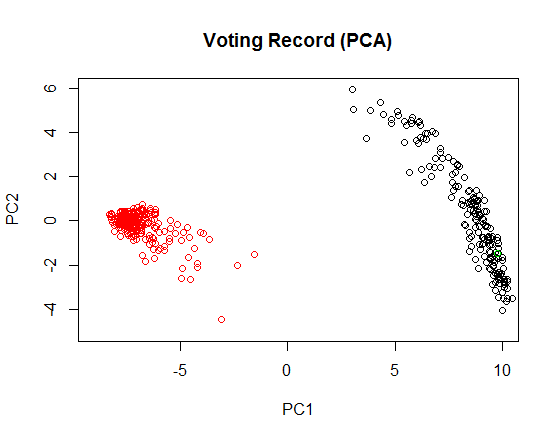
Audrey Chou

22690210

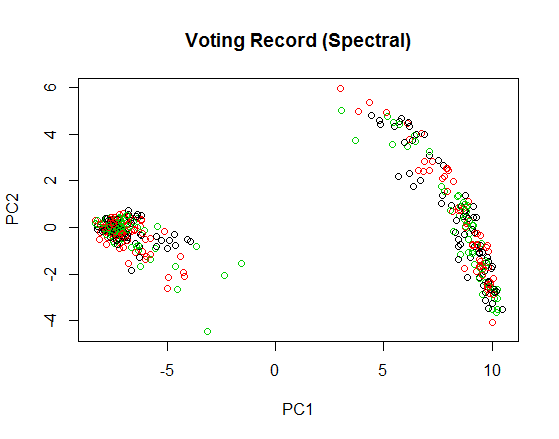
DIS 102

Stat 154 Homework 2

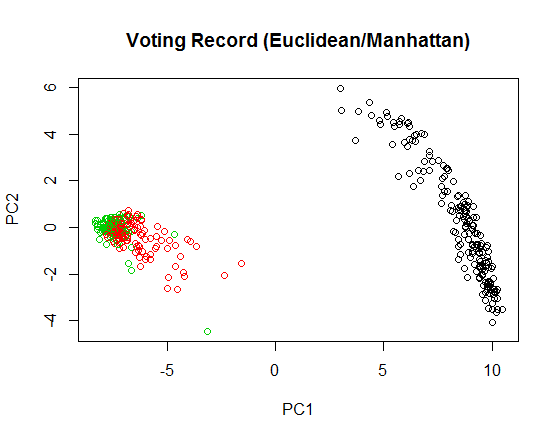
PROBLEM 1



This plot is colored by party -- Democrats were colored black, Republicans red, and third parties green. From here, it's pretty clear that Democratic and Republican voting records are radically different from each other. However, it also seems that Republicans tend to vote more similarly to each other than Democrats, as the black points arch across the plot while the red points cluster together. The one green point is nested well within the Democratic cluster, indicating that the third-party representative probably shares many beliefs with the Democrats.

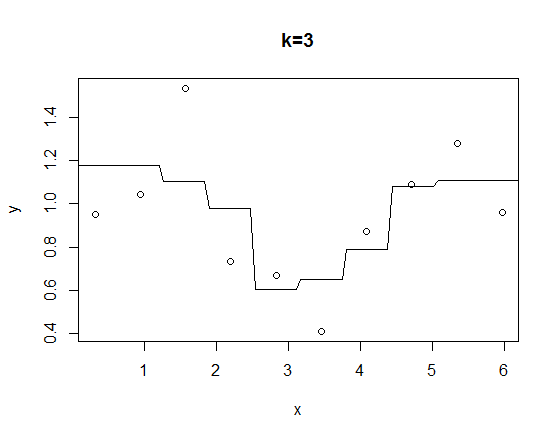
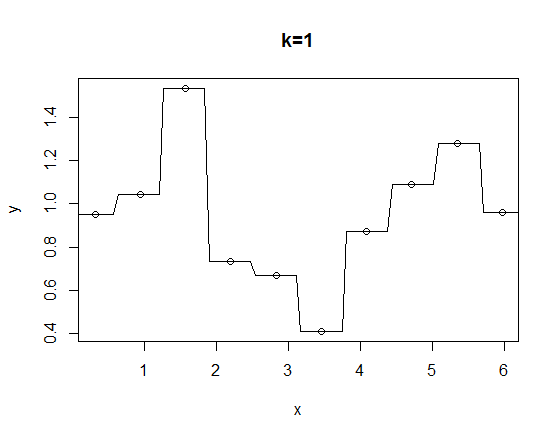


