PSL (F20) Coding Assignment 1

Zihe Liu

08/20/2020

In this sample report, I hide most of my 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 and load required R packages. You are only allowed to use two packages `class` for KNN classification, and, if needed, `ggplot2` for graphics.

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

For illustration purpose, I start with a description on how to generate data and how to perform the four classification procedures on one pair of training/test sets. In your report, you do NOT need to include (1) to (4); you only need to include part (5).

1) Generate the training and test data

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. You can use the same 20 centers for all simulations or each time generate different 20 centers; it's up to you.

In my code, `data` contains both the training and sets data sets.

```
data = generateData(M, s, n, 200, 10000)
```

```
names(data)
```

```
## [1] "train" "test"
```

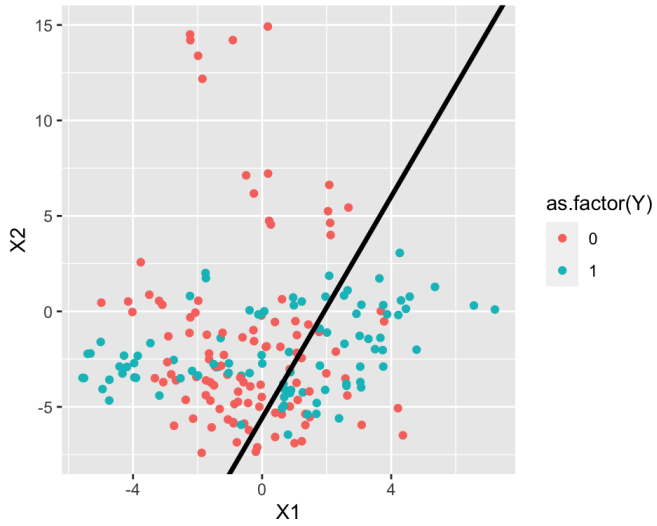
2) Classification based on linear regression

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

```
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
```

```
linearModel = linearPerf$model
ggplot(data = data$train, aes(x = X1, y = X2, color = as.factor(Y))) +
    geom_point() + geom_abline(slope = -linearModel$coefficients[2]/linearModel$coefficients[3],
    intercept = (0.5 - linearModel$coefficients[1])/linearModel$coefficients[3],
    size = 1.2)
```

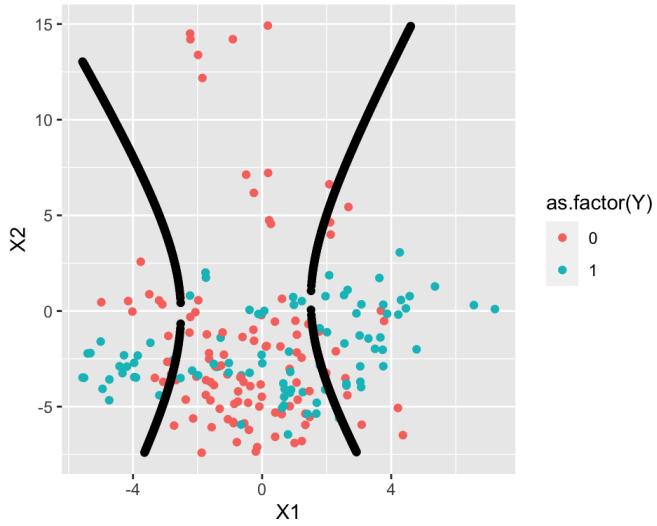


3) Classification based on quadratic regression

Fit a quardratic regression model on the training data and use cut-off value .5 to transform numerical outcomes to binary outcomes. Below is the misclassification rates of a particular pair of training/test datasets (I store all the outputs in an R object `quadraticPerf`). I also draw the corresponding decision boundary along with the training data.

```
quadraticPerf
```

```
## $model
##
## Call:
## lm(formula = Y ~ X1 + X2 + I(X1 * X2) + I(X1^2) + I(X2^2), data = data$train)
##
## Coefficients:
## (Intercept)           X1           X2   I(X1 * X2)      I(X1^2)      I(X2^2)
##    0.399578     0.026306     0.001849     0.001020     0.018733    -0.003049
##
##
## $trainError
## [1] 0.295
##
## $testError
## [1] 0.3324
```



4) Classification based on KNN with K chosen by CV.

How to compute the 10-fold CV with a particular K value? First, divide the training data equally into ten folds, then compute the prediction error on each fold, using the trained KNN classifier 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 out the index of samples in the 3rd fold, i.e., `testSetIndex`, then train a kNN model without data in `testSetIndex` and form prediction on data in `testSetIndex`.

