Samet Yazak
1604262

UNIVERSITY OF BAHÇEŞEHİR

Department of Computer Engineering

CMP5133 Artficial Neural Networks – Assignment 2

1. Abstract

The subject of the assigment is to implement RBF network using two dimensional data, one for input, one for output. To implement RBF network, it is required to use hybrid lerning; K-means clustering algorithm will be implemented for unsupervised part, Single Layer Perceptron for the supervised part.

2. Implementation

I implemented the RBF network using R Studio. You can find the source code (Section 5) at the end of the document. To run the code successfully, you need to set input data directory as working directory. Source file and data should be in same folder.

Before implementing network, I didn't apply any preprocessing step.

For K-means, first I initiated centroids randomly. Then, I iterated process until no input data is assigned to different cluster (convergence). In case it doesn't converge, I limited iteration count to 10. I my tests, it always converged in less than 10 iterations. Each K-means iteration 3 steps:

- Assign input values to clusters by finding the closest one to input
- Re-calculate centroid values
- If any input changed cluster, go step 1.

During K-means clustering, I also calculated the most distant inputs in clusters to calculate spread after it converges.

For Single Perceptron step, I first calculated perceptive field values, using centroid and spread values which are outputs of K-means process. For each K value (from 1 to 10), I applied basic single perceptron algorithm using epoch as 1000 and learning rate as 0.2 whose seem to be enough for every K value.

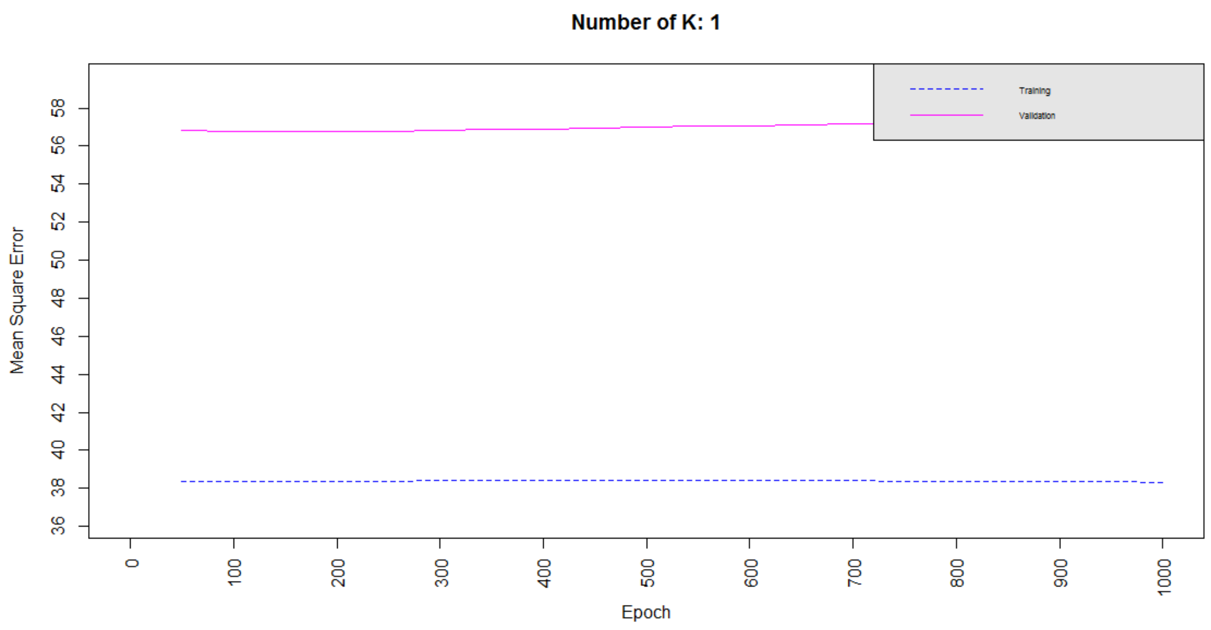
Results of RBF network are plotted and listed in the following section.

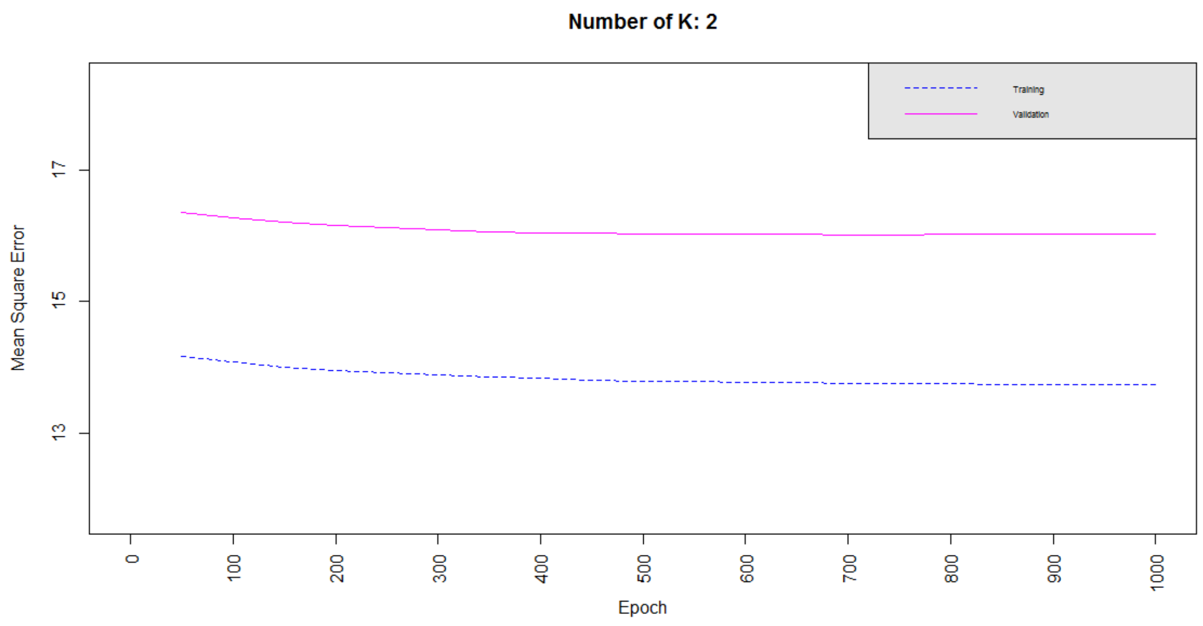
3. Plots

In this section, there are 3 types of plots; training and validation mean squared errors, training input and RBF network outputs, training data and network output together.

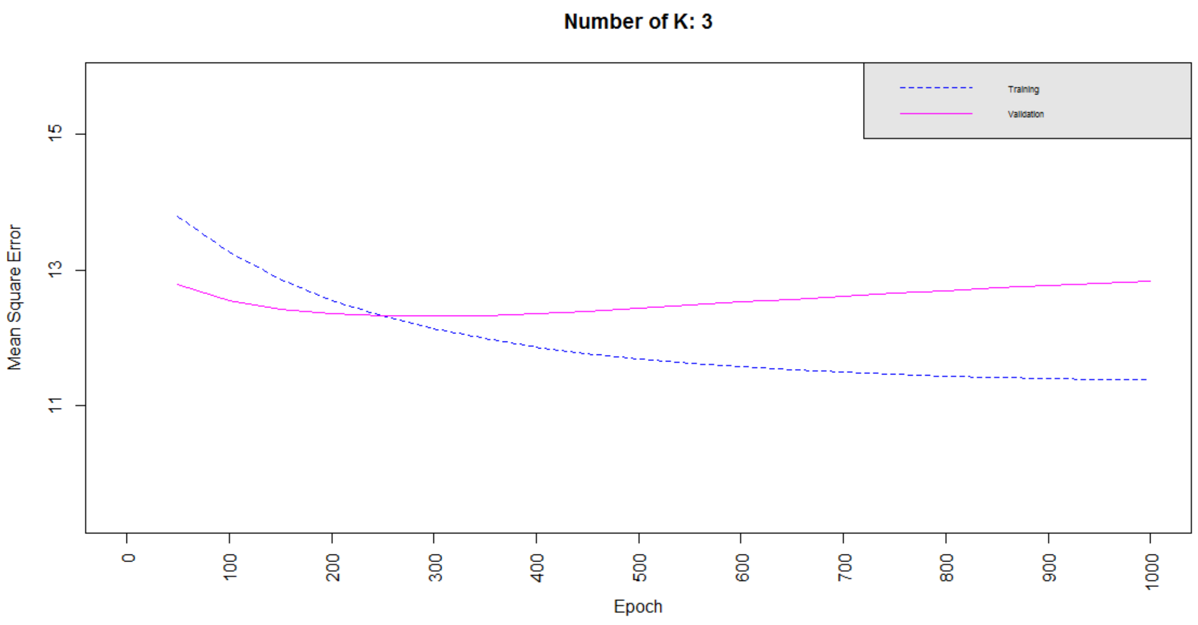
Samet Yazak
1604262

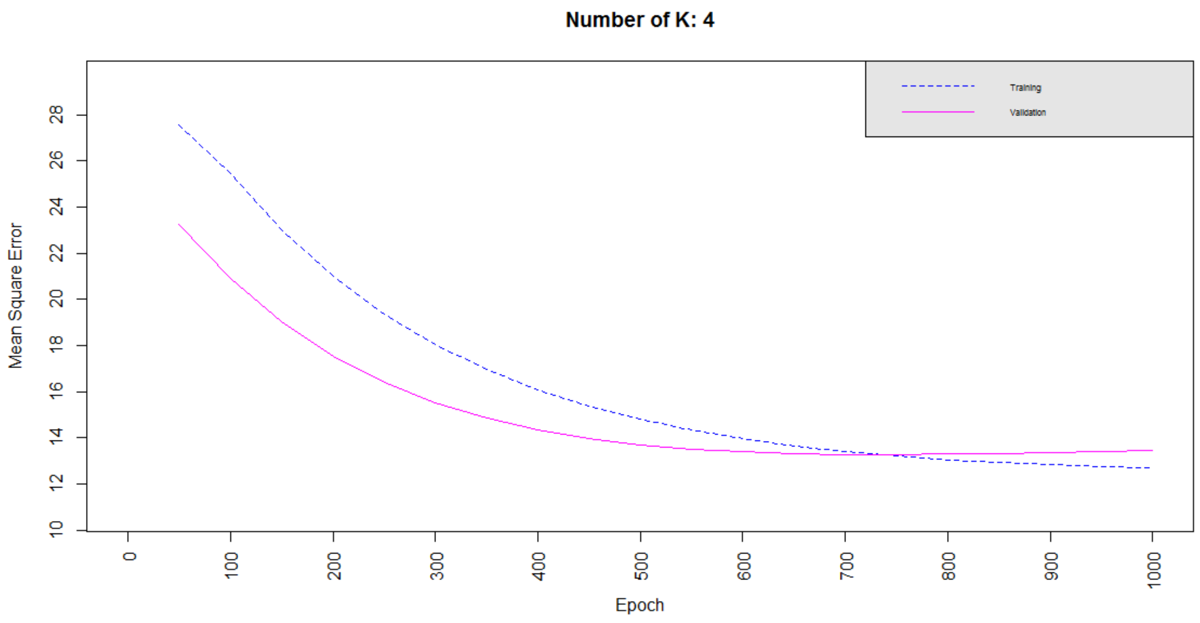
3.1 Training and Validation Mean Squared Errors



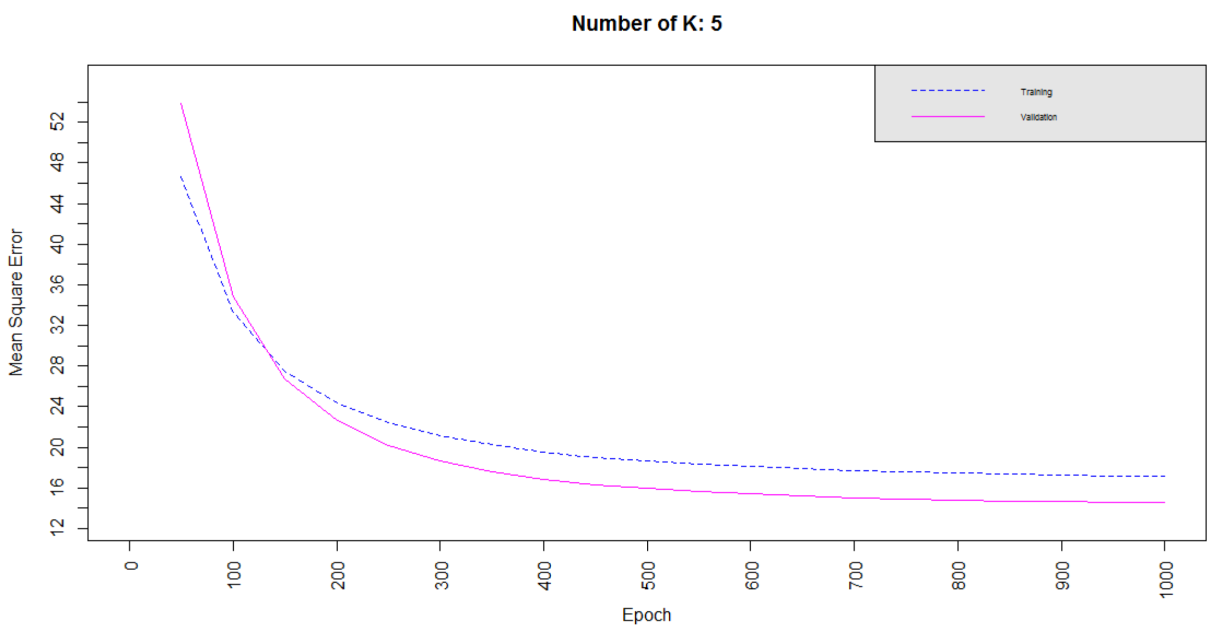


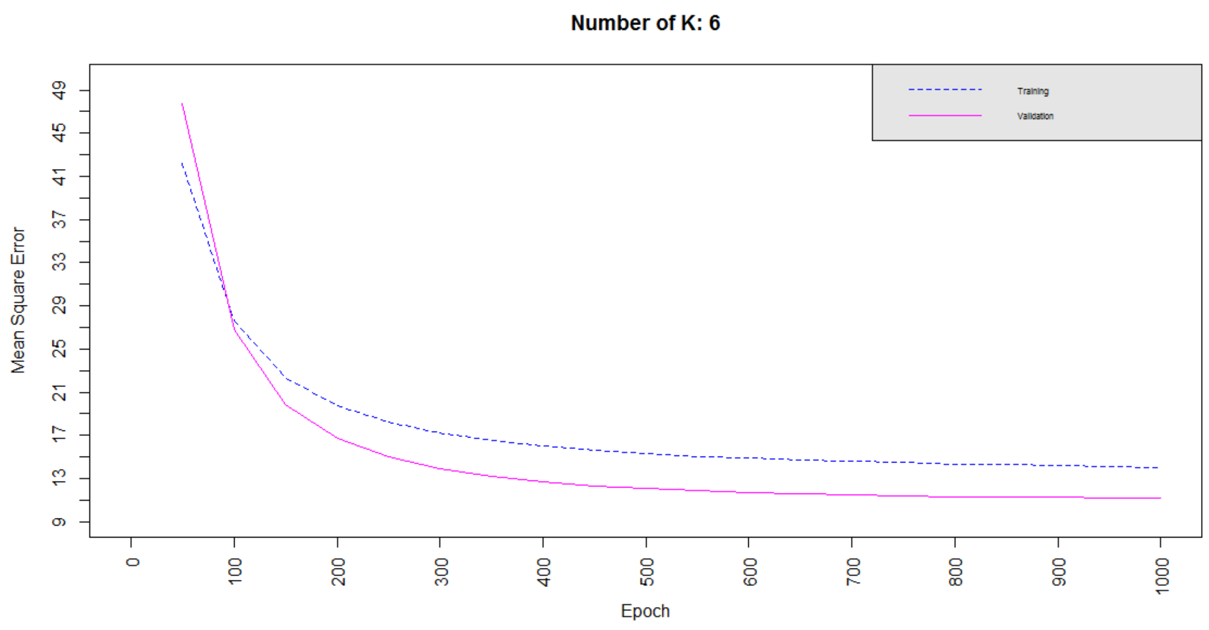
Samet Yazak
1604262



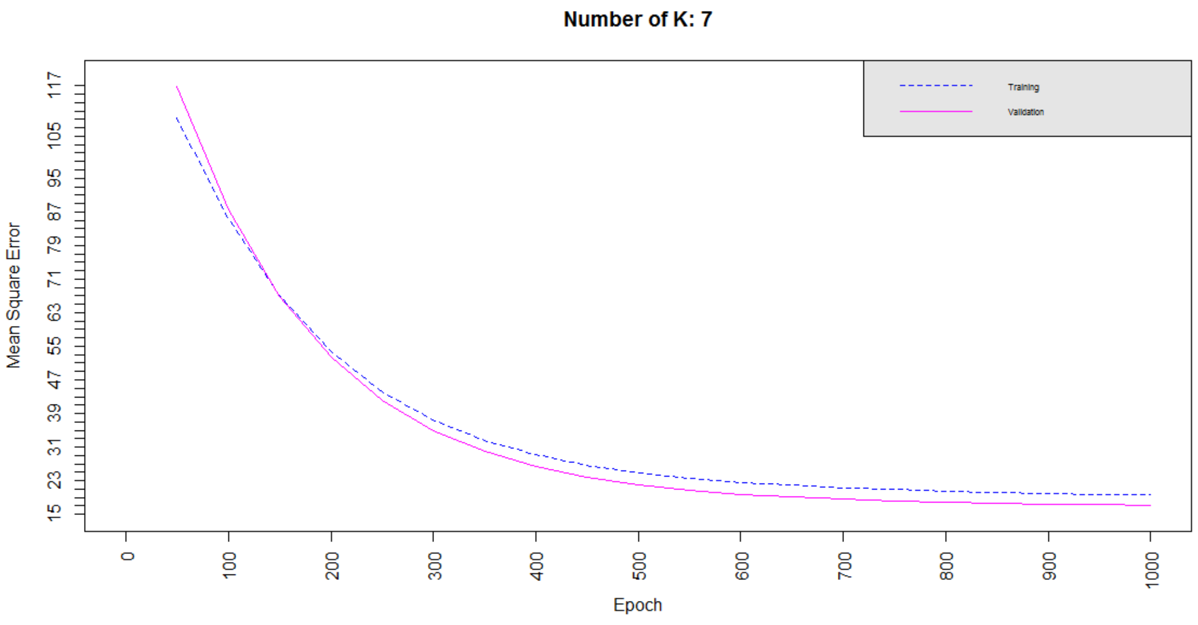


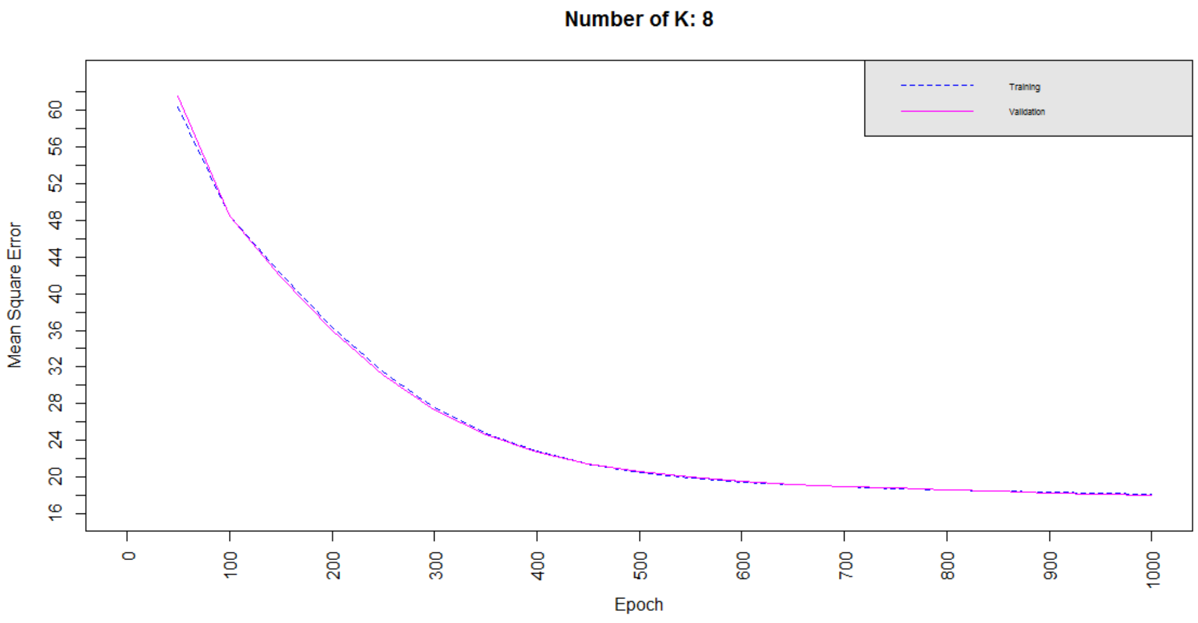
Samet Yazak
1604262





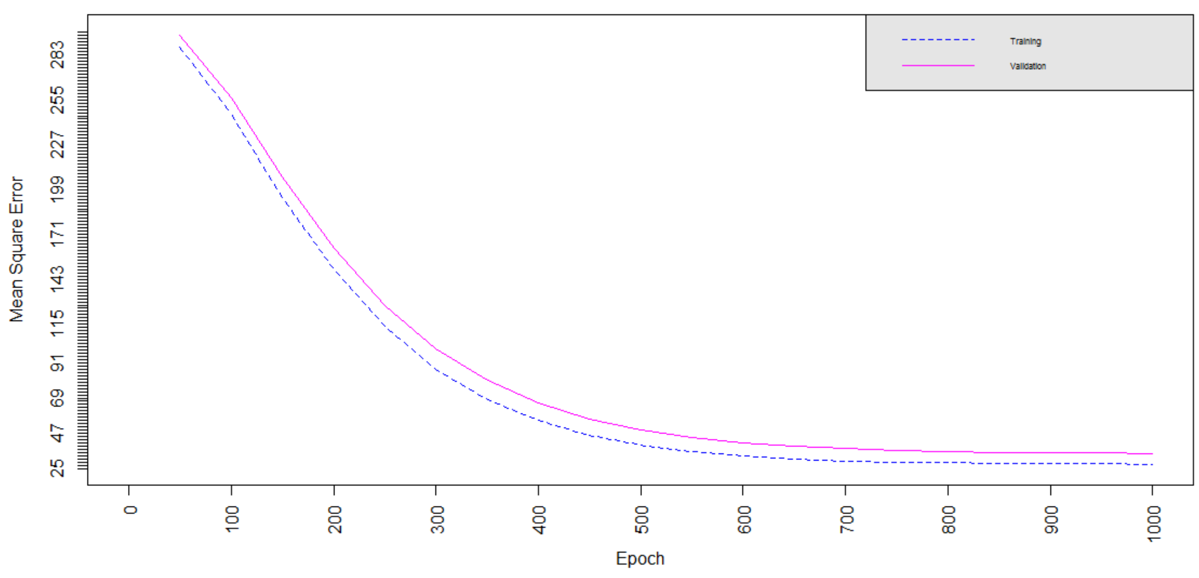
Samet Yazak
1604262



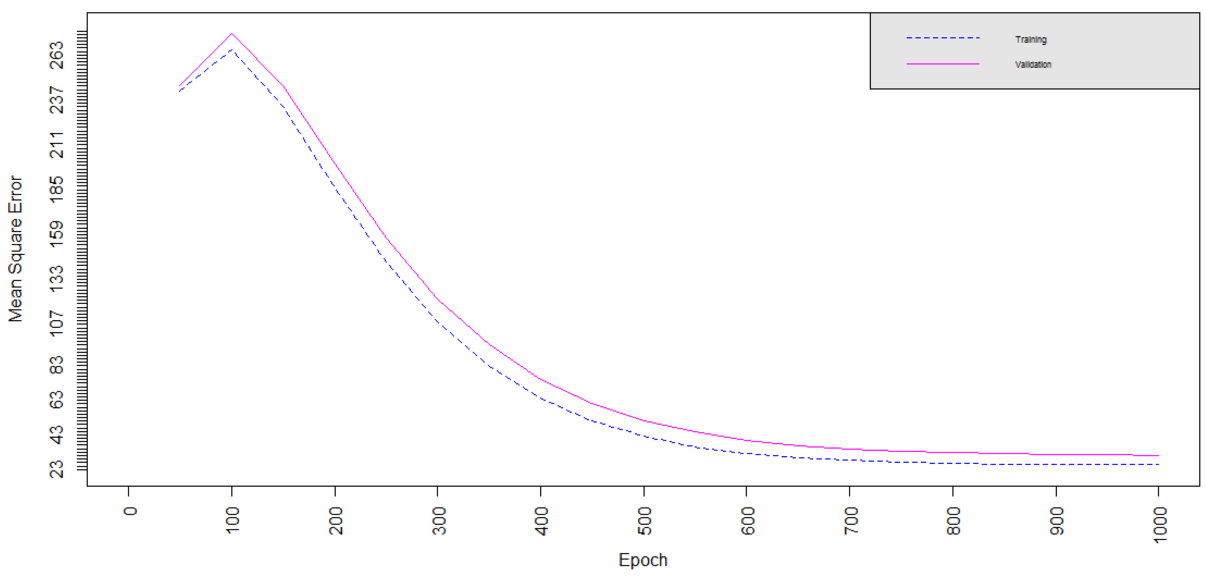


Samet Yazak
1604262

Number of K: 9



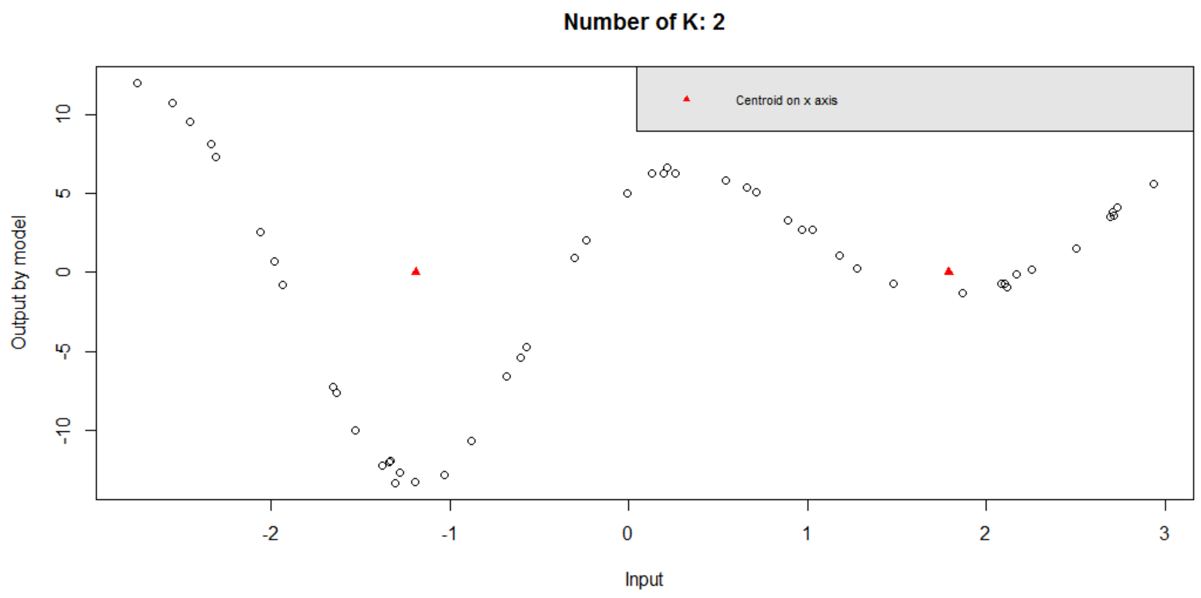
Number of K: 10



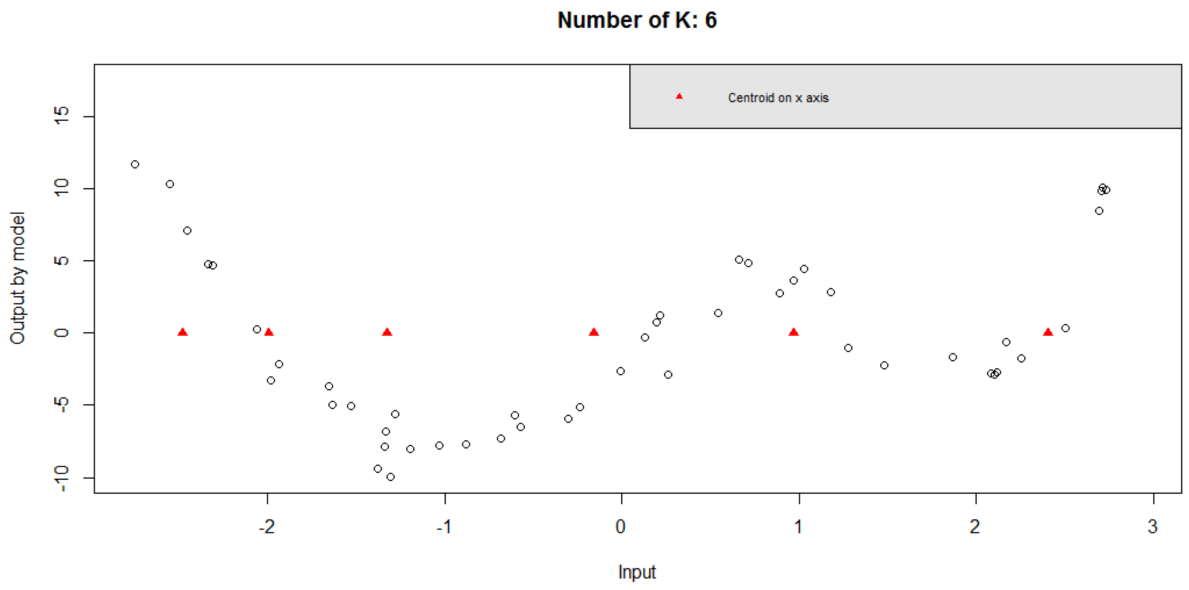
3.2 Training Input and RBF Network Output

