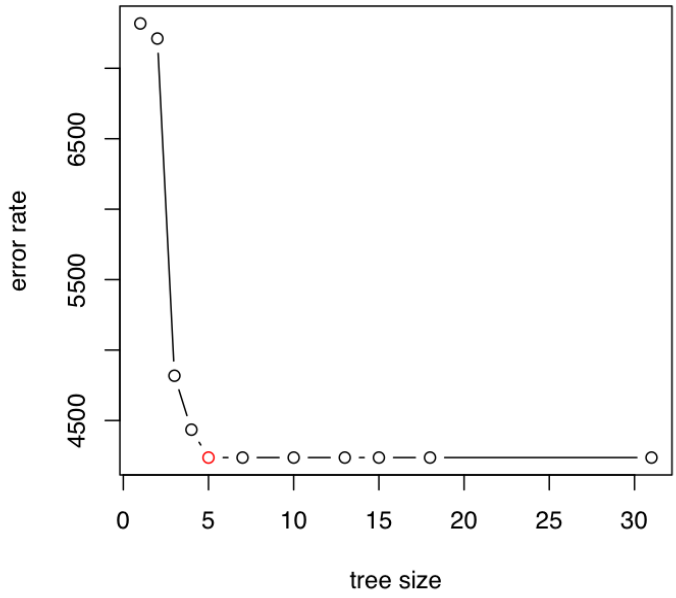
To find out the best number of nodes for my tree model, I used the function cv.tree. To maintain consitency, I also used 10-folds cross validation for this model. It shows that I get the minimum validation error when the tree size is 5.

```
set.seed(1)
cv=cv.tree(tree.train,rand=folds,FUN=prune.misclass,K=10)
cv
```

```
## $size
```

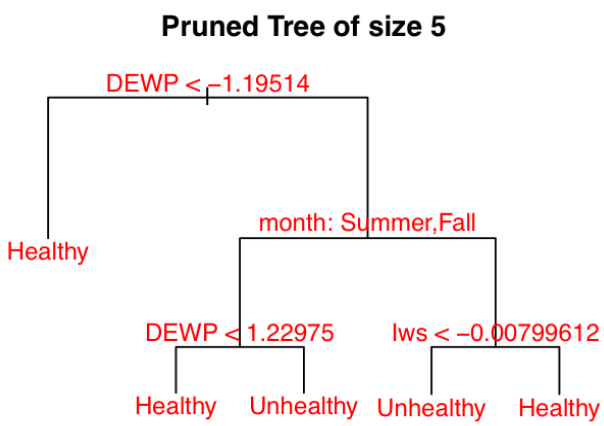
```
##  [1] 31 18 15 13 10  7  5  4  3  2  1
##
## $dev
##  [1] 4237 4237 4237 4237 4237 4237 4237 4435 4818 7211 7318
##
## $k
##  [1]       -Inf    0.00000    9.00000   11.50000   24.33333   49.33333
##  [7]   92.50000  239.00000  396.00000 1127.00000 1245.00000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv$size, cv$dev,ylab="error rate",xlab="tree size",type="b",col=ifelse(cv$size==5,"red","black"))
```



This is the pruned tree with 5 nodes.

```
tree.pruned=prune.misclass(tree.train,best=5)
plot(tree.pruned)
text(tree.pruned, pretty=0,col = "red")
title("Pruned Tree of size 5")
```




```
set.seed(4)
Xtrain = dplyr::select(AQ.train,-pm2.5)
Ytrain = AQ.train$pm2.5
Xtest  = dplyr::select(AQ.test,-pm2.5)
Ytest = AQ.test$pm2.5
```

With the ideal pruned tree of 5 nodes, I predicted the labels for PM 2.5 for both training set and test set and compared the result to the actual labels. The train error is 29.09% and the test error is 30.91%. Since the the small difference between the two types of errors, there is no hidden overfitting and underfitting issue.

```
pred.tree.test=predict(tree.pruned,Xtest,type="class")
tree.test.error.rate=erate(pred.tree.test,Ytest)
records.bin[1,2]<-tree.test.error.rate
pred.tree.train=predict(tree.pruned,Xtrain,type="class")
tree.train.error.rate=erate(pred.tree.train,Ytrain)
records.bin[1,1]<-tree.train.error.rate
records.bin
```

```
##               train.error(bin.) test.error(bin.)
## tree                  0.2908901        0.3091018
## knn                   0.1812239        0.2778443
## logistic                     NA               NA
## random forest                NA               NA
```

My third method is random forest(non-parametric). There is no cross validation involved in this modeling. The summary table shows the out-of-bag error (test error) is 16.1%

```
set.seed(1)
num_predictors=ncol(AQ.train)-1
bag.tree=randomForest(pm2.5~.,data=AQ.train,mtry=num_predictors,importance=TRUE,ntree=1000)
bag.tree
```

```
##
## Call:
##  randomForest(formula = pm2.5 ~ ., data = AQ.train, mtry = num_predictors,      importance = TRUE, n
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 10
##
```

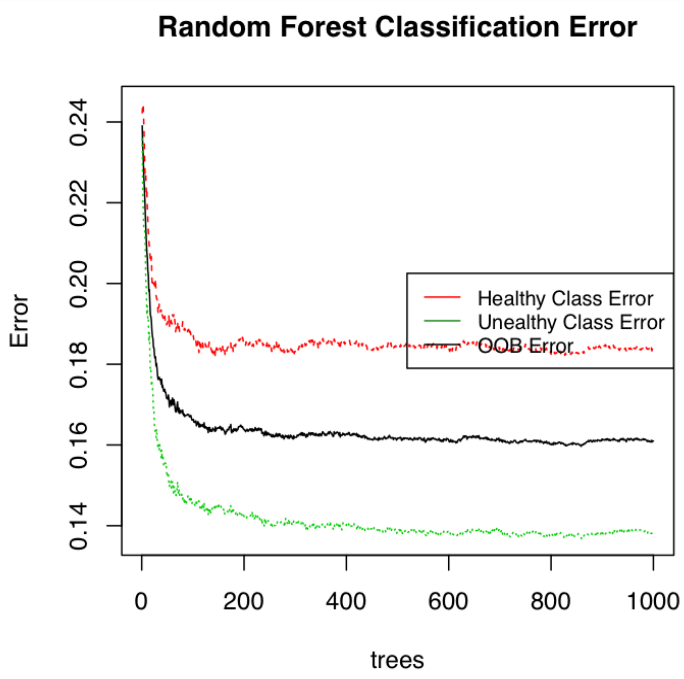
```
##         OOB estimate of  error rate: 16.1%
## Confusion matrix:
##           Healthy Unhealthy class.error
## Healthy      5870      1320   0.1835883
## Unhealthy     995      6195   0.1383866
```