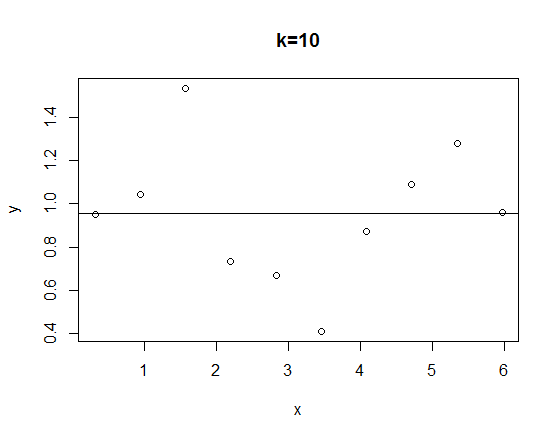
The coloration for this plot was done through spectral clustering; here it seems that the clusters are dispersed evenly throughout the points. This may imply that voting record was completely uncorrelated to the party, but we know this isn't true in real life. Therefore, this clustering is probably inaccurate for this sort of data.



The coloration for this plot was determined using k-medoids clustering. I tried both the Euclidean distance formula and the Manhattan distance formula in calculating this, but both returned the same plot. The plot implies the arc on the left is all one cluster, but the group on the left is actually two clusters. This may indicate that two parties vote very similarly to each other, while the third is radically different; incidentally, the first plot had a similar display, except the coloring was switched.

PROBLEM 2 - PART 1





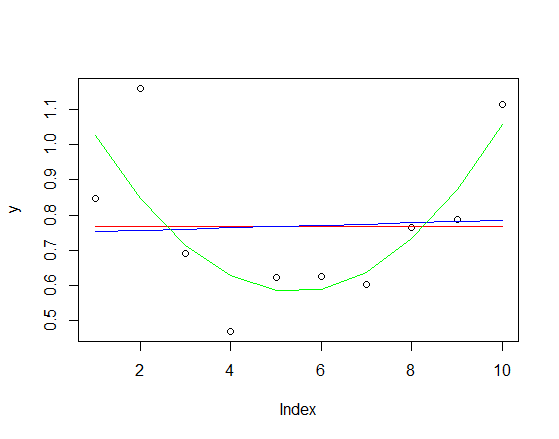
The lines on these plots indicate the predictions from k-nearest neighbors, for k = 1, 3, and 10. k=1 is obviously the most accurate of the three.

|  |  |  |
| --- | --- | --- |
| k | EPE(pi) | E(EPE(x)) |
| 1 | 3.992827 | 1.055985 |
| 3 | 4.389062 | 1.612919 |
| 10 | 5.566966 | 1.620004 |

From these values, we can surmise that the expected prediction error increases as k increases. This makes sense, since the smaller nearest-neighbors are more accurate; this could be because using less neighbors in the algorithm means the calculations are more specific and less generalized.

In addition, the variance of the nearest neighbors prediction decreases (in fact, it is actually 0 for k=10 because we are setting all the predictions to the same value), which means bias increases as k increases. So while the confidence interval of the prediction may be smaller with more neighbors, there is more bias in the data, which may confound the prediction.

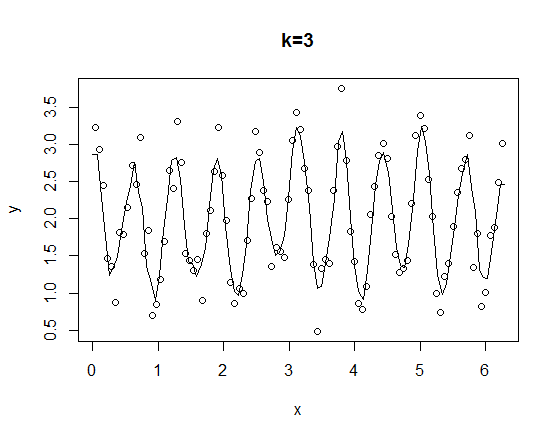
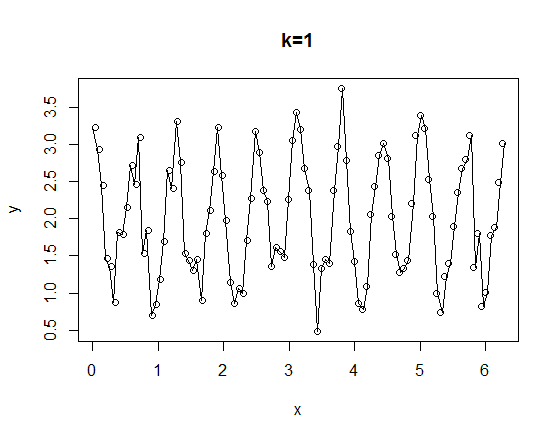
Below are models fitted to the original ten points (red for constant, blue for linear, green for quadratic).

