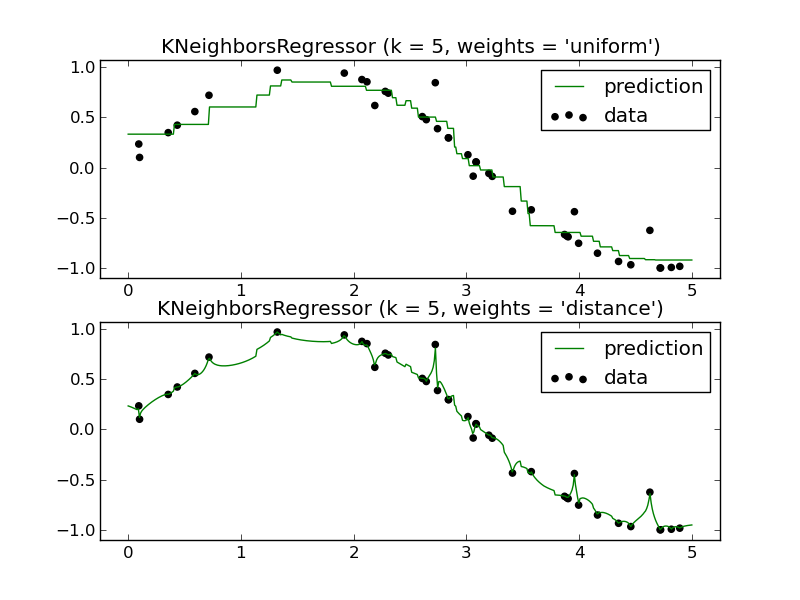
**11.0 – Nearest Neighbor Regression**

In nearest neighbor regression we estimate the fitted/predicted value at a target set of predictor values or better yet using term notation is simply the average or weighted average of the response values for the nearest neighbors to the target point within the training data. More explicity if we denote the set of indices (observation numbers) of the -nearest neighbors as then the fitted/predicted value is given by:

or if we use a weighted average then we have of observations in the neighborhood,

It should be intuitive that in the computing the weighted average we would give more weight in the calculation to points that are nearest the target point and decreasing weights as we move away from the target point, in other words the weights we could be inversely proportional to the distance between the target point and the neighboring point .

Clearly having a way to compute distances between the target point and the observations in the training set is critical to this process! In the next section we discuss measuring distances between multidimensional observations, possibly with a mixture of data types (i.e. continuous, ordinal, and nominal).



**11.1 – Measuring Distances**

There are several statistical methods that require some measure of distance between observations, nearest neighbor regression or classification being one of them. Another common method that starts with pairwise differences between all observations is ***cluster analysis***, which is thoroughly discussed in ***unsupervised learning***. (DSCI 415).

As an example consider the data on the 77 largest U.S. cities.  
  
> names(City)

[1] "pop1" "pop2" "growth" "area" "popdens" "black.pop"

[7] "hisp.pop" "AIP.pop" "pct.black" "pct.hisp" "pct.AIP" "pctenr"

[13] "pctdeg" "pctold" "income" "taxes" "welfare" "poverty"

[19] "pct1par" "unemprt" "laborchg" "femlab" "pctmanu" "ptrans"

[25] "medv" "medrent" "pctrent" "oldhous" "pctcondo" "pct1hous"

[31] "infmort" "crime" "july" "precip"

> summary(City) 🡨 results for all variables not shown

pop1 pop2 growth area

Min. : 201896 Min. : 148283 Min. :-18.60 Min. : 14.9

1st Qu.: 274162 1st Qu.: 238647 1st Qu.: -4.90 1st Qu.: 61.9

Median : 382816 Median : 357870 Median : 6.50 Median : 108.7

Mean : 623597 Mean : 581205 Mean : 13.69 Mean : 181.6

3rd Qu.: 551675 3rd Qu.: 562994 3rd Qu.: 25.60 3rd Qu.: 217.8

Max. :7311966 Max. :7071639 Max. : 94.60 Max. :1697.6

popdens black.pop hisp.pop AIP.pop

Min. : 145 Min. : 4821 Min. : 1038 Min. : 1478

1st Qu.: 2411 1st Qu.: 34301 1st Qu.: 7681 1st Qu.: 4355

Median : 3546 Median : 71064 Median : 23197 Median : 8910

Mean : 4914 Mean : 160818 Mean : 110520 Mean : 35156

3rd Qu.: 6526 3rd Qu.: 160283 3rd Qu.: 100717 3rd Qu.: 23185

Max. :23671 Max. :2102512 Max. :1783511 Max. :512719

welfare poverty pct1par unemprt

Min. : 2.70 Min. : 5.90 Min. :15.90 Min. : 2.300

1st Qu.: 6.40 1st Qu.:13.40 1st Qu.:23.80 1st Qu.: 5.400

Median : 9.30 Median :18.10 Median :28.80 Median : 6.400

Mean :10.22 Mean :17.98 Mean :30.38 Mean : 6.832

3rd Qu.:13.60 3rd Qu.:21.90 3rd Qu.:38.00 3rd Qu.: 7.700

Max. :26.10 Max. :32.40 Max. :53.90 Max. :13.100

medv medrent pctrent oldhous

Min. : 25600 Min. :308.0 Min. :37.00 Min. : 0.6

1st Qu.: 56700 1st Qu.:379.0 1st Qu.:43.70 1st Qu.: 7.3

Median : 71700 Median :422.0 Median :48.60 Median :14.5

Mean : 93779 Mean :445.8 Mean :50.15 Mean :22.1

3rd Qu.: 96600 3rd Qu.:476.0 3rd Qu.:55.40 3rd Qu.:36.9

Max. :353900 Max. :755.0 Max. :76.90 Max. :68.1

pctcondo pct1hous infmort crime

Min. : 0.700 Min. :16.60 Min. : 4.90 Min. : 5364

1st Qu.: 2.500 1st Qu.:27.40 1st Qu.: 9.90 1st Qu.: 8458

Median : 4.200 Median :30.50 Median :11.30 Median : 9894

Mean : 5.387 Mean :30.12 Mean :12.04 Mean :10259

3rd Qu.: 7.100 3rd Qu.:33.50 3rd Qu.:13.90 3rd Qu.:11337

Max. :29.700 Max. :41.50 Max. :23.20 Max. :18953

> X = City[,c(15,17,18)] 🡨 extract income, welfare, and poverty only

**Measuring Distance/Similarity Between Cities (distance between observations)**On the basis of these three measured characteristics (median income, percent of population receiving welfare, and percentage of the population below the poverty line) how can we measure how different or similar two cities are, e.g. Detroit, MI and Minneapolis, MN?