The following plot shows the miscalssifaction error for each class. The error is higher for "healthy" label. The variable importance plot shows the prediction power of the random forest model. So if we remove the top few variables from this model, then the accuracy of the prediction will be reduced. The bottom factors do not affect the prediction too much. For this dataset, "DEWP",TEMP", and"PRES" are important components.

```
bag.tree
```

```
##
## Call:
##  randomForest(formula = pm2.5 ~ ., data = AQ.train, mtry = num_predictors,      importance = TRUE, n
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 16.1%
## Confusion matrix:
##           Healthy Unhealthy class.error
## Healthy      5870      1320   0.1835883
## Unhealthy     995      6195   0.1383866
```

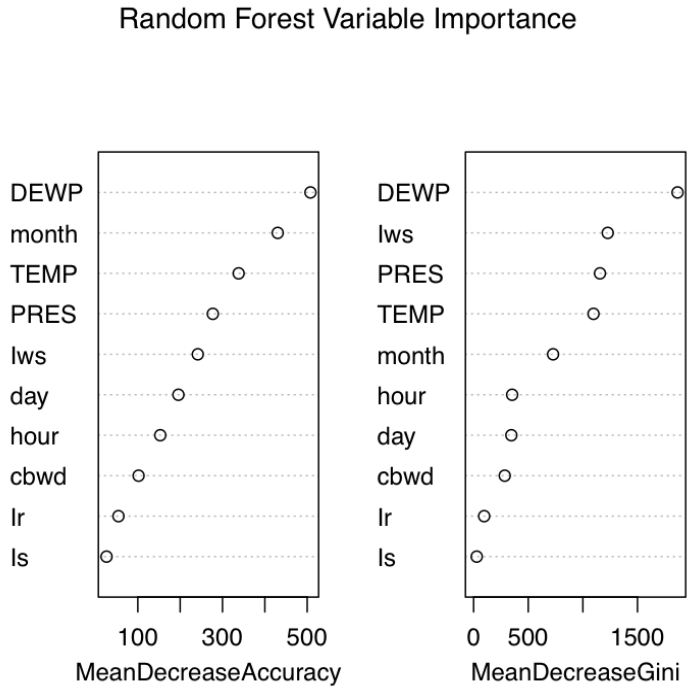
```
plot(bag.tree,main="Random Forest Classification Error")
legend("right",legend=c("Healthy Class Error","Unealthy Class Error","OOB Error"),col=c("red","forestgr
```



```r
importance(bag.tree)
```

```
##         Healthy  Unhealthy MeanDecreaseAccuracy MeanDecreaseGini
## month 128.98085 193.963626            430.39097        727.38755
## day    58.69970 198.845291            195.82706        345.42782
## hour   62.59100 147.157994            153.01713        352.62960
## DEWP   97.07758 323.854859            507.76728       1866.88353
## TEMP   35.80725 301.349145            338.10074       1097.37064
## PRES  110.04831 189.132334            277.13073       1156.73234
## cbwd   18.73051 114.041426            101.75241        284.84247
## Iws    65.35197 249.305646            241.65460       1227.75706
## Is     37.63226   3.993105             26.10431         28.43147
## Ir     50.54896  25.053760             54.29273         96.43456
```

```r
varImpPlot(bag.tree,main="Random Forest Variable Importance",)
```



It is interesting that the train error for random forest model is almost 0. The random forest algorithm uses bootstrap aggregation, so each new tree is grown from a bootstrap sample of the training observations. So even though each tree in the forest is not a perfect fit, which resulted the out-of-bag error to be 17.39% in this case, the whole forest think itself to be perfect. Therefore resulted 0.0006965% training error. In general we don't take consider of this training error.

```r
pred.bag.tree.train=predict(bag.tree,Xtrain,type="class")
pred.bag.tree.train.error.rate=erate(pred.bag.tree.train,Ytrain)
pred.bag.tree.test=predict(bag.tree,Xtest,type="class")
pred.bag.tree.test.error.rate=erate(pred.bag.tree.test,Ytest)
```

```
records.bin[4,1]<-pred.bag.tree.train.error.rate
records.bin[4,2]<-pred.bag.tree.test.error.rate
records.bin
```

```
##               train.error(bin.) test.error(bin.)
## tree               0.2908901252        0.3091018
## knn                0.1812239221        0.2778443
## logistic                     NA               NA
## random forest      0.0006954103        0.1736527
```

The last method I used was logistic regression(parametric). With the "do.chunk.glm" function, I can find a threshold for classifying PM 2.5 Index. So any prediction that is larger than the threshold would be classified as "unhealthy", and any prediction that is less than or equal to the threshold would be labeld as "healthy".

```
do.chunk.glm <- function(chunkid, folddef, Xdat, p){

  train = (folddef!=chunkid)

  Xtr = Xdat[train,-4]
  Ytr = Xdat[train,4]

  Xvl = Xdat[!train,-4]
  Yvl = Xdat[!train,4]

  fit.glm = suppressWarnings(glm(pm2.5~.,family=binomial,data=Xdat[train,]))

  prob.training=predict(fit.glm,Xtr,type="response")
  prob.testing=predict(fit.glm,Xvl,type="response")

  pred.Ytr = ifelse(prob.training>p, "Unhealthy", "Healthy")
  pred.Yvl = ifelse(prob.testing>p,"Unhealthy","Healthy")

  data.frame(train.error = erate(pred.Ytr, Ytr),
             val.error = erate(pred.Yvl, Yvl))
}
```

```r
errors.glm = NULL
set.seed(1)
cand=seq(0,1,by=0.01)
for(p in 1:length(cand)){
  tmp = ldply(1:nfold,do.chunk.glm,
              folddef=folds,
              Xdat=AQ.train,
              p=cand[p])    %>%  ## compute fold-wise errors
        summarise_all(funs(mean)) %>%           ## mean training/test errors
        mutate(probability = cand[p])

  errors.glm = rbind(errors.glm, tmp)
}
```

The best threshold for classifying air quality is 0.45.

