HW1_dbagchi2

Diptendra Nath Bagchi (dbagchi2@illinois.edu)

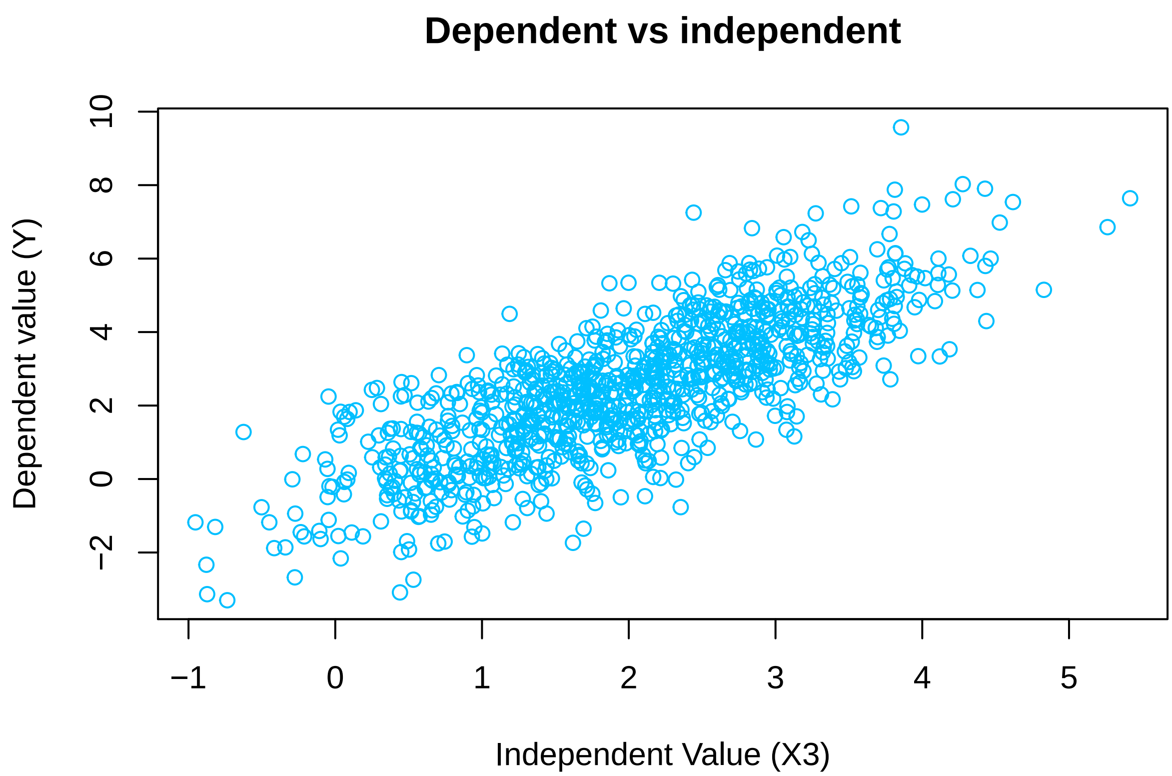
03 February, 2020

Question 1

a) Multivariate Gaussian Simulation

```
set.seed(1)
epsilon = rnorm(n = 1000, mean = 0, sd = 1)
mvr_generation = function(mu, sigma) {
  no_of_variates = length(mu)
  lower_tri = t(chol(sigma))
  std_random = rnorm(no_of_variates)
  (lower_tri %*% std_random + mu)
}
mu = c(0, 1, 2)
sigma = rbind(c(1.0, 0.5, 0.5),
              c(0.5, 1.0, 0.5),
              c(0.5, 0.5, 1.0))
multi_matrix = matrix(0, nrow = 1000, ncol = 3)
for (i in 1:1000) {
  multi_matrix[i, ] = mvr_generation(mu = mu, sigma = sigma)
}

# y is generated as per
y = 0.25 * multi_matrix[, 1] + 0.5 * multi_matrix[, 2] + multi_matrix[, 3] + epsilon
```

```
plot(x = multi_matrix[, 3], y = y,
    main = "Dependent vs independent",
    xlab = "Independent Value (X3)",
    ylab = "Dependent value (Y)",
    col = "deepskyblue")
```