The Euclidean distance between two dimensional vectors is given by

> View(City)

> xd = X[9,] 🡨 extract data for Detroit

> xd

income welfare poverty

18742.0 26.1 32.4

> xm = X[47,] 🡨 extract data for Minneapolis  
> xm

income welfare poverty

25324.0 10.5 18.5

> t(xd-xm)%\*%(xd-xm) 🡨 Euclidean distance squared

[,1]

[1,] 43323161

> sqrt(t(xd-xm)%\*%(xd-xm)) 🡨 Euclidean distance

[,1]

[1,] 6582.033

Calculations “by hand”:

= 6582.03

Clearly the distance between Detroit and Minneapolis (and any other two cities for that matter) is dominated by the median income. Thus the discrepancies between the percent of the cities population on welfare and the percent below poverty level have little to do with the total dissimilarity between these two cities on the basis of these characteristics.

If we standardize the variables first, we put them all on the same scale.  
> sX = scale(X)

> sxd = sX[9,] 🡨 Detroit

> sxd

income welfare poverty

-1.451120 3.182602 2.410516   
> sxm = sX[47,] 🡨 Minneapolis

> sxm

income welfare poverty

-0.29123182 0.05672995 0.08767880

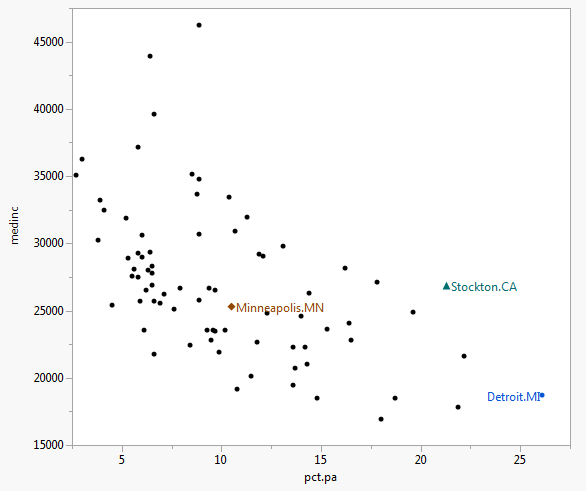
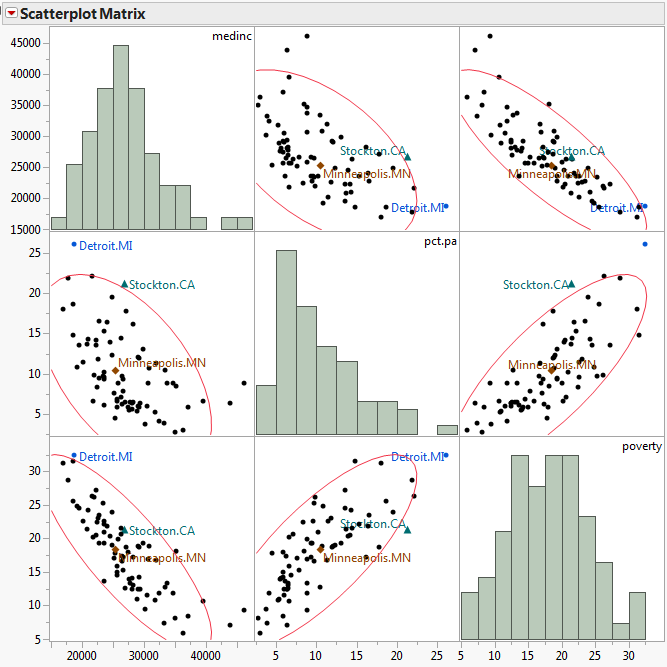
> sqrt(t(sxd-sxm)%\*%(sxd-sxm))

[1,] 4.063495

By “hand”,

= 4.0635

In the standardized scale the discrepancies between the percentages on welfare and below the poverty level are the largest contributors to the distance between Detroit and Minneapolis. Using a similar process the Euclidean distance between Detroit and Stockton, CA is 8134 and the standardized scale the distance is 2.52. Given the scatterplot below, which measure of discrepancy/distance is more appropriate? Note: These plots were created in JMP.

* *

As this example illustrates, simple Euclidean distance is not appropriate when variables involved in measure distances between observations are on different scales. Thus standardizing or scaling numeric/quantitative variables is almost done before measuring distances and will be for nearest neighbor regression.

Standardizing Variables – Two common ways to standardize a continuous variable are shown below for the variable **.**

or

Euclidean distance, on scaled or unscaled variables, is not the only way in which we can measure distance between observations based entirely upon a set of *p* numeric/quantitative variables. Below are some examples of other metrics that are used.

**Distances between observations based on numeric variables**

**Euclidean Distance**

**Minkowski Distance**

Note: *m = 2* is Euclidean distance

**Manhattan Distance** (Minkowski *m = 1* or *Taxi cab* metric)

**Chebyshev’s Distance**

**Canberra Distance**

What about other data types, e.g. ordinal or nominal? Can they be used in computing distances between observations? The answer is yes, but requires the creation of dummy variables for the levels of the ordinal/nominal predictors.

