STA6704 Final Report: Data Ming on Parkinsons Dataset

Abstract: In this project, firstly we do dataset description, statistical summary and feature selection for Parkinsons dataset . Then, support vector regression(SVR) with kernels and boosted regression(BRT) tree are introduced to fit models with the Unified Parkinson's Disease Rating Scale as the response. Our outcomes on the test dataset illustrate that the BRT model outperforms the SVR model on prediction accuracy indicated by the root mean squared error(RMSE), and the tuned models generally work better than the untuned.

Keywords: feature selection, support vector regression, kernel, regression tree, boosting.

1 Introduction

Parkinson's disease is a disorder of central nervous system that affects movements. After diagnosis, special treatments can relieve the symptoms, but there is no cure. Then, researchers endeavour to track patients' state data to evaluate sickness. Pakinsons dataset involves a range of biomedical voice measurements according to 42 patients with early-stage Parkinson's disease gathered to a six-month trial by a telemonitoring device for remote symptom progression monitoring. The samples were automatically collected in the patient's homes.

The primary goal for this project is to develop models that can correctly predict the Unified Parkinson's Disease Rating Scale (UPDRS), corresponding to 'total_UPDRS', based on each recorded predictor. Specifically, the main works focus on:

(1)using support vector regression with kernels to fit a prediction model .

(2)using regression tree with boosting to fit a prediction model .

(3)selecting kernel functions and tuning fitting parameters.

(4)comparing the performance of different models.

2 Dataset Preparation

There are 5875 samples with 22 predictors(numerical and categorical) in the dataset containing subject number, subject age, subject gender, time interval from baseline recruitment date, motor UPDRS, total UPDRS, and sixteen biomedical voice measures. Each sample corresponds to one of 5,875 observations from these patients.

2.1 Dependent/independent Variables

The dependent variable y is a numeric variable created from the "total_UPDRS" variable, indicating the Parkinson's Disease Rating Scale for patients. For the possible independent variables, we study all the 21 listed variables to identify the key variables that may be considered for doing statistical analysis. There are 2 categorical variables and 19 quantitative variables. A summary of the key variables name, the data type, a brief description, and summary statistics included in Tab.1 and Tab.2.

2.2 Statistical Description

For categorical variables, we can use the

bar charts to demonstrate their statistical properties, shown in Fig.1 about 'sex' bar chart. For numeric variables, the scatterplot can be used to display relationship between two numeric variables, shown in Fig.2. To display the outcome distribution, the histogram is performed here displayed in Fig.3. similarly, Fig.4 and Fig.5 are respectively the boxplot of total_UPDRS vs sex and the heatmap of variable correlations.

2.3 Missing Values

Missing values occur when no data is recorded for an observation. A couple of methods have been suggested to deal with missing value issues: (1) ignore the record, (2) fill the missing value manually, (3) use a global constant, (4) replace the missing value with the mean, (5) replace the missing value with the mean of that category, (6) use the most likely value through the help of regression. In this study, we have 5875 observations in total without any missing values.

2.4 Outliers

An outlier is a point for which y_i is far from the value predicted by the model. Outliers can arise for a variety of reasons, such as incorrect recording of an observation during data collection. Often outliers decrease the accuracy and efficiency of the models. The detection of outliers of the continuous/ quantitative variables can be done through the determination of the upper and lower limits, which is normally the $\pm$ 3 standard deviation from the mean value of that variable. In our study, several outliers are adjusted to be $\pm$ 3 standard deviation of the mean.

2.5 Variable Selection

Variable selection is an approach for excluding irrelevant variables from a regression model. The stepwise regression is the most commonly used method for selecting the variables to be included in the regression models because it was found to produce the most parsimonious model. The default entry and stay significance level is 0.05. The selection criterion is the Akaike's Information Criterion (AIC). This means that the final model selected has the lowest value of AIC statistic. Through the "backward" and "both" selection methods, we obtain the same variable selection result with 15 variables when AIC=8232.99 shown in Tab.3, which differs from keeping all vaiables from the "forward" selection method with AIC=8241.77.

3 Methodology

3.1 Support Vector Regression

Unlike in classification problems concentrating on the classification boundary, support vector regression aims to minimize mean squared error:

$$f(x) = x^T \beta + \beta_0$$

To estimate β, the problem is converted to minimize

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2} \| \beta \|^2$$

For any form of V(r), the solution $\hat{f}(x) = \sum \hat{\beta}_m h_m(x) + \hat{\beta}_0$ has the form

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{a}_i K(x, x_i)$$

where $K(x, x_i)$ is an inner product kernel.

3.2 Kernel functions

If $K(x, x_i)$ is a linear kernel, then it can be

$$K(x, x_i) =< x, x_i >= x^T x_i$$

Several kernel function forms can be represented as below

Kernel	Mathematical representation
linear	$K(x_i, x_j) =< x_i, x_j >= x_i^T x_j$
polynomial	$K(x_i, x_j) = (x_i^T x_j)^d \qquad d \geq 1$
Gaussian	$K(x_i, x_j) = \exp(-\frac{\| x_i - x_j \|^2}{2\sigma^2})$ $\sigma > 0$
Sigmoid	$K(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$ $\beta > 0, \theta < 0$

3.2 Regression Tree with Boosting

In this project, we apply the gradient boosting to lift the original regression tree model, with the general idea of training on the residuals and tweaking parameters iteratively to minimize the cost function. Generally, boosted trees are grown sequentially, using residual information from previously grown trees. The basic algorithm for boosted regression trees can be generalized as below with x as the independent variables and y as the dependent variable.

