STAT 542: Homework 4

Shashi Roshan (sroshan2)

Contents

Question 1 [40 Points] K-Means Clustering

Let's consider coding our own K-means algorithm. Perform the following:

- [15 Points] Write your own code of k-means that iterates between two steps, and stop when the cluster membership does not change. Use the ℓ_2 norm as the distance metric. Write your algorithm as efficiently as possible. Avoiding extensive use of for-loop could help.
  - updating the cluster means given the cluster membership
  - updating the cluster membership based on the cluster means

```
kmeans_clustering = function(x, k, num_random_init){

  print_info = F

  n = nrow(x)

  within_cluster_ss_array = c()
  cluster_assignment_matrix = matrix(NA, num_random_init, n)

  # for each random initialization of clusters
  for(random_init in 1:num_random_init){

    if(print_info)
      cat('\nRandom initialization of cluster labels : ', random_init, '\n')

    # random initialization of cluster labels
    cluster_assignment = floor(runif(nrow(x), min = 0, max = k)) + 1
    if(print_info)
      cat('First ten random cluster initialization : ', cluster_assignment[1:min(nrow(x), 10)], '\n')

    cluster_mean = matrix(NA, k, ncol(x))

    iteration = 1
```

```r
  # repeat until convergence, or till max iterations are reached
  repeat{

    if(print_info)
      cat('iteration : ', iteration, '\n')

    # step 1 : update the cluster means
    for(cluster in 1:k){
      cluster_mean[cluster, ] =  colMeans(x[cluster_assignment==cluster, ])
    }

    # calculate within_cluster_ss
    within_cluster_ss = sum(sapply(1:nrow(x), function(idx)
        {
          cluster_centroid = cluster_mean[cluster_assignment[idx], ]
          sum((x[idx, ] - cluster_centroid)^2)
        }
      ))

    # step 2 : update the cluster membership
    cluster_assignment_before_update = cluster_assignment

    l2_norm_matrix = outer(1:nrow(x), 1:nrow(cluster_mean), FUN = Vectorize(function(i, j)
      sum((x[i,] - cluster_mean[j,]) ^ 2)))

    cluster_assignment = apply(l2_norm_matrix, 1, function(x)
      which(x == min(x, na.rm = TRUE)))

    if (print_info) {
      print('First ten cluster_assignment after step 2 : ')
      for (idx in 1:min(nrow(x), 10)) {
        print(cluster_assignment[idx])
      }
    }

    if(print_info){
      cat(
        'random_init : ',
        random_init,
        '. iteration : ',
        iteration,
        '. within_cluster_ss : ',
        within_cluster_ss,
        '\n'
      )
    }

    # check if cluster assignments changed
    if(all(cluster_assignment_before_update==cluster_assignment)==TRUE){
      if(print_info)
        {
          cat('No change in cluster assignment. Clustering completed. Iteration : ',
              iteration)
```

```
          }
        break
      }

      iteration = iteration + 1

      # check if max number of iterations are reached
      if(iteration > 100){
        print('Max number of iterations reached.')
        break
      }

    }

    within_cluster_ss_array = c(within_cluster_ss_array, within_cluster_ss)
    cluster_assignment_matrix[random_init, ] = cluster_assignment
    if(print_info)
      cat('First ten cluster_assignments : ', cluster_assignment[1:min(nrow(x), 10)], '\n\n')

  }

  # return the cluster assignment corresponding to the minimum within_cluster_ss
  cat('\n\n')
  idx_min_within_cluster_ss_array = order(within_cluster_ss_array)[1]

  if(print_info){
    cat('within_cluster_ss_array for all iterations: ', within_cluster_ss_array, '\n')
    cat('idx_min_within_cluster_ss_array : ', idx_min_within_cluster_ss_array, '\n\n')
  }

  cat('Minimum within cluster sum of squares : ',
      within_cluster_ss_array[idx_min_within_cluster_ss_array],
      '\n')

  return(cluster_assignment_matrix[idx_min_within_cluster_ss_array, ])

}
```

- [5 points] Test your code with this small-sized example by performing the following. Make sure that you save the random seed.

```
set.seed(4)
x = rbind(matrix(rnorm(100), 50, 2), matrix(rnorm(100, mean = 0.5), 50, 2))
y = c(rep(0, 50), rep(1, 50))
```

+ Run your algorithm on the data using __ONE__ random initialization, output the within-cluster distance, and compare the resulting clusters with the underlying truth.

```
set.seed(4)
cluster_assignment = kmeans_clustering(x, k = 2, num_random_init = 1)
```

