Stat 154 Problem Set Two

Jinze Gu SID:24968967

February 21, 2014

Problem One

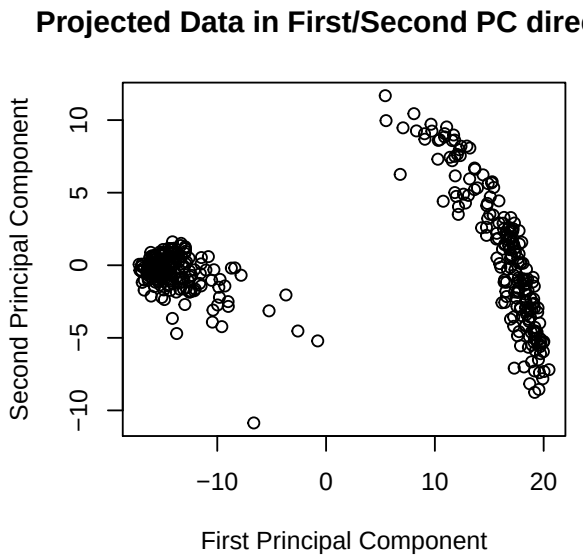
I used three types of dimension reduction method here, namely, PCA, kernel PCA and cmd scaling to reduce the dimension of data. Besides, I did different clusters after the dimension reduction. This is a classic example showing that we are able to cluster data in a better way after PCA. I suppose this is due to the fact that predictors are more correlated under the setup of problem is voting scenario.
Furthermore, we can do a loadings analysis, which shows us that there are mainly three types of voting results or there are three groups of voters in these 669 number of votings, namely, one group of people are against the proposal, another group is not against and the other group have no opinions. Although it is really naive result, I feel it is impressive to know that this is the trend from the perspective of data analysis.

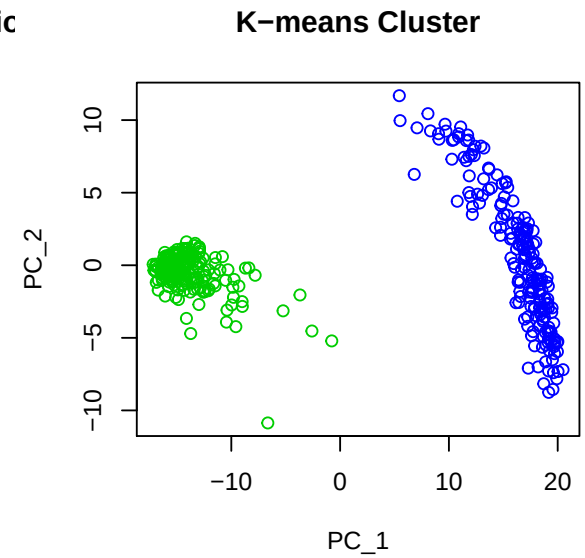
Comment:
As we can see from the model:
Firstly, as k increases, variance will explain less EPE while bias will explain more EPE for knn method. Expected EPE is also decreasing with increasing k.

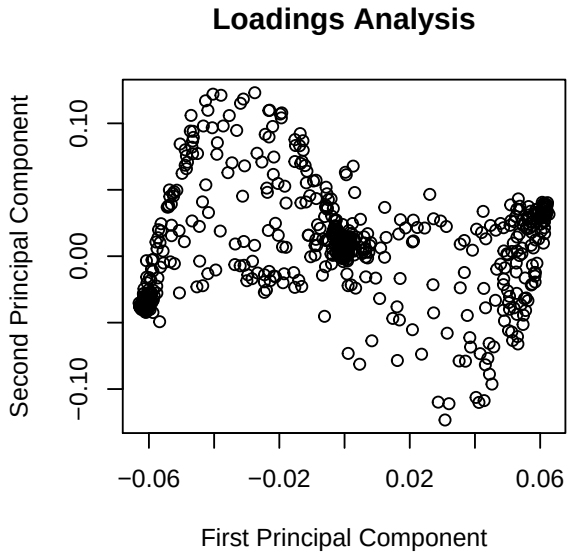
Secondly, linear fit and quadratic fit is no better than knn fit, which is what we should have noticed; because we know that the data is generated by an oscillating model and the limited points are not enough for us to draw a big picture of data.