1. Fit a decision tree : m1(x) = y,

2. Fit the next decision tree to the residuals of the previous: h1(x) = y- m1(x),

3. Add this new tree to our algorithm: m2(x) = m1(x) + h1(x),

4. Fit the next decision tree to the residuals of h2(x) = y - m2(x),

5. Add this new tree to our algorithm: m3(x) = m2(x) + h2(x),

6. Continue this process until the iteration number reaches the maximum.

3.3 Random Forest

Random forests are a kind of ensemble learning method, used for classification and regression tasks. It's actually an improvement of bagging over a small tweak that decorrelates the trees. Typically, we choose m=p^(1/2) as the number of predictors included at each split in building a random forest, where p is the total number of predictors. Unlike the predictions from the bagged trees will be highly correlated. Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Accordingly, on average (p − m)/p of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. It is this process that decorrelates the trees, thereby making the average of the resulting trees less variable and hence more reliable. The key diffffference between bagging and random

forests is the choice of predictor subset size.

4 Analysis and Conclusion

4.1 Results by SVR with kernels

Based on the training set, we fit the support vector regression model for Parkinsons dataset before and after variable selection, with the intercept and all corresponding slopes of variables, listed in Tab.4(a) and (b). Additionally, the untuned and tuned models by SVR with linear,radial, polynomial and sigmoid kernels are fitted and the outcomes are listed in Tab.4(c).

4.2 Results by Gradient Boosted Tree

Here we set the boosting iteration number=400, and interestingly find that both the training and the test error tend to be invariant when the iteration number is about 200. The outcomes from boosted tree with or without variable selection shown in Fig.6(a) and (b). Meanwhile, we do a tuning operation for gradient boosted tree by adjusting the number of split depth and the total number of trees. The final results before and after tuning are shown in Tab.5.

4.3 Results by Random Forest

Through performing random forest, we can obtain the outcomes in Tab.6 and Fig.7 before and after tuning the number of variables included in fitting model. Also, the importance of each variable in random forest model is drawn in Fig.8.

4.3 Comparison and Conclusion

After testing on the test dataset, we compute the prediction accuracy of both models by root mean squared error(RMSE), listed in Tab.4(c)-Tab.7(a),(b).

From Tab.4(c), we can see that SVR with radial kernel works best but worst with sigmoid kernel before and after tuning. From Tab.5 and Tab.6, random forest performs better than gradient boosted tree before tuning while the case is opposite after tuning. From Tab.7(a) and (b), stepwise selection seems to have no influence on SVR with linear kernel, however, the gradient boosted tree gets two times worse test RMSE than the one without stepwise selection. Overall, tuning parameters indeed brings about a great improvement in test RMSE regardless of the method type.

Tab.1 Description of 22 variables

Variables	Type	Description
subject	categorical	uniquely identifies each subject
age	Numeric	subject age
sex	categorical	male-0,female-1
Test_time	Numeric	time since recruitment into the trial. The integer

Variables	Type	Description
		part is the number of days since recruitment.
motor_UPDRS	Numeric	clinician's motor UPDRS score, linearly interpolated
total_UPDRS	Numeric	clinician's total UPDRS score, linearly interpolated
Jitter,Jitter_Abs, Jitter_RAP, Jitter_PPQ5, Jitter_DDP	Numeric	five measures of variation in fundamental frequency
Shimmer, Shimmer_dB, Shimmer_APQ3,Shimmer_ APQ5,Shimmer_APQ11, Shimmer_DDA	Numeric	six measures of variation in amplitude
NHR,HNR	Numeric	Two measures of ratio of noise to tonal components in the voice
RPDE	Numeric	- A nonlinear dynamical complexity measure
DFA	Numeric	Signal fractal scaling exponent
PPE	Numeric	A nonlinear measure of fundamental frequency variation
Total: 22 variables		5875 samples

Tab.2 Statistical descriptions (part of variables)

```
   test_time        motor_UPDRS      total_UPDRS
 Min.   : -4.263   Min.   : 5.038   Min.   : 7.00
 1st Qu.: 46.847   1st Qu.:15.000   1st Qu.:21.37
 Median : 91.523   Median :20.871   Median :27.58
 Mean   : 92.864   Mean   :21.296   Mean   :29.02
 3rd Qu.:138.445   3rd Qu.:27.596   3rd Qu.:36.40
 Max.   :215.490   Max.   :39.511   Max.   :54.99

   Shimmer_dB      Shimmer_APQ3      Shimmer_APQ5
 Min.   :0.026   Min.   :0.00161   Min.   :0.00194
 1st Qu.:0.175   1st Qu.:0.00928   1st Qu.:0.01079
 Median :0.253   Median :0.01370   Median :0.01594
```

```
      Mean   :0.311   Mean   :0.01716   Mean   :0.02014
      3rd Qu.:0.365   3rd Qu.:0.02057   3rd Qu.:0.02375
      Max.   :2.107   Max.   :0.16267   Max.   :0.16702
```

