Comparing Supervised Classification Methods for Spatial Data

Richard E. Plant
September 16, 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

Suppose we have a set of \( n \) data records, or cases, each of which includes a single attribute, denoted \( Y \), whose values we wish to subdivide into a set of classes. Each value of \( Y \) is associated with a set of \( p \) attributes, denoted \( X_i, i = 1, \ldots, p \), on which to base the classification. Typically \( Y \) will be a nominal scale (aka categorical, see SDA2 Section 4.2.1) quantity. Supervised classification uses cases for which the values of \( Y \) are known to subdivide the attribute data space into regions such that any measured set of values of the \( X_i \) can be associated with a single value of \( Y \). The set of known values of \( Y \) together with the associated attribute values \( X_i \) is called the training data. In previous Additional Topics we have described a number of supervised classification methods, including the K nearest neighbor method, support vector machines, and linear discriminant analysis. SDA2 itself contains a description of recursive partitioning, which is another such method. The objective of this Additional Topic is to discuss the comparison of methods such as these for classifying spatial data.

The objective of this Additional Topic is not to compare these methods per se. Instead, the idea is to discuss how an analyst who wishes to apply these methods effectively to a problem involving spatial data can compare them so that they can be used most effectively in the study of that particular problem. Depending on the problem and the data, some methods might be more effective than others for a particular application. The type of problem we have in mind is one in which the data set consists of a set of spatially referenced data, and that the objective is to use both the spatial and the attribute data to most effectively carry out a supervised classification.

As usual we will use the augmented Data Set 2 from SDA2 as our example data set. This data set is based on the Wieslander survey of oak species in California (Wieslander, 1935). It contains
records of the presence or absence of blue oak (*Quercus douglasii, QUDO*), coast live oak (*Quercus agrifolia, QUAG*), canyon oak (*Quercus chrysolepis, QUCH*), black oak (*Quercus kelloggii, QUKE*), valley oak (*Quercus lobata, QULO*), and interior live oak (*Quercus wislizeni, QUWI*). 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.

![Map showing the locations of most of the data records in Data Set 2U.](image)

Figure 1. Map showing the locations of most of the data records in Data Set 2U.

We can use the *mapview* package (Appelhans et al., 2017), described in the [Additional Topic on Spatial Data Exploration](#), to visualize the spatial distribution of the data (Fig. 1). It is evident
from the figure that there is a very high level of spatial autocorrelation and structure. We can quantify the level of spatial autocorrelation using join-count statistics (SDA2 Section 4.3). The spdep (Bivand and Piras, 2015) functions joincount.test() and joincount.multi() both accommodate “multicolor” data, that is, data with more than two class values. Here we will show only the results using joincount.test().

```r
> nlist.d <- dnearneigh(data.Set2U, d1 = 0, d2 = 100, longlat = TRUE)
Warning message:
In dnearneigh(data.Set2U, d1 = 0, d2 = 100, longlat = TRUE) :
dnearneigh: longlat overriden for Spatial object

The warning message provides an interesting opportunity to explore code in an R package (see Exercise 1).

```r
> W.d <- nb2listw(nlist.d, style = "W")
> joincount.test(data.Set2U$species, W.d)
Join count test under nonfree sampling
data:  data.Set2U$species
Std. deviate for QUAG = 57.72, p-value < 2.2e-16
Std. deviate for QUCH = 14.398, p-value < 2.2e-16
Std. deviate for QUOD = 56.826, p-value < 2.2e-16
Std. deviate for QUKE = 57.319, p-value < 2.2e-16
Std. deviate for QULO = 14.764, p-value < 2.2e-16
Std. deviate for QUWI = 27.842, p-value < 2.2e-16

Evidently all of the classes are highly autocorrelated. It will prove useful to carry along (mostly in the Exercises) a subset of the data with a bit more interesting structure. Exploration on the computer screen of the map shown in Fig. 1 indicated that the region between the Silicon Valley and Monterey Bay has a more complex spatial structure. Accordingly, we can create and display a “small” data set by restricting the coordinates to this region.

```r

Fig. 2 shows a screen shot of these data. Here is the application of the join count test.

```r
> nlist.d <- dnearneigh(data.Set2Small, d1 = 0, d2 = 100)
> W.d <- nb2listw(nlist.d, style = "W")
> joincount.test(data.Set2Small$species, W.d)
Join count test under nonfree sampling
data:  data.Set2Small$species
Std. deviate for QUAG = -1.4252, p-value = 0.9229
Std. deviate for QUCH = 1.1354, p-value = 0.2956
Std. deviate for QUOD = 1.9847, p-value = 0.02359
Std. deviate for QUKE = 0.53716, p-value = 0.2956
Std. deviate for QULO = 3.1355, p-value = 0.0008577
Std. deviate for QUWI = 0.86115, p-value = 0.1946
```
These data for the most part display much less spatial autocorrelation, and indeed the fact that some of the species are highly autocorrelated both corresponds to their natural history (Jepson, 1910) and makes the data more interesting.

Fig. 2. A “small” data set with a somewhat more complex spatial structure than the whole data set.

Both data sets are highly unbalanced.

```
> table(data.Set2U$species)
QUAG QUCH QUDO QUKE QULO QUWI
  551   99  731  717   47  122
> table(data.Set2Small$species)
QUAG QUCH QUDO QUKE QULO QUWI
  174   12  101   29   20   31
```

This also makes the analysis more interesting, since in classification problems it is often the rarest instances whose accurate classification is most important. In the Additional Topic on linear discriminant analysis we tried applying the SMOTE algorithm (Siriseriwan, 2018) to the data, with mixed results. Like most algorithms for dealing with unbalanced data, SMOTE works best (and was designed for) binary classification problems. Methods to deal with imbalance generally involve either oversampling, to increase the number of values of rare classes, or undersampling, to reduce the number values of common classes. Since the object is to increase the accuracy of classification of rare classes, and since the common classes are highly autocorrelated, I decided
to employ in this comparison an ad hoc undersampling method based on spatial proximity. The idea is that since the data are spatially autocorrelated, a reasonable way to undersample is to randomly select data records and then eliminate these records’ nearest neighbors, on the grounds that they would convey similar information.

Here is the code for the function `undersample()`. It uses the function `pointDistance()` of the raster package (Hijmans, 2016). I could have also used the spdep function `knearneigh()` or `dnearneigh()`, but for some reason I find `pointDistance()` easier to use.

```r
> library(raster)
> undersample <- function(Data, target, K, sp){
+   data.sp <- data.Set2U[which(data.Set2U$species == sp),]
+   n.iter <- round(nrow(data.sp) / K, 0)
+   data.reduced <- data.sp
+   while (nrow(data.reduced) > target + K){
+     dist.sp <- numeric(nrow(data.reduced) - 1)
+     X <- sample(nrow(data.reduced), 1)
+     data.noX <- data.reduced[-X,]
+     for (i in 1:(nrow(data.reduced) - 1)) dist.sp[i] <-
+       pointDistance(data.reduced[X,], data.noX[i,], TRUE)
+     min.d <- sort(dist.sp)[1:K]
+     M <- numeric(K)
+     for (i in 1:K)
+       for (j in 1:length(dist.sp))
+         if (abs(min.d[i] - dist.sp[j]) < 1e-10) M[i] <- j
+     data.reduced <- data.reduced[-M,]
+   }
+   return(data.reduced$ID)
+ }
```

The function reduces the size of the class to approximately `target` by repeatedly selecting a point at random from the set and deleting its `K` geographically nearest neighbors. It is implemented to reduce the size of each of the four largest classes to a target size of 75.