A. For underfitting

Samet Yazak
1604262

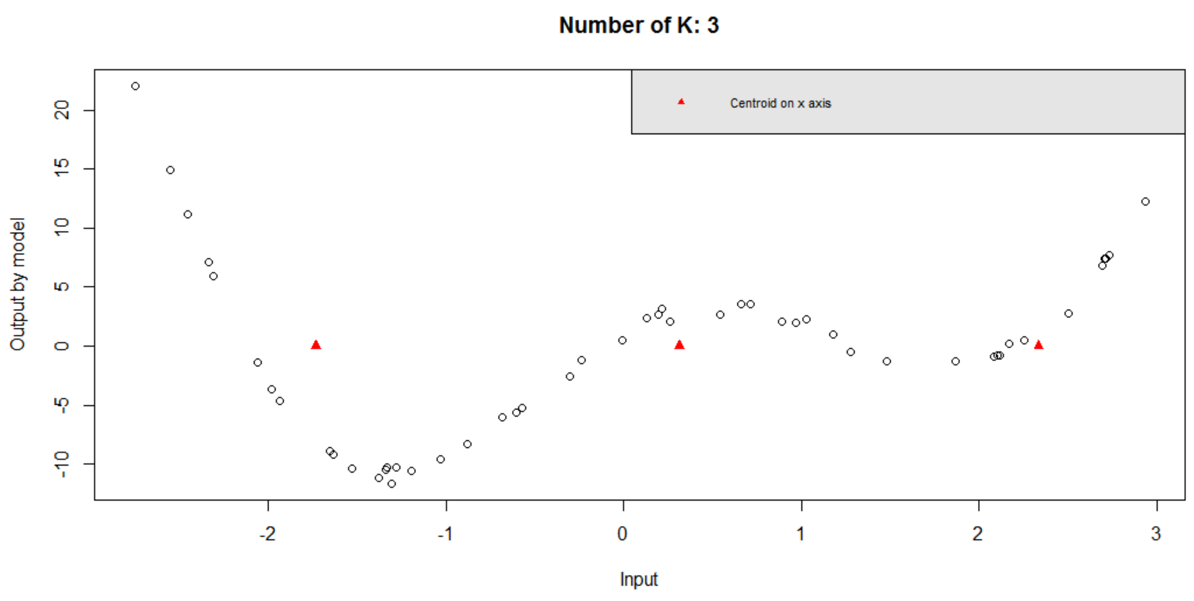


B. For good fit



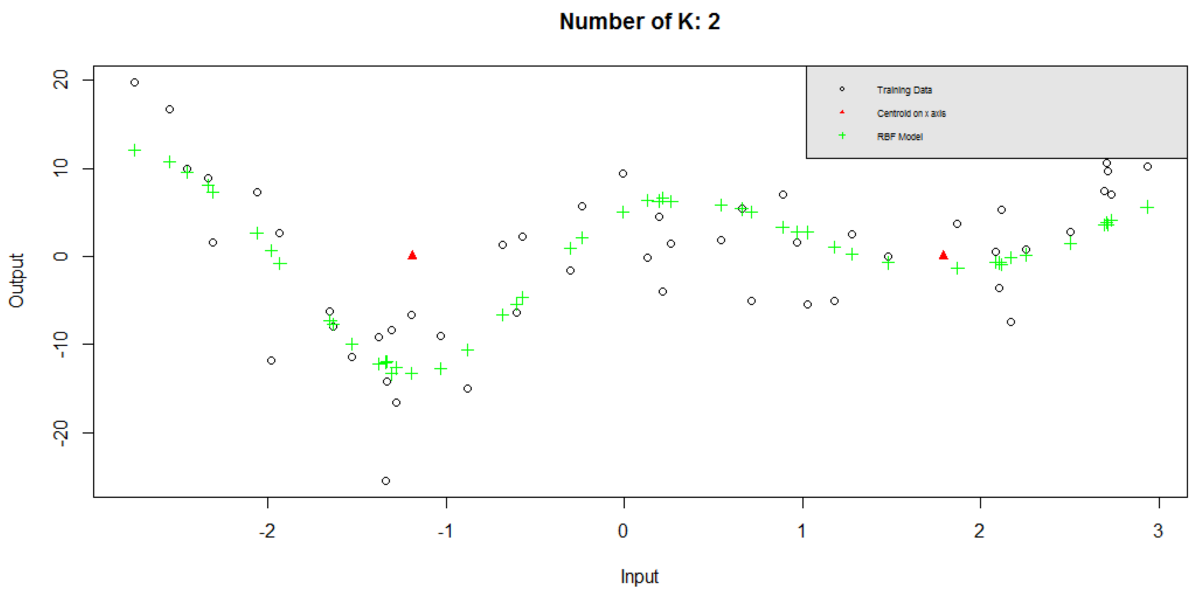
C. For overfitting

Samet Yazak
1604262



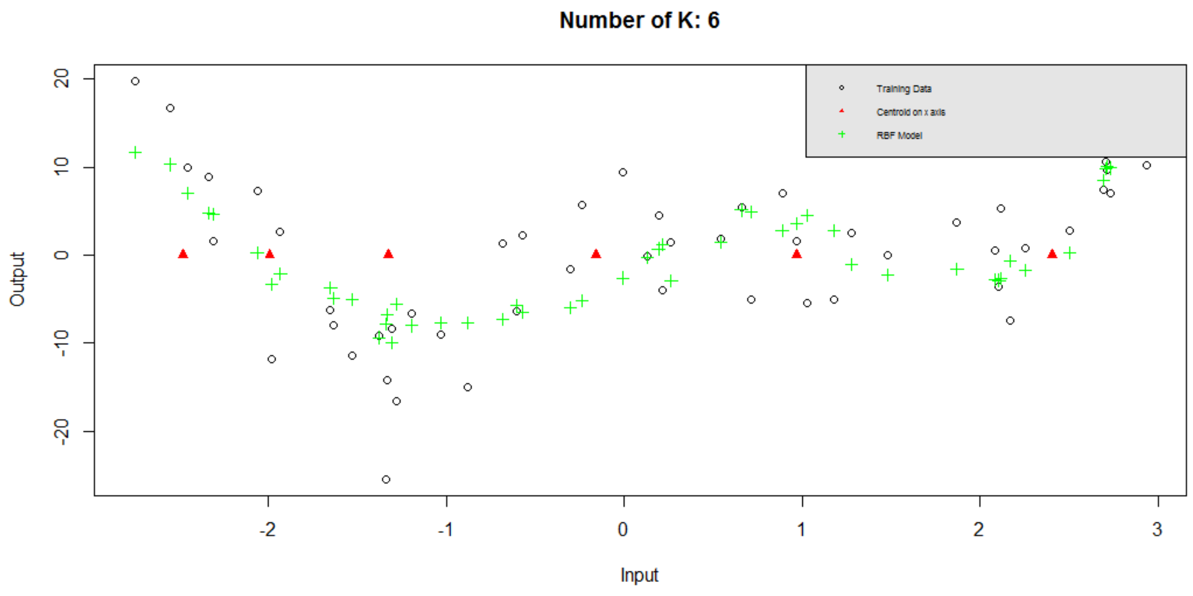
3.3 Training Data and RBF Values

A. For Underfitting

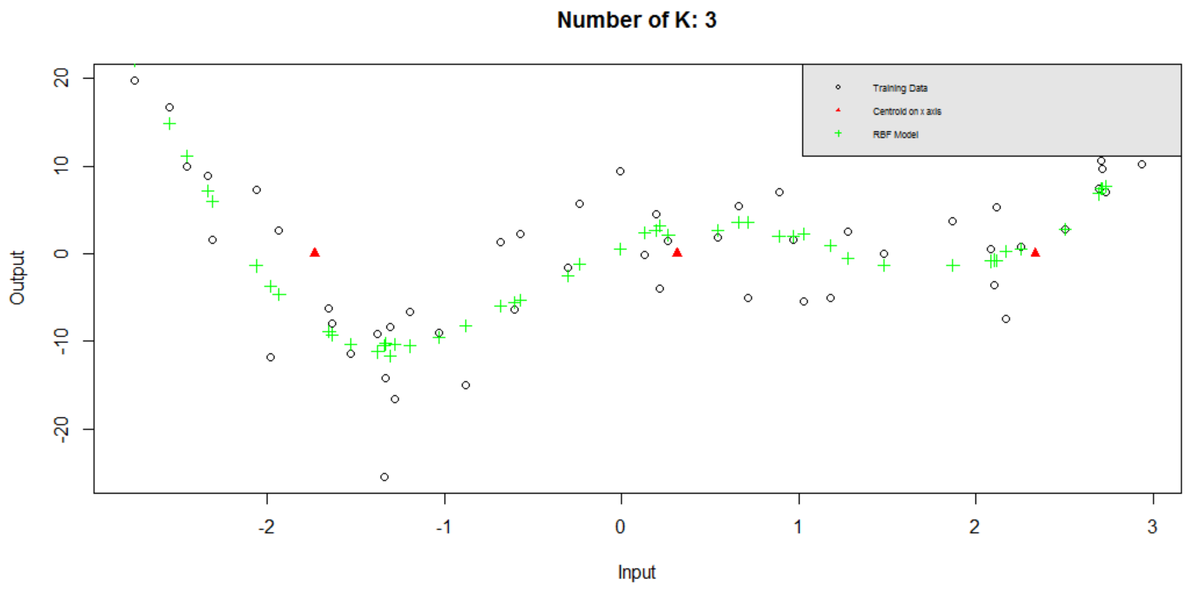


B. For Good Fit

Samet Yazak
1604262



C. For Overfitting



4. Conclusion

As a result of implementation of RBF network for regression, I can say that training and validation error decrease until K (number of clusters) value around 6, 7. Network fits best in K value 6 or 7. When K is increased more than 7, error tends to start again. But, these results highly depends on K-means centroid which are initialized randomly which can give a different result in every execution. These given results are general characteristics of data in RBF network.

Execution time of the training the network is quite fast. It takes 10 seconds avarage to finish for almost every K value (number of cluster).

Samet Yazak
1604262

5. Source Code

You can find the copied source code below. Source code and other material (data, plots, report) for this assignment is also available on github; https://github.com/yazaksamet/RBF .

```
#setwd("C:/Users/sametyazak/Desktop/ynwa/bau/2017 - Spring/ann/Assignment2")

#### hyper parameters

numberOfClusters = 10

epoch = 1000

lerningRateDefault = 0.2

epochLogNumber = 50

## read data

trainingData = read.table("d_reg_tra.txt")

trainingData = data.frame(trainingData, 0, 0)

colnames(trainingData) <- c("Input","Output","ClusterIndex", "ModelOutput")

testData = read.table("d_reg_val.txt")

testData = data.frame(testData, 0)

colnames(testData ) <- c("Input","Output", "ClusterIndex")

trainingDataPlot = data.frame(trainingData$Input, trainingData$Output)

#plot(trainingDataPlot)

clusterCentroids = matrix(sample(c(0:0), numberOfClusters, TRUE), 1, numberOfClusters)

clusterSpreads = matrix(sample(c(0:0), numberOfClusters, TRUE), 1, numberOfClusters)

## assign initial centroids for K-means randomly
```