##

```
##
## Minimum within cluster sum of squares :  129.8448
```

```
# min within_cluster_ss:  129.8448

yhat = cluster_assignment - 1
# since the function kmeans_clustering() assigns clusters starting from 1

cat('Accuracy : ', mean(y == yhat)*100, '%') # 52
```

```
## Accuracy :  52 %
```

```
+ Run your algorithm on the data using __20__ random initialization, output the within-cluster distance
and compare the resulting clusters with the underlying truth.
```

```
set.seed(4)
cluster_assignment = kmeans_clustering(x, k = 2, num_random_init = 20)
```

```
##
##
## Minimum within cluster sum of squares :  116.2215
```

```
# min within_cluster_ss:  116.2215

yhat = cluster_assignment - 1
# since the function kmeans_clustering() assigns clusters starting from 1

cat('Accuracy : ', mean(y == yhat)*100, '%') # 42
```

```
## Accuracy :  42 %
```

```
+ Do you observe any difference? Note that the result may vary depending on the random seed.
```

The accuracy when one random initialization is used (52%) is greater than the accuracy when 20 random initializations are used. This is unexpected, since using more number of random initializations for clustering should have given better results. Following can be the reasons for this :
- The data was generated in such a way, that the underlying clusters of the data are very close to each other, and maybe overlapping with each other (Cluster 0 : Normal distribution with mean 0, Cluster 1 :Normal distribution with mean 0.5).
- The initial cluster assignments depends on the seed set. Changing the seed can give different results.

- [10 Points] Load the `zip.train` (handwritten digit recognition) data from the `ElemStatLearn` package and the goal is to identify clusters of digits based on only the pixels. Apply your algorithm on the handwritten digit data with $k = 15$, with ten random initialization. Then, compare your cluster membership to the true digits.

```
library(ElemStatLearn)
data("zip.train")
data("zip.test")
```

```
x = zip.train[, 2:257]
y = zip.train[, 1]
k = 15

set.seed(4)
cluster_assignment = kmeans_clustering(x,
                   k = k,
                   num_random_init = 10) # min within_cluster_ss:  511239.3
```

```
##
##
## Minimum within cluster sum of squares :  511239.3
```

+ For each of your identified clusters, which is the representative (most dominating) digit?

```
# Find mode of a vector
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

for(cluster in 1:15){
  cat('cluster : ', cluster, '\n')
  print(table(y[cluster_assignment==cluster]))
  cat('Most dominating digit : ', Mode(y[cluster_assignment==cluster]))
  cat('\n\n')
}
```

```
## cluster :  1
##
##   0   2   5   6
## 198  16   9 103
## Most dominating digit :  0
##
## cluster :  2
##
##   0   2   4   5   6   8
## 261   2   1   4   3   1
## Most dominating digit :  0
##
## cluster :  3
##
##   0   1   2   3   4   5   6   7   8   9
##   3   3  32  42   2  20   1   2 419   7
## Most dominating digit :  8
##
## cluster :  4
##
##   2   3   4   5   6   7   8   9
##   6   2 277   8   1  27   8 143
## Most dominating digit :  4
##
```

```
## cluster :  5
##
##   1   2   4   6   7   8   9
## 998   4  19   2   1   8   6
## Most dominating digit :  1
##
## cluster :  6
##
##   0   2   3   4   5   6   7   8   9
##  14 166  10 204  36  23   7  19   3
## Most dominating digit :  4
##
## cluster :  7
##
##   0   2   4   5   6
##   3   1   2   7 302
## Most dominating digit :  6
##
## cluster :  8
##
##   0   1   2   3   4   5   7   8   9
##   1   1   9   7 135   5 171  26 438
## Most dominating digit :  9
##
## cluster :  9
##
##   0   2   3   5   6   8
## 264   1   2   2   2   2
## Most dominating digit :  0
##
## cluster :  10
##
##   0   1   2   3   4   5   6   7   8
##  67   2   7  16   4  63 219   1  14
## Most dominating digit :  6
##
## cluster :  11
##
##   0   2   3   5   6   8
## 377   1   2   2   6   5
## Most dominating digit :  0
##
## cluster :  12
##
##   2   3   4   5   7   8   9
##  11   2   8   2 434   1  45
## Most dominating digit :  7
##
## cluster :  13
##
##   2   3   5   7   8
## 440   9   1   2   3
## Most dominating digit :  2
##
```

```
## cluster :  14
##
##   0   2   3   5   8   9
##   3  35 542  14  27   2
## Most dominating digit :  3
##
## cluster :  15
##
##   0   1   3   5   6   8
##   3   1  24 383   2   9
## Most dominating digit :  5
```