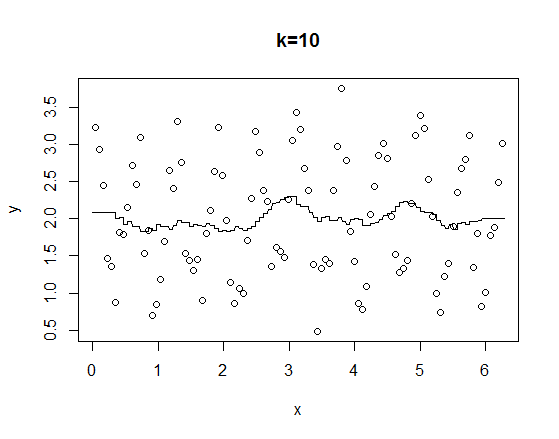


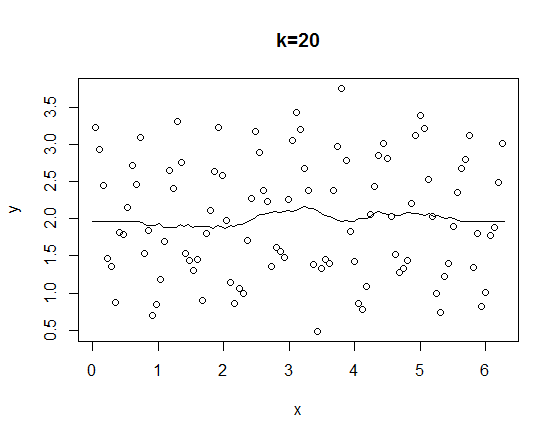
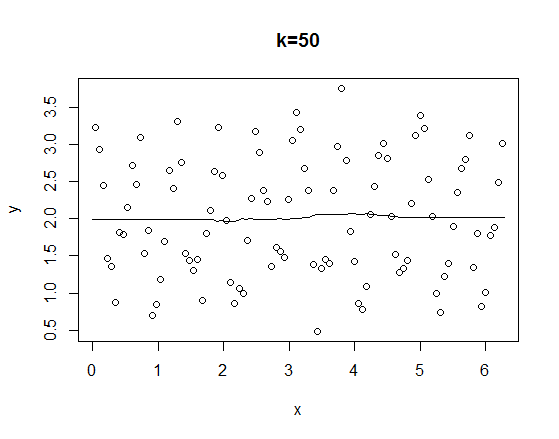
|  |  |
| --- | --- |
| Function | E(EPE(x)) |
| Constant | 0.04432481 |
| Linear | 0.04421924 |
| Quadratic | 0.01704385 |

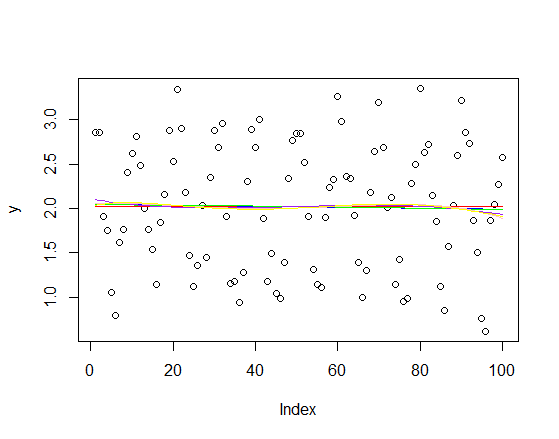
The quadratic model appears to have the smallest EPE not just among the three models, but of all 6 models. This could be attributed to the fact that the models are fitted to the actual data, while the nearest neighbors were attributed to training data only; in addition, the actual data follows a cosine trend, whose curve is mirrored by the quadratic curve.

