Classification: the K Nearest Neighbor Method

Richard E. Plant

June 1, 2020

Additional topic to accompany Spatial Data Analysis in Ecology and Agriculture using R, Second Edition

http://psfaculty.plantsciences.ucdavis.edu/plant/sda2.htm

Chapter and section references are contained in that text, which is referred to as SDA2.

1. Introduction

Section 8.4 of SDA2 discussed the generalized linear model (GLM) and its application to the construction of a logistic regression model to predict the presence or absence of blue oaks (Quercus douglasii, represented by the variable QUDO) in Data Set 2. This may be viewed as a classification problem in which each location is classed as either QUDO = 0 (absent) or QUDO = 1 (present). Suppose, however, that we have a classification problem in which there are more than two classes. Although logistic regression can be used in that case as well, this method is not applied as frequently. Instead, other methods, including the K nearest neighbor (KNN) method, support vector machines, and discriminant analysis, are commonly used. Here we introduce classification methods by discussing the KNN method. As with regression, these classification methods were originally developed for non-spatial data, and so we will have to pay attention to how the spatial nature of our data affects their results as well as how we might use this spatial nature to our advantage. By the way, the KNN method should not be confused with K-means clustering, which is a completely different method, used for unsupervised classification in which there is no training data. K-means clustering is discussed in SDA2 Section 12.6.1. In this context, KNN is a supervised classification method in which there is a training data set whose classes are known.

2. The data set

Our application will be the classification of oak woodlands in California using an augmented version of Data Set 2. In SDA2 that data set was only concerned with the presence or absence of blue oaks, denoted in Data Set 2 by the variable QUDO. The original Wieslander survey on which this data set is based (Wieslander, 1935), however, contained records for other oak species.
as well. Here we employ an augmented version of Data Set 2, denoted Data Set 2A, that also contains records for coast live oak (\textit{Quercus agrifolia}, \textit{QUAG}), canyon oak (\textit{Quercus chrysolepis}, \textit{QUCH}), black oak (\textit{Quercus kelloggii}, \textit{QUKE}), valley oak (\textit{Quercus lobata}, \textit{QULO}), and interior live oak (\textit{Quercus wislizeni}, \textit{QUWT}). The data set contains records for the basal area of each species as well as its presence/absence, but we will only use the presence/absence data here. Of the 4,101 records in the data set, 2,267 contain only one species and we will analyze this subset, denoted Data Set 2U.

```r
> data.Set2A <- read.csv("set2\set2Adata.csv", header = TRUE)
> data.Set2U <- data.Set2A[which(data.Set2A$QUAG + data.Set2A$QUWI +
+    data.Set2A$QULO + data.Set2A$QUDO + data.Set2A$QUKE +
+    data.Set2A$QUCH == 1),]
```

Thus, the classification problem will be to predict which of the six classes a record falls into based on the data contained in that record. To continue the analysis we will establish a factor valued variable \textit{species} that takes on appropriate values for each of the species.

```r
> species <- character(nrow(data.Set2U))
> species[which(data.Set2U$QUAG == 1)] <- "QUAG"
> species[which(data.Set2U$QUWI == 1)] <- "QUWI"
> species[which(data.Set2U$QULO == 1)] <- "QULO"
> species[which(data.Set2U$QUDO == 1)] <- "QUDO"
> species[which(data.Set2U$QUKE == 1)] <- "QUKE"
> species[which(data.Set2U$QUCH == 1)] <- "QUCH"
> data.Set2U$species <- as.factor(species)
```

Let’s take a look at the species counts.

```r
> table(data.Set2U$species)
QUAG QUCH QUDO QUKE QULO QUWI
 551   99  731  717   47  122
```

The data set is highly unbalanced. The numbers of positive values for three of the species are reasonably close in size and high, and the other three are low. We can use the \texttt{mapview} package \cite{Appelhans:2017} discussed earlier to view the distribution of the six species:

```r
> library(mapview)
> library(sp)
> coordinates(data.Set2U) <- c("Longitude", "Latitude")
> proj4string(data.Set2U) <- CRS("+proj=longlat +datum=WGS84")
> mapview(data.Set2U, zcol = "species", legend = TRUE,
+    col.regions = c("red", "yellow", "green", "blue", "orange", "purple"))
```

Fig. 1 shows the result.
Figure 1. Spatial distribution of the six oak species.

We can see at a glance that the distance from the Pacific Coast, represented by the variable \textit{CoastDist}, is highly predictive of the species. This is a situation similar to that encountered by Venables and Dichmont (2004) in their study of tiger prawn fisheries off the Australian coast. The focus in SDA2 was generally on using data analyses to develop explanations for the ecological processes underlying the data. As was discussed in the book, this is a fraught problem in the sense that it involves using the statistical tools at the bleeding edge of their validity. The alternative use is simple prediction, in which we just want to predict the class into which a data record falls (in our case, which species is present). For example, distance from the coast clearly does not have a direct ecological influence on oak species; rather, it works through its association with processes such as climate and possibly soil conditions. In SDA2 we used the terms \textit{exogenous} and \textit{endogenous} to describe the difference between these types of influence. As mentioned in SDA2, these terms are motivated by their use in economics (Wooldridge, 2013), but they are defined differently. In the SDA2 context, distance from the coast is an exogenous variable with no explanatory value, but great predictive value. In our discussion of the K nearest
neighbor method we will assume that we are interested strictly in prediction, and so we will refer to the attribute variables involved as predictor variables. As it is traditionally used, the K nearest neighbors are considered to be the K data records whose values at some chosen subset of the predictor data are closest to those of the point in question. With spatial data it is also possible, of course, to base the classification on the K records geographically closest to the point in question, and we will consider this as well.