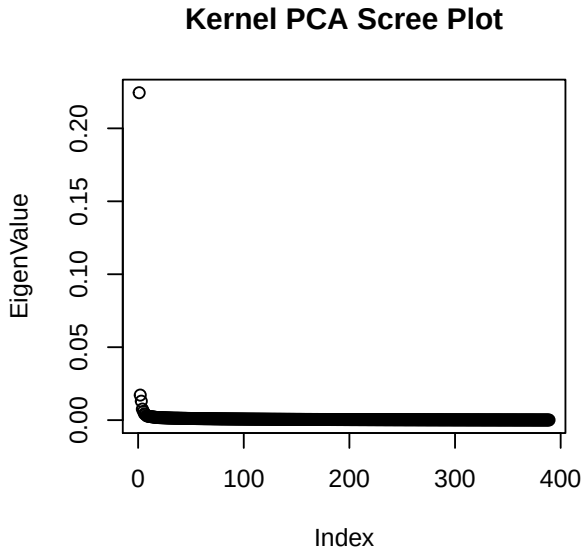
```
setwd("/Volumes/æIJĽèČ¡åĞžæšą/Stat 154/HW_2")
par(mfcol = c(2, 2))
Vote <- read.table("VotingRecord.txt")
HouseRep <- read.table("HouseMember.txt", sep = "\t")
Vote <- as.data.frame(Vote)
# PCA
pc_vote <- prcomp(Vote, rtex = TRUE)
plot(pc_vote$x[, 1], pc_vote$x[, 2], xlab = "First Principal Component", ylab = "Second Principal Compor
    main = "Projected Data in First/Second PC direction")
plot(pc_vote$rotation[, 1], pc_vote$rotation[, 2], xlab = "First Principal Component", ylab = "Second Pr
    main = "Loadings Analysis")
# Now we can cluster the data based on first and second dimension
cl_kmean <- kmeans(cbind(pc_vote$x[, 1], pc_vote$x[, 2]), centers = 2, nstart = 1)
plot(cbind(pc_vote$x[, 1], pc_vote$x[, 2]), col = (cl_kmean$cluster + 2), xlab = "PC_1", ylab = "PC_2",
    main = "K-means Cluster")
# Kernel PCA
library(kernlab)

## Warning:  package 'kernlab' was built under R version 3.0.2

rbf <- rbfdot(sigma = 0.001)
kpcma_vote <- kernelMatrix(rbf, as.matrix(Vote))
```

```
kpc_vote <- kpca(kpcma_vote)
plot(eig(kpc_vote), main = "Kernel PCA Scree Plot", ylab = "EigenValue")
```



