Week 2 Assignment - Validation Using SVM & KNN and K-Means Clusters

Omer Farooq (EDx ID: mfarooq4)

1/22/2020

Table of Contents

QUESTION 3.1

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier:

Loading all needed libraries first.

```
library(kernlab)
library(kknn)
library(caret)
library(ggplot2)
library(reshape2)
```

- **(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional)**

I tried cross-validation for both KKNN and SVM models to see how results compared. I loaded the data and split it into 80% training & 20% testing data sets. For cross-validation, I trained the model on training dataset first and then used the testing data set to check the performance of the selected model.

```
#loading data
my_data <- read.delim("data_3.1/credit_card_data-headers.txt")
nrow(my_data)

## [1] 654

#Setting seed early on so that same data sets are reproduced in the future
set.seed(101)

#splitting data to training and validation
```

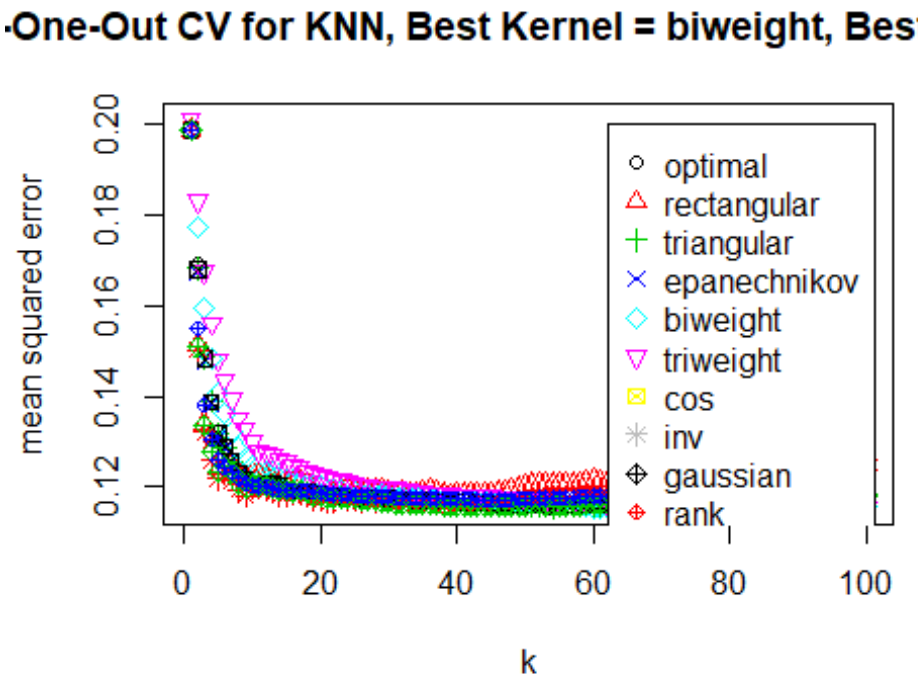
```
sample <- sample.int(n = nrow(my_data), size = floor(.80*nrow(my_data)),
replace = F)
train_data <- my_data[sample,]
test_data  <- my_data[-sample,]

nrow(train_data)

## [1] 523

nrow(test_data)

## [1] 131

head(train_data)

##     A1    A2    A3    A8 A9 A10 A11 A12 A14 A15 R1
## 430  1 20.00 7.000 0.500  0   1   0   1   0   0  0
## 95   0 28.50 1.000 1.000  1   0   2   0 167 500  0
## 209  1 48.17 3.500 3.500  1   1   0   1 230   0  1
## 442  0 23.00 1.835 0.000  0   0   1   1 200  53  0
## 351  1 39.42 1.710 0.165  0   1   0   1 400   0  0
## 315  1 34.83 2.500 3.000  0   1   0   1 200   0  0
```

There were couple of options to do cross-validation for KKNN model - TRAIN.KKNN or CV.KKNN. According to the R Documentation (https://www.rdocumentation.org/packages/kknn/versions/1.3.1/topics/train.kknn) Train.kknn performs leave-one-out cross-validation and is computationally very efficient. On the other hand, cv.kknn performs k-fold cross-validation and is generally slower and does not yet contain the test of different models. Due to this reason, I went with train.kknn function.

Train.kknn has the functionality to take a vector of different kernel options and try out different k values as well. This makes this function very handy.

```
kknn_cv <- train.kknn(R1~.,
                      #using the 80% partioned training data for cross-
validation
                      train_data,

                      #trying out upto 100 K values
                      kmax = 100,

                      #trying out all kernel options
                      kernel =
c("optimal","rectangular","triangular","epanechnikov","biweight","triweight",
"cos","inv", "gaussian", "rank"),

                      #scaling data
                      scale =TRUE
                      )
```

```
#plotting output of the model
plot(kknn_cv)

#variables for best kernel and best K value from the model
kernel <- kknn_cv$best.parameters$kernel
bestk <- kknn_cv$best.parameters$k
title(sprintf("Leave-One-Out CV for KNN, Best Kernel = %s, Best K Value =
%s", kernel, bestk))
```




```
#printing the model output
kknn_cv

##
## Call:
## train.kknn(formula = R1 ~ ., data = train_data, kmax = 100, kernel =
c("optimal",     "rectangular", "triangular", "epanechnikov", "biweight",
"triweight", "cos", "inv", "gaussian", "rank"), scale = TRUE)
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1988528
## Minimal mean squared error: 0.1152105
## Best kernel: biweight
## Best k: 70
```

Above plot showed the performance of different kernels for different K values. The output object had a suggestion that biweight kernel is the best performant and the best K value is

70. After leave-one-out cross validation on different models, the train function automatically trained the model on the full training data after a model and K selection had been made.