3. Introducing the K nearest neighbor method

Rather than jumping into the analysis of the full data set right away, we will introduce the KNN method by considering the restricted problem of QUDO presence/absence on a restricted data set, and we will limit the predictor variables on which to base the classification to mean annual temperature, MAT, and mean annual precipitation, Precip. Because the scale of MAT is very different from that of Precip, we will normalize both variables, using the combined data sets. Here is the code.

```r
> normalize <- function(x) (x - mean(x)) / sqrt(var(x))
> oaks <- sample(which(data.Set2U$QUDO == 1), 25)
> no.oaks <- sample(which(data.Set2U$QUDO == 0), 25)
```

We will randomly select 25 data points from the original Data Set 2 for which QUDO = 1 and 25 for which QUDO = 0 and use this as the training data set. We will select another 25 of each type as a test data set. Here is the code for the training data. True confession: I fudged around with the random number seeds until I got a particular result I wanted. We will see this result in the discussion below.

```r
> set.seed(1)
> oaks <- round(runif(25, 0, length(which(data.Set2U$QUDO == 1))))
> no.oaks <- round(runif(25, 0, length(which(data.Set2U$QUDO == 0))))
> Set2.Train <- data.Set2U[c(oaks, no.oaks),]
> Set2.Train$Color <- "red"
> Set2.Train$Color[which(Set2.Train$QUDO == 0)] <- "green"
> with(Set2.Train@data, plot(x = MAT, y = Precip, pch = 16, col = Color,
+ main = "Training Data")
> legend(14.75, 1500, c("QUDO = 0", "QUDO = 1"), pch = 16,
+ col = c("green", "red"))
```

Fig. 2a shows the resulting plot. The code for the test data set is similar. Fig. 2b shows that plot.
We now have everything set up to introduce the KNN method. The method is very simple. For any set of values of the predictor variables (in this case, any value of the pair \((\text{MAT}, \text{Precip})\) suitably normalized), the corresponding class is chosen to be for some pre-specified \(K\) the value of the majority of the \(K\) nearest neighbors. To keep things simple we will always choose \(K\) to be odd. Ties are still possible with more than two predictors and are settled at random.

Although KNN is a very simple classification system, in one theoretical sense it is fairly close to optimal. To understand the theory, we will need some notation. We will follow the notational conventions of SDA2, so that variables are indicated with upper case letters and lower case is reserved for spatial coordinates. Using the notation of Fukunaga (1990) and Duda et al. (2001), let the two classes \(\text{QUDO} = 0\) and \(\text{QUDO} = 1\) be denoted \(\omega_0\) and \(\omega_1\), respectively. We will use the set theoretic notion \(Y_i \in \omega_j\) to mean that \(Y_i\) is contained in the set \(\omega_j\), so that for example if \(j = 0\) then the value of \(\text{QUDO}\) at any location \(i\) such that \(Y_i \in \omega_j\) is 0.

To bring things back to our case, let \(\{X_1, X_2, \ldots, X_n\}\) be a sequence of predictor values. In our case \(X_i\) represents the \(i^{th}\) instance of the normalized pair \((\text{MAT.n}, \text{Precip.n})\). Let \(\{Y_1, Y_2, \ldots, Y_n\}\) represent the corresponding values of \(\text{QUDO}\). Following James et al. (2013), we call these the class labels. Let \(\hat{Y}_i\) represent the predicted value of class label \(Y_i\) based on the classification
algorithm. In order to assess the quality of a classification algorithm we define the error rate follows. Let $I(\hat{Y}_i \neq Y_i)$ be an indicator function whose value is 1 if $\hat{Y}_i \neq Y_i$ and 0 otherwise (James et al, 2013, p. 37). The error rate of the classification algorithm is then defined as

$$E = \frac{1}{n} \sum_{i=1}^{n} I(\hat{Y}_i \neq Y_i).$$

(1)

In other words, $E$ is the fraction of records that are incorrectly classified. Other notions of accuracy are also possible (Exercise 1).