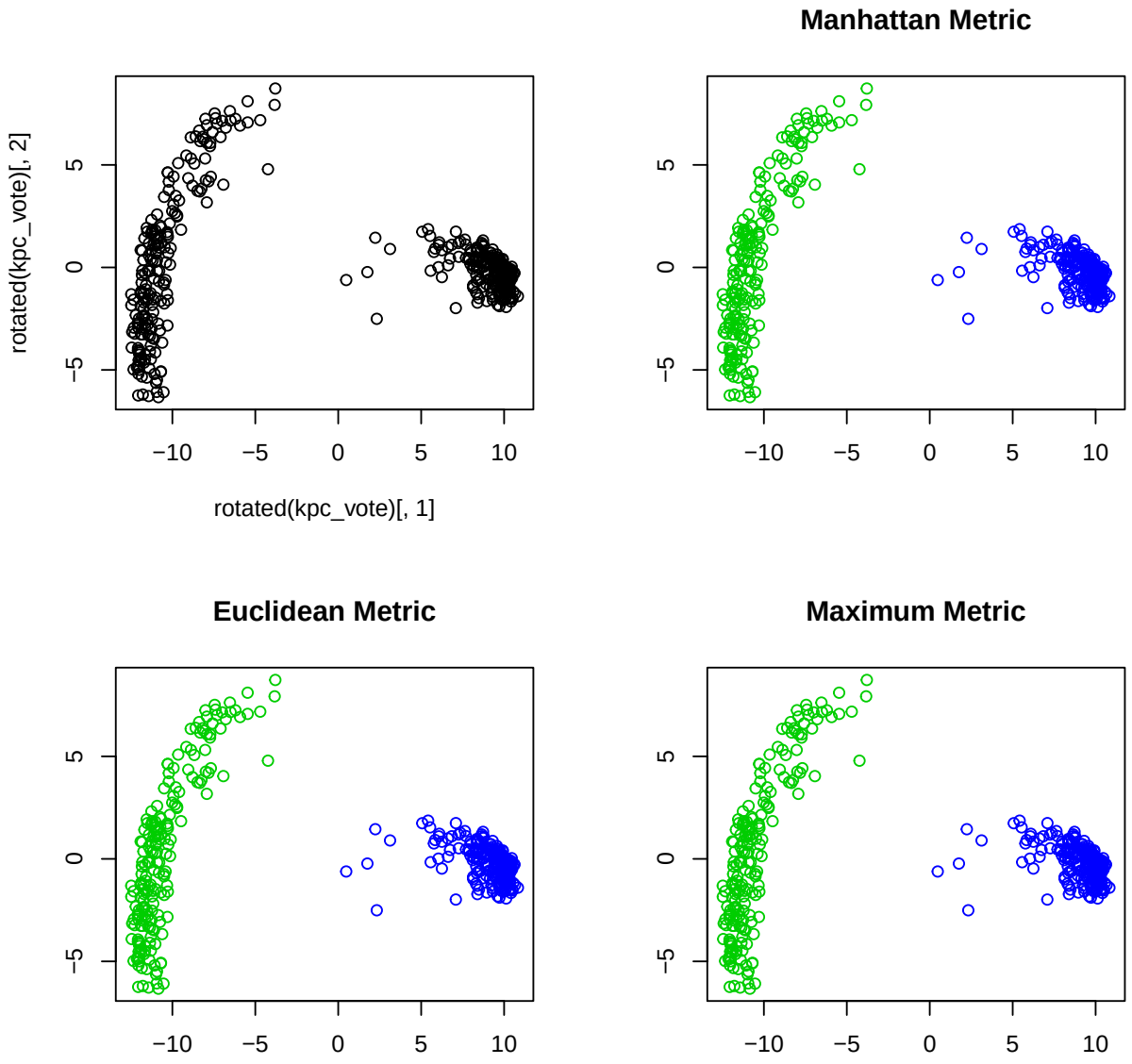






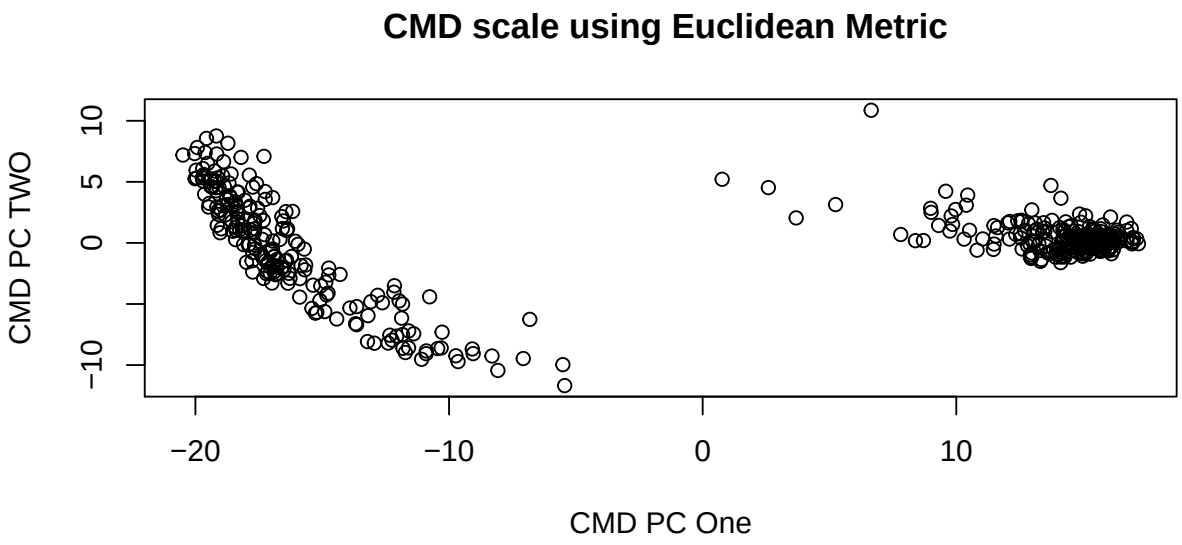

```
plot(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2])
## Now I will use K-medoids to cluster the data using Euclidean distance
library(cluster)
diss_vote_1 <- dist(cbind(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2]), method = "euclidean")
kmed_vote_1 <- pam(diss_vote_1, k = 2)
plot(cbind(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2]), col = (kmed_vote_1$clustering +
    2), xlab = "", ylab = "", main = "Euclidean Metric")
diss_vote_2 <- dist(cbind(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2]), method = "manhattan")
kmed_vote_2 <- pam(diss_vote_2, k = 2)
plot(cbind(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2]), col = (kmed_vote_2$clustering +
```

