**13 – Nearest Neighbor Classification**

R packages: kknn, class, klaR

**13.1 – Nearest Neighbor Classification (vs. Regression)**

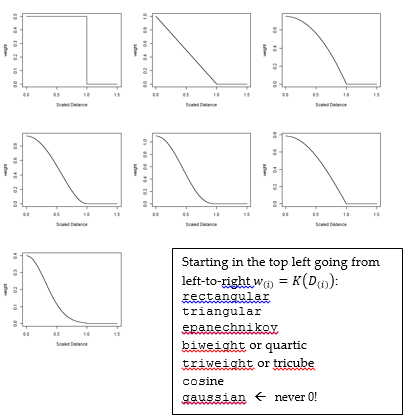
Recall, in nearest neighbor regression we estimate the fitted/predicted value at a target set of predictor values or better yet using term notation is simply the average or weighted average of the response values for the nearest neighbors to the target point within the training data. More explicity if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

This works fine when the response is numeric, but what if the response is nominal or ordinal? Then we need to consider the class or level of the nearest neighbors and classify a future observation into the most common class or level amongst the k-nearest neighbors. If then we classify a new observation to the class or level of its nearest neighbor. If then there is a possibility of a tie. For example in binary classification problem ( and then there is always the possibility that the number of neighbors where and where are equal. For this reason we typically use a weighted nearest neighbor classifier where the classes of the points nearest the target point are more important than the classes of points that are further away.

For unweighted nearest neighbor classification, suppose the response has levels and let denote the number of observations amongst the nearest neighbors that belong to class , i.e.

Then a new observation is predicted into class with,

To incorporate weights into this process we use the algorithm below which uses the same notation and kernel/weight functions as in the weighted nearest neighbor regression discussion from Section 11.



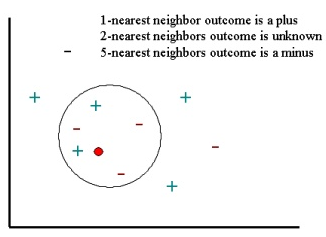
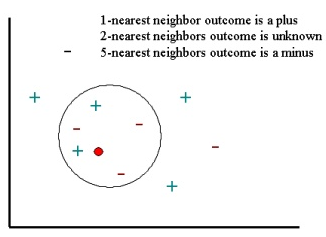
Kernel weight functions – see Section 11 of the notes.

**Weighted k-NN Classification 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 class is calculated using the formula below,

where,

Thus the classification is based upon the weighted vote of being from each class.

**Unweighted kNN Classification Weighted kNN Classification**

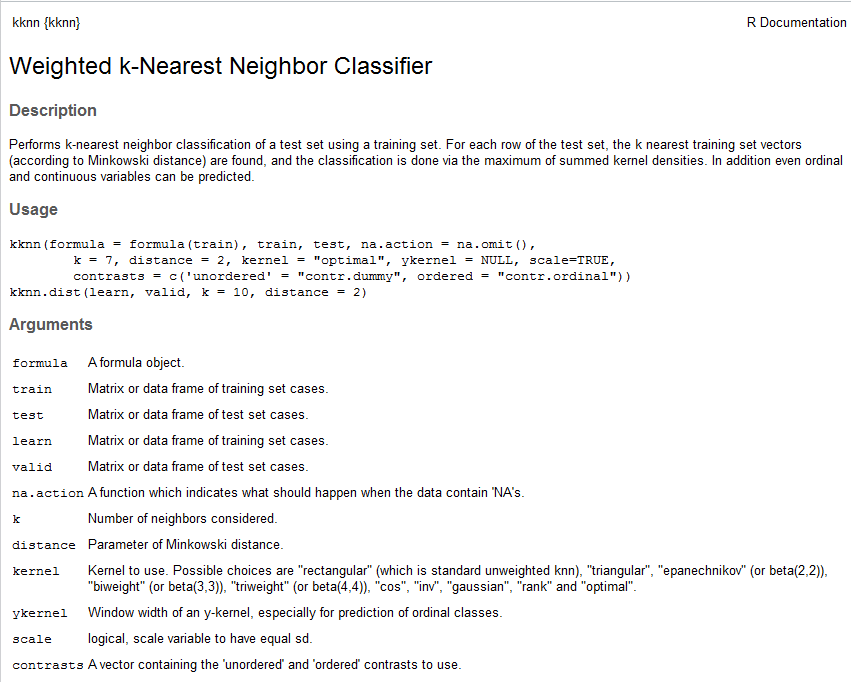
**1-nearest neighbor outcome is a plus  
2-nearest neighbors outcome is a plus  
4-nearest? 5-nearest? minus**

We can also obtain estimates of the conditional probabilities of new observation being each of the levels as,

Thus, we could also simply classify a future observation as being from the class with the largest estimated conditional probability.