The following discussion makes use of Bayesian statistics. This topic is covered in Chapter 14 of SDA2 and if you are unfamiliar with the concepts involved, you should review that chapter or a similar discussion. In the Bayesian context, suppose we knew the prior probabilities $P\{Y_i \in \omega_j\}$, $j = 0, 1$, for each point in the data space. These are the probabilities before any evidence is observed that $Y_i$ is a member of each of the two classes. We don’t know them, but suppose we did. After the data $X$ are observed, for each class $\omega_j$ we could then compute the posterior probabilities $P\{Y_i \in \omega_j \mid X\}$ from Bayes’ rule,

$$P\{Y_i \in \omega_j \mid X\} = \frac{P\{X \mid Y_i \in \omega_j\}P\{Y_i \in \omega_j\}}{P\{X\}}.$$

(2)

Since the denominator $P\{X\}$ is the same for $j = 0$ and $j = 1$, we can take it out of the decision. Therefore, we would maximize the conditional posterior probability $P\{Y_i \in \omega_j \mid X\}$ if given the data $X$ we chose the class to maximize the numerator $P\{X \mid Y_i \in \omega_j\}P\{Y_i \in \omega_j\}$. This is called the Bayes classifier (Fukunaga, 1990, p. 51).

For real data we do not know $P\{Y_i \in \omega_j\}$. When we have training data $X^{Tr}$, such as that of Fig. 2a however, we know $P\{\omega_j\}$ for these data. Namely, if $X^{Tr}_i = j$ then $P\{Y^{Tr}_i \in \omega_j\} = 1$, and otherwise it is zero. The KNN method approximates the Bayes classifier so that for each point $i$ in the data space (represented by the grid of small dots in Fig. 3 below), based on the training data $X^{Tr}$ the KNN estimates the class $j$ for which $P\{Y_i \in \omega_j \mid X\}$ is largest. Specifically, for each point $i$, let $KNN_i$ represent the $K$ points in the data space closest to point $i$. Then we estimate $P\{\omega_j \mid X\}$ as
\[ P(\omega_j \mid X) \equiv \frac{1}{K} \sum_{i \in \text{KNN}_i} I(Y_i \in \omega_j), \]  

(3)

where \( I(Y_i \in \omega_j) = 1 \) if \( Y_i \in \omega_j \), and it equals 0 otherwise. In other words, for each point \( i \) we define the class label of that point to be the class label of the majority of its nearest neighbors.

We can define the training error rate \( E_{Tr} \) to be

\[ E_{Tr} = \frac{1}{n} \sum_{X_i \in X^n} I(\hat{Y}_i \neq Y_i). \]  

(4)

This is somewhat analogous to the \( R^2 \) of ordinary linear regression in that it measures the fit based on the data used to develop the fit. If we have a test data set \( X^{Te} \) such as that shown in Fig. 2b, then the test error rate

\[ E_{Te} = \frac{1}{n} \sum_{X_i \in X^{Te}} I(\hat{Y}_i \neq Y_i) \]  

(5)

computed using the test data provides a better indication of the quality of the classifier. It turns out (Hastie et al., 2009, p. 465) that (roughly speaking) the Bayes classifier has the lowest possible test error rate of any classifier, and that the KNN method often has a test error rate reasonably close to that of the Bayes classifier.

To graphically demonstrate the method we will create a 100 by 100 grid of points within the data space defined by the range of normalized values of MAT and Precip.

```r
> x.min <- min(data.Set2U$MAT.n)
> x.max <- max(data.Set2U$MAT.n)
> y.min <- min(data.Set2U$Precip.n)
> y.max <- max(data.Set2U$Precip.n)
> n.cells <- 100
> xcell <- (x.max - x.min) / n.cells
> ycell <- (y.max - y.min) / n.cells
> grid.vals <- matrix(0, n.cells^2, 2)
> ij <- 1
> for (i in 1:n.cells)
+    for (j in 1:n.cells){
+       grid.vals[ij,1] <- x.min + xcell/2 + xcell * (j-1) %% n.cells
+       grid.vals[ij,2] <- y.max - ycell/2 - ycell * (i-1) %% n.cells
+       ij <- ij + 1
+    }
```

Next we will write a function to carry out the KNN analysis. Not surprisingly, such a function exists; it is the function \texttt{knn()} in the package \texttt{class} (Venables and Ripley, 2002). We will turn to it eventually, but for purposes of exposition we start with a set of homemade functions.

Because we are going to explore different values of \( K \) and compare the results, we will organize
the code in the form of three functions, one to compute the $K$ nearest neighbors, one to classify the data and the one to plot the results. The first function is called `KNN()`, and is actually quite simple.

```r
> KNN <- function(K, X, Data){
+  # K = number of neighbors, X = location to be classified
+  # Data = locations of Training data
+  D <- numeric(nrow(Data))
+  for (i in 1:nrow(Data)) D[i] <- (X[1] - Data[i,1])^2 +
+      (X[2] - Data[i,2])^2
+  min.d <- sort(D)[1:K]
+  M <- numeric(K)
+  for (i in 1:K)
+     for (j in 1:length(D))
+         if (abs(min.d[i] - D[j]) < 1e-10) M[i] <- j
+  return(M)
+ }
```

Here the argument $K$ specifies the number $K$ of neighbors, $X$ specifies the location in the predictor data space of the point in question, and $Data$ is a three-column matrix that specifies the locations of the potential neighbors and their class labels. Note that a point in the training data set is considered as one of its own neighbors, and if $K = 1$ then it is the only neighbor.