The below code compared the final biweight kernel model based in k=70 to the test data. The model showed an accuracy of 91.6% and seemed to be performing pretty well.

```
#comparing the model output with the test data
predictions <- predict(kknn_cv,newdata = test_data[,1:10])

#converting probabilities to 0 and 1 (0.5 and below to 0 and 0.5 and above to
1)
predictions01 <- as.integer(predictions+0.5)

#creating a confusion matrix and basic stats of the final model using test
data set
confusionMatrix(factor(test_data[,11], levels =
c(1,0)),factor(predictions01,levels=c(1,0)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 51  4
##          0  7 69
##
##                Accuracy : 0.916
##                  95% CI : (0.8547, 0.9573)
##     No Information Rate : 0.5573
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8289
##
##  Mcnemar's Test P-Value : 0.5465
##
##             Sensitivity : 0.8793
##             Specificity : 0.9452
##          Pos Pred Value : 0.9273
##          Neg Pred Value : 0.9079
##              Prevalence : 0.4427
##          Detection Rate : 0.3893
##    Detection Prevalence : 0.4198
##       Balanced Accuracy : 0.9123
##
##        'Positive' Class : 1
##
```

Next, I tried the cross-validation training of KSVM model as well. KSVM function has an argument “Cross”. A greater than 0 value of Cross argument in KSVM performs the k-fold validation of the training data. E.g. Cross=10 will perform 10 fold cross validation of the

training data. Since it was mentioned in the lecture that K=10 for k-fold validation is common, I used that value as well.

The code below tried several values of C for the KSVM model with 10 fold validation on training data set.

```
#range of C values to test
TestCs <- c(10**(-10:10))

iter = length(TestCs)

#a blank matrix to record accuracies
results <- matrix(NA, nrow=iter, ncol=2)
colnames(results) <- c("Cvalue","Accuracy")

#looping Cs on the model with cross-validation for 10 folds and documenting
accuracy percentage
for(i in 1:iter){
    ksvm_cv <- ksvm( x = as.matrix(train_data[,1:10]),
                     y = as.factor(train_data[,11]),
                     type='C-svc',
                     kernel= 'vanilladot',
                     C=TestCs[[i]],
                     Cross = 10,
                     scaled=TRUE
                     )
    predictions <- predict(ksvm_cv,newdata = test_data[,1:10])
    results[i,] <- c(TestCs[[i]], sum(predictions == test_data[,11]) /
nrow(test_data))
}

##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters

results

##       Cvalue  Accuracy
##  [1,]  1e-10 0.5801527
##  [2,]  1e-09 0.5801527
##  [3,]  1e-08 0.5801527
##  [4,]  1e-07 0.5801527
##  [5,]  1e-06 0.5801527
##  [6,]  1e-05 0.5801527
##  [7,]  1e-04 0.5801527
##  [8,]  1e-03 0.9083969
##  [9,]  1e-02 0.9236641
## [10,]  1e-01 0.9236641
## [11,]  1e+00 0.9236641
## [12,]  1e+01 0.9236641
## [13,]  1e+02 0.9236641
## [14,]  1e+03 0.9236641
## [15,]  1e+04 0.9236641
## [16,]  1e+05 0.9236641
## [17,]  1e+06 0.6717557
## [18,]  1e+07 0.4045802
## [19,]  1e+08 0.4503817
## [20,]  1e+09 0.6412214
## [21,]  1e+10 0.6412214
```

Based on the accuracy measure for several C values, it appeared that C=100 to C=100,000 have the same accuracy. Thus, I picked the lower C=100 moving forward with this analysis.

Next, I tried several non-linear kernels as well with C=100 and Cross=10.

```
#different kernels to try
kernels <-
c('rbfdot','polydot','tanhdot','laplacedot','besseldot','anovadot','splinedot
')

iter = length(kernels)

#matrix to record accuracies for different kernels
kernel_results <- matrix(NA, nrow=iter, ncol=2)
colnames(kernel_results) <- c("Kernel","Accuracy")

#looping different kernel on the model and documenting accuracy percentage
for C=100
for(i in 1:iter){
    ksvm_kernels <- ksvm( x = as.matrix(train_data[,1:10]),
                    y = as.factor(train_data[,11]),
                    type='C-svc',
                    kernel= kernels[[i]],
```

```
                    C=100,
                    Cross=10,
                    scaled=TRUE
                    )
    predc100 <- predict(ksvm_kernels,newdata = test_data[,1:10])
    kernel_results[i,] <- c(kernels[[i]],
                            sum(predc100 == test_data[,11]) /
nrow(test_data))
}

##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters

kernel_results

##      Kernel       Accuracy
## [1,] "rbfdot"     "0.847328244274809"
## [2,] "polydot"    "0.923664122137405"
## [3,] "tanhdot"    "0.824427480916031"
## [4,] "laplacedot" "0.893129770992366"
## [5,] "besseldot"  "0.801526717557252"
## [6,] "anovadot"   "0.908396946564885"
## [7,] "splinedot"  "0.824427480916031"
```

Results of above code revealed that highest accuracy was 92.37% which was the same as linear kernel. Thus, I could safely stick to linear kernel, C=100 and Cross=10 for our final KSVM model.