Proof:

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

b) Function to implement own K-Nearest Neighbours

```
myknn = function(xtest, xtrain, ytrain, k){
  n_test = nrow(xtest)
  n_pred = numeric(n_test)
  iter = seq(1, n_test)
  for(i in iter){
    xtest_i = xtest[i, ]
    distance <- rowSums(abs(sweep(xtrain, 2, xtest_i)))
    dist_y = cbind(distance, ytrain)
    min_dist = dist_y[order(dist_y[, 1], decreasing=FALSE),]
    n_pred[i]=sum(min_dist[1:k, 2]) / k
  }
  n_pred
}
```

c) Calculating the mse from KNN for k = 5

```
# Divinding the data into training and testing data
xtrain = multi_matrix[1:400, ]
ytrain = y[1:400]
xtest = multi_matrix[401:nrow(multi_matrix), ]
```

```r
ytest = y[401:length(y)]
knn_pred = myknn(xtest, xtrain, ytrain, 5)
```

```r
calc_mse = function(predicted, actual) {
  mean((actual - predicted) ^ 2)
}
```

```r
calc_mse(knn_pred, ytest)
```

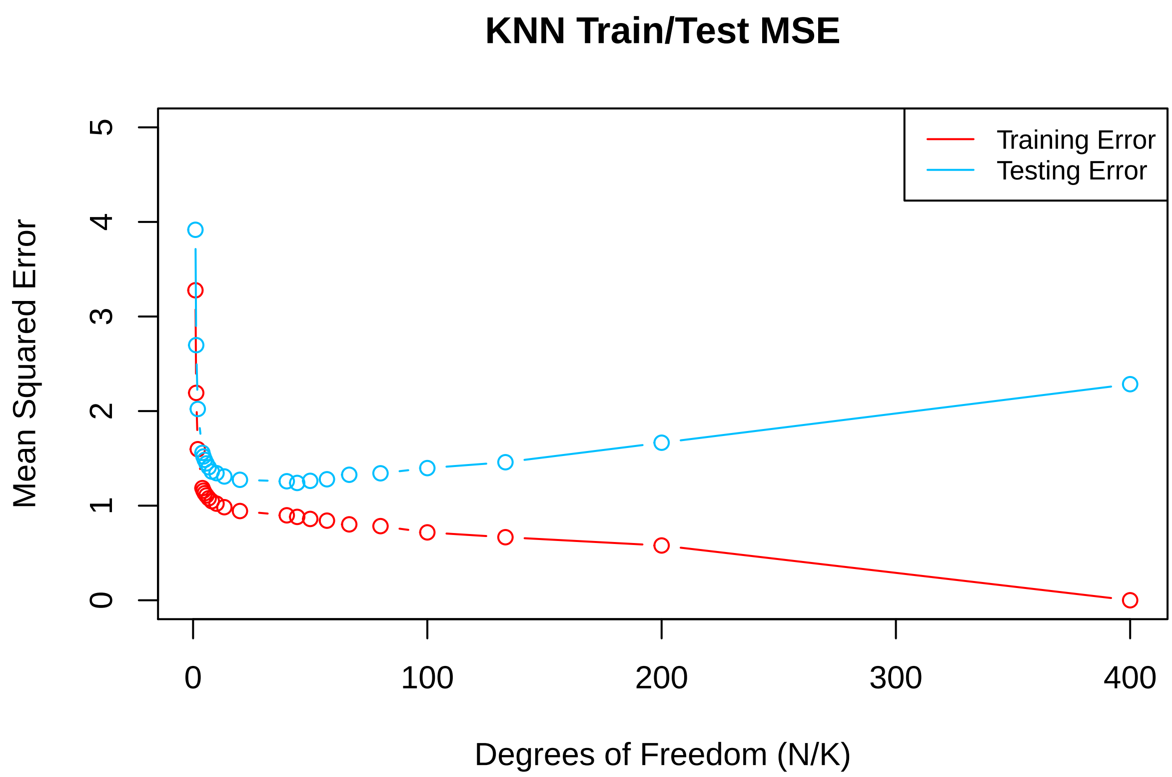
```
## [1] 1.342291
```

The mean squared error is 1.34.