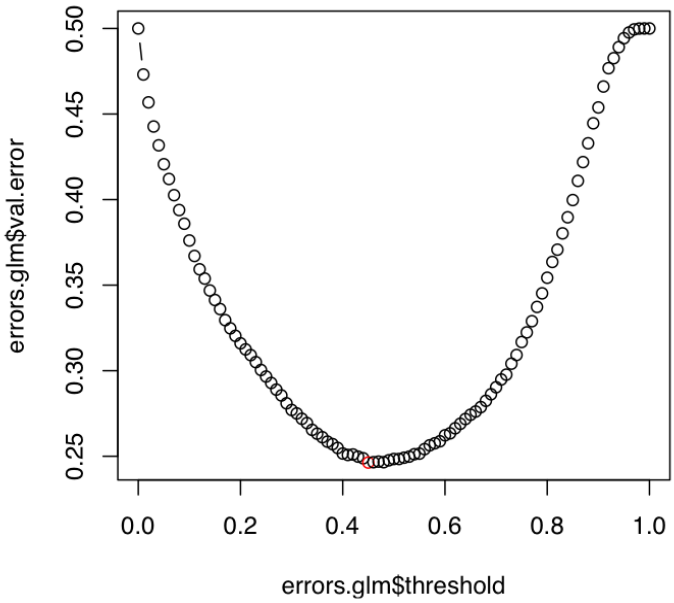
```
errors.glm[which.min(errors.glm$val.error),]
```

```
##    train.error val.error probability
## 46   0.2468629 0.2463143        0.45
```

```
best.threshold<-(errors.glm[which.min(errors.glm$val.error),]$probability)
best.threshold
```

```
## [1] 0.45
```

```
plot(errors.glm$probability, errors.glm$val.error,xlab="errors.glm$threshold",type="b",col=ifelse(error
```



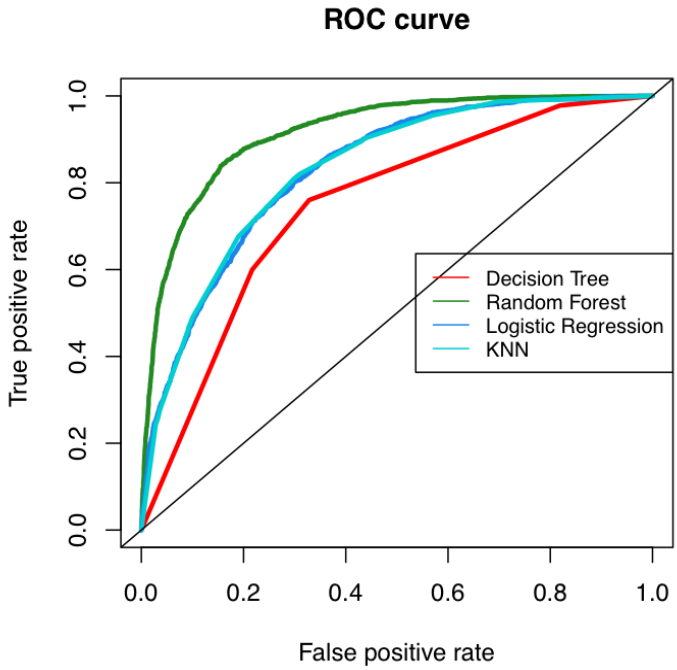
The result shows that logistic regression is also a good method. After modeling, the training error is 5% lower than the test dataset. Even though logistic regression is not a good fit for large feature space (in this case I have feture dimension of 11), for this particular dataset, it gives a pretty good accuracy.

```
glm.fit=suppressWarnings(glm(pm2.5~.,data=AQ.train,family="binomial"))
glm.pred.train=predict(glm.fit,Xtrain,type="response")
pred.Ytr = ifelse(glm.pred.train>best.threshold, "Unhealthy", "Healthy")
glm.train.error=erate(pred.Ytr,Ytrain)

glm.pred.test=predict(glm.fit,Xtest,type="response")
pred.Yvl = ifelse(glm.pred.test>best.threshold, "Unhealthy", "Healthy")
glm.test.error=erate(pred.Yvl,Ytest)

records.bin[3,1]<-glm.train.error
records.bin[3,2]<-glm.test.error
records.bin
```

```
##               train.error(bin.) test.error(bin.)
## tree             0.2908901252        0.3091018
## knn              0.1812239221        0.2778443
## logistic         0.2470792768        0.2996407
## random forest    0.0006954103        0.1736527
```

The ROC curve is a visual aid that shows the competence of each model. For ROC curves, the closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. Therefore, the random forest method seems to be the most accurate one which also shows in the record table with 17.38% error rate. For the rest three methods, the Logistic Regression and KNN method have same performance from the ROC curve. In the record table, the KNN does a better job.Even though Decision Tree is not as good as the rest three, the result is more reliable because there is no overfitting and underfitting issue; the difference between train error and test error is not too big.

