MSDS 680 – Machine Learning in R
KNN Algorithm
Alla Topp

Heart Disease Prediction

Objective: to predict heart disease in patients.

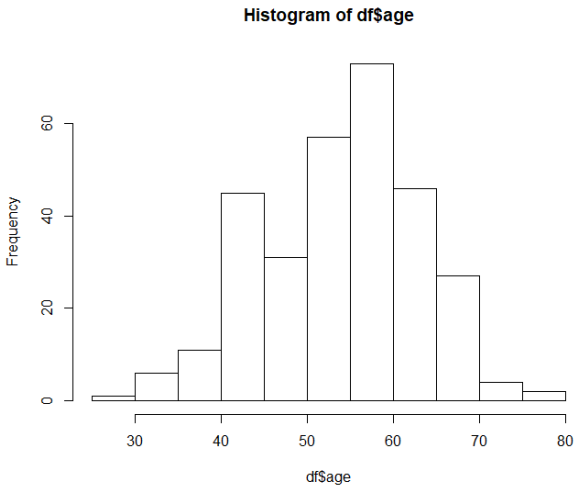
1.1) Data Exploration

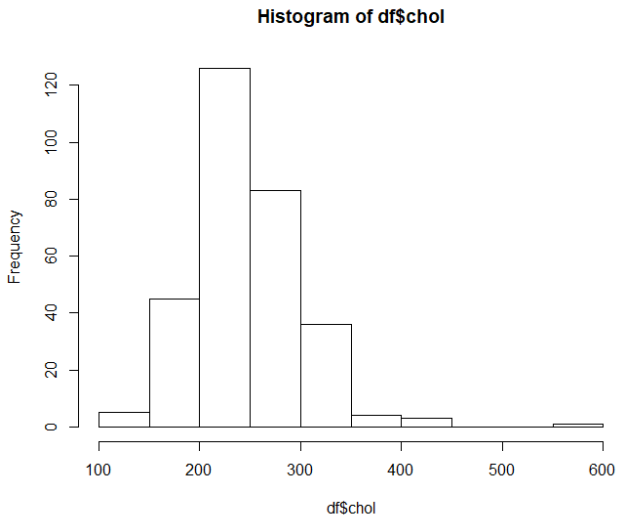
Only 14 attributes used (columns):
1. (age) age in years
2. (sex) (1 = male; 0 = female)
3. (cp) chest pain type
4. (trestbps) resting blood pressure (in mm Hg on admission to the hospital)
5. (chol) serum cholestoral in mg/dl
6. (fbs) (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. (restecg) resting electrocardiographic results
8. (thalach) maximum heart rate achieved
9. (exang) exercise induced angina (1 = yes; 0 = no)
10. (oldpeak) = ST depression induced by exercise relative to rest
11. (slope) the slope of the peak exercise ST segment
12. (ca) number of major vessels (0-3) colored by flourosopy
13. (thal) 3 = normal; 6 = fixed defect; 7 = reversable defect
14. (num) (the predicted attribute) diagnosis of heart disease (angiographic disease status)

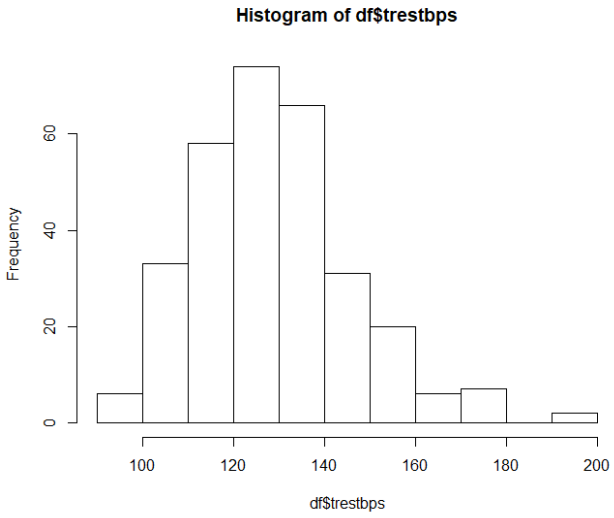
We start our analysis with exploration of the data. We can see the summary of the table we have below:

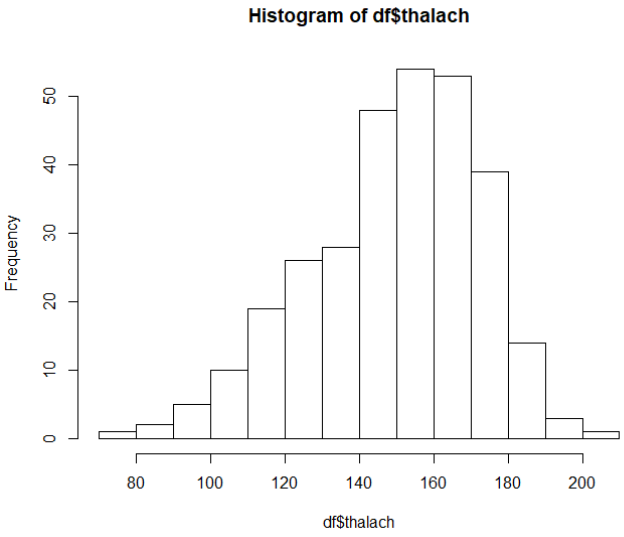
```
> summary(df)
      age             sex               cp          trestbps         chol            fbs        
 Min.   :29.00   Min.   :0.0000   Min.   :1.000   Min.   : 94.0   Min.   :126.0   Min.   :0.0000  
 1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:120.0   1st Qu.:211.0   1st Qu.:0.0000  
 Median :56.00   Median :1.0000   Median :3.000   Median :130.0   Median :241.0   Median :0.0000  
 Mean   :54.44   Mean   :0.6799   Mean   :3.158   Mean   :131.7   Mean   :246.7   Mean   :0.1485  
 3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0   3rd Qu.:275.0   3rd Qu.:0.0000  
 Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0   Max.   :564.0   Max.   :1.0000  

    restecg          thalach          exang           oldpeak          slope             ca        
 Min.   :0.0000   Min.   : 71.0   Min.   :0.0000   Min.   :0.00   Min.   :1.000   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.:133.5   1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000  
 Median :1.0000   Median :153.0   Median :0.0000   Median :0.80   Median :2.000   Median :0.0000  
 Mean   :0.9901   Mean   :149.6   Mean   :0.3267   Mean   :1.04   Mean   :1.601   Mean   :0.6722  
 3rd Qu.:2.0000   3rd Qu.:166.0   3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000  
 Max.   :2.0000   Max.   :202.0   Max.   :1.0000   Max.   :6.20   Max.   :3.000   Max.   :3.0000  
                                                                                  NA's   :4       

      thal            pred       
 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
```

Some of the column can show some numbers visually. I built histogram plots of four columns where we could see the age of patients, serum cholesteral, resting blood pressure, maximum heart rate achieved. Below we can see that age of patients mostly between 55 and 60, also we can see that resting blood pressure is 120-130 in most cases and maximum heart rate achieved at 150-160.









1.2) Divide the data into a training set and a test set randomly with ratio 70:30. Make the prediction based on 1-nearest neighbor. What is the error rate of this approach? Report the confusion matrix and accuracy. Interpret the results.

At this part we will identify which rows and columns have missing values, we can see it after running function view() and we can see that column 12 and column 14 together have 6 missing values. After that we remove the rows with missing values with function df.omit().

```
> df.omit <- na.omit(df) # assign result to a new object
> summary(df.omit) # there are no NA's (in new object)
      age             sex               cp           trestbps          chol            fbs        
 Min.   :29.00   Min.   :0.0000   Min.   :1.000   Min.   : 94.0   Min.   :126.0   Min.   :0.0000  
 1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:120.0   1st Qu.:211.0   1st Qu.:0.0000  
 Median :56.00   Median :1.0000   Median :3.000   Median :130.0   Median :243.0   Median :0.0000  
 Mean   :54.54   Mean   :0.6768   Mean   :3.158   Mean   :131.7   Mean   :247.4   Mean   :0.1448  
 3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0   3rd Qu.:276.0   3rd Qu.:0.0000  
 Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0   Max.   :564.0   Max.   :1.0000  
    restecg          thalach         exang           oldpeak         slope             ca        
 Min.   :0.0000   Min.   : 71.0   Min.   :0.0000   Min.   :0.000   Min.   :1.000   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.:133.0   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:0.0000  
 Median :1.0000   Median :153.0   Median :0.0000   Median :0.800   Median :2.000   Median :0.0000  
 Mean   :0.9966   Mean   :149.6   Mean   :0.3266   Mean   :1.056   Mean   :1.603   Mean   :0.6768  
 3rd Qu.:2.0000   3rd Qu.:166.0   3rd Qu.:1.0000   3rd Qu.:1.600   3rd Qu.:2.000   3rd Qu.:1.0000  
 Max.   :2.0000   Max.   :202.0   Max.   :1.0000   Max.   :6.200   Max.   :3.000   Max.   :3.0000  
      thal            pred       
 Min.   :3.000   Min.   :0.0000  
 1st Qu.:3.000   1st Qu.:0.0000  
 Median :3.000   Median :0.0000  
 Mean   :4.731   Mean   :0.9461  
 3rd Qu.:7.000   3rd Qu.:2.0000  
 Max.   :7.000   Max.   :4.0000  
```