d) What is the optimal tuning parameter?

```r
k = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400)

knn_test_pred = lapply(k, myknn, xtest = xtest, xtrain = xtrain, ytrain = ytrain)

mse_test = sapply(knn_test_pred, calc_mse, actual = ytest)

knn_train_pred = lapply(k, myknn, xtest = xtrain, xtrain = xtrain, ytrain = ytrain)

mse_train = sapply(knn_train_pred, calc_mse, actual = ytrain)
```

```r
dof = 400 / k
plot(dof, mse_train, type = "b", col = "red",
     ylim = c(0, 5), main = "KNN Train/Test MSE",
     xlab = "Degrees of Freedom (N/K)",
     ylab = "Mean Squared Error"
    )
lines(dof, mse_test, col = "deepskyblue", type = "b")
legend("topright", legend=c("Training Error", "Testing Error"),
       col=c("red", "deepskyblue"), lty=1:1, cex=0.8)
```



The optimal tuning parameter is `r k[which.min(mse_test)]`.

e) Linear Regression

```
train_df = data.frame("x" = xtrain, "y" = ytrain)
lmod = lm(y ~ x.1 + x.2 + x.3, data = train_df)
lmod_pred = predict(object = lmod, data.frame("x" = xtest))
lmod_mse = calc_mse(predicted = lmod_pred, ytest)
```

Degress of freedom of linear model is p which is much less than the dof of KNN $400/9 \sim 45$ - which makes KNN a complex model than the linear regression. Also, as the dependent variable is defined as a linear combination of the independent variables - which also makes the linear model a better choice. The mse of linear regression is 1.1688589.

f) Try a new model

```
y = 0.25 * multi_matrix[, 1] + 0.5 * multi_matrix[, 2] + (multi_matrix[, 3] ^ 2) + epsilon

xtrain = multi_matrix[1:400, ]
ytrain = y[1:400]
xtest = multi_matrix[401:nrow(multi_matrix), ]
ytest = y[401:length(y)]
```

```
k = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400)

knn_test_pred = lapply(k, myknn, xtest = xtest, xtrain = xtrain, ytrain = ytrain)

mse_test = sapply(knn_test_pred, calc_mse, actual = ytest)
```

```
knn_train_pred = lapply(k, myknn, xtest = xtrain, xtrain = xtrain, ytrain = ytrain)

mse_train = sapply(knn_train_pred, calc_mse, actual = ytrain)
```

```
train_df = data.frame("x" = xtrain, "y" = ytrain)
lmod = lm(y ~ x.1 + x.2 + x.3, data = train_df)
lmod_pred = predict(object = lmod, data.frame("x" = xtest))
lmod_mse = calc_mse(predicted = lmod_pred, ytest)
```

KNN model is performing better than K-Nearest Neighbours by a significant margin. The test mse of knn is 2.377 and the test mse of linear model is 3.5763353. Also the specification of the model is incorrect as there is square terms which are missing from the linear model but in KNN it does not effect much because of it non-parametric nature. Hence, in this case KNN performes better than the linear model.