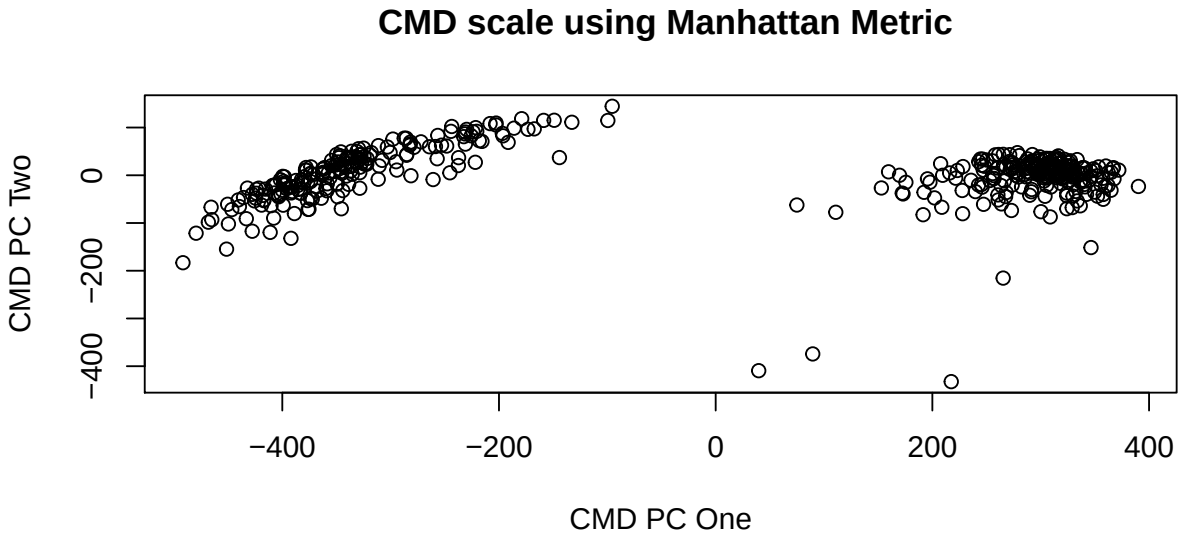
```
    2), xlab = "", ylab = "", main = "Manhattan Metric")
diss_vote_3 <- dist(cbind(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2]), method = "maximum")
kmed_vote_3 <- pam(diss_vote_3, k = 2)
plot(cbind(rotated(kpc_vote)[, 1], rotated(kpc_vote)[, 2]), col = (kmed_vote_3$clustering +
    2), xlab = "", ylab = "", main = "Maximum Metric")
```




```
## After kernel PCA, we can do a H-cluster
dis_euc <- hclust(diss_vote_1, method = "complete")
plclust(dis_euc, labels = FALSE, main = "Complete Metric H-cluster")
dis_man <- hclust(diss_vote_2, method = "single")
plclust(dis_man, labels = FALSE, , main = "Single Metric H-cluster")
dis_max <- hclust(diss_vote_3, method = "average")
plclust(dis_max, labels = FALSE, , main = "Average Metric H-cluster")
# Using CMDscale
```

```r
par(mfcol = c(2, 1))
plot(cmdscale(dist(Vote), k = 2), main = " CMD scale using Euclidean Metric", xlab = "CMD PC One",
    ylab = "CMD PC TWO")
plot(cmdscale(dist(Vote, method = "manhattan"), k = 2), main = " CMD scale using Manhattan Metric",
    xlab = "CMD PC One", ylab = "CMD PC Two")
```





