S$P500 stock index prediction

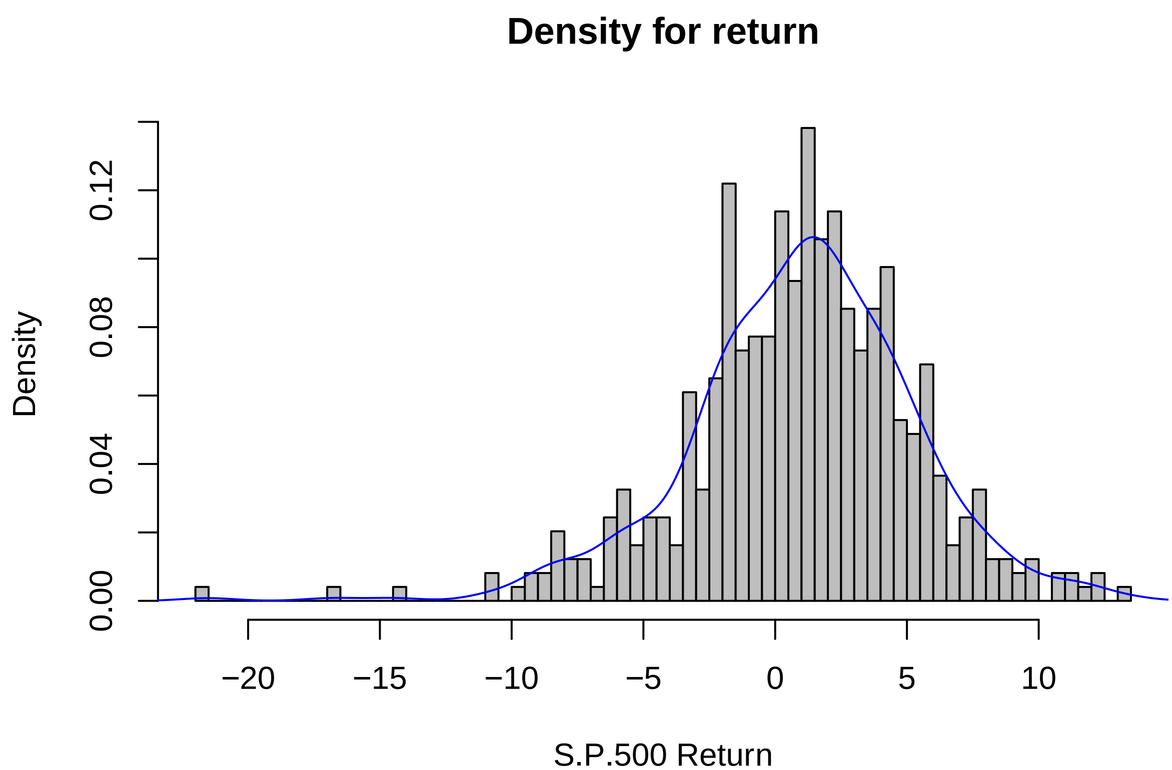
Zhihao Liu

January 23, 2017

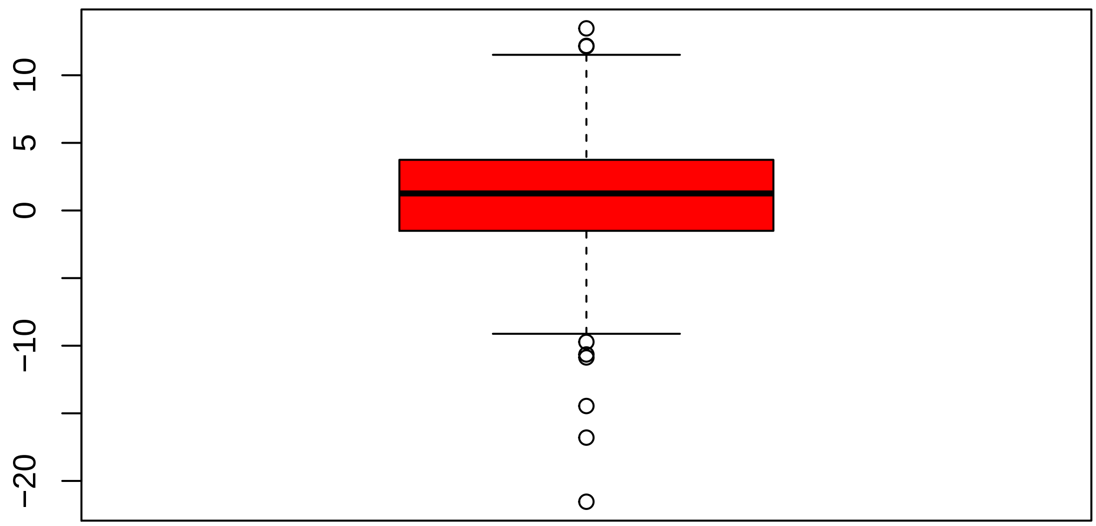
Basic data description

```
Mydata <- read.csv("D:/Statistics material/Convex capital management/dataset for prediction.csv")

hist(Mydata$S.P.500.Returns,breaks=100,col="grey",border="black",xlab = "S.P.500 Return", main="Density
