Knnmodel

Dr.metales

12/16/2019

Introduction

In this paper we will explore the **k nearest neighbors** model using two data sets, the first is **Tiatanic** data to which we will fit this model for classification, and the second data is **BostonHousing** data (from **mlbench** package) that will be used to fit a regression model.

Classification

We do not repeat the whole process for data preparation and missing values imputation. you can click here to see all the detail in my paper about **support vector machine** model.

Data partition

All the codes for the first steps are grouped in one chunk. If you notice we are using the same specified parameter values and seed numbers to be able to compare the results of the tow models **svm** and **knn** for **classification** (Using titanic data) and for regression (using BostonHousing data)

The last things we should do before training the model is converting the factors to be numerics and standardizing all the predictors for both sets (train and test), and finally we rename the target variable levels

```
train1 <- train %>% mutate_at(c(2,3,8),funs(as.numeric))
test1 <- test %>% mutate_at(c(2,3,8),funs(as.numeric))

processed<-preProcess(train1[,-1],method = c("center","scale"))
train1[,-1]<-predict(processed,train1[,-1])
test1[,-1]<-predict(processed,test1[,-1])

train1$Survived <- fct_recode(train1$Survived,died="0",surv="1")
test1$Survived <- fct_recode(test1$Survived,died="0",surv="1")
```

Train the model

The big advantage of the **k nearest neighbors** model is that it has one single parameters which make the tuning process very fast. Here also we will make use of the same seed as we did with **svm** model. for the resampling process we will stick with the default bootstrapped method with 25 resampling iterations.

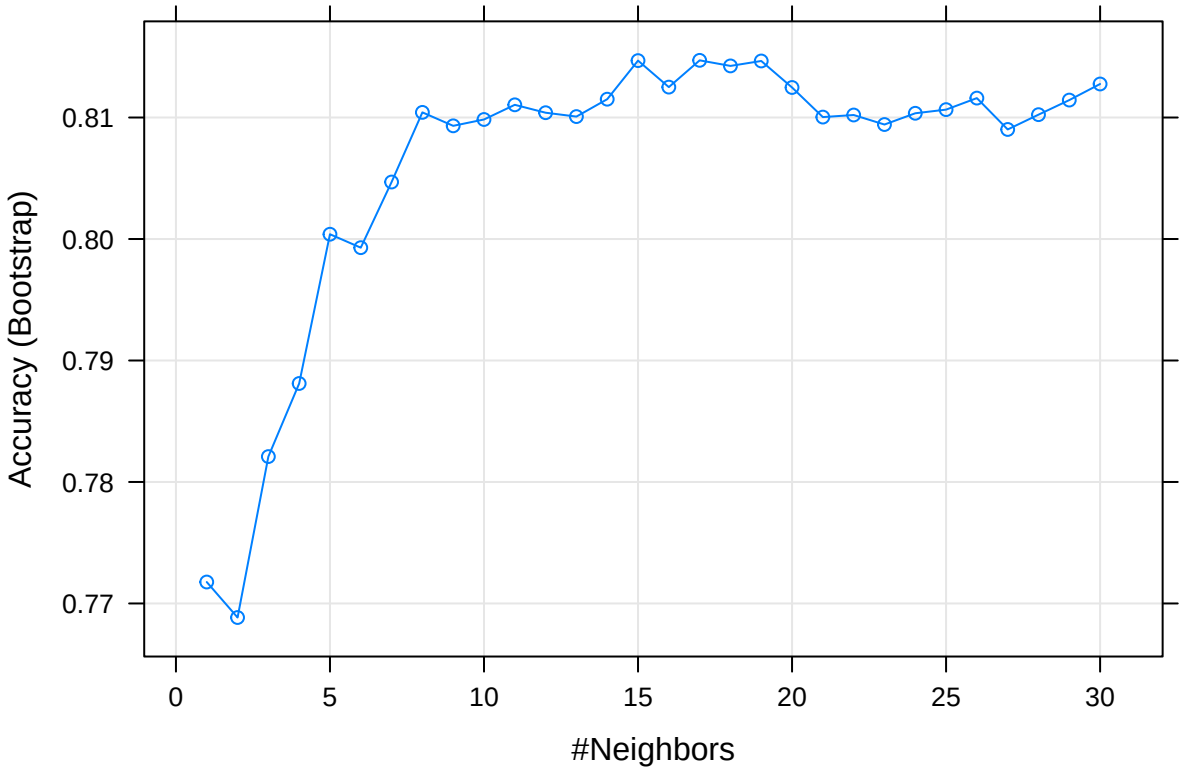
Let's now launch the model and get the summary.

```
set.seed(123)
modelknn <- train(Survived~., data=train1,
                  method="knn",
                  tuneGrid=expand.grid(k=1:30))
modelknn
```

```
## k-Nearest Neighbors
##
## 714 samples
##   7 predictor
##   2 classes: 'died', 'surv'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 714, 714, 714, 714, 714, 714, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.7717650  0.5165447
##    2  0.7688433  0.5088538
##    3  0.7820906  0.5370428
##    4  0.7881072  0.5487894
##    5  0.8003926  0.5733224
##    6  0.7992870  0.5711806
##    7  0.8046907  0.5827968
##    8  0.8104254  0.5950159
##    9  0.8093172  0.5927121
##   10  0.8098395  0.5937574
##   11  0.8110456  0.5957105
##   12  0.8103966  0.5942937
##   13  0.8100784  0.5939193
##   14  0.8115080  0.5960496
##   15  0.8146848  0.6026109
##   16  0.8125027  0.5979064
##   17  0.8147065  0.6015528
##   18  0.8142485  0.6002677
##   19  0.8146543  0.6003686
##   20  0.8124733  0.5960520
##   21  0.8100367  0.5906732
##   22  0.8102084  0.5893078
##   23  0.8094241  0.5873995
##   24  0.8103509  0.5891549
##   25  0.8106517  0.5895533
##   26  0.8116000  0.5909129
##   27  0.8090177  0.5853052
##   28  0.8102358  0.5882055
##   29  0.8114371  0.5905057
##   30  0.8127604  0.5937279
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 17.
```

The metric used to get the best parameter value is the **accuracy** rate , for which the best value is about 81.47% obtained at k=17. we can also get these values from the plot

```
plot(modelknn)
```



For the contributions of the predictors, the measure of importance scaled from 0 to 100 shows that the most important one is far the **Sex**, followed by **Fare** and **Pclass** , and the least important one is **SibSp**

```
varImp(modelknn)
```

```
## ROC curve variable importance
##
##           Importance
## Sex          100.000
## Fare          62.476
## Pclass        57.192
## Embarked      17.449
## Parch         17.045
## Age            4.409
## SibSp          0.000
```

Prediction and confusion matrix

Let's now use the test set to evaluate the model performance.

```
pred<-predict(modelknn,test1)
confusionMatrix(as.factor(pred),as.factor(test1$Survived))
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction died surv
##       died   99   26
##       surv   10   42
##
##                Accuracy : 0.7966
##                  95% CI : (0.7297, 0.8533)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 1.87e-07
##
##                   Kappa : 0.5503
##
##  Mcnemar's Test P-Value : 0.01242
##
##             Sensitivity : 0.9083
##             Specificity : 0.6176
##          Pos Pred Value : 0.7920
##          Neg Pred Value : 0.8077
##              Prevalence : 0.6158
##          Detection Rate : 0.5593
##    Detection Prevalence : 0.7062
##       Balanced Accuracy : 0.7630
##
##        'Positive' Class : died
##
```

We see that the accuracy has slightly decreased from 81.47% to 79.66. the closeness of this rates is a good sign that we do not face the **overfitting** problem.

