**Nearest neighbor classification (NNC)**

Unlike DA, NNC does not rely on any distributional assumptions! Simply, NNC works by looking at the K observations closest to an observation of interest. This observation of interest is classified as coming from the population in which the majority of the K observations belong to.

Example: Three populations

Suppose we would like to classify the RED DOT observation in the plot below as coming from population 1, 2, or 3 based on x1 and x2.



We can use Euclidean distance to measure which observations are the nearest to the red dot. Because the nearest observation corresponds to population #1, the observation is classified into population #1.

Alternatively, we could look at K = 3 nearest neighbors. In this case, the red dot would be classified into population #2.

What if K = 2?

There is not a set way used by statistical software packages to handle this situation. R breaks the tie at random; i.e., the red dot is assigned to population 1 with probability of 0.5 (the population #2 with probability 0.5). Other software packages will go to the NEXT nearest neighbor to break the tie.

Comments:

* Because no distributional assumptions are made for the data, NNC is a nonparametric procedure.
* Similar to what we saw with cluster analysis, the data are usually standardized before using NNC.
* The main NNC functions in R are knn() and knn.cv(), and these are available from the class package. The knn() function can be used in a similar manner as resubstitution was used to measure classification accuracy. Simply, the observation being classified is part of the K nearest neighbors. The “cv” of knn.cv() stands for cross-validation. Thus, the observation being classified is not part of the K nearest neighbors.

Example: 4 observations (4obs.R)

The purpose of this example is to provide a very simple situation where we can see how the NNC functions work. Below are the data:

> set1 <- data.frame(x1 = c(0, 0, 1, 1), x2 = c(0, 1, 0,

 1), pop = c(1, 1, 2, 2))

> set1

 x1 x2 pop

1 0 0 1

2 0 1 1

3 1 0 2

4 1 1 2

> plot(x = set1$x1, y = set1$x2, xlab = expression(x[1]),

 ylab = expression(x[2]), panel.first = grid(), xlim =

 c(-0.5, 1.5), ylim = c(-0.5, 1.5), pch = set1$pop,

 col = set1$pop) #Colors 1 and 2 are black and red,

 respectively

> legend(x = 0, y = 1.5, legend = c("Pop. #1", "Pop. #2"),

 col = c(1, 2), pch = c(1, 2), bty = "n", ncol = 2)

> text(x = set1$x1, y = set1$x2 + 0.1, labels =

 1:nrow(set1))

> dist.mat <- dist(x = set1[,-3], method = "euclidean")

> dist.mat

 1 2 3

2 1.000

3 1.000 1.414

4 1.414 1.000 1.000



By design, I have purposely put multiple observations equal distance from each other.

Below is how I used use knn() with K = 1. Note that the cl argument corresponds to the population of the observations given in the train argument.

> library(class)

> NNC1 <- knn(train = set1[,-3], test = set1[,-3], cl =

 set1[,3], k = 1, prob = TRUE)

> NNC1

[1] 1 1 2 2

attr(,"prob")

[1] 1 1 1 1

Levels: 1 2

> names(NNC1) #Does not work

NULL

> class(NNC1)

[1] "factor"

> attributes(NNC1)

$levels

[1] "1" "2"

$class

[1] "factor"

$prob

[1] 1 1 1 1

> attributes(NNC1)$prob

[1] 1 1 1 1

> data.frame(set1, classification = NNC1)

 x1 x2 pop classification

1 0 0 1 1

2 0 1 1 1

3 1 0 2 2

4 1 1 2 2

The knn() function does not return a list unlike functions like princomp(), factanal(), or lda(). Rather, an object of a factor class type is returned which contains the classification of an observation. For example, observation #1 is classified into population 1 with a probability of 1. Also, observation #4 is classified into population 2 with a probability of 1. The attributes() function can be used to pull the probabilities out of the factor.

