A Tempered Approach to Interpreting the Results of Clustering

We have described some of the issues associated with clustering. However, clustering can be a very useful and valid statistical tool if used properly. We mentioned that small decisions in how clustering is performed, such as how the data are standardized and what type of linkage is used, can have a large effect on the results. Therefore, we recommend performing clustering with different choices of these parameters, and looking at the full set of results in order to see what patterns consistently emerge. Since clustering can be non-robust, we recommend clustering subsets of the data in order to get a sense of the robustness of the clusters obtained. Most importantly, we must be careful about how the results of a clustering analysis are reported. These results should not be taken as the absolute truth about a data set. Rather, they should constitute a starting point for the development of a scientific hypothesis and further study, preferably on an independent data set.

10.4 Lab 1: Principal Components Analysis

In this lab, we perform PCA on the `USArrests` data set, which is part of the base `R` package. The rows of the data set contain the 50 states, in alphabetical order.

```
> states=row.names(USArrests)
> states
```

The columns of the data set contain the four variables.

```
> names(USArrests)
[1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

We first briefly examine the data. We notice that the variables have vastly different means.

```
> apply(USArrests, 2, mean)
  Murder   Assault  UrbanPop      Rape
    7.79   170.76     65.54     21.23
```

Note that the `apply()` function allows us to apply a function—in this case, the `mean()` function—to each row or column of the data set. The second input here denotes whether we wish to compute the mean of the rows, 1, or the columns, 2. We see that there are on average three times as many rapes as murders, and more than eight times as many assaults as rapes. We can also examine the variances of the four variables using the `apply()` function.

```
> apply(USArrests, 2, var)
  Murder   Assault  UrbanPop      Rape
    19.0    6945.2     209.5      87.7
```

Not surprisingly, the variables also have vastly different variances: the UrbanPop variable measures the percentage of the population in each state living in an urban area, which is not a comparable number to the number of rapes in each state per 100,000 individuals. If we failed to scale the variables before performing PCA, then most of the principal components that we observed would be driven by the Assault variable, since it has by far the largest mean and variance. Thus, it is important to standardize the variables to have mean zero and standard deviation one before performing PCA.

We now perform principal components analysis using the prcomp() function, which is one of several functions in R that perform PCA. prcomp()

```
> pr.out=prcomp(USArrests, scale=TRUE)
```

By default, the prcomp() function centers the variables to have mean zero. By using the option scale=TRUE, we scale the variables to have standard deviation one. The output from prcomp() contains a number of useful quantities.

```
> names(pr.out)
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

The center and scale components correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```
> pr.out$center
  Murder   Assault  UrbanPop     Rape
    7.79    170.76     65.54    21.23
> pr.out$scale
  Murder   Assault  UrbanPop     Rape
    4.36     83.34     14.47     9.37
```

The rotation matrix provides the principal component loadings; each column of pr.out$rotation contains the corresponding principal component loading vector.[2]

```
> pr.out$rotation
             PC1     PC2     PC3     PC4
Murder   -0.536   0.418 -0.341   0.649
Assault  -0.583   0.188 -0.268 -0.743
UrbanPop -0.278 -0.873 -0.378   0.134
Rape     -0.543 -0.167   0.818   0.089
```

We see that there are four distinct principal components. This is to be expected because there are in general $\min(n-1, p)$ informative principal components in a data set with n observations and p variables.

[2]This function names it the rotation matrix, because when we matrix-multiply the $\mathbf{X}$ matrix by pr.out$rotation, it gives us the coordinates of the data in the rotated coordinate system. These coordinates are the principal component scores.

Using the `prcomp()` function, we do not need to explicitly multiply the data by the principal component loading vectors in order to obtain the principal component score vectors. Rather the 50×4 matrix `x` has as its columns the principal component score vectors. That is, the kth column is the kth principal component score vector.

```
> dim(pr.out$x)
[1] 50  4
```

We can plot the first two principal components as follows:

```
> biplot(pr.out, scale=0)
```

The `scale=0` argument to `biplot()` ensures that the arrows are scaled to represent the loadings; other values for `scale` give slightly different biplots with different interpretations. `biplot()`

Notice that this figure is a mirror image of Figure 10.1. Recall that the principal components are only unique up to a sign change, so we can reproduce Figure 10.1 by making a few small changes:

```
> pr.out$rotation=-pr.out$rotation
> pr.out$x=-pr.out$x
> biplot(pr.out, scale=0)
```

The `prcomp()` function also outputs the standard deviation of each principal component. For instