STAT 542: Homework 1

Shashi Roshan (sroshan2)

Feb 3, 2020

Contents

Question 1 [50 Points] KNN

For this question, you **cannot** use (load) any additional `R` package. Complete the following steps.

a. [15 points] Generate the 1000 training data from the following multivaraite Normal distribution:

$$X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu} = (0, 1, 2)^T$ and $\boldsymbol{\Sigma}$ is a covariance matrix with diagnal elements 1 and off-diagnal elements 0.5. Show with a rigorous derivation that your code will indeed generate observations from this distribution. Then, Y is generated from

$$Y = 0.25 \times X_1 + 0.5 \times X_2 + X_3 + \epsilon,$$

with i.i.d. standard normal ϵ. Set a random seed to 1. For the covariates, output the estimated mean and covariance matrix your data. Plot the outcome against your third variable $X^{(3)}$ and center the figure.

```r
# part a
set.seed(1)
mean = c(0, 1, 2)
cov_matrix = matrix(c(1, 0.5, 0.5, 0.5, 1, 0.5, 0.5, 0.5, 1),
                    nrow = 3,
                    ncol = 3)

correlated_gaussian_distribution = function(mean, cov_matrix) {
  C = chol(cov_matrix)
  Z = rnorm(length(mean), mean = 0, sd = 1)
  return(mean + t(C) %*% Z)
}

n_row = 1000
n_col = 3
```

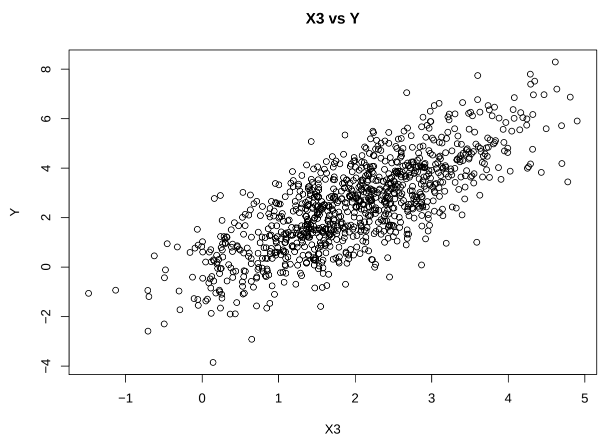
```r
X = matrix(0, nrow = n_row, ncol = n_col)

for (i in 1:n_row) {
  X[i,] = correlated_gaussian_distribution(mean, cov_matrix)
}

# estimated mean of data
c(mean(X[, 1]), mean(X[, 2]), mean(X[, 3]))
```

```
## [1] 0.0123461 1.0108998 1.9829230
```

```r
# estimated covariance matrix of data
cov(X)
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0762681 0.5332667 0.5253742
## [2,] 0.5332667 1.0487646 0.5442602
## [3,] 0.5253742 0.5442602 1.0871155
```

```r
set.seed(1)
epsilon = rnorm(n_row, mean = 0, sd = 1)

Y = (0.25 * X[, 1]) + (0.5 * X[, 2]) + X[, 3] + epsilon

# Plot of X3 vs Y
plot(X[, 3],
     Y,
     main = 'X3 vs Y',
     xlab = 'X3',
     ylab = 'Y')
```



Derivation to show that the code will generate observations from the specified distribution :

Let $CC^t = \Sigma$.

We can generate the data using

$$X = \mu + CZ$$

where $\boldsymbol{\mu} = (0, 1, 2)^T$, and $Z \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$
Hence,

$$mean(X) = mean(\mu + CZ) = mean(\mu) + mean(CZ) = \mu + C.mean(Z) = \mu + C.0 = \mu$$