Fine tuning the model

to seek improvements we can alter the metric. the best function that gives three importante metrics, **sensitivity**, **specivicity** and area under the **ROC** curve for each resampling iteration is **twoClassSummary**. Also we expand the grid search for the neighbors number to 30.

```
control <- trainControl(classProbs = TRUE,
                        summaryFunction = twoClassSummary)

set.seed(123)
modelknn1 <- train(Survived~., data=train1,
                method = "knn",
                trControl = control,
                tuneGrid = expand.grid(k=1:30))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
## not in the result set. ROC will be used instead.
```

```
modelknn1
```

```
## k-Nearest Neighbors
##
```

```
## 714 samples
##   7 predictor
##   2 classes: 'died', 'surv'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 714, 714, 714, 714, 714, 714, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    1  0.7637394  0.8092152  0.7114938
##    2  0.7959615  0.8102352  0.7013654
##    3  0.8212495  0.8217986  0.7180595
##    4  0.8351414  0.8302266  0.7201146
##    5  0.8455418  0.8448702  0.7283368
##    6  0.8543141  0.8441066  0.7269378
##    7  0.8564044  0.8477382  0.7350766
##    8  0.8590356  0.8526960  0.7421475
##    9  0.8617600  0.8511745  0.7414201
##   10  0.8611361  0.8512356  0.7424516
##   11  0.8621287  0.8546357  0.7399914
##   12  0.8633050  0.8542288  0.7392237
##   13  0.8647328  0.8526082  0.7407331
##   14  0.8656300  0.8572596  0.7369673
##   15  0.8663956  0.8612937  0.7388392
##   16  0.8657711  0.8595923  0.7359633
##   17  0.8658168  0.8652505  0.7322408
##   18  0.8659659  0.8657088  0.7301132
##   19  0.8667079  0.8685106  0.7261585
##   20  0.8668361  0.8657052  0.7252522
##   21  0.8673051  0.8641660  0.7212182
##   22  0.8672610  0.8701453  0.7118060
##   23  0.8675945  0.8703195  0.7094977
##   24  0.8677684  0.8724153  0.7087639
##   25  0.8681884  0.8733028  0.7080003
##   26  0.8681201  0.8768128  0.7048740
##   27  0.8680570  0.8748635  0.7011357
##   28  0.8685130  0.8745234  0.7047600
##   29  0.8686459  0.8756557  0.7055821
##   30  0.8681316  0.8754088  0.7094507
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 29.
```

This time we use the **ROC** to choose the best model which gives a different value of 29 with 0.8686 for the **ROC**.

```
pred<-predict(modelknn1,test1)
confusionMatrix(pred,test1$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction died surv
##       died   99   29
##       surv   10   39
##
##                Accuracy : 0.7797
##                  95% CI : (0.7113, 0.8384)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 2.439e-06
##
##                   Kappa : 0.5085
##
##  Mcnemar's Test P-Value : 0.003948
##
##             Sensitivity : 0.9083
##             Specificity : 0.5735
##          Pos Pred Value : 0.7734
##          Neg Pred Value : 0.7959
##              Prevalence : 0.6158
##          Detection Rate : 0.5593
##    Detection Prevalence : 0.7232
##       Balanced Accuracy : 0.7409
##
##        'Positive' Class : died
##
```

Using the **ROC** metric we get worse result for the accuracy rate which has decreased from 77.97% to 77.97.

Comparison between knn and svm model

Now let's train svm model with the same resamling method and we compare between them.

```
control<-trainControl(method="boot",number=25,
                      classProbs = TRUE,
                      summaryFunction = twoClassSummary)

modelsvm<-train(Survived~., data=train1,
                method="svmRadial",
                trControl=control)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
## not in the result set. ROC will be used instead.
```

```
modelsvm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 714 samples
##   7 predictor
##   2 classes: 'died', 'surv'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
```

```
## Summary of sample sizes: 714, 714, 714, 714, 714, 714, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##   0.25  0.8703474  0.8735475  0.7602162
##   0.50  0.8706929  0.8858278  0.7456306
##   1.00  0.8655619  0.8941179  0.7327856
##
## Tuning parameter 'sigma' was held constant at a value of 0.2282701
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2282701 and C = 0.5.
```

And let's get the confusion matrix.

```
pred<-predict(modelsvm,test1)
confusionMatrix(pred,test1$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction died surv
##       died  101   27
##       surv    8   41
##
##                Accuracy : 0.8023
##                  95% CI : (0.7359, 0.8582)
##     No Information Rate : 0.6158
##     P-Value [Acc > NIR] : 7.432e-08
##
##                   Kappa : 0.5589
##
##  Mcnemar's Test P-Value : 0.002346
##
##             Sensitivity : 0.9266
##             Specificity : 0.6029
##          Pos Pred Value : 0.7891
##          Neg Pred Value : 0.8367
##              Prevalence : 0.6158
##          Detection Rate : 0.5706
##    Detection Prevalence : 0.7232
##       Balanced Accuracy : 0.7648
##
##        'Positive' Class : died
##
```