Samet Yazak
1604262

```
for (c in 1:numberOfClusters) {

  randomCentroidIndex =sample(1:dim(trainingData)[1], 1);

  clusterCentroids[1,c] = trainingData[randomCentroidIndex,1]

}


numberOfClusterChanges = 0

kmeansIterationCount = 0


# iterate until no cluster changes or 10 iterations

while (kmeansIterationCount == 0 || (numberOfClusterChanges > 0 && kmeansIterationCount < 10))
{

  kmeansIterationCount = kmeansIterationCount + 1

  ## assign training data to clusters


  numberOfClusterChanges = 0


  clusterCentroidTotals = matrix(sample(c(0:0), numberOfClusters, TRUE), numberOfClusters, 2)


  for (i in 1:dim(trainingData)[1]) {

    ## find closest cluster


    innerClusterIndex = 0

    inputValue = trainingData[i,1]

    bestDistance = -1


    for (c2 in 1:numberOfClusters) {

      clusterDistance = abs(inputValue - clusterCentroids[1,c2])


      if (bestDistance == -1 || clusterDistance < bestDistance) {

        innerClusterIndex = c2

        bestDistance = clusterDistance

      }

    }
    # assign found cluster

    if (trainingData[i,3] != innerClusterIndex) {

      trainingData[i,3] = innerClusterIndex

      numberOfClusterChanges = numberOfClusterChanges + 1

    }


    clusterCentroidTotals[innerClusterIndex,1] = clusterCentroidTotals[innerClusterIndex,1] +
inputValue

    clusterCentroidTotals[innerClusterIndex,2] = clusterCentroidTotals[innerClusterIndex,2] + 1


    # calculate most distant instance for spread

    if (abs(inputValue - clusterCentroids[1,innerClusterIndex]) > clusterSpreads[1,innerClusterIndex]) {

      clusterSpreads[1,innerClusterIndex] = abs(inputValue - clusterCentroids[1,innerClusterIndex])

    }


  }



  ## re-calculate centroid values

  for (c3 in 1:numberOfClusters) {

    if (clusterCentroidTotals[c3,2] != 0) {

      clusterCentroids[1,c3] = clusterCentroidTotals[c3,1] / clusterCentroidTotals[c3,2]

    }

    else {

      clusterCentroids[1,c3] = 0

    }

  }

}


## calculate cluster spreads


clusterSpreads = clusterSpreads / 2
```