Tab.3 variable selection outcome (marked by *) by backward selection

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.087e+00  1.311e+00   1.592 0.111466
subject      4.698e-02  4.954e-03   9.483  < 2e-16 ***
age          7.598e-02  6.589e-03  11.531  < 2e-16 ***
sex         -1.840e+00  1.384e-01 -13.297  < 2e-16 ***
motor_UPDRS  1.209e+00  7.546e-03 160.171  < 2e-16 ***
Jitter      -4.191e+02  8.811e+01  -4.756 2.05e-06 ***
Jitter_Abs   1.774e+04  4.192e+03   4.232 2.37e-05 ***
Jitter_PPQ5  1.400e+02  7.063e+01   1.983 0.047478 *
Jitter_DDP   1.574e+02  3.661e+01   4.298 1.77e-05 ***
Shimmer     -5.002e+01  1.439e+01  -3.475 0.000516 ***
Shimmer_APQ5 8.750e+01  2.290e+01   3.821 0.000135 ***
Shimmer_APQ11 -3.295e+01 8.534e+00 -3.861 0.000115 ***
HNR         -9.416e-02  2.882e-02  -3.268 0.001094 **
RPDE         2.313e+00  7.631e-01   3.032 0.002450 **
DFA         -2.651e+00  9.062e-01  -2.926 0.003460 **
PPE         -3.839e+00  1.205e+00  -3.185 0.001461 **
```

Tab.4(a) all slopes of variables in SVR before variable selection

name	Intercept	subject	age	sex	test_time	motor_UPDRS
slope	0.04325446	0.050912038	0.054585547	-0.061388363	0.006704666	0.901243781
name	Jitter	Jitter_Abs	Jitter_RAP	Jitter_PPQ5	Jitter_DDP	
slope	-0.008412261	0.081360108	-0.024379412	0.000338388	-0.006608806	