PROBLEM 2 - PART 2



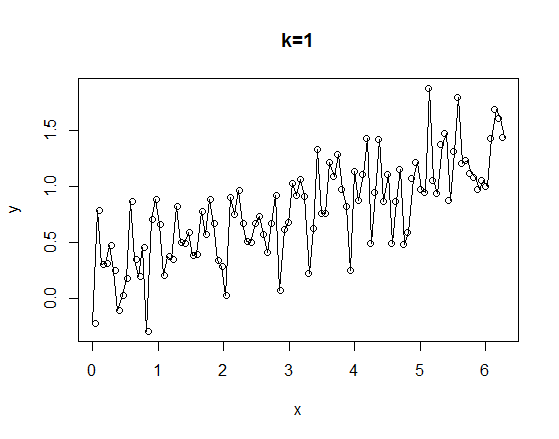
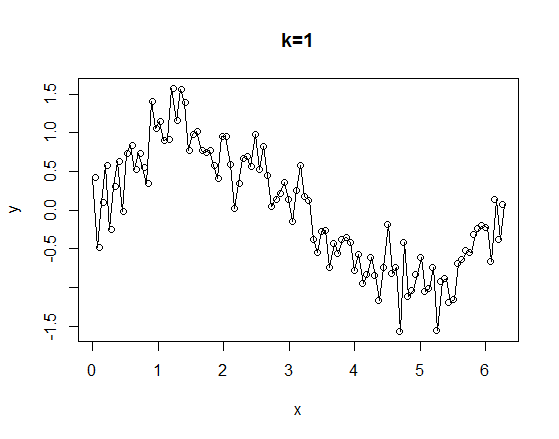




For N = 100, it appears that k = 1 is the most accurate way to estimate the data; it also has the smallest EPE of all the models. Fitting a function to the data no longer works, as there are so many points with errors that the distribution of the points seems almost random. Therefore, the quadratic function no longer has the smallest EPE.

Below are the k=1 graphs for sin(x) and .1 + .2x (I really do not want to paste several graphs and tables, a college student can only afford so much ink...). Both of these graphs also indicate that k=1 is the best fit for the data, as the predictions accurately fit the true points.



If the variance of the errors increase, the data will be less likely to clearly follow the original formula. Therefore, it will be difficult to fit a function to the data, and using the k-nearest neighbors algorithm will probably be the best option. Specifically, 1-nearest neighbors will provide the most accurate prediction of the data.

PROBLEM 3

For N=50, p=5 and f(x) = sin(x), these were the EPEs.

|  |  |  |
| --- | --- | --- |
| k/function | EPE(pi) | E(EPE(X)) |
| 1 | 12.40861 | 4.225254 |
| 10 | 19.07333 | 5.212887 |
| Constant |  | 3.112627 |

As expected, when k is larger, the error is also larger. In addition, fitting a function seems to produce a smaller EPE than using nearest-neighbors. There is not much difference in the comparison of these values when dimensionality is added, but the errors are much larger when there are more dimensions than when we were only dealing with one dimension.

PROBLEM 4

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.247 1.419 1.476 1.479 1.538 1.780

The differences appear to center around 1.4 to 1.5, which is roughly where the square root of 2 is. This could be because the values are picked from a normal distribution with SD 1, which means the average distance from the mean is 1. This implies that the points are roughly 1 apart from each other.

Appendix

PROBLEM 1

rdr2005 = reduced\_voting\_record2005[,1:669]

rhp2005 = reduced\_house\_party2005[,1:401]

rdr.PCA = prcomp(rdr2005)$x

plot(rdr.PCA[which(rhp2005==1),], col = "black", xlim = c(-9,10), ylim = c(-5, 6),

main = "Voting Record (PCA)")

points(rdr.PCA[which(rhp2005==0),], col = "red")

points(rdr.PCA[which(rhp2005==2),][1], rdr.PCA[which(rhp2005==2),][2], col = "green")

#Rs more unified than Ds

###Spectral

library(kernlab)

x = c(x1,x2,x3)

y = c(y1,y2,y3)

xy = cbind(x,y)

rdr.spec=specc(rdr2005, 3)

plot(rdr.PCA, col = rdr.spec, main = "Voting Record (Spectral)")

###K-medoids (Euclidean)

install.packages("proxy")

library(proxy)

library(cluster)

dim(rdr2005)

###[1] 401 669

rdrmed.euclid = pam(rdr2005, k = 3, metric = "euclidean")

rdrmed.manhattan = pam(rdr2005, k = 3, metric = "euclidean")

plot(rdr.PCA, col=rdrmed.euclid$clustering, main = "Voting Record (Euclidean/Manhattan)")

plot(rdr.PCA, col=rdrmed.manhattan$clustering)

PROBLEM 2-PART 2

N=10

x=numeric(N)

for (i in 1:N){

x[i] = (i-1/2)/N\*2\*pi

}

y=numeric(N)

epsilon = rnorm(10, 0, sqrt(.1))

y = cos(10\*x) + 2 + epsilon

###k=1

test = seq(0, 2\*pi, length.out=100)

test2 = numeric()

for (i in 1:100){

test2[i] = knn.reg(train=x, test=test[i], y=y, k=1)$pred

}

plot(x,y, main = "k=1")

points(test, test2, type="l")

