MSDS680 - Week 2 - KNN

Benjamin Siebold

Sep, 07, 2020

Introduction

In this project, the KNN algorithm will be used to build an algorithm to predict whether heart disease is present in patients based of a few classifiers or features about each patient. The KNN algorithm is a strong, effecient algorithm to class data points based off features in the dataset. It uses a training dataset to learn how to classify the data, then applies those classes to a test dataset to ensure the accuracy of "unkown" datapoints. This training/test data allows for creation of the algorithm to then be applied to data without the classes to actually predict the class of data. The data for this was taken from UC Irvine using a semi cleaned dataset to reduce the features down to 14. Some of these include age, sex, and cholesterol levels.

1: Load libraries and data

The first step in analysis is to load the data and make sure it is easily readable. To do this, the data is first loaded, and then the columns are changed from numeric values to the correlating column names from the UCI website. Ones the names are changed, a summary can be provided to get an initial understanding of the data.

```r
library(class)
library(gmodels)
library(caret)
```

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

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

```r
library(DataExplorer)
library(data.table)
library(e1071)

heart_data <- fread('https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.c

col_names <- c('age','sex','cp','trestbps', 'chol',
          'fbs', 'restecg', 'thalach', 'exang',
          'oldpeak', 'slope', 'ca', 'thal', 'num'
          )
names(heart_data) <- col_names
set.seed(476)

summary(heart_data)
```

```
##       age             sex               cp           trestbps
##  Min.   :29.00   Min.   :0.0000   Min.   :1.000   Min.   : 94.0
##  1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:120.0
```

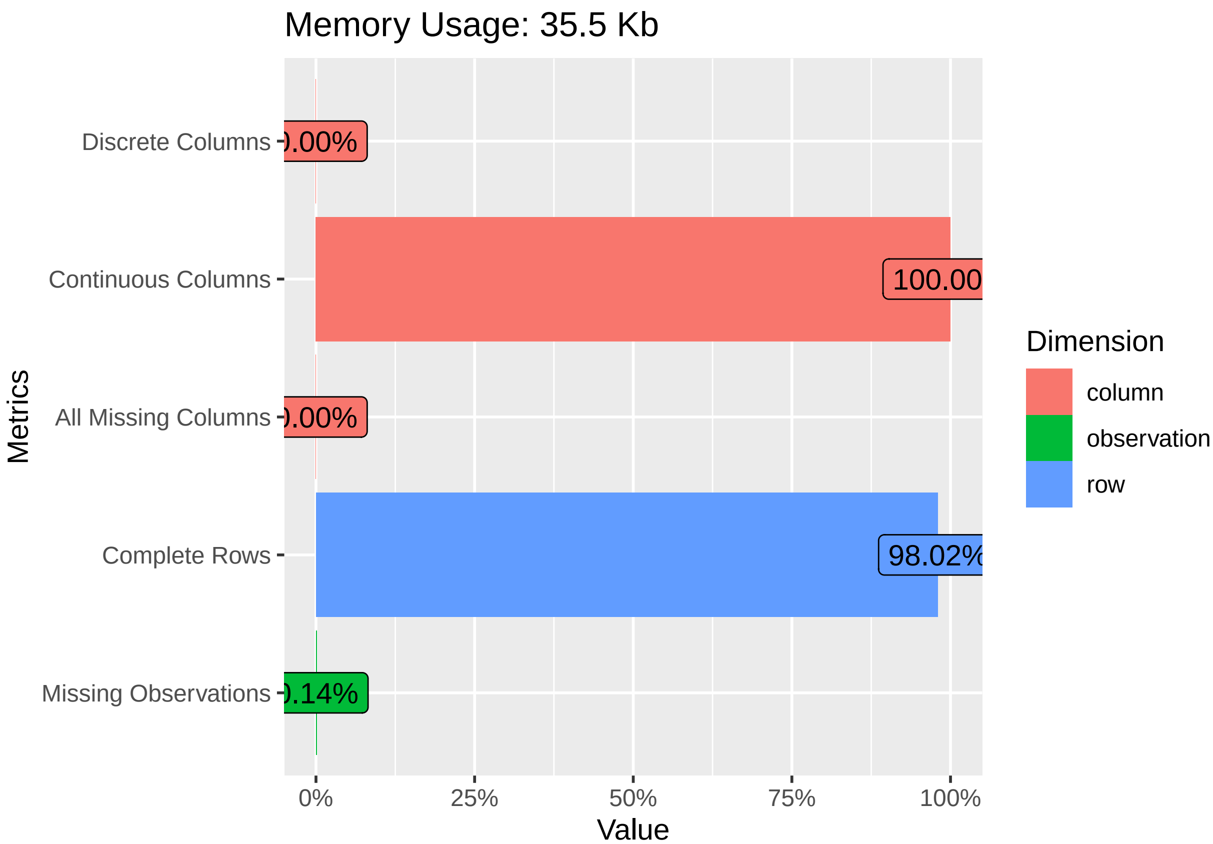
```
##  Median :56.00   Median :1.0000   Median :3.000   Median :130.0
##  Mean   :54.44   Mean   :0.6799   Mean   :3.158   Mean   :131.7
##  3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0
##  Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0
##       chol            fbs            restecg          thalach
##  Min.   :126.0   Min.   :0.0000   Min.   :0.0000   Min.   : 71.0
##  1st Qu.:211.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.5
##  Median :241.0   Median :0.0000   Median :1.0000   Median :153.0
##  Mean   :246.7   Mean   :0.1485   Mean   :0.9901   Mean   :149.6
##  3rd Qu.:275.0   3rd Qu.:0.0000   3rd Qu.:2.0000   3rd Qu.:166.0
##  Max.   :564.0   Max.   :1.0000   Max.   :2.0000   Max.   :202.0
##      exang           oldpeak         slope            ca
##  Min.   :0.0000   Min.   :0.00   Min.   :1.000   Length:303
##  1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   Class :character
##  Median :0.0000   Median :0.80   Median :2.000   Mode  :character
##  Mean   :0.3267   Mean   :1.04   Mean   :1.601
##  3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000
##  Max.   :1.0000   Max.   :6.20   Max.   :3.000
##      thal                num
##  Length:303         Min.   :0.0000
##  Class :character   1st Qu.:0.0000
##  Mode  :character   Median :0.0000
##                     Mean   :0.9373
##                     3rd Qu.:2.0000
##                     Max.   :4.0000
```