we see that the accuracy fo this model is much higher with 80.23% than the knn model with 77.97% (the **modelknn1**). If we have a large number of models to be compared, there exists a function in **caret** called **resamples** to compare between models,but the models should have the same tarincontrol prameter values.

```
comp<-resamples(list( svm = modelsvm,
                      knn = modelknn1))

summary(comp)
```

```
##
## Call:
## summary.resamples(object = comp)
##
## Models: svm, knn
## Number of resamples: 25
##
## ROC
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## svm 0.8472858 0.8617944 0.8691093 0.8706929 0.8744979 0.9043001    0
## knn 0.8298966 0.8577167 0.8670815 0.8686459 0.8792487 0.9135638    0
##
## Sens
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## svm 0.8117647 0.8666667 0.8870056 0.8858278 0.9030303 0.9559748    0
## knn 0.8266667 0.8523490 0.8816568 0.8756557 0.8950617 0.9117647    0
##
## Spec
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## svm 0.6774194 0.7096774 0.7428571 0.7456306 0.7714286 0.8425926    0
## knn 0.5865385 0.6741573 0.6989247 0.7055821 0.7252747 0.8191489    0
```

Regression

First we call the **BostonHousing** data.

```
library(mlbench)
data("BostonHousing")
glimpse(BostonHousing)
```

```
## Observations: 506
## Variables: 14
## $ crim    <dbl> 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, ...
## $ zn      <dbl> 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5,...
## $ indus   <dbl> 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, ...
## $ chas    <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ nox     <dbl> 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524...
## $ rm      <dbl> 6.575, 6.421, 7.185, 6.998, 7.147, 6.430, 6.012, 6.172...
## $ age     <dbl> 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0,...
## $ dis     <dbl> 4.0900, 4.9671, 4.9671, 6.0622, 6.0622, 6.0622, 5.5605...
## $ rad     <dbl> 1, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, ...
## $ tax     <dbl> 296, 242, 242, 222, 222, 222, 311, 311, 311, 311, 311,...
## $ ptratio <dbl> 15.3, 17.8, 17.8, 18.7, 18.7, 18.7, 15.2, 15.2, 15.2, ...
## $ b       <dbl> 396.90, 396.90, 392.83, 394.63, 396.90, 394.12, 395.60...
## $ lstat   <dbl> 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.9...
## $ medv    <dbl> 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, ...
```

We will train a knn model to this data using the continuous variable as target **medv**

```
set.seed(1234)
index<-sample(nrow(BostonHousing),size = floor(0.8*(nrow(BostonHousing))))
train<-BostonHousing[index,]
```

```
test<-BostonHousing[-index,]

scaled<-preProcess(train[,-14],method=c("center","scale"))
trainscaled<-predict(scaled,train)
testscaled<-predict(scaled,test)
```

We are ready now to train our model.

```
set.seed(123)
modelknnR <- train(medv~., data=trainscaled,
                   method = "knn",
                   tuneGrid = expand.grid(k=1:60))
modelknnR
```

