comparing_oldnewmix

Ali Sanjari

Monday, June 13, 2016

This document contained an R implementation for creating a indext trading stratagy using rpart library, with a goal to improve the current model. We start with the current model which is based on a longterm learning stratgy. We introduce a short term learning stratgy, and shows that due to its good performance, one can obtain a better performance by mixing the two model .

The current model

The current model uses the *90th percentile* of the last 200 observation as trading signal, if the last vol difference be lower than the 90th percentile, we consider the market bearish and take 0.5 position in VIX and -1 in SPX. Otherwise, we go +1 long in the SPX.

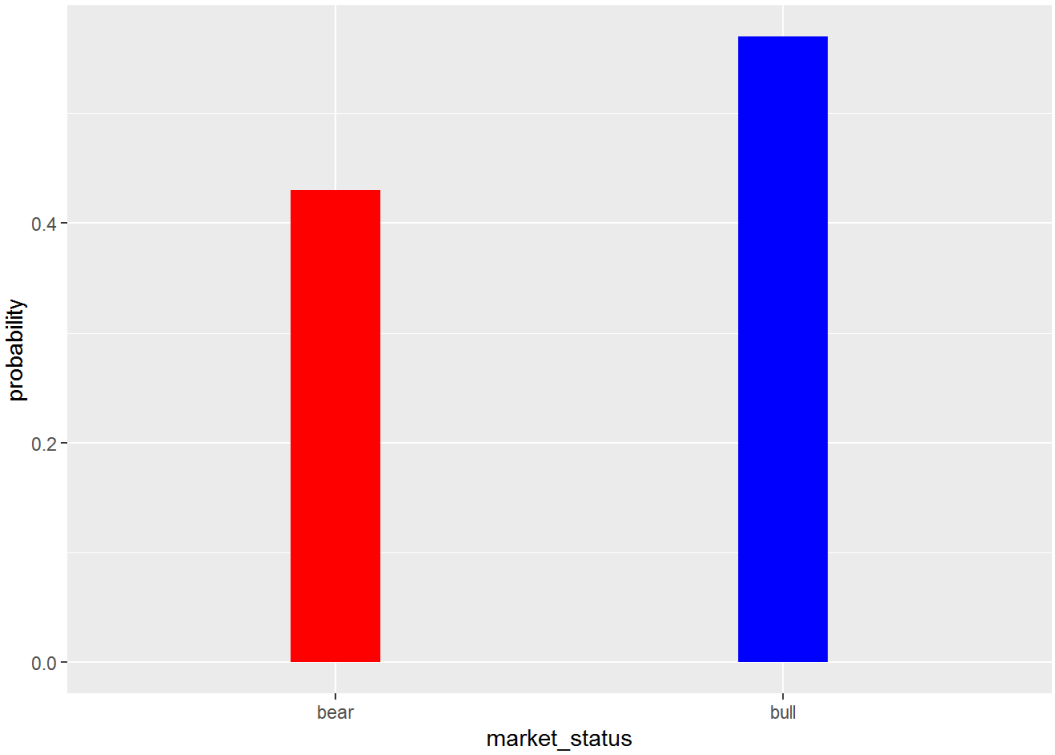
For measuring the performance, not only index but also we look at the performance as categorical outcome. Meaning, in each time step, if the outcome of of bearish stratgy be higher, we denote the **categorical.outcome** by 1 and if the bullish create a better performance the categorical.outcome is 0.

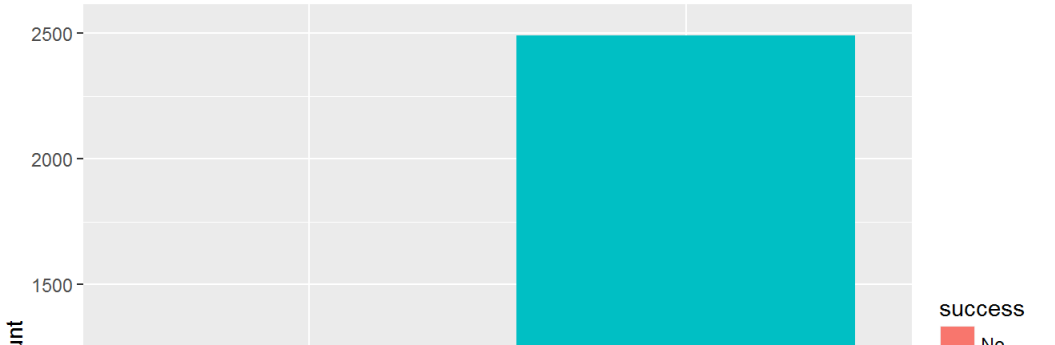
```
##   last.observed.vol.diff     Trig categorical.outcome
## 1               3.642679 2.057156                   0
## 2               3.657427 2.057156                   0
## 3               2.575423 2.050185                   0
## 4               1.734472 2.012524                   1
## 5               2.190382 1.954902                   0
```

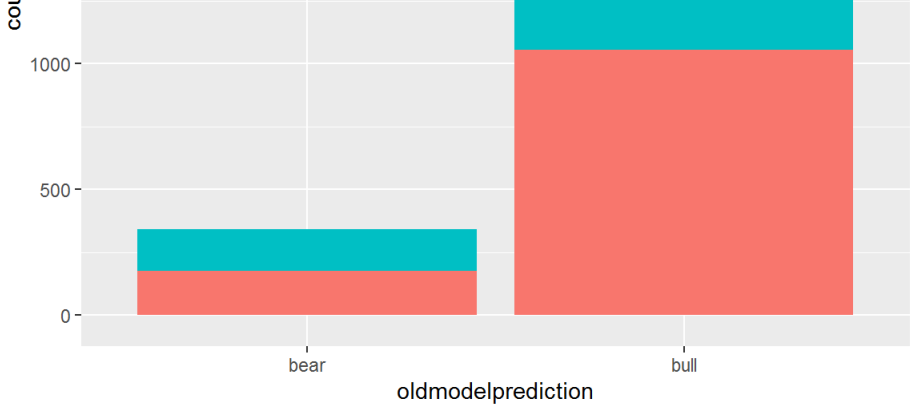
Let's look at the probablity of these two class

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.2.5
```







With higher chance of having bull market, we can use the most likely outcome event as the default prediction i.e. always predicting the categorical.outcome to be 0, this way we can achieve 57% performance. Suprisingly we see that we achivee an overall performance 0.5657244 in the current model which is even worse than the default prediction. So normaly by such prediction we should not end up in index higher index, however this is not the case.

Let's see the rate of sucess in the current model more deeply

```
##        trueresult
## predict bear bull  Sum
##    bear  164  176  340
##    bull 1053 1437 2490
##    Sum  1217 1613 2830
```

looking into the performance of the two category independently. We will see that these two classes of *bull* and *bear* have different performance. The current model has the following performance in successfully predicting the state of market for the two group.

```
##         sucess   failure     total
## bear 0.4823529 0.5176471 0.1201413
## bull 0.5771084 0.4228916 0.8798587
```

Let's put the categorical outcome aside for a moment and look at the index performance in these two class. First, let's expend the data as follow.

```
##   good.outcome  bad.outcome categorical.outcome
## 1  0.008460251 -0.010444354                   0
## 2  0.006890889 -0.022562891                   0
## 3  0.003178046 -0.028301247                   0
## 4  0.007240238 -0.002765695                   1
## 5  0.011043038 -0.043462306                   0
## 6  0.005832958 -0.001088917                   1
```

Here **bad.outcome** means the performance result of not predicting the correct **categorical.outcome** and **good.outcome** is the performance result of correct prediction. As we can see the performance in bearish is more dramatic then the bull situation. We have the following mean performances

```
##   prediction     correct        wrong
## 1       bear 0.026416222 -0.022047403
## 2       bull 0.007072108 -0.008708269
```

It's now clear that the reason for having a such a good index in the current model is that when we predict bear, we are correct about 50% of the times, with higher rate of return in its *good.outcome* compare to *bad.outcome*. It's no suprise that we do overall better than majority voting.

Let's denote by X the performance of the outcome. Assume that our prediction model on average predict bear with $P(A) = p$ and hence bull with probablity $P(A^c) = 1 - p$. Hence the expected performance will be

$$E(X) = E(X|\text{predicting bull})P(A) + E(X|\text{predicting bear})P(A^c)$$

If in each category the rate of sucess be equal to p_{bull} and p_{bear} then

$$E(X|\text{predicting bear}) = p_{bear}m_{beargain} + (1 - p_{bear})m_{bearLost}$$

$$E(X|\text{predicting bull}) = p_{bull}m_{bullgain} + (1 - p_{bull})m_{bullLost}$$

With estimates of m values from the matrix, and assuming that the p_{bull} always to be almost 57% we conclude that for maximizing the outcome we need to maximize p_{bear}. With an easy calculation we can see that any model that can predict the bearish market with a rate of sucess better than 0.4630634 will beat the SPX. With this in mind, we now propose a new model which can improve the rate of sucess on bearish market.

In other words, for an interval of size N which we finely split it to two subset of size N_{bear} and N_{bull} for bull and bear prediction, and number of success

In other words, for an interval of size N which we finaly split it to two subset of size N_{bull} and N_{bear} for bull and bear predection, and number of success in each subset be denoted by N^s_{bear} and N^s_{bull} and number of failure is denoted by N^f_{bear} and N^f_{bull}. The expected performance at each step

$$\frac{1}{N}\left((N^s_{bull}.0.02641622) - ((N^f_{bull}.0.0220474) + (N'_s.*0.007072108) - (N'_s.0.008708269)\right)$$

The following function compute the expected performance in each time step.

```
Eperf <- function(performanceonBear , performanceonBull , Bearpredictingratio){
  bearPerf<-(performanceonBear *  0.02641622 ) -  ( (1- performanceonBear )*0.0220474)
  bullPerf<-(performanceonBull *  0.007072108 ) -( (1-performancecnBull )*0.008708269)

  totalperf<- (bearPerf* Bearpredictingratio) + (bullPerf*(1-Bearpredictingratio))

  return(totalperf)
}
```

Using the previous data the expected performance of the current model can be obtained as follow

But also we have

$$N_{bull}\left((p_{bull}.0.02641622) - ((1-p_{bull}).0.0220474)\right) + N_{bear}\left((p_{bear}.*0.007072108) - ((1-p_{bear}).0.008708269)\right)$$

Normally increasing tries in N_{bull} could decrease the efficency on predicting it meaning decease p_{bull}. Similary increase in N_{bear} decrease p_{bear}.

So between two prediction model with similar level of performance in predicting bear. The one with higher rate of bear prediction compare is desireable. One can also bring the risk effect to the proble, having more bear with higher rate of return and loss make a startgy more risky.