Next we will examine how to measure distance using these variables types and then how to calculate dissimilarities based on set variables with a mixture of data types.

For ordinal variables (e.g. Cancer Grade = 1,2,3,4,5) we can either treat them as numeric/continuous or we can create dummy variables as shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cancer Grade |  |  |  |  |
| 1 | 1 | 1 | 1 | 1 |
| 2 | -1 | 1 | 1 | 1 |
| 3 | -1 | -1 | 1 | 1 |
| 4 | -1 | -1 | -1 | 1 |
| 5 | -1 | -1 | -1 | -1 |

When computing difference between two observations, the number of non-zero differences corresponds to the order of the difference between them.

For nominal variables we create a separate dummy variable for each level of the variable. For example if then we create for dummy terms as shown below,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fuel Type |  |  |  |  |
| Oil | 1 | 0 | 0 | 0 |
| Gas | 0 | 1 | 0 | 0 |
| Electric | 0 | 0 | 1 | 0 |
| Other | 0 | 0 | 0 | 1 |

There is no difference between two homes say that have the same fuel type and there will be two non-zero entries in the difference between them if they do not have the same fuel type.

We then standardize all dummy variables by dividing the dummy variables by,

and

Also because ordinal/nominal variables with a large number of levels will have more potential differences we divide the difference between the standardized dummy differences by for nominal and for ordinal variables.

Once we have standardized all ordinal/nominal dummy variables as above, we treat them the same as continuous. This allows us to use metrics like Euclidean distance or the -norm to find pairwise distances between observations.

There are other ways to handle a mixture of data types in distance calculations. See the Chapter 2 – Measuring Distance Between Observations and Variables in the DSCI 415 notes.

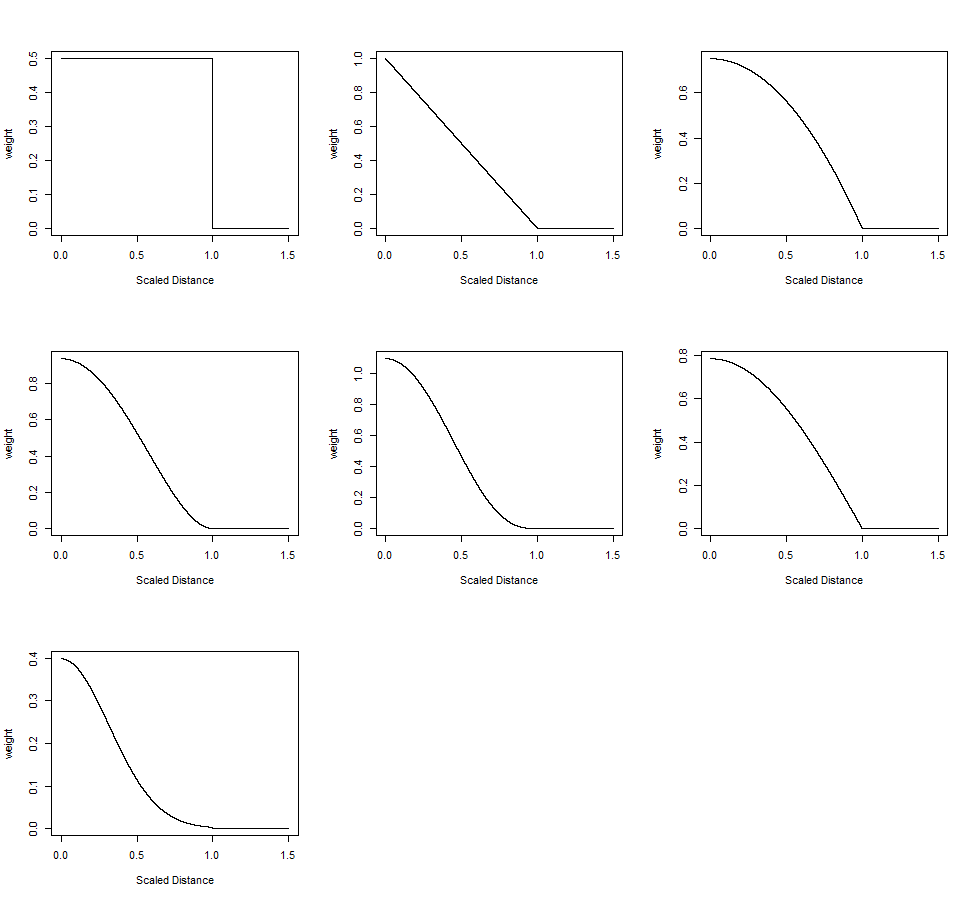
**11.2 – k-NN Regression**

Having chosen a metric to apply to the properly standardized variables we wish to use in our nearest neighbor regression we have means for fitting a unweighted *k-*nearest neighbor (*k*-NN) regression. Given a target point we compute the pairwise distances between the target point and each observation in the training data set. We can rank order these pairwise distances,

and take the average of the response values corresponding to the *k*-smallest distances: .

For a weighted *k*-NN regression estimate we need to use a weighting function that gives more weight to observations that are closer to the target point and decreasing weight as we move further away. In order to do this we divide the distances for nearest neighbors by the distance to the neighbor, . That is we compute,

and if we add to the denominator. There are various weighting functions that can be used in weighted k-NN regression and they are shown below.