```
> set.seed(123)
> QUAG.ID <- undersample(data.Set2U, 75, 10, "QUAG")
> QUKE.ID <- undersample(data.Set2U, 75, 10, "QUKE")
> QUDO.ID <- undersample(data.Set2U, 75, 10, "QUDO")
> QUWI.ID <- undersample(data.Set2U, 75, 10, "QUWI")
> QUCH.ID <- data.Set2U[which(data.Set2U$species == "QUCH"),]$ID
> QULO.ID <- data.Set2U[which(data.Set2U$species == "QULO"),]$ID
> data.under <- data.Set2U[which((data.Set2U$ID %in% QUAG.ID) |
+   (data.Set2U$ID %in% QUKE.ID) | (data.Set2U$ID %in% QUDO.ID) |
+   (data.Set2U$ID %in% QUWI.ID) | (data.Set2U$ID %in% QUCH.ID) |
+   (data.Set2U$ID %in% QULO.ID))],]
> mapview(data.under, zcol = "species", legend = TRUE, alpha = 0,
+   col.regions = c("red", "yellow", "green", "blue", "orange", "purple"))
> table(data.under$species)
QUAG QUCH QUDO QUKE QULO QUWI
  81  99  81  77  47  82
*Quercus lobata* is still relatively small, but the other classes are about the same size. Fig. 3 shows a map of the undersampled data set. There is still quite a bit of spatial autocorrelation (Exercise 2). Notice, by the way, that in this figure, unlike Figs. 1 and 2, the colored dots do not have black rings around them. This is accomplished by including the argument `alpha = 0` in the call to `mapview()`.

Figure 3. The undersampled data set

The fact that the full data set is so highly autocorrelated begs a question: for each tree, why not just classify its species as that of its nearest geographic neighbor? This only works in theory, since in a real application we probably wouldn’t know the species of the nearest neighbor either, but it may be of value if we are considering a second sampling campaign. We will compute the error rate (also known as the misclassification rate). This quantity is discussed extensively in the Additional Topic on [the K nearest neighbor method](#), and this discussion will not be repeated here.

```r
> # Try simply classifying based on nearest neighbor
> nl <- knearneigh(data.Set2U, k = 1)
```

*Warning message:*

```r
```
In knearneigh(data.Set2U, k = 1) : knearneigh: identical points found
> pred.sp <- data.Set2U$species[n1$nn]
> misclass <- 0
> for (i in 1:nrow(data.Set2U))
+   if (data.Set2U$species[i] != pred.sp[i]) misclass <- misclass + 1
> misclass / nrow(data.Set2U)
[1] 0.1874724

I included the warning message for a reason. Upon exploration, I discovered that two sets of data records have the same coordinates (you are asked to identify these in Exercise 3). As shown in the exercise, the data are different for the different records. Whether these represent actual different records taken at approximately the same location or errors in the data will probably never be known. In any case, my students and I have been working with these data for over twenty years. I never noticed this before, and if any of my students did, they never told me. One can always learn something new about a data set. Returning to the problem at hand, the simple use of the nearest tree species as a classification method provides a theoretical baseline for comparing the other methods that we will study. This simple method does much worse when applied to the small data set (Exercise 4).

Before beginning the study of the different methods, let’s take a look at additional tools besides the simple error rate that will be useful in making comparisons (Lo and Young, 2007, p. 119). The first is the confusion matrix. This is a cross tabulation of predicted and actual values, in which the rows represent the classification values and the columns represent the true values, so that each term $c_{ij}$ represents the number of data records classified as category $i$ that are actually members of category $j$. The R function table() is used to compute this. The first argument gives the predicted values (rows) and the second gives the true values (columns). Here is the code.

> print(conf.mat <- table(pred.sp, data.Set2U$species))
pred.sp QUAG QUCH QUDO QUKE QULO QUWI
   QUAG  474   13   24    9   11   15
   QUCH    9   27   10   39    3    7
   QUDO   25    6 638   20   14   22
   QUKE    9   47   26 630    4   18
   QULO   17    1    9    3   14    1
   QUWI   17    5   24   16    1  59

The confusion matrix can be used to compute two species-specific quantities (Lo and Young, 2007, p. 121). The first, called the producer’s accuracy, is the probability of a data record being correctly classified. It is called the producer’s accuracy because it represents the accuracy of classification at the time the data are produced, using information available to the producer. For each column of the confusion matrix the producer’s accuracy is computed as the diagonal
element divided by the column sum. The producer’ accuracy is also one minus the species-specific misclassification rate. The user’s accuracy is defined as the probability that a record classified as having a particular value actually has that value. It is computed as the diagonal element divided by the row sum. It is called the user’s accuracy because it represents the quantity presumably of most interest to a user who does not have access to the true values. Here are the computations.

```r
> # Species specific accuracy
> spname <- c("QUAG", "QUCH", "QUDO", "QUKE", "QULO", "QUWI")
> prodacc.sp <- numeric(length(spname))
> useracc.sp <- numeric(length(spname))
> for (i in 1:length(sp.name)){
+   prodacc.sp[i] <- round(conf.mat[i,i] / sum(conf.mat[,i]), 2)
+   useracc.sp[i] <- round(conf.mat[i,i] / sum(conf.mat[i,]), 2)
+ }
> rbind(spname, prodacc.sp)
spname   "QUAG" "QUCH" "QUDO" "QUKE" "QULO" "QUWI"
prodacc.sp "0.86" "0.27" "0.87" "0.88" "0.3"  "0.48"
> rbind(spname, useracc.sp)
spname   "QUAG" "QUCH" "QUDO" "QUKE" "QULO" "QUWI"
useracc.sp "0.87" "0.28" "0.88" "0.86" "0.31" "0.48"
```

The classification accuracy of the rarer species is quite poor.

We are almost in a position to begin the study of the methods. Before jumping into the direct application, however, it is useful to examine the properties of the attribute data on which the classification will be based.

### 2. Properties of the attribute data.

The discussions of the methods in the Additional Topics indicated that some of them are sensitive to the scale of data (scale in the sense of magnitude, not measurement scale), while others are not. In order to establish a fair comparison, we will use an attribute data set in which all of the data fields are normalized. First we pick out the variables we will use in the classification.

```r
> D <- cbind(coordinates(data.Set2U), data.Set2U@data[,c(2:19, 28, 41)])
> names(D)
[1] "Longitude" "Latitude" "CoastDist" "MAT" "Precip" "JuMin" "JuMax" "JuMean" "JaMin" "JaMax" "JaMean" "TempR" "GS32" "GS28" "PE" "ET" "Elevation" "Texture" "AWCAvg" "Permeab" "SolRad" "species"
```
Most of the names are self-explanatory (see also SDA2 Table 7.2). The variable \( GS32 \) is the length of the growing season above 32\(^\circ\)F, and \( GS28 \) is defined similarly; \( PE \) and \( ET \) are potential and measured evapotranspiration, respectively; \( AWC_{\text{Avg}} \) is average measured soil available water content; and \( \text{Texture} \) and \( \text{Permeab} \) are ordinal variables measuring soil texture and permeability.