Question 2

```
# part 1
set.seed(1)
x = seq(from = 1, to = 97, by = 1)
add_97 = lapply(x, function(x) {rnorm(1000, mean = 0, sd = 1)})
add_97_type_cast = matrix(unlist(add_97), ncol = 97)

x_matrix_a = cbind(multi_matrix, add_97_type_cast)
y2_a = y
```

```
# part 2
set.seed(1)
matrix_a_temp = lapply(x, function(x) {runif(3, min = 0, max = 1)})
matrix_a_temp = matrix(unlist(matrix_a_temp), ncol = 97)
x_trans_a = multi_matrix %*% matrix_a_temp
x_matrix_b = cbind(multi_matrix, x_trans_a)
y2_b = y
```

```
myknn = function(xtest, xtrain, ytrain, k) {
  n_test = nrow(xtest)
  n_pred = numeric(n_test)
  iter = seq(1, n_test)
  for(i in iter){
    xtest_i = xtest[i, ]
    distance <- rowSums(abs(sweep(xtrain, 2, xtest_i)) ^ 2)
    dist_y = cbind(distance, ytrain)
    min_dist = dist_y[order(dist_y[, 1], decreasing=FALSE),]
    n_pred[i]=sum(min_dist[1:k, 2]) / k
  }
  n_pred
}
```

```
# running knn for both the covariates
xtrain = x_matrix_a[1:400, ]
ytrain = y2_a[1:400]
xtest = x_matrix_a[401:nrow(x_matrix_a), ]
ytest = y2_a[401:length(y2_a)]

k = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400)
```

```
knn_test_pred = lapply(k, myknn, xtest = xtest, xtrain = xtrain, ytrain = ytrain)

mse_test = sapply(knn_test_pred, calc_mse, actual = ytest)

knn_train_pred = lapply(k, myknn, xtest = xtrain, xtrain = xtrain, ytrain = ytrain)

mse_train = sapply(knn_train_pred, calc_mse, actual = ytrain)
```

The optimal K for part I is 10 and the test MSE is 15.6818354.

```
xtrain = x_matrix_b[1:400, ]
ytrain = y2_b[1:400]
xtest = x_matrix_b[401:nrow(x_matrix_b), ]
ytest = y2_b[401:length(y2_b)]

k = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400)

knn_test_pred = lapply(k, myknn, xtest = xtest, xtrain = xtrain, ytrain = ytrain)

mse_test = sapply(knn_test_pred, calc_mse, actual = ytest)

knn_train_pred = lapply(k, myknn, xtest = xtrain, xtrain = xtrain, ytrain = ytrain)

mse_train = sapply(knn_train_pred, calc_mse, actual = ytrain)
```

The optimal K for part II is 2 and the test MSE is 2.7566101.

Part II outperforms part I primarily becasue of the fact that the covarites are highly correlated in with the originial three variables and hence, the mse is lower than the other part because in that case all the covariates are independent and therefore the originial x's are unable to explain the model fully.

Question 3

```
library(ElemStatLearn)
trn = as.data.frame(zip.train)
tst = as.data.frame(zip.test)

df = rbind(trn, tst)
df$V1 = as.factor(as.character(df$V1))


nfold = 10
index_fold = sample(rep(1:nfold, length.out = nrow(df)))

all_k = c(seq(from = 10, to = 100, by = 10))

accu_matrix = matrix(NA, length(all_k), nfold)

for (l in 1:nfold) {
  for (k in 1:length(all_k)) {
    knn_fit = kknn(V1 ~ ., train = df[index_fold != l, ],
                   test = df[index_fold == l, ], k = all_k[k])
    accu_matrix[k, l] = mean(knn_fit$fitted.values == df$V1[index_fold == l])
```

```r
  }
}


best_k = all_k[which.min(apply(accu_matrix, 1, mean))]


find_closest = function(xtest, xtrain, ytrain, k) {
  ytest = rep(0, nrow(xtest))
  ids = rep(0, nrow(xtrain))
    dist = rep(0, nrow(xtrain))
    for (i in 1:nrow(xtrain)) {
      ids[i] = i
      dist[i] = sum((xtest[1, 2:257] - xtrain[i, 2:257]) ^ 2)
    }
    df = data.frame("ids" = ids, "distance" = dist)
    df = df[order(df$distance), ][1:k+1, ]
    return(df)

}

temp = find_closest(df[1, ], df, df[, 1], best_k)

ids = temp$ids
```