The second function is called `classify()` and is also fairly simple.

```r
> classify <- function(K, X.Class, X.Train, Y.Train){
+  # K = number of classes, X.Class = points at which Y is to be classified
+  # X.Train = training data X values, Y.Train = training data Y values
+  # Y will hold the classification values
+  Y <- numeric(nrow(X.Class))
+  for (i in 1:nrow(X.Class)){
+     # Determine the K nearest neighbors
+     NNK <- KNN(K, X.Class[i,], X.Train)
+     # Assign the points the value of the K nearest neighbors
+     sumK <- 0
+     for (k in 1:K) sumK <- sumK +
+        if (sumK >= round(K/2 + .001)) Y[i] <- 1
+  }
+  return(Y)
+ }
```

The function returns the class label, either 0 or 1, for each of the points in $X.Class$, based on the training data $X.Train$ and $Y.Train$. The code for the function `plot.classes(title, X.grid, Y.grid, X.pts, Y.pts, misclass)`, which plots the figures, is fairly long and not particularly interesting. Its arguments $X.grid$ and $Y.grid$ specify the grid points to be plotted to illustrate the regions $a_0$ and $a_1$; $X.pts$ and $Y.pts$ are the points at which the class labels are estimated; and $misclass$, if it is not zero, is a vector of the same length as $Y.pts$ indicating which points are
misclassified. If \texttt{misclass} is specified then blue circles are drawn around the misclassified points.

We will first run the code for \( K = 5 \) to classify the training data.

```r
> X.Class <- grid.vals
> X.Train <- with(Set2.Train@data, cbind(MAT.n, Precip.n))
> Y.Train <- Set2.Train$QUDO
> # Classification with 5 classes
> K <- 5
> grid.QUDO <- classify(K, X.Class, X.Train, Y.Train)
> # Classify the training data
> Pred.Train <- classify(K, X.Train, X.Train, Y.Train)
```

Next we compute the error rate.

```r
> miscl <- which((Y.Train == 1 & Pred.Train == 0) |
+     (Y.Train == 0 & Pred.Train == 1))
> print(E.Train <- length(miscl) / length(Y.Train))
[1] 0.14
```

![KNN Training Data Classification, K = 5](image)

![KNN Test Data Classification, K = 5](image)

Figure 3. (a) Classification regions in data space for \( K = 5 \); with misclassified points circled in blue. (b) Same figure for the test data. In both figures, the small green dots represent the predicted subspace \( \omega_0 \) in which \( QUDO = 0 \), and the small red dots represent the predicted subspace \( \omega_1 \) in which \( QUDO = 1 \), based on the KNN estimate of the Bayes classifier.

Now we plot the results, which are shown in Fig. 3(a). Next we repeat the process for the test data.
X.Test <- cbind(Set2.Test$MAT.n, Set2.Test$Precip.n)
Y.Test <- Set2.Test$QUDO
# Classify the test data
Pred.Test <- classify(K, X.Test, X.Train, Y.Train)
# Compute the test error rate
misc1 <- which((Y.Test == 1 & Pred.Test == 0) |
+   (Y.Test == 0 & Pred.Test == 1))
print(E.Test <- length(misc1) / length(Y.Test))
[1] 0.26

There is some interspersion of the class labels, so that the set \( \omega_1 \) is somewhat disconnected and localized. As a result, the test data error rate is worse than that of the training data.

Now let’s try repeating the procedure for other values of \( K \). Fig. 4 shows the results.

![Classification Results](image)

(a) K = 1, (b) K = 5, and (c) K = 10.

<table>
<thead>
<tr>
<th></th>
<th>( K = 1 )</th>
<th>( K = 5 )</th>
<th>( K = 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_{Train} )</td>
<td>0</td>
<td>0.14</td>
<td>0.28</td>
</tr>
<tr>
<td>( E_{Test} )</td>
<td>0.34</td>
<td>0.26</td>
<td>0.32</td>
</tr>
</tbody>
</table>
Table 1. Error rates vs. $K$

Table 1 shows the training and test error rates for the three values of $K$. This is where my fudging with the random number seeds came in. I wanted $K = 5$ to be in the middle with the training data and do the best with the test data. This is “textbook” behavior, although as we will see in the next section it doesn’t always work out exactly this way. The number of neighbors $K$ is somewhat analogous to the number of splines in the fit of the generalized additive model (SDA2, Section 9.2), except that it works in the opposite direction. In the case of the KNN method, a very low value generates a good fit to the training data, but may not fit the test data as well, and a value that is too high reduces the fit to the training data. This an example of the bias-variance tradeoff, which is discussed in Section 8.2 of SDA2. We will meet the bias-variance tradeoff again in the next section in the context of the number of predictors. The theory of the bias-variance tradeoff is a bit more complicated in classification problems than it is in regression problems such as those considered in SDA2. Nevertheless, the principle is the same. This is that the mean square error can be represented schematically as

$$MSE = \sum_{i=1}^{n} \text{bias}_i^2 + \sum_{i=1}^{n} \text{var}\{\hat{Y}_i\}$$

(6)