**13.2 - Nearest Neighbor Classification in R**

The function kknn in the kknn library performs weighted nearest neighbor classification for test cases based on a training dataset. A portion of the help file for the function is shown below.



You may also need a function to determine the percent of observations misclassified. Here fit is the predicted class from the model and y is actual response categories.

> misclass = function(fit,y) {

temp <- table(fit,y)

cat("Table of Misclassification\n")

cat("(row = predicted, col = actual)\n")

print(temp)

cat("\n\n")

numcor <- sum(diag(temp))

numinc <- length(y) - numcor

mcr <- numinc/length(y)

cat(paste("Misclassification Rate = ",format(mcr,digits=3)))

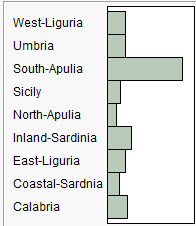
cat("\n")

}

**Example 13.1 – Predicting Growing Area for Italian Olive Oils**

The goal is classify an olive oil as being from one of nine individual growing areas in Italy (Area Name) – East Liguria, West Liguria, Umbria, North-Apulia, South-Apulia, Sicily, Coastal Sardinia, Inland-Sardinia, and Calabria. The map below should help in your understanding of where these areas are located in Italy.





Puglia = Apulia

Sardegna = Sardinia

Sicilia = Sicily

The bar graph above shows the number of olive oils in these data from each area.

***The fatty acids measured (i.e. the predictors) are as follows:***

Palmitic

Palmitoleic

Strearic

Oleic

Linoleic

Eicosanoic

Linolenic

Reading Data and Forming a Training and Validation/Test Set  
> OliveOils = read.table(file.choose(),header=T,sep=”,”) 🡨 read in **OliveOils.csv**  
> names(OliveOils)[1] "Area.name" "palmitic" "palmitoleic" "strearic" "oleic" "linoleic" [7] "eicosanoic" "linolenic"   
> dim(OliveOils)[1] 572 8> train = sample(1:572,floor(.75\*572),replace=F)

> test = -(train)

> Olive.train = OliveOils[train,]

> Olive.test = OliveOils[test,]

> dim(Olive.train)

[1] 429 8

> dim(Olive.test)

[1] 143 8  
  
We begin by fitting a simple k-NN classification (i.e. unweighted) to these data using the sknn function from the klaR library. A portion of the help file for the sknn function is shown below. This function can also be used to fit a weighted k-NN classifier as well, but the weights are determined differently than defined above, instead the weight is given by

therefore the points in the neighborhood are down-weighted based upon their distance from the target point **.**



As nearest neighbors are based on the distances, we need to make sure the variables/predictors are not on vastly different scales.

> summary(Olive.train)

Area.name palmitic palmitoleic strearic oleic

South-Apulia :152 Min. : 610 Min. : 15.0 Min. :152.0 Min. :6367

Inland-Sardinia: 45 1st Qu.:1094 1st Qu.: 81.0 1st Qu.:204.0 1st Qu.:7004

Umbria : 43 Median :1201 Median :110.0 Median :223.0 Median :7320

Calabria : 42 Mean :1228 Mean :123.9 Mean :228.6 Mean :7333

East-Liguria : 41 3rd Qu.:1359 3rd Qu.:167.0 3rd Qu.:247.0 3rd Qu.:7710

West-Liguria : 35 Max. :1753 Max. :280.0 Max. :375.0 Max. :8410

(Other) : 71

linoleic eicosanoic linolenic

Min. : 448.0 Min. : 0.00 Min. : 0.00

1st Qu.: 743.0 1st Qu.:26.00 1st Qu.: 50.00

Median :1010.0 Median :32.00 Median : 61.00

Mean : 965.5 Mean :32.04 Mean : 57.53

3rd Qu.:1176.0 3rd Qu.:41.00 3rd Qu.: 70.00

Max. :1470.0 Max. :74.00 Max. :105.00

> Olive.train[,2:8] = scale(Olive.train[,2:8])