Starting in the top left going from left-to-right :

rectangular  
triangular

epanechnikov  
biweight or quartic  
triweight or tricube  
cosine

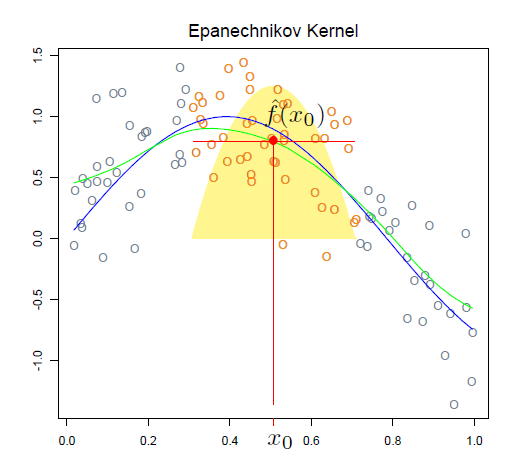
gaussian 🡨 never 0!

Finally the fitted/predicted value is given by,

The weighted k-NN regression algorithm is summarized below.

**Weighted k-NN Regression Algorithm**

1. Let be the training set of observations.
2. Find the nearest neighbors by sorting the distances **.**
3. The distance to the neighbor is used to standardize the smallest distance via the formula,
4. The weights are computed based on the and the chosen weight (kernel) function , .
5. Finally the fitted/predicted value is calculated using the formula for above.



**Example 11.1: Saratoga NY Home Prices**

The package kknn from CRAN contains functions to fit weighted k-NN regression models. We will use this method to obtain predicted home prices for a validation/test set of homes using a training set.

> summary(Saratoga)

Price Lot.Size Waterfront Age Land.Value

Min. : 5000 Min. : 0.0000 0:1713 Min. : 0.00 Min. : 200

1st Qu.:145000 1st Qu.: 0.1700 1: 15 1st Qu.: 13.00 1st Qu.: 15100

Median :189900 Median : 0.3700 Median : 19.00 Median : 25000

Mean :211967 Mean : 0.5002 Mean : 27.92 Mean : 34557

3rd Qu.:259000 3rd Qu.: 0.5400 3rd Qu.: 34.00 3rd Qu.: 40200

Max. :775000 Max. :12.2000 Max. :225.00 Max. :412600

New.Construct Central.Air Fuel.Type Heat.Type Sewer.Type Living.Area Pct.College

0:1647 0:1093 2:1197 2:1121 1: 12 Min. : 616 Min. :20.00

1: 81 1: 635 3: 315 3: 302 2: 503 1st Qu.:1300 1st Qu.:52.00

4: 216 4: 305 3:1213 Median :1634 Median :57.00

Mean :1755 Mean :55.57

3rd Qu.:2138 3rd Qu.:64.00

Max. :5228 Max. :82.00

Bedrooms Fireplaces Bathrooms Rooms

Min. :1.000 Min. :0.0000 Min. :0.0 Min. : 2.000

1st Qu.:3.000 1st Qu.:0.0000 1st Qu.:1.5 1st Qu.: 5.000

Median :3.000 Median :1.0000 Median :2.0 Median : 7.000

Mean :3.155 Mean :0.6019 Mean :1.9 Mean : 7.042

3rd Qu.:4.000 3rd Qu.:1.0000 3rd Qu.:2.5 3rd Qu.: 8.250

Max. :7.000 Max. :4.0000 Max. :4.5 Max. :12.000

Notice that Waterfront, New.Construct, Central.Air, Fuel.Type, Heat.Type, and Sewer.Type are all nominal variables, i.e. R interprets them as factors.

Form a training and validation/test set using the original data set.

> dim(Saratoga)

[1] 1728 16

> sam = sample(1:1728,floor(1728\*.6666),replace=F)

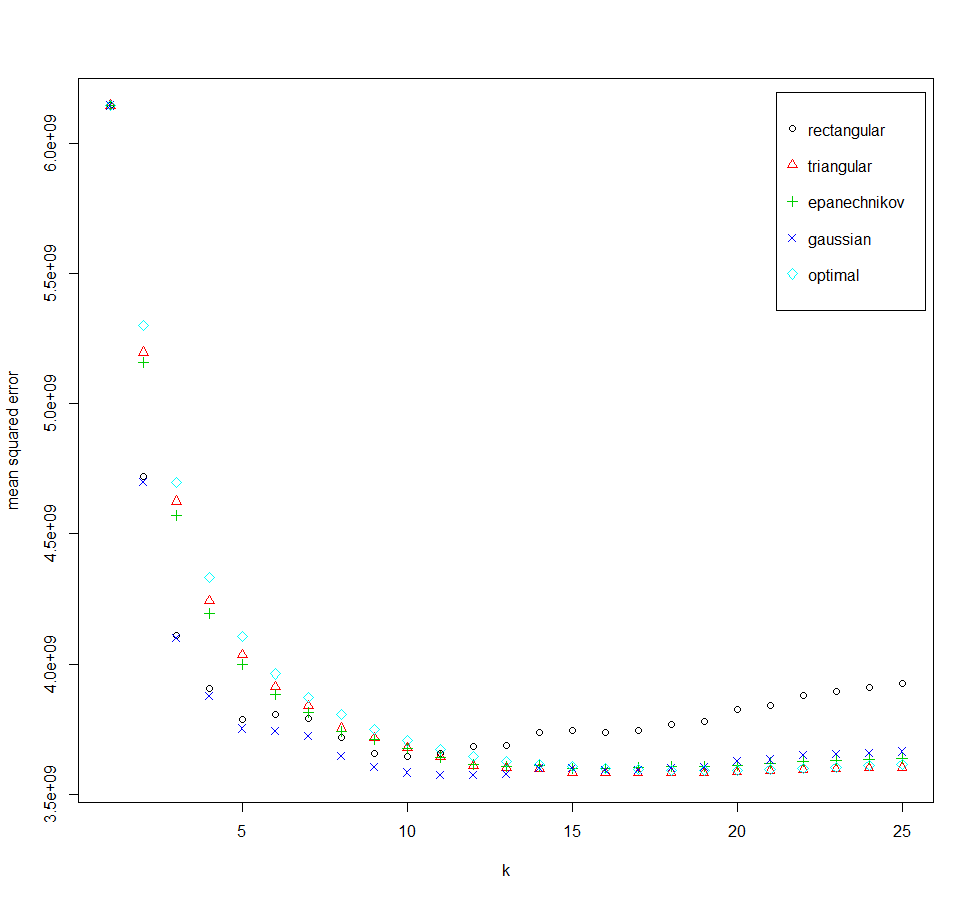
> Saratoga.train = Saratoga[sam,]