Attributes were used for R much more in the past. The knn() function originated many years ago so this is why the resulting object from the function is set-up in its format.

The default for the prob argument is FALSE. I used TRUE here so that we can see the probabilities of classification for demonstration purposes. Not that these probabilities for the group which the observation was classified into. For example. The probability for observation #1 in population #1 is 1. The probability for observation #4 in population #2 is 1.

Questions:

* Why are all of the probabilities of classification equal to 1? K = 1
* Why will K = 1 always result in perfect accuracy with knn()? Resubstitution is used

Below is the code and output for K = 2, 3, and 4 with knn():

> NNC2 <- knn(train = set1[,-3], test = set1[,-3], cl =

 set1[,3], k = 2, prob = TRUE)

> NNC2

[1] 1 1 2 2

attr(,"prob")

[1] 0.6667 0.6667 0.6667 0.6667

Levels: 1 2

> NNC3 <- knn(train = set1[,-3], test = set1[,-3], cl =

 set1[,3], k = 3, prob = TRUE)

> NNC3

[1] 1 1 2 2

attr(,"prob")

[1] 0.6667 0.6667 0.6667 0.6667

Levels: 1 2

> NNC4 <- knn(train = set1[,-3], test = set1[,-3], cl =

 set1[,3], k = 4, prob = TRUE)

> NNC4

[1] 1 1 1 2

attr(,"prob")

[1] 0.5 0.5 0.5 0.5

Levels: 1 2

K = 2: The 2nd nearest observation (1st is the observation itself) results in a distance tie (there are always two observations with a distance of 1). The knn() function then includes all ties which results in the same outcome as with K = 3 here.

K = 3: The observation of interest and the two observations with a distance of 1 from it are being used in the calculation. There are always 2 of the 3 observations from the observation of interest’s population.

K = 4: The probabilities are always 2/4 because the whole data set is being used (there are 2 observations from each population). R breaks the tie at random! Thus, you should set a seed number in practice before running knn() so that you can reproduce the same result each time.

The knn() function should not generally be used with test data frame being the same as the train data frame. I used the same data frame here for demonstration purposes. If one wants to estimate the accuracy using the training data set, then knn.cv() should be used:

> NNC1.cv <- knn.cv(train = set1[,-3], cl = set1[,3], k = 1, prob = TRUE)

> NNC1.cv

[1] 1 1 2 1

attr(,"prob")

[1] 0.5 0.5 0.5 0.5

Levels: 1 2

> NNC2.cv <- knn.cv(train = set1[,-3], cl = set1[,3], k = 2,

 prob = TRUE)

> NNC2.cv

[1] 1 1 2 1

attr(,"prob")

[1] 0.5 0.5 0.5 0.5

Levels: 1 2

> NNC3.cv <- knn.cv(train = set1[,-3], cl = set1[,3], k = 3,

 prob = TRUE)

> NNC3.cv

[1] 2 2 1 1

attr(,"prob")

[1] 0.6667 0.6667 0.6667 0.6667

Levels: 1 2

K = 1: The 1st nearest observation results in a distance tie (there are always two observations with a distance of 1). The knn.cv() function then includes all ties which results in the same outcome as with K = 2. Because the two observations are from different populations, R breaks the tie at random.

K = 2: The two observations that are nearest neighbors are at a distance of 1 from the observation of interest. Because the two observations are from different populations, R breaks the tie at random.

K = 3: The probabilities are always 2/3 because the whole data set is being used, excluding the observation of interest. Notice that this always results in every observation being misclassified!

**Deciding on the value of K**

The value of K chosen obviously can affect the classification accuracy. Scatter plots of the data may help you see its structure (how observations from different populations are aligned) which may lead you to choose a particular value of K. Often, the value of K is just chosen by trying a few different values and seeing which produces the most accurate classifications.

Example: Placekicking data (PlacekickNNC.R, Placekick.csv, valid.csv)

When working with the validation data, I standardized its values using the sample means and standard deviations from the model building data. Why?