> Olive.test[,2:8] = scale(Olive.test[,2:8])

Nearest Neighbors (k = 1)

> olive.sknn = sknn(Area.name~.,data=Olive.train,kn=1)

> yhat = predict(olive.sknn,newdata=Olive.test)

> attributes(yhat)

$names

[1] "posterior" "class"

> misclass(ypred$class,Olive.test$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 12 0 0 0 0

Coastal-Sardinia 0 10 0 0 0

East-Liguria 0 0 5 0 0

Inland-Sardinia 0 0 0 20 0

North-Apulia 0 0 4 0 3

Sicily 2 0 0 0 0

South-Apulia 0 0 0 0 0

Umbria 0 0 0 0 1

West-Liguria 0 0 0 0 0

y

fit Sicily South-Apulia Umbria West-Liguria

Calabria 4 0 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 1 0 0 1

Inland-Sardinia 0 0 0 0

North-Apulia 0 1 0 0

Sicily 3 0 0 0

South-Apulia 1 53 0 0

Umbria 0 0 8 0

West-Liguria 0 0 0 14

Misclassification Rate = 0.105

Nearest Neighbors (k = 3)  
> olive.sknn = sknn(Area.name~.,data=Olive.train,kn=3)

> yhat = predict(olive.sknn,newdata=Olive.test)

> misclass(yhat$class,Olive.test$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 12 0 0 0 0

Coastal-Sardinia 0 10 0 0 0

East-Liguria 1 0 6 0 0

Inland-Sardinia 0 0 0 20 0

North-Apulia 0 0 3 0 3

Sicily 1 0 0 0 0

South-Apulia 0 0 0 0 0

Umbria 0 0 0 0 1

West-Liguria 0 0 0 0 0

y

fit Sicily South-Apulia Umbria West-Liguria

Calabria 5 0 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 1 0 0 1

Inland-Sardinia 0 0 0 0

North-Apulia 0 0 0 0

Sicily 2 1 0 0

South-Apulia 1 53 0 0

Umbria 0 0 8 0

West-Liguria 0 0 0 14

Misclassification Rate = 0.105

Nearest Neighbors (k = 5)  
> olive.sknn = sknn(Area.name~.,data=Olive.train,kn=5)

> yhat = predict(olive.sknn,newdata=Olive.test)

> misclass(yhat$class,Olive.test$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 12 0 0 0 0

Coastal-Sardinia 0 10 0 0 0

East-Liguria 1 0 6 0 0

Inland-Sardinia 0 0 0 20 0

North-Apulia 0 0 3 0 3

Sicily 1 0 0 0 0

South-Apulia 0 0 0 0 0

Umbria 0 0 0 0 1

West-Liguria 0 0 0 0 0

y

fit Sicily South-Apulia Umbria West-Liguria

Calabria 4 0 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 0 0 0 0

Inland-Sardinia 0 1 0 0

North-Apulia 0 0 0 0

Sicily 4 1 0 0

South-Apulia 1 52 0 0

Umbria 0 0 8 0

West-Liguria 0 0 0 15

Misclassification Rate = 0.0909

We can also try weighted k-NN classification by specifying in the call to the sknn function.

> olive.wknn = sknn(Area.name~.,data=Olive.train,kn=5,gamma=4)

> ypred = predict(olive.wknn,newdata=Olive.test)

> misclass(ypred$class,Olive.test$Area.name)