The code below trained the final model with training data and checked its accuracy on test data.The model achieves an accuracy of 92.37%.

```
#final KSVM model with 10 fold cross validation.
ksvm_cv <- ksvm( x = as.matrix(train_data[,1:10]),
                    y = as.factor(train_data[,11]),
                    type='C-svc',
                    kernel= 'vanilladot',
                    C=100,
                    Cross = 10,
                    scaled=TRUE
                    )

##  Setting default kernel parameters

#testing the model on training data set
final_predictions <- predict(ksvm_cv,newdata = test_data[,1:10])
confusionMatrix(factor(test_data[,11], levels = c(1,0)),final_predictions)

## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 68  8
##          1  2 53
##
##                Accuracy : 0.9237
##                  95% CI : (0.8641, 0.9628)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8456
##
##  Mcnemar's Test P-Value : 0.1138
##
##             Sensitivity : 0.9714
##             Specificity : 0.8689
##          Pos Pred Value : 0.8947
##          Neg Pred Value : 0.9636
##              Prevalence : 0.5344
##          Detection Rate : 0.5191
##    Detection Prevalence : 0.5802
##       Balanced Accuracy : 0.9201
##
##        'Positive' Class : 0
##
```

- **(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).**

For this part of the homework, I split the total data set into 3 parts - training (60%), validation (20%) and test (20%).

```
sections = c(train = .6, test = .2, validate = .2)

sec = sample(cut(seq(nrow(my_data)),
                 nrow(my_data)*cumsum(c(0,sections)),
                 labels = names(sections)))

new_data = split(my_data, sec)

nrow(new_data$train)

## [1] 392

nrow(new_data$validate)

## [1] 131

nrow(new_data$test)

## [1] 131
```

Next, I used the training dataset to test several C values for the KSVM model and checked the performance of all models on validation data set.

```
#range of C values to test
TestCs <- c(10**(-10:10))

iter = length(TestCs)

#a blank matrix to record accuracies
results_new <- matrix(NA, nrow=iter, ncol=2)
colnames(results_new) <- c("Cvalue","Accuracy")

#looping Cs on the model using training data set and documenting accuracy
percentage
for(i in 1:iter){
    ksvm_train <- ksvm( x = as.matrix(new_data$train[,1:10]),
                    y = as.factor(new_data$train[,11]),
                    type='C-svc',
                    kernel= 'vanilladot',
                    C=TestCs[[i]],
                    scaled=TRUE
                    )

    #checking models' performance on validation data
    predictions_new <- predict(ksvm_train,newdata = new_data$validate[,1:10])
    results_new[i,] <- c(TestCs[[i]], sum(predictions_new ==
new_data$validate[,11]) / nrow(new_data$validate))
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
results_new
```

```
##       Cvalue  Accuracy
##  [1,]  1e-10 0.5648855
##  [2,]  1e-09 0.5648855
##  [3,]  1e-08 0.5648855
##  [4,]  1e-07 0.5648855
##  [5,]  1e-06 0.5648855
##  [6,]  1e-05 0.5648855
##  [7,]  1e-04 0.5648855
##  [8,]  1e-03 0.7709924
##  [9,]  1e-02 0.8702290
## [10,]  1e-01 0.8702290
## [11,]  1e+00 0.8702290
## [12,]  1e+01 0.8702290
## [13,]  1e+02 0.8702290
## [14,]  1e+03 0.8702290
## [15,]  1e+04 0.8702290
## [16,]  1e+05 0.8778626
## [17,]  1e+06 0.6564885
## [18,]  1e+07 0.6793893
## [19,]  1e+08 0.3664122
## [20,]  1e+09 0.7709924
## [21,]  1e+10 0.7709924
```

As before, C=100 came out to be a good C values with 87.02% accuracy. Next, I trained the model with C=100 and checked its performance on testing daa set. The model showed the same 87.02% accuracy.

```
#building final model using C=100
ksvm_train_final <- ksvm( x = as.matrix(new_data$train[,1:10]),
                    y = as.factor(new_data$train[,11]),
                    type='C-svc',
                    kernel= 'vanilladot',
                    C=100,
                    scaled=TRUE
                    )
```

```
##  Setting default kernel parameters
```

```
#checking performance of the model on test data
predictions_final <- predict(ksvm_train_final,newdata = new_data$test[,1:10])
confusionMatrix(factor(new_data$test[,11], levels =
c(1,0)),predictions_final)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 62 16
##          1  1 52
```

```
##
##                Accuracy : 0.8702
##                  95% CI : (0.8004, 0.9226)
##     No Information Rate : 0.5191
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7423
##
##  Mcnemar's Test P-Value : 0.000685
##
##             Sensitivity : 0.9841
##             Specificity : 0.7647
##          Pos Pred Value : 0.7949
##          Neg Pred Value : 0.9811
##              Prevalence : 0.4809
##          Detection Rate : 0.4733
##    Detection Prevalence : 0.5954
##       Balanced Accuracy : 0.8744
##
##        'Positive' Class : 0
##
```

I also used the 3 data sets on KKNN model. I checked the KKNN model for 100 K values using training data set and checked performance on validation data.