Notice that the spatial quantities \( \text{Longitude}, \text{Latitude}, \) and \( \text{CoastDist} \) are included in the attribute data. Sometimes an appropriate parameterization treats the same or all of the spatial variables as attributes, rather than being considered as distinct features.

Next we normalize the data.

```R
normalize <- function(x) (x - mean(x)) / sqrt(var(x))
for (i in 1:(ncol(D) - 1)) D[,i] <- normalize(D[,i])
```

```
  Longitude Latitude CoastDist  MAT  Precip  JuMin  JuMax  JuMean  JaMin  JaMax
1     1.15     -0.088    1.36  0.78 -0.66  1.07    1.0   1.2  -0.35   0.25
2    -0.25     -0.264   -0.38 1.24 -1.33  0.94    1.2   1.3   0.19   0.35
JaMean TempR GS32  GS28   PE     ET Elevation Texture AWC_{\text{Avg}} Permeab
1  -0.035    0.85 -0.17 -0.20   0.45  0.023 -0.56  -0.65   -0.84   -0.61
2   0.304    0.68 -0.25 -0.35   0.45   0.15  0.18   -0.49  -0.61  
  SolRad species
1   0.81    \text{QUDO}
2  -0.65    \text{QUDO}
```

The process can repeated with the small and undersampled data set to create the objects \( D_{\text{small}} \) and \( D_{\text{u}} \) (not shown).

For most of the methods the operation of selecting the “best” representative of that method to compare with the others will consist of two steps, a variable selection step and a tuning step. Both of these processes must be carried out in the context of the bias-variance tradeoff. This is covered in SDA2 Section 8.2, and further elaborated in the Additional Topic on support vector machines, so it will not be discussed here, except to say that it is a manifestation of the tradeoff between a model underfitting and overfitting the data. If a model underfits the data then it lacks predictive power because it does not make sufficient use of all the information contained in the training data. If a model overfits the data then it lacks predictive power because it is too closely matched to the training data set. Understanding the concept of the bias-variance tradeoff is crucial to the selection of the best method for supervised classification.

The variable selection step consists of selecting which of the (in this case) twenty-one attribute variables will be used in the model. In the context of underfitting vs. overfitting, a variable omitted from the model might contain important information, while the inclusion of too many predictive variables might include too much information specific to the training data set. In the
earlier Additional Topics we mostly selected the variables by manually comparing them. R, however, contains numerous packages that carry our variable selection and we will use the discussion of the different methods to explore some of them. Most of the methods make use of some form of stepwise selection process, in which the computer adds or subtracts variables from the model and compares the results. Stepwise variable selection is very well developed for linear regression and is discussed in that context in SDA2, Section 8.2.1. There are several alternative methods that can be used to compare the models, but one of the most common is cross-validation. This is discussed in SDA2 Section 9.3.2, and in the Additional Topic on support vector machines we created a homemade function that carries out the process, so you can see directly how it works.

Once the variable selection step has been completed, the second step in the model construction step is the tuning process. Many of the methods have one or more tuning parameters, which are quantities whose value influences the model prediction. An obvious example is the number of neighbors to include in the K nearest neighbor method. Again, R contains packages with functions to carry out the tuning process, and we will use the comparison process to discuss some of them.

The last preliminary step before embarking on the comparison is to familiarize ourselves with the relationships among the attribute data. The scatterplot matrix provides a useful tool for this. In Chapter 7 of SDA2 we used the lattice package function splom() for this, but there are a number of other alternatives. A nice one is the function pairs.panels() of the psych package (Revelle, 2017). Applying the function to all 21 attribute variables of the data set D is not useful (try it!), so first we apply it only to the some of the temperature related variables.

```r
> library(psych)
> pairs.panels(D[,c(4, 6:11)])
```

The result is shown in Fig. 4a. Some of the variables (e.g., the June temperature variables) are highly correlated, and some have a parabolic relationship. Applying the function to some of the other variables indicates no comparable level of correlation (Fig. 4b). In particular, potential and measured evapotranspiration are not highly correlated.

Having completed this last preliminary step, we can take up our comparison. We begin with the oldest of the methods, linear discriminant analysis (LDA).
3. Linear discriminant analysis

We will take up the discussion of LDA where we left off in the Additional Topic on linear discriminant analysis. That discussion, as well as those of the others in the group of topics, focused primarily on the $p = 2$ case, i.e., the case in which there are two predictors, but now we are considering the general “multicolor” case. We start the process with variable selection. In the case of LDA we will use the function `stepclass()` of the package `klaR` (Weihs et al., 2005) to carry out a stepwise selection process (SDA2, p. 261). The function `stepclass()` includes the argument `direction`, which can take on the values “forward”, “backward”, or “both”, with obvious meanings. As a default `stepclass()` uses ten-fold cross validation at each step to select the best variable to enter or leave, and we will stick with this. One of the arguments of `stepclass()` is `improvement`, which specifies the threshold for stopping the selection process.

The process of stepwise variable selection is very organic, with lots of trial and error, and I am going to give an idealized “textbook” version. When you do it with your own data, you should spend quite a bit more effort in the process, exploring various alternatives.

We will start using forward selection with an 0.5% stopping criterion. By default, in the “forward” and “both” directions `stepclass()` starts with no variables in the model.
> set.seed(123)
> stepclass(species ~ ., data = D, method = "lda",
+  direction = "both", improvement = 0.005)
``stepwise classification', using 10-fold cross-validated correctness rate of
method lda'.
2267 observations of 18 variables in 6 classes; direction: both
stop criterion: improvement less than 0.5%.
correctness rate: 0.6559; in: "JaMean"; variables (1): JaMean
correctness rate: 0.73047; in: "Precip"; variables (2): JaMean, Precip
correctness rate: 0.75472; in: "CoastDist"; variables (3): JaMean, Precip,
  CoastDist
correctness rate: 0.77103; in: "Permeab"; variables (4): JaMean, Precip,
  CoastDist, Permeab
correctness rate: 0.77766; in: "Elevation"; variables (5): JaMean, Precip,
  CoastDist, Permeab, Elevation
method : lda
final model : species ~ CoastDist + Precip + JaMean + Elevation + Permeab
correctness rate = 0.7777

Next we will try starting with this model and lowering the stopping criterion.

> set.seed(123)
> stepclass(species ~ ., data = D, method = "lda",
+  direction = "both", start.vars = c("JaMean", "Precip", "CoastDist",
+  "Elevation", "Permeab"), improvement = 0.001)
``stepwise classification', using 10-fold cross-validated correctness rate of
method lda'.
2267 observations of 21 variables in 6 classes; direction: both
stop criterion: improvement less than 0.1%.
correctness rate: 0.77766; starting variables (5): CoastDist, Precip,
  JaMean, Elevation, Permeab
correctness rate: 0.77987; in: "AWCAvg"; variables (6): CoastDist, Precip,
  JaMean, Elevation, Permeab, AWCAvg
correctness rate: 0.78295; in: "SolRad"; variables (7): CoastDist, Precip,
  JaMean, Elevation, Permeab, AWCAvg, SolRad
correctness rate: 0.78889; in: "Texture"; variables (8): CoastDist, Precip,
  JaMean, Elevation, Permeab, AWCAvg, SolRad, Texture
correctness rate: 0.79001; in: "Longitude"; variables (9): CoastDist,
  Precip, JaMean, Elevation, Permeab, AWCAvg, SolRad, Texture, Longitude
correctness rate: 0.79927; in: "JuMean"; variables (10): CoastDist, Precip,
  JaMean, Elevation, Permeab, AWCAvg, SolRad, Textur, Longitude, JuMean
correctness rate: 0.8006; in: "PE"; variables (11): CoastDist, Precip,
  JaMean, Elevation, Permeab, AWCAvg, SolRad, Texture, Longitude, JuMean, PE
correctness rate: 0.80634; in: "GS32"; variables (12): CoastDist, Precip,
  JaMean, Elevation, Permeab, AWCAvg, SolRad, Texture, Longitude, JuMean, PE,
  GS32
correctness rate = 0.8063
There were 30 warnings (use warnings() to see them)
Seven new variables come into the model, but the correctness rate only goes from 0.77766 to
0.80634 so we will stick with the five variable model. Pursuing the warnings indicates that some
of the introduced variables are collinear, which is another reason to leave them out. It is useful to
know that the model is not very sensitive to the number of included variables. In an actual
analysis it is worthwhile to try the method with a few different random number seeds to
determine the robustness of the selected model. I tried it with one other seed and got the same
results (not shown).