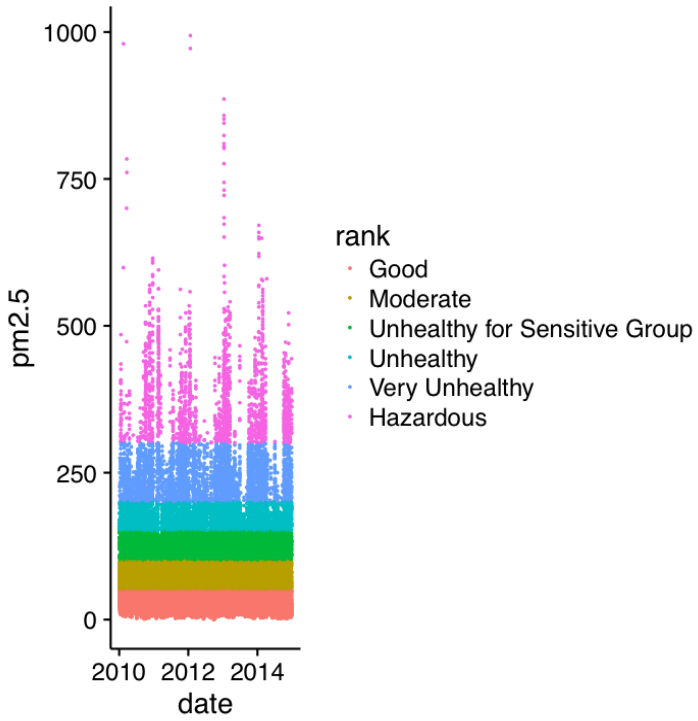
```
library(ROCR)
pred.tree=predict(tree.pruned,Xtest)
pred.tree=data.frame(pred.tree[,2])
newpred.tree=prediction(pred.tree,Ytest)
perf.tree=performance(newpred.tree,measure="tpr", x.measure = "fpr")
plot(perf.tree, col="red", lwd=3, main="ROC curve")

pred.bag=predict(bag.tree,Xtest, type="prob")
pred.bag=data.frame(pred.bag[,2])
newpred.bag=prediction(pred.bag,Ytest)
perf.bag=performance(newpred.bag,measure="tpr", x.measure = "fpr")
plot(perf.bag, add=TRUE, col="forestgreen", lwd=3, main="ROC curve")

fit2.glm=suppressWarnings(glm(pm2.5~., data=AQ.train, family="binomial"))
pred.glm=data.frame(predict(fit2.glm,Xtest))
newpred.glm=prediction(pred.glm,Ytest)
perf.glm=performance(newpred.glm,measure="tpr", x.measure = "fpr")
plot(perf.glm, add=TRUE,col="dodgerblue2", lwd=3, main="ROC curve")

pred.knn = knn(train=Xtrain_knn,test=Xtest_knn, cl=Ytrain_knn, k=best.k,prob=TRUE)
prob<-attr(pred.knn,"prob")
prob[which(pred.knn=="Healthy")]=1-prob[which(pred.knn=="Healthy")]
newpred.knn=prediction(prob,Ytest)
perf.knn=performance(newpred.knn,measure="tpr", x.measure = "fpr")
plot(perf.knn, add=TRUE, col="darkturquoise", lwd=3, main="ROC curve")
abline(0,1)
legend("right",legend=c("Decision Tree","Random Forest","Logistic Regression","KNN"),col=c("red","fores
```



Next, I will build similar models on multiclass situation.

First, I divided PM2.5 into 6 categories, according to the official guideline. 0-50 "Good",51-100 "Moderate", 101-150 "Unhealthy for Sensitive Group", 151-200 "Unhealthy", 201-300 "Very Unhealthy", >301 "Hazardous".

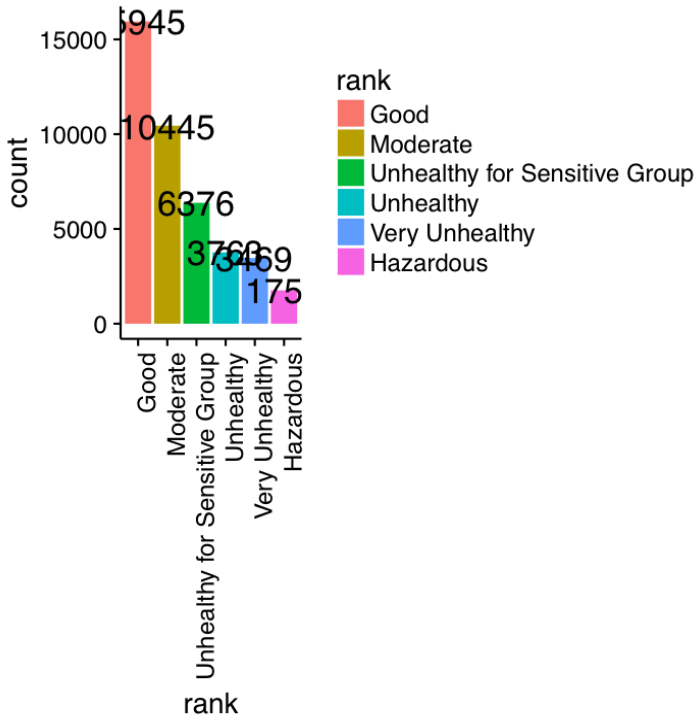
```
AQ$rank<- cut(AQ$pm2.5, breaks=c(-1,50,100,150,200,300,max(AQ$pm2.5)),
labels=c("Good","Moderate","Unhealthy for Sensitive Group","Unhealthy","Very Unhealthy","Hazardous"))
general_plot <- ggplot(AQ, aes(date,pm2.5, color = rank)) +
  geom_point(size=0.2)
general_plot
```




```
AQ=AQ[-c(1,2,6,14,15)]
```

Then I did similar data processing as the binary class case. I also allocated 80% to be the training set and 20% to be the testing set.

```
count_plot <- ggplot(AQ,aes(x=rank,fill=rank)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=6,stat ="count", aes(label = ..count.., y = ..count..))
count_plot
```