lines(density(Mydata$S.P.500.Returns),col="blue")
```

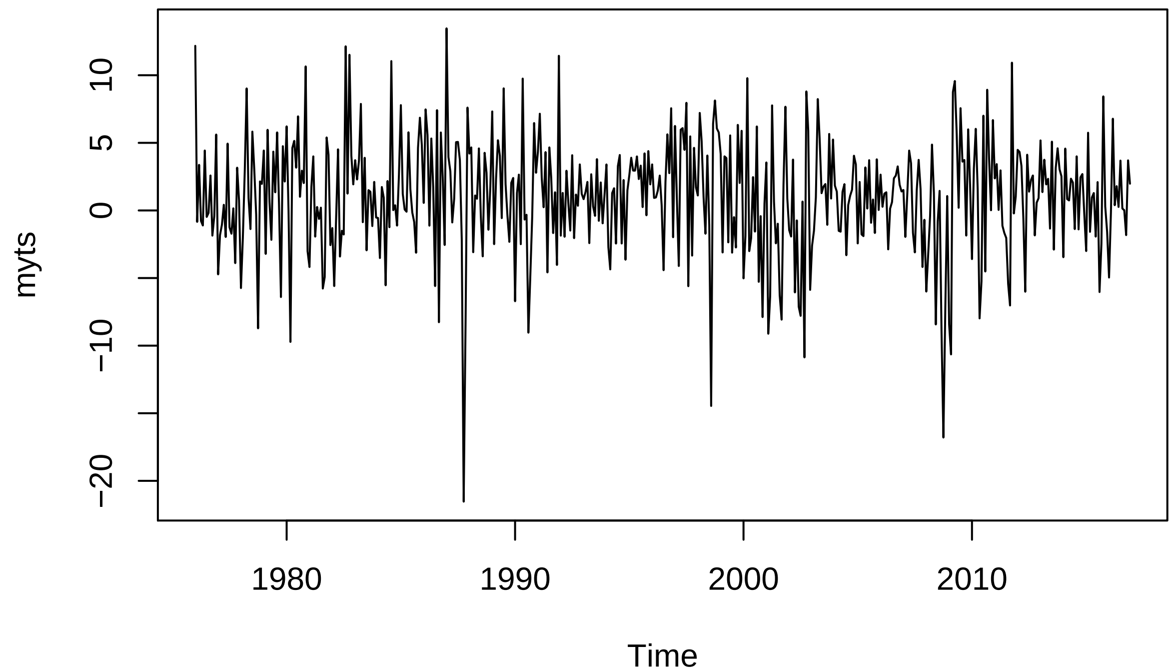



```
boxplot(Mydata$S.P.500.Returns,col="red")
```



set returns as time series data

```
myts<-ts(Mydata$S.P.500.Returns,start = c(1976,1),end =c(2016,12),frequency = 12)
plot(myts)
```