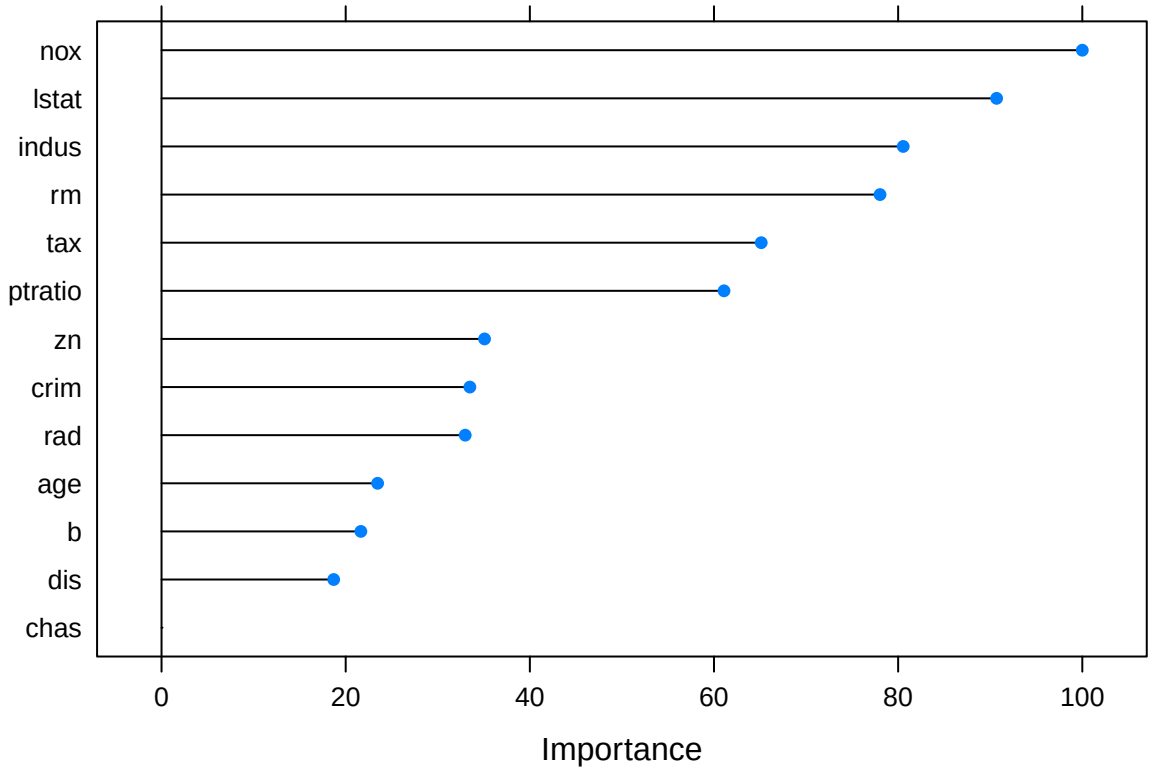
```
## k-Nearest Neighbors
##
## 404 samples
##  13 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 404, 404, 404, 404, 404, 404, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  4.711959  0.7479439  3.047925
##    2  4.600795  0.7545325  3.010235
##    3  4.554112  0.7583915  3.001404
##    4  4.416511  0.7733563  2.939100
##    5  4.414384  0.7736985  2.953741
##    6  4.405364  0.7758010  2.962082
##    7  4.375360  0.7799181  2.955250
##    8  4.409134  0.7773310  2.975489
##    9  4.427529  0.7770847  2.973016
##   10  4.414577  0.7804842  2.957983
##   11  4.447188  0.7787709  2.968389
##   12  4.475134  0.7767642  2.984709
##   13  4.489486  0.7760909  3.000489
##   14  4.518792  0.7746895  3.026858
##   15  4.554107  0.7717809  3.043645
##   16  4.583672  0.7694136  3.058097
##   17  4.599290  0.7695640  3.067001
##   18  4.632439  0.7671729  3.079895
##   19  4.670589  0.7643210  3.098643
##   20  4.708318  0.7614855  3.118593
##   21  4.736963  0.7596509  3.137784
##   22  4.756688  0.7590899  3.151654
##   23  4.781692  0.7577281  3.166203
##   24  4.813669  0.7554223  3.186575
##   25  4.843954  0.7533415  3.200120
##   26  4.872096  0.7513071  3.224031
##   27  4.896463  0.7502052  3.238489
##   28  4.920242  0.7497138  3.252959
```

```
##   29  4.944899  0.7484320  3.269227
##   30  4.966726  0.7479621  3.282756
##   31  4.996149  0.7460973  3.303607
##   32  5.024602  0.7438775  3.321013
##   33  5.055147  0.7420656  3.338457
##   34  5.083713  0.7403972  3.360867
##   35  5.108994  0.7388352  3.373694
##   36  5.132420  0.7372288  3.389177
##   37  5.156841  0.7354463  3.409025
##   38  5.175413  0.7349417  3.422294
##   39  5.196438  0.7340164  3.434986
##   40  5.225990  0.7314822  3.452499
##   41  5.249335  0.7299159  3.467267
##   42  5.275185  0.7281473  3.484101
##   43  5.300558  0.7263045  3.502388
##   44  5.322795  0.7251719  3.519220
##   45  5.349383  0.7232707  3.539266
##   46  5.376209  0.7210830  3.560509
##   47  5.398400  0.7199706  3.580476
##   48  5.424020  0.7180096  3.595497
##   49  5.445069  0.7166620  3.609308
##   50  5.469650  0.7145816  3.625718
##   51  5.492104  0.7127439  3.644329
##   52  5.515714  0.7107894  3.659286
##   53  5.535354  0.7092366  3.672172
##   54  5.562260  0.7063225  3.690854
##   55  5.581394  0.7049997  3.705917
##   56  5.600579  0.7036881  3.720464
##   57  5.623071  0.7018951  3.739874
##   58  5.645828  0.6999889  3.755824
##   59  5.662777  0.6990085  3.771570
##   60  5.682182  0.6976068  3.787733
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

The best model with k=7 for which the minimum RMSE is about 4.3757.

We can also get the importance of the predictors.

