PSL (S21) Coding Assignment 1

Code

Zihe Liu

01/25/2021

In this sample report, I hide most of the code; in your report, however, all code used to produce your results must be shown in the submitted HTML file.

First set the seed to be the last four digits of your UIN. Drop any zeros at the beginning; for example, if the last four digits of your UIN are 0268, then use 268 as your seed.

Load required R packages. Two packages are allowed: `class` for KNN classification, and, if needed, `ggplot2` for graphics.

Hide

```
set.seed(1234)
library(class)
library(ggplot2)
```

---

1) Data Generation

We need to sample two-dimensional data $X \in \mathbf{R}^2$ in each class from a mixture of 10 different bivariate Gaussian distributions with uncorrelated components and different means, i.e.,

$$X \mid Y = k,\ Z = l\ \sim\ \mathcal{N}(\mathbf{m}_{kl}, s^2\mathbf{I}_2),$$

where

- $k = 0, 1; l = 1:10$
- $P(Y = k) = 1/2$
- $P(Z = l) = 1/10$.

In other words, given the centers and $Y = k$, X follows a mixture distribution with density function

$$\frac{1}{10}\sum_{l=1}^{10}\left(\frac{1}{\sqrt{2\pi s^2}}\right)^2 e^{-\|\mathbf{x}-\mathbf{m}_{kl}\|^2/(2s^2)}.$$

To sample a data point from a mixture of 10 bivariate Gaussian distributions, we need to repeat the following two steps **for each sample**:

1. randomly select one of the 10 centers;
2. generate a sample from the bivariate Gaussian with the center chosen in step 1.

The R code for Example 2 in "Rcode_Week1_SimulationStudy" is just a compact and efficient way to sample all data points simultaneously.

Let's define two functions: `generateCenters` produces 20 center points which will be used throughout the simulation study; `generateData` returns simulated data sets (*train* and *test*).

Hide

```
# nc  : number of cluster centers
# sdc : standard deviation for generating the cluster centers
#
generateCenters = function(nc, sdc) {
  ####################
  # YOUR CODE
  ####################
  return(list("center0" = m0,
              "center1" = m1))
}

# centers : cluster center points
# ntr     : number of training samples
# nte     : number of test samples
# sdd     : standard deviation for generating the data
#
generateData = function(centers, ntr, nte, sdd) {
  ####################
  # YOUR CODE
  ####################
  return(list("train" = train_set,
              "test" = test_set))
}
```

2) Procedures

For illustration purpose, I demonstrate how to perform the four classification procedures on one pair of training/test sets (stored in `data`). I also plot the corresponding decision boundaries along with the training data; the decision boundary plots are not required for this assignment.