Split data

```
train_x=Mydata[c(1:300),c(3:10)]
train_y=myts[c(1:300)]
train_x=ts(train_x,start = c(1976,1),end=c(2000,12),frequency = 12)
train_y=ts(train_y,start = c(1976,1),end=c(2000,12),frequency = 12)
traindata=data.frame(x=train_x,y=train_y)
test_x=Mydata[c(301:492),c(3:10)]
test_y=myts[c(301:492)]
test_x=ts(test_x,start = c(2001,1),end=c(2016,12),frequency = 12)
test_y=ts(test_y,start = c(2001,1),end=c(2016,12),frequency = 12)
testdata=data.frame(x=test_x,y=test_y)
```

Time series ARIMA model and mutiple linear regression model combine

```
library(tseries)
adf.test(myts)
```

```
## Warning in adf.test(myts): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  myts
## Dickey-Fuller = -7.3993, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

```r
library(forecast)
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: timeDate

## This is forecast 7.3
```

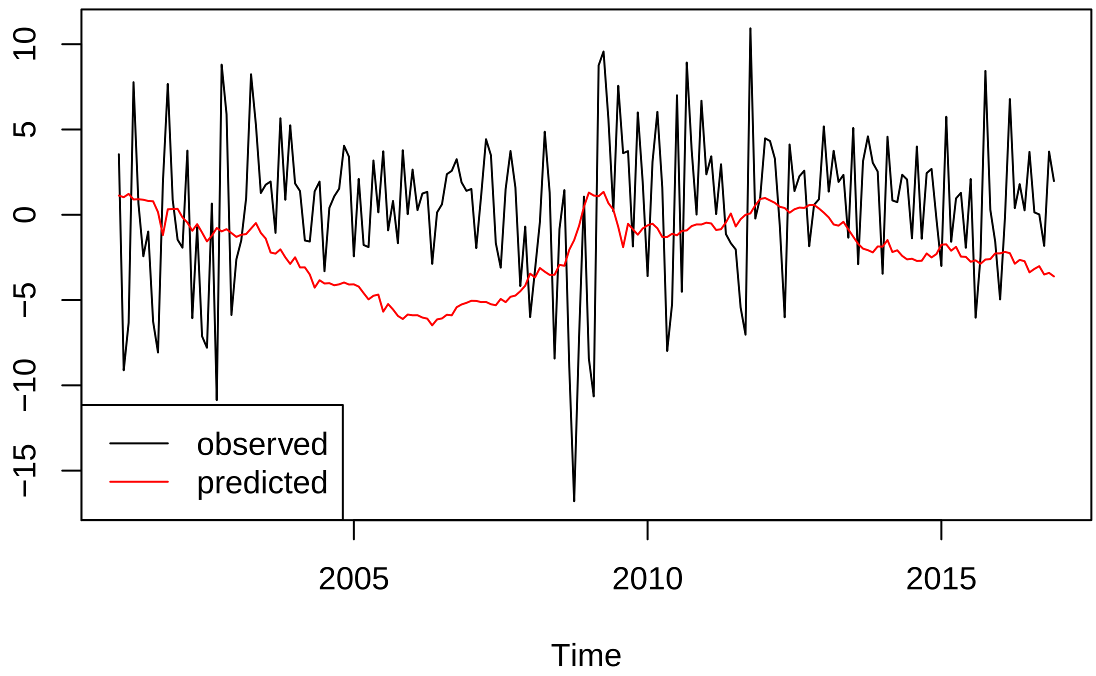
```r
tsmodel<-auto.arima(train_y,xreg = train_x)
summary(tsmodel)
```