```
performanceonBear = 0.482659
performanceonBull = 0.5768302
Bearpredictingratio = 0.1221751

Eperf( performanceonBear , performanceonBull ,Bearpredictingratio )
```

```
## [1] 0.0005103555
```

Since this value is only based on our classification performance it will be a better measurement than index. Clearly the final index will be $exp(N*r)$ times its inital value.

The function Eperf later should be used as kernel for any classification we want to use. In fact the matrix loss in tree classification has this property.

The new model(short signal)

The problem in hand is a forcasting problem, given the previous **vol.diff**s we need to forcast the **categorical.outcome**. There is a rich classification libraries in R that we like to use for this problem. At the same time, we need to consider time dimention as well. The data points are not i.i.d. . This create a great limitation, so instead we look to expand the feature space in a way that time ordering problem be avoided.

For example if we define

$$Y_t = (X_{t-2}, X_{t-3}, X_{t-4} \ldots, X_{t-n})$$

Then try to train a model from the set

$$\{Y_{100}, Y_{101}, \ldots, Y_{200}\}$$

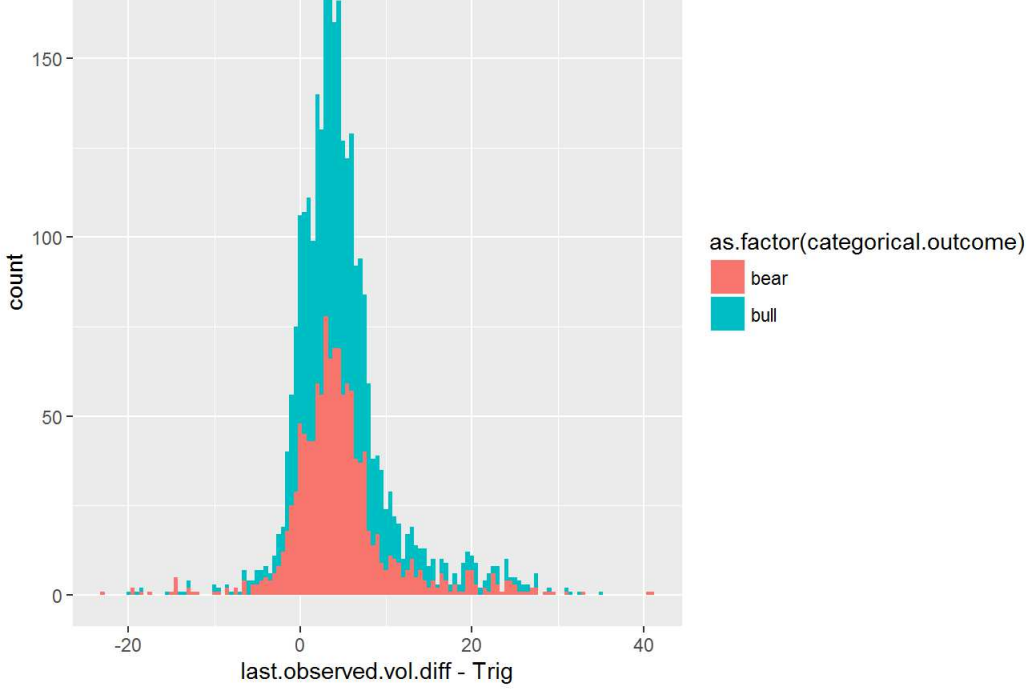
Then for example in fitting the point Y_{150} from other points, a lot of weight will be putted on the points $Y_{152}, Y_{153}, \ldots$ because they have X_{150} as one of their component and that obviously effect the categorical.outcome enormously.

X_t is the observed value at the end of the day t that we are trying to predict its performance.

The feature space(feature engineering)

The current model claims that by looking at the vol diff and trigger value, we can identify the type of the outcome. Let's look at the distribution of bear/bull relative to this value.

```
qplot(data = sdata , last.observed.vol.diff - Trig , geom= 'histogram' , fill= as.factor(categorical.outcome) , binwidth=0.5 )
```



Now we like to look at a different set of features and examine the possiblity of learning about the categorical.outcome through their structure. For each data point, we consider the following features.

For each point in time, we look at past 9 available observation of voll difference and find the regression line between these points. Then we use the slope, intercept and the variance of error of the regression line as our new 3 features.

- slope
- intercept
- sig

Moreover, we look at the difference of the *last.observed.vol.diff* and the fitted value from regression line. Also we look at this difference divided by sig(Since we will use trees and the only classifaction hyperplan provided by classification trees are step function we need to do these calculation and add them to data manually)

- err
- rate
- last.observed.vol.diff

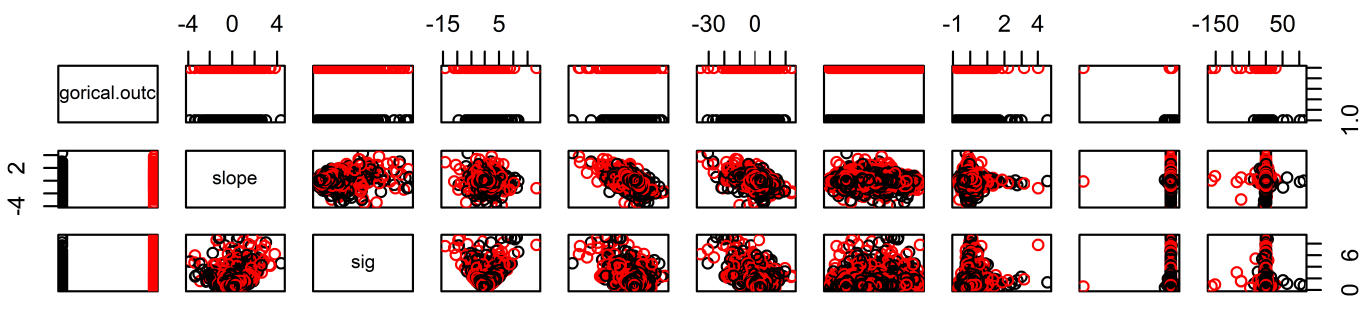
And finally we take the difference of the slope, intercept and sig from t and t-1 (difference divided by previous value)

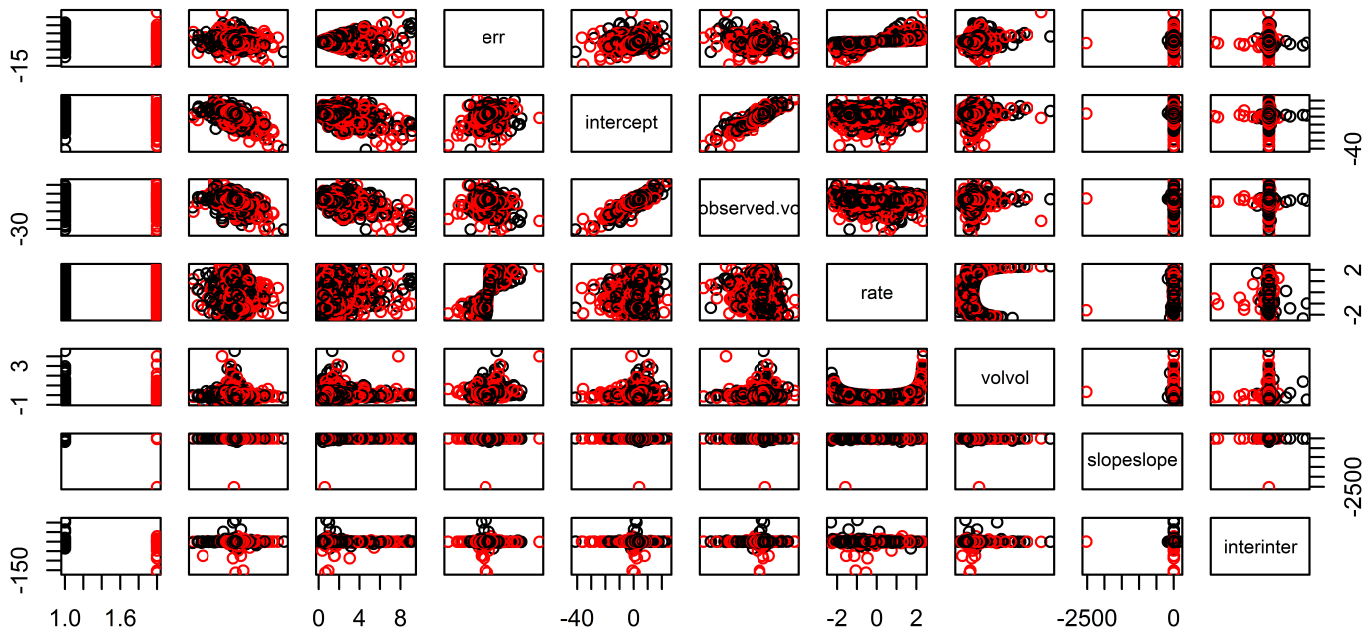
- volvol
- interinter
- slopeslope

At the end we have a feature space as follow

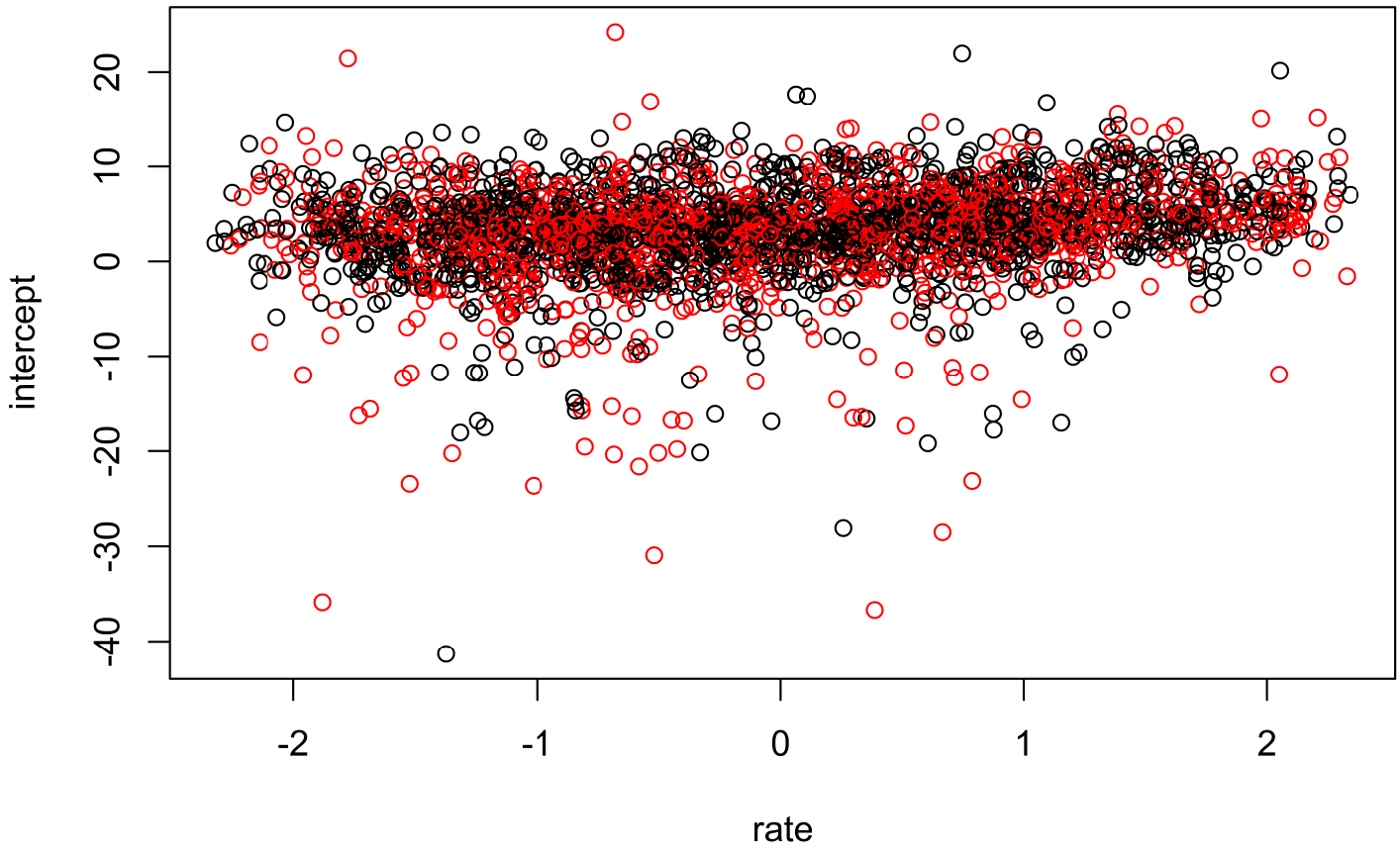
categorical.outcome	slope	sig	err	intercept	last.observed.vol.diff	rate	volvol	slopeslope	interinter
2	0	-0.0425793	0.5975576	0.4622022	4.1622083	3.657427	0.7734856	0.0544481	-0.5908136	-0.0625889
3	0	0.1212791	0.6277794	0.8324724	3.2866165	2.575423	1.3260587	0.0505756	-3.8483110	-0.2103671
4	1	0.3234256	0.4815847	0.7026587	2.1137047	1.734472	1.4590552	-0.2328759	1.6667875	-0.3568752
5	0	0.3895456	0.3339706	-0.2376623	1.5631745	2.190382	-0.7116263	-0.3065174	0.2044366	-0.2604574
6	1	0.3671014	0.3834600	-0.5793189	1.4157170	2.362137	-1.5107676	0.1481849	-0.0576164	-0.0943321
7	0	0.3647502	0.3806277	0.2651653	0.9925126	1.092097	0.6966528	-0.0073862	-0.0064046	-0.2989329

The goal is to use tree to identifty the possible trend in these variables that could help use idenifiningy the "color" of these points correctly. Here's the a picture of possible relation between these variables





As an example looking deeper to relationship between rate and intercept we see a higher percentange in red color points in the lower left corner



The idea is that we assume although we might not gain any information about the points where we have a high density, but probably there could be information embedd in the points with lower density around them.

Tuning parameters and measures

Let's pick our tuning parameters:

- **LTeststep** shows how often we renew our model
- **skip** shows how many interval we drop before start the model, if we pick a small value for this, the resulting decision tree which are based on a few

- **skip** shows how many interval we drop before start the model, if we pick a small value for this, the resulting decision tree which are based on a few number of element will be too bushy.
- **NTeststep** is the final time period
- **gap** distance between the largest point in training set and test set.

(NTeststep-skip)*LTeststep will be the number of predictions