Having selected the model, we can compute the error rate, print the confusion matrix, and
determine the class-specific accuracy.

```r
> class.lda <- lda(species ~ JaMean + Precip + CoastDist + Elevation +
+   Permeab, data = D)
> pred.lda <- predict(class.lda)
> length(which(D$species != pred.lda$class)) / length(pred.lda$class)
[1] 0.2209969
> print(conf.mat <- table(pred.lda$class, data.Set2U$species))

QUAG  QUCH  QUDO  QUKE  QULO  QUWI
  QUAG 486  20  50  22  17  36
  QUCH  0  25   4  39   1   0
  QUDO 65  9 634  43  27  53
  QUKE  0 43 26 605  2  17
  QULO  0   0   0   0   0   0
  QUWI  0   2  17   8  16

> # Species specific accuracy
> prodacc.sp <- numeric(length(spname))
> useracc.sp <- numeric(length(spname))
> for (i in 1:length(spname)) {
+   prodacc.sp[i] <- round(conf.mat[i,i] / sum(conf.mat[,i]), 2)
+   useracc.sp[i] <- round(conf.mat[i,i] / sum(conf.mat[i,]), 2)
+ }
> rbind(spname, prodacc.sp, useracc.sp)

spname  "QUAG"  "QUCH"  "QUDO"  "QUKE"  "QULO"  "QUWI"
prodacc.sp "0.88" "0.25" "0.87" "0.84" "0" "0.13"
useracc.sp "0.77" "0.36" "0.76" "0.87" "NaN" "0.37"
```

The overall error rate is considerably worse than the simple geographic classification carried out
in Section 1, and the species-specific error rate is also poor, particularly with the rarer species. In
particular, it did not classify any data records as *Quercus lobata* (*QULO*).

To try to improve the predictive capacity for the rare species, we can train the model using the
undersampled data set and test it on the full data set. First we train the model using the
undersampled data set.

```r
> set.seed(123)
> stepclass(species ~ ., data  = D.u, method = "lda",
+   direction = "both", improvement = 0.001)

'stepwise classification', using 10-fold cross-validated correctness rate of
method lda'.
467 observations of 21 variables in 6 classes; direction: both
stop criterion: improvement less than 0.5%.
```
correctness rate: 0.39001; in: "CoastDist"; variables (1): CoastDist
correctness rate: 0.50125; in: "Elevation"; variables (2): CoastDist, Elevation
correctness rate: 0.58492; in: "Precip"; variables (3): CoastDist, Elevation, Precip
correctness rate: 0.60009; in: "Texture"; variables (4): CoastDist, Elevation, Precip, Texture
correctness rate: 0.62549; in: "Permeab"; variables (5): CoastDist, Elevation, Precip, Texture, Permeab
correctness rate: 0.6296; in: "SolRad"; variables (6): CoastDist, Elevation, Precip, Texture, Permeab, SolRad
correctness rate: 0.63603; in: "JaMean"; variables (7): CoastDist, Elevation, Precip, Texture, Permeab, SolRad, JaMean

After playing around with other random seeds, I decided to go with a six variable model. This uses the undersampled data set to train the classifier.

```r
> class.lda <- lda(species ~ CoastDist + Elevation + Precip + Texture + Permeab + SolRad, data = D.u)
```

Next we test the model on the full data set.

```r
> pred.lda <- predict(class.lda, D)
> length(which(data.Set2U$species != pred.lda$class)) / length(pred.lda$class)
[1] 0.3484782
> print(conf.mat <- table(pred.lda$class, D$species))

QUAG  QUCH  QUDO  QUKE  QULO  QUWI
QUAG  454   5  142    8   16   30
QUCH   27   69   43  145    7   17
QUDO   34    1  399    6    9    8
QUKE    1   16    9  478    0    2
QULO   26    3  516    6  13    1
QUWI    9    5   87   74    2   64

> rbind(spname, prodacc.sp)
spname  "QUAG"  "QUCH"  "QUDO"  "QUKE"  "QULO"  "QUWI"
prodacc.sp "0.82"  "0.7"  "0.55"  "0.67"  "0.28"  "0.52"
```

Comparison with the earlier results indicates that the model based on undersampled data does considerably worse in predicting the common classes, resulting in a much reduced overall accuracy, but somewhat better at predicting the members of the rarer classes. In Exercise 5 you are asked to carry out the classification of the small data set. We will leave discussion of LDA where it is for now and return to it after examining the behavior of other classification methods.
4. The $K$ nearest neighbor method

The $K$ nearest neighbor method, as it is traditionally implemented, constructs a classification based on each data record’s nearest neighbors in the $X$ attribute data space. In the Additional Topic on this method we also examined a hybrid method in which the classification is based on a combination of proximity in attribute space and geographic space, and of course the classification carried out in Section 1 is carried out in geographic space with $K = 1$. Since our main interest here is in studying how to compare methods rather than in picking the best one, we will restrict ourselves to the traditional attribute-based KNN method.

In working with LDA we used the function `stepclass()` to carry out a stepwise variable selection. In the case of the KNN method, we must carry out the two-step process of variable selection and tuning discussed earlier. The tuning parameter is the number of neighbors. The klaR package contains the function `sknn()` that does stepwise variable selection for the KNN method. The documentation indicates that is uses as a default $K = 3$ (this can be confirmed by looking at the source code, which is here), and that seems like a good start, so we will go with it for variable selection. The output shows how hard it will be to do better than the theoretical geographic model with this highly autocorrelated data set.

```r
> set.seed(123)
> class.sknn <- stepclass(species ~ ., data = D, method = "sknn",
+ direction = "both", improvement = 0.005)
'Stepwise classification', using 10-fold cross-validated correctness rate of method sknn'.
2267 observations of 19 variables in 6 classes; direction: both
stop criterion: improvement less than 0.5%.
correctness rate: 0.68547; in: "CoastDist"; variables (1): CoastDist
correctness rate: 0.81472; in: "Latitude"; variables (2): CoastDist,
Latitude
correctness rate: 0.82662; in: "Longitude"; variables (3): CoastDist,
Latitude, Longitude
```

Reducing the value of `improvement` to 0.001 makes only a slight difference in the correctness rate (not shown).