###E(EPE(x))

mean((cos(10\*test)+2) - test2)

#[1] 1.055985

var(test2)

#[1] 0.09056013

mean((cos(10\*test)+2) - test2)-var(test2)

#[1] 0.9654247

###k=3

test = seq(0, 2\*pi, length.out=100)

test2 = numeric()

for (i in 1:100){

test2[i] = knn.reg(train=x, test=test[i], y=y, k=3)$pred

}

plot(x,y, main = "k=3")

points(test, test2, type="l")

###E(EPE(x))

mean(((cos(10\*test)+2) - test2)^2)

#[1] 1.612919

var(test2)

#[1] 0.04241028

mean((cos(10\*test)+2) - test2)-var(test2)

#[1] 0.9900557

###k=10

test = seq(0, 2\*pi, length.out=100)

test2 = numeric()

for (i in 1:100){

test2[i] = knn.reg(train=x, test=test[i], y=y, k=10)$pred

}

plot(x,y, main = "k=10")

points(test, test2, type="l")

###E(EPE(x))

mean(((cos(10\*test)+2) - test2)^2)

#[1] 1.620004

var(test2)

#[1] 0

mean((cos(10\*test)+2) - test2)-var(test2)

#[1] 1.055985

#EPE(pi)

((cos(10\*pi) + 2 + rnorm(1, 0, sqrt(.1))) - (knn.reg(train=x, test=pi, y=y, k=1)$pred))^2

#[1] 3.992827

((cos(10\*pi) + 2 + rnorm(1, 0, sqrt(.1))) - (knn.reg(train=x, test=pi, y=y, k=3)$pred))^2

#[1] 4.389062

((cos(10\*pi) + 2 + rnorm(1, 0, sqrt(.1))) - (knn.reg(train=x, test=pi, y=y, k=10)$pred))^2

#[1] 5.566966

#Fitting models

test2 = cos(10\*x) + 2 + rnorm(10, 0, sqrt(.1))

plot(test2)

points(lm(test2~1)$fitted.values, type = "l", col = "red")

points(lm(test2~x)$fitted.values, type = "l", col = "blue")

points(lm(test2~x + I(x^2))$fitted.values, type = "l", col = "green")

#EPE of models

mean(((cos(10\*x)+2+rnorm(1, 0, sqrt(0.1))) - lm(test2~1)$fitted.values)^2)

#[1] 0.004806404

mean(((cos(10\*x)+2+rnorm(1, 0, sqrt(0.1))) - lm(test2~x)$fitted.values)^2)

#[1] 0.01021356

mean(((cos(10\*x)+2+rnorm(1, 0, sqrt(0.1))) - lm(test2~x + I(x^2))$fitted.values)^2)

#[1] 0.01660585

PROBLEM 3

N=50

p=5

xi=data.frame(V1=runif(p, 0, (2\*pi)))

for (i in 2:N){

xi[,i]=runif(p, 0, (2\*pi))

}

epsilon=rnorm(N, 0, 1)

y=numeric()

y = sum(sin(xi)) + epsilon

plot(y)

###EPE(pi), k=1

1 + (sin(pi) - y[1])^2 + 1

#[1] 12.40861

###EPE(pi), k=10

1 + (sin(pi) - (1/10)\*(sum(y[1:10])))^2 + 1/10

#[1] 19.07333

###E(EPE(x)), k=1

epe = numeric(N)

for (i in 1:N){

epe[i] = mean(1 + (y - y[i])^2 + 1)

}

mean(epe)

#[1] 4.225254

###E(EPE(x)), k=10

epe = numeric(N)

for (i in 1:(N-10)){

epe[i] = mean(1 + (y - (1/10)\*(sum(y[i:10+i])))^2 + 1/10)

}

mean(epe, na.rm=TRUE)

#[1] 5.212887

###E(EPE(x))

mean(1 + (lm(y~1)$fitted.values - y)^2 + 1)

#[1] 3.112627

PROBLEM 4

df = data.frame(rnorm(100, 0, 1))

for (i in 2:1000){

df[,i]=rnorm(100, 0, 1)

}

i0 = sample(x = 1:1000, 1)

xi0 = df[,i0]

xj= df[,-c(i0)]

p=100

diff = c()

for(i in 1:999){

diff[i] = (sqrt(sum((xi0-xj[,i])^2)))\*(1/sqrt(p))

}