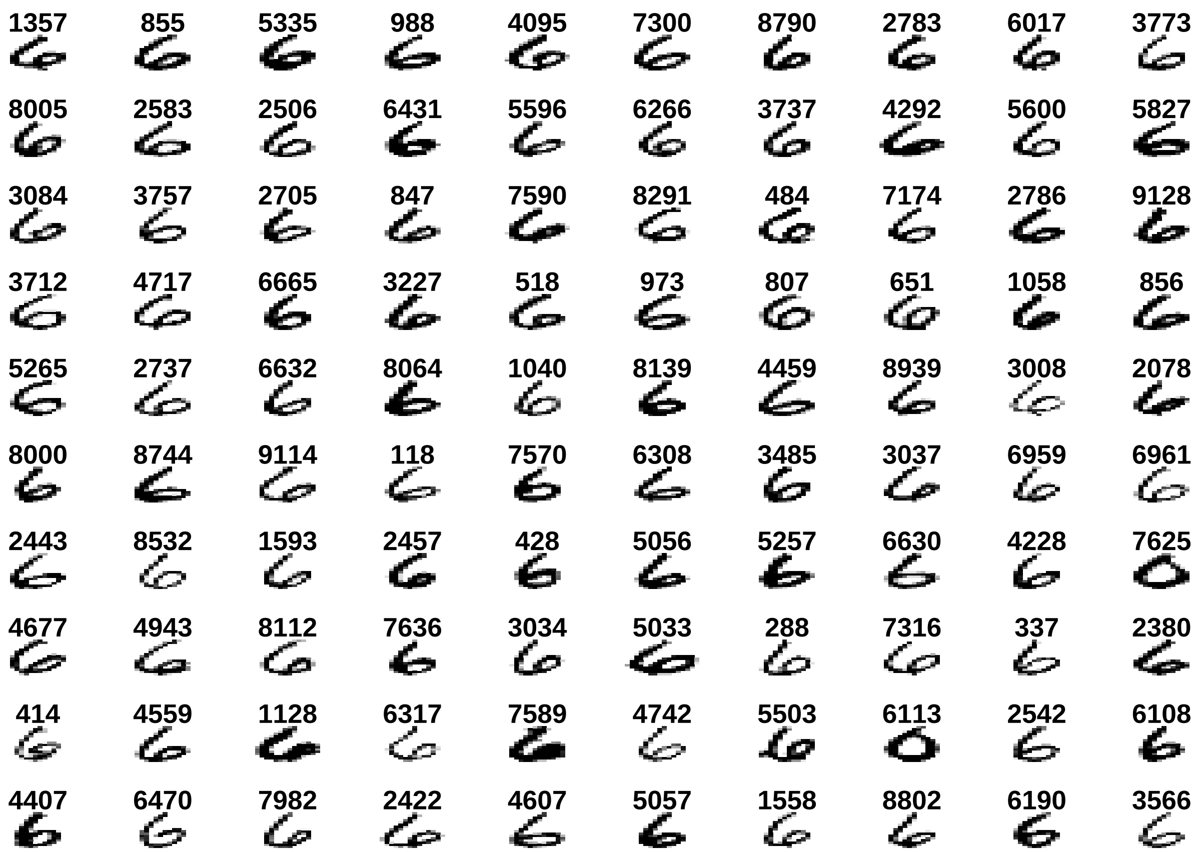
```r
par(mfrow=c(10,10), mar=c(1,1,1,1))
for(i in 1:best_k) {
  image(zip2image(rbind(zip.train, zip.test), temp$id[i]), col=gray(256:0/256), zlim=c(0,1), xlab="", yl
}
```

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

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

## [1] "digit  6  taken"

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

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

```
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  0  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  6  taken"
## [1] "digit  0  taken"
```

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



The obsevation is correctly classified using K-Nearest Neighbours based on the optimal value of k.

KNN seems to perform well on this data set as the representation of the points (images of hand written) are similar in the data set. It means 6 is very similarly written in all the images of 6 and hence the pixel values are also very similarly spread out in the images. Therefore, the distances are close to other numbers and hence KNN performes well on this data set.