> Saratoga.test = Saratoga[-sam,]

> library(kknn)

> saratoga.train = train.kknn(Price~.,data=Saratoga.train,kmax=25,  
 kernel=c("rect","triang","epan","gauss","optimal"))

> plot(saratoga.train)



> saratoga.train

Call:

train.kknn(formula = Price ~ ., data = Saratoga.train, kmax = 25, kernel = c("rect", "triang", "epan", "gauss", "optimal"))

Type of response variable: continuous

minimal mean absolute error: 42304.92

Minimal mean squared error: 3573277478 🡨 Minimal Root Mean Squared Error = 59776.9

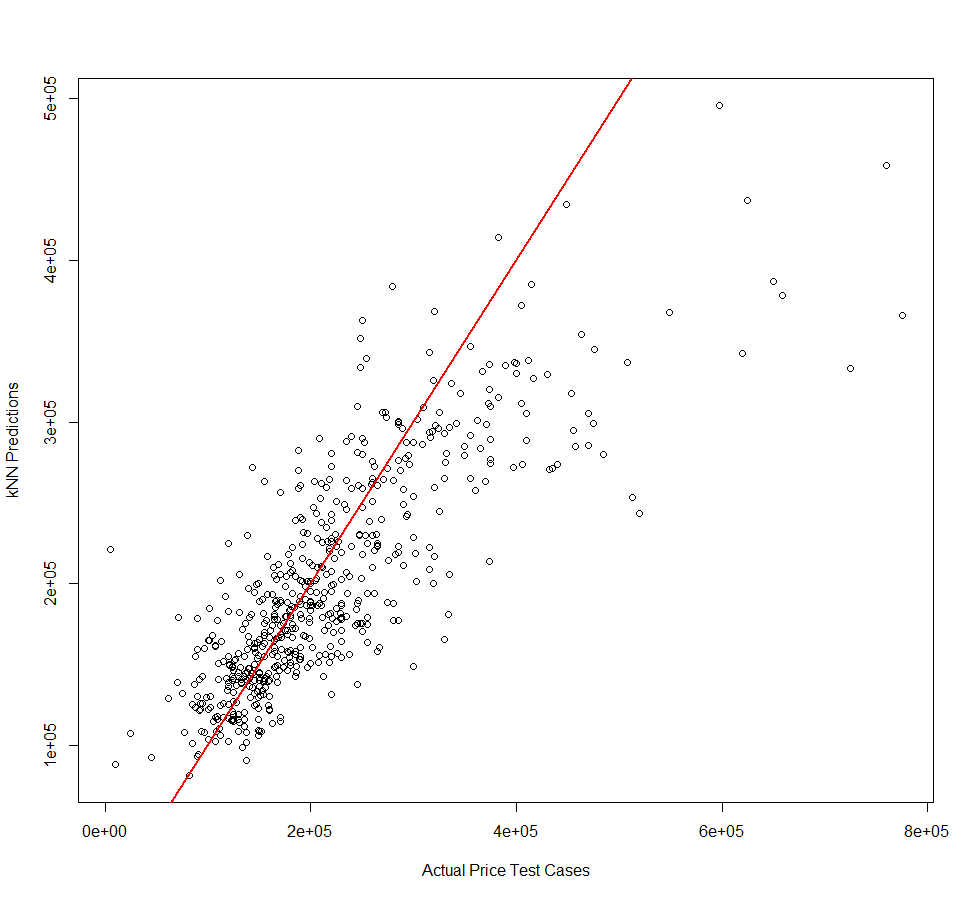
Best kernel: gaussian

Best k: 11

> ypred = predict(saratoga.train,newdata=Saratoga.test)

> ypred = exp(ypred)

> plot(Saratoga.test$Price,ypred,xlab="Actual Price Test Cases",ylab="kNN Predictions")

> abline(0,1,col="red",lwd=2)  


> cor(Saratoga.test$Price,ypred)

[1] 0.8142955

> cor(Saratoga.test$Price,ypred)^2

[1] 0.6630771

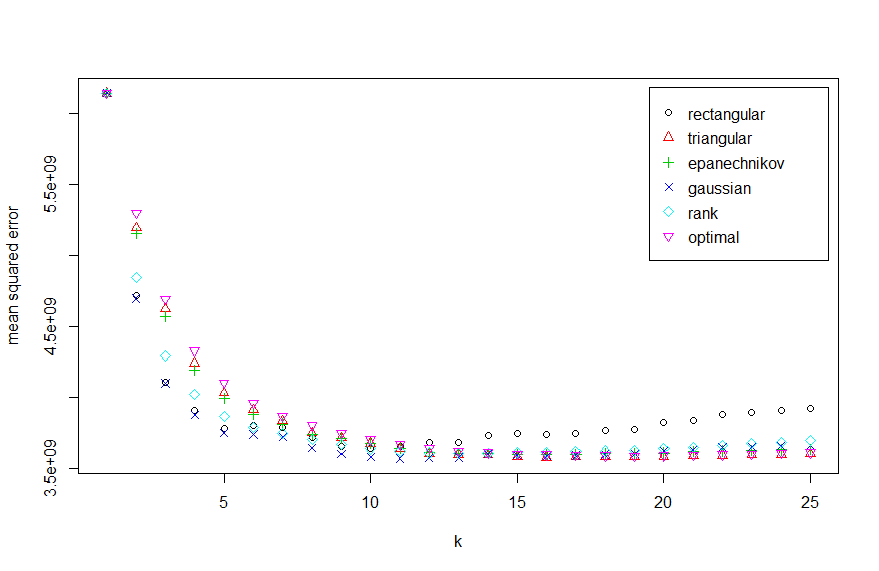
Try k-NN model with the response, home price, in the original scale.