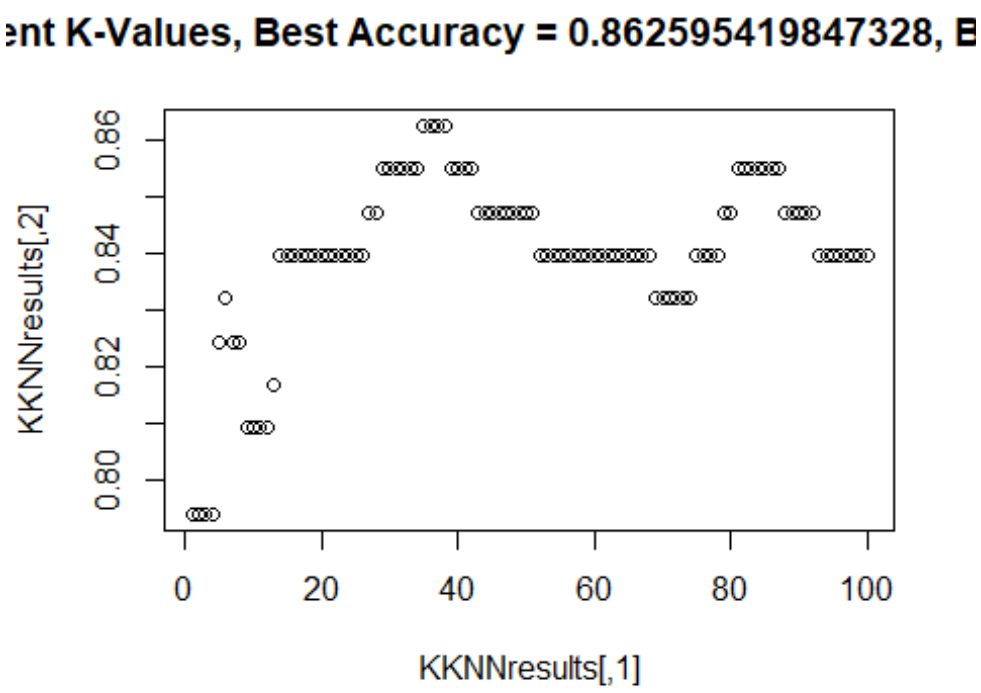
```
iter = 100

KKNNresults <- matrix(NA, nrow=iter, ncol=2)
colnames(results) <- c("K-Value","Accuracy")

#looping Ks on the model on training data and documenting accuracy percentage
for(i in 1:iter){
    model_kknn <- kknn( R1~.,
                    new_data$train,
                    new_data$validate,
                    k=i,
                    distance = 2,
                    kernel = "optimal",
                    scale= TRUE
                    )
    predictions_knn <- predict(model_kknn,newdata = new_data$validate[,1:10])
    predictions_knn_01 <- as.integer(predictions_knn+0.5)
    KKNNresults[i,] <- c(i, sum(predictions_knn_01 == new_data$validate[,11])
/ nrow(new_data$validate))
}

KKNNresults
```

```
##        [,1]      [,2]
##   [1,]    1 0.7938931
```

```
##   [2,]    2 0.7938931
##   [3,]    3 0.7938931
##   [4,]    4 0.7938931
##   [5,]    5 0.8244275
##   [6,]    6 0.8320611
##   [7,]    7 0.8244275
##   [8,]    8 0.8244275
##   [9,]    9 0.8091603
##  [10,]   10 0.8091603
##  [11,]   11 0.8091603
##  [12,]   12 0.8091603
##  [13,]   13 0.8167939
##  [14,]   14 0.8396947
##  [15,]   15 0.8396947
##  [16,]   16 0.8396947
##  [17,]   17 0.8396947
##  [18,]   18 0.8396947
##  [19,]   19 0.8396947
##  [20,]   20 0.8396947
##  [21,]   21 0.8396947
##  [22,]   22 0.8396947
##  [23,]   23 0.8396947
##  [24,]   24 0.8396947
##  [25,]   25 0.8396947
##  [26,]   26 0.8396947
##  [27,]   27 0.8473282
##  [28,]   28 0.8473282
##  [29,]   29 0.8549618
##  [30,]   30 0.8549618
##  [31,]   31 0.8549618
##  [32,]   32 0.8549618
##  [33,]   33 0.8549618
##  [34,]   34 0.8549618
##  [35,]   35 0.8625954
##  [36,]   36 0.8625954
##  [37,]   37 0.8625954
##  [38,]   38 0.8625954
##  [39,]   39 0.8549618
##  [40,]   40 0.8549618
##  [41,]   41 0.8549618
##  [42,]   42 0.8549618
##  [43,]   43 0.8473282
##  [44,]   44 0.8473282
##  [45,]   45 0.8473282
##  [46,]   46 0.8473282
##  [47,]   47 0.8473282
##  [48,]   48 0.8473282
##  [49,]   49 0.8473282
##  [50,]   50 0.8473282
##  [51,]   51 0.8473282
```

```
##  [52,]   52 0.8396947
##  [53,]   53 0.8396947
##  [54,]   54 0.8396947
##  [55,]   55 0.8396947
##  [56,]   56 0.8396947
##  [57,]   57 0.8396947
##  [58,]   58 0.8396947
##  [59,]   59 0.8396947
##  [60,]   60 0.8396947
##  [61,]   61 0.8396947
##  [62,]   62 0.8396947
##  [63,]   63 0.8396947
##  [64,]   64 0.8396947
##  [65,]   65 0.8396947
##  [66,]   66 0.8396947
##  [67,]   67 0.8396947
##  [68,]   68 0.8396947
##  [69,]   69 0.8320611
##  [70,]   70 0.8320611
##  [71,]   71 0.8320611
##  [72,]   72 0.8320611
##  [73,]   73 0.8320611
##  [74,]   74 0.8320611
##  [75,]   75 0.8396947
##  [76,]   76 0.8396947
##  [77,]   77 0.8396947
##  [78,]   78 0.8396947
##  [79,]   79 0.8473282
##  [80,]   80 0.8473282
##  [81,]   81 0.8549618
##  [82,]   82 0.8549618
##  [83,]   83 0.8549618
##  [84,]   84 0.8549618
##  [85,]   85 0.8549618
##  [86,]   86 0.8549618
##  [87,]   87 0.8549618
##  [88,]   88 0.8473282
##  [89,]   89 0.8473282
##  [90,]   90 0.8473282
##  [91,]   91 0.8473282
##  [92,]   92 0.8473282
##  [93,]   93 0.8396947
##  [94,]   94 0.8396947
##  [95,]   95 0.8396947
##  [96,]   96 0.8396947
##  [97,]   97 0.8396947
##  [98,]   98 0.8396947
##  [99,]   99 0.8396947
## [100,]  100 0.8396947
```

```r
plot(KKNNresults)
kmax <- max(KKNNresults[,2])
kwhich <- which.max(KKNNresults[,2])
title(sprintf("Different K-Values, Best Accuracy = %s, Best K = %s", kmax,
kwhich))
```