```
#1-tuning parameters Lstep*(Nstep+1)<2800
LTeststep=20
NTeststep=140
gap=5
skip<-10
```

The following are the parameters we use for measuring the final performance of stratgy on different level

- **trade** count the number of time we swich from bearish to bullish and vice versa
- **index** and **trajectory** track the resulting index of our startgy at each point in time
- **correctpredecition** and **predictationratio** track the number of times we make a correct predictions

At the end since the performance on indiviual class are important too we track the success and faliure on each group indiviually

- **cor1** is the number of correct prediction for 1 (i.e. bear) and **total1** is the total number of predicting 1
- **cor0** is the number of correct prediction for 0 (i.e. bear) and **total0** is the total number of predicting 0
- **cor0list** is a list of **cor0** after each **LTeststep** interval. Similarly we have the other 3 lists to keep track of the other partial results.
- **fittingratio** is the list of rate of fitting the model on the training data, similary we have **fittingratio0** and **fittingratio0** for each individual group of of bull and bear.

```
#1-measures
trade=0

#index
index=100.0
trajectory=rep(0,2000)

#total prediction
correctpredecition=0
predictationratio=rep(0,NTeststep)
fittingratio=rep(0,NTeststep)
fittingratio0=rep(0,NTeststep)
fittingratio1=rep(0,NTeststep)

#each class prediction
cor0=0
cor1=0
total1=0
total0=0


cor0list=rep(0,NTeststep)
cor1list=rep(0,NTeststep)
total1list=rep(0,NTeststep)
total0list=rep(0,NTeststep)

observedones=rep(0,NTeststep)
```

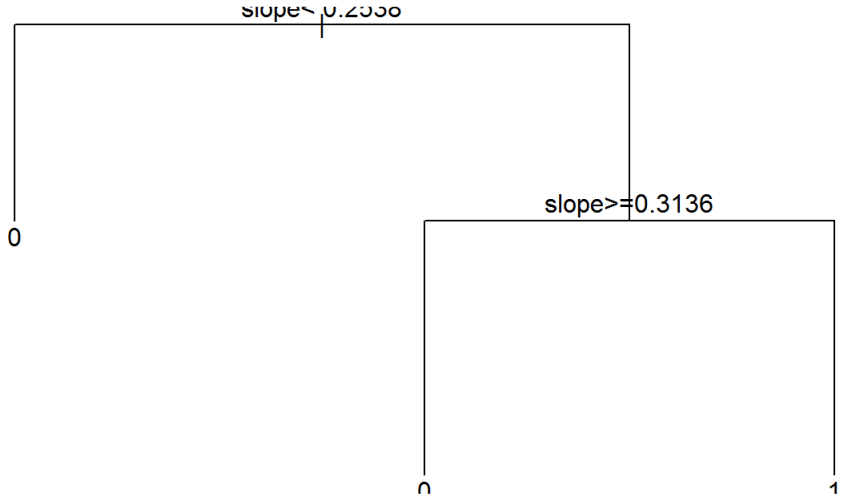
In each step we look at the

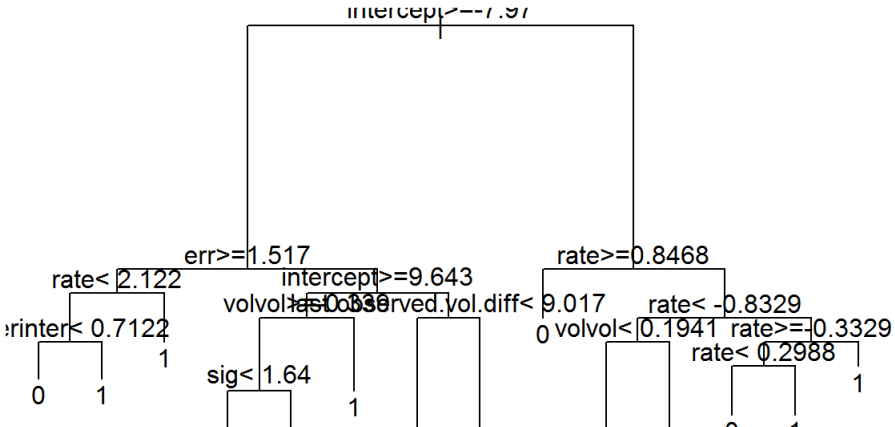
```
require(rpart)
## Loading required package: rpart
## Warning: package 'rpart' was built under R version 3.2.5
for (i in c(skip:NTeststep)){
  N=i*LTeststep
  #3-sample of the first N observation i.e. train data
  traindata<-subdata[c(1:N),]

  maxdep=2
  if (N>1500) {maxdep=3}
  if (N>1900) {maxdep=5}
```

```
if (N>2300) {maxdep=7}

subdata.rpart= rpart(categorical.outcome ~ .   ,data=traindata , control = rpart.control(maxdepth = maxdep, cp = 0.0001))

#4-predict on the next Lstep observation i.e. test data
begin<-N+gap+1
end<-N+LTeststep+gap
testdata<-subdata[c(begin:end),]
testoutcome<-rdata[c(begin:end),]

oldpredoutcome<-data[c(begin:end),'oldmodelprediction']

rpart.pred=predict(subdata.rpart, testdata ,type="class" )