```
## Series: train_y
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##           ar1  intercept      ISM  FEDFUNDS     DGS10  CPIAUCSL  UMCSENT
##       -0.0607    10.7191  -0.1065    0.0558   -0.3672    0.0431   0.0382
## s.e.   0.0593     3.8271   0.0471    0.1632    0.2821    0.0420   0.0348
##       ALTSALES    HSN1F  CSUSHPINSA
##        -0.2431  -0.0001     -0.0944
## s.e.    0.2381   0.0036      0.0730
##
## sigma^2 estimated as 18.16:  log likelihood=-855.5
## AIC=1732.99   AICc=1733.91   BIC=1773.73
##
## Training set error measures:
##                       ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 0.001993927 4.190031 3.189083 91.97087 160.9453 0.6915823
##                     ACF1
## Training set -0.00323567
```

```r
predts<-predict(tsmodel,n.ahead = 16*12,newxreg = test_x)
library(hydroGOF)
mse(predts$pred,test_y)
```

```
## [1] 29.40489
```

```r
ts.plot(test_y,predts$pred,col=c(1,2),lty=c(1,1))
legend("bottomleft",c("observed","predicted"),col=c(1,2),lty=c(1,1))
```



KNN Regression

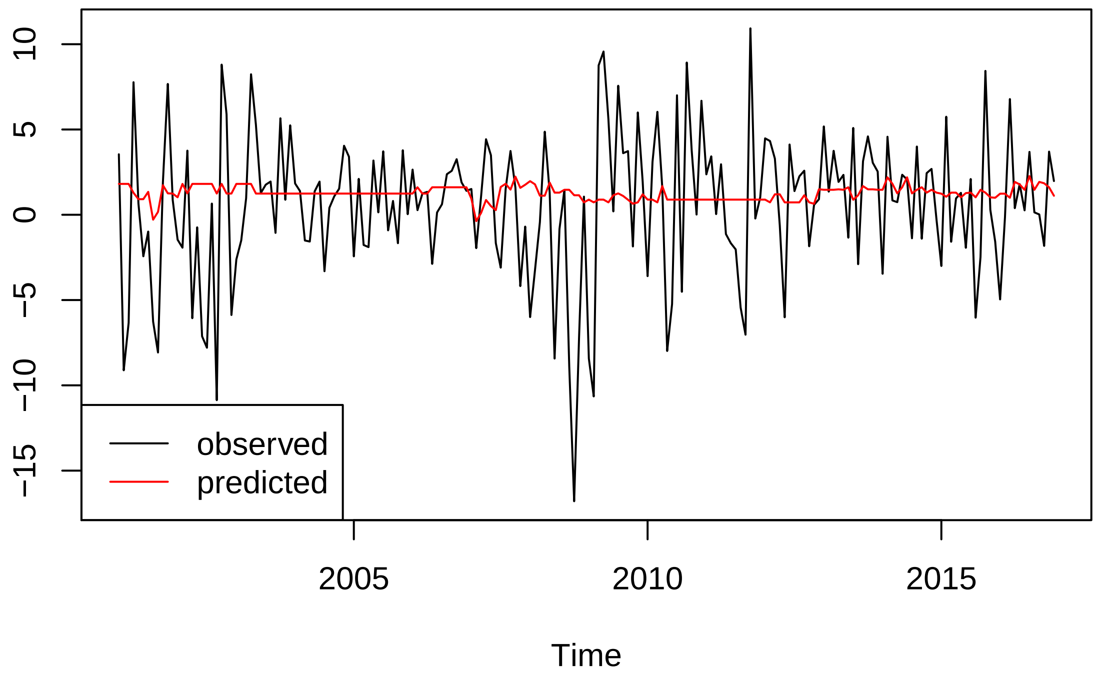
```
Mydata2=read.csv("D:/Statistics material/Convex capital management/dataset2.csv")
train.x=as.matrix(Mydata2[c(1:300),c(3:11)])
train.y=Mydata2[c(1:300),2]
train.x=ts(train.x,start = c(1976,1),end=c(2000,12),frequency = 12)
train.y=ts(train.y,start = c(1976,1),end=c(2000,12),frequency = 12)
train.data=data.frame(x=train.x,y=train.y)
test.x=as.matrix(Mydata2[c(301:492),c(3:11)])
test.y=Mydata2[c(301:492),2]
test.x=ts(test.x,start = c(2001,1),end=c(2016,12),frequency = 12)
test.y=ts(test.y,start = c(2001,1),end=c(2016,12),frequency = 12)
test.data=data.frame(x=test.x,y=test.y)
```