Problem Two

```r
set.seed(25041)
par(mfcol = c(2, 2))
require(FNN)
```

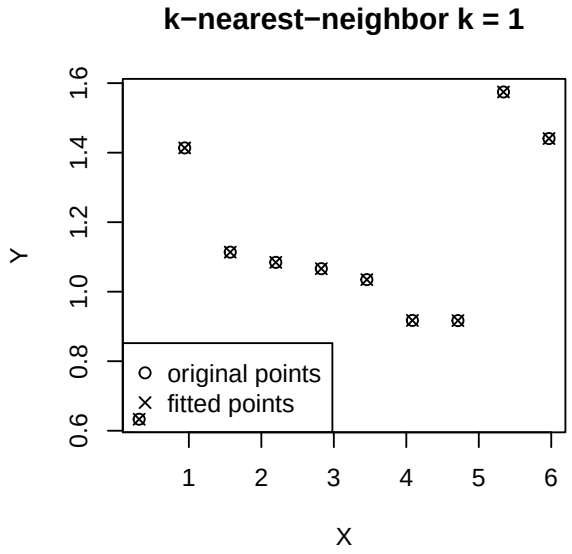
```
## Loading required package:  FNN
## Warning:  package 'FNN' was built under R version 3.0.2
```

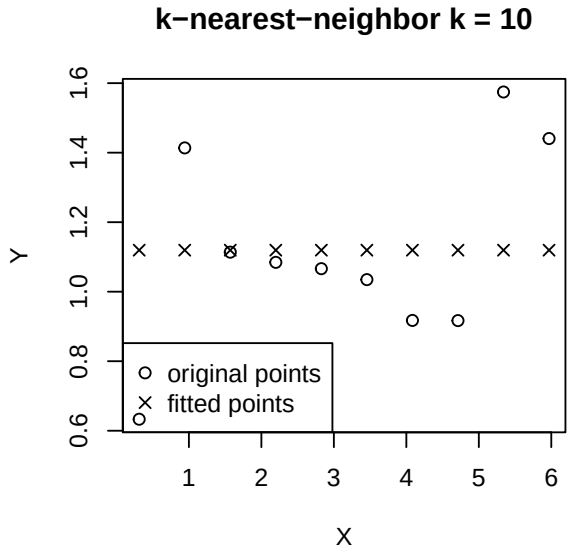
```
require(fields)
```

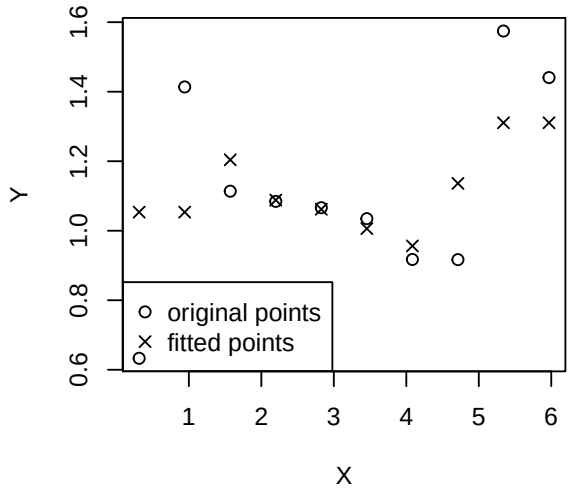
```
## Loading required package:  fields
## Loading required package:  spam
## Loading required package:  grid
## Spam version 0.40-0 (2013-09-11) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g.  'help( chol.spam)'.
##
## Attaching package:  'spam'
##
## äÿÑåĹŮåŕźèśęèćńåśŔèŤ¡äžĘfrom 'package:base':
##
##    backsolve, forwardsolve
##
## Loading required package:  maps
```