Table of Misclassification

(row = predicted, col = actual)

y

fit Calabria Coastal-Sardinia East-Liguria Inland-Sardinia North-Apulia

Calabria 13 0 0 0 0

Coastal-Sardinia 0 10 0 0 0

East-Liguria 0 0 6 0 0

Inland-Sardinia 0 0 0 20 0

North-Apulia 0 0 3 0 3

Sicily 1 0 0 0 0

South-Apulia 0 0 0 0 0

Umbria 0 0 0 0 1

West-Liguria 0 0 0 0 0

y

fit Sicily South-Apulia Umbria West-Liguria

Calabria 4 0 0 0

Coastal-Sardinia 0 0 0 0

East-Liguria 1 0 0 1

Inland-Sardinia 0 0 0 0

North-Apulia 0 1 0 0

Sicily 3 0 0 0

South-Apulia 1 53 0 0

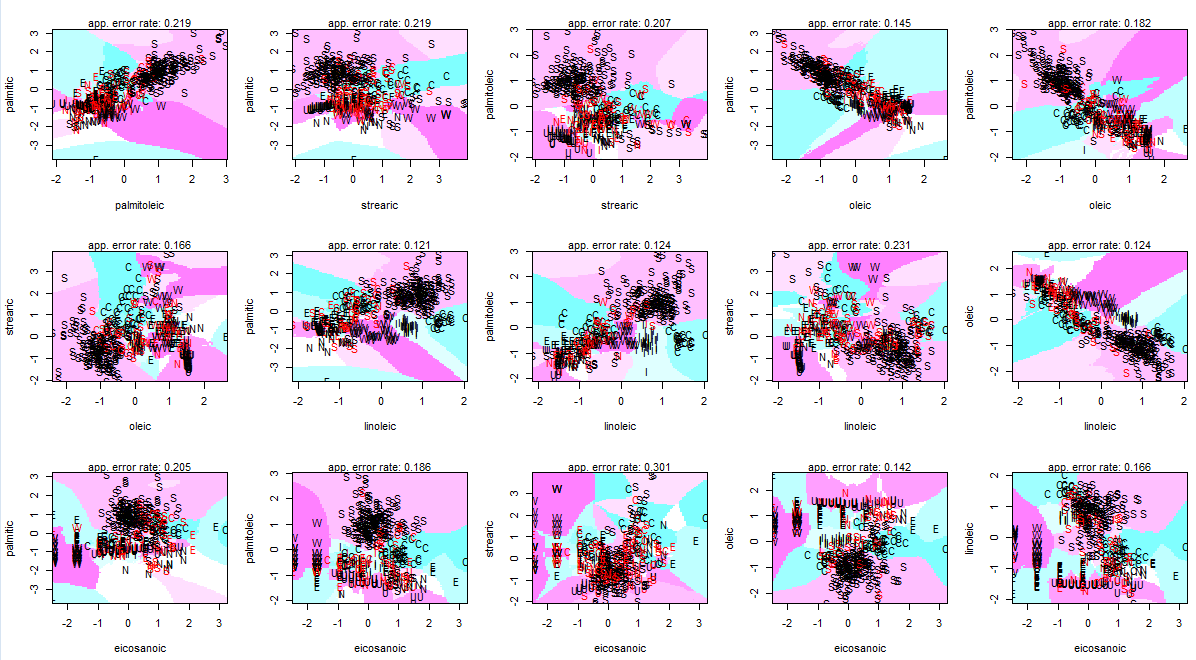
Umbria 0 0 8 0

West-Liguria 0 0 0 1

Misclassification Rate = 0.0909

After trying different combinations of neighbors (kn) and weighting parameter () this was the best misclassification rate on the test cases I could find. We can visualize the neighborhoods using variables at a time using the partimat function in the klaR library. As this utilizes only two of the fatty acids at a time in developing the classification regions the apparent error rates are much higher than we would find using all of the fatty acids.

> partimat(Area.name~.,data=Olive.train,method="sknn",kn=5,gamma=4)



**Warning: This functions takes A LONG time to run!**

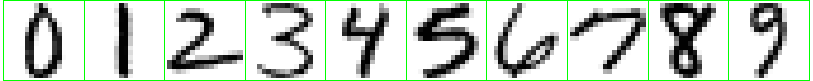
**Example 13.2 - Handwritten Digit Recognition on UPS Zip Codes**

The package ElemStatLearn is package that contains functions and data sets used in The Elements of Statistical Learning by Hastie, Tibshirani, and Friedman. One of the data sets used for illustrating classification problems is the handwritten digits in U.S. ZIP codes. These data are pre-split into a training and validation/test set (called zip.train and zip.test respectively).

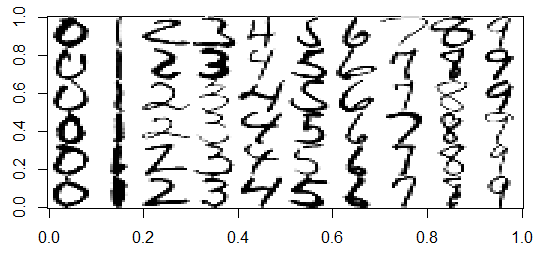
> library(ElemStatLearn)

> digits.knn = knn(zip.train[,-1],zip.test[,-1],cl=zip.train[,1],k=3)

Note the first column in zip.train and zip.test is the correct digit. The other 256 columns are 16 X 16 pixel readings of the darkness. Samples are shown below.



Another image of some handwritten digits in these data.



> misclass(digits.knn,zip.test[,1])

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 355 0 7 2 0 3 3 0 4 2

1 0 258 0 0 2 0 0 1 0 0

2 2 0 182 2 0 2 1 1 1 0

3 0 0 1 154 0 4 0 1 4 0

4 0 3 1 0 182 0 2 4 0 3

5 1 0 0 6 2 145 0 0 2 0

6 0 2 1 0 2 1 164 0 0 0

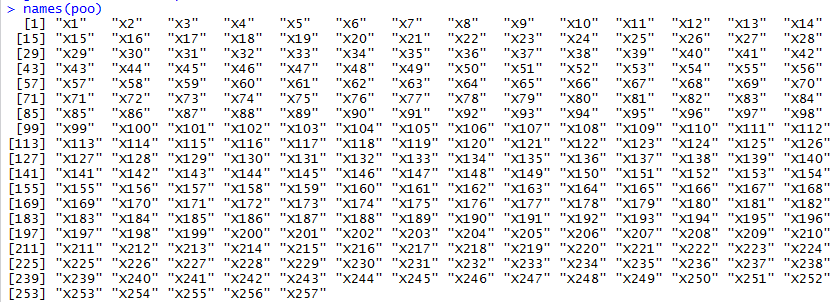
7 0 1 2 1 2 0 0 138 1 4

8 0 0 4 0 1 1 0 1 151 0

9 1 0 0 1 9 4 0 1 3 168

Misclassification Rate = 0.0548

We now consider weighted nearest-neighbor classification for these data using the kknn package and train.kknn function.

> data(zip.train)  
> poo = data.frame(zip.train)  
  
> ZIP.train = data.frame(Digits=as.factor(poo$X1),poo[,-1])  
> digit.kknn = train.kknn(Digits~.,data=ZIP.train,kmax=10,kernel=c("triangular", "rectangular", "epanechnikov", "optimal"), distance = 2)

> digit.kknn

Call:

train.kknn(formula = Digits ~ ., data = ZIP.train, kmax = 10, distance = 2, kernel = c("triangular", "rectangular", "epanechnikov", "optimal"))

Type of response variable: nominal

Minimal misclassification: 0.03319161

Best kernel: triangular

Best k: 4  
  
> data(zip.test)

> goo = data.frame(zip.test)  
> ZIP.test = data.frame(Digits=as.factor(goo$X1),goo[,-1])

> ypred = predict(digit.kknn,newdata=ZIP.test)  
  
> misclass(ypred,ZIP.test$Digits)

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 354 0 8 2 0 3 4 0 4 1

1 0 257 0 0 2 1 0 0 0 0

2 2 0 175 4 3 1 3 1 2 0

3 1 0 2 151 0 4 0 0 5 0

4 0 4 0 0 180 0 1 5 0 2

5 0 0 2 6 1 141 2 0 2 0

6 0 2 2 0 2 1 160 0 0 0

7 1 1 4 1 3 4 0 139 2 5

8 0 0 5 0 0 4 0 1 148 1

9 1 0 0 2 9 1 0 1 3 168

Misclassification Rate = 0.0668

Next we consider using the Manhattan or Taxi-cab metric, i.e. distance = 1.

> digit.kknn = train.kknn(Digits~.,data=ZIP.train,kmax=10,kernel=c("triangular", "rectangular", "epanechnikov", "optimal"), distance = 1) **🡨 Very slow!!**

> digit.kknn

Call:

train.kknn(formula = Digits ~ ., data = ZIP.train, kmax = 10, distance = 1, kernel = c("triangular", "rectangular", "epanechnikov", "optimal"))

Type of response variable: nominal

Minimal misclassification: 0.03428885

Best kernel: triangular

Best k: 9

> misclass(ypred,ZIP.test$Digits)

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 354 0 9 1 0 3 5 0 4 1

1 0 258 0 0 3 0 0 2 1 0

2 2 0 180 3 1 2 2 1 1 0

3 0 0 1 151 0 3 0 0 5 0

4 0 3 1 0 183 0 1 5 0 2

5 0 0 0 7 1 148 1 0 2 0

6 1 2 1 0 2 0 161 0 0 0

7 1 1 2 2 2 0 0 138 1 6

8 0 0 4 0 0 1 0 1 150 0

9 1 0 0 2 8 3 0 0 2 168

Misclassification Rate = 0.0578

Neither metric along with the weighted nearest-neighbor approach outperforms the simple nearest neighbor classifier from knn. We now consider using the sknn function in the klaR library to fit both unweighted and weighted nearest-neighbor classification.  
  
> digits.sknn = sknn(Digits~.,data=ZIP.train)

Prediction is SLOW!

> ypred = predict(digits.sknn,newdata=ZIP.test)

> attributes(ypred)

$names

[1] "posterior" "class"

> misclass(ypred$class,ZIP.test$Digits)

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 355 0 8 2 0 3 3 0 4 1

1 0 258 0 0 2 0 1 1 0 0

2 3 0 182 2 0 2 1 1 1 0

3 0 0 1 154 0 4 0 1 4 0

4 0 3 2 0 182 0 2 4 0 3

5 0 0 0 6 2 146 0 0 2 1

6 0 2 0 0 2 0 163 0 0 0

7 0 1 2 1 3 0 0 138 1 4

8 0 0 3 0 1 1 0 1 151 0

9 1 0 0 1 8 4 0 1 3 168

Misclassification Rate = 0.0548

Try increasing the number of neighbors (k = 6)  
  
> digits.sknn = sknn(Digits~.,k=6,data=ZIP.train)

> ypred = predict(digits.sknn,newdata=ZIP.test)  
> misclass(ypred$class,ZIP.test$Digits)

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 354 0 7 2 0 4 3 0 5 1

1 0 258 0 0 3 0 0 3 2 0

2 3 0 182 2 1 2 3 1 0 0

3 0 0 1 154 0 5 0 0 3 0

4 0 4 1 0 181 0 2 4 0 2

5 0 0 0 4 0 144 0 1 4 1

6 1 2 1 0 2 0 161 0 1 0

7 0 0 2 2 2 0 0 137 2 5

8 0 0 4 0 0 1 1 0 147 0

9 1 0 0 2 11 4 0 1 2 168

Misclassification Rate = 0.0603

Try weighting with (k = 3)  
> digits.sknn = sknn(Digits~.,data=ZIP.train,gamma=4)

> ypred = predict(digits.sknn,newdata=ZIP.test)

> misclass(ypred$class,ZIP.test$Digits)

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 355 0 6 3 0 2 0 0 5 0

1 0 255 1 0 3 1 0 1 0 0

2 2 0 183 2 1 2 1 1 1 1

3 0 0 2 154 0 4 0 1 6 0

4 0 6 1 0 182 0 2 4 1 2

5 0 0 0 5 1 145 3 0 1 0

6 0 2 0 0 2 2 164 0 0 0

7 1 1 2 0 2 0 0 139 1 4

8 0 0 2 0 1 3 0 0 148 1

9 1 0 0 2 8 1 0 1 3 169

Misclassification Rate = 0.0563

Try weighting with (k = 6)

> digits.sknn = sknn(Digits~.,k=6,data=ZIP.train,gamma=4)

> ypred = predict(digits.sknn,newdata=ZIP.test)

> misclass(ypred$class,ZIP.test$Digits)