From the above, two columns look to be problematic. Both ca and thal are listed as characters, which does not seem to to line up with their column descriptions.

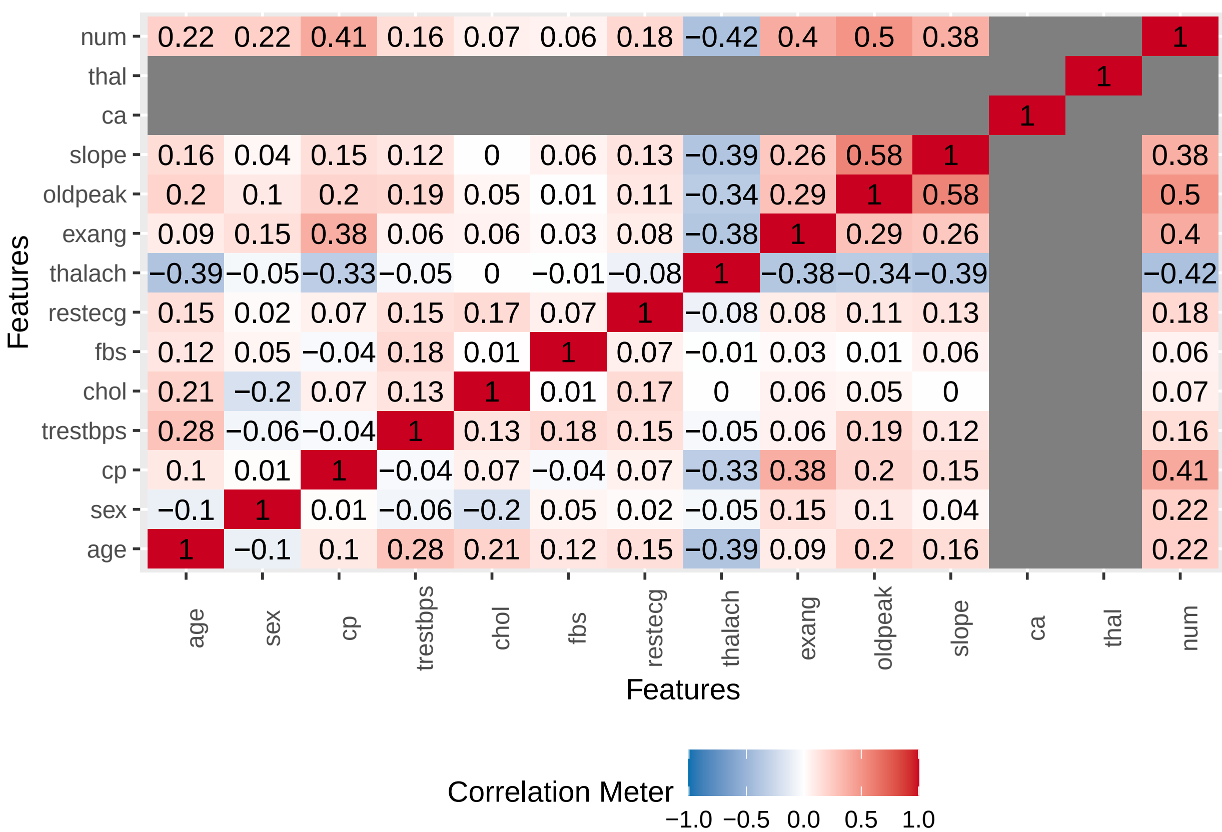
2: Explore and Clean Data

Now that the data has been loaded, the first step is to convert the columns listed above to numeric columns, then use the DataExplorer package to build some basic visuals about the data, provide a summary with the corrected columns, and then based off the plots and summary, determine what data cleaning may be necessary.