> Saratoga.orig = train.kknn(Price~.,data=Saratoga.train,kmax=25,

+ kernel=c("rectangular", "triangular", "epanechnikov", "gaussian",

+ "rank", "optimal"))

> plot(Saratoga.orig)



> summary(Saratoga.orig)

Call:

train.kknn(formula = Price ~ ., data = Saratoga.train, kmax = 25, kernel = c("rectangular", "triangular", "epanechnikov", "gaussian", "rank", "optimal"))

Type of response variable: continuous

minimal mean absolute error: 42304.92

Minimal mean squared error: 3573277478

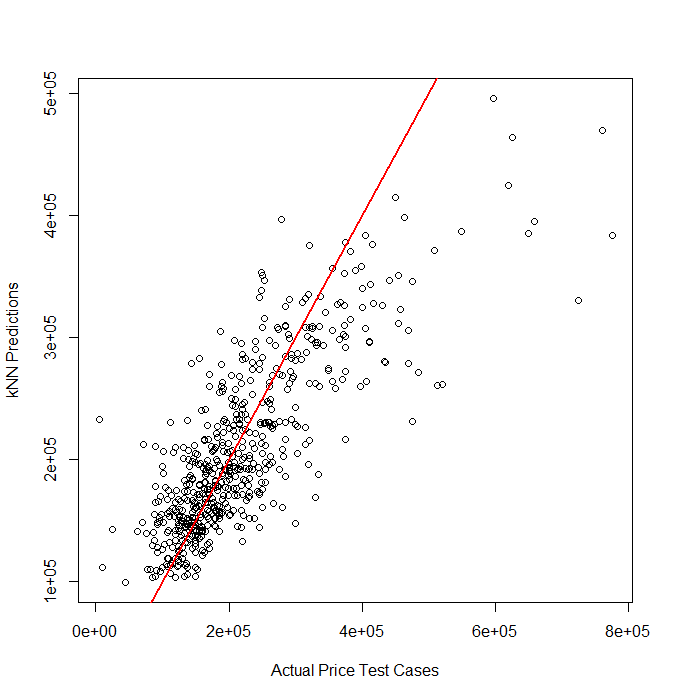
Best kernel: gaussian

Best k: 11

> ypred = predict(Saratoga.orig,newdata=Saratoga.test)

> plot(Saratoga.test$Price,ypred,xlab="Actual Price Test Cases",ylab="kNN Predictions")

> abline(0,1,col="red",lwd=2)



> cor(Saratoga.test$Price,ypred)

[1] 0.8022112

> cor(Saratoga.test$Price,ypred)^2

[1] 0.6435427

As expected (?) the log-transformed response k-NN model produces more accurate predictions as measured by the .

**Example 11.2 – Concrete Strength Experiment**Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients.

**Variable Information:**  
Given below are the variables contained the file **Concrete.csv** on course website. These data come from a collection of 17 experiments where the compressive strength (MPa) of concrete was determined under different formulations and length of curing (days). These data consist of n = 1030 observations on nine variables (8 predictors and 1 response). There are no cases with missing values!  
  
Name / Data Type / Description/Measurement Units   
(red denotes variable has zeroes)  
Cement () - continuous – kg of cement per cubic meter of concrete   
Blast Furnace Slag () - continuous – kg of slag per cubic meter of concrete   
Fly Ash () - continuous -- kg of fly ash per cubic meter of concrete   
Water () - continuous -- kg of water per cubic meter of concrete   
Superplasticizer () - continuous -- kg of superplasticizer per cubic meter of concrete   
Coarse Aggregate () - continuous -- kg of course aggregate per cubic meter of concrete   
Fine Aggregate () - continuous -- kg of fine aggregate per cubic meter of concrete   
Age - discrete – age of concrete measured in days (1-365)  
Concrete compressive strength - continuous – compressive strength in *megapascals* ()

Data source: I-Cheng Yeh, "*Modeling of strength of high performance concrete using artificial neural networks*," Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998).

> dim(Concrete)

[1] 1030 9

> sam = sample(1:1030,size=floor(1030\*.6666),replace=F)

> pairs.plus(Concrete)

> Concrete$Age = log(Concrete$Age)

> Concrete$Strength = sqrt(Concrete$Strength)

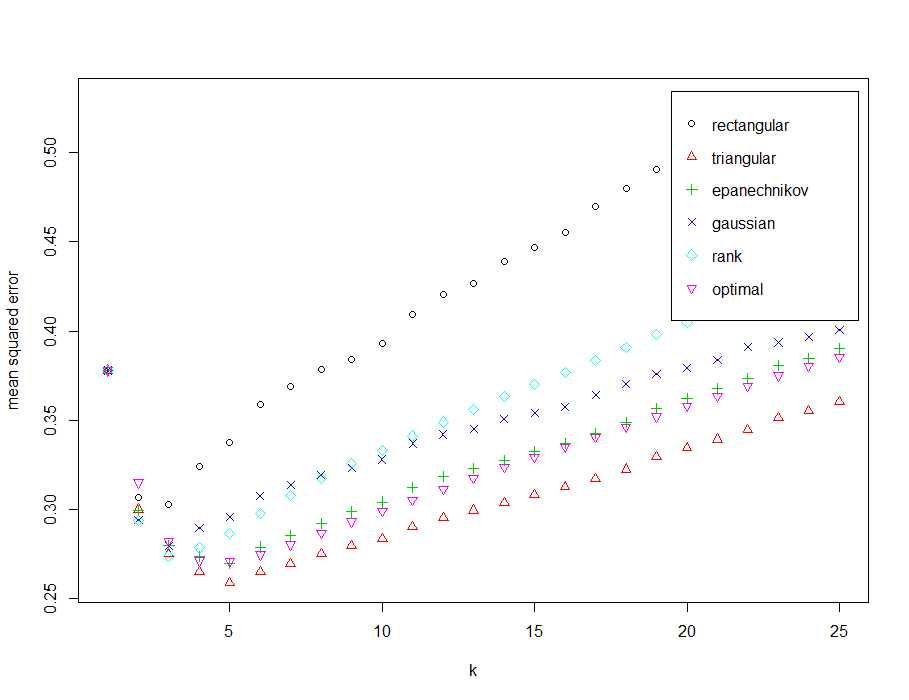
> Concrete.train = Concrete[sam,]