> placekick <- read.csv(file = "Placekick.csv")

> valid <- read.csv(file = "valid.csv")

> Z <- scale(placekick[,-9])

> #head(Z)

> colMeans(placekick[,-9])

 week distance change elap30 PAT

 9.36000000 27.54526316 0.25192982 12.29181165 0.53543860

 type field wind

 0.72842105 0.47859649 0.07578947

> colMeans(valid[,-9])

 week distance change elap30 PAT

 9.64539007 28.08865248 0.25531915 12.33084823 0.51063830

 type field wind

 0.24468085 0.51063830 0.08865248

> apply(X = placekick[,-9], MARGIN = 2, FUN = sd)

 week distance change elap30 PAT

 4.9247369 10.5520157 0.4342736 8.7451998 0.4989176

 type field wind

 0.4449300 0.4997170 0.2647539

> apply(X = valid[,-9], MARGIN = 2, FUN = sd)

 week distance change elap30 PAT

 5.1159297 10.8455320 0.4368156 8.6536126 0.5007755

 type field wind

 0.4306621 0.5007755 0.2847468

> Z.v <- scale(valid[,-9], center = colMeans(placekick[,-9]), scale = apply(X = placekick[,-9], MARGIN = 2, FUN = sd) )

I am unsure why the type variable is so different for the two data sets!

I apply NNC with K = 1 and use my summarize.class() function from the DA sub-section of the course.

> set.seed(7812)

> NNC1 <- knn(train = Z, test = Z.v, cl = placekick[,9], k = 1, prob = TRUE)

> head(NNC1)

[1] 1 1 1 1 1 1

Levels: 0 1

> set.seed(6615)

> NNC1.cv <- knn.cv(train = Z, cl = placekick[,9], k = 1,

 prob = TRUE)

> head(NNC1.cv)

[1] 1 1 1 1 1 1

Levels: 0 1

> summarize.class <- function(original, classify) {

 class.table <- table(original, classify)

 numb <- rowSums(class.table)

 prop <- round(class.table/numb,4)

 overall <- round(sum(diag(class.table)) /

 sum(class.table), 4)

 list(class.table = class.table, prop = prop,

 overall.correct = overall)

 }

> summarize.class(original = valid[,9], classify = NNC1)

$class.table

 classify

original 0 1

 0 17 21

 1 15 229

$prop

 classify

original 0 1

 0 0.4474 0.5526

 1 0.0615 0.9385

$overall.correct

[1] 0.8723

> summarize.class(original = placekick[,9], classify =

 NNC1.cv)

$class.table

 classify

original 0 1

 0 42 121

 1 125 1137

$prop

 classify

original 0 1

 0 0.2577 0.7423

 1 0.0990 0.9010

$overall.correct

[1] 0.8274

I am a little surprised to see the amount of difference between the validation and cross-validation overall correct percentages. Both data sets used are large. However, there are only 38 failures in the validation data set so this could lead to some variability in our results.

To find the best value of K, I used a for loop to examine K = 1, …, 10.

> set.seed(4511)

> save.results <- matrix(data = NA, nrow = 10, ncol = 4)

> save.results.cv <- matrix(data = NA, nrow = 10, ncol = 4)

> for (K in 1:10) {

 NNC <- knn(train = Z, test = Z.v, cl = placekick[,9], k = K)

 NNC.cv <- knn.cv(train = Z, cl = placekick[,9], k = K)

 NNC.accuracy <- summarize.class(original = valid[,9],

 classify = NNC)

 NNC.cv.accuracy <- summarize.class(original =

 placekick[,9], classify = NNC.cv)