Samet Yazak
1604262

```
## calculate ph values


# + 1 means bias unit

trainingDataPerceptives = matrix(sample(c(0:0), numberOfClusters + 1, TRUE), dim(trainingData)[1],
numberOfClusters + 1)

testDataPerceptives = matrix(sample(c(0:0), numberOfClusters + 1, TRUE), dim(testData)[1],
numberOfClusters + 1)


# assign bias unit

trainingDataPerceptives[,1] = 1

testDataPerceptives[,1] = 1


for (pi in 1:dim(trainingData)[1]) {

  for (ph in 1:numberOfClusters) {

    pht = exp(-((abs(trainingData[pi,1] - clusterCentroids[1,ph]))^2 / 2*(clusterSpreads[1,ph])^2))

    trainingDataPerceptives[pi,ph+1] = pht

  }

}


for (pi in 1:dim(testData)[1]) {

  for (ph in 1:numberOfClusters) {

    pht = exp(-((abs(testData[pi,1] - clusterCentroids[1,ph]))^2 / 2*(clusterSpreads[1,ph])^2))

    testDataPerceptives[pi,ph+1] = pht

  }

}



## train sinple layer perceptron


# weight initialization


inputWeight = matrix(sample(c(-100:100), numberOfClusters + 1, TRUE) / 10000, 1,
numberOfClusters + 1)
```

Samet Yazak
1604262

```
errorMatrix = matrix(0, ncol = 4, nrow = ((epoch / epochLogNumber)))

errorSet = data.frame(errorMatrix)

colnames(errorSet)<-c("K","epoch","trainingMSE", "validationSME")



# run epochs


learningRate = lerningRateDefault


for (e in 1:epoch){
 for(x in 1:dim(trainingData)[1]) {
   y = sum(inputWeight[1,] * trainingDataPerceptives[x,])
   r = trainingData[x,2]

   deltaWeight = learningRate*(r-y)*trainingDataPerceptives[x,]
   inputWeight = inputWeight + deltaWeight

   trainingData[x,4] = y
 }

 if (e %% epochLogNumber == 0) {
   trainingErrorSum = 0
   validationErrorSum = 0

   # calculate training data
   for(x in 1:dim(trainingData)[1]) {
     y = sum(inputWeight[1,] * trainingDataPerceptives[x,])
     r = trainingData[x,2]

     trainingErrorSum = trainingErrorSum + 0.5 * (r-y)^2
   }
```

Samet Yazak
1604262

```
    # mean square error
    trainingMSE = trainingErrorSum / dim(trainingData)[1]

    for(x in 1:dim(testData)[1]) {
      y = sum(inputWeight[1,] * testDataPerceptives[x,])
      r = testData[x,2]

      validationErrorSum = validationErrorSum + 0.5 * (r-y)^2
    }

    # mean square error
    validationMSE = validationErrorSum / dim(testData)[1]

    # log error
    epochLogIndex = (e / epochLogNumber)
    errorSet[epochLogIndex,1] = numberOfClusters
    errorSet[epochLogIndex,2] = e
    errorSet[epochLogIndex,3] = trainingMSE
    errorSet[epochLogIndex,4] = validationMSE
  }

  learningRate = learningRate * 0.998
  print(paste0("K: ", numberOfClusters, ", epoch:", e))
}

## plot validation and training error

subErrorSet = errorSet
subTrainingError = data.frame(subErrorSet$epoch, subErrorSet$trainingMSE)
subValidationError = data.frame(subErrorSet$epoch, subErrorSet$validationSME)

maxValidationError = max(subValidationError[,2])
minValidationError = min(subValidationError[,2])
```

Samet Yazak
1604262

```
maxTrainingError = max(subTrainingError[,2])

minTrainingError = min(subTrainingError[,2])


minError = min(minValidationError,minTrainingError) - 2

maxError = max(maxTrainingError,maxValidationError) + 2


plot(subTrainingError,type="l",col=4, lty = 2, xlab="Epoch", ylab="Mean Square Error",
   xlim=c(0,epoch),
   ylim=c(minError, maxError),
   axes = FALSE)
axis(1, xaxp=c(0, epoch, (epoch/(epochLogNumber*2))), las=2)
axis(side = 2, at = seq(as.integer(minError), as.integer(maxError), by = 2))
#axis(2, xaxp=c(10, 1000, 10), las=2)
lines(subValidationError,col=6,lty = 1, pch = 4)
title(paste0("Number of K: ", numberOfClusters))
legend("topright", c("Training", "Validation"), col = c(4, 6),
    text.col = "black", lty = c(2,1),
    bg = "gray90", cex = 0.5)
box()


## plot input and model output for training data


trainingDataPlot = data.frame(trainingData$Input, trainingData$ModelOutput)
clusterCentroidsPlot = matrix(sample(c(0:0), numberOfClusters, TRUE), numberOfClusters, 2)
clusterCentroidsPlot[,1] = t(clusterCentroids)


plot(trainingDataPlot, xlab="Input", ylab="Output by model")
points(clusterCentroidsPlot, col="red", pch=17)
title(paste0("Number of K: ", numberOfClusters))
legend("topright", c("Centroid on x axis"), col = c("red"),
    text.col = "black", pch=17,
    bg = "gray90", cex = 0.7)
```

Samet Yazak
1604262

```
## plot both training data and model output


trainingDataPlot2 = data.frame(trainingData$Input, trainingData$Output)


plot(trainingDataPlot2, xlab="Input", ylab="Output")

points(clusterCentroidsPlot, col="red", pch=17)

points(trainingDataPlot, col="green", pch=3)

title(paste0("Number of K: ", numberOfClusters))

legend("topright", c("Training Data","Centroid on x axis", "RBF Model"), col = c("black","red",
"green"),

    text.col = "black", pch= c(1,17,3),

    bg = "gray90", cex = 0.5)
```