> Concrete.test = Concrete[-sam,]

> conc.train = train.kknn(Strength~.,data=Concrete.train,kmax=25,

+ kernel=c("rectangular", "triangular", "epanechnikov", "gaussian", "rank", "optimal"))

> plot(conc.train)



> summary(conc.train)  
Call:

train.kknn(formula = Strength ~ ., data = Concrete.train, kmax = 25, kernel = c("rectangular", "triangular", "epanechnikov", "gaussian","rank", "optimal"))

Type of response variable: continuous

minimal mean absolute error: 0.3674123

Minimal mean squared error: 0.2586658

Best kernel: triangular

Best k: 5

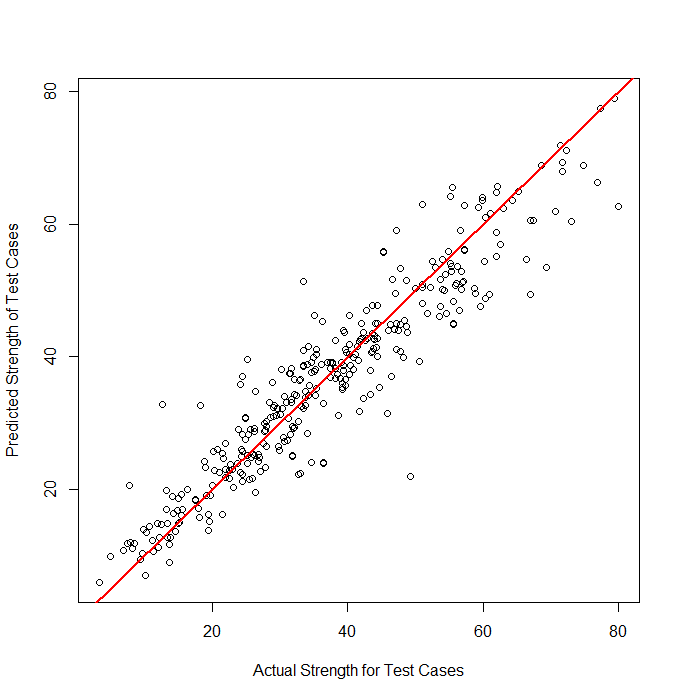
> srypred = predict(conc.train,newdata=Concrete.test)

> ypred = srypred^2

> yact = Concrete.test$Strength^2

> plot(yact,ypred,xlab="Actual Strength for Test Cases",ylab="Predicted Strength of Test Cases")

> abline(0,1,col="red",lwd=2)



> cor(yact,ypred)

[1] 0.9416205

> cor(yact,ypred)^2 🡨

[1] 0.8866492

> RMSEP = sqrt(mean((yact-ypred)^2))

> RMSEP

[1] 5.569844

> MAEP = mean(abs(yact-ypred))

> MAEP

[1] 3.992409

> MAPEP = mean(abs(yact-ypred)/yact)\*100

> MAPEP

[1] 13.38368

**Monte Carlo Split-Sample Cross-Validation Function** (knn.sscv)  
This function takes a full data set as the first argument with the first column containing the response () and the remaining columns contain the predictors. By default a .667/.333 train/test split is formed (p = .667), the number of nearest neighbors is set to   
k = 5 by default but this should be changed based upon the results from train.kknn as shown in the examples above, similarly for the kernel function which is set to “triangular” by default. Finally the Minkowski metric power, (distance = 2) by default, can be set.

kknn.sscv = function(data,p=.667,B=10,k=5,kernel="triangular",distance=2) {

n <- dim(data)[1]

y = data[,1] #Data must have response in first column!

MSE <- rep(0,B)

MAE = rep(0,B)

MAPE = rep(0,B)

for (i in 1:B) {

ss <- floor(n\*p)

sam <- sample(1:n,ss,replace=F)

fit <- kknn(formula(data),train=data[sam,],  
 test=data[-sam,],k=k,kernel=kernel,distance=distance)

yhat = fitted(fit)  
 yact = data[-sam,1]

MSE[i] = mean((yact]-yhat)^2)

MAE[i] = mean(abs(yact – yhat))

MAPE[i] = mean(abs(yact - yhat)/yact)

}

cat("RMSEP =",sqrt(mean(MSE))," MAEP=",mean(MAE)," MAPEP=",mean(MAPE))

cv = return(data.frame(RMSEP=sqrt(MSE),MAEP=MAE,MAPEP=MAPE))

}  
  
  
> results = kknn.sscv(Concrete,k=6,kernel="optimal")

RMSEP = 0.5137433 MAEP= 0.3738794 MAPEP= 0.07168817

> results = kknn.sscv(Concrete,k=10,kernel="optimal")

RMSEP = 0.5411321 MAEP= 0.4034663 MAPEP= 0.07745662

> results = kknn.sscv(Concrete,k=10,kernel="gauss")

RMSEP = 0.5436912 MAEP= 0.4130066 MAPEP= 0.08122225

> results = kknn.sscv(Concrete,k=10,kernel="rect")