#plot(subdata.rpart);text(subdata.rpart,pretty=0  )
if (i==75){
  tree75av=TRUE
  tree75=subdata.rpart
  }
if (i==100){
  tree100av=TRUE
  tree100=subdata.rpart
  }

explained<-as.numeric(predict(subdata.rpart, traindata ,type="class" ))-as.numeric(traindata$categorical.outcome)
explained<- 1- (sum(explained^2)/length(explained))
fittingratio[i]=explained


train1<-traindata[traindata$categorical.outcome==1,]
train0<-traindata[traindata$categorical.outcome==0,]

explained0 <- as.numeric(predict(subdata.rpart, train0 ,type="class" ))-as.numeric(train0$categorical.outcome)
explained1 <- as.numeric(predict(subdata.rpart, train1 ,type="class" ))-as.numeric(train1$categorical.outcome)

explained0<- 1- (sum(explained0 ^2)/length(explained0))
explained1<- 1- (sum(explained1 ^2)/length(explained1))

fittingratio0[i]=explained0
fittingratio1[i]=explained1

#5-performance in Lstep
for (j in c(1:LTeststep)) {
  if (rpart.pred[j]==testdata$categorical.outcome[j] ) {
    index=index*(1+testoutcome$good.outcome[j])
    correctpredecition=correctpredecition+1
    predictationratio[i]=predictationratio[i]+1

  }
  else{
    index=index*(1+testoutcome$bad.outcome[j])
  }
  trajectory[N+j]=index


  if (rpart.pred[j]==1){
    total1=total1+1
    if (rpart.pred[j]==testdata$categorical.outcome[j] ) {
      cor1=cor1+1
      }
    }

  else{
    total0=total0+1
```

```
      if (rpart.pred[j]==testdata$categorical.outcome[j] ) {
        cor0=cor0+1
        }
      }

  }


  observedones[i] <- sum(as.numeric(testdata$categorical.outcome) - 1)


  cor0list[i]=cor0
  cor1list[i]=cor1

  total0list[i]=total0
  total1list[i]=total1


  #6-number of trades in Lstep
  for (j in c(2:LTeststep) )  if (rpart.pred[j]!= rpart.pred[j-1]) {trade=trade+1}

}
```

Before peresneting the perferormance following this model let's peresent two example of the decision trees we used. The smaller tree was obtained after having 1500 training point, and the deeper one after having 2000 training point



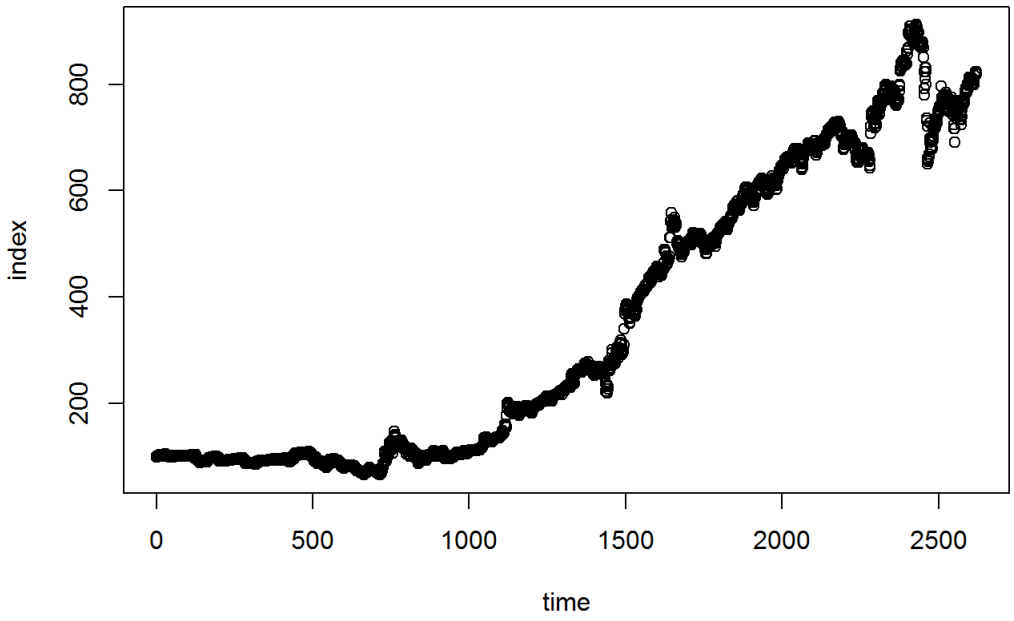


We have manually allowed increase in the depth of trees as the number of training points increased. By consistantaly having a small number for depth, as time increase the level of fitting the data decrease. So really one can also put a critirion for depth of tree such as a level of fitting achived on the traning set. Looking at the fitting curves explain this idea. Another approach could be to use pruning methods for all trees. ##perfromance

Let's look at the performance from this prediction model

Index

With number of trade=232 we reach to final index=819.4241805. The following is the plot of index



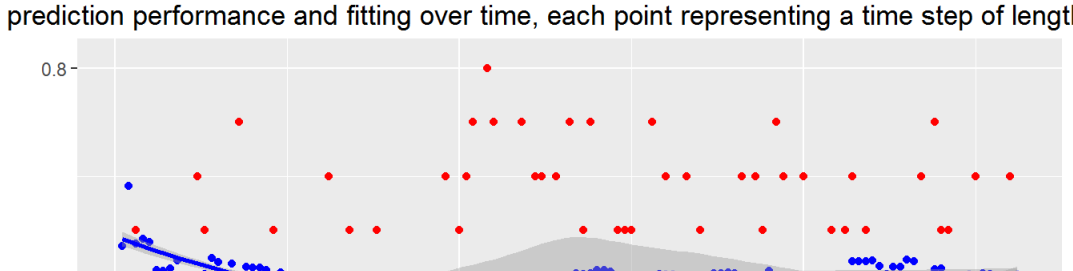
Success

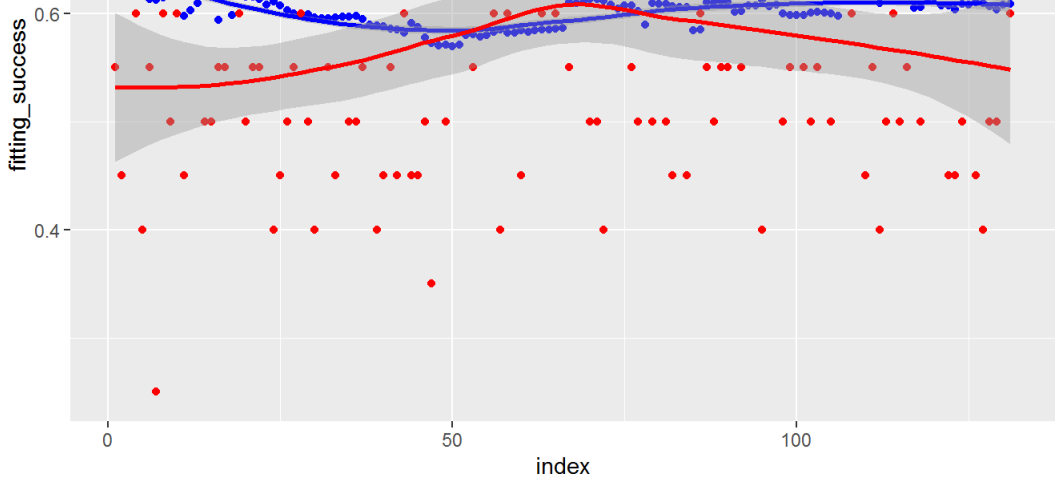
In terms of overall fitting the training set and performance on the test set, the total performance has the rate of success 0.5683206 But more importantly the rate of sucess on predicting bear is $\frac{cor1}{total1} = 0.5054545$ The rate of sucess on bull is $\frac{cor0}{total0} = 0.575693$ and the bear prediction ratio is $\frac{total1}{total1+total0} = 0.1049618$ and hence from the Eperf function we have

```
performanceonBear = cor1/total1
performanceonBull = cor0/total0
Bearpredictingratio = total1/(total1+total0)