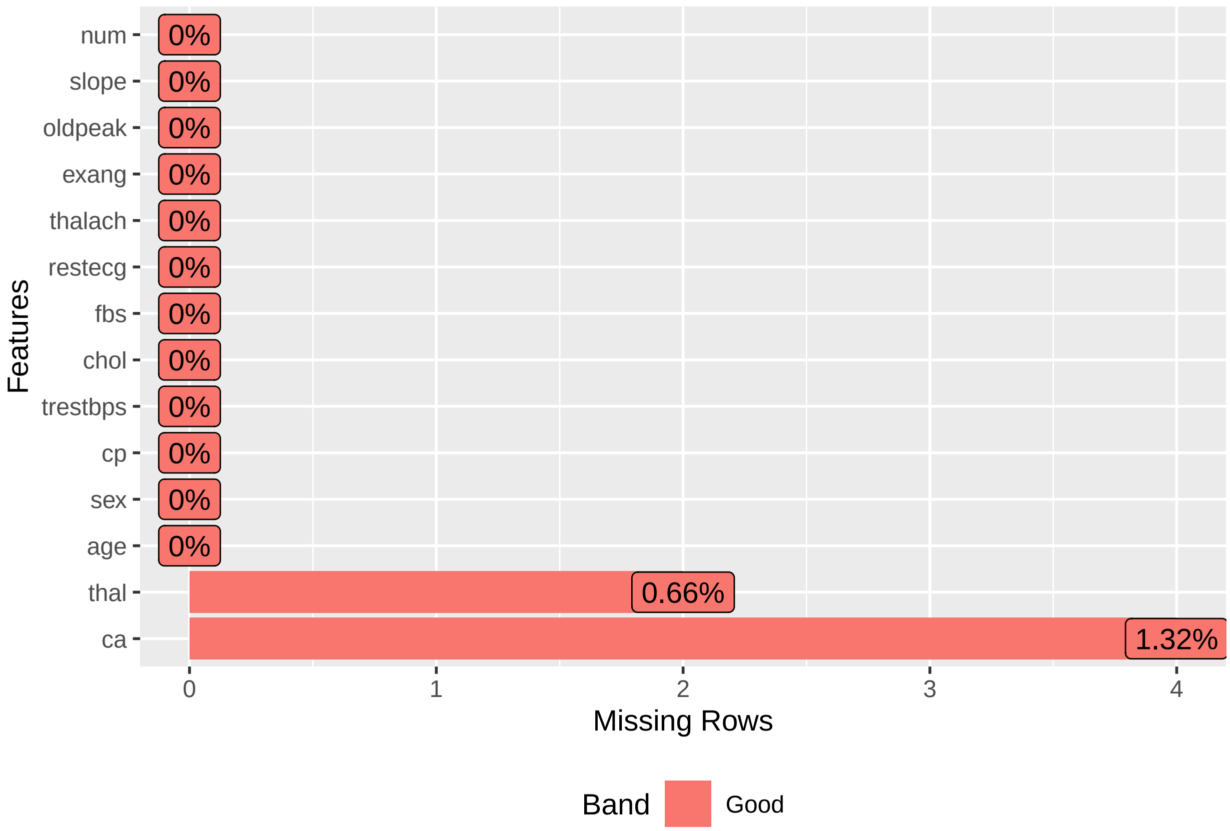
```r
heart_data$ca <- as.numeric(heart_data$ca)
heart_data$thal <- as.numeric(heart_data$thal)
```




```
## Warning: Removed 50 rows containing missing values (geom_text).
```



```
##       age             sex               cp           trestbps    
##  Min.   :29.00   Min.   :0.0000   Min.   :1.000   Min.   : 94.0  
##  1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:120.0  
##  Median :56.00   Median :1.0000   Median :3.000   Median :130.0  
##  Mean   :54.44   Mean   :0.6799   Mean   :3.158   Mean   :131.7  
##  3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0  
##  Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0  
##                                                                  
##       chol             fbs            restecg          thalach     
##  Min.   :126.0   Min.   :0.0000   Min.   :0.0000   Min.   : 71.0  
##  1st Qu.:211.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.5  
##  Median :241.0   Median :0.0000   Median :1.0000   Median :153.0  
##  Mean   :246.7   Mean   :0.1485   Mean   :0.9901   Mean   :149.6  
##  3rd Qu.:275.0   3rd Qu.:0.0000   3rd Qu.:2.0000   3rd Qu.:166.0  
##  Max.   :564.0   Max.   :1.0000   Max.   :2.0000   Max.   :202.0  
##                                                                   
##      exang           oldpeak        slope             ca        
##  Min.   :0.0000   Min.   :0.00   Min.   :1.000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000  
##  Median :0.0000   Median :0.80   Median :2.000   Median :0.0000  
##  Mean   :0.3267   Mean   :1.04   Mean   :1.601   Mean   :0.6722  
##  3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000  
##  Max.   :1.0000   Max.   :6.20   Max.   :3.000   Max.   :3.0000  
##                                                  NA's   :4       
##       thal            num        
##  Min.   :3.000   Min.   :0.0000  
##  1st Qu.:3.000   1st Qu.:0.0000  
##  Median :3.000   Median :0.0000  
##  Mean   :4.734   Mean   :0.9373  
##  3rd Qu.:7.000   3rd Qu.:2.0000  
##  Max.   :7.000   Max.   :4.0000  
##  NA's   :2                       
```




```
##     age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
##  1:  63   1  1      145  233   1       2     150     0     2.3     3  0    6
##  2:  67   1  4      160  286   0       2     108     1     1.5     2  3    3
##  3:  67   1  4      120  229   0       2     129     1     2.6     2  2    7
##  4:  37   1  3      130  250   0       0     187     0     3.5     3  0    3
##  5:  41   0  2      130  204   0       2     172     0     1.4     1  0    3
##  6:  56   1  2      120  236   0       0     178     0     0.8     1  0    3
##  7:  62   0  4      140  268   0       2     160     0     3.6     3  2    3
##  8:  57   0  4      120  354   0       0     163     1     0.6     1  0    3
##  9:  63   1  4      130  254   0       2     147     0     1.4     2  1    7
## 10:  53   1  4      140  203   1       2     155     1     3.1     3  0    7
##     num
##  1:   0
##  2:   2
##  3:   1
##  4:   0
##  5:   0
##  6:   0
##  7:   3
##  8:   0
##  9:   2
## 10:   1
```

Above it can be seen there were two missing values in thal, and 4 in ca. Based of the missing plot the amount of data does not seem significant enough to impute data, so dropping rows with NAs is a reasonable method, and thus is done.