The next step is to determine the appropriate value of $K$. The package e1071 (Meyer et al., 2107) contains the function `tune()`, which works with several commonly used R functions for supervised classification. It is most easily used (in my opinion) by accessing it through a “wrapper function,” that is, a small function that organizes the arguments and passes them to the base function. Here is the code for the full data set used with the wrapper function `tune.knn()`.

```r
> set.seed(123)
```
> x <- D[,c("CoastDist", "Longitude", "Latitude")]
> y <- D[,"species"]
> tune.KNN <- tune.knn(x, y, k = 1:8)
> summary(tune.KNN)

Parameter tuning of 'knn.wrapper':
- sampling method: 10-fold cross validation
- best parameters:
  k
  6
- best performance: 0.1711551
- Detailed performance results:
  k  error dispersion
  1 1 0.1865931 0.01562581
  2 2 0.2042571 0.02690045
  3 3 0.1768956 0.02282605
  4 4 0.1777860 0.01394561
  5 5 0.1760048 0.01719648
  6 6 0.1711551 0.01611525
  7 7 0.1720342 0.02130787
  8 8 0.1786422 0.02320402

As a default, tune() uses ten-fold cross validation, but other options are available. One can also plot the error values.
> plot(tune.KNN)

Fig. 5a shows the result. The best value of $K$ is indicated to be 6.

![Performance of 'knn.wrapper'](a)

![Performance of 'knn.wrapper'](b)

Figure 5. Plots of the error rate vs. $K$ for (a) the full data set, and (b) the undersampled data set.

We can now generate the confusion matrices and the accuracy values for the two data sets. To do this we use the function knn() from the class package (Venables and Ripley, 2002). The first
argument of \texttt{knn()} gives the attribute values of the training data, the second gives the attribute values of the test data, the third gives the class values of the training data, and the fourth gives the number of neighbors. As usual, your results may differ slightly from mine.

```r
> library(class)
> set.seed(123)
> pred.KNN <- knn(x, x, y, 6)
> length(which(data.Set2U$species != pred.KNN)) / length(data.Set2U$species)
[1] 0.1349801
```

```
print(conf.mat <- table(pred.KNN, D$species))
pred.KNN QUAG QUCH QUDO QUKE QULO QUWI
   QUAG  511   10  24   10  17  19
   QUCH   2  26   0   8   0  1
   QUDO  22   5 668  15  12  18
   QUKE   7  51  27 674   4 15
   QULO   3   3   2  13   0
   QUWI   6   4  10   8   1  69
```

```
spname     "QUAG" "QUCH" "QUDO" "QUKE" "QULO" "QUWI"
prodacc.sp "0.93" "0.26" "0.91" "0.94" "0.28" "0.57"
```

```
spname     "QUAG" "QUCH" "QUDO" "QUKE" "QULO" "QUWI"
useracc.sp "0.86" "0.7" "0.9" "0.87" "0.57" "0.7"
```

The overall misclassification rate is a considerable improvement over both the simple nearest tree method and the LDA method. The producer’s accuracy of the rare species is the weakest point.

We now turn to the undersampled data set to see if that provides some improvement. Here is the selection process.

```r
> set.seed(123)
> stepclass(species ~ ., data = D.u, method = "sknn",
+   direction = "both", improvement = 0.0001)

'stepwise classification', using 10-fold cross-validated correctness rate of method sknn'.
467 observations of 21 variables in 6 classes; direction: both
stop criterion: improvement less than 0.01%.
correctness rate: 0.50791; in: "AWCAvg"; variables (1): AWCAvg
correctness rate: 0.5765; in: "Latitude"; variables (2): AWCAvg, Latitude
correctness rate: 0.63386; in: "Precip"; variables (3): AWCAvg, Latitude, Precip
correctness rate: 0.63844; in: "Elevation"; variables (4): AWCAvg, Latitude, Precip, Elevation
correctness rate: 0.65574; in: "JaMax"; variables (5): AWCAvg, Latitude, Precip, Elevation, JaMax
correctness rate: 0.67493; in: "JuMean"; variables (6): AWCAvg, Latitude, Precip, Elevation, JaMax, JuMean
correctness rate: 0.67507; in: "TempR"; variables (7): AWCAvg, Latitude, Precip, Elevation, JaMax, JuMean, TempR
Attribute values take on greater importance in the undersampled data set, which has less spatial autocorrelation. Now we apply the function `tune()` to this data set.

```r
> x.u <- D.u[,c("AWCAvg", "Latitude", "Precip", "Elevation", "JaMax", + "JuMean", "TempR")]
> y.u <- D.u[,"species"]
> set.seed(123)
> tune.KNN <- tune.knn(x.u, y.u, k = 1:8)
```

This results in Fig. 4b. The value $K = 3$ is indicated. Now we access the function `knn()`, using the undersampled data as the training set and the full data as the test set.

```r
> x <- D[,c("AWCAvg", "Latitude", "Precip", "Elevation", "JaMax", + "JuMean", "TempR")]
> pred.KNN <- knn(x.u, x, y.u, 3)
> length(which(data.Set2U$species != pred.KNN)) / length(data.Set2U$species)
[1] 0.4005293
> print(conf.mat <- table(pred.KNN, D$species))
pred.KNN QUAG QUCH QUDO QUKE QULO QUWI
QUAG 307 1 56 9 4 4
QUCH 38 75 38 237 2 18
QUDO 44 2 453 11 3 5
QUKE 2 13 10 402 3 4
QULO 81 2 37 10 33 2
QUWI 79 6 137 48 2 89
> rbind(spname, prodacc.sp)
spname     "QUAG" "QUCH" "QUDO" "QUKE" "QULO" "QUWI"
prodacc.sp "0.56" "0.76" "0.62" "0.56" "0.7"  "0.73"
> rbind(spname, useracc.sp)
spname     "QUAG" "QUCH" "QUDO" "QUKE" "QULO" "QUWI"
useracc.sp "0.81" "0.18" "0.87" "0.93" "0.2"  "0.25"
```

The performance is an improvement over that of the LDA method in the same application. As with that method, using the undersampled data set sacrifices prediction accuracy of the common species to improve that of the rare species. In Exercise 6 you are asked to carry out this analysis for the small data set. We will move on to our next candidate method, support vector machines.

### 5. Support vector machines

Support vector machines function by determining the data values (the “vectors”) that most effectively define (“support”) a set of boundaries, or kernel, that can be used to separate the attribute data space into classification regions. Different types of kernel are available, but in this section we will only discuss the radial kernel, which is often the most effective. We will consider two different packages that include a support vector machine classifier, e1071, which we
introduced in Section 4, and caret (Kuhn et al., 2018), a very general package for supervised classification.

We begin with the caret package. The name “caret” stands for “Classification And REgression Training.” This is an extremely comprehensive package, and our discussion will only scratch the surface. The interested reader can probe more deeply using the references given in Section 8. The package provides several variable selection alternatives, including backward elimination, genetic algorithms, and simulated annealing. Here we will only discuss backward elimination. Within the backward elimination context caret provides several methods for assessing the predictor combinations, including cross-validation and bootstrapping. We will only consider tenfold cross-validation. The emphasis within caret is on the use of a training and a test data set, similar to what we do in the Additional Topics, but because the cross-validation procedure implicitly carries out this subdivision, we will not do it explicitly. Again, we are only giving the tiniest scratch of the surface of all of the facilities that the caret package has to offer.

Among the many alternative classification methods included the package are several versions of support vector machines. As already stated, we will only try one with a radial kernel. This is one of a group of functions represented in caret by the term caretFuncs. The process that we have been calling backward elimination is referred to in caret as recursive feature elimination and is handled by the function rfe(). Generally when working with caret the first step is to use a control function to establish the parameters under which the function will operate. In the case of rfe() this function is rfeControl(). We will make the call with three arguments.