$$Cov(X) = Cov(\mu + CZ) = 0 + Cov(CZ) = C.Cov(Z).C^t = CIC^t = CC^t = \Sigma$$

Since Z follows a Normal distribtuon,

$$X \sim N(\mu, \Sigma)$$

b. [15 Points] Write a function `myknn(xtest, xtrain, ytrain, k)` that fits a KNN model and predict multiple target points `xtest`. Here `xtrain` is the training dataset covariate value, `ytrain` is the training data outcome, and `k` is the number of nearest neighbors. Use the ℓ_1 distance to evaluate the closeness between any two points. Please note that for this question, you can only use the base R (hence no additional package).

```r
# part b
n_train = 400

x_train = X[1:n_train,]
y_train = Y[1:n_train]

x_test = X[(n_train + 1):n_row,]
y_test = Y[(n_train + 1):n_row]

# KNN
get_l1_norm = function(a, b) {
  return(sum(abs(a - b)))
}

get_x_test_yhat = function(x_test_row, xtrain, ytrain, k) {
  # for each sample in x_train
  l1_norm = apply(
    xtrain,
    MARGIN = 1,
    FUN = function(x_train_row)
      get_l1_norm(x_train_row, x_test_row)
  )

  # find k nearest samples in x_train (sort one vector based on another)
  k_closest_neighbors = order(l1_norm)[1:k]

  # find the mean of those k nearest neighbors
  y_hat = mean(ytrain[k_closest_neighbors])

  return(y_hat)
}

myknn = function(xtest, xtrain, ytrain, k) {
  # for each row in x_test
  y_hat = apply(
    x_test,
    MARGIN = 1,
    FUN = function(x_test_row)
      get_x_test_yhat(x_test_row, xtrain, ytrain, k)
  )

  return(y_hat)
}
```

c. [10 Points] Use the first 400 observations as the training data and the rest as testing data. Predict the Y values using your KNN function with `k = 5`. Evaluate the prediction accuracy using mean squared error

$$\frac{1}{N}\sum_i (y_i - \widehat{y}_i)^2$$

```r
# part c
k = 5
y_hat = myknn(x_test, x_train, y_train, k)

# MSE
sum((y_test - y_hat) ^ 2) / length(y_test) # 1.351289
```

```
## [1] 1.351289
```

The Mean Squared Error is low.

d. [10 Points] Consider k being 1, 2, ..., 10, 20, 30, ..., 100, 200, 300, 400, 500. Use the degrees of freedom as the horizontal axis. Demonstrate your results in a single, easily interpretable figure with proper legends. What is your optimal tuning parameter?

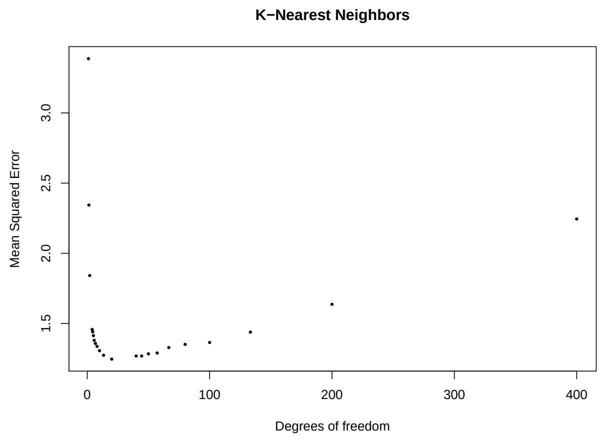
```r
# part d
k_possible = c(seq(1, 10, 1), seq(20, 100, 10), seq(200, 400, 100))

mse_array = c()

for (k_value in k_possible) {
  y_hat = myknn(x_test, x_train, y_train, k_value)
  mse = sum((y_test - y_hat) ^ 2) / length(y_test)
  mse_array = c(mse_array, mse)
}

dof = n_train / k_possible

# MSE
min(mse_array) # 1.246254
```

```
## [1] 1.246254
```

```r
# Optimal value of K
k_possible[order(mse_array)][1] # 20
```

```
## [1] 20
```

```r
# dof = 20

plot(
  dof,
  mse_array,
  pch = 19,
  cex = 0.4,
  xlab = "Degrees of freedom",
  ylab = "Mean Squared Error",
  main = "K-Nearest Neighbors"
)
```



e. [10 Points] Fit a linear regression to this data (with the same train/test split). What is the degree of freedom of this model? Is it better or worse than your optimal KNN? Why?

```r
# part e
df_train = data.frame(X1 = x_train[, 1],
                      X2  = x_train[, 2],
                      X3  = x_train[, 3],
                      Y = y_train)

linear_model = lm(Y ~ X1 + X2 + X3, data = df_train)

y_hat_lm = predict(linear_model, newdata = data.frame(X1 = x_test[, 1], X2  = x_test[, 2], X3  = x_test

# MSE
sum((y_hat_lm - y_test) ^ 2) / length(y_test) # 1.174045
```

```
## [1] 1.174045
```