3: Create Factor and Dummy columns, and combine Data

In addition to removing data, it can be seen in the table there are a few columns that need to be scaled to prevent features from being dominate. Additionally, there are a few columns based of descriptions that are identifiers, and thus need to be split into multiple dummy columns. The following scales the columns necessary, and creates factors of the others, allowing them to be converted into dummy variables. Lastly, the scaled, dummy, and remaining variables are all merged back into a cleaned dataset which will be used to apply a knn algorithm.

```
scaled_heart_cols <- as.data.frame(lapply(heart_no_na[,c(1,4,5,8,10)], scale))
factor_heart_cols <- as.data.frame(lapply(heart_no_na[,c(3,7,11:13)],as.factor))
dummy_heart <- dummyVars(~., data=factor_heart_cols,fullRank=TRUE)
dummy_heart_cols <- as.data.frame(predict(dummy_heart, newdata=factor_heart_cols))

heart_no_na$num <- ifelse(heart_no_na$num >= 1,1,0)
heart_no_na$pred <- as.factor(heart_no_na$num)
rest_heart_columns <- heart_no_na[,c(2,6,9,15)]

clean_heart <- cbind(scaled_heart_cols, dummy_heart_cols, rest_heart_columns)
```

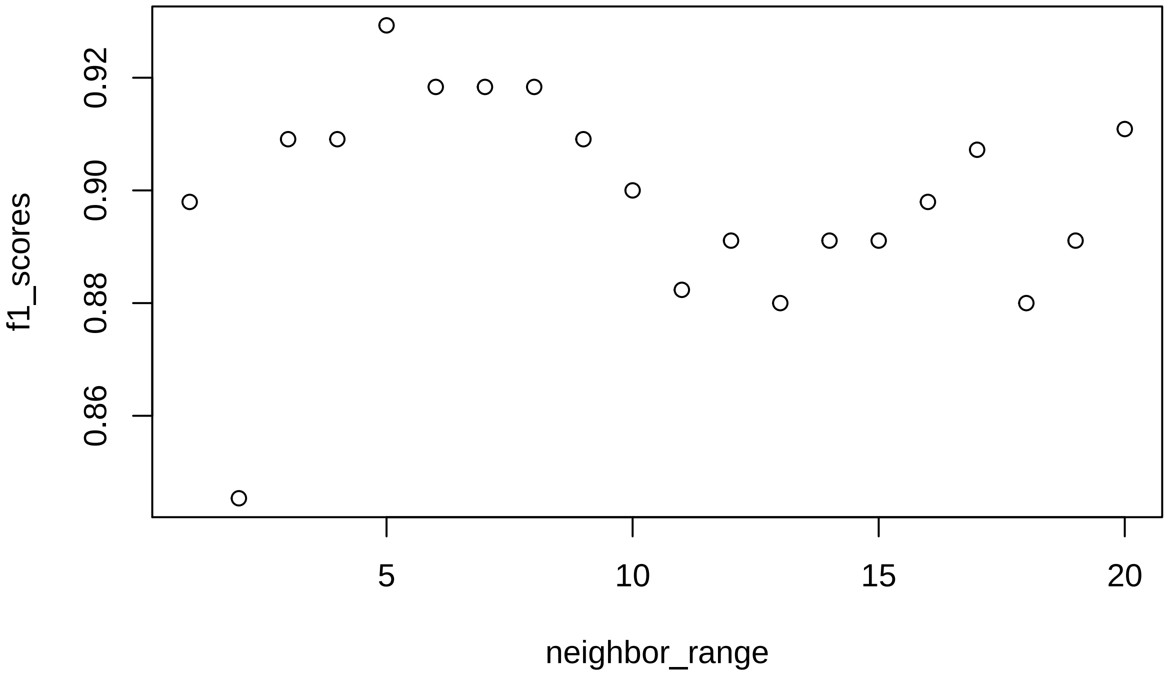
4: Build Function to apply to multiple datasets