Eperf( performanceonBear , performanceonBull ,Bearpredictingratio )
```

```
## [1] 0.0005939032
```

The predecition performance on each time interval, and the level of fitting the training set are displayed in the following plots



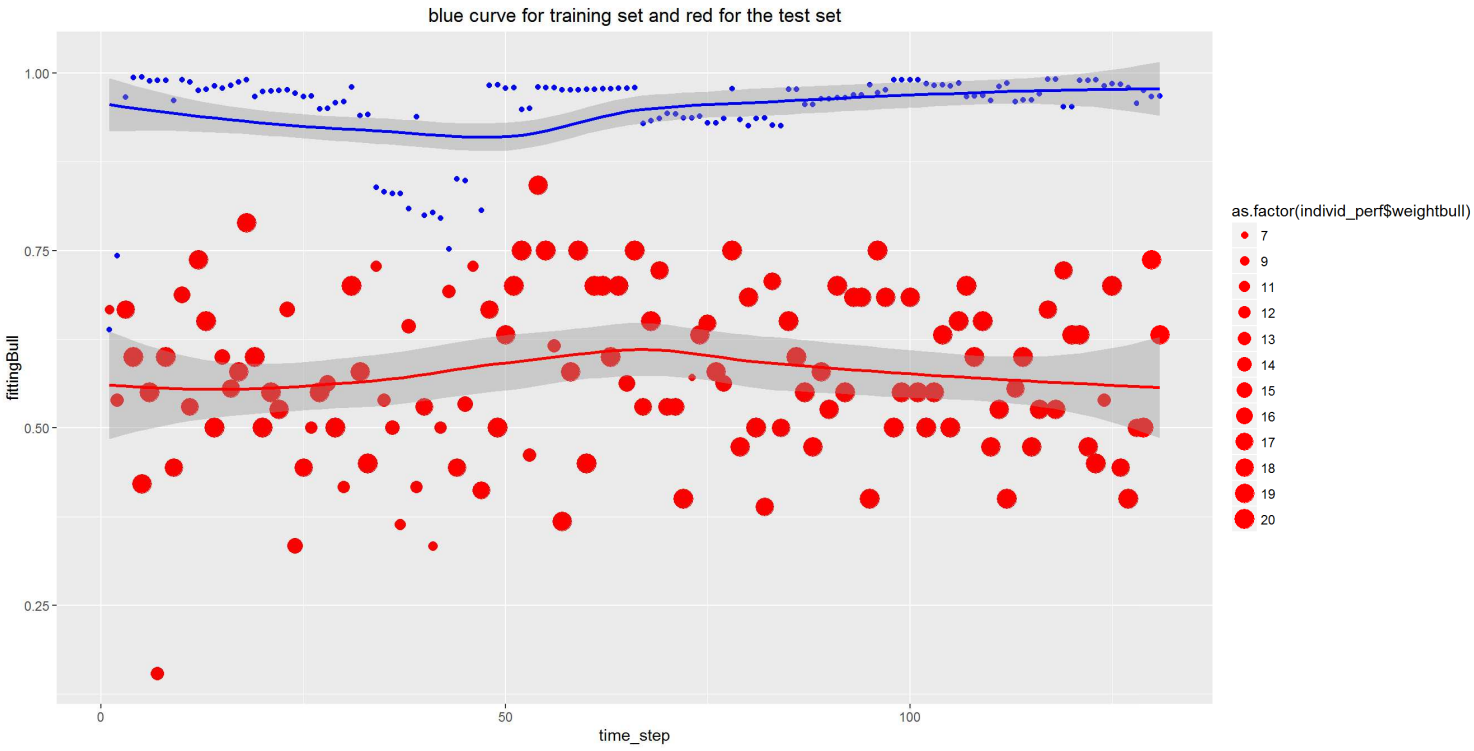


Once again looking at the overall performance in terms of sucess in prediction and fitting the model on the training set is not helpful. In the next two graph we present the individual performances of fitting and prediction on bear and bull seprataely.

In the first graph we have the performance restricted on bull market. The blue points (blue curve is the trend of blue points) are the level we fitted the bull data at each time step, and the red points(red curve is the trend of red points) is the level of performance on predicting bull market.

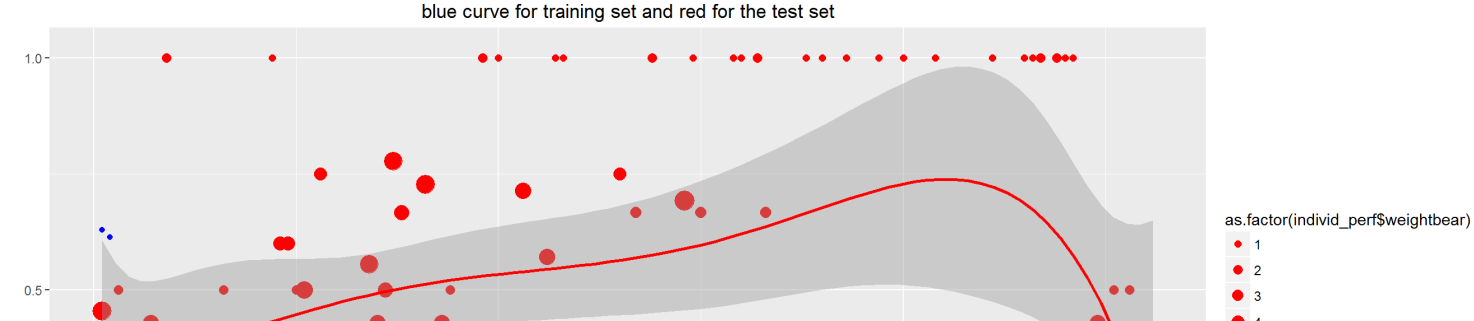
We see that because our prediction model tends to put more effort in fitting the bull points(because of being the majority compare to bear), the blue curve is almost at perfect level. (We tend to overfit on bulls) As a results of this overfitting the model doesnt do as well for the test set.

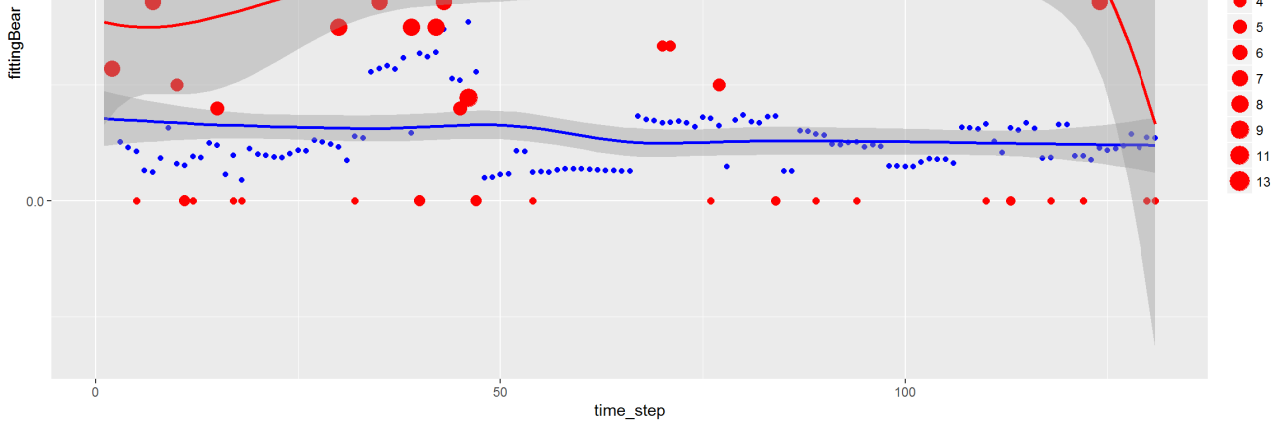
```
## Warning: Using size for a discrete variable is not advised.
```



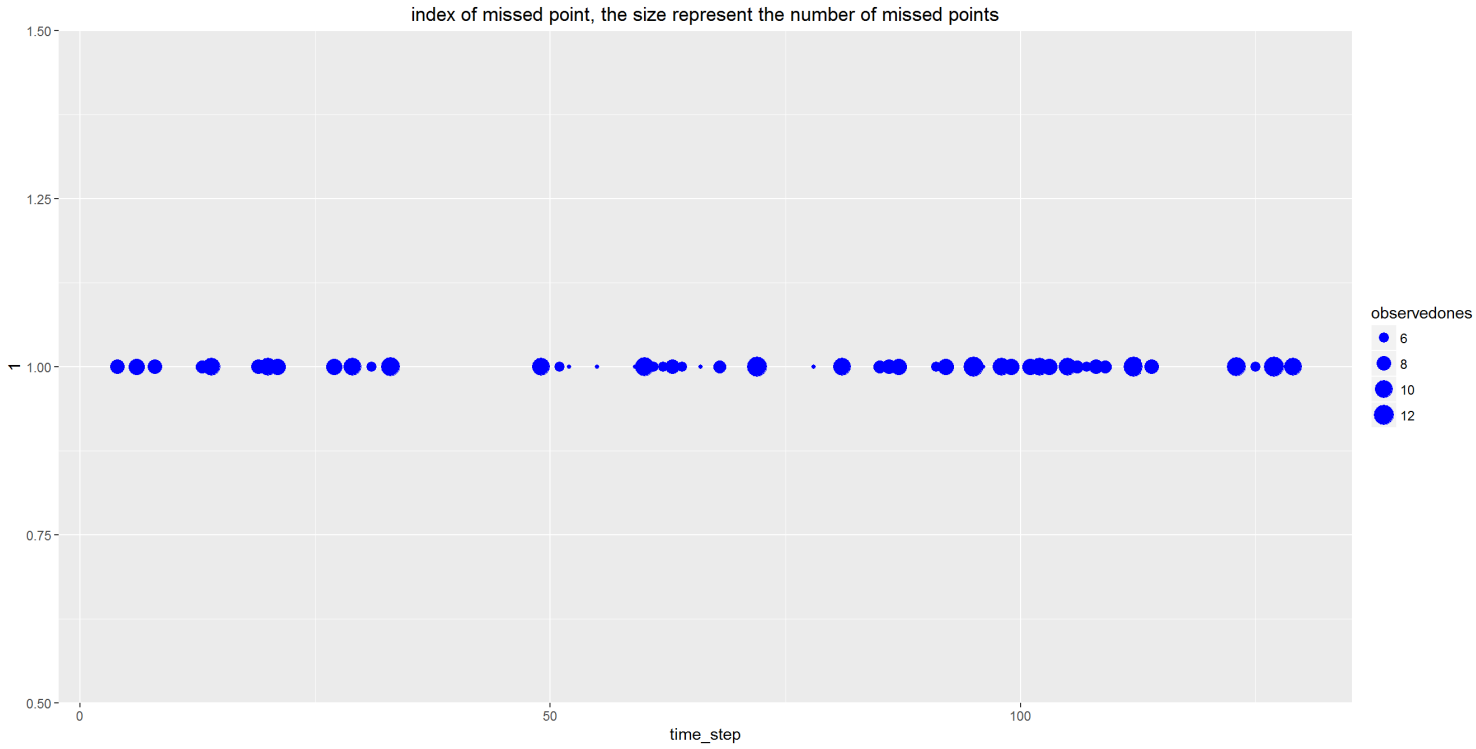
On the other hand, in predicting the Bear market, since they are in minority compare to bulls, the models tends to make less effort in fitting these points, so the blue curve is very close to zero. on the other hand the performance on the test set is higher, but with a much more wider confidence interval.(Here we miss a lots of bear points but when we predict the model tends to be very accurte on the points that it decides the to be bear.)