Here the bias represents the error caused by the model not fitting the data correctly and the variance represents the deviation due to random error in the data. In regression high bias is associated with underfitting the data due to too simple a model and high variance is associated with overfitting the data due to too complex a model. Regarding the bias-variance tradeoff in the case of the KNN method, we can do no better than quoting James et al. (2013, p. 105): “A small value for $K$ provides the most flexible fit, which will have low bias but high variance. This variance is due to the fact that the prediction in a given region is entirely dependent on just one observation. In contrast, larger values of $K$ provide a smoother and less variable fit; the prediction in a region is an average of several points, and so changing one observation has a smaller effect. However, the smoothing may cause bias by masking some of the structure in $f(X)$.”
4. The full KNN method

Now that we have introduced the method with two classes and two predictors, let’s move on to the more general case. In Section 2 we saw that the full species data set is highly unbalanced, with three species having over 500 data records and three species having far fewer. In this section we will consider only the three most common species to avoid this imbalance. In place of the homemade function of the last section we will employ the function knn() of the package class (Venables and Ripley, 2002). First we check to see that the results of using this function match those obtained in the last section

```r
> library(class)
> # Check that knn() generates the same classification as Section 3
> train.knn <- data.frame(Set2.Train$MAT.n, Set2.Train$Precip.n)
> test.knn <- data.frame(Set2.Test$MAT.n, Set2.Test$Precip.n)
> cl.knn <- as.factor(Set2.Train$QUDO)
> Pred.knn <- as.numeric(knn(train.knn, test.knn, cl.knn, 5)) - 1
> Pred.Sec3 <- classify(5, X.Test, X.Train, Y.Train)
> all.equal(Pred.Sec3, Pred.knn)
[1] TRUE
```

By comparing the code above with that of Section 3 you can figure out what the arguments of knn() represent. The first argument is the training predictor data $X^{Tr}$, the second is the test predictor data $X^{Te}$, the third is the training class label data $Y^{Tr}$, and the fourth is the number of classes.

Before proceeding further we need to pause and consider our objective. In an actual application we would want to use the KKN method to classify objects whose class label we do not know. In our case, however, we know all the objects, and the objective is not to classify them but to learn how to use the method effectively. This can be accomplished by using cross validation to estimate the error; in our case, we will use ten-fold cross validation. This is discussed in SDA2 Section 6.3.3 and also by Hastie et al. (2009) and James et al. (2013). In brief, we will divide the entire data set at random into ten subsets, or folds. We then sequentially hold out one of the folds, train the algorithm on the nine remaining folds, and test it on the held-out fold. We repeat this ten times, giving us ten estimates of the classification error. This provides the ability to evaluate the method using a summary statistic based on these ten errors. We will use the root mean squared error, or RMSE as our representation of the classification error. We could also use the mean error rate; the RMSE penalizes especially large errors. The package e1071 (Meyer et
al., 2017) contains a function `tune()` that does cross validation, but it is worth the trouble to write our own, both to see how it works and because we will need to make another version later. Here is the code for the function `cross.validate()`, which returns the RMSE.

```r
> cross.validate <- function(data.set, tr.n, cl.n, K) {
+   n.rand <- sample(1:nrow(data.set))
+   set.seed(123)
+   n <- nrow(data.set) / 10
+   folds <- matrix(n.rand, nrow = n, ncol = 10)
+   SSE <- 0
+   for (i in 1:10) {
+     test.fold <- i
+     if (i == 1) train.fold <- 2:10
+     if (i == 10) train.fold <- 1:9
+     if ((i > 1) & (i < 10)) train.fold <- c(1:(i-1), (i+1):10)
+     train <- data.frame(data.set[folds[,train.fold], tr.n])
+     test <- data.frame(data.set[folds[,test.fold],
+                                tr.n])
+     cl <- data.set[folds[,train.fold], cl.n]
+     true.sp <- data.set[folds[,test.fold],
+                                cl.n]
+     Pred.1 <- knn(train, test, cl, K)
+     E <- length(which(Pred.1 != true.sp)) / length(true.sp)
+     SSE <- SSE + E^2
+   }
+   RMSE <- sqrt(SSE / 10)
+   return(RMSE)
+ }
```

The arguments are respectively `data.set`, a data frame containing all the data, `tr.n`, a vector whose elements are the column number or numbers of `data.set` containing the training data, `cl.n`, a number whose value is the column number of `data.set` containing the class labels, and `K`, the number of neighbors on which to base the classification.

```r
> # Classify the three most common species
> data.Set2U.3sp <- subset(data.Set2U, subset = (species == "QUAG") | (species == "QUDO") | (species == "QUKE"))
> nrow(data.Set2U.3sp)
[1] 1999
```

There are precisely one fewer data values than we need to have a nice round number of records, so (avert your eyes please) we will double count the first record to get an even 2000.

```r
> data.Class <- data.Set2U.3sp@data
> data.Class[2000,] <- data.Class[1,]
```

As was mentioned in Section 1, our situation is very similar to that encountered by Venables and Dichmont (2004) in that distance from the coast obviously has great predictive power for these
data (Fig. 1). Therefore our first test will be to use \textit{CoastDist} as the sole predictor. Since we plan to add others, we will normalize it first. \textbf{NOTE:} the function \texttt{knn()} does some randomization itself, so your results may vary slightly from mine.