The following function allows for knn models with different amount of features to be compared to each other quickly. This function will create the test and training datasets, along with run a loop of the knn algorithm over the number of features in the dataset and return a plot of the f1 scores to allow for selection of the most accurate number of neighbors.

```
knn_range <- function(data) {
  set.seed(476)
  neighbor_range <- c(1:(ncol(data) - 1))
  f1_scores <- list()
  max_f1 = 0
  k_opt = 0
  idx <- createDataPartition(data$pred, p=0.7, list=FALSE)
  heart.train <- (data[idx,])
  heart.test <- (data[-idx,])
  train.labels <- heart.train$pred
  test.labels <- heart.test$pred
  for (i in neighbor_range){
    prediction <- knn(heart.train, heart.test, heart.train$pred, k=i)
    c <- confusionMatrix(prediction, test.labels)
    f1 <- as.numeric(c$byClass['F1'])
    f1_scores[[i]] <- f1
    if (f1>max_f1){
      max_f1 <- f1
      k_opt <- i
    }
  }
return(plot(neighbor_range, f1_scores))
}
```

5: Apply Function to clean_heart

With the function written, it will be applied to the clean heart dataset to determine how many neighbors will be the best prediction.



With the function applied, it can be seen the most accurate option is to apply five neighbors to the data, and the results can be inspected below

```
set.seed(476)
cidx <- createDataPartition(clean_heart$pred, p=0.7, list=FALSE)
cheart.train <- (clean_heart[cidx,])
cheart.test <- (clean_heart[-cidx,])
ctrain.labels <- cheart.train$pred
ctest.labels <- cheart.test$pred
cprediction <- knn(cheart.train, cheart.test, ctrain.labels, k=5) ###replace this value
c <- confusionMatrix(cprediction, ctest.labels)
c
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 46  5
##          1  2 36
##
##                Accuracy : 0.9213
##                  95% CI : (0.8446, 0.9678)
##     No Information Rate : 0.5393
##     P-Value [Acc > NIR] : 3.453e-15
##
##                   Kappa : 0.8409
##
##  Mcnemar's Test P-Value : 0.4497
##
```

```
##              Sensitivity : 0.9583
##              Specificity : 0.8780
##           Pos Pred Value : 0.9020
##           Neg Pred Value : 0.9474
##               Prevalence : 0.5393
##           Detection Rate : 0.5169
##     Detection Prevalence : 0.5730
##        Balanced Accuracy : 0.9182
##
##         'Positive' Class : 0
##
```

```r
as.numeric(c$byClass['F1'])
```