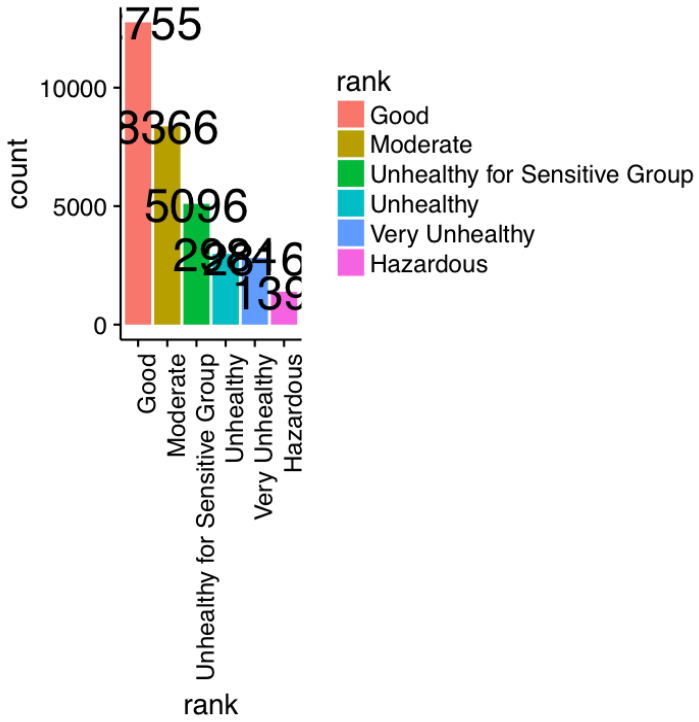



```
set.seed(1)
test=sample(1:nrow(AQ),8350)
AQ.test=AQ[test,]
AQ.train=AQ[-test,]
```

```
good=AQ.train%>%
  filter(rank=="Good")
moderate=AQ.train%>%
  filter(rank=="Moderate")
unhealthy_sens=AQ.train%>%
  filter(rank=="Unhealthy for Sensitive Group")
unhealthy=AQ.train%>%
  filter(rank=="Unhealthy")
very_unhealthy=AQ.train%>%
  filter(rank=="Very Unhealthy")
hazardous=AQ.train%>%
  filter(rank=="Hazardous")
```

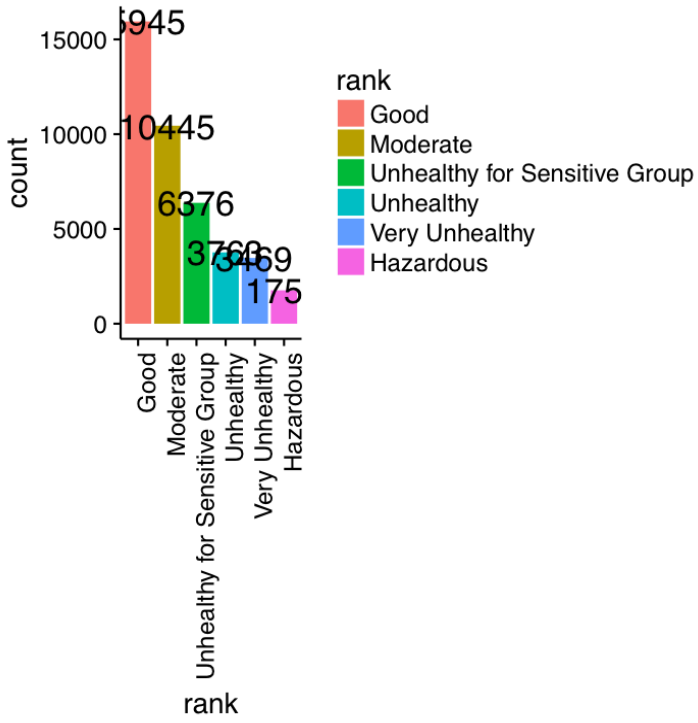
From the bar plot, we see that the training set is very imbalanced.

```
train_count_plot <-
  ggplot(AQ.train,aes(x=rank,fill=rank)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=8,stat ="count", aes(label = ..count.., y = ..count..))
train_count_plot
```



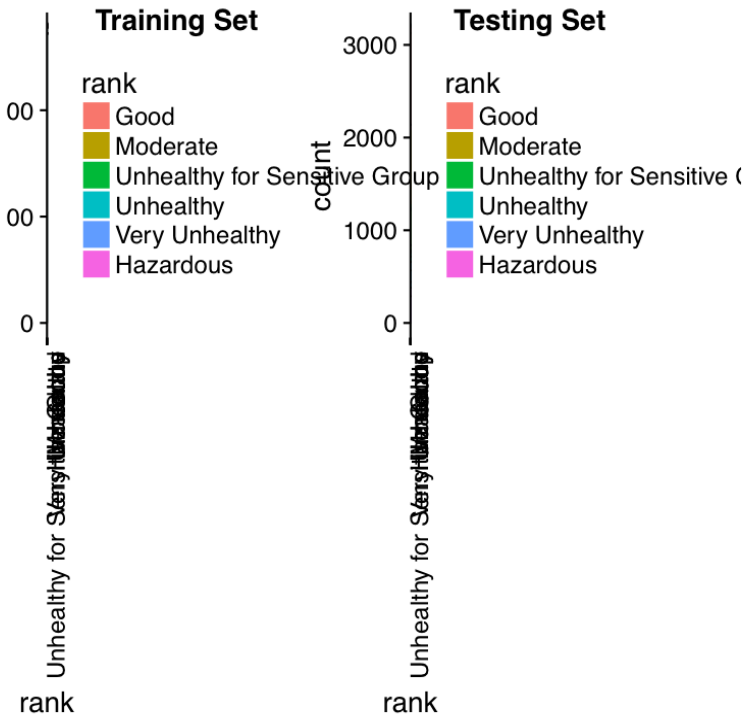
I still chose undersample method and randomly sampled 1390 observations from each class.

```
set.seed(1)
good=good[sample(nrow(good),1390),]
moderate=moderate[sample(nrow(moderate),1390),]
unhealthy_sens=unhealthy_sens[sample(nrow(unhealthy_sens),1390),]
unhealthy=unhealthy[sample(nrow(unhealthy),1390),]
very_unhealthy=very_unhealthy[sample(nrow(very_unhealthy),1390),]
AQ.train=rbind(good,moderate,unhealthy_sens,unhealthy,very_unhealthy,hazardous)

new_count_plot <- ggplot(AQ,aes(x=rank,fill=rank)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=6,stat ="count", aes(label = ..count.., y = ..count..))
new_count_plot
```