```
plot(varImp(modelknnR))
```



Then we get the prediction and the root mean squared error **RMSE** as follows.

```
pred<-predict(modelknnR,testscaled)
head(pred)
```

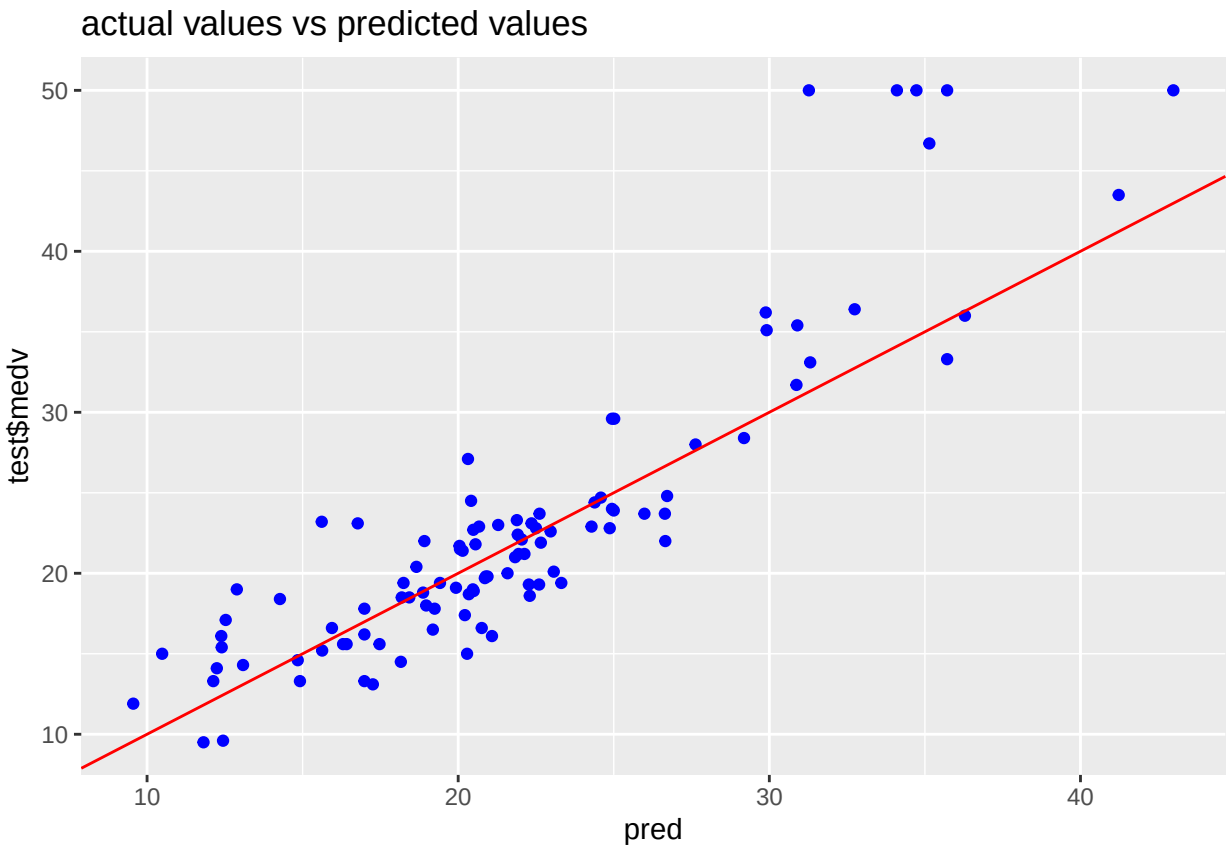
```
## [1] 24.94286 29.88571 20.67143 20.31429 19.18571 20.28571
```

```
RMSE(pred,test$medv)
```

```
## [1] 4.416328
```

The RMSE using the test set is about **4.4163** which is slightly greater than that of the training set **4.3757** . Finally we can plot the predictted values vs the observed values to get insight about their relationship.

```
ggplot(data.frame(predicted=pred,observed=test$medv),aes(pred,test$medv))+
  geom_point(col="blue")+
  geom_abline(col="red")+
  ggtitle("actual values vs predicted values")
```


actual values vs predicted values
test$medv
pred
50
40
30
20
10
10
20
30
40