+ How well does your clustering result recover the truth?
Which digits seem to get mixed up (if any) the most?
Again, this may vary depending on the random seed.
Hence, make sure that you save the random seed.

```r
# Find train data accuracy
x = zip.train[, 2:257]
y = zip.train[, 1]

# Find mean of each cluster
cluster_mean = matrix(NA, k, ncol(x))
cluster_label = c()

for(cluster in 1:k){
  x_cluster = x[cluster_assignment==cluster, ]
  cluster_mean[cluster, ] =  colMeans(x_cluster)
  cluster_label = c(cluster_label, Mode(y[cluster_assignment==cluster]))
}

yhat = c()

for(idx in 1:nrow(x)){

  l2_norm_from_cluster_means = c()

  for(cluster in 1:k){
      l2_norm_from_cluster_mean = sqrt(sum((x[idx, ] - cluster_mean[cluster, ])^2))
      l2_norm_from_cluster_means = c(l2_norm_from_cluster_means, l2_norm_from_cluster_mean)
  }

  yhat = c(yhat, cluster_label[order(l2_norm_from_cluster_means)][1])

}

# Train data accuracy
mean(y == yhat) # 0.7894665
```

```
## [1] 0.7894665
```

Which digits seem to get mixed up (if any) the most?
In cluster 1, digit 0 and 6 seem to get mixed up.

In cluster 4, digit 4 and 9 seem to get mixed up.
In cluster 6, digit 2 and 4 seem to get mixed up.
In cluster 8, digit 4, 7 and 9 seem to get mixed up.
In summary, following digits seem to get mixed up : (0, 6), (4, 7, 9), (2, 4).

- [10 Points] If you are asked to use this clustering result as a predictive model and suggest (predict) a label for each of the testing data `zip.test`, what would you do? Properly describe your prediction procedure and carry out the analysis. What is the prediction accuracy of your model? Note that you can use the training data labels for prediction.

Procedure to predict the labels of test data :
- From the clusters formed using the train data, find the mean of each cluster. Also, find the label of the clusters (most dominating digit).
- For each observation in the test data, find it's l2_norm from the clusters means. Assign it to the cluster for which the l2_norm is the least. It's predicted yhat will be the label of that cluster (most dominating digit).

```
# Find test data accuracy
x = zip.train[, 2:257]
y = zip.train[, 1]

# Find mean of each cluster
cluster_mean = matrix(NA, k, ncol(x))
cluster_label = c()

for(cluster in 1:k){
  x_cluster = x[cluster_assignment==cluster, ]
  cluster_mean[cluster, ] =  colMeans(x_cluster)
  cluster_label = c(cluster_label, Mode(y[cluster_assignment==cluster]))
}

data('zip.test')
x_test = zip.test[, 2:257]
y_test = zip.test[, 1]
yhat = c()

for(idx in 1:nrow(x_test)){

  l2_norm_from_cluster_means = c()

  for(cluster in 1:k){
      l2_norm_from_cluster_mean = sqrt(sum((x_test[idx, ] - cluster_mean[cluster, ])^2))
      l2_norm_from_cluster_means = c(l2_norm_from_cluster_means, l2_norm_from_cluster_mean)
  }

  yhat = c(yhat, cluster_label[order(l2_norm_from_cluster_means)][1])

}

# Test data accuracy
mean(y_test == yhat) # 0.7513702
```

```
## [1] 0.7513702
```

Question 2 [20 Points] Kernel Density Estimation

If you do not use latex to type your answer, you will lose 2 points. During our lecture, we analyzed the consistency of a kernel density estimator, defined as

$$\widehat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x, x_i)$$

Let's consider a 2-dimensional case, where $x = (z, w)^T$, and $x_i = (z_i, w_i)^T$. If we use a 2-dimensional kernel, constructed as the product of two kernel functions:

$$\mathbf{K}_\lambda(x, x_i) = K_\lambda(z, z_i)K_\lambda(w, w_i)$$

We would expect our kernel density estimation still being an unbiased estimator. Show this by extending the one-dimensional proof we did during the lecture to this new case. Please be aware of the following and make sure that you properly address them during the proof.

$$\widehat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_\lambda(z, z_i) * K_\lambda(w, w_i) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\lambda^2} K_\lambda(\frac{z - z_i}{\lambda})K_\lambda(\frac{w - w_i}{\lambda})$$