```
## Warning: Using size for a discrete variable is not advised.
```





moreover we see that we have a consistent level of fitting till time step 80, but after that since our model is not reach enough its going to do poorly on the test set. We also see that there are couple of point removed, that's because the number of predicted bear in these time steps are equal to zero so resulting 0/0 which is undefined. The increase in the number of interval without bear prediction at the end of plot is another indicator that we can fit more aggressively on these points, or we can shrink the training set by excluding the early data. (Or we can just add a weight to our prediction model and lower the weight for the data in the early time steps.)

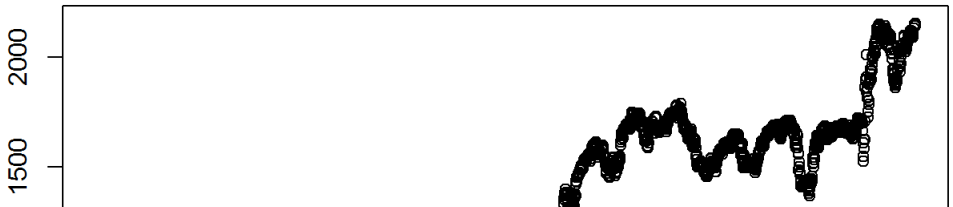


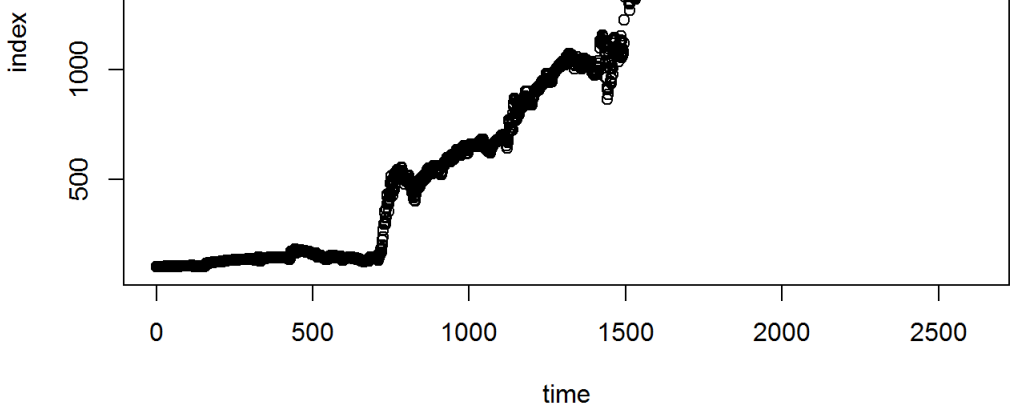
To play around with the dynamic of the the past two curves and also minimize the missing points, we have couple options for forther improvements:

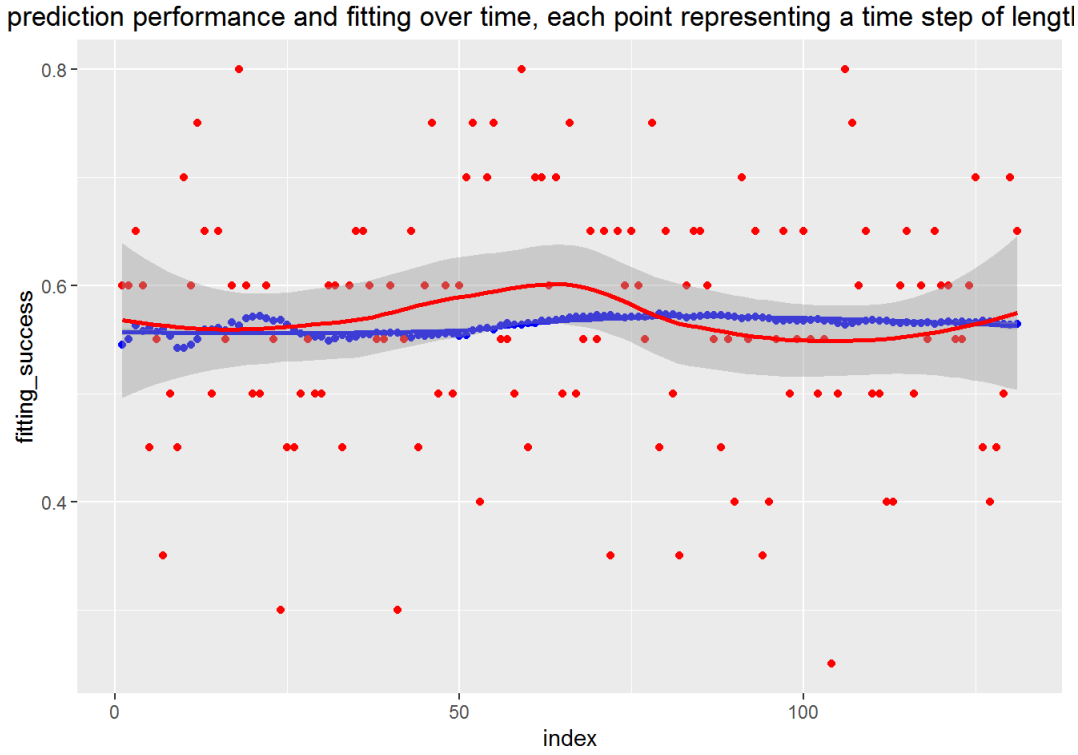
- we can ask our prediction model to put more weight on the latest points
- we can dynamically change the control features of the decesion tree(increase the depth as we get more sample)
- we can split the training set by time, and use multiple decision tree and ask each about the prediction and then take a weighted average of their results
- use for example 5 set in cross-validation smart. choose time points mod 5.
- adaboost with initial weights as general strategy, but we alwasys need to remind the importance?! or just the first step is enough

current model performance

Now that we have a better picture of why we like to keep track of some measurements, let's look at these measurement on the current model. For the total performance in terms of fitting and resulting rate on the test set we have :

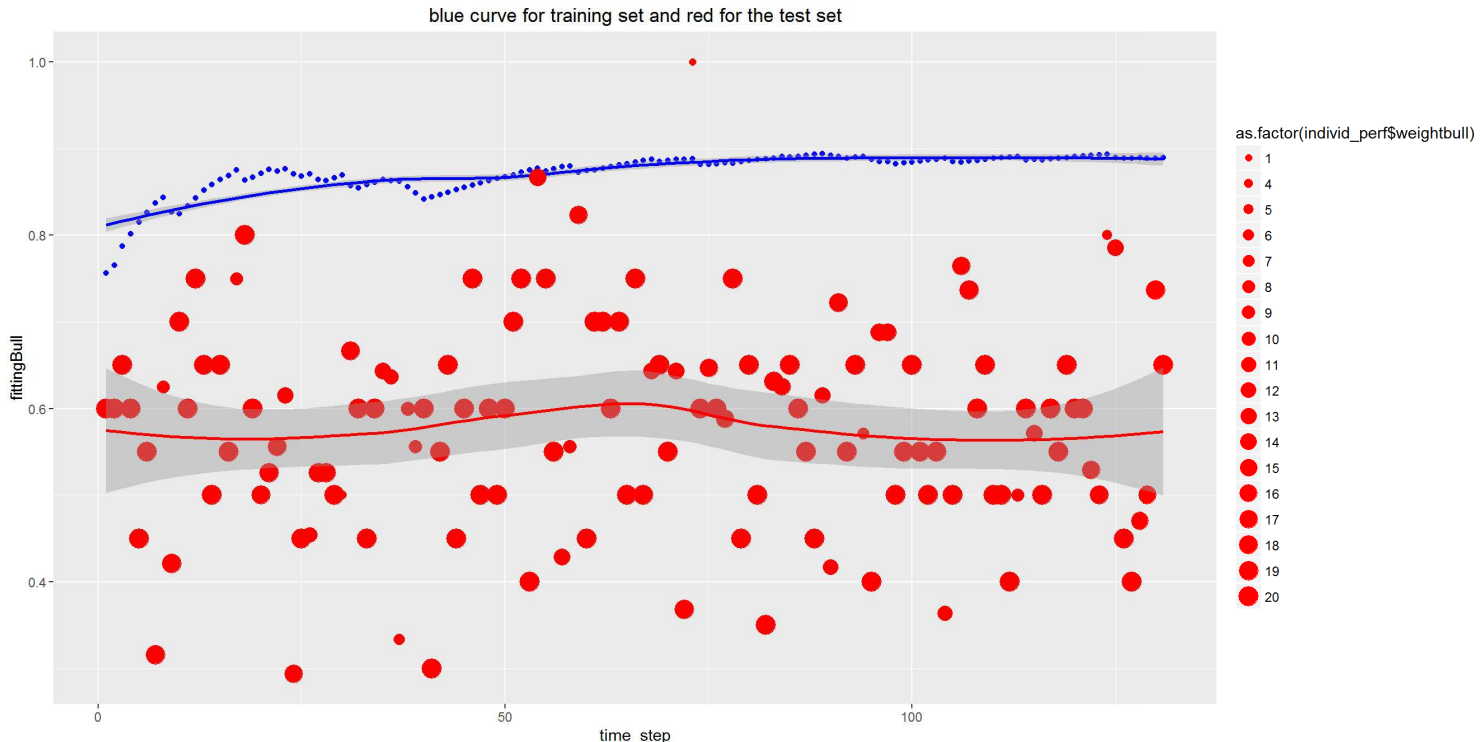






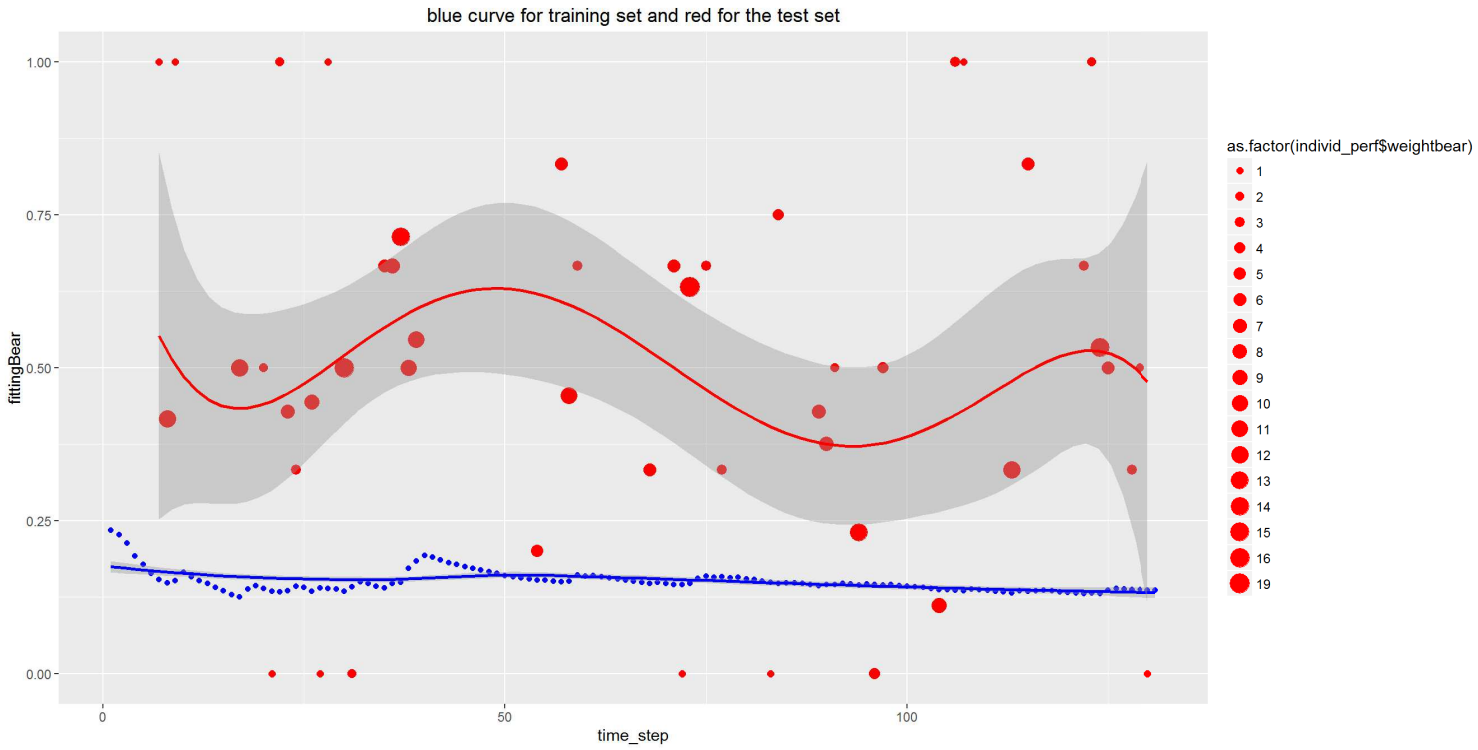
In fitting bull events we have:

```
## Warning: Using size for a discrete variable is not advised.
```

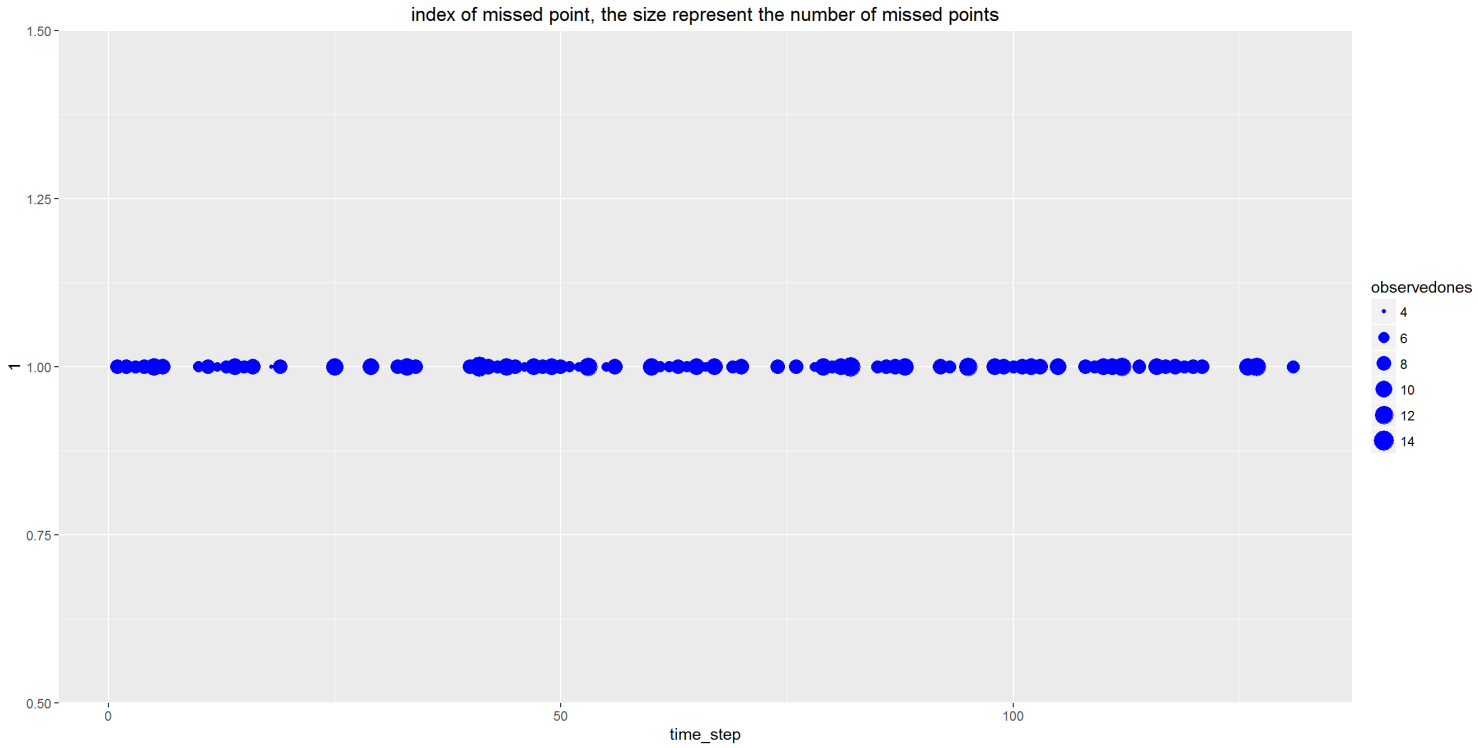


In fitting bear events we have:

```
## Warning: Using size for a discrete variable is not advised.
```



The interval that



mixed model

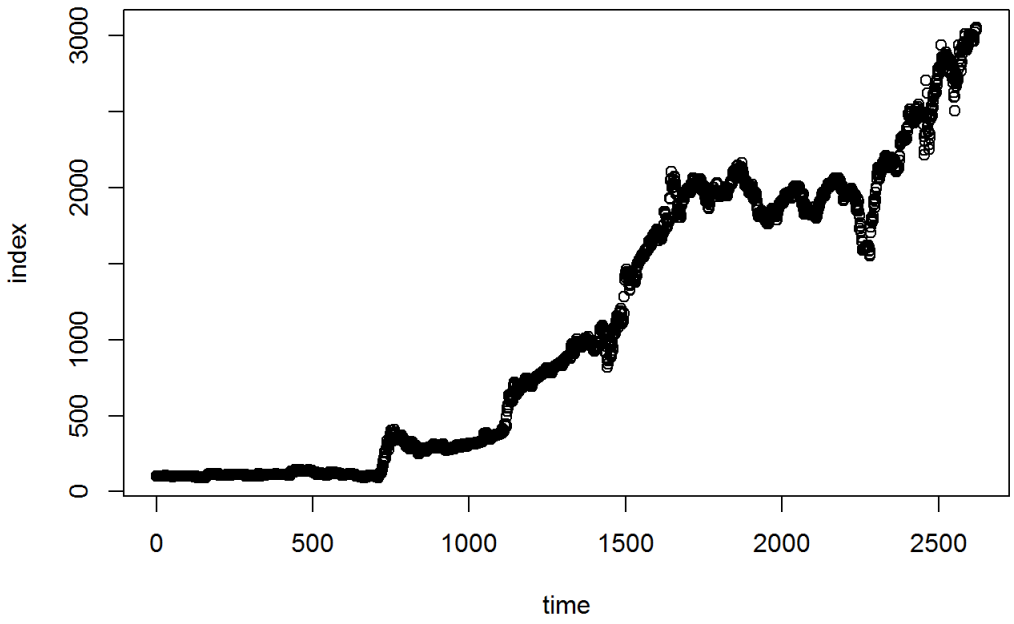
We mixed the prediction outcome from the two model, since predicting *bear* correctly improve the performance much more, we define the mixed model, to take the outcome of the two model, and assing bear even if one of the two model predict it. We skip the detail of the model and jum to its performance.

performance

Index

With number of trade=309 we reach to final index=3036.5866621. The following is the plot of index

With number of trade 509 we reach to finalIndex 3050.5000211. The following is the plot of index



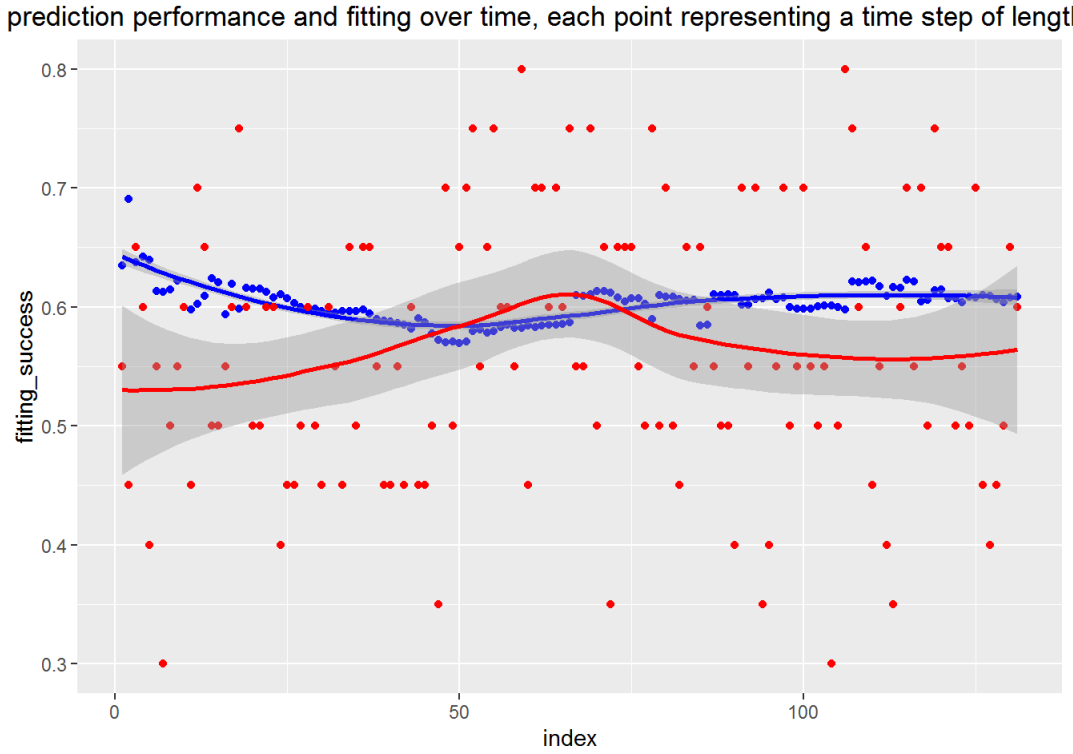
Success

In terms of overall fitting the training set and performance on the test set, the total performance has the rate of success 0.5637405 But more importantly the rate of sucess on predicting bear is $\frac{cor1}{total1} = 0.4906832$ The rate of sucess on bull is $\frac{cor0}{total0} = 0.5802527$ and the bear prediction ratio is $\frac{total1}{total1+total0} = 0.1843511$ and hence from the Eperf function we have