Table of Misclassification

(row = predicted, col = actual)

y

fit 0 1 2 3 4 5 6 7 8 9

0 355 0 6 3 0 2 0 0 5 0

1 0 255 1 0 3 1 0 1 0 0

2 2 0 183 2 1 2 1 1 1 1

3 0 0 2 154 0 4 0 1 6 0

4 0 6 1 0 182 0 2 4 1 2

5 0 0 0 5 1 145 3 0 1 0

6 0 2 0 0 2 2 164 0 0 0

7 1 1 2 0 2 0 0 139 1 4

8 0 0 2 0 1 3 0 0 148 1

9 1 0 0 2 8 1 0 1 3 169

Misclassification Rate = 0.0563

We can use Monte Carlo Split-Sample cross-validation to assess the predictive performance of classifiers as well. Below is code for a MC/SS cross-validation function that uses the nearest-neighbor classifier fit using the train.kknn function in the kknn library.

kknn.sscv = function(train,y=train[,1],B=25,p=.333,kmax=3,kernel="optimal",distance=2) {

y = as.factor(y)

data = data.frame(y=y,train[,-1])

n = length(y)

cv <- rep(0,B)

leaveout = floor(n\*p)

for (i in 1:B) {

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

fit <- train.kknn(y~.,data=data[-sam,],kmax=kmax,kernel=kernel,distance=distance)

ypred = predict(fit,newdata=data[sam,])

tab <- table(y[sam],ypred)

mc <- leaveout - sum(diag(tab))

cv[i] <- mc/leaveout

}

cv

}

The arguments are the training data which will then be split randomly times into a training and validation sets in each Monte Carlo loop. The nearest-neighbor classifier will fit to the training portion of the data and the misclassification rate from predicting the validation cases will be saved. The data set (train) must have the factor response () in the first column! The remaining settings should be self-explanatory as they are all settings used in the call to the train.kknn function.

We begin by forming a full dataset consisting of both the zip.train and zip.test data sets from the ElemStatLearn package. Using a large value for is not advised, at least for these data, as the single pass through the Monte Carlo loop is quite slow!!

> ZIPdigits = data.frame(rbind(ZIP.train,ZIP.test))   
> dim(ZIPdigits) 🡨 all cases available are used as the full data set.

[1] 9298 257

> results = kknn.cv(ZIPdigits,B=10,kmax=4,kernel="triangular")

> results

[1] 0.04748062 0.04457364 0.04554264 0.04554264 0.04554264 0.04812661 0.05006460 0.04909561 0.04909561

[10] 0.04618863

> summary(results)

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

0.04457 0.04554 0.04683 0.04713 0.04885 0.05006

> results = kknn.cv(ZIPdigits,B=10,kmax=9,kernel="triangular",distance=1) 🡨 **SUPER SLOW!!**

**(About 45 min.)**> results

[1] 0.04683463 0.04425065 0.04618863 0.04295866 0.04295866 0.04328165 0.04521964 0.04521964 0.04618863

[10] 0.04231266  
> summary(results)

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

0.04231 0.04304 0.04474 0.04454 0.04595 0.04683

**Example 13.3: Classifying Species of Water Bear**Water bears (*Phylum Tardigrada*) are microscopic invertebrates often considered among the least known animal groups. Water bears are known for their ability to withstand environmental extremes by undergoing cryptobiosis, a kind of suspended animation. This ability allows them to occupy a wide variety of habitats, including stream sediments as well as moss and lichens that periodically wet and dry. Water bears are important members of these understudied “meiofaunal” communities. Some species are herbivorous and some are carnivorous, but the natural history and ecology of most species are unknown. These data were collected by Paul Bartels a biologist at Warren Wilson College who has been studying water bears in the Great Smoky Mountain National Park. He believes that there may be as many as 100 species living in the GSMNP, 12 of which are new species to science. Here we consider only 4 species and use six different body measurements as predictors to classify them.