RMSEP = 0.6062504 MAEP= 0.4663587 MAPEP= 0.0901011

> results = kknn.sscv(Concrete,k=6,kernel="optimal",distance=2,B=100)

RMSEP = 0.5176283 MAEP= 0.3792127 MAPEP= 0.0725839

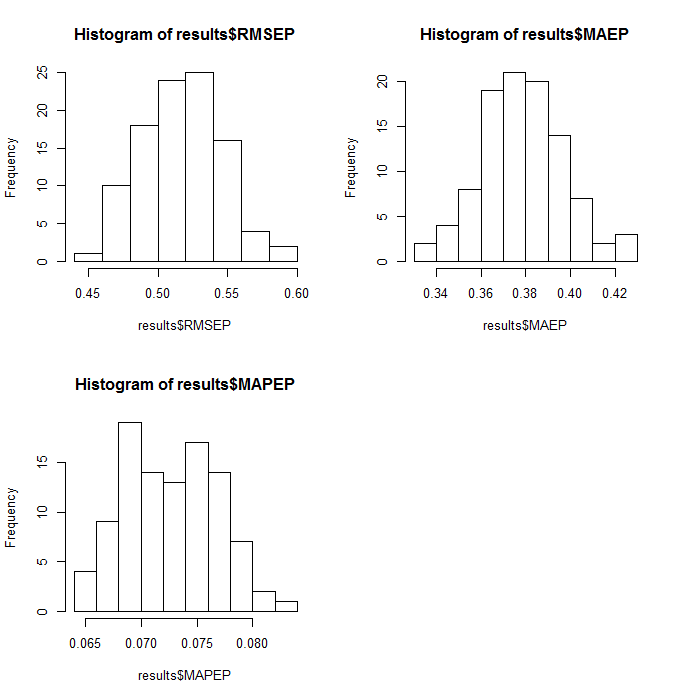
> results = kknn.sscv(Concrete,k=6,kernel="optimal",distance=2,B=100)

RMSEP = 0.5176283 MAEP= 0.3792127 MAPEP= 0.0725839

> par(mfrow=c(2,2))

> hist(results$RMSEP)

> hist(results$MAEP)

> hist(results$MAPEP)  


> par(mfrow=c(1,1))

To measure the prediction accuracy in the original scale we need square both the response and the predictions as the response was the simulations above.

To do this replace the lines highlighted above by:

yhat = yhat^2

yact = data[-sam,1]^2

> results = kknn.sqcv(Concrete,k=6,kernel="optimal",distance=2,B=200)

RMSEP = 6.083248 MAEP= 4.362191 MAPEP= 0.1529428

> results = kknn.sqcv(Concrete,k=5,kernel="optimal",distance=2,B=200)

RMSEP = 6.000394 MAEP= 4.268868 MAPEP= 0.1496128

> results = kknn.sqcv(Concrete,k=4,kernel="optimal",distance=2,B=200)

RMSEP = 5.991023 MAEP= 4.243102 MAPEP= 0.1487972

> results = kknn.sqcv(Concrete,k=3,kernel="optimal",distance=2,B=200)

RMSEP = 6.069018 MAEP= 4.259349 MAPEP= 0.1507846

Trying Different Kernels  
> results = kknn.sqcv(Concrete,k=3,kernel="trian",distance=2,B=200)

RMSEP = 5.957123 MAEP= 4.079824 MAPEP= 0.1423356 🡨 Better than “optimal” kernel?

> results = kknn.sqcv(Concrete,k=4,kernel="trian",distance=2,B=200)

RMSEP = 5.937648 MAEP= 4.117216 MAPEP= 0.1434213

> results = kknn.sqcv(Concrete,k=5,kernel="trian",distance=2,B=200)

RMSEP = 5.937845 MAEP= 4.160549 MAPEP= 0.1447503

**Problem with Nearest Neighbor Methods (Curse of Dimensionality)**

One of problems with nearest neighbor regression is when the number of predictors (dimensions) gets large the nearest neighbors may not be all that close to the target point.

To illustrate this consider a situation where all of the predictors are uniformly distributed between . Suppose that we want to base our weighted mean by growing a hypercube to contain a specified fraction of neighbors around a target point inside the hypercube

As the volume of the entire predictor space is 1, we need the volume of this hypercube to be . Therefore the expected edge length of such a hypercube is as the volume of such a cube is .

Now suppose and we want to base our nearest neighbor estimate on 1% (i.e. ) of the training data. For example if , then which is certainly reasonable choice for the number of neighbors to use in a -NN regression.

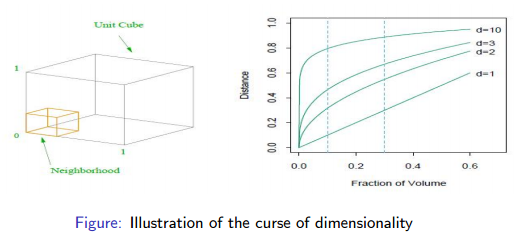
The size the necessary hypercube is . Thus we would need to extend the cube 63% of way along each dimension around the target point . Since the entire range is only 1 for each dimension, this hardly represents a local neighborhood around the target point.

Figure taken from *The Elements of Statistical Learning* by Hastie, Tibshirani, & Friedmanp.23

**11.3 - Summary of Nearest Neighbor Regression (k-NN Regression)**

-NN regression is intuitively appealing and can certainly produce reasonable predictions in some situations. However, it is important to be wary of situations where the predictors are of mixed data types, the number of predictors () is large (i.e. curse of dimensionality), and also when the number of observations () is large. That being said, many Kaggle.com winners incorporate some version of nearest neighbor methodology in their predictions, though it is definitely the case that -NN methods are the sole source of their predictions. For example, in fitting treed regression models (Cubist) we saw that a variation of the nearest neighbor concept can be used to improve predictive performance.