```r
> rfectrl <- rfeControl(functions = caretFuncs, method = "cv",
+   verbose = TRUE)
```

The argument functions = caretFuncs specifies that we are going to be using one of the methods that can be used with the caret function train() (support vector machines in our case). The argument method = "cv" specifies that model fit is going to be evaluated by cross-validation. There is an optional argument number whose default value is 10, and we won’t change that. Finally, the argument verbose specifies that the algorithm will print a line after each step. The default value is FALSE, but if you leave it set at that when you run the code then after a while you will have a very difficult time avoiding the belief that your computer has died. After the control parameters have been set the next step is the variable selection procedure. The function rfe() works with x and y representing the attribute and class variables respectively.
The argument *sizes* establishes the allowable range of sizes of the final model. Based on our experience with the other methods, we will set this as between 1 and 6.

```r
> x <- D[, -which(names(D) == "species")]
> y <- D[, which(names(D) == "species")]
> print(var.select <- rfe(x, y, sizes = 1:6, rfeControl = rfectrl))
```

+ (rfe) fit Fold01 size: 21
  *           *             *
  note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .  *           *             *

Recursive feature selection
Outer resampling method: Cross-Validated (10 fold)
Resampling performance over subset size:

<table>
<thead>
<tr>
<th>Variables</th>
<th>Accuracy</th>
<th>Kappa</th>
<th>AccuracySD</th>
<th>KappaSD</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5426</td>
<td>0.3762</td>
<td>0.08624</td>
<td>0.11316</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.7764</td>
<td>0.6896</td>
<td>0.02184</td>
<td>0.02990</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.8086</td>
<td>0.7327</td>
<td>0.01348</td>
<td>0.01851</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.8275</td>
<td>0.7594</td>
<td>0.02290</td>
<td>0.03182</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.8434</td>
<td>0.7821</td>
<td>0.01512</td>
<td>0.02121</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.8465</td>
<td>0.7868</td>
<td>0.01576</td>
<td>0.02135</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0.8500</td>
<td>0.7912</td>
<td>0.01707</td>
<td>0.02340</td>
<td></td>
</tr>
</tbody>
</table>

The top 5 variables (out of 21):
  Precip, TempR, CoastDist, Elevation, JaMax

There were 50 or more warnings (use warnings() to see the first 50)
```r
> warnings()
```

Warning messages:  
1: In randomForest.default(x, y, mtry = param$mtry, ...) :  
   invalid mtry: reset to within valid range

The method selects Precip, TempR, CoastDist, Elevation, and JaMax. The output provides a series of notes about complexity parameters and a series of warnings about the parameter *mtry*. Both of these involve parameters automatically set by the program that are reset because they turn out to be outside the permissible range. Neither warrants additional attention.

The next step is to use the functions `trainControl()` and `train()` to generate the support vector machine model and select the optimal value of the cost parameter *C* (see the Additional Topic on support vector machines for a discussion of the cost parameter).

```r
> trctrl <- trainControl(method = "cv", number = 10)
> print(svm.caret <- train(species ~ Precip + CoastDist + TempR + Elevation +  
+    Latitude, data = D, method = "svmRadial", trControl = trctrl,  
+    preProcess = c("center", "scale"), tuneLength = 10))
```

Support Vector Machines with Radial Basis Function Kernel

2267 samples  
5 predictor  
6 classes: 'QUAG', 'QUCH', 'QUDO', 'QUKE', 'QULO', 'QUWI'

Pre-processing: centered (5), scaled (5)  
Resampling: Cross-Validated (10 fold)  
Summary of sample sizes: 2040, 2042, 2041, 2041, 2039, 2039, ...
Resampling results across tuning parameters:

<table>
<thead>
<tr>
<th>C</th>
<th>Accuracy</th>
<th>Kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.8063475</td>
<td>0.7245419</td>
</tr>
<tr>
<td>0.50</td>
<td>0.8178229</td>
<td>0.7421647</td>
</tr>
<tr>
<td>1.00</td>
<td>0.8230995</td>
<td>0.7507283</td>
</tr>
<tr>
<td>2.00</td>
<td>0.8319434</td>
<td>0.7637058</td>
</tr>
<tr>
<td>4.00</td>
<td>0.8310487</td>
<td>0.7628570</td>
</tr>
<tr>
<td>8.00</td>
<td>0.8301501</td>
<td>0.7618072</td>
</tr>
<tr>
<td>16.00</td>
<td>0.8367562</td>
<td>0.7717917</td>
</tr>
<tr>
<td>32.00</td>
<td>0.8354385</td>
<td>0.7702360</td>
</tr>
<tr>
<td>64.00</td>
<td>0.8310138</td>
<td>0.7645199</td>
</tr>
<tr>
<td>128.00</td>
<td>0.8279223</td>
<td>0.7607058</td>
</tr>
</tbody>
</table>

Tuning parameter 'sigma' was held constant at a value of 0.3936526
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.3936526 and C = 16.

The final step is to run the model and determine its statistics. The caret package has a function

\texttt{confusionMatrix()} that provides these.

```
> pred.caret <- predict(svm.caret)
> length(which(D$species != pred.caret)) / length(D$species)
[1] 0.134539
> confusionMatrix(pred.caret, D$species)

Confusion Matrix and Statistics

Reference Prediction QUAG QUCH QUDO QUKE QULO QUWI
QUAG   527    9  32    8   24   29
QUCH    1   32    1    1    0    2
QUKE    21   4 656   11   10   21
QULO    0   50  28  689    3   21
QUWI    2    4  10    7    1   49

Overall Statistics

Accuracy : 0.8655
95% CI : (0.8507, 0.8792)
No Information Rate : 0.3225
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8123
McNemar's Test P-Value : NA
Statistics by Class:

<table>
<thead>
<tr>
<th>Class</th>
<th>QUAG</th>
<th>QUCH</th>
<th>QUDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.9564</td>
<td>0.32323</td>
<td>0.8974</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.9406</td>
<td>0.99769</td>
<td>0.9564</td>
</tr>
<tr>
<td>Pos Pred Value</td>
<td>0.8378</td>
<td>0.86486</td>
<td>0.9073</td>
</tr>
<tr>
<td>Neg Pred Value</td>
<td>0.9853</td>
<td>0.96996</td>
<td>0.9514</td>
</tr>
<tr>
<td>Prevalence</td>
<td>0.2431</td>
<td>0.04367</td>
<td>0.3225</td>
</tr>
<tr>
<td>Detection Rate</td>
<td>0.2325</td>
<td>0.01412</td>
<td>0.2894</td>
</tr>
<tr>
<td>Detection Prevalence</td>
<td>0.2775</td>
<td>0.01632</td>
<td>0.3189</td>
</tr>
<tr>
<td>Balanced Accuracy</td>
<td>0.9485</td>
<td>0.66046</td>
<td>0.9269</td>
</tr>
</tbody>
</table>
Class: QUKE  Class: QULO  Class: QUWI