Also, we know that,

$$E[K_\lambda(\frac{z - z_0}{\lambda})] = E[K_\lambda(\frac{z - z_1}{\lambda})] = E[K_\lambda(\frac{z - z_2}{\lambda})] = ... = E[K_\lambda(\frac{z - z_n}{\lambda}))]$$

$$E[K_\lambda(\frac{w - w_0}{\lambda})] = E[K_\lambda(\frac{w - w_1}{\lambda})] = E[K_\lambda(\frac{w - w_2}{\lambda})] = ... = E[K_\lambda(\frac{w - w_n}{\lambda})]$$

Hence, we can find the expected value :

$$E[\widehat{f}(x)] = E[\frac{1}{n}\sum_{i=1}^{n} \frac{1}{\lambda^2} K_\lambda(\frac{z - z_i}{\lambda})K_\lambda(\frac{w - w_i}{\lambda})] = \frac{1}{\lambda^2} E[K_\lambda(\frac{z - z_0}{\lambda})K_\lambda(\frac{w - w_0}{\lambda})]$$

$$= \frac{1}{\lambda^2}\int\int K_\lambda(\frac{z - z_0}{\lambda})K_\lambda(\frac{w - w_0}{\lambda}) f(z_0, w_0) dz_0 dw_0$$

Let $z = z_0 + \lambda * u$, $w = w_0 + \lambda * v$
Using variable transformation :

$dz_0 = -\lambda du$, $dw_0 = -\lambda dv$

Hence,

$$E[\widehat{f}(x)] = \int\int K_\lambda(u)K_\lambda(v) f\left(z - \lambda u, w - \lambda v\right) dudv$$

$= \int\int K_\lambda(u)K_\lambda(v)[f(z, w) - f_z(z, w) * \lambda u - f_w(z, w) * \lambda v + \frac{1}{2}(f_{zz}(z, w) * \lambda^2 u^2 + f_{ww}(z, w) * \lambda^2 v^2 + 2 * \lambda^2 * uv * f_{zw}(z, w)) + O(\lambda^3)]dudv$

(Above is derived from Taylor Expansion :

$$f(z - \lambda u, w - \lambda v) = f(z, w) - f_z(z, w) * \lambda u - f_w(z, w) * \lambda v + \frac{1}{2}(f_{zz}(z, w) * \lambda^2 u^2 + f_{ww}(z, w) * \lambda^2 v^2 + 2 * \lambda^2 * uv * f_{zw}(z, w)) + O(\lambda^3)$$

where
$f_w = \frac{df}{dw}$, $f_z = \frac{df}{dz}$, $f_{ww} = \frac{df_w}{dw}$, $f_{zz} = \frac{df_z}{dz}$, $f_{zw} = \frac{df_z}{dw}$)

$$E[\widehat{f}(x)] = \int\int K_\lambda(u)K_\lambda(v)f(z,w)dudv - \int\int K_\lambda(u)K_\lambda(v)f_z(z,w) * \lambda ududv -$$

$$\int\int K_\lambda(u)K_\lambda(v)f_w(z,w) * \lambda vdudv + \frac{1}{2}[\int\int K_\lambda(u)K_\lambda(v)f_{zz}(z_0,w_0) * \lambda^2u^2dudv +$$

$$\int\int K_\lambda(u)K_\lambda(v)f_{ww}(z,w) * \lambda^2v^2dudv + \int\int K_\lambda(u)K_\lambda(v) * 2 * \lambda^2 * uv * f_{zw}(z,w))dudv] + \int\int K_\lambda(u)K_\lambda(v)O(\lambda^3)]dudv$$

Assuming that the kernel function satifies following conditions :

$\int\int K_\lambda(u)K_\lambda(v)f_w(z,w) * \lambda vdudv = 0$, $\int\int K_\lambda(u)K_\lambda(v)f(z,w)dudv = 1$, and
$\int\int K_\lambda(u)K_\lambda(v)f_z(z,w) * \lambda ududv = 0$

Hence,

$$E[\widehat{f}(x)] = f(z,w) + 0 + 0 + c_1 * \lambda^2 + c_2 * O(\lambda^3)$$

Therefore,

$$Bias^2 = \int [E[\widehat{f}(x)] - f(x)]^2dx = \int (c_1 * \lambda^2 + c_2 * O(\lambda^3))^2dx$$

If $\lambda \to 0$, $E[\widehat{f}(x)] = f(z,w) = f(x)$

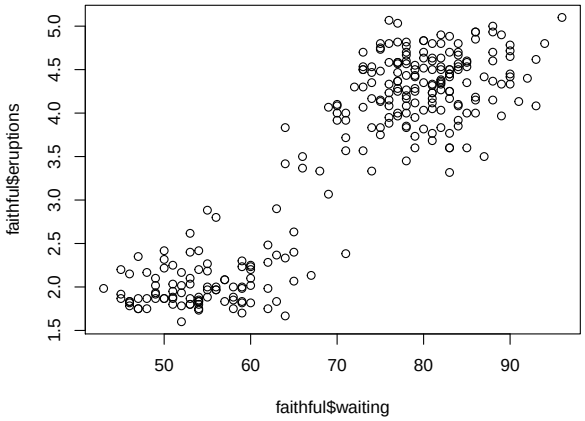
Hence, it is a consistent estimator.