name	Shimmer	Shimmer_dB	Shimmer_APQ3	Shimmer_APQ5	Shimmer_APQ11	Shimmer_DDA
slope	-0.055707684	0.021152256	-0.014221483	0.089615986	-0.06740332	-0.008805761
name	NHR	HNR	RPDE	DFA	PPE	
slope	-0.009966145	-0.01943393	0.034061277	-0.027898726	-0.042329173	

Tab.4(b) all slopes of variables in SVR after variable selection

name	intercept	subject	age	sex	motor_UPDRS	
slope	0.04160305	0.05084577	0.055368061	-0.062386609	0.902504089	

name	Jitter	Jitter_Abs	Jitter_PPQ5	Jitter_DDP	Shimmer	
slope	0.004041978	0.08130422	-0.003791853	-0.040305377	-0.05985931	
name	Shimmer_APQ5	Shimmer_APQ11	HNR	RPDE	DFA	PPE
slope	0.076007755	-0.058010696	-0.02079426	0.032228242	-0.027508683	-0.04344122

Tab.4(c) The outcomes of SVR with various kernel functions

Error/Methods	Linear kernel	Radial kernel	Polynomial	Sigmoid
Training RMSE(untuned)	3.263230	1.619339	3.211809	1148.464
Test RMSE(untuned)	3.277364	1.927910	11.84785	1155.724
Training RMSE(tuned)	0.3280411	0.01235608	2.349622	591.1260
Test RMSE(tuned)	0.3756980	0.05400357	2.406732	620.8180

Tab.5 The outcomes of gradient boosted tree

Error/Methods	gradient boosted tree (untuned)	gradient boosted tree (tuned)
Training RMSE	0.2144060	0.0003068737
Test RMSE	0.2700032	0.0073808500

Tab.6 The outcomes of random forest

Error/method	Random Forest (untuned)	Random Forest (tuned)