```
performanceonBear = cor1/total1
performanceonBull = cor0/total0
Bearpredictingratio = total1/(total1+total0)

Eperf( performanceonBear , performanceonBull ,Bearpredictingratio )
```

```
## [1] 0.0006851452
```

The predecition performance on each time interval, and the level of fitting the training set are displayed in the following plots



Once again looking at the overall performance in terms of sucess in prediction and fitting the model on the training set is not helpful. In the next two graph we present the individual performances of fitting and prediction on bear and bull seprataely.

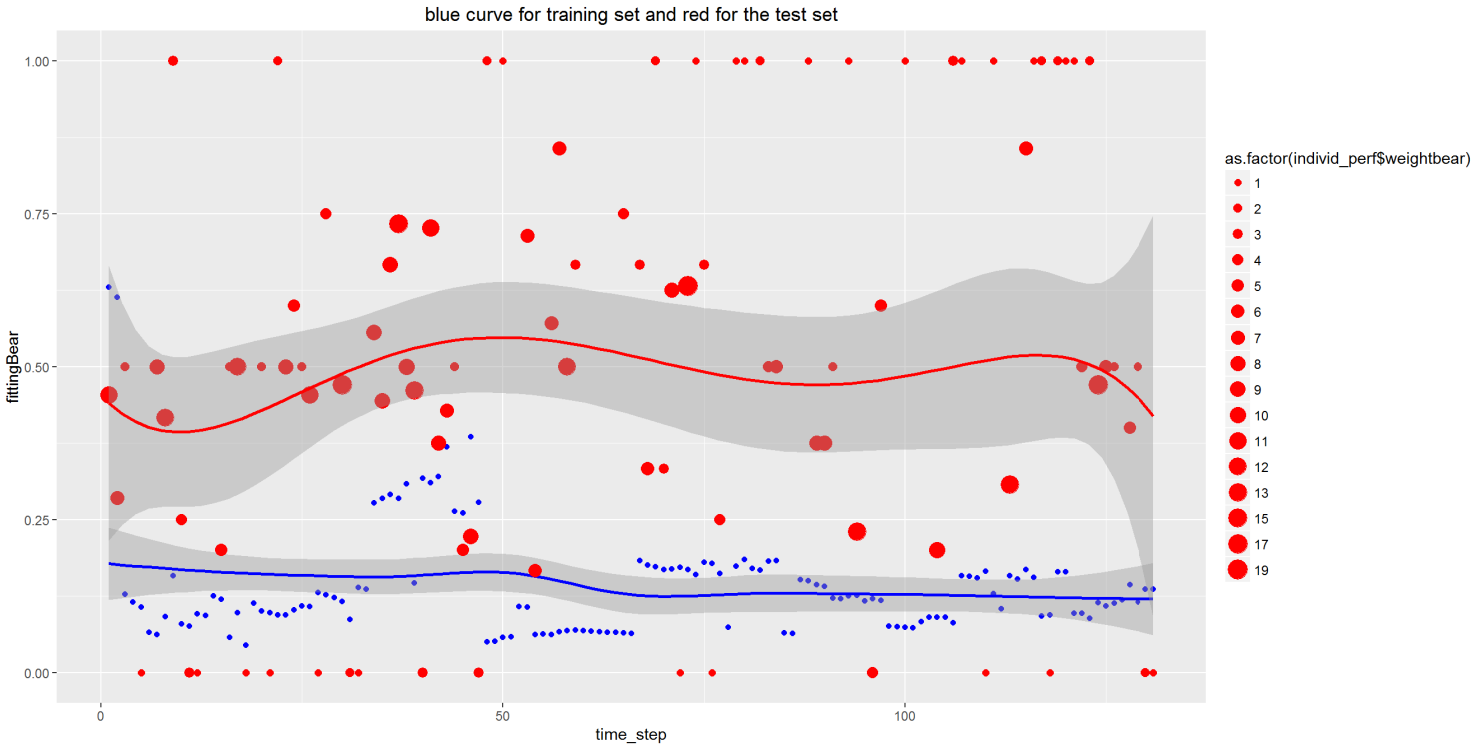
In the first graph we have the performance restricted on bull market. The blue points (blue curve is the trend of blue points) are the level we fitted the bull data at each time step, and the red points(red curve is the trend of red points) is the level of performance on predicting bull market.

We see that because our prediction model tends to put more effort in fitting the bull points(because of being the majority compare to bear), the blue curve is almost at perfect level. (We tend to overfit on bulls) As a results of this overfitting the model doesnt do as well for the test set.

```
## Warning: Using size for a discrete variable is not advised.
```

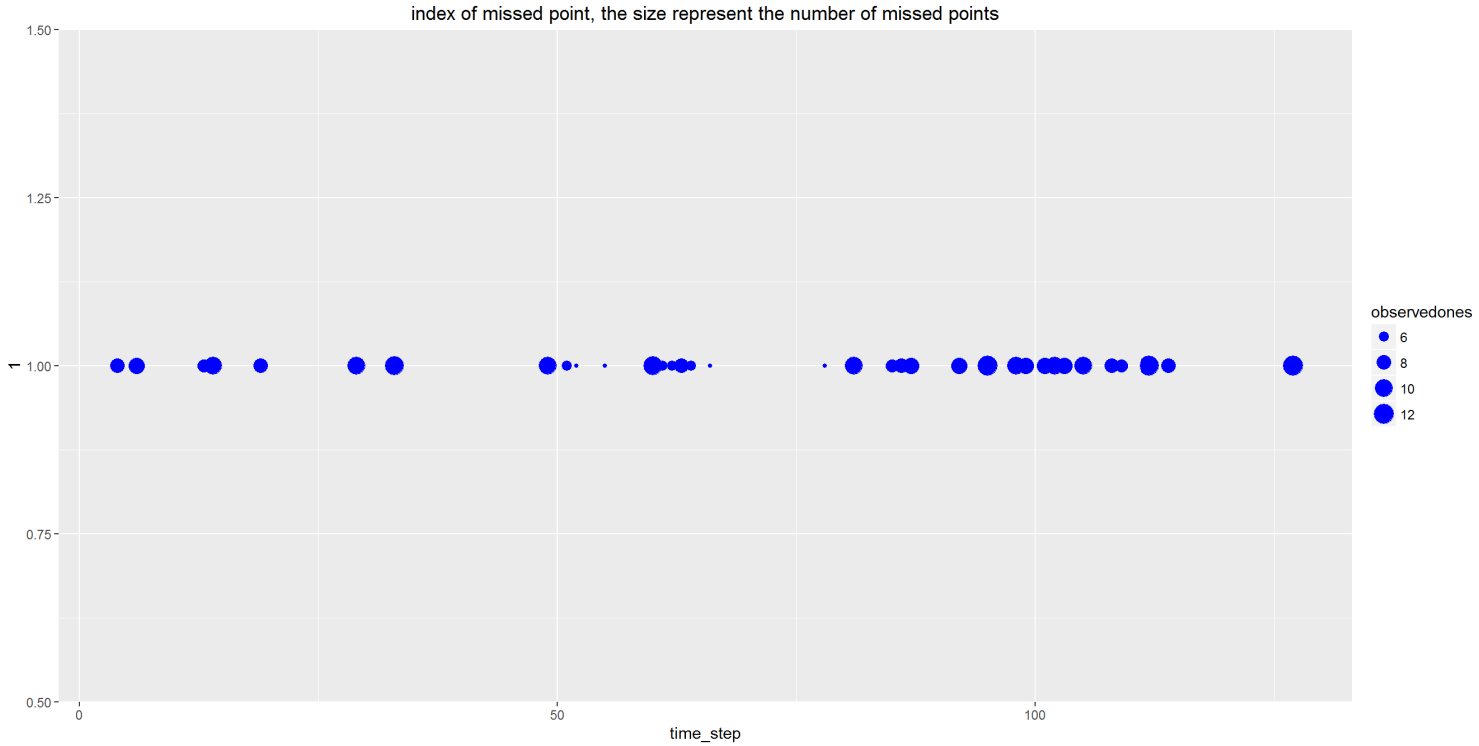
On the other hand, in predicting the Bear market, since they are in minority compare to bulls, the models tends to make less effort in fitting these points, so the blue curve is very close to zero. on the other hand the performance on the test set is higher, but with a much more wider confidence interval.(Here we miss a lots of bear points but when we predict the model tends to be very accurte on the points that it decides the to be bear.)

```
## Warning: Using size for a discrete variable is not advised.
```



moreover we see that we have a consistent level of fitting till time step 80, but after that since our model is not reach enough its going to do poorly on the test set. We also see that there are couple of point removed, that's because the number of predicted bear in these time steps are equal to zero so

test set. We also see that there are couple of point removed, that's because the number of predicted bear in these time steps are equal to zero so resulting 0/0 which is undefined. The increase in the number of interval without bear prediction at the end of plot is another indicator that we can fit more aggressively on these points, or we can shrink the training set by excluding the early data. (Or we can just add a weight to our prediction model and lower the weight for the data in the early time steps.)



The following table compare all three models:

	performanceonBull	performanceonBear	Bearpredictingratio	performance daily index	final index
current model	0.4826590	0.5768302	0.1221751	0.0005104	2141.0800
new model	0.5054545	0.5756930	0.1049618	0.0005939	819.4242
mixed model	0.4906832	0.5802527	0.1843511	0.0006851	3036.5867