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

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

```
## [1] 0.24
```

In the code above, the 200 training samples are **sequentially** divided into 10 folds. As a student pointed out on Piazza, this could be dangerous 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:

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

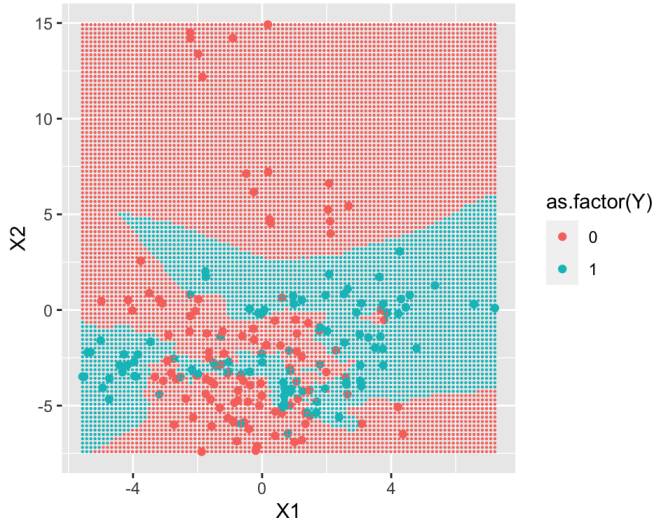
Note: it is possible that there are multiple K values that give the smallest CV error; when this happens, I pick the largest one among them, since the larger the K value, the simplier the model.

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

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

```
KNNResult
```

```
## $cvBestK
## [1] 3
##
## $cvError
## [1] 0.26
##
## $trainError
## [1] 0.12
##
## $testError
## [1] 0.2611
```

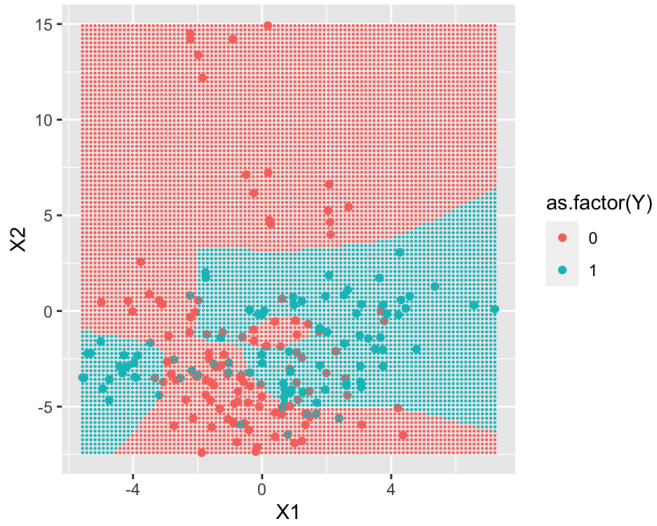


4) Classification based on the Bayes Rule

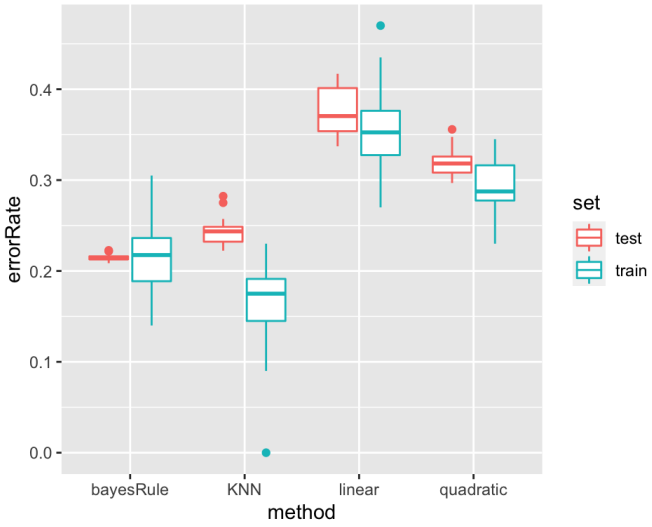
Report the misclassification rates on training/test and draw the corresponding decision boundary.

```
BayesRulePerf
```

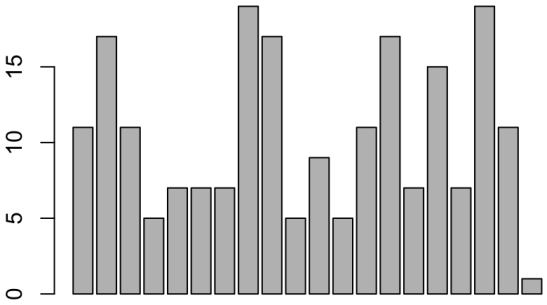
```
## $trainError
## [1] 0.23
##
## $testError
## [1] 0.2174
```



5) Finally, repeat simulation 20 times and compare the performance for all methods.



best K selected by 10 fold cross validation for KN



Report the mean and sd of selected K values.

```
mean(KNNCvBestK)
```

```
## [1] 10.4
```

```
sd(KNNCvBestK)
```

```
## [1] 5.31532
```