Training RMSE	0.001778002	0.002654275
Test RMSE	0.027274570	0.012694700

Tab.7(a) comparison of models performance without variable selection

Error/method	SVR with linear kernel	Regression tree with Boosting
Training RMSE	3.263230	0.2144060
Test RMSE	3.277364	0.2700032

Tab.7(b) comparison of models performance with variable selection

Error/method	SVR with linear kernel	Regression tree with Boosting
Training RMSE	3.261565	0.4596417
Test RMSE	3.280485	0.550301

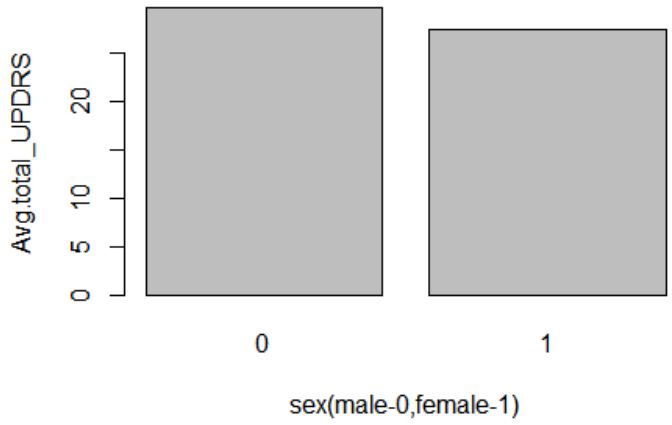

Fig.1 bar chart for sex vs avg.total_UPDRS

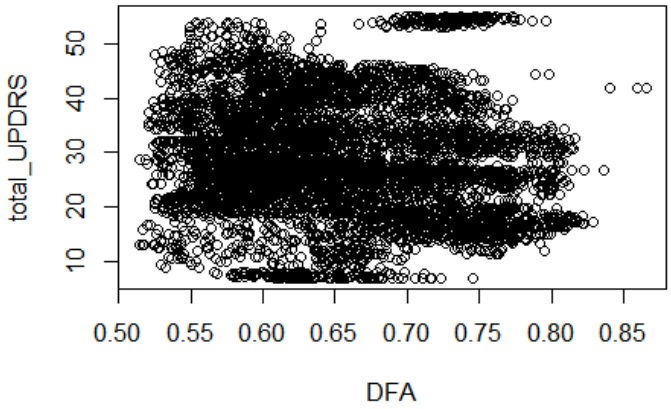

Fig.2 scatterplot of total_UPDRS vs DFA

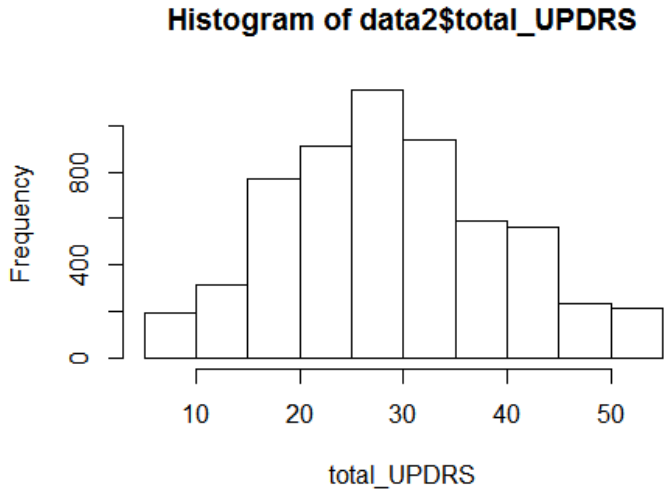

Fig.3 Histogram of total_UPDRS

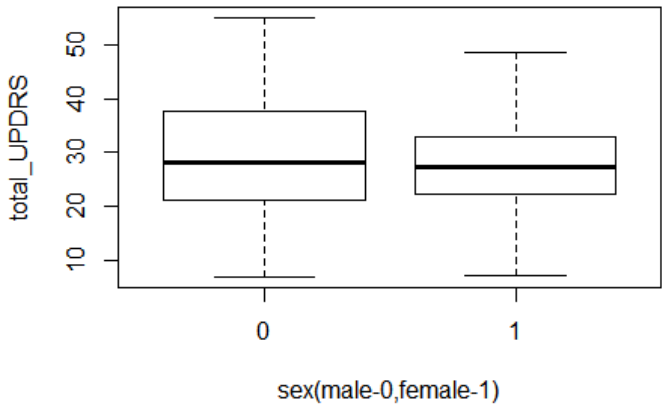

Fig4. boxplot of total_UPDRS vs sex

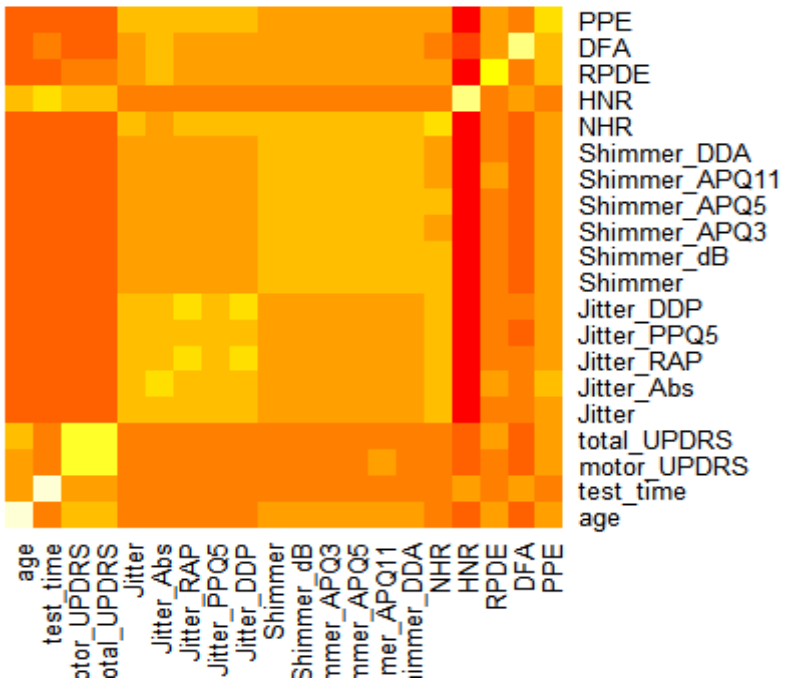

Fig.5 heatmap of variable correlations

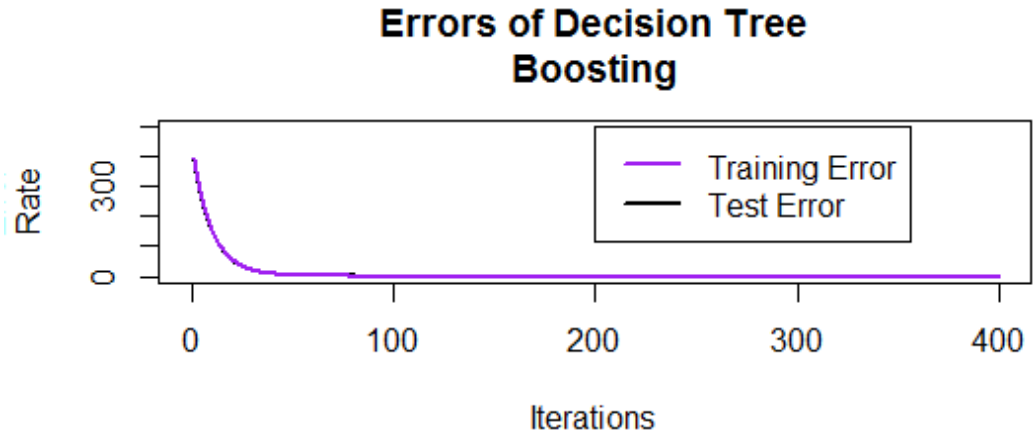

Fig.6(a) training and test error at each iteration without variable selection

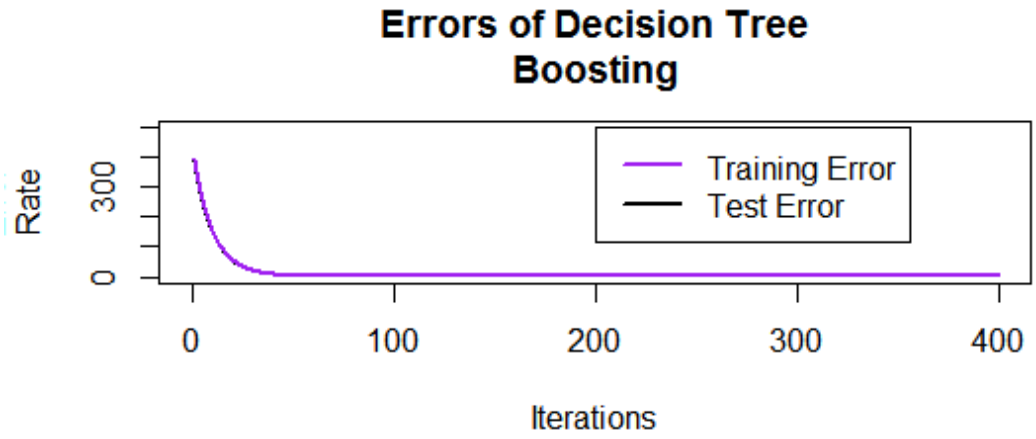

Fig.6(b) training and test error at each iteration with variable selection

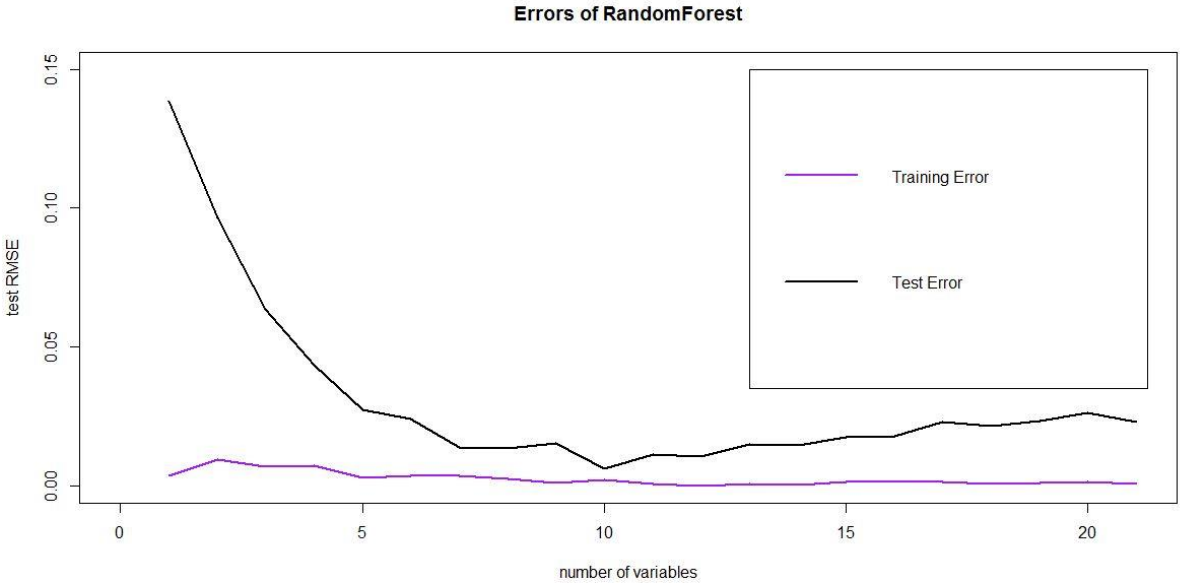

Fig.7 Training and test errors when tuning the number of variables

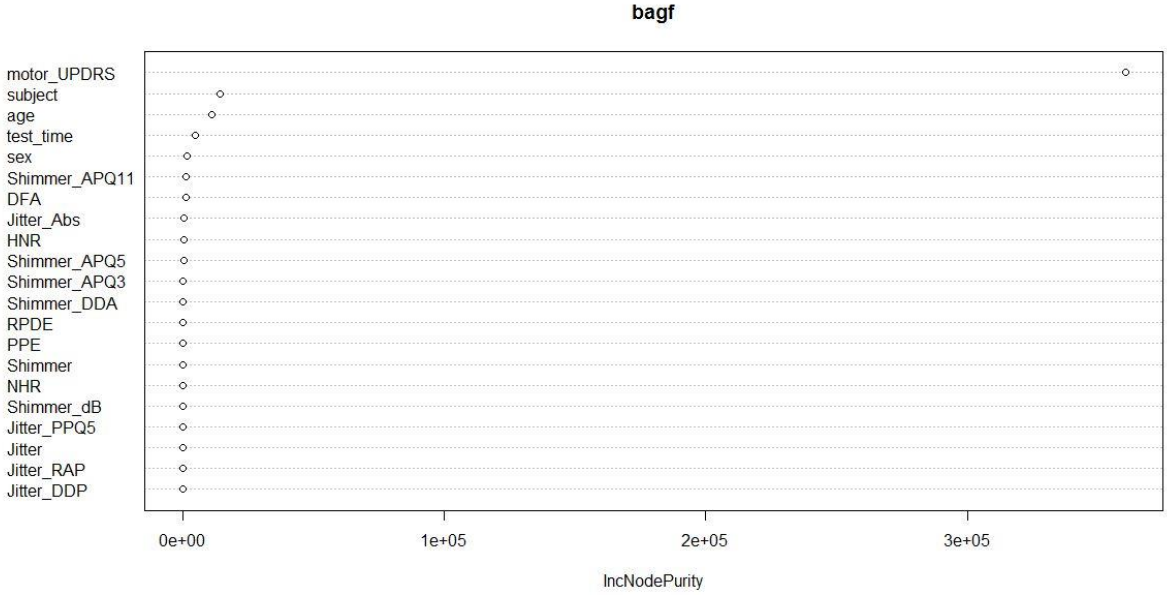

Fig.8 the importance of each variable in random forest model

Appendix