```
library(class)
library(FNN)
```

```
##
## Attaching package: 'FNN'

## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```
KNNpredict<-knn.reg(train.x,test.x,train.y,k=20)
ts.plot(test.y,KNNpredict$pred,col=c(1,2),lty=c(1,1))
legend("bottomleft",c("observed","predicted"),col=c(1,2),lty=c(1,1))
```




```r
mse(KNNpredict$pred,test.y)
```

```
## [1] 18.64799
```

GAM with spline Prediction

```r
library(splines)
library(gam)
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.14
```

```r
smooth.spline(train.data$x.ISM,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.ISM, y = train.data$y)
##
## Smoothing Parameter  spar= 1.499956  lambda= 40.64147 (25 iterations)
## Equivalent Degrees of Freedom (Df): 2.005546
## Penalized Criterion: 2980.141
## GCV: 18.35142
```

```r
smooth.spline(train.data$x.FEDFUNDS,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.FEDFUNDS, y = train.data$y)
```

```
##
## Smoothing Parameter  spar= 1.3221  lambda= 0.2944697 (11 iterations)
## Equivalent Degrees of Freedom (Df): 2.829615
## Penalized Criterion: 4691.552
## GCV: 18.46002
```

```r
smooth.spline(train.data$x.DGS10,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.DGS10, y = train.data$y)
##
## Smoothing Parameter  spar= 1.171101  lambda= 0.07790088 (15 iterations)
## Equivalent Degrees of Freedom (Df): 3.541058
## Penalized Criterion: 4628.25
## GCV: 18.50078
```

```r
smooth.spline(train.data$x.CPIAUCSL,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.CPIAUCSL, y = train.data$y)
##
## Smoothing Parameter  spar= 1.49996  lambda= 51.29152 (25 iterations)
## Equivalent Degrees of Freedom (Df): 2.014927
## Penalized Criterion: 5186.947
## GCV: 18.60867
```

```r
smooth.spline(train.data$x.UMCSENT,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.UMCSENT, y = train.data$y)
##
## Smoothing Parameter  spar= 1.49994  lambda= 24.42988 (24 iterations)
## Equivalent Degrees of Freedom (Df): 2.016908
## Penalized Criterion: 3895.404
## GCV: 18.61963
```

```r
smooth.spline(train.data$x.ALTSALES,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.ALTSALES, y = train.data$y)
##
## Smoothing Parameter  spar= 1.150766  lambda= 0.0107334 (13 iterations)
## Equivalent Degrees of Freedom (Df): 4.913939
## Penalized Criterion: 5207.638
## GCV: 18.30465
```