To make our data more precise without overfitting we should normalize this dataset and to rescale the features to a standard range of values. Now data range is between 0 and 1.

```
#Normalize/standardize data
normalize <- function(x) {return((x-min(x))/(max(x)-min(x)))}
df_norm <- as.data.frame(lapply(df.omit[,c(1,4,5,8,10)],normalize)) #normalized columns 1,4,5,8,10
View(df_norm)
```

	age	trestbps	chol	thalach	oldpeak
1	0.7083333	0.48113208	0.24429224	0.6030534	0.37096774
2	0.7916667	0.62264151	0.36529680	0.2824427	0.24193548
3	0.7916667	0.24528302	0.23515982	0.4427481	0.41935484
4	0.1666667	0.33962264	0.28310502	0.8854962	0.56451613
5	0.2500000	0.33962264	0.17808219	0.7709924	0.22580645
6	0.5625000	0.24528302	0.25114155	0.8167939	0.12903226
7	0.6875000	0.43396226	0.32420091	0.6793893	0.58064516
8	0.5833333	0.24528302	0.52054795	0.7022901	0.09677419
9	0.7083333	0.33962264	0.29223744	0.5801527	0.22580645
10	0.5000000	0.43396226	0.17579909	0.6412214	0.50000000
11	0.5833333	0.43396226	0.15068493	0.5877863	0.06451613
12	0.5625000	0.43396226	0.38356164	0.6259542	0.20967742
13	0.5625000	0.33962264	0.29680365	0.5419847	0.09677419
14	0.3125000	0.24528302	0.31278539	0.7786260	0.00000000
15	0.4791667	0.73584906	0.16666667	0.6946565	0.08064516

Showing 1 to 15 of 297 entries

As seen above, cholestoral which used to have a min of 126 and a max of 564, now has values ranging between 0 and 1. The same with age, trestbps, thalach and oldpeak.

```
#Convert to dummy variables
library(caret)
cat_vars <- as.data.frame(lapply(df.omit[,c(3,7,11:13)],as.factor)) # takes a list and
dummy <- dummyVars(~.,data=cat_vars,fullRank = TRUE)
df_dummy <- as.data.frame(predict(dummy,newdata=cat_vars))
View(df_dummy)
```

KNN.R* | df_dummy | df_norm | df

Filter

	cp.2	cp.3	cp.4	restecg.1	restecg.2	slope.2	slope.3	ca.1	ca.2	ca.3	thal.6	thal.7
1	0	0	0	0	1	0	1	0	0	0	1	0
2	0	0	1	0	1	1	0	0	0	1	0	0
3	0	0	1	0	1	1	0	0	1	0	0	1
4	0	1	0	0	0	0	1	0	0	0	0	0
5	1	0	0	0	1	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	1	0	1	0	1	0	1	0	0	0
8	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	1	0	1	1	0	1	0	0	0	1
10	0	0	1	0	1	0	1	0	0	0	0	1
11	0	0	1	0	0	1	0	0	0	0	1	0
12	1	0	0	0	1	1	0	0	0	0	0	0
13	0	1	0	0	1	1	0	1	0	0	1	0
14	1	0	0	0	0	0	0	0	0	0	0	1
15	0	1	0	0	0	0	0	0	0	0	0	1

```
> #Aggregate dependent
> df.omit$pred <- with(df.omit, ifelse(df.omit$pred >=1, 1, 0)) #change pred to 0 or 1
> df.omit$pred <- as.factor(df.omit$pred) #change predicted variable to factor (for classification probl
em)
> df.omit$pred
  [1] 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1
 [49] 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 1
 [97] 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0
[145] 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0
[193] 1 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0
[241] 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 0 1 0 1
[289] 1 1 0 1 1 1 1 1 1
Levels: 0 1
```

```
> #Combine variable back
> df_all <- cbind(df.omit$ca,df.omit$fbs,df_norm,df_dummy,df.omit$pred) # combine features with cbind()
> df_all
   df.omit$ca df.omit$fbs       age   trestbps       chol   thalach    oldpeak cp.2 cp.3 cp.4
1           0           1 0.7083333 0.48113208 0.24429224 0.6030534 0.37096774    0    0    0
2           3           0 0.7916667 0.62264151 0.36529680 0.2824427 0.24193548    0    0    1
3           2           0 0.7916667 0.24528302 0.23515982 0.4427481 0.41935484    0    0    1
4           0           0 0.1666667 0.33962264 0.28310502 0.8854962 0.56451613    0    1    0
5           0           0 0.2500000 0.33962264 0.17808219 0.7709924 0.22580645    1    0    0
6           0           0 0.5625000 0.24528302 0.25114155 0.8167939 0.12903226    1    0    0
7           2           0 0.6875000 0.43396226 0.32420091 0.6793893 0.58064516    0    0    1
8           0           0 0.5833333 0.24528302 0.52054795 0.7022901 0.09677419    0    0    1
9           1           0 0.7083333 0.33962264 0.29223744 0.5801527 0.22580645    0    0    1
10          0           1 0.5000000 0.43396226 0.17579909 0.6412214 0.50000000    0    0    1
11          0           0 0.5833333 0.43396226 0.15068493 0.5877863 0.06451613    0    0    1
12          0           0 0.5625000 0.43396226 0.38356164 0.6259542 0.20967742    1    0    0
13          1           1 0.5625000 0.33962264 0.29680365 0.5419847 0.09677419    0    1    0
14          0           0 0.3125000 0.24528302 0.31278539 0.7786260 0.00000000    1    0    0
15          0           1 0.4791667 0.73584906 0.16666667 0.6946565 0.08064516    0    1    0
16          0           0 0.5833333 0.52830189 0.09589041 0.7862595 0.25806452    0    1    0
17          0           0 0.3958333 0.15094340 0.23515982 0.7404580 0.16129032    1    0    0
18          0           0 0.5208333 0.43396226 0.25799087 0.6793893 0.19354839    0    0    1
19          0           0 0.3958333 0.33962264 0.34018265 0.5190840 0.03225806    0    1    0
20          0           0 0.4166667 0.33962264 0.31963470 0.7633588 0.09677419    1    0    0
```

Before we run KNN algorithm we should split data into training and test sets to the 70/30.

```
> #Split data into training and test set
> set.seed(555)
> idx <- sample(2, nrow(df.omit), replace=TRUE, prob=c(0.7, 0.3))  #create 2 samples with ratio 70:30
> idx
  [1] 1 2 1 1 1 1 2 1 1 2 2 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1
 [49] 1 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 2 2 1 1 1 2 1 1 1 2 1 2
 [97] 2 1 1 2 2 2 1 2 1 1 1 2 1 1 1 2 2 1 1 2 1 2 1 1 2 2 1 1 1 2 1 2 1 1 1 1 1 2 2 2 1 1 2 1 2 1 1
[145] 2 2 1 2 1 2 1 1 1 2 1 1 2 1 1 2 1 1 2 2 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1
[193] 2 1 1 2 1 2 2 1 1 1 1 2 1 2 2 1 2 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 2 2 1 1 1
[241] 1 1 1 1 2 1 2 1 2 1 1 1 1 2 2 2 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 1 1 2 1 1 1 2 2 1 1 1 1 1 1 1 1
[289] 1 2 1 1 2 1 2 1 1
> df.train <- df.omit[idx==1, ] #sample 1 for training set
> df.test <- df.omit[idx==2,] # sample 2 for test set
```

Now we have them and we can run KNN with different values of K, so we can identify the most optimal value and accurate result.

1.3) Use different values for K, what is the optimal value of K from your experiments?