```
## import the original dataset
setwd("E:/Desk/2019/STA6704-data mining II/proj")
rm(list=ls())
##############################
#data2= read.table("parkinsons_updrs.data.txt",header=TRUE,sep=",",dec=".")
data2= read.table(file.choose(),header=TRUE,sep=",",dec=".")
dim(data2)
str(data2)
data2_sta=summary(data2)
#partion dataset into traininng and validation sets
set.seed(10)
train.index=sample(c(1:dim(data2)[1]),0.6*dim(data2)[1])
traind=data2[train.index,]
validd=data2[-train.index,]
names(traind)
names(validd)
#variable selection
################################
library(glmnet)
data2.fit=lm(total_UPDRS~.,data=traind)
## forward
forward.fit=step(data2.fit,direction = "forward")
summary(forward.fit)      #AIC=8241.77    15 variables
##backward
backward.fit=step(data2.fit,direction = "backward")
summary(backward.fit)    #AIC=8232.99    15 variables
```

```
##both

both.fit=step(data2.fit,direction = "both")

summary(both.fit)    #AIC=8232.99      15 variables

library(dplyr)

data2=select(data2,-test_time,-Jitter_RAP,-Shimmer_dB,-Shimmer_APQ3,-Shimmer_DDA,-NHR)

dim(data2)

# SVR with     kernels

################################

library(e1071)

## SVR with linear kernel

################################

data.svm =svm(total_UPDRS~.,data=traind,kernel="linear", shrinking=TRUE,cross=5)

summary(data.svm)

data.svmpred=predict(data.svm,validd)

data.svmpred1=predict(data.svm,traind)
sqrt(mean((data.svmpred-validd$total_UPDRS)^2))

sqrt(mean((data.svmpred1-traind$total_UPDRS)^2))

#Find value of W (slope)

W = t(data.svm$coefs) %*% data.svm$SV

write.csv(W,'E:/Desk/2019/STA6704-data mining II/proj/svr_slope_stepwise.csv')

#Find value of b (intercept)

b = data.svm$rho #0.04325446      0.04160305

# perform a grid search by tune()

################################

tuneResult <- tune(svm,total_UPDRS~., data =traind,kernel="linear",

                                ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9)))

print(tuneResult)

plot(tuneResult)
```

```
plot(tuneResult)
bestml=tuneResult$best.model
summary(bestml)
tunepre.t=predict(bestml,traind)
tunepre.v=predict(bestml,validd)
sqrt(mean(tunepre.t-traind$total_UPDRS)^2)
sqrt(mean(tunepre.v-validd$total_UPDRS)^2)


## kernels: polynoimal,Gaussian(radial),signmoid
##radial kernel
###################################
data.svm.r =svm(total_UPDRS~.,data=traind,kernel="radial",
                        shrinking=TRUE,cross=5)
summary(data.svm.r)
data.svm.rpred=predict(data.svm.r,validd)
data.svm.rpred1=predict(data.svm.r,traind)
sqrt(mean((data.svm.rpred-validd$total_UPDRS)^2))
sqrt(mean((data.svm.rpred1-traind$total_UPDRS)^2))
#Find value of W (slope)
W1 = t(data.svm.r$coefs) %*% data.svm.r$SV
write.csv(W1,'E:/Desk/2019/STA6704-data mining II/proj/svr_radial.csv')
#Find value of b (intercept)
b1 = data.svm.r$rho #0.001403138
## tune with radial kernel
tuneResult <- tune(svm,total_UPDRS~., data =traind,kernel="radial",
                                ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9)))
print(tuneResult)
# Draw the tuning graph
```

```
plot(tuneResult.r)

bestmr=tuneResult.r$best.model

summary(bestmr)

tunepre.rt=predict(bestmr,traind)

tunepre.rv=predict(bestmr,validd)

sqrt(mean(tunepre.rt-traind$total_UPDRS)^2)

sqrt(mean(tunepre.rv-validd$total_UPDRS)^2)


##polynomial kernel

data.svm.p =svm(total_UPDRS~.,data=traind,kernel="polynomial",

                        shrinking=TRUE,cross=5)

summary(data.svm.p)

data.svm.ppred=predict(data.svm.p,validd)

data.svm.ppred1=predict(data.svm.p,traind)

sqrt(mean((data.svm.ppred-validd$total_UPDRS)^2))

sqrt(mean((data.svm.ppred1-traind$total_UPDRS)^2))

#Find value of W (slope)

W2 = t(data.svm.p$coefs) %*% data.svm.p$SV

write.csv(W2,'E:/Desk/2019/STA6704-data mining II/proj/svr_polynomial.csv')