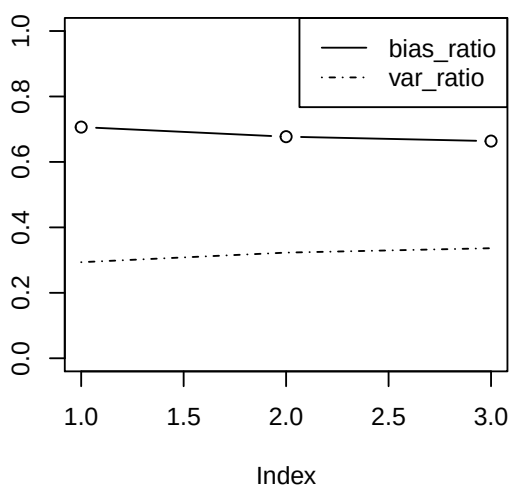
```
X <- (c(1:10) - 1/2)/10 * 2 * pi
Y <- cos(10 * X) + 2 + rnorm(10, sd = sqrt(0.1))
# 1) # This is the k-nn funtion that returns the estimated value of some point using k
# nearest neighbors with Euclidean metric
knn <- function(x, y, xseq, k) {
    if (k < length(x)) {
        dmat <- rdist(x, xseq)
        indices <- order(dmat)[2:(k + 1)]  # If you need to find less than 10 neighbors, it will not tak
        return(mean(y[indices]))
    } else {
        dmat <- rdist(x, xseq)
        indices <- order(dmat)[1:k]
        # If you need to find 10 neighbors, it will take the points itself as a neighbor
        return(mean(y[indices]))
    }
}
# Plot knn function for k = 1,3,10
knn_one <- sapply(X, knn, y = Y, xseq = X, k = 1)
plot(X, Y, main = "k-nearest-neighbor k = 1")
points(X, knn_one, pch = 4)
legend("bottomleft", c("original points", "fitted points"), pch = c(1, 4))
knn_thr <- sapply(X, knn, y = Y, xseq = X, k = 3)
plot(X, Y, main = "k-nearest-neighbor k = 3")
points(X, knn_thr, pch = 4)
legend("bottomleft", c("original points", "fitted points"), pch = c(1, 4))
knn_ten <- sapply(X, knn, y = Y, xseq = X, k = 10)
plot(X, Y, main = "k-nearest-neighbor k = 10")
points(X, knn_ten, pch = 4)
legend("bottomleft", c("original points", "fitted points"), pch = c(1, 4))
# EPE(pi) and E(EPE(X)) For each fixed X(notation is pi in problem), I decided to generate
# its response 100 times(with different errors) and calculate EPE(pi). Then I conduct the
# same procedure for 500 values randomly generated from UNIF(0, 2*pi) and calculate the
# E(EPE(X)). In order to reduce the error effect, I use the same error 500 times for
```

```r
# different fixed X.
set.seed(12345)
Eps <- matrix(rep(rnorm(100, sd = sqrt(0.1)), 500), ncol = 500)  # Firstly I generate random
X_pre <- (c(1:500) - 1/2)/500 * 2 * pi  # The 500 randomly generated number from        UNIF(0, 2*pi)
Simu_Y <- t(matrix(1, nrow = 500, ncol = 100) * (cos(10 * X_pre) + 2)) + Eps
# This is the model used to fit a prediction data(data_X) with original model constructed
# by Y and X, I use
knn_model <- function(data_X, X, Y, k) {
    fit <- sapply(data_X, knn, y = Y, xseq = X, k = k)
    EPE <- matrix(NA, nrow = length(data_X), ncol = 3)
    for (i in 1:length(data_X)) {
        EPE[i, 1] <- mean((Simu_Y[, i] - fit[i])^2)
        EPE[i, 2] <- mean((Simu_Y[, i] - mean(Simu_Y[, i]))^2)
        EPE[i, 3] <- mean((fit[i] - mean(Simu_Y[, i]))^2)
    }
    # Since X's are draw from uniform distribution, so we can estimate the Expected EPE by