\begin{verbatim}
> # First classify according to CoastDist alone
> data.Class$CD.n <- normalize(data.Class$CoastDist)
> tr <- which(names(data.Class) == "CD.n")  # Column(s) of the training data
> cl <- which(names(data.Class) == "species")  # Columns of the classification
> K <- 5
> print(RMSE.1 <- cross.validate(data.Class, tr, cl, K))
[1] 0.1833712
\end{verbatim}

Obviously \textit{CoastDist} does a pretty good job all by itself. We can visualize the data by plotting a histogram (Fig. 5).

\begin{verbatim}
> ggplot(data.Class, aes(x = CoastDist, fill = species)) +
+   geom_histogram(binwidth = 10, alpha = 0.5, position = "dodge") +
+   ggtitle("Histogram of Coast Distance by Species")
\end{verbatim}

![Histogram of Coast Distance by Species](image)

\textbf{Figure 5.} Histograms of the three most common species by \textit{CoastDist}.  

The Additional Topic on the comparison of supervised classification methods contains a discussion of variable selection for KNN predictor data. Here we will do something that is very inelegant but makes the expository point. We will simply add other predictor variables one by one, in the order of appearance in the data frame, and see what happens to the RMSE.

```r
> X <- c(3,4,5,6,8,9,12,14,15,17,18,19)
> names(data.Class)[X]
[1] "MAT"     "Precip"  "JuMin"   "JuMax"   "JaMin"   "JaMax"
[7] "GS32"    "PE"      "ET"      "Texture" "AWCAvg"  "Permeab"
> RMSE <- numeric(1 + length(X))
> RMSE[1] <- RMSE.1
> for (i in 1:length(X)){
+   data.Class[, (ncol(data.Class)+1)] <- normalize(data.Class[,X[i]])
+   tr <- c(tr, ncol(data.Class))
+   RMSE[i+1] <- cross.validate(data.Class, tr, cl, K)
+ }
> RMSE.df = data.frame(N = 1:length(RMSE), RMSE = RMSE)
> ggplot(data = RMSE.df, aes(x = N, y = RMSE)) +
+   geom_point() + geom_line() + ggtitle("RMSE vs. n")
```

Fig. 6a shows the result.

![Figure 6a](image)

Figure 6. a) Plot of the RMSE versus number of predictor variables, b) Plot of the RMSE versus $K$.

As expected the bias-variance tradeoff manifests itself in the reduction of the RMSE up to a point, followed by its increase as the number of predictors is increased further. We can make the same test against increasing values of $K$ (code not shown), which gives the result shown in Fig. 6b.
5. The effect of unbalanced data

The three data fields analyzed in the previous section are relatively balanced; that is, they each have approximately the same number of records. Unbalanced data can play havoc with the analysis of variance (Kutner et al. 2005, p. 951), and one might expect it to cause problems with classification as well. To see the effect, first let’s run the KNN method on the full data set, basing our choice of $K$ on Fig. 6b.

```r
> nrow(data.Set2U)
[1] 2267
> data.Class <- data.Set2U@data[1:2260,]
> data.Class$CD.n <- normalize(data.Class$CoastDist)
> tr <- which(names(data.Class) == "CD.n") # Column(s) of the training data
> cl <- which(names(data.Class) == "species") # Columns of the classification variable
> X.full <- X[1:9]
> for (i in 1:length(X.full)){
+   data.Class[, (ncol(data.Class)+1)] <- normalize(data.Class[,X.full[i]])
+   tr <- c(tr, ncol(data.Class))
+ }
> cross.validate(data.Class, tr, cl, 3)
[1] 0.1844077
```

The overall RMSE is not too bad. In Exercise 3 you are asked to modify the function `cross.validate()` so that it returns the RMSE for each of the class labels. Here is are the results

```r
> print(species <- names(data.Class)[29:34])
[1] "QUAG" "QUWI" "QULO" "QUDO" "QUKE" "QUCH"
> cross.validate2(data.Class, tr, cl, 5, species)
[1] 0.1278790 0.7209301 0.8145808 0.1120078 0.1019234 0.8976459
```

The RMSE of each of the three rarest species is over 0.7. In many applications, for example in disease diagnosis, this is exactly the opposite of the desired result. The rare cases, which may for example represent instances of a disease, are precisely the ones we would like to classify most accurately. A number of methods have been proposed to address the problem of unbalanced data. Here we focus on one of the most widely used ones, called the Systematic Minority Over-sampling Technique, or SMOTE (Chawla et al., 2002). The idea behind SMOTE is the following. One easy way to reduce the error of the rare classes is to reduce the number of records of the common classes. The problem with this is that it throws away data. Another is to generate replicates of the rare data. The problem with this is that it doesn’t add any worthwhile information. The idea behind SMOTE is to reduce the size of the common data somewhat, and to generate new records from the rare data set not by simply duplicating them but rather by creating
new records using a process of interpolation of the existing data. SMOTE was designed primarily for the two-class problem.