#Find value of b (intercept)

b2 = data.svm.p$rho #

## tune with polynomial kernel

tuneResult <- tune(svm,total_UPDRS~., data =traind,kernel="polynomial",

                           ranges = list(epsilon = seq(0,0.5,0.1), cost = 2^(2:6)))

print(tuneResult)

plot(tuneResult)

tunedModel <- tuneResult$best.model

tunedModelv=predict(tunedModel,validd)
```

```
tunedModelt <- predict(tunedModel, traind)
sqrt(mean((tunedModelv-validd$total_UPDRS)^2))
sqrt(mean((tunedModelt-traind$total_UPDRS)^2))

##sigmoid kernel
data.svm.s =svm(total_UPDRS~.,data=traind,kernel="sigmoid",
                shrinking=TRUE,cross=5)
summary(data.svm.s)
data.svm.spred=predict(data.svm.s,validd)
data.svm.spred1=predict(data.svm.s,traind)
sqrt(mean((data.svm.spred-validd$total_UPDRS)^2))
sqrt(mean((data.svm.spred1-traind$total_UPDRS)^2))
#Find value of W (slope)
W3 = t(data.svm.s$coefs) %*% data.svm.s$SV
write.csv(W3,'E:/Desk/2019/STA6704-data mining II/proj/svr_sigmoid.csv')
#Find value of b (intercept)
b3 = data.svm.s$rho #
## tune with polynomial kernel
tuneResult1 <- tune(svm,total_UPDRS~., data =traind,kernel="sigmoid",
                    ranges = list(epsilon = seq(0,0.5,0.1), cost = 2^(2:6)))
print(tuneResult1)
tunedModel <- tuneResult1$best.model
tunedModelv=predict(tunedModel,validd)
tunedModelt <- predict(tunedModel, traind)
sqrt(mean((tunedModelv-validd$total_UPDRS)^2))
sqrt(mean((tunedModelt-traind$total_UPDRS)^2))

## ensemble tree by boosting with stepwise
```

```
##############################################
set.seed(10)
train.index=sample(c(1:dim(data2)[1]),0.6*dim(data2)[1])
train_df=data2[train.index,]
valid_df=data2[-train.index,]
library(dplyr)
x_train= train_df[,c(1:4,6:16)]
y_train= train_df[,5]
x_test=    valid_df[,c(1:4,6:16)]
y_test=    valid_df[,5]
names(data2)
str(x_train)
niter <- 400    # literation number
loss <- function(y,yhat){0.5*(y - yhat)^2} # mean loss function
##################
library(rpart)
v=0.05# learning rate/shrinkage parameter
#train_df=Boston[train,]
fit=rpart(total_UPDRS~.,data=train_df)
train_yp=predict(fit,x_train)              # y of training set
test_yp=predict(fit,x_test)                 # y of test set
train_df$yr=train_df$total_UPDRS - v*train_yp #
train_YP=v*train_yp
test_YP=v*test_yp
train_errors=rep(0,niter )
test_errors=rep(0,niter )
for(i in seq(niter))    {
   fit=rpart(yr~subject+age+sex+motor_UPDRS+Jitter+
```

```
                    Jitter_Abs+Jitter_PPQ5+Jitter_DDP+Shimmer+
                    Shimmer_APQ5+Shimmer_APQ11+
                    +HNR+RPDE+DFA+PPE,data=train_df,
                  control = list(minsplit = 10, maxdepth = 12, xval = 10))
  train_yp=predict(fit,x_train)
  test_yp=predict(fit,x_test)
  train_df$yr=train_df$yr - v*train_yp
  train_YP=cbind(train_YP,v*train_yp)
  test_YP=cbind(test_YP,v*test_yp)
  ytrain_hat=apply(train_YP,1,sum)
  ytest_hat=apply(test_YP,1,sum)
  train_error = mean(loss(y_train,ytrain_hat))
  train_errors[i] <- train_error
  test_errors[i] <- mean(loss(y_test,ytest_hat))## without stepwise,    with stepwise
  cat(i,"error:",train_error,"\n")       # 200                  ,0.214406            0.4596417
  cat(i,"error:",test_errors[i],"\n")    # 200 iteration, 0.2700032         0.550301
  print(i)
}
par(mfrow=c(1,2))
#plot(errors,main="Training errors of Decision Tree Boosting")
plot(seq(1,niter ),test_errors,type="l",xlim=c(0,400),ylim=c(0,500),ylab="Error
Rate",xlab="Iterations",lwd=2, main='Errors of Decision Tree
Boosting')
lines(train_errors,lwd=2,col="purple")
legend(200,500,c("Training Error","Test Error"),
           col=c("purple","black"),lwd=2)

## ensemble tree by boosting without stepwise
```

```
##############################################

data2= read.table(file.choose(),header=TRUE,sep=",",dec=".")

dim(data2)

str(data2)

data2_sta=summary(data2)

set.seed(10)

train.index=sample(c(1:dim(data2)[1]),0.6*dim(data2)[1])

train_df=data2[train.index,]

valid_df=data2[-train.index,]

library(dplyr)

x_train= train_df[,c(1:5,7:22)]

y_train= train_df[,6]

x_test=    valid_df[,c(1:5,7:22)]

y_test=    valid_df[,6]

names(data2)

str(x_train)

niter <- 400    # literation number

loss <- function(y,yhat){0.5*(y - yhat)^2} # mean loss function

##################

library(rpart)

v=0.05# learning rate/shrinkage parameter

#train_df=Boston[train,]

fit=rpart(total_UPDRS~.,data=train_df)