This helped determine the best k = 35 which I used to build the final model using training data set and checked its performance on testing data which showed 86.26% accuracy.

```r
#building final KKNN modelbased on K=35 and using train and test data
model_kknn_final <- kknn( R1~.,
                    new_data$train,
                    new_data$test,
                    k=35,
                    distance = 2,
                    kernel = "optimal",
                    scale= TRUE
                    )

#assessing model performance with test data
predictions_knn_final <- predict(model_kknn_final,newdata =
new_data$test[,1:10])
predictions_knn_final_01 <- as.integer(predictions_knn_final+0.5)
confusionMatrix(factor(new_data$test[,11], levels =
c(1,0)),factor(predictions_knn_final_01,levels=c(1,0)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 48  5
##          0 13 65
##
##                Accuracy : 0.8626
##                  95% CI : (0.7916, 0.9165)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : 1.343e-15
##
##                   Kappa : 0.7215
##
##  Mcnemar's Test P-Value : 0.09896
##
##             Sensitivity : 0.7869
##             Specificity : 0.9286
##          Pos Pred Value : 0.9057
##          Neg Pred Value : 0.8333
##              Prevalence : 0.4656
##          Detection Rate : 0.3664
##    Detection Prevalence : 0.4046
##       Balanced Accuracy : 0.8577
##
##        'Positive' Class : 1
##
```

QUESTION 4.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

At my job at the T-Mobile HQ in the Seattle area, my team helps get analytics products built for our procurement and supply chain teams. One of the key insights we provide to our network supply chain organization is the location of the new local warehouses (called Market Staging Locations - MSLs). Our supply chain organization supports the planning, procurement, warehousing and logistics of materials needed to build out and enchance T-Mobile's wireless network. The MSLs are the small warehouses closer to the cities where the materials are shipped from the larger regional distribution center. The material is then issued out to the general contractors for build-out at the actual tower location.

The MSLs need to be continuously optimized with the upcoming build schedule of the tower locations. Some MSLs are closed down and new locations, closer to where high build activity is expected, are set up. This is a good clustering problem where clusters and cluster centers would indicate a good location for the new MSLs.

Further details are given below:

- **Total features (columns) available:** 100+
- **Why is this important?** A timely supply of network gear and materials is extremely important to build out or enhance T-Mobile's wireless network. The Market Staging Locations play a pivotal role in the supply chain.
- **Predictors Useful for the Model:**
  - **Planned Tower Sites** - There are a few predictors available in the data which tell us about the upcoming demand of how many new tower locations are expected to be built or enhanced. Where those locations will be and when is the build scheduled to start.
  - **Bill of Material**- Each planned tower site has an associated Bill of Material (BOM). BOM tells us exactly which material is needed and how much of each material is needed. This helps determine the total expected material need.
  - **SKU Details** - Details of each SKU included in the BOM like part size, weight, dimensions etc. This helps determine the pallet requirements, which ultimately helps with storage capacity analysis.
  - **Prioritization** - Each planned tower location has a prioritization score assigned by the engineering team. This helps determine prioritization of material among locations planned within the same time frame.

QUESTION 4.2

The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). *The response values are only given to see how well a specific method performed and should not be used to build the model*. Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

First of all, I loaded the **iris** data set and applied scaling on the 4 predictors.

```
#loading data
iris <- read.table(file = "data_4.2/iris.txt", row.names = 1)
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
```

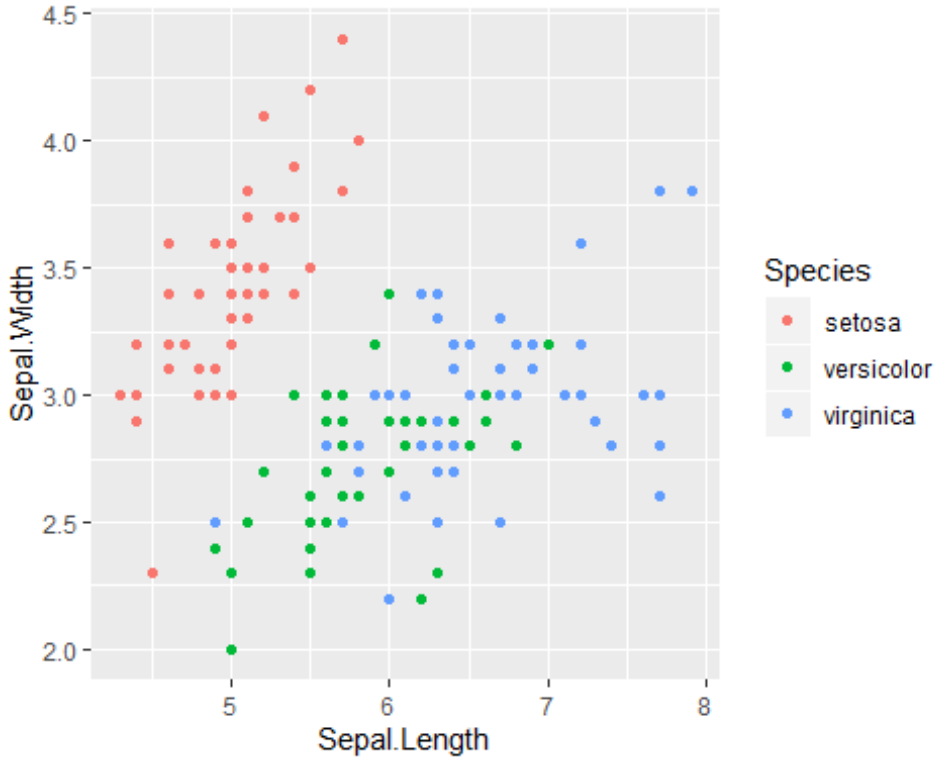
```
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
#scaling the predictors
iris_scale <- scale(iris[,1:4])
head(iris_scale)
```