**Datafile: WaterBears.csv**

> WaterBears = read.table(file.choose(),header=T,sep=”,”) # read in **WaterBears.csv**

> WaterBears = WaterBears[,-1]

> WaterBears.X = scale(WaterBears[,-1])

> WaterBears2 = data.frame(Species=WaterBears$Species,WaterBears.X)  
> head(WaterBears2)

Species SSI BTWa BTWs BTWp BTWL BTPA

1 smokiensis 0.07958566 1.3204245 1.8986322 1.9946594 1.8406373 1.243460

2 smokiensis 0.12270916 0.4647074 1.3268796 1.5145209 1.3558442 1.477321

3 smokiensis 0.42936520 -0.9663225 0.2106008 0.1317219 0.2246602 1.243460

4 smokiensis 0.71206374 0.1069499 0.8095797 0.9345135 0.8710510 1.126529

5 smokiensis -0.04020186 0.2423176 0.8990375 1.1188867 0.8710510 1.243460

6 smokiensis 1.01871978 0.2471522 0.7901323 1.0651112 0.7094533 1.126529

> wb.sknn1 = sknn(Species~.,data=WaterBears2)

> misclass(predict(wb.sknn1)$class,WaterBears2$Species)

Table of Misclassification

(row = predicted, col = actual)

y

fit bohleberi roanensis smokiensis unakaensis

bohleberi 10 0 0 0

roanensis 0 9 0 1

smokiensis 0 0 9 0

unakaensis 0 1 1 10

Misclassification Rate = 0.0732

>

As we have seen previously, a weighted nearest neighbor approach can yield superior results. A weighted nearest neighbor approach gives more weight to observations that are close to the target point with the weight being determined from the distance using:

).

> wb.sknn2 = sknn(Species~.,data=WaterBears2,gamma=4)

> misclass(predict(wb.sknn2)$class,WaterBears2$Species)

Table of Misclassification

(row = predicted, col = actual)

y

fit bohleberi roanensis smokiensis unakaensis

bohleberi 10 0 0 0

roanensis 0 10 0 0

smokiensis 0 0 10 0

unakaensis 0 0 0 11

Misclassification Rate = 0

The weighted-knn approach gives no misclassifications on the training data. Cross-validation of methods for classification is still critical, especially when considering methods that have the potential to over fit the training data.

**Monte Carlo Split-Sample Cross-validation function for knn()**  
knn.cv = function(train,y,B=25,p=.333,k=3) {

y = as.factor(y)

data = data.frame(y,train)

n = length(y)

cv <- rep(0,B)

leaveout = floor(n\*p)

for (i in 1:B) {

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

pred <- knn(train[-sam,],train[sam,],y[-sam],k=k)

tab <- table(y[sam],pred)

mc <- leaveout - sum(diag(tab))

cv[i] <- mc/leaveout

}

cv

}

**Monte Carlo Split-Sample Cross-validation function for sknn()**

sknn.cv = function(train,y,B=25,p=.333,k=3,gamma=0) {

y = as.factor(y)

data = data.frame(y,train)

n = length(y)

cv <- rep(0,B)

leaveout = floor(n\*p)

for (i in 1:B) {

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

temp <- data[-sam,]

fit <- sknn(y~.,data=temp,k=k,gamma=gamma)

pred = predict(fit,newdata=train[sam,])$class

tab <- table(y[sam],pred)

mc <- leaveout - sum(diag(tab))

cv[i] <- mc/leaveout

}

cv

}

> names(WaterBears2)

[1] "Species" "SSI" "BTWa" "BTWs" "BTWp" "BTWL" "BTPA"

> X = WaterBears2[,-1]

> y = WaterBears2[,1]

> results = knn.cv(X,y,B=1000,p=.25)

> summary(results)

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

0.0000 0.1000 0.1000 0.1502 0.2000 0.5000

> results = sknn.cv(X,y,B=1000,p=.25)

> summary(results)

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

0.0000 0.1000 0.1000 0.1431 0.2000 0.5000

> results = sknn.cv(X,y,B=1000,p=.25,gamma=4) <- Weighted-knn (

> summary(results)

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

0.0000 0.1000 0.1000 0.1238 0.2000 0.5000

> results = sknn.cv(X,y,B=1000,p=.25,gamma=2)

> summary(results)

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

0.0000 0.0000 0.1000 0.1131 0.2000 0.5000

> results = sknn.cv(X,y,B=1000,p=.25,gamma=1)

> summary(results)

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

0.0000 0.1000 0.1000 0.1303 0.2000 0.5000

> partimat(Species~.,data=WaterBears,method=”sknn”,kn=3,gamma=2) 🡨 Fairly Slow!