train_yp=predict(fit,x_train)              # y of training set

test_yp=predict(fit,x_test)                 # y of test set

train_df$yr=train_df$total_UPDRS - v*train_yp #

train_YP=v*train_yp

test_YP=v*test_yp
```

```
train_errors=rep(0,niter )
test_errors=rep(0,niter )
for(i in seq(niter))    {
            fit=rpart(yr~subject+age+sex++test_time+motor_UPDRS+Jitter+
                        Jitter_Abs+Jitter_RAP+Jitter_PPQ5+Jitter_DDP+Shimmer+
                        Shimmer_dB+Shimmer_APQ3+Shimmer_APQ5+Shimmer_APQ11+
                        Shimmer_DDA+NHR+HNR+RPDE+DFA+PPE,data=train_df,
          control = list(minsplit = 10, maxdepth = 12, xval = 10))    ## tune by control()
   train_yp=predict(fit,x_train)
   test_yp=predict(fit,x_test)
   train_df$yr=train_df$yr - v*train_yp
   train_YP=cbind(train_YP,v*train_yp)
   test_YP=cbind(test_YP,v*test_yp)
   ytrain_hat=apply(train_YP,1,sum)
   ytest_hat=apply(test_YP,1,sum)
   train_error = mean(loss(y_train,ytrain_hat))
   train_errors[i] <- train_error
   test_errors[i] <- mean(loss(y_test,ytest_hat))## without stepwise,    with stepwise
   cat(i,"error:",train_error,"\n")       # 200                  ,0.214406            0.4596417
   cat(i,"error:",test_errors[i],"\n")    # 200 iteration, 0.2700032         0.550301
   print(i)
}
par(mfrow=c(1,2))
#plot(errors,main="Training errors of Decision Tree Boosting")
plot(seq(1,niter ),test_errors,type="l",xlim=c(0,400),ylim=c(0,500),ylab="Error
        Rate",xlab="Iterations",lwd=2, main='Errors of Decision Tree
        Boosting')
lines(train_errors,lwd=2,col="purple")
```

```
legend(200,500,c("Training Error","Test Error"),
            col=c("purple","black"),lwd=2)
## gradient boosted tree    tuned by train()
#########################################################
set.seed(10)
train.index=sample(c(1:dim(data2)[1]),0.6*dim(data2)[1])
train_df=data2[train.index,]
valid_df=data2[-train.index,]
library(dplyr)
x_train= train_df[,c(1:5,7:22)]
y_train= train_df[,6]
x_test=    valid_df[,c(1:5,7:22)]
y_test=    valid_df[,6]
## tune by train()
install.packages("tidyr")
install.packages("caret")
library(caret)
library(kernlab)
fitControl <- trainControl(## 5-fold CV
   method = "repeatedcv",
   number = 5,
   ## repeated 5 times
   repeats = 5)
## set a search grid
gbmGrid <-    expand.grid(interaction.depth = c(1, 5, 9),
                                    n.trees = (5:15)*100,
                                    shrinkage = 0.1,
                                    n.minobsinnode = 20)
```

```
nrow(gbmGrid) # 33 times

## tuning and using gradient boosting

set.seed(825)

gbmFit2 <- train(total_UPDRS~ ., data = train_df,

                 method = "gbm",

                 trControl = fitControl,

                 verbose = FALSE,

                 ## Now specify the exact models

                 ## to evaluate:

                 tuneGrid = gbmGrid)

gbmFit2

##plot depth vs iterations

trellis.par.set(caretTheme())

plot(gbmFit2)

# get final best model

tunetreet=predict(gbmFit2,train_df)

tunetreev=predict(gbmFit2,valid_df)

sqrt(mean(tunetreet-train_df$total_UPDRS)^2)    #0.0003068737

sqrt(mean(tunetreev-valid_df$total_UPDRS)^2)    #0.00738085

######################

## random forest

####################

library(randomForest)

rfNews()

bagf=randomForest(total_UPDRS~.,data=train_df,mtry=21,improtance=TRUE)

bagp=predict(bagf,newdata=valid_df)

bagp1=predict(bagf,newdata=train_df)

plot(bagp,valid_df$total_UPDRS)
```

```
abline(0,1)
sqrt(mean(bagp-valid_df$total_UPDRS)^2)        # 0.02727457
sqrt(mean(bagp1-train_df$total_UPDRS)^2)       # 0.001778002
importance(bagf)
varImpPlot(bagf)
###tune the random forest by changing mtry from 1:21
trainE=rep(0,21)
validE=rep(0,21)
for (i in 1:21)
   {
bagf=randomForest(total_UPDRS~.,data=train_df,mtry=i,improtance=TRUE)
bagp=predict(bagf,newdata=valid_df)
bagp1=predict(bagf,newdata=train_df)
validE[i]=sqrt(mean(bagp-valid_df$total_UPDRS)^2)
trainE[i]=sqrt(mean(bagp1-train_df$total_UPDRS)^2)
print(i)
print(trainE[i])
print(validE[i])
}
plot(seq(1,21),validE,type="l",xlim=c(0,21),ylim=c(0,0.15),ylab="test RMSE",
        xlab="number of variables",lwd=2, main='Errors of RandomForest')
lines(trainE,lwd=2,col="purple")
legend(13,0.15,c("Training Error","Test Error"),
            col=c("purple","black"),lwd=2)
##optimal mtry=10 random forest select 10 variables with the best outcome
bagf=randomForest(total_UPDRS~.,data=train_df,mtry=10,improtance=TRUE)
bagp=predict(bagf,newdata=valid_df)
bagp1=predict(bagf,newdata=train_df)
```

```
sqrt(mean(bagp-valid_df$total_UPDRS)^2)    #0.0126947

sqrt(mean(bagp1-train_df$total_UPDRS)^2)    # 0.002654275

importance(bagf)
```