First, we try to run the algorithm **with K=1**, the we evaluate the model by building Cross Table with true positive and negative results and false positive and negative results.

```
> #Build the KNN model / evaluate the model
> library(class)
> knn_pred <- knn(train=df.train, test=df.test,cl=df.train$pred,k=1) #change to your object names and vary
 k value
> knn_pred
 [1] 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 1 0 0 0 1 0
[49] 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1
Levels: 0 1
```

```
#Evaluate the model
install.packages("gmodels")
library(gmodels)
CrossTable(x=df.test$pred ,y= knn_pred, prop.chisq=FALSE)
```

```
Total Observations in Table:  92


             | knn_pred
df.test$pred |         0 |         1 | Row Total |
-------------|-----------|-----------|-----------|
           0 |        33 |        17 |        50 |
             |     0.660 |     0.340 |     0.543 |
             |     0.589 |     0.472 |           |
             |     0.359 |     0.185 |           |
-------------|-----------|-----------|-----------|
           1 |        23 |        19 |        42 |
             |     0.548 |     0.452 |     0.457 |
             |     0.411 |     0.528 |           |
             |     0.250 |     0.207 |           |
-------------|-----------|-----------|-----------|
Column Total |        56 |        36 |        92 |
             |     0.609 |     0.391 |           |
-------------|-----------|-----------|-----------|
```

The accuracy rate = (33+19) /(33+17+23+19) =0.57 or 57% based on the confusion matrix.

In the above table we can see that 33% of patients don't have heart disease (true negative) and that 19% have heart disease (true positive).

As seen in the table, 77/100 cases were accurately predicted total of 23% were incorrectly classified. This is a good start, but the model must improve if it is to be really used to diagnose patients. Mistakes in this domain are extremely consequential. We are going to try to improve the model by making the k value 5.

If K=3

```
             | knn_pred
df.test$pred |         0 |         1 | Row Total |
-------------|-----------|-----------|-----------|
           0 |        33 |        17 |        50 |
             |     0.660 |     0.340 |     0.543 |
             |     0.647 |     0.415 |           |
             |     0.359 |     0.185 |           |
-------------|-----------|-----------|-----------|
           1 |        18 |        24 |        42 |
             |     0.429 |     0.571 |     0.457 |
             |     0.353 |     0.585 |           |
             |     0.196 |     0.261 |           |
-------------|-----------|-----------|-----------|
Column Total |        51 |        41 |        92 |
             |     0.554 |     0.446 |           |
-------------|-----------|-----------|-----------|
```

The accuracy rate = (33+24) / (33+17+18+24) = 0.62(62%)

If k=5

df.test$pred	knn_pred 0	1	Row Total
0	34	16	50
	0.680	0.320	0.543
	0.642	0.410	
	0.370	0.174	
1	19	23	42
	0.452	0.548	0.457
	0.358	0.590	
	0.207	0.250	
Column Total	53	39	92
	0.576	0.424	

The accuracy rate = (34+23)/ (34+16+19+23) = 0.62 (62%)

If k=15

df.test$pred	knn_pred 0	1	Row Total
0	32	18	50
	0.640	0.360	0.543
	0.653	0.419	
	0.348	0.196	
1	17	25	42
	0.405	0.595	0.457
	0.347	0.581	
	0.185	0.272	
Column Total	49	43	92
	0.533	0.467	

The accuracy rate = (32+25) / (32+18+17+25) = 0.62 (62%)

If k=22

df.test$pred	knn_pred 0	1	Row Total
0	34	16	50
	0.680	0.320	0.543
	0.739	0.348	
	0.370	0.174	
1	12	30	42
	0.286	0.714	0.457
	0.261	0.652	
	0.130	0.326	
Column Total	46	46	92
	0.500	0.500	

The accuracy rate = (34+30) / (34+16+12+30) = 0.70 (70%)

K value	True positive	True Negative	False negative	False Positive	Percent classified incorrect
1	33	19	23	17	23
3	33	24	18	17	18
5	34	23	19	16	19
15	32	25	17	18	17
22	**34**	**30**	**12**	**16**	**12**

We can see that the most optimal result of the algorithm with k =22, the accuracy rate is 70% based on the measurement the performance of classification method. In order to calculate the result, we needed to follow the following formula:

Accuracy = (True positive + true negative) / (true positive = true negative +false positive + false negative)

We can state that for this problem we identified that accuracy rate of the prediction of the heart rate is 70%. It's still not accurate enough to use to diagnose heart disease but might be good enough for the assignment! A better idea for this problem is to use a larger training set - more data would certainly help. Although, KNN might not be the optimal machine learning method for this problem.