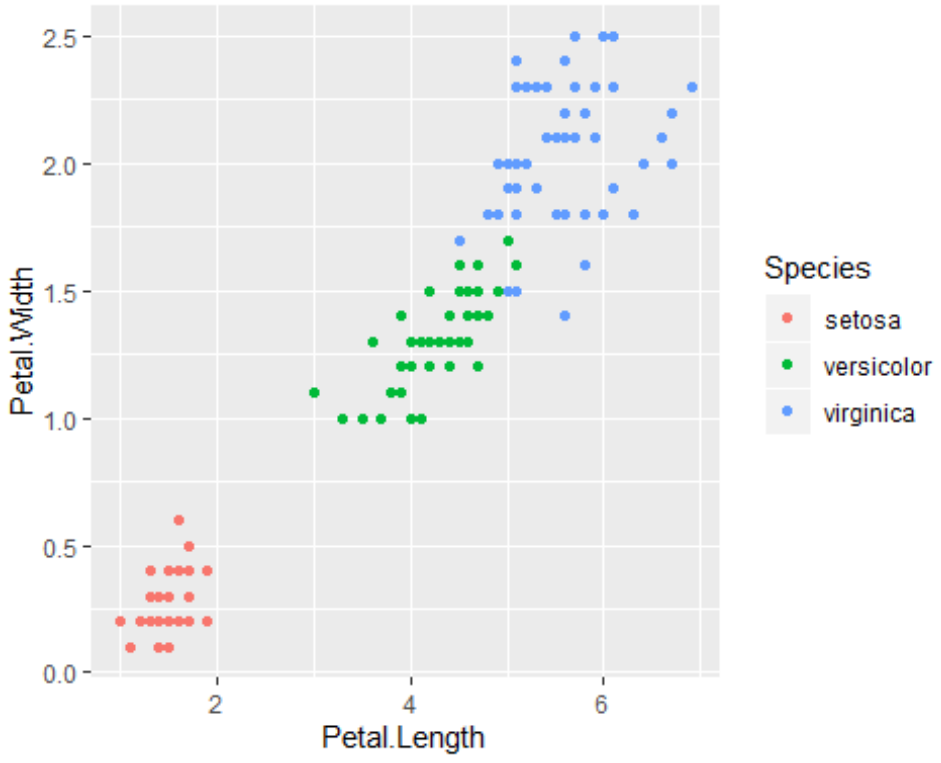
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1   -0.8976739  1.01560199    -1.335752   -1.311052
## 2   -1.1392005 -0.13153881    -1.335752   -1.311052
## 3   -1.3807271  0.32731751    -1.392399   -1.311052
## 4   -1.5014904  0.09788935    -1.279104   -1.311052
## 5   -1.0184372  1.24503015    -1.335752   -1.311052
## 6   -0.5353840  1.93331463    -1.165809   -1.048667
```

Next step, I plotted the data points against 4 different combinations of 2 sets of predictors i.e sepal length vs sepal width, petal length vs petal width, petal length vs sepal width and sepal length vs petal width. Goal of this exercise wasto visually check how the data points formed clusters, if any, and got indication on which predictors might be useful in the K-means algorithm.

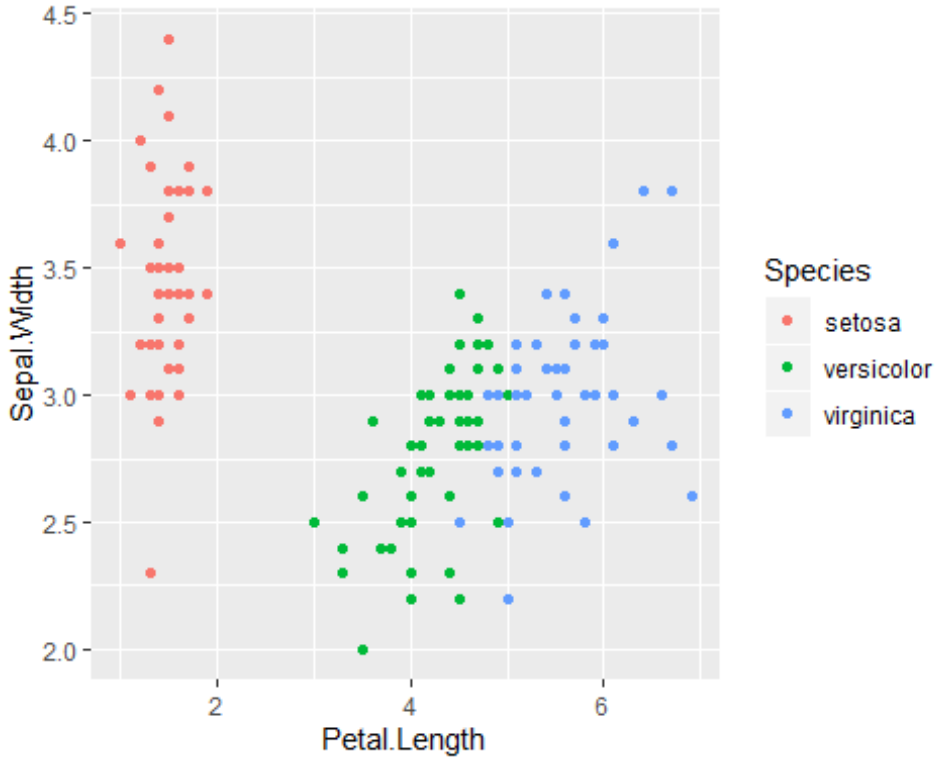
```
#plotting different 2 predictor combinations to visually see clusters
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species))+geom_point()
```



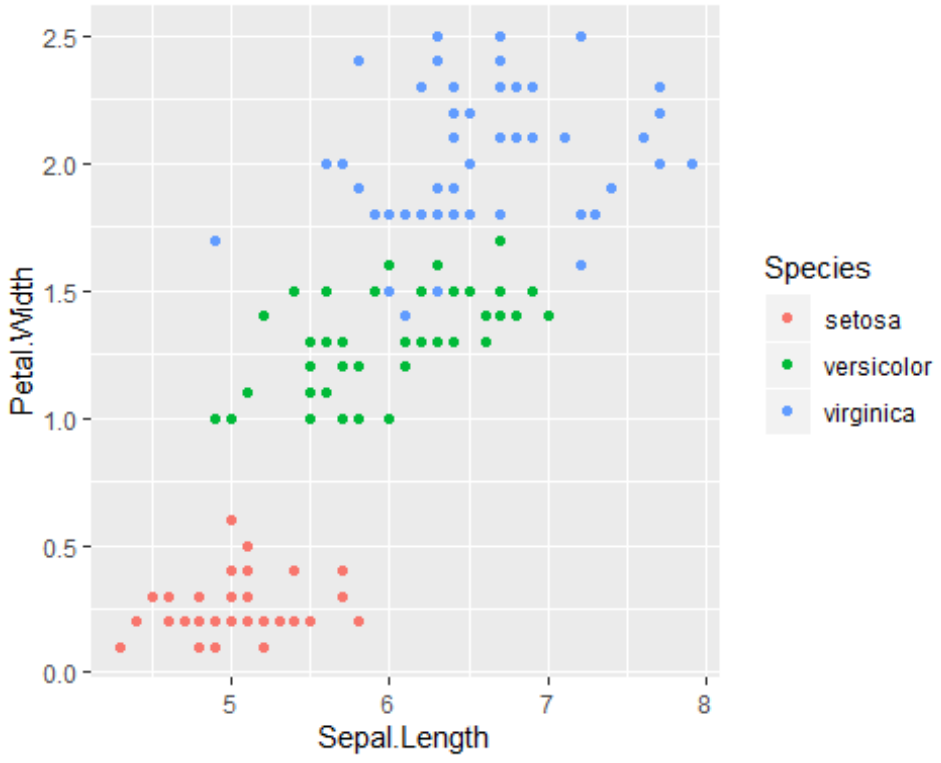