```r
smooth.spline(train.data$x.HSN1F,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.HSN1F, y = train.data$y)
##
## Smoothing Parameter  spar= 1.499945  lambda= 29.59261 (25 iterations)
## Equivalent Degrees of Freedom (Df): 2.014742
## Penalized Criterion: 3918.008
## GCV: 18.62173
```

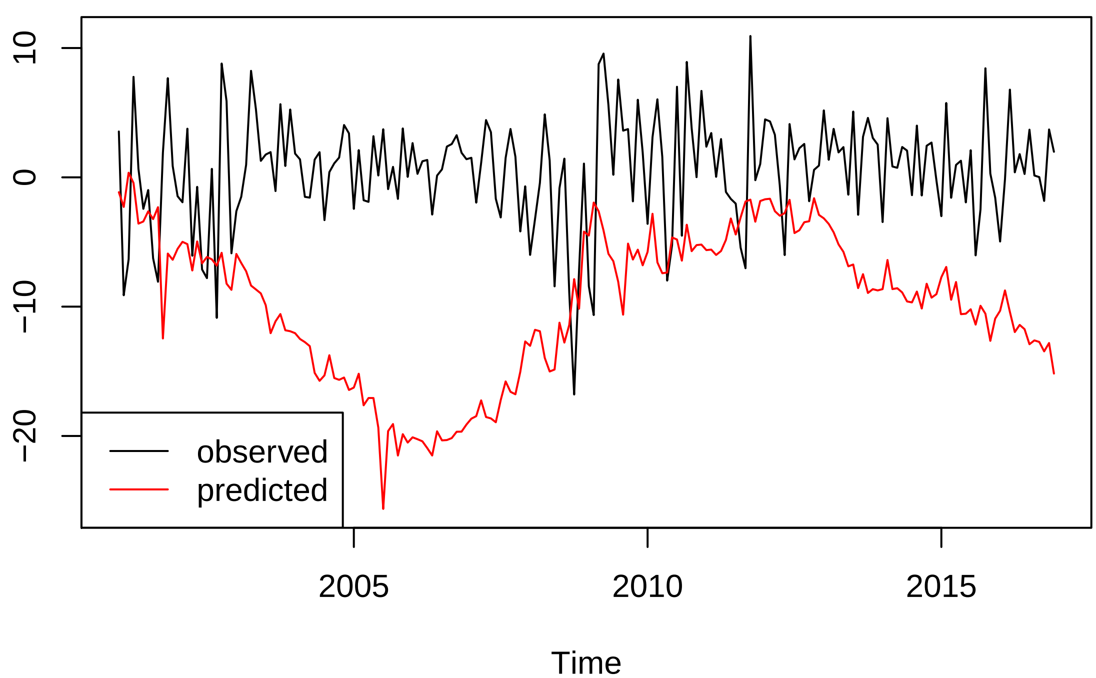
```r
smooth.spline(train.data$x.CSUSHPINSA,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.CSUSHPINSA, y = train.data$y)
##
## Smoothing Parameter  spar= 1.499935  lambda= 0.648352 (24 iterations)
## Equivalent Degrees of Freedom (Df): 2.574179
## Penalized Criterion: 5362.476
## GCV: 18.63495
```

```
smooth.spline(train.data$x.t.1.return,train.data$y)
```

```
## Call:
## smooth.spline(x = train.data$x.t.1.return, y = train.data$y)
##
## Smoothing Parameter  spar= 1.184129  lambda= 0.01474934 (13 iterations)
## Equivalent Degrees of Freedom (Df): 4.325691
## Penalized Criterion: 5343.476
## GCV: 18.53453
```

```
gam_pred<-gam(y~s(x.ISM,2.005546)+s(x.FEDFUNDS,2.829615)+s(x.DGS10,3.541058)+s(x.CPIAUCSL,2.014927)+s(x
pred<-predict(gam_pred,newdata=test.data)
ts.plot(test.y,pred,col=c(1,2),lty=c(1,1))
legend("bottomleft",c("observed","predicted"),col=c(1,2),lty=c(1,1))
```