- In the one-dimensional proof, we require assumptions on the kernel function $K(\cdot)$. You can automatically assume these in your proof.
- A two-dimensional Taylor expansion is needed.
- Is this kernel density estimator consistent? Yes. As λ goes to 0, the expected value of f(x) is equal to f(x). Hence, it is a consistent kernel density estimator.
- What is the rate (in terms of λ) of the bias term following your derivation? Is the rate slower or faster than the one-dimensional case? The bias-square term decreases at the rate of λ^2 as lambda goes to zero (If we ignore the higher lambda order terms). Also, the impact of the higher order terms of λ is negligible, hence, we can conclude that bias-square term will approach zero at almost the same rate as the 1D-kernel density estimator.
- What do you think about the variance of this estimator? You do NOT need to give proof. Providing some thoughts is sufficient. This question does not count towards your grade.
  The variance of will increase as bandwidth becomes smaller.

Question 3 [40 Points] Two-dimensional Kernel Regression

For this question, you need to write your own functions. The goal is to model and predict the eruption duration using the waiting time in between eruptions of the Old Faithful Geyser. You can download the data from [kaggle: Old Faithful] or our course website.

```
load("faithful.Rda")
plot(faithful$waiting, faithful$eruptions)
```



Perform the following:

- [10 points] Use a kernel density estimator to estimate the density of the eruption waiting time. Use a bandwidth with the theoretical optimal rate (the Silverman rule-of-thumb formula). Plot the estimated density function. You should consider two kernel functions:
  - Gaussian Kernel
  - Uniform kernel

```
set.seed(4)
n = nrow(faithful)
x = faithful$waiting

# Use a bandwidth with the theoretical optimal rate (the Silverman rule-of-thumb formula)
lambda = 1.06*sd(x)*n^(-1/5)

xgrid = seq(40, 100, 0.01)
# xgrid = sort(x)

par(mar=rep(2, 4))

# Create a placeholder plot
plot(
  xgrid,
  dnorm(xgrid, 0, 1),
  type = "l",
  lwd = 3,
  col = "deepskyblue",
  xlab = "x",
  ylab = "density",
  ylim = c(0, 0.05)
)

# Gaussian kernel
den_gaussian = matrix(NA, length(x), length(xgrid))
```