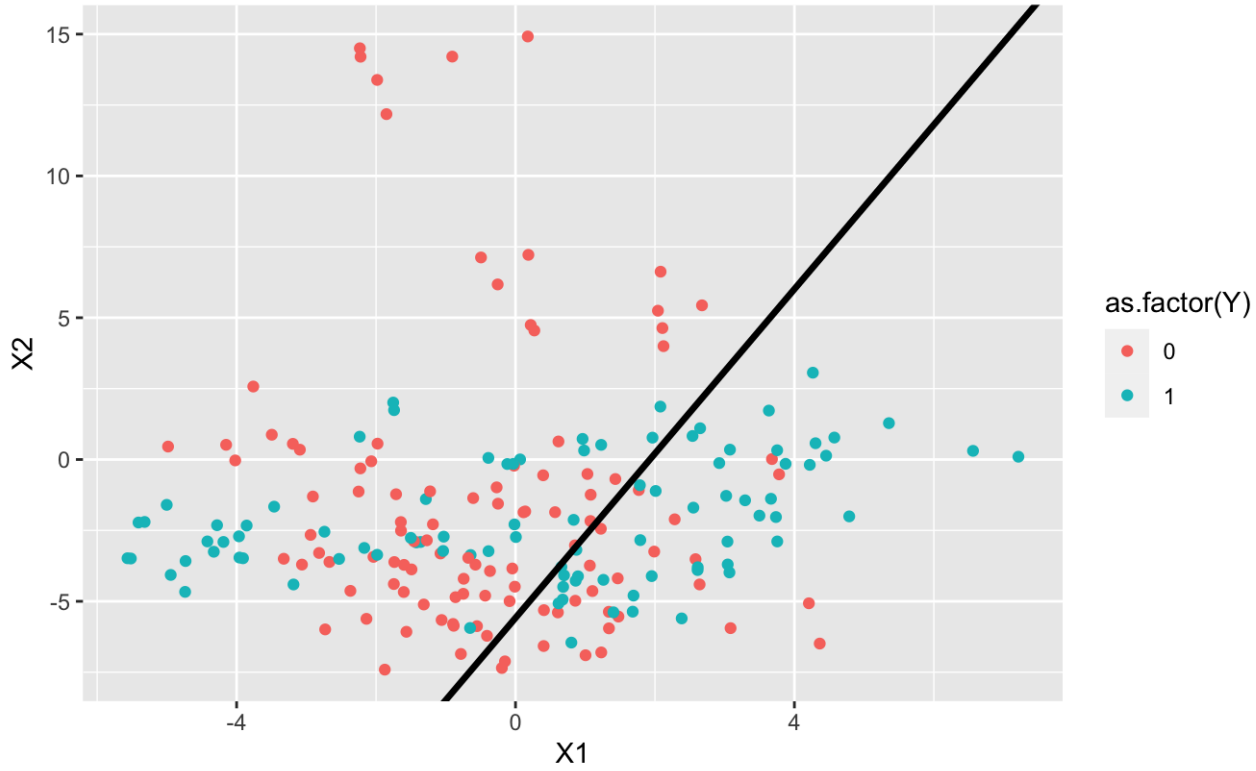
2.a) Linear regression

Fit a linear regression model on the training data and use cut-off value .5 to transform numerical outcomes to binary outcomes. Below are the misclassification rates of a particular pair of training/test data `data` (I store all the outputs in an R object `linearPerf`) and the corresponding decision boundary.

Hide

```
linearPerf
```

```
$model

Call:
lm(formula = Y ~ X1 + X2, data = data$train)

Coefficients:
(Intercept)           X1           X2
    0.44410      0.02901     -0.01001


$trainError
[1] 0.355

$testError
[1] 0.3647
```