<table>
<thead>
<tr>
<th></th>
<th>QUKE</th>
<th>QULO</th>
<th>QUWI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.9609</td>
<td>0.191489</td>
<td>0.40164</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.9342</td>
<td>0.997748</td>
<td>0.98881</td>
</tr>
<tr>
<td>Pos Pred Value</td>
<td>0.8710</td>
<td>0.642857</td>
<td>0.67123</td>
</tr>
<tr>
<td>Neg Pred Value</td>
<td>0.9810</td>
<td>0.983134</td>
<td>0.96673</td>
</tr>
<tr>
<td>Prevalence</td>
<td>0.3163</td>
<td>0.020732</td>
<td>0.05382</td>
</tr>
<tr>
<td>Detection Rate</td>
<td>0.3039</td>
<td>0.003970</td>
<td>0.03220</td>
</tr>
<tr>
<td>Detection Prevalence</td>
<td>0.3489</td>
<td>0.006176</td>
<td>0.03220</td>
</tr>
<tr>
<td>Balanced Accuracy</td>
<td>0.9476</td>
<td>0.594619</td>
<td>0.69523</td>
</tr>
</tbody>
</table>

The **Accuracy** of 0.8655 is 1 minus the error rate of 0.134539. The term **Kappa** refers to Cohen’s kappa statistic (Cohen, 1960). This measures how the classification fares in comparison to a simple random assignment of values to the cases. The output contains many other quantities, the definitions of which are given here. Comparison of formulas or values indicates that our producer’s accuracy corresponds to **Sensitivity** and our user’s accuracy corresponds to **Positive Predictive Value**. The other quantities are more useful in dichotomous, as opposed to “multicolor,” classification. Probably the best known is **Specificity**, which in dichotomous classification is commonly displayed along with sensitivity. The caret functions are applied to the small and undersampled data sets in Exercises 7 and 8.

Support vector machine classification is also available in the e1071 package. For comparison purposes we will apply it to the same model as the one selected by caret. We first use `tune()` to determine the optimal value of the **cost** parameter.

```r
> summary(tune(svm, species ~ Precip + CoastDist + TempR + Elevation +
+    Latitude, data = D, kernel = "radial",
+    ranges =list(cost=c(1, 10, 25, 50, 100, 1000))))

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
  cost 50
- best performance: 0.16805
- Detailed performance results:
  cost   error dispersion
  1     0.1918814  0.02527992
  2     0.1698082  0.02799086
  3     0.1720167  0.03181862
  4     0.1680500  0.02859629
  5     0.1689330  0.02537048
  6     0.1773128  0.02464438
```

This picks a value of 50 as the optimal cost parameter, while caret picked 16. The results, however, are not at all sensitive to this value. Now we can test the model.

```r
> class.svm <- svm(species ~ Precip + CoastDist + TempR + Elevation +
+    Latitude, data = D, kernel = "radial", cost = 50)
> pred.svm <- predict(class.svm)
```
The results are very close to those obtained using `caret`.

### 6. Recursive partitioning

Recursive partitioning (Breiman et al., 1984), more commonly known as classification and regression trees, or CART, is discussed extensively in SDA2 Section 9.3, and we will not describe it at all here. As was done in SDA2, we will apply recursive partitioning using the `rpart` package (Therneau et al., 2015). We first determine the appropriate value of `cp` using the function `plotcp()`.

```r
> library(rpart)
> class.rpart <- rpart(species ~ ., data = D, method = "class", cp = 0.001)
> plotcp(class.rpart)
```

This turns out to be `cp = 0.008`. Next we develop the classification tree, shown in Fig. 5.

```r
> cont.parms <- rpart.control(minsplit = 5, cp = 0.008)
> class.rpart <- rpart(species ~ ., data = D, method = "class",
+    control = cont.parms)
> plot(class.rpart)
> text(class.rpart)
```

Finally, as usual we use the `predict()` function to generate the classification statistics.

```r
> pred.rpart <- predict(class.rpart, type = "class")
> length(which(D$species != pred.rpart)) / length(D$species)
[1] 0.2033524
> print(conf.mat <- table(pred.rpart, D$species))
```
The method does not do very well, particularly with the rare species.

Figure 5. Classification tree for the five species. It does not classify any records as *QUCH* or *QULO*.

7. Discussion

Based only on the simple metric of the misclassification rate, we might be tempted to declare support vector machines as the “winner,” and indeed SVM is a very powerful method. Before closing the book, however, we have to think about a few things. One is the overall intended use of the delivered product. At several points in the discussion of the various methods we have stated that the context of this discussion is that the intended use is purely predictive. In other
words, the intent is to build a tool that one could use to enter the requisite data and predict as “best” as possible which of the five species of oak corresponds to these data. Often in ecological and agricultural research, however, there is some interest, perhaps not explicitly recognized, in explanation, that is, in generating an increased understanding of the ecological processes underlying the observations. If this is the case, then a classification tree such as that generated in Figure 5, together with results of a random forest study (this is discussed in SDA2 Section 9.4) may have value.

Suppose, however, that we really are only interested in prediction. That implies that there are some locations, where we may or may not have already collected data, where we would like to make a prediction, in this case of the oak species at that location. Suppose first that we already have all of the attribute data values $X$ that we will need, and we just want to use these data to predict the unknown class values $Y$. A relevant question then is whether we are more interested in an accurate prediction of some classes than of others. Let’s suppose that we are most interested in accurate classification of the rarest classes. We have seen two class specific measures of accuracy: the producer’s accuracy (which is also the species-specific misclassification rate) and the user’s accuracy. For ease of discussion the output of the SVM analysis of the full data set is reproduced here.

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

> length(which(D$species != pred.svm)) / length(pred.svm)
[1] 0.1371857
> print(conf.mat <- table(pred.svm, D$species))
pred.svm       QUAG QUCH QUDO QUKE QULO QUWI
QUAG    522    9  33    9  25  28
QUCH     1  31    0    1    0    3
QUDO    27    4 659  13  11  21
QUKE     0  51  28 688   2  22
QULO     0    0    3    0    8    0
QUWI     1    4    8    6    1  48

> rbind(spname, prodacc.sp)
spname   "QUAG"  "QUCH"  "QUDO"  "QUKE"  "QULO"  "QUWI"
prodacc.sp "0.95" "0.31" "0.9"  "0.96" "0.17" "0.39"