```
mse(pred,test.y)
```

```
## [1] 157.8719
```

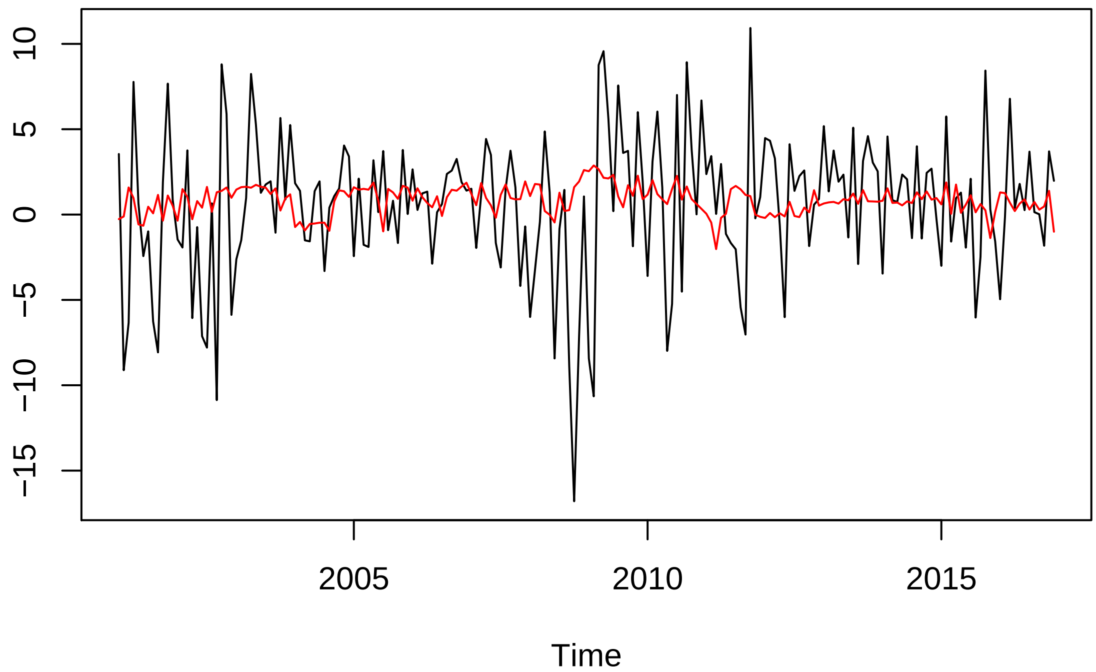
Random Forest

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```
rfpredict<-randomForest(y~.,data=train.data,mtry=3,nodesize=5,importance=TRUE)
predrf<-predict(rfpredict,newdata=test.data)
ts.plot(test.y,predrf,col=c(1,2))
```




```
mse(predrf,test.y)
```

```
## [1] 18.40225
```

boosting

```
library(gbm)
```

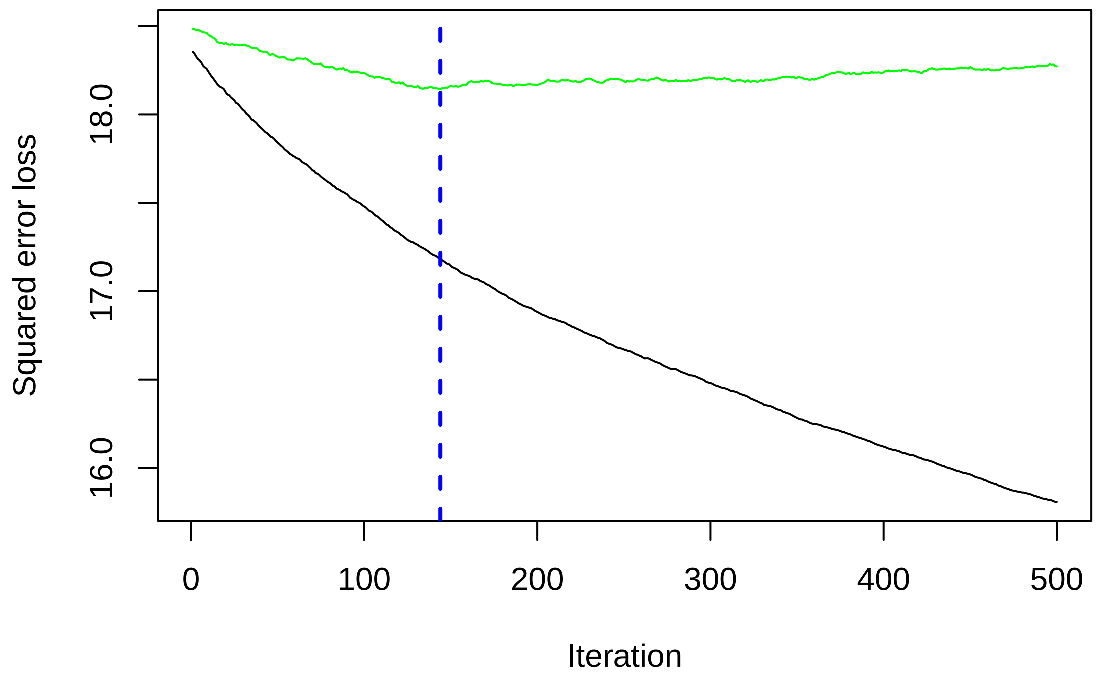
```
## Loading required package: survival
```

```
## Loading required package: lattice
```