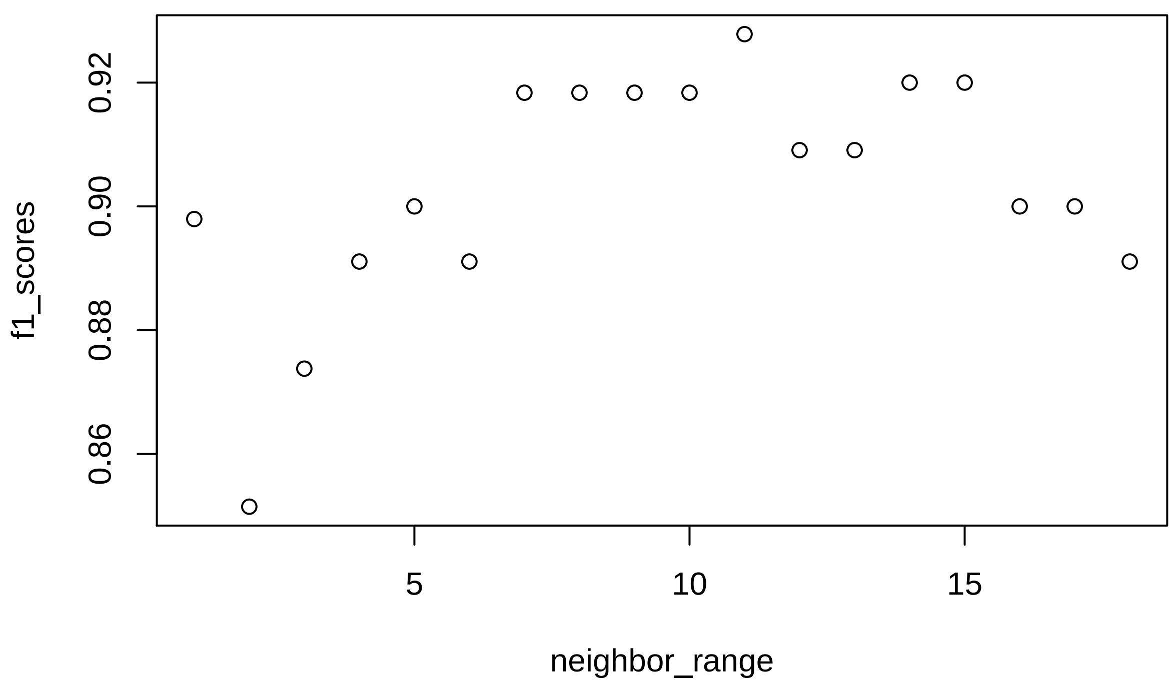
```
## [1] 0.9292929
```

Above it can be seen the overall accuracy of the is 92.13%, and the f1 score which is most commonly used to determine performance is 92.92%.

6: Feature Reduction and compare

With the function written, the correlation matrix can be looked at from the beginning to see there were a few metrics that did not have heavy impact on the prediction column in the dataset. Both the fbs and chol columns have almost no correlation to the prediction column, thus for sake of investigation will be dropped and the knn_range function will be run again.

```r
reduced_heart <- subset(clean_heart, select= -c(fbs,chol))
knn_range(reduced_heart)
```




```r
set.seed(476)
ridx <- createDataPartition(reduced_heart$pred, p=0.7, list=FALSE)
rheart.train <- (reduced_heart[ridx,])
rheart.test <- (reduced_heart[-ridx,])
rtrain.labels <- rheart.train$pred
```

```r
rtest.labels <- rheart.test$pred
rprediction <- knn(rheart.train, rheart.test, rtrain.labels, k=11)
c <-confusionMatrix(rprediction, rtest.labels)
c
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 45  4
##          1  3 37
##
##                Accuracy : 0.9213
##                  95% CI : (0.8446, 0.9678)
##     No Information Rate : 0.5393
##     P-Value [Acc > NIR] : 3.453e-15
##
##                   Kappa : 0.8414
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9375
##             Specificity : 0.9024
##          Pos Pred Value : 0.9184
##          Neg Pred Value : 0.9250
##              Prevalence : 0.5393
##          Detection Rate : 0.5056
##    Detection Prevalence : 0.5506
##       Balanced Accuracy : 0.9200
##
##        'Positive' Class : 0
##
```

```r
as.numeric(c$byClass['F1'])
```

```
## [1] 0.9278351
```

From above, the model with 11 neighbors is the most accurate, and when applied, the overall accuracy is the same at 92.13% and the f1 score is only slightly lower at 92.78%. The difference in accuracy of .14% is not significant, and although these models run extremely fast due to the size of data, on a larger dataset this difference would not be significant enough to justify keeping the features. Thus for this data the best option would be reducing the features of the dataset and applying 11 neighbors to the knn algorithm as shown above.

Conclusion

From the project above, the utility of the knn algorithm can be seen to be effective at predicting the class of datapoints based off features. This example is simple in the fact there are only two classes for the datapoints to be assigned too. Additionally, the function written to output the neighbor range was not an efficient method, as the variables were not stored, and to work the model again with a static knn value required the training and testing split, along with labels to be entered, creating repeated work and code.