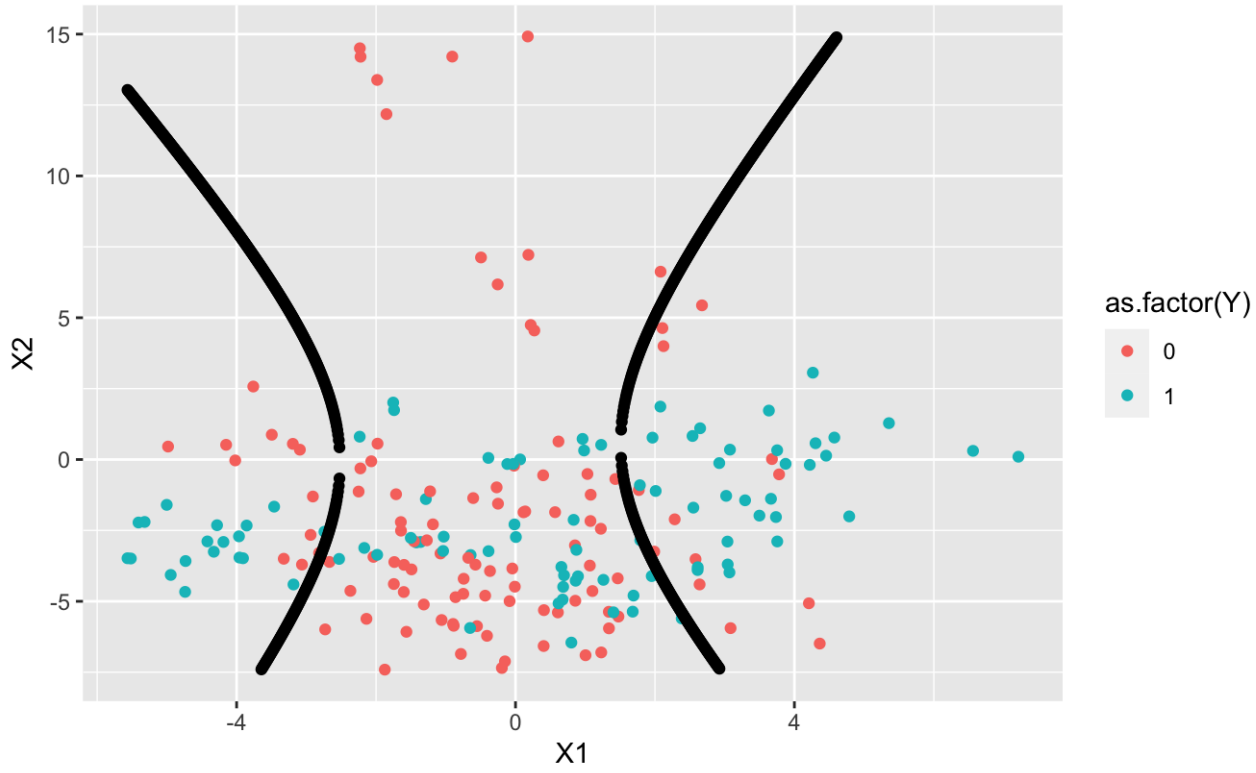
2.b) Quadratic regression

Fit a quardratic regression model on the training data and use cut-off value .5 to transform numerical outcomes to binary outcomes. Below are the misclassification rates (I store all the outputs in an R object `quadraticPerf`) and the corresponding decision boundary.

Hide

```
quadraticPerf
```

```
$model

Call:
lm(formula = Y ~ X1 + X2 + I(X1 * X2) + I(X1^2) + I(X2^2), data = data$tra
in)

Coefficients:
(Intercept)           X1           X2   I(X1 * X2)      I(X1^2)      I(X2^
2)
   0.399578     0.026306     0.001849     0.001020     0.018733    -0.0030
49


$trainError
[1] 0.295

$testError
[1] 0.3324
```



2.c) CV-KNN

How to compute the 10-fold CV error with a particular K value? First, divide the training data equally into ten folds, then compute the prediction error on each fold, using the KNN classifier trained based on the other nine folds.

Specially, in the code below, I set `K = 3` and loop over `runId = 1:10` to compute the CV error. For example, when `runId = 3`, we find the indices of samples in the 3rd fold (stored in `testSetIndex`), then train a KNN model without data in `testSetIndex` and finally form prediction on data in `testSetIndex`.

Note: CV errors are computed only on the training data.

Hide

```
dataSet = data$train[, c("X1", "X2", "Y")]  ## 200-by-3 training data
foldNum = 10
foldSize = floor(nrow(dataSet)/foldNum)
K = 3
error = 0
for(runId in 1:foldNum){
  testSetIndex = ((runId-1)*foldSize + 1):(ifelse(runId == foldNum, nrow(d
ataSet), runId*foldSize))
  trainX = dataSet[-testSetIndex, c('X1', 'X2')]
  trainY = as.factor(dataSet[-testSetIndex, ]$Y)
  testX = dataSet[testSetIndex, c('X1' , 'X2')]
  testY = as.factor(dataSet[testSetIndex, ]$Y)
  predictY = knn(trainX, testX, trainY, K)
  error = error + sum(predictY != testY)
}
error = error / nrow(dataSet)
error
```

In the code above, the 200 training samples are *sequentially* divided into 10 folds. This could be **problematic** if the order of the training data is not random, e.g., all samples with `Y=1` are arranged at the beginning, or if data are arranged by time. It is not an issue here since the order of the training data is indeed random in my data generating function `generateData`. In general, to avoid this problem, one can read `testSetIndex` from a shuttled index set (1 to n) as the following:

Hide

```
myIndex = sample(1 : nrow(dataSet))
...
testSetIndex = ((runId-1)*foldSize + 1):(ifelse(runId == foldNum, nrow(dat
aSet),
                                                runId*foldSize))
testSetIndex = myIndex[testSetIndex]
```

I put the code above in a function `cvKNNAveErrorRate`. Then wrote a function `cvKNN` that returns the best K value based on 10-fold CV errors. Specifically, I store the possible K values in vector `KVector`, and then store the corresponding CV errors in

`cvKNNAveErrorRates` (which, for example, can be computed using a `for` -loop, but I used `sapply` here).

Note: it is possible that there are multiple K values that give the smallest CV error; when this happens, I pick the largest one among them, since the larger the K value, the simplier the model.