> rbind(spname, useracc.sp)
spname   "QUAG"  "QUCH"  "QUDO"  "QUKE"  "QULO"  "QUWI"
useracc.sp "0.83" "0.86" "0.9"  "0.87" "0.73" "0.71"
```

For the common species the producer’s and use’s accuracies tend to be similar, while for the rare species these quantities tend to be inverted. Let’s consider the case of QULO Recall that the
producer’s accuracy is the diagonal element divided by the column sum. Thus it is the fraction of all QULO trees, for example, that are classified as QULO. The user’s accuracy is the fraction of all trees classified as QULO that actually are QULO. Which of these provides the better measure depends on the consequences of an error in classification. If we want to get as many QULO (or some other specific species) right as possible and we don’t care how many we get wrong, then we would like to maximize the producer’s accuracy. If, on the other hand, we want to have the highest faction of trees classified as QULO actually be members of that species, then we are more interested in the user’s accuracy. For specificity, let’s suppose that the latter is the case.

Here is a reproduction of the relevant line of output for each method.

<table>
<thead>
<tr>
<th>spname</th>
<th>QUAG</th>
<th>QUCH</th>
<th>QUOD</th>
<th>QUKE</th>
<th>QULO</th>
<th>QUWI</th>
</tr>
</thead>
<tbody>
<tr>
<td>useracc.sp</td>
<td>0.87</td>
<td>0.28</td>
<td>0.88</td>
<td>0.86</td>
<td>0.31</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>0.77</td>
<td>0.36</td>
<td>0.76</td>
<td>0.87</td>
<td>NaN</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>0.86</td>
<td>0.7</td>
<td>0.9</td>
<td>0.87</td>
<td>0.57</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>0.83</td>
<td>0.86</td>
<td>0.9</td>
<td>0.87</td>
<td>0.73</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>NaN</td>
<td>0.82</td>
<td>0.82</td>
<td>NaN</td>
<td>0.64</td>
</tr>
</tbody>
</table>

#Simple nearest tree

For four of the five species SVM provides the greatest user’s accuracy.

Did our simple undersampling method had any impact on the ability to accurately classify the rare cases? The table is taken from the material in the sections as well as the results of Exercise 7.

<table>
<thead>
<tr>
<th>spname</th>
<th>QUAG</th>
<th>QUCH</th>
<th>QUOD</th>
<th>QUKE</th>
<th>QULO</th>
<th>QUWI</th>
</tr>
</thead>
<tbody>
<tr>
<td>useracc.sp</td>
<td>0.69</td>
<td>0.22</td>
<td>0.87</td>
<td>0.94</td>
<td>0.13</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>0.81</td>
<td>0.18</td>
<td>0.87</td>
<td>0.93</td>
<td>0.2</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>0.86</td>
<td>0.91</td>
<td>0.93</td>
<td>0.87</td>
<td>0.76</td>
<td>0.76</td>
</tr>
</tbody>
</table>

#LDA

#KNN

#SVM

The KNN method does not fare particularly well, but the other two methods are improved somewhat. Interestingly, for this data set the user’s accuracy of the common classes seems to be improved as well as that of the rare classes.

How important a role does spatial autocorrelation play? To address this we can look at the results of testing the methods on the small data set, carried out in Exercises 4, 5, 6.

<table>
<thead>
<tr>
<th>spname</th>
<th>QUAG</th>
<th>QUCH</th>
<th>QUOD</th>
<th>QUKE</th>
<th>QULO</th>
<th>QUWI</th>
</tr>
</thead>
<tbody>
<tr>
<td>useracc.sp</td>
<td>0.79</td>
<td>0.25</td>
<td>0.85</td>
<td>0.38</td>
<td>0.35</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>0.71</td>
<td>NaN</td>
<td>0.8</td>
<td>0.5</td>
<td>0.44</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>0.5</td>
<td>0.86</td>
<td>0.6</td>
<td>0.53</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>1.0</td>
<td>0.92</td>
<td>0.74</td>
<td>0.87</td>
<td>0.60</td>
</tr>
</tbody>
</table>

#Simple nearest tree

#LDA

#KNN

#SVM

Again SVM generally seems to give the best results for this particular data set. A perhaps more interesting observation is that the dramatic reduction in spatial autocorrelation does not have much effect on the results. This may to some extent reflect the fact that a large part of the effect
of spatial autocorrelation on other statistical operations, such as hypothesis testing, is due to the fact that each observation provides some information about other geographically nearby observations, when the assumptions associated with hypothesis testing require this not to be true (see SDA2, Ch. 13). In the case of supervised classification, no such independence assumptions are made.

So far we have assumed that all of the relevant attribute data have already been gathered. Suppose, on the other hand, that we know the locations where we want to make the predictions but we have to gather the relevant data in order to do so. If there is some level of cost or inconvenience associated with this data collection, this might influence our choice of method.

We can assume that the geographic location of each new case is known, so that the simple geographic method would not be affected, but we can consider the effect of reducing the number of attribute values $X$, that we can add to the model. The LDA method prediction was based on $JaMean$, $Precip$, $CoastDist$, $Elevation$, and $Permeab$; the KNN method prediction was based on $CoastDist$, $Longitude$, and $Latitude$; the SVM method prediction was based on $Precip$, $CoastDist$, $TempR$, $Elevation$, and $Latitude$; and the CART prediction was based on $Precip$, $CoastDist$, $TempR$, $Elevation$, and $Longitude$. Suppose that some of these attribute values are difficult to obtain for new cases, and that the prediction must be made based on a reduced data set. The ability of a method to maintain good performance under this condition is sometimes called graceful degradation. One cannot make a fair hypothetical comparison of the methods without knowing which specific attributes are expensive to measure, since each method depends on different attributes. Since these are spatial data, so that geographic information will be available for new cases, and since $CoastDist$ is present in all of the attribute sets and $Latitude$ is present in two of them, we will test for graceful degradation using these attributes.

<table>
<thead>
<tr>
<th>spname</th>
<th>&quot;QUAG&quot;</th>
<th>&quot;QUCH&quot;</th>
<th>&quot;QUDO&quot;</th>
<th>&quot;QUKE&quot;</th>
<th>&quot;QULO&quot;</th>
<th>&quot;QUWI&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>useracc.sp</td>
<td>0.64</td>
<td>NaN</td>
<td>0.69</td>
<td>0.62</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>useracc.sp</td>
<td>0.84</td>
<td>0.72</td>
<td>0.9</td>
<td>0.87</td>
<td>0.52</td>
<td>0.66</td>
</tr>
<tr>
<td>useracc.sp</td>
<td>0.77</td>
<td>0.83</td>
<td>0.8</td>
<td>0.86</td>
<td>NaN</td>
<td>0.52</td>
</tr>
<tr>
<td>useracc.sp</td>
<td>0.79</td>
<td>0.67</td>
<td>0.81</td>
<td>0.84</td>
<td>NaN</td>
<td>0.63</td>
</tr>
</tbody>
</table>

Perhaps it is not terribly surprising that the KNN displays for most species the most graceful degradation for this spatial data set.
8. Further reading

More information on multicolor join-count statistics can be found in Cliff and Ord (1981, p. 19) and in Upton and Fingleton (1985, p. 164). As always, two excellent sources for a discussion of the algorithms are James et al. (2013) at an introductory level and Hastie et al. (2009) at a more advanced level. Duda et al. (2001) approach the subject from what might be called a more computational perspective, and Lattin et al. (2003) provide what might be called a more classical perspective. Both are useful sources. The caret package is described by Kuhn (2019) and by Kuhn and Johnson (2013). This last book is an excellent resource for applied work, and should be very high on the reading list of any practitioner working in this area.

9. Exercises

1) Use the fact that you can view the code of a contributed package function by typing the name of the function without the parentheses to explore the code of the function `dnearneigh()` and see why the warning message is generated when the code is used to generate `nlist.d`. In the Additional Topic on the K nearest neighbor method we saw that at the spatial scale of our data there is no difference in results whether or not a correction is made for the shape of the earth.

2) Compute the join count statistics for the undersampled data set `data.under`.

3) Identify the sets of points in the data set `data.Set2U` with identical coordinates and determine their properties.

4) Classify species in the small data set based on the tree’s nearest neighbor.

5) Use the LDA method to obtain a classification of the small data set.

6) Use the KNN method to obtain a classification of the small data set.

7) Use the SVM method as implemented in caret to obtain a classification of the undersampled data set.
8) Use the SVM method as implemented in caret to obtain a classification of the small data set.

**References**


Copyright © 2020 Richard E. Plant