So now each label has 1390 observations. The training set is balanced.

```
p.train <- ggplot(AQ.train,aes(x=rank,fill=rank)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=4,stat ="count", aes(label = ..count.., y = ..count..))
p.test<- ggplot(AQ.test,aes(x=rank,fill=rank)) +
  geom_bar(stat="count")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_text(size=4,stat ="count", aes(label = ..count.., y = ..count..))
plot_grid(p.train, p.test, labels=c("Training Set", "Testing Set"), ncol = 2, nrow = 1,scale=1)
```



Same cross validation method.

```
nfold = 10
set.seed(1)
folds = seq.int(nrow(AQ.train)) %>%       ## sequential obs ids
    cut(breaks = nfold, labels=FALSE) %>%   ## sequential fold ids
    sample()                                  ## random fold ids
```

```
errors = NULL
set.seed(1)
for(k in 1:15){
  tmp = ldply(1:nfold, do.chunk,
              folddef=folds,
              Xdat=dplyr::select(AQ.train,-rank,-month,-day,-hour,-cbwd),
              Ydat=AQ.train$rank, k = k)    %>%  ## compute fold-wise errors
        summarise_all(funs(mean)) %>%           ## mean training/test errors
        mutate(neighbors = k)
  errors = rbind(errors, tmp)
}
```

The best number of neighbor is 8 for multiple class case.

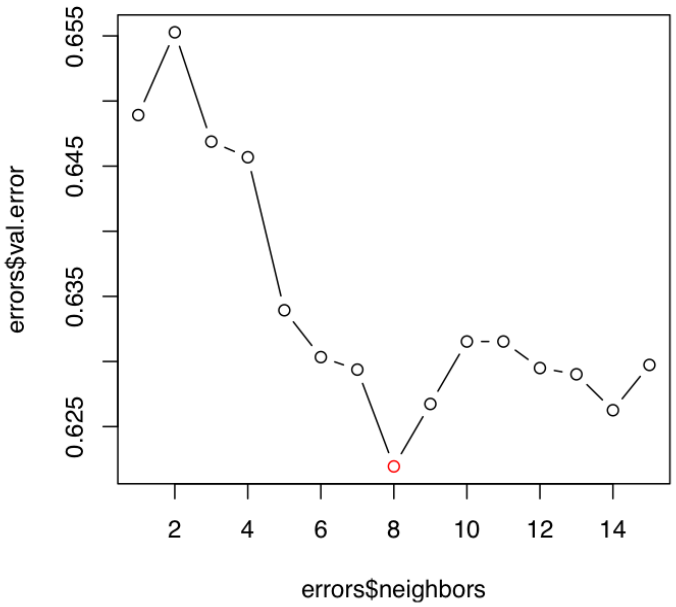
```
errors[which.min(errors$val.error),]
```

```
##   train.error val.error neighbors
## 8   0.4693046 0.6219424         8
```

```
best.k<-(errors[which.min(errors$val.error),]$neighbors)
best.k
```

```
## [1] 8
```

```
plot(errors$neighbors, errors$val.error,type="b",col=ifelse(errors$neighbors==8,"red","black"))
```




```
set.seed(4)
Xtrain_knn = dplyr::select(AQ.train,-month,-day,-hour,-rank,-cbwd)
Ytrain_knn = AQ.train$rank
Xtest_knn  = dplyr::select(AQ.test,-month,-hour,-day,-rank,-cbwd)
Ytest_knn = AQ.test$rank
```

Similar to the binary case, the KNN in this case also has a big difference between the training and testing error. The overall error rate is much higher than that of the binary class. However, 40% accuracy is considerably ideal in the case of six labels.

```
pred.Ytrain = knn(train=Xtrain_knn,test=Xtrain_knn,cl=Ytrain_knn,k=best.k)

train.error.rate = erate(pred.Ytrain,Ytrain_knn)
records.mul[2,1]<-train.error.rate

pred.Ytest = knn(train=Xtrain_knn, test=Xtest_knn, cl=Ytrain_knn, k=best.k)
test.error.rate = erate(pred.Ytest,Ytest_knn)
records.mul[2,2]<-test.error.rate
records.mul
```

```
##               train.error(mul.) test.error(mul.)
## tree                         NA               NA
## knn                   0.4673861        0.6180838
## random forest                NA               NA
```

```
## N/A                           NA               NA
```

The decision tree in this case expands from Spring and Winter. The "cbwd" also plays an important role. In the previous decision tree, I did not see the wind direction as a significant factor.

```r
tree_control<-tree.control(nobs=nrow(AQ.train), mincut = 5, minsize = 15, mindev = 0.002)
tree.train=tree(rank~.,data=AQ.train, control = tree_control)
summary(tree.train)
```

```
##
## Classification tree:
## tree(formula = rank ~ ., data = AQ.train, control = tree_control)
## Variables actually used in tree construction:
## [1] "month" "DEWP"  "Iws"   "TEMP"  "PRES"  "cbwd"  "day"
## Number of terminal nodes:  24
## Residual mean deviance:  2.907 = 24170 / 8316
## Misclassification error rate: 0.6128 = 5111 / 8340
```

```r
plot(tree.train)
text(tree.train,pretty=0,cex=0.8,srt=35,col="red")
title("Classification Tree of Size 24")
```