Hide

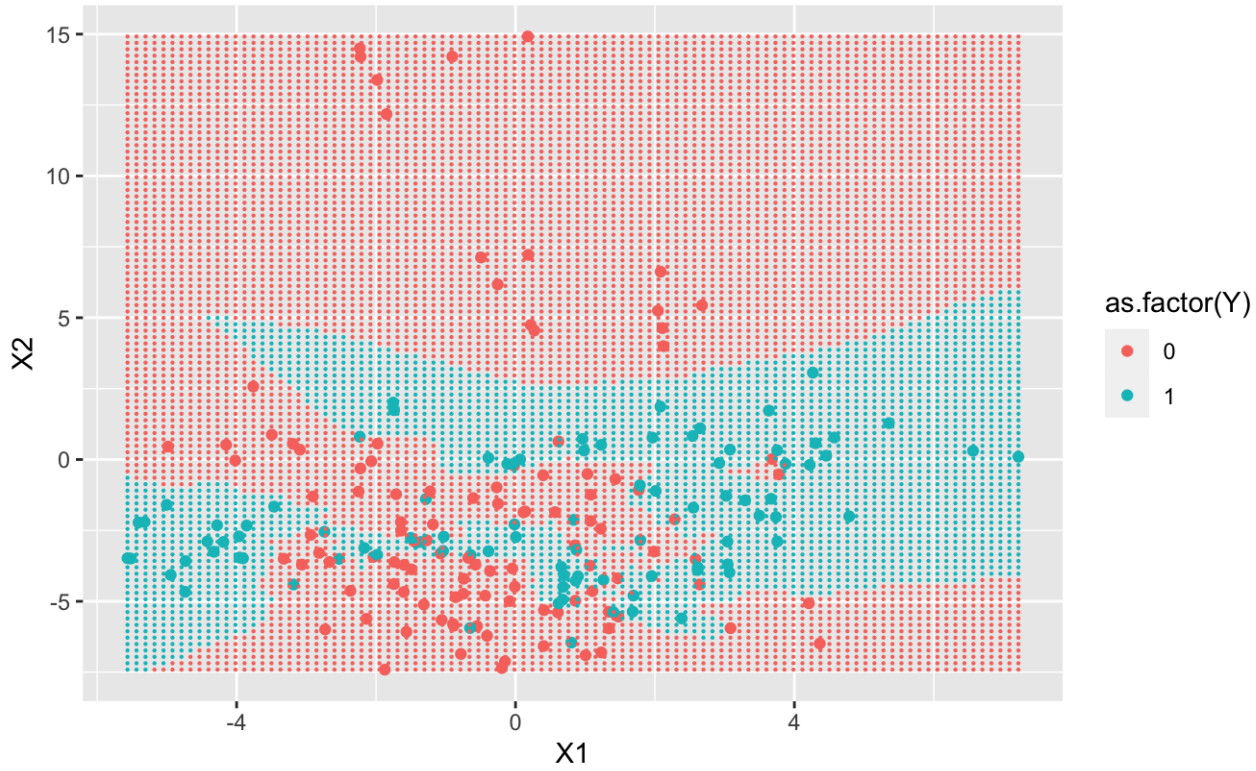
```
cvKNN = function(dataSet, foldNum) {
  foldSize = floor(nrow(dataSet)/foldNum)
  KVector = seq(1, (nrow(dataSet) - foldSize), 2)
  cvKNNAveErrorRates = sapply(KVector, cvKNNAveErrorRate, dataSet, foldSiz
e,
                              foldNum)
  result = list()
  result$bestK = max(KVector[cvKNNAveErrorRates == min(cvKNNAveErrorRate
s)])
  result$cvError = cvKNNAveErrorRates[result$bestK]
  result
}
```

Report the chosen K and the misclassification rates on training/test and draw the corresponding decision boundary.

Hide

```
KNNResult
```

```
$cvBestK
[1] 3

$cvError
[1] 0.26

$trainError
[1] 0.12

$testError
[1] 0.2611
```