 #Column #1 = K, Column #2 = (0,0), Column #3 = (1,1),

 Column #4 = overall

 save.results[K,] <- c(K, NNC.accuracy$prop[1,1],

 NNC.accuracy$prop[2,2], NNC.accuracy$overall.correct)

 save.results.cv[K,] <- c(K, NNC.cv.accuracy$prop[1,1],

 NNC.cv.accuracy$prop[2,2],

 NNC.cv.accuracy$overall.correct)

 }

> save.results

 [1,] 1 0.4474 0.9385 0.8723

 [2,] 2 0.3158 0.9180 0.8369

 [3,] 3 0.3421 0.9713 0.8865

 [4,] 4 0.3947 0.9467 0.8723

 [5,] 5 0.2895 0.9713 0.8794

 [6,] 6 0.2368 0.9754 0.8759

 [7,] 7 0.1842 0.9754 0.8688

 [8,] 8 0.1579 0.9713 0.8617

 [9,] 9 0.1842 0.9795 0.8723

[10,] 10 0.1579 0.9713 0.8617

> save.results.cv

 [,1] [,2] [,3] [,4]

 [1,] 1 0.2577 0.9010 0.8274

 [2,] 2 0.2515 0.8970 0.8232

 [3,] 3 0.1902 0.9429 0.8568

 [4,] 4 0.2209 0.9366 0.8547

 [5,] 5 0.1902 0.9635 0.8751

 [6,] 6 0.1595 0.9628 0.8709

 [7,] 7 0.1411 0.9651 0.8709

 [8,] 8 0.1411 0.9635 0.8695

 [9,] 9 0.0798 0.9754 0.8730

[10,] 10 0.0675 0.9794 0.8751

> dev.new(width = 10)

> par(mfrow = c(1,2))

> plot(x = save.results[,1], y = save.results[,2], ylim =

 c(0, 1), main = "Validation data", panel.first =

 grid(), type = "o", col = "red", xlab = "K", ylab =

 "Accuracy")

> points(x = save.results[,1], y = save.results[,3], ylim =

 c(0, 1), type = "o", col = "blue")

> points(x = save.results[,1], y = save.results[,4], ylim =

 c(0, 1), type = "o", col = "black")

> abline(h = 0.8546, lty = "dashed", col = "black") #Best

 overall from LDA and QDA, lty = "dashed" is line type

 #2

> legend(x = 5, y = 0.8, legend = c("Correct failures",

 "Correct successes", "Correct overall", "Correct

 overall LDA"), col = c("red", "blue", "black",

 "black"), bty = "n", cex = 0.75, lty = c(1,1,1,2), pch

 = c(1,1,1,NA))

> plot(x = save.results.cv[,1], y = save.results.cv[,2],

 ylim = c(0, 1), main = "Cross-validation", panel.first

 = grid(), type = "o", col = "red", xlab = "K", ylab =

 "Accuracy")

> points(x = save.results.cv[,1], y = save.results.cv[,3],

 ylim = c(0, 1), type = "o", col = "blue")

> points(x = save.results.cv[,1], y = save.results.cv[,4],

 ylim = c(0, 1), type = "o", col = "black")

> abline(h = 0.8737, lty = "dashed", col = "darkgreen")

 #Best overall from LDA and QDA, lty = "dashed" is line

 type #2

> legend(x = 5, y = 0.8, legend = c("Correct failures",

 "Correct successes", "Correct overall", "Correct

 overall LDA"), col = c("red", "blue", "black",

 "black"), bty = "n", cex = 0.75, lty = c(1,1,1,2), pch

 = c(1,1,1,NA))



The dashed line on the plot indicates the best accuracy found from using DA in the last sub-section. We see that the overall correct percentage is similar for cross-validation when comparing NNC (K ≥ 4) to DA. NNC tends to be a little better than DA with the validation data.

I am still a little hesitant to say NNC is truly better because

1. There are only 38 failures in the validation data and
2. The cross-validation results are so close.

It may be interesting to look at the accuracy for LDA and NNC for the failures and successes too.