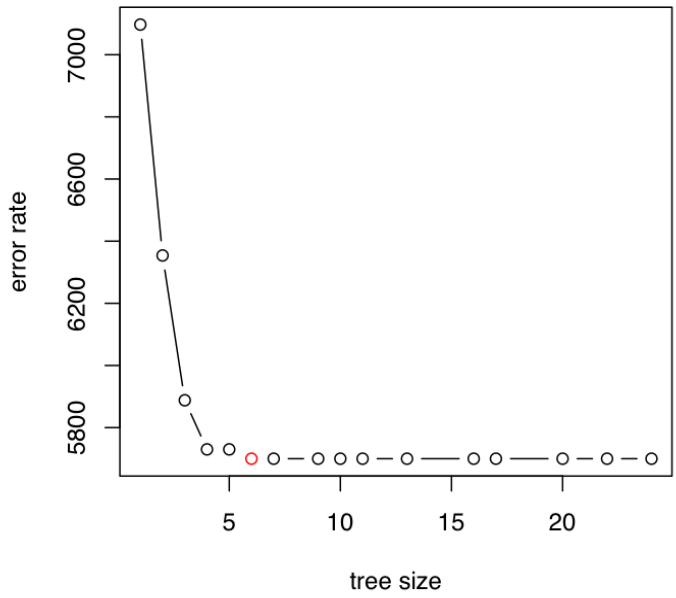
\begin{center}\includegraphics{Project_131_ver2_files/figure-latex/unnamed-chunk-35-1} \end{center}

I did the same 10-fold cross validation for finding the ideal tree size. It shows that the error is minimized when the tree has 6 nodes.

```r
set.seed(1)
cv=cv.tree(tree.train,rand=folds,FUN=prune.misclass,K=10)
cv
```

```
## $size
##  [1] 24 22 20 17 16 13 11 10  9  7  6  5  4  3  2  1
##
## $dev
##  [1] 5700 5700 5700 5700 5700 5700 5700 5700 5700 5700 5700 5730 5730 5888
## [15] 6354 7097
##
## $k
##  [1]       -Inf   0.000000   8.500000   9.666667  12.000000  13.333333
##  [7]  16.000000  26.000000  40.000000  41.000000  49.000000  71.000000
## [13]  72.000000 164.000000 453.000000 752.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(cv$size, cv$dev,ylab="error rate",xlab="tree size",type="b",col=ifelse(cv$size==6,"red","black"))
```



After pruning the tree, I noticed that the "DEWP" factor has always be an important indicator.

```r
tree.pruned=prune.misclass(tree.train,best=6)
plot(tree.pruned)
text(tree.pruned, pretty=0,col = "red")
title("Pruned tree of size 6")
```

\begin{center}\includegraphics{Project_131_ver2_files/figure-latex/unnamed-chunk-37-1} \end{center}

```r
set.seed(4)
Xtrain = dplyr::select(AQ.train,-rank)
Ytrain = AQ.train$rank
Xtest  = dplyr::select(AQ.test,-rank)
Ytest = AQ.test$rank
```

The error for decision tree shows that the model only worses the accuracy. The training error is 5% higher than the test error, which indicates that this model is not a good fit.

```r
pred.tree.test=predict(tree.pruned,AQ.test[-c(11)],type="class")
tree.test.error.rate=erate(pred.tree.test,Ytest)
records.mul[1,2]<-tree.test.error.rate
pred.tree.train=predict(tree.pruned,AQ.train[-c(11)],type="class")
tree.train.error.rate=erate(pred.tree.train,Ytrain)
records.mul[1,1]<-tree.train.error.rate
records.mul
```

```
##               train.error(mul.) test.error(mul.)
## tree                  0.6520384        0.6037126
## knn                   0.4673861        0.6180838
## random forest                NA               NA
## N/A                          NA               NA
```

```
set.seed(1)
num_predictors=ncol(AQ.train)-1
num_predictors
```

```
## [1] 10
```

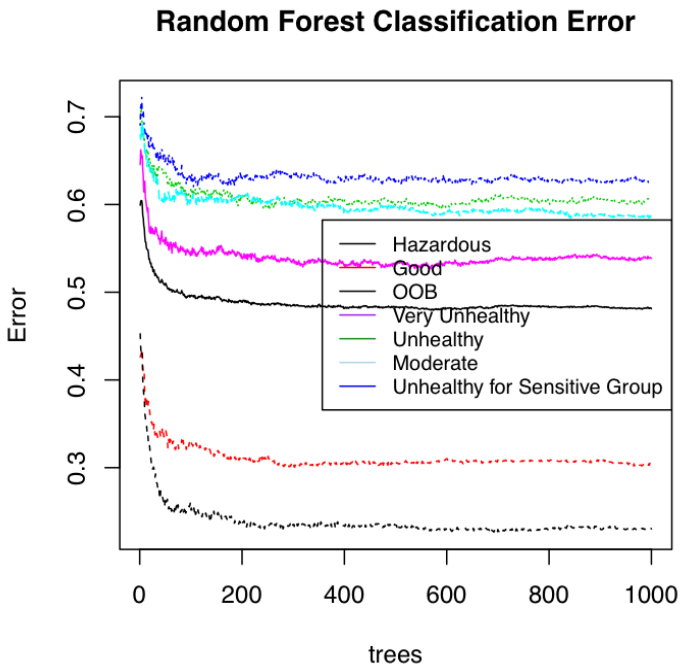
```
bag.tree=randomForest(rank~.,data=AQ.train,mtry=num_predictors,importance=TRUE,ntree=1000)
```

The summary of the Random Forest gives an out-of-bag error 48.12% which is very low for multiple classes. As we see from the classification error plot, all errors become steady after we have 300 trees in the forest. "Unhealty for Sensitive people" and "Moderate" are the two classes that have higher misclassification error. The Ramdom Forest Variable Importance shows that "DEWP", "TEMP" are still the two important factors that affect the prediction power, which is similar to the previous binary case.

```
bag.tree
```

```
##
## Call:
##  randomForest(formula = rank ~ ., data = AQ.train, mtry = num_predictors,      importance = TRUE, nt
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 48.12%
## Confusion matrix:
##                               Good Moderate Unhealthy for Sensitive Group
## Good                           968      212                            96
## Moderate                       235      550                           281
## Unhealthy for Sensitive Group   95      261                           519
## Unhealthy                       62      139                           260
## Very Unhealthy                  33       73                           120
## Hazardous                        4       23                            39
##                               Unhealthy Very Unhealthy Hazardous
## Good                             58             38        18
## Moderate                        156            105        63
## Unhealthy for Sensitive Group   267            147       101
## Unhealthy                       579            221       129
## Very Unhealthy                  231            641       292
## Hazardous                        71            183      1070
##                               class.error
## Good                        0.3035971
## Moderate                    0.6043165
## Unhealthy for Sensitive Group 0.6266187
## Unhealthy                   0.5834532
## Very Unhealthy              0.5388489
## Hazardous                   0.2302158
```