    # taking the average of 500 different EPE
    MeanEPE <- mean(EPE)/(2 * pi)
    var_ratio <- mean(EPE[, 2]/EPE[, 1])
    bias_ratio <- mean(EPE[, 3]/EPE[, 1])
    return(data.frame(Mean_EPE = MeanEPE, var_ratio = var_ratio, bias_ratio = bias_ratio))
}
knn_model(X_pre, X, Y, 1)

##   Mean_EPE var_ratio bias_ratio
## 1   0.1712    0.2936     0.7064

knn_model(X_pre, X, Y, 3)

##   Mean_EPE var_ratio bias_ratio
## 1   0.1652    0.3228     0.6772

knn_model(X_pre, X, Y, 10)

##   Mean_EPE var_ratio bias_ratio
## 1   0.1635    0.3363     0.6637

plot(rbind(knn_model(X_pre, X, Y, 1)$bias_ratio, knn_model(X_pre, X, Y, 3)$bias_ratio, knn_model(X_pre,
    X, Y, 10)$bias_ratio), type = "b", ylim = c(0, 1), ylab = "", main = "Variance and Bias Ratio Behavi
lines(rbind(knn_model(X_pre, X, Y, 1)$var_ratio, knn_model(X_pre, X, Y, 3)$var_ratio, knn_model(X_pre,
    X, Y, 10)$var_ratio), lty = 4)
legend("topright", c("bias_ratio", "var_ratio"), lty = c(1, 4))
```










```
# Comment: It is obvious that as k increases, variance accounts for less and less in the
# Expected Prediction Error and bias accounts for more and more in EPE.
# Fit a constant function(The same as fitting a knn with k = 10 since we only have ten
# points in the trainning sample)
knn_model(X_pre, X, Y, 10)

##   Mean_EPE var_ratio bias_ratio
## 1   0.1635    0.3363     0.6637

# Comment: For constant fit, it is equivalent to fit a knn model with 10 points.
# Fit a linear model
fit_linear <- lm(Y ~ X)
predict_linear <- X_pre * fit_linear$coefficients[2] + fit_linear$coefficients[1]
EPE_linear <- matrix(NA, 500)
```

```r
for (i in 1:500) {
    EPE_linear[i] <- mean((predict_linear[i] - Simu_Y[, i])^2)
}
mean(EPE_linear)/(2 * pi)  # This is the estimated E(EPE(X)) under linear model
```

```
## [1] 0.2475
```

```r
Var_linear <- sum((mean(predict_linear) - predict_linear)^2)
Bias_linear <- sum((Simu_Y[1, ] - mean(predict_linear))^2)
Var_linear
```

```
## [1] 7.238
```

```r
Bias_linear
```

```
## [1] 817.8
```

```r
# Fit a quadratic function
fit_quadra <- lm(Y ~ X + I(X^2))
predict_quadra <- X_pre^2 * fit_quadra$coefficients[3] + X_pre * fit_quadra$coefficients[2] +
    fit_quadra$coefficients[1]
EPE_quadra <- matrix(NA, 500)
for (i in 1:500) {
    EPE_quadra[i] <- mean((predict_quadra[i] - Simu_Y[, i])^2)
}
mean(EPE_quadra)  # This is the estimated E(EPE(X)) under quadratic model
```

```
## [1] 1.556
```

```r
Var_quadra <- sum((mean(predict_quadra) - predict_quadra)^2)
Var_quadra
```

```
## [1] 8.798
```

```r
Bias_quadra
```

```
## Error:  æĽ¿äÿĐåĹřåŕźèšą'Bias_quadra'
```