```
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species))+geom_point()
```




```
ggplot(iris, aes(Petal.Length, Sepal.Width, color = Species))+geom_point()
```




```
ggplot(iris, aes(Sepal.Length, Petal.Width, color = Species))+geom_point()
```



Above plots clearly showed that petal width and petal length or any combination that involved petal width or petal length formed better clusters. But this hypothesis had to be tested systematically.

Following code checked all 15 combinations of predictors for 1 to 10 centers for each combination and logged the total withinss (total sum of squares) in a matrix. Total withinss is the measure of compactness of clusters. Smaller value of total withinss indicates better clusters.

```
k.max <- 10

combs <-
list(c(1,2,3,4),c(1,2,3),c(1,2,4),c(1,3,4),c(2,3,4),c(1,2),c(1,3),c(1,4),c(2,
3),c(2,4),c(3,4),c(1),c(2),c(3),c(4))

predictors <- c(1234, 123, 124, 134, 234, 12, 13, 14, 23, 24, 34, 1, 2, 3, 4)

iter = 15

kmeans_results <- matrix(NA, nrow=iter, ncol=11)
colnames(kmeans_results) <-
c("Predictors","K=1","K=2","K=3","K=4","K=5","K=6","K=7","K=8","K=9","K=10")


for(i in 1:iter){
  #print(combs[[i]])
  kmeans_model <- sapply(1:k.max,function(k){kmeans(iris_scale[,combs[[i]]],
k, nstart = 25, iter.max = 20)$tot.withinss})
  #print(round(kmeans_model,digits=2))
  kmeans_results[i,] <- c(predictors[[i]],round(kmeans_model,digits=2))

}

kmeans_results
```

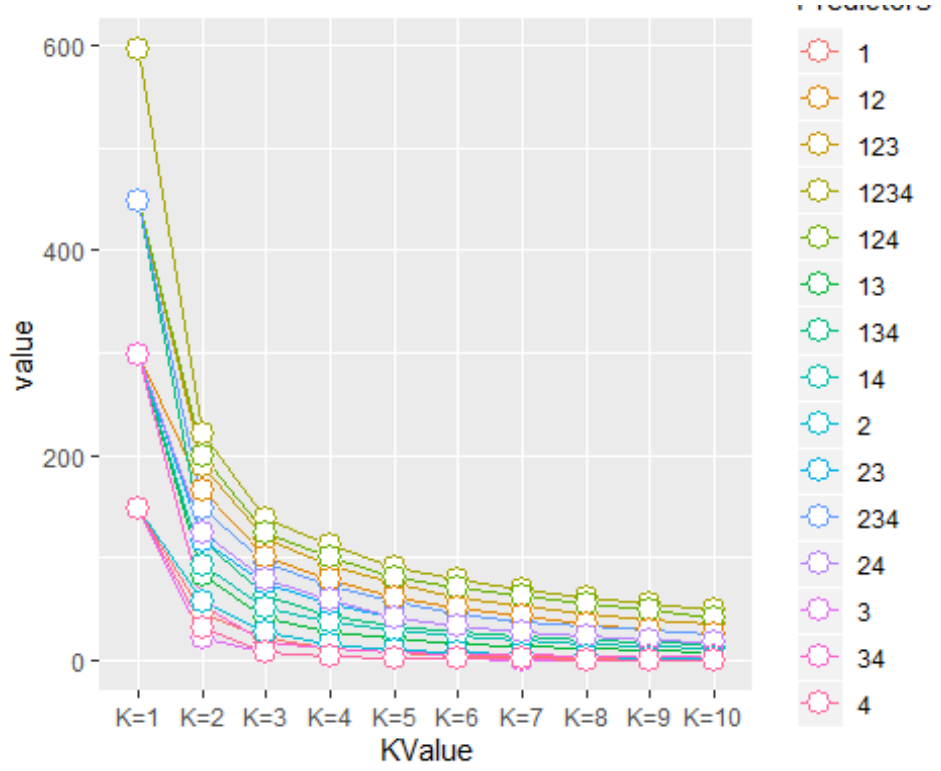
```
##       Predictors K=1    K=2    K=3    K=4   K=5   K=6   K=7   K=8   K=9  K=10
##  [1,]       1234 596 220.88 138.89 113.33 90.20 80.10 70.19 62.03 54.74 48.92
##  [2,]        123 447 189.20 118.34  94.01 75.04 62.43 53.90 45.96 39.55 35.27
##  [3,]        124 447 198.75 124.50 100.83 80.39 71.04 63.05 55.22 49.33 42.09
##  [4,]        134 447 117.62  62.62  44.64 34.39 28.73 23.63 20.52 18.23 16.17
##  [5,]        234 447 148.57  94.68  73.78 56.73 45.54 38.82 33.57 29.27 26.02
##  [6,]         12 298 165.84 101.93  79.22 61.40 52.09 43.84 35.90 30.22 26.43
##  [7,]         13 298  84.16  42.75  28.30 22.25 16.73 14.40 11.64  9.78  8.18
##  [8,]         14 298  93.40  50.80  38.34 30.01 24.37 19.68 17.10 14.52 12.43
##  [9,]         23 298 115.88  74.62  54.71 40.92 34.75 28.49 23.43 20.92 17.82
## [10,]         24 298 124.53  78.71  58.79 42.61 34.48 28.33 23.56 20.06 17.51
## [11,]         34 298  53.81  17.91  12.20  9.09  7.12  5.95  5.09  4.36  3.85
## [12,]          1 149  45.09  22.98  12.04  8.07  5.29  3.99  2.89  2.19  1.70
## [13,]          2 149  56.83  27.69  16.04 10.17  7.81  5.74  3.70  3.21  2.60
## [14,]          3 149  21.69   7.87   4.04  2.79  1.90  1.36  1.09  0.84  0.70
## [15,]          4 149  31.68   8.46   4.75  2.90  2.22  1.57  1.21  0.86  0.66
```

From the table above, I needed to find two answers:

- What is the best number of centers for our analysis (K)?
- What is the best combination of predictors?

I plotted the total withinss of each predictors combination against K=1 to 10 values to get elbow diagrams for each predictors combination. This helped determine the best K.