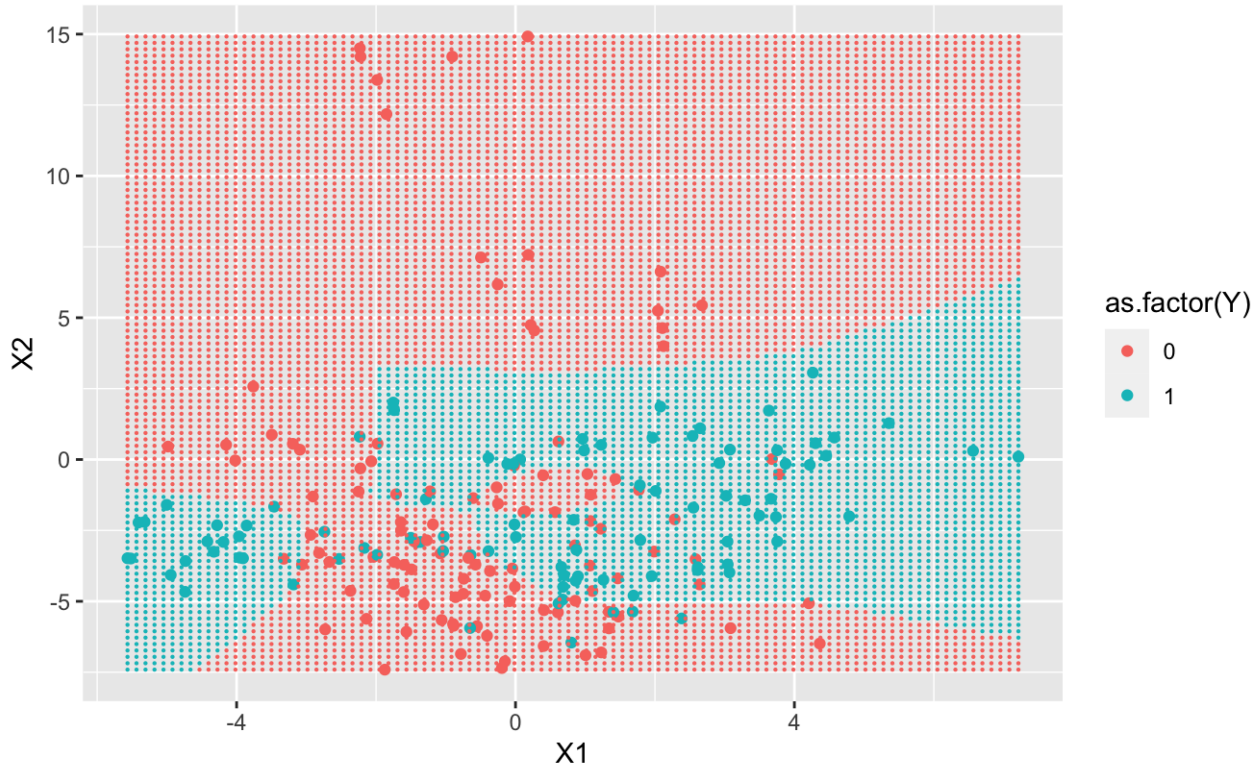
2.d) Bayes Rule

When calculating the misclassification rates using the Bayes Rule, we need to repeatedly evaluate the following ratio, which I define a function to compute.

$$\text{mixnorm\_ratio} = \frac{\sum_{l=1}^{10} \exp\left\{-\frac{1}{2s^2}\|m_{1l} - x\|^2\right\}}{\sum_{l=1}^{10} \exp\left\{-\frac{1}{2s^2}\|m_{0l} - x\|^2\right\}}$$