As for choosing K, there is not a clear-cut best value. It looks like some values of K between 3 and 7 give good results for both the validation data and by using cross-validation.

Below are PC score plots similar to those obtained in the DA sub-section. I used K = 4.

> set.seed(1211)

> NNC.cv <- knn.cv(train = Z, cl = placekick[,9], k = 4)

> summarize.class(original = placekick[,9], classify =

 NNC.cv)

$class.table

 classify

original 0 1

 0 35 128

 1 68 1194

$prop

 classify

original 0 1

 0 0.2147 0.7853

 1 0.0539 0.9461

$overall.correct

[1] 0.8625

<PCA and plotting code excluded>





> set.seed(8187)

> NNC <- knn(train = Z, test = Z.v, cl = placekick[,9], k =

 4)

> summarize.class(original = valid[,9], classify = NNC)

$class.table

 classify

original 0 1

 0 15 23

 1 10 234

$prop

 classify

original 0 1

 0 0.3947 0.6053

 1 0.0410 0.9590

$overall.correct

[1] 0.883

> # Need to apply scoring from original PCA

> score.cor.v <- predict(object = pca.cor, newdata = valid)

> head(score.cor.v)

 Comp.1 Comp.2 Comp.3 Comp.4

[1,] 0.09245704 0.3888439 -1.568663 -0.391825344

[2,] 2.64186454 -0.3783378 -1.118069 -0.001312755

[3,] 2.62688967 -0.1034796 -1.735363 -1.573657836

[4,] -1.12563035 2.0254916 -1.181561 -1.176564141

[5,] -0.83923331 1.9359812 -1.198912 0.534908697

[6,] -1.39460536 0.6695345 -1.525803 -1.126650914

 Comp.5 Comp.6 Comp.7 Comp.8

[1,] -0.6992308 -0.7420951 -1.8426711 1.24985957

[2,] -1.6419234 1.0217201 1.1072842 -0.63930930

[3,] -0.2332031 0.1124153 -1.8111713 -0.88299562

[4,] -0.8928756 -0.5137668 -0.5648615 -0.12833148

[5,] -1.6135864 0.3554374 -0.5081935 -0.08548003

[6,] -0.4184636 -0.6170969 -1.8948667 -0.13086857

<Plotting code excluded>





Comments:

* Accuracy values are different here than those obtained in the for loop due to the different seed numbers.
* Compare the placekick NNC cross-validation PC plots to those in the DA notes. The NNC results allow for the classified failures to be less “clumped together” than with DA. Remember that DA essentially draws straight lines in p-dimensional space to divide how classifications are made. NNC does not have this same restriction.
* Given the accuracy of DA was similar to that of NNC, we can see the robustness to non-normality that often occurs with DA.
* How could we do variable selection here? I have not seen this done with NNC.

**Additional considerations**

* When there are more than two populations, the knn() and knn.cv() functions still only return one probability for each observation. This probability corresponds to the population that the observation is classified into. One cannot obtain probabilities for the other populations.
* Once estimates of the mean vectors and the covariance matrices are obtained when performing a DA, each observation in the training data set is no longer needed. The same is not true for NNC. You need to keep each observation in the training data set to determine the nearest neighbors.
* For very large data sets, knn() and knn.cv() could take a long time to find all of the nearest neighbors. The reason is because distances are found between all pairs of observations. Specific algorithms have been developed to avoid this type of process to find the nearest neighbors. The FNN package has its own knn() function to implement these algorithms.
* Observations within the K nearest neighbors to an observation of interest can be weighted when finding the probabilities of belonging to a particular population. For example, it may be of interest to assign larger weights to those observations closer to the observation of interest. This weighting is often done by using some type of named probability distribution. The CoreModel() function of the CORElearn package implements this approach.
* Distance measures other than Euclidean distance can be used when determining the nearest neighbors. For example, the kknn() function of the kknn package uses the Minkowski distance measure.