The R package smotefamily (Siriseriwan, 2018) provides an implementation of the SMOTE method. The function SMOTE() takes as arguments a data frame containing the numerical $X$ predictor data; a vector representing the $Y$ class label data; and $K$, which has a default value of 5 (there are other arguments as well with default values). We will demonstrate the SMOTE algorithm using a data set containing only the records for $QUCH$ and $QULO$. The former has about twice as many records as the latter.

```r
> Sp2 <- data.Class[which(data.Class$species == "QUCH" | data.Class$species == "QULO"),] > nrow(Sp2)
[1] 146
> Sp2.C1 <- Sp2[1:140, c(41, tr.10)]
> table(Sp2.C1$species)
QUAG QUCH QUDO QUKE QULO QUWI
0   97    0    0   43    0
> cross.validate2(Sp2.C1, (2:ncol(Sp2.C1)), 1, 3, c("QUCH", "QULO"))
[1] 0.1332703 0.3435921
```

The RMSE for the rare class $QULO$ is about twice that of the common class $QUCH$. Here is the call to SMOTE().

```r
> library(smotefamily)
> X.SMOTE <- Sp2.C1[, (2:ncol(Sp2.C1))]
> target.SMOTE <- as.character(Sp2.C1[,1])
> Sp2.SMOTE <- SMOTE(X.SMOTE, target.SMOTE, K = 5)
```

The function SMOTE() returns quite a lot of information, but the value most important to us is `data`, which is a data frame consisting of the new data set created by the SMOTE algorithm.

```r
> Sp2.SMOTE <- SMOTE(Sp2.C1[, (2:ncol(Sp2.C1))], as.character(Sp2.C1[,1]), +    K = 5)
> class(Sp2.SMOTE$data)
[1] "data.frame"
> names(Sp2.SMOTE$data)
[1] "CD.n" "V45" "V46" "V47" "V48" "V49" "V50" "V51" [9] "V52" "V53" "class"
> nrow(Sp2.SMOTE$data)
[1] 183
> table(Sp2.SMOTE$data[1:180, "class"])
QUCH QULO
94  86
```

In this application the SMOTE algorithm has reduced the number of $QUCH$ records slightly and doubled the number of $QULO$ records. Let’s see what happens.
> cross.validate2(Sp2.SMOTE$data[1:180,], (1:(ncol(Sp2.SMOTE$data) - 1)), + ncol(Sp2.SMOTE$data), 5, c("QUCH", "QULO"))
[1] 0.21678998 0.07772269

The RMSE of the rare class is reduced dramatically at the expense in RMSE of the common class. SMOTE is not a panacea, but in some cases it can reduce the effect of imbalance in data without needlessly reducing the precision of the analysis.

6. Nearest spatial neighbors

Applying the K nearest neighbors algorithm to spatial data invites and obvious question: why not consider the nearest geographical neighbors rather than the nearest neighbors in attribute space? If we do this then obviously we give up any hope of establishing an explanatory relationship between the predictors and the class labels. If, however, it really is true that all we want to do is predict the class label at a particular location, and if the data have a high degree of positive spatial autocorrelation, then geographic location might have great predictive value. Moreover, knowing this provides some sense of the spatial distribution of the classes. In this section we will explore this supposition with two examples. The second will show that using geographical neighbors can indeed sometimes provide some ecological insight.

The first example we will consider is the use of geographical coordinates as the predictors in the example of Section 3, which involves 25 training records and 25 test records of *Quercus douglasii* presence/absence data. The spdep (Bivand et al. 2011) function knearneigh(), discussed in SDA2, Section 3.6, is available to select neighbors. However, it is instructive to keep our homemade function and modify it to take into account that the coordinates for the data set are given in longitude and latitude. We will use the predictDistance() function of the raster package (Hijmans, 2016) to compute the geographic distance between records. We will save the original version of KNN() for use below. Here is the code.

```r
> KNN.attr <- KNN
> KNN <- function(K, X, Data){
+    # K = number of neighbors, X = location to be classified
+    # Data = locations of Training data
+    D <- numeric(nrow(Data))
+    # Use pointDistance() to compute the distance between coordinates
+    for (i in 1:nrow(Data)) D[i] <- pointDistance(X, Data[i,], TRUE)
+    min.d <- sort(D)[1:K]
+    M <- numeric(K)
+    for (i in 1:K)
+       for (j in 1:length(D))
+          if (abs(min.d[i] - D[j]) < 1e-10) M[i] <- j
+    return(M)
```
With this version of `KNN()` we can repeat the example of Section 3.

```r
> X.Train <- coordinates(Set2.Train)
> Y.Train <- Set2.Train$QUDO
> # Classification with 5 classes
> K <- 5
> # Classify the training data
> Pred.Train <- classify(K, X.Train, X.Train, Y.Train)
> # Compute the training error rate
> miscl <- which((Y.Train == 1 & Pred.Train == 0) |
+     (Y.Train == 0 & Pred.Train == 1))
> print(E.Train <- length(miscl) / length(Y.Train))
[1] 0.1
> X.Test <- coordinates(Set2.Test)
> Y.Test <- Set2.Test$QUDO
> # Classify the test data
> Pred.Test <- classify(K, X.Test, X.Train, Y.Train)
> # Compute the test error rate
> miscl <- which((Y.Test == 1 & Pred.Test == 0) |
+     (Y.Test == 0 & Pred.Test == 1))
> print(E.Test <- length(miscl) / length(Y.Test))
[1] 0.18
```

The training and test errors are similar to the corresponding values in Table 1. In SDA2 Section 7.2 we noted that “McClaran (1986), citing Jepson (1910), describes blue oak woodland as being characterized by its exclusivity, with a complete lack of other tree species, with the occasional exception of foothill pine (*Pinus sabiniana*).” In other words, if you are standing near an oak tree of unknown species and its nearest neighbors are blue oaks, then the unknown species is almost certainly a blue oak.