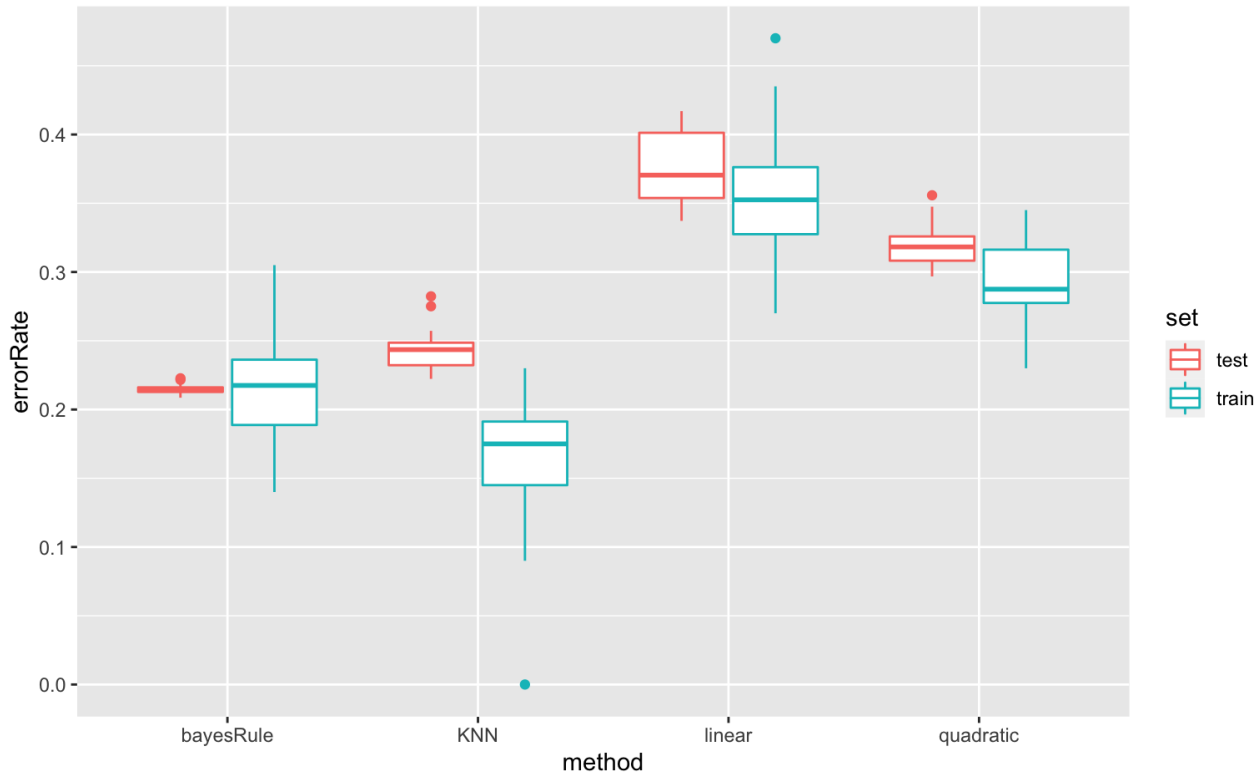
Hide

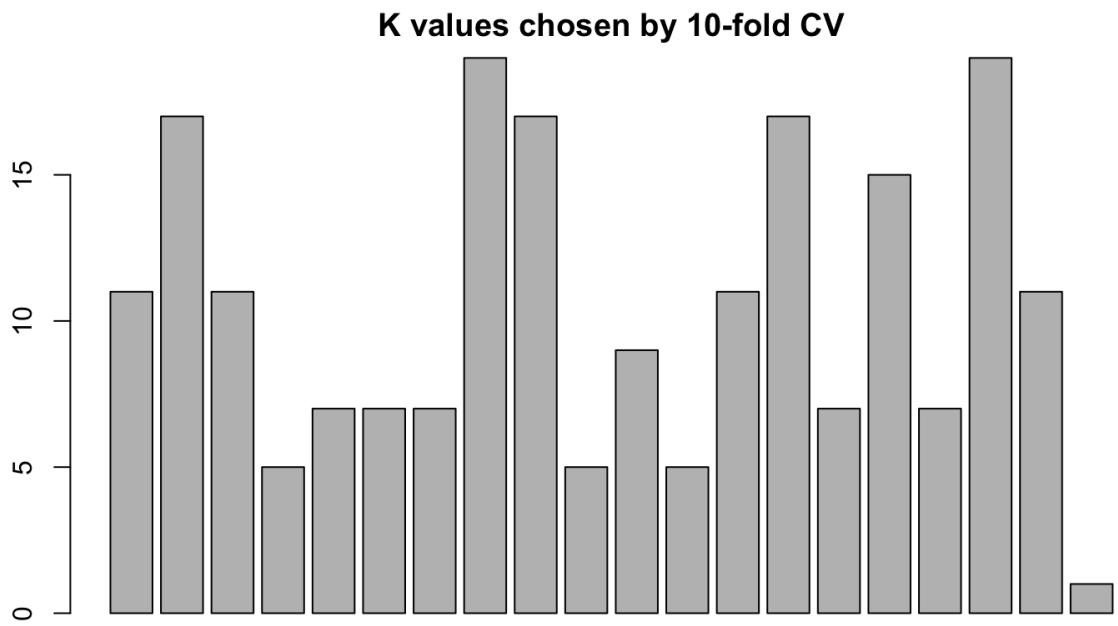
```
mixnorm = function(x, centers0, centers1, s){
  ## return the density ratio for a point x, where each
  ## density is a mixture of normal with multiple components
  d1 = sum(exp(-apply((t(centers1) - x)^2, 2, sum) / (2 * s^2)))
  d0 = sum(exp(-apply((t(centers0) - x)^2, 2, sum) / (2 * s^2)))

  return (d1 / d0)
}
```



3) Simulations

We are ready to perform our simulations. Repeat simulation 20 times (using the same 20 centers) and compare the performance for all methods.





Report the mean and sd of selected K values.

Hide

```
mean(KNNCvBestK)
```

```
[1] 10.4
```

Hide

```
sd(KNNCvBestK)
```

```
[1] 5.31532
```