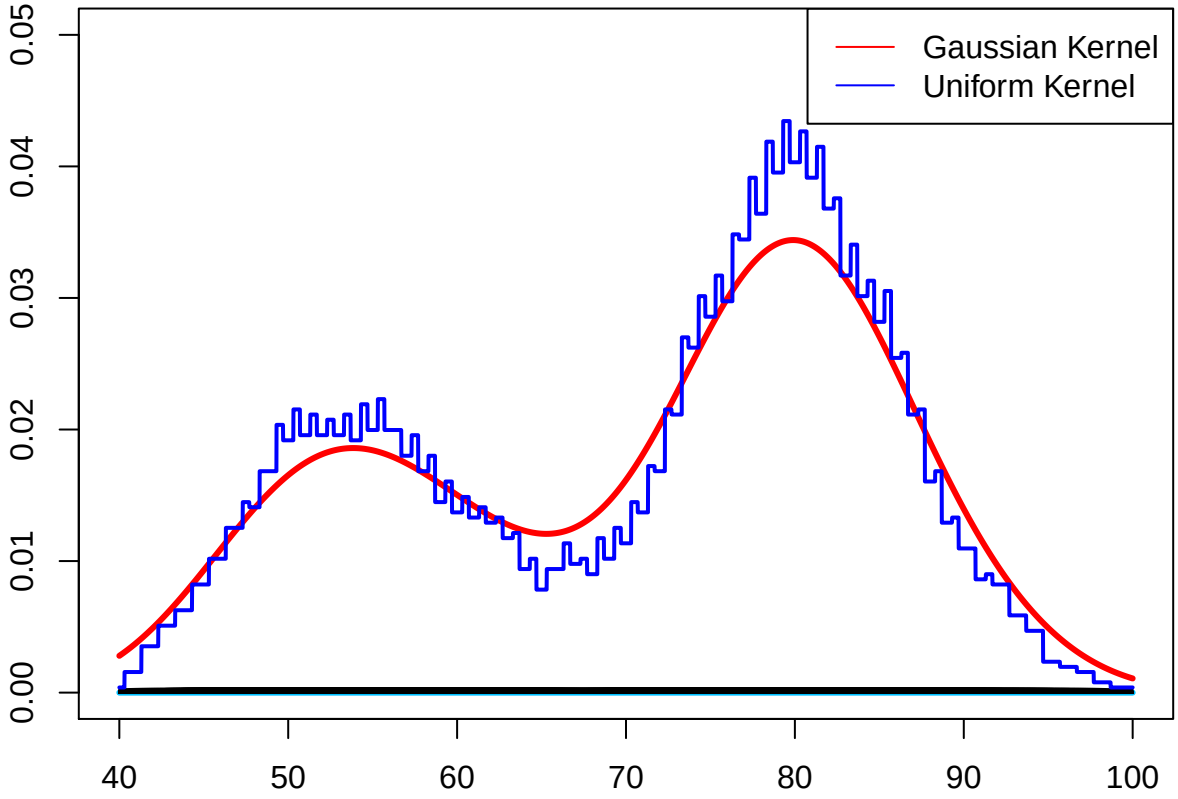
```r
for (i in 1:length(x))
    {
        den_gaussian[i, ] = exp(-0.5*(x[i] - xgrid)^2 / lambda^2)/sqrt(2*pi)/lambda/length(x)
        points(xgrid, den_gaussian[i, ], type = "l")
    }

lines(xgrid, colSums(den_gaussian), type = "l", lwd = 3, col = 2)

# Uniform kernel, with width = 2
den_uniform = matrix(NA, length(x), length(xgrid))
for (i in 1:length(x))
    {
      den_uniform[i, ] = (abs(xgrid - x[i]) <= lambda)/length(x)/2/lambda
    }

lines(xgrid, colSums(den_uniform), type = "l", lwd = 2, col = 4)

legend("topright", c("Gaussian Kernel", "Uniform Kernel"), col = c(2,4), lty = 1)
```



- [15 points] Use a Nadaraya-Watson kernel regression estimator to model the conditional mean eruption duration using the waiting time. Use a 10-fold cross-validation to select the optimal bandwidth. You should use the Epanechnikov kernel for this task. Report the optimal bandwidth and plot the fitted regression line using the optimal bandwidth.

```r
library(caret)

# eruptions ~ waiting
# x = faithful$waiting
# y = faithful$eruptions

# store indexes for 10 fold cross-validation
set.seed(4)
index_fold = caret::createFolds(faithful$eruptions, k = 10)

calc_rmse = function(actual, predicted){
  return(sqrt(mean((actual-predicted)^2)))
}

epanechnikov_kernal = function(xtest, xtrain, lambda){
  x1 = sweep(as.data.frame(xtrain), 2, as.numeric(xtest))
  x2 = sweep(x1,2,lambda, "/")
  x3 = apply(x2, 1, function(x) (1 - x^2)  )
  return((3/4)*x3*(abs(x2) <= 1))
}

get_test_data_yhat = function(train_data, test_data, lambda){

  y_hats = c()

  for(test_data_sample in test_data){
     K = epanechnikov_kernal(test_data_sample, train_data$waiting, lambda) / lambda
     y_hat = sum(K * train_data$eruptions) / sum(K)
     y_hats = c(y_hats, y_hat)
  }

  return(y_hats)
}

lambda_grid = c(seq(from = 2, to = 25, by= 0.1))
mean_rmse_for_each_lambda = c()

for(lambda_ in lambda_grid){

  rmse_ = c()

  for (idx in 1:10){
    train_data = faithful[-index_fold[[idx]],]
    test_data = faithful[index_fold[[idx]],]
    y_hats = get_test_data_yhat(train_data, test_data$waiting, lambda = lambda_)
    rmse_ = c(rmse_, calc_rmse(y_hats, test_data$eruptions))
  }

  mean_rmse_for_each_lambda = c(mean_rmse_for_each_lambda, mean(rmse_))
}

lambda_min = lambda_grid[order(mean_rmse_for_each_lambda)][1]
```

```r
# optimal lambda
lambda_min # 10
```

```
## [1] 10
```