Conclusion :
The degrees of freedom of the Linear regression model is 4.
Linear model is performing better than KNN model, according to Test MSE. The data was generated in a linear way, hence the Linear model is able to capture it.
The reason might be that Linear model has less complexity as compared to KNN model. Also, the degrees of freedom of Linear model (4) is less than KNN's degrees of freedom (20), hence the Linear model is less complex, and is performing well. Hence, we choose the Linear model.

f. [10 Points] Try a new model and re-generate just your Y values:

$$Y = 0.25 \times X_1 + 0.5 \times X_2 + X_3^2 + \epsilon$$

and redo your analysis of the KNN and linear regression. Which one is better? Why?

```
# part f
set.seed(1)
epsilon = rnorm(1000, mean = 0, sd = 1)

Y_new = (0.25 * X[, 1]) + (0.5 * X[, 2]) + I(X[, 3] ^ 2) + epsilon

n_train = 400

x_train = X[1:n_train,]
y_train = Y_new[1:n_train]

x_test = X[(n_train + 1):n_row,]
y_test = Y_new[(n_train + 1):n_row]

# KNN
k_possible = c(seq(1, 10, 1), seq(20, 100, 10), seq(200, 400, 100))

mse_array = c()

for (k_value in k_possible) {
  y_hat = myknn(x_test, x_train, y_train, k_value)
  mse = sum((y_test - y_hat) ^ 2) / length(y_test)
  mse_array = c(mse_array, mse)
}

dof = n_train / k_possible

# MSE
min(mse_array) # 2.045156
```

```
## [1] 2.045156
```

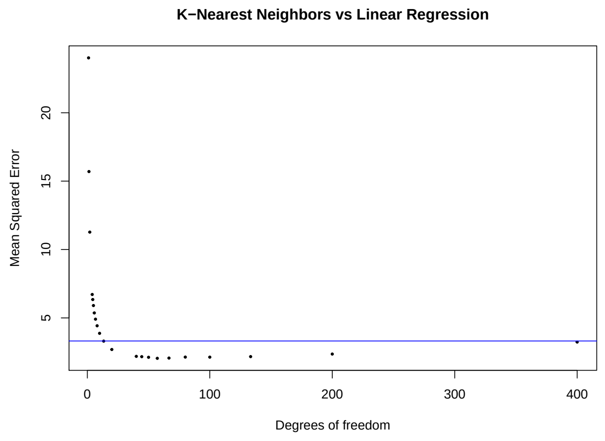
```
# Optimal K
k_possible[order(mse_array)][1] # 7
```

```
## [1] 7
```

```r
# Linear model
df_train = data.frame(X1 = x_train[, 1],
                      X2  = x_train[, 2],
                      X3  = x_train[, 3],
                      Y = y_train)

linear_model = lm(Y ~ X1 + X2 + X3, data = df_train)

y_hat_lm = predict(linear_model, newdata = data.frame(X1 = x_test[, 1], X2  = x_test[, 2], X3  = x_test

# MSE
sum((y_hat_lm - y_test) ^ 2) / length(y_test) # 3.31277
```

```
## [1] 3.31277
```

```r
plot(
  dof,
  mse_array,
  pch = 19,
  cex = 0.4,
  xlab = "Degrees of freedom",
  ylab = "Mean Squared Error",
  main = "K-Nearest Neighbors vs Linear Regression"
)
abline(h = sum((y_hat_lm - y_test) ^ 2) / length(y_test), col = "blue")
```



Conclusion :
In the plot, the blue line is the mean squared error of the linear model.
Linear model is not able to capture the non-linear relationship using which the data was generated.
Hence, KNN is performing well, since it can capture non-linear relationship in the data.

Question 2 [15 Points] Curse of Dimensionality

Following the previous analysis of the nonlinear model, let's consider a high-dimensional setting. For this problem, use the standard Euclidean distance. Also, keep the Y values the same as part f. in the previous question. We consider two cases that both generate an additional set of 97 covariates:

- Generate another 97-dimensional covariate with all independent Gaussian entries