A stronger test of geographical neighbors is that of Section 5, in which we consider all of six of the species. In that section we used the function `knn()`, which does not offer the option of using distances corrected for the sphericity of the surface of the earth. Let’s see how much difference this actually makes in the classification of the three nearest neighbors.

```r
> KNN.geog <- KNN
> P <- coordinates(data.Set2U)
> n.diff <- 0
> K <- 3
> for (i in 1:10){
+   N1 <- KNN.attr(K, P[i,], P)
+   N2 <- KNN.geog(K, P[i,], P)
+   if (max(abs(N1 - N2)) != 0) n.diff <- n.diff + 1
+ }
> n.diff
[1] 3
```
Virtually no difference at all. We will therefore simply use the Euclidean distance between longitude and latitude coordinates as our distance measure.

```r
> data.Class <- data.Set2U$data[1:2260,]
> LatLong <- coordinates(data.Set2U)[1:2260,]
> nc <- ncol(data.Class)
> tr <- c((nc+1), (nc+2))
> data.Class[, tr] <- LatLong
> cl <- which(names(data.Class) == "species")
> print(species <- names(data.Class)[29:34])
[1] "QUAG" "QUWI" "QULO" "QUDO" "QUKE" "QUCH"
> cross.validate2(data.Class, tr, cl, 5, species)
[1] 0.11003248 0.56549310 0.87557565 0.11096926 0.08230218 0.87479051
```

Comparing this result with the RMSE estimates of Section 5 shows that they are very close. There is an interesting ecological point here. While they are not improved much, the estimated RMSE of the three least common species are also not increased much by using spatial coordinates as predictors. This indicates that these relatively rare species still show a fair degree of spatial homogeneity. This point is explored further in Exercise 5.

It is also possible to combine geographic and attribute distances into a single distance of the form

\[
d_{\text{comb}}(X_1, X_2) = \alpha d_{\text{attr}}(X_1, X_2) + (1 - \alpha) d_{\text{geog}}(X_1, X_2),
\]

where \(d_{\text{attr}}\) and \(d_{\text{geog}}\) are normalized attribute and geographic distances respectively and \(\alpha\) is a tuning constant whose value is between zero and one. You are asked in Exercise 6 to explore this possibility. In practice in can provide improved performance over either distance function taken separately and in addition provides an indication of the influence of spatial autocorrelation on the data.

### 7. Further reading

For a good discussion of the two uses of data analysis, prediction and explanation, see James et al. (2013, p. 17). James et al. (2013) also provides a good introduction to the K nearest neighbor method and Hastie et al. (2009) provides a more advanced treatment. Both of these texts have a more modern, “machine learning” approach (regression, supervised classification, and unsupervised classification are sometimes considered the three primary components of machine learning). For a more traditional approach see Fukunaga (1990) and Duda et al. (2001). The paper by Chawla et al. (2002) describing the SMOTE algorithm also provides a very nice overview of classification problems in general.
7. Exercises

1. a) One measure of the performance of a classification algorithm is the confusion matrix (also called the error matrix) (cf. SDA2 Table 3.1). This can be represented for the Section 3 test data with \( K = 5 \) as shown in Table 2. Generate this table.

<table>
<thead>
<tr>
<th>Actual QULO = 0</th>
<th>Predicted QULO = 0</th>
<th>Predicted QULO = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Confusion matrix for the QULO presence/absence data.

In this context, we can define the producer’s accuracy as of each classification as the diagonal element of each classification divided by the row sum (Lo and Yeung, 2007, p. 119). It is called the producer’s accuracy because the producer has access to the actual data. Similarly, we can define the user’s accuracy of each classification as the diagonal element of each classification divided by the column sum. It is called the user’s accuracy because the user only has access to the predicted data.

b) Using half the records as the training data and the other half as the test data, generate a confusion matrix for the full six-species data set studied in Section 5.

2. Repeat the analysis of the training and test data sets of Section 3 using logistic regression and compare the results with K nearest neighbor classification.

3. (This is a pure programming exercise) Create the function `cross.validate2()` used in Section 4 that returns the RMSE for all class labels.

4. Use `cross.validate2()` to determine the RMSE for the three least common oak species using all predictors.

5. Repeat Exercise 4 using only spatial coordinates as predictors.
6. Modify the functions `classify()` and `KNN()` of Section 3 to incorporate the combined normalized attribute and geographic distances formula given in Equation (7). Show that this can lead to improved results in the classification of the data of Section 2.

8. References


Copyright © 2020 Richard E. Plant