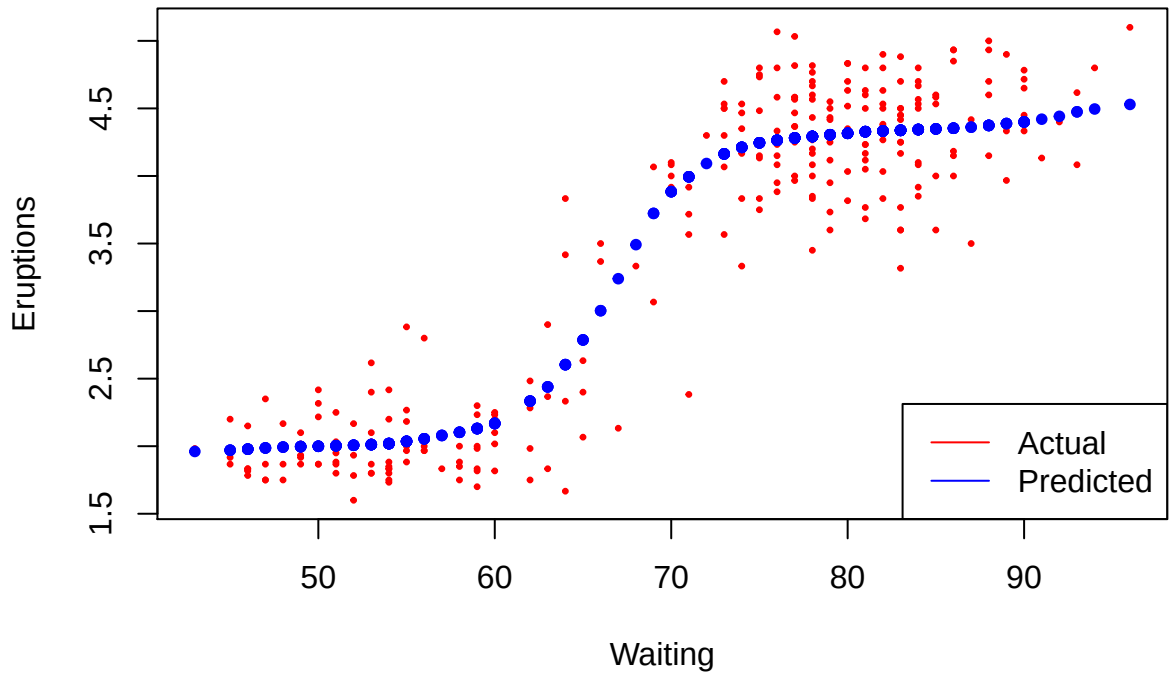
```r
# Mean RMSE for optimal lambda
mean_rmse_for_each_lambda[order(mean_rmse_for_each_lambda)][1] # 0.3659248
```

```
## [1] 0.3659248
```

```r
# plot the fitted regression line using the optimal bandwidth
y_hats = get_test_data_yhat(faithful, faithful$waiting, lambda = lambda_min)

plot(
  faithful$waiting,
  faithful$eruptions,
  col = 2,
  pch = 20,
  cex = 0.5,
  xlab = 'Waiting',
  ylab = 'Eruptions'
)
points(faithful$waiting, y_hats, col=4, pch=20, cex=1)

legend("bottomright", c("Actual", "Predicted"), col = c(2,4), lty = 1)
```



```
# > y_hats[1:10]
# [1] 4.305100 2.019440 4.212094 2.333655 4.348290 2.035941 4.374204
# [8] 4.348290 2.003494 4.348290
```

- [15 points] Fit a local linear regression to model the conditional mean eruption duration using the waiting time. Use a 10-fold cross-validation to select the optimal bandwidth. You should use the Gaussian kernel for this task. Report the optimal bandwidth and plot the fitted regression line using the optimal bandwidth.

```
library(caret)

# eruptions ~ waiting
# x = faithful$waiting
# y = faithful$eruptions

# store indexes for 10 fold cross-validation
set.seed(4)
index_fold = caret::createFolds(faithful$eruptions, k = 10)

calc_rmse = function(actual, predicted){
  return(sqrt(mean((actual-predicted)^2)))
}

get_test_data_yhat = function(train_data, test_data, lambda){

  y_hats = c()

  for(test_data_sample in test_data){

     x = train_data$waiting
     y = train_data$eruptions
     x0 = test_data_sample

     # Guassian kernel
     K = exp(-0.5*((x - x0)/lambda)^2)/sqrt(2*pi)/lambda

     # Local linear regression
     wX = sweep(cbind(1, x), 1, sqrt(K), FUN = "*")
     wy = y*sqrt(K)
     b = solve(t(wX) %*% wX) %*% t(wX) %*% wy

     y_hat = b[1]+x0*b[2]
     y_hats = c(y_hats, y_hat)
  }

  return(y_hats)
}

lambda_grid = c(seq(from = 1, to = 30, by= 0.1))
mean_rmse_for_each_lambda = c()

for(lambda_ in lambda_grid){
```

```
  rmse_ = c()

  for (idx in 1:10){
    train_data = faithful[-index_fold[[idx]],]
    test_data = faithful[index_fold[[idx]],]
    y_hats = get_test_data_yhat(train_data, test_data$waiting, lambda = lambda_)
    rmse_ = c(rmse_, calc_rmse(y_hats, test_data$eruptions))
  }

  mean_rmse_for_each_lambda = c(mean_rmse_for_each_lambda, mean(rmse_))
}

lambda_min = lambda_grid[order(mean_rmse_for_each_lambda)][1]

# Optimal lambda
lambda_min # 2.7
```

```
## [1] 2.7
```