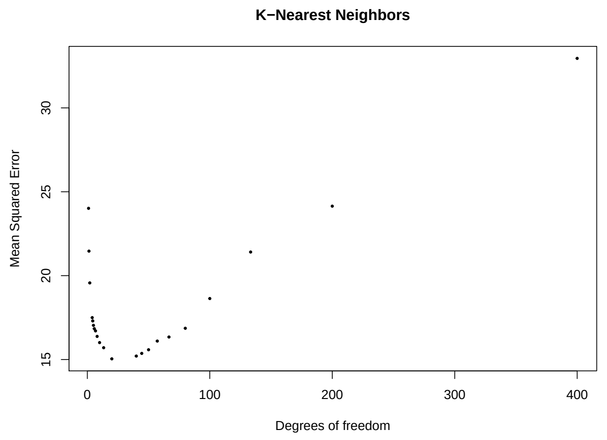
```r
# part a
library(caret)

n_col = 97

set.seed(1)
X_temp = matrix(rnorm(97 * 1000, mean = 0, sd = 1), nrow = 1000, ncol = 97)

X_2_a = cbind(X, X_temp)

# mean correlation of features
corr_array = c()
for (idx1 in 1:99) {
  for (idx2 in (idx1+1):100) {
    corr_array = c(corr_array, cor(X_2_a[idx1,], X_2_a[idx2,]))
  }
}
# mean(corr_array)

n_train = 400

x_train = X_2_a[1:n_train,]
y_train = Y_new[1:n_train]

x_test = X_2_a[(n_train + 1):n_row,]
y_test = Y_new[(n_train + 1):n_row]

k_possible = c(seq(1, 10, 1), seq(20, 100, 10), seq(200, 400, 100))

mse_array = c()

for (k_value in k_possible) {
  knn_model = knnreg(x_train, y_train, k_value)
  y_hat = predict(knn_model, x_test)
  mse = sum((y_test - y_hat) ^ 2) / length(y_test)
  mse_array = c(mse_array, mse)
}

dof = n_train / k_possible

# MSE
min(mse_array) # 15.04395
```

```
## [1] 15.04395
```

```r
# Optimal K
k_possible[order(mse_array)][1] # 20
```

```
## [1] 20
```

```r
plot(
  dof,
  mse_array,
  pch = 19,
  cex = 0.4,
  xlab = "Degrees of freedom",
  ylab = "Mean Squared Error",
  main = "K-Nearest Neighbors"
)
```



- Generate another 97-dimensional covariate using the formula $X^T A$, where X is the original 3-dimensional vector, and A is a 3×97 dimensional matrix that remains the same for all observations. Generate A using i.i.d. uniform entries. Intuitively present your results and comment on your findings.

```r
# part b
set.seed(1)
A = matrix(runif(97 * 3, min = 0, max = 1), nrow = 3, ncol = 97)

X_temp = X %*% A

X_2_b = cbind(X, X_temp)

# mean correlation of features
corr_array = c()
for (idx1 in 1:99) {
  for (idx2 in (idx1+1):100) {
    corr_array = c(corr_array, cor(X_2_b[idx1,], X_2_b[idx2,]))
  }
}
# mean(corr_array)

n_train = 400

x_train = X_2_b[1:n_train,]
y_train = Y_new[1:n_train]

x_test = X_2_b[(n_train + 1):n_row,]
```

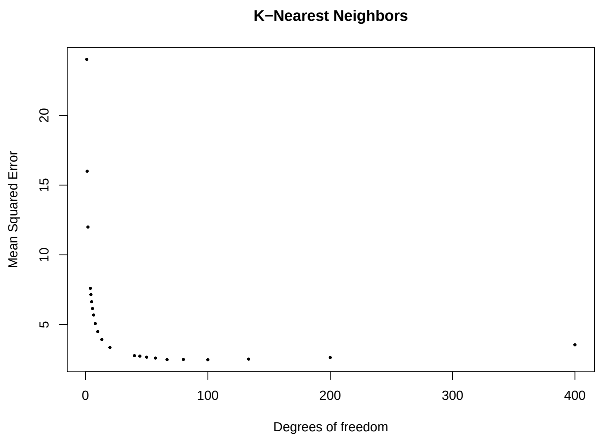
```
y_test = Y_new[(n_train + 1):n_row]

k_possible = c(seq(1, 10, 1), seq(20, 100, 10), seq(200, 400, 100))

mse_array = c()

for (k_value in k_possible) {
  knn_model = knnreg(x_train, y_train, k_value)
  y_hat = predict(knn_model, x_test)
  mse = sum((y_test - y_hat) ^ 2) / length(y_test)
  mse_array = c(mse_array, mse)
}

dof = n_train / k_possible

# MSE
min(mse_array) # 2.479886
```

```
## [1] 2.479886
```

```
# Optimal K
k_possible[order(mse_array)][1] # 4
```

```
## [1] 4
```

```
plot(
  dof,
  mse_array,
  pch = 19,
  cex = 0.4,
  xlab = "Degrees of freedom",
  ylab = "Mean Squared Error",
  main = "K-Nearest Neighbors"
)
```



Conclusion :
In the first case, there is no correlation in the predictors. Hence, this data is high dimensional.