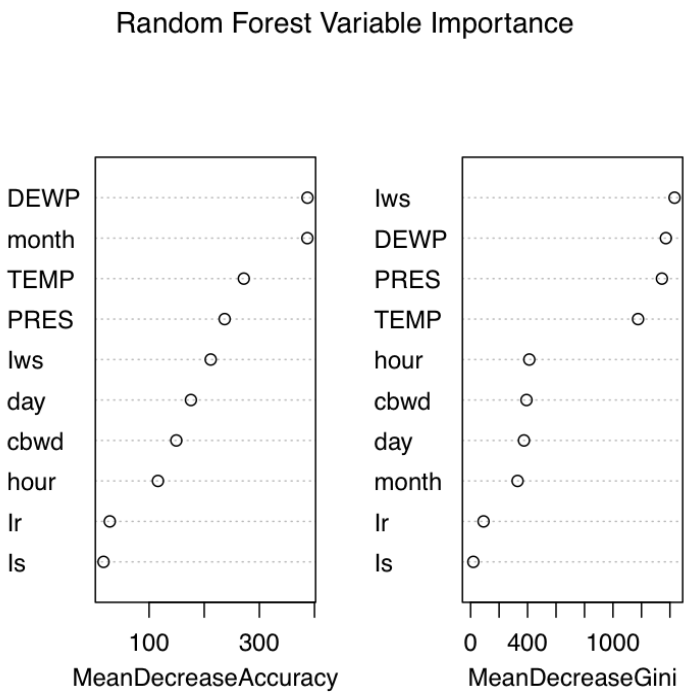
```
plot(bag.tree,main="Random Forest Classification Error")
legend("right",legend=c("Hazardous","Good","OOB","Very Unhealthy","Unhealthy","Moderate","Unhealthy for
```




```
importance(bag.tree)
```

```
##             Good   Moderate Unhealthy for Sensitive Group  Unhealthy
## month  81.289141 104.352806                     131.680964 124.264919
## day    24.528562  32.547739                      39.496179  66.239769
## hour   24.396419  30.523005                      36.301480  47.922630
## DEWP  128.681082  92.889780                     162.418028 171.706288
## TEMP   40.243904  48.399133                      69.019017  78.591714
## PRES   99.757618  43.074107                      39.210667  83.836509
## cbwd   79.222482  44.119463                      46.965983  72.435395
## Iws   136.987303  41.097812                      41.184655  67.430335
## Is      4.664019   6.195824                       8.037603   6.396165
## Ir      7.475511  19.534066                       8.242387   9.766329
##       Very Unhealthy  Hazardous MeanDecreaseAccuracy MeanDecreaseGini
## month      71.059371 342.401535             387.00014         329.8630
## day        90.684324 163.983071             175.87576         374.6667
## hour       46.149429  86.859988             116.06440         412.1456
## DEWP       92.020884 158.930512             387.22835        1371.4932
## TEMP      101.479761 242.366310             271.79179        1175.9984
## PRES       91.555810 153.598839             237.02525        1343.6276
## cbwd       57.045921  47.117931             149.35375         392.4920
## Iws        73.051370 150.623194             211.74504        1432.3711
## Is         10.554009   5.559168              17.37957          19.0424
## Ir          6.939294  17.567396              28.72591          90.7856
```

```
varImpPlot(bag.tree,main="Random Forest Variable Importance")
```



The Random Forest method gives a better result than the previous two. Similar to the binary case, the Random Forest method also have very low training error. So we are not going to take the training error in to account.

```
pred.bag.tree.train=predict(bag.tree,Xtrain,type="class")
pred.bag.tree.train.error.rate=erate(pred.bag.tree.train,Ytrain)
pred.bag.tree.test=predict(bag.tree,Xtest,type="class")
pred.bag.tree.test.error.rate=erate(pred.bag.tree.test,Ytest)
records.mul[3,1]<-pred.bag.tree.train.error.rate
records.mul[3,2]<-pred.bag.tree.test.error.rate
records.mul
```

```
##               train.error(mul.) test.error(mul.)
## tree                 0.652038369        0.6037126
## knn                  0.467386091        0.6180838
## random forest        0.001438849        0.4661078
## N/A                           NA               NA
```

Since the logistic regression would only work in binary classes, I chose not to include this modeling method for the multiclass situation.

```
cbind(records.bin,records.mul)
```

```
##               train.error(bin.) test.error(bin.) train.error(mul.)
## tree             0.2908901252        0.3091018       0.652038369
## knn              0.1812239221        0.2778443       0.467386091
```

```
## logistic           0.2470792768        0.2996407       0.001438849
## random forest      0.0006954103        0.1736527                NA
##               test.error(mul.)
## tree                 0.6037126
## knn                  0.6180838
## logistic             0.4661078
## random forest               NA
```

Conclusion:

In order to better predict the PM 2.5, I established several models based on the data of PM 2.5 Index and meteorological records in Beijing from 2010-2014, through methods including KNN, decision tree, random forest and logistic regression, were adopted separately. As the result, the random forest and the KNN methods give the least test and train errors, even though the high accuracy may arise from overfitting.The maximum accuracy of the four models is 83% in binary case, which proves the utility of my models. For the furture work, it would be useful to collect other revelant variables, such as CO2 emission1 etc, which may gave a more accurate prediction.

Finally, I would like to acknowledge TA Arron Zhou for technical assistance and Professor Sang-Yun Oh for lecturing.

Reference:

https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data