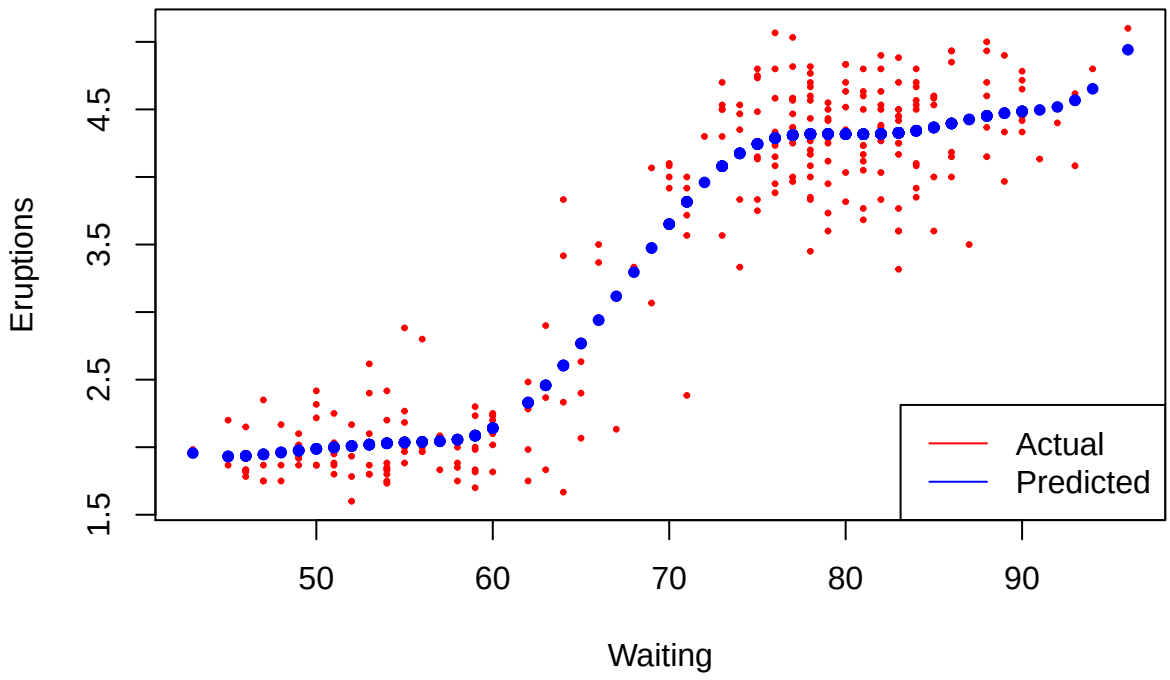
```
# Mean RMSE for optimal lambda
mean_rmse_for_each_lambda[order(mean_rmse_for_each_lambda)][1] # 0.3676489
```

```
## [1] 0.3676489
```

```
# plot the fitted regression line using the optimal bandwidth
y_hats = get_test_data_yhat(faithful, faithful$waiting, lambda = lambda_min)

plot(faithful$waiting, faithful$eruptions, col=2, pch=20, cex=0.5, xlab = 'Waiting', ylab='Eruptions')
points(faithful$waiting, y_hats, col=4, pch=20, cex=1)

legend("bottomright", c("Actual", "Predicted"), col = c(2,4), lty = 1)
```




```
# y_hats[1:10]
#  [1] 4.318595 2.028934 4.174764 2.330271 4.366429 2.035605 4.452157
#  [8] 4.366429 1.998534 4.366429
```