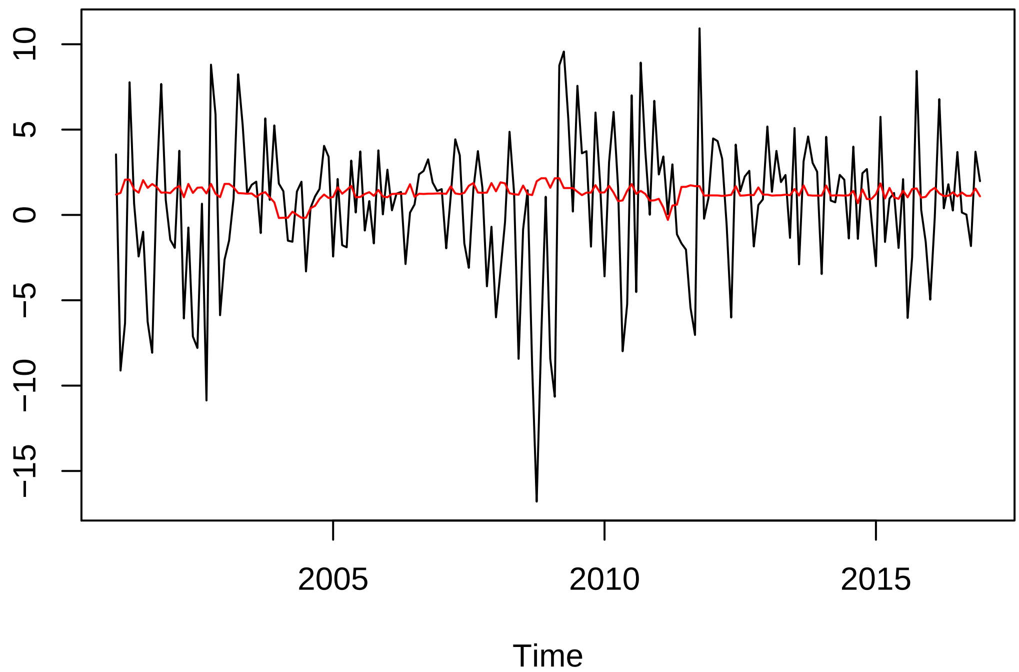
```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
gbmpredict<-gbm(y~., data=train.data,distribution = "gaussian",n.trees=500, shrinkage=0.01,interaction.
usetree = gbm.perf(gbmpredict, method="cv")
```




```
predgbm<-predict(gbmpredict,newdata=test.data, n.trees=usetree)
ts.plot(test.y,predgbm,col=c(1,2))
```




```
mse(predgbm,test.y)
```

```
## [1] 18.87073
```