```
#converting table to dataframe and converting predictor column to characters
kmeans_results_df<- as.data.frame(kmeans_results)
kmeans_results_df$Predictors <- as.character(kmeans_results_df$Predictors)

#resharing the dataframe
new_df <- melt(kmeans_results_df, id.vars="Predictors", value.name="value",
variable.name="KValue")

#plotting
ggplot(data=new_df, aes(x=KValue, y=value, group = Predictors, colour =
Predictors)) +
    geom_line() +
    geom_point( size=4, shape=21, fill="white")
```



Looking at the elbow diagrams above, it appeared that K=3 was a good number. The total sum of squares did not drop significantly for K>3 (could also be seen in the values in the table above).

Next, I separated out the K=3 total withinss values for all predictors combinations and sorted them low to high. Lowest value of total withinss indicated the best predictor combinations.

Unsurprisingly, the 3 lowest total withinss values were for predicotors 3, 4 and (3,4). Predictor 3 is Sepal length and 4 is petal width indicating these two as the best predictors for our analysis.

```
kmeans_results_new <- kmeans_results[,c(1,4)]
kmeans_results_new[order(kmeans_results_new[,2]),]
```

```
##       Predictors    K=3
##  [1,]          3   7.87
##  [2,]          4   8.46
##  [3,]         34  17.91
##  [4,]          1  22.98
##  [5,]          2  27.69
##  [6,]         13  42.75
##  [7,]         14  50.80
##  [8,]        134  62.62
##  [9,]         23  74.62
## [10,]         24  78.71
## [11,]        234  94.68
## [12,]         12 101.93
## [13,]        123 118.34
## [14,]        124 124.50
## [15,]       1234 138.89
```

Now that I had answers to both key questions i.e. ideal number of centers of 3 and best predictors being petal length and petal width, I tried the Kmeans algorithm with these parameters to find the clusters.

I used nstart = 25 meaning 25 different random starting assignments were tried and then the one with lowest total withinss was selected. Thus, a sligthly higher nstart was better.

I picked iter.max = 20. Default value is 10 but I doubled it to allow for more iterations to ensure accuracy.

```
plpw_cluster <- kmeans(iris_scale[,3:4],3,nstart = 25, iter.max=20)
table(plpw_cluster$cluster,iris$Species)
```

```
##
##     setosa versicolor virginica
##   1      0         48         4
##   2      0          2        46
##   3     50          0         0
```

Above shown above, Using both petal length and petal width as predictors got an accuracy of 144/ 150 = 96%.

I then tried the model with only petal length as the predictor as well and got an accuracy of 142/ 150 = 94.6%

```
pl_cluster <- kmeans(iris_scale[,3],3,nstart = 25, iter.max=20)
table(pl_cluster$cluster,iris$Species)

##
##     setosa versicolor virginica
##   1      0          2        44
##   2      0         48         6
##   3     50          0         0
```

Finally, I tried with only petal width as predictor and got an accuracy of 96%. This indicated that all 3 models with petal length, petal width or both used as predictors were very accurate and provided good clusters.

```
pw_cluster <- kmeans(iris_scale[,4],3,nstart = 25, iter.max=20)
table(pw_cluster$cluster,iris$Species)

##
##     setosa versicolor virginica
##   1      0         48         4
##   2      0          2        46
##   3     50          0         0
```

Just for reference, building the model with all 4 predictors dropped the accuracy to 125/150 = 83.3%. This confirmed that my selection of predicotrs was correct and it provided the best clustering of the iris data.

```
all4_cluster <- kmeans(iris_scale,3,nstart = 25, iter.max=20)
table(all4_cluster$cluster,iris$Species)

##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         39        14
##   3      0         11        36
```