In the second case, there is high correlation in the predictors. Hence, this data is not truely high dimensional data.
In the case of high dimensional data, there is bias introduced in the model. For example, while using KNN in unit cube, to capture 1% or 10% of the data to form a local average, we must cover 63% or 80% of the range of each input variable. Such neighborhoods are no longer local.
Hence, the model performs well in second case (where data is not high dimensional), whereas in first case, the model does not perform well due to curse of dimensionality.

Question 3 [15 Points] Classification of Handwritten Digit Data

Use the Handwritten Digit Data from the `ElemStatLearn` package. Combine the train and test data defined in the package. For this question, you are asked to perform 10-fold cross-validation. Choose a grid of 10 different k values reasonably and select the best tuning of k. Select the first observation, and find and plot the closest k neighbors (in the rest of the dataset) based on your optimal k. Is this observation correctly classified? Comment on the overall performence of KNN on this dataset.

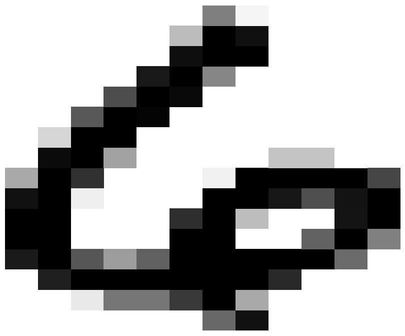
```r
library(ElemStatLearn)

data("zip.train")
data("zip.test")

train_test_combined = rbind(zip.train, zip.test)

df_train_test_combined = as.data.frame(train_test_combined)

library(caret)

df_train_test_combined[, 1] = as.factor(df_train_test_combined[, 1])


set.seed(1)
knn_model = train(
  V1 ~ .,
  data = df_train_test_combined,
  method = 'knn',
  trControl = trainControl(method = 'cv',
                           number = 10),
  tuneGrid = expand.grid(k = seq(1, 50, 5))
)

# K = 1 is the optimal choice of K for KNN
knn_model
```

```
## k-Nearest Neighbors
##
## 9298 samples
##  256 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8367, 8369, 8368, 8367, 8369, 8369, ...
## Resampling results across tuning parameters:
```

```
##
##   k   Accuracy   Kappa
##    1  0.9690225  0.9652707
##    6  0.9624649  0.9579105
##   11  0.9563338  0.9510274
##   16  0.9502043  0.9441427
##   21  0.9445041  0.9377408
##   26  0.9410618  0.9338767
##   31  0.9366535  0.9289226
##   36  0.9326752  0.9244525
##   41  0.9307392  0.9222736
##   46  0.9272968  0.9184032
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
# For the first observation, find and plot the closest k neighbors based on optimal k
library(FNN)

test_data = df_train_test_combined[1, 2:ncol(df_train_test_combined)]
train_data = df_train_test_combined[, 2:ncol(df_train_test_combined)]

# Since we need to find 1 nearest neighbor in "rest of the data", taking K = 2, and ignoring the index
k_value = 2

knn_model = knn(
  train = train_data,
  test = test_data,
  cl = df_train_test_combined[, 1],
  k = k_value
)

indices = attr(knn_model, "nn.index")

for (idx in indices) {
  if (idx == 1){
    next
  }
  print(idx)
  image(
    zip2image(train_test_combined, idx),
    col = gray(256:0 / 256),
    zlim = c(0, 1),
    xlab = "",
    ylab = "",
    axes = FALSE
  )
  cat('\n')
}
```

```
## [1] 1357
## [1] "digit  6  taken"
```



Conclusion :
In this case, K = 1 is the optimal choice of K for KNN.
The test observation is correctly classified using 1NN.

Overall Performance of the model :
The cross validation accuracy is 96.9 %, for K = 1.
This is because the data has very low noise. If clusters will be formed using this data, the clusters will be very distant from each other (without overlap). Hence, 1NN will perform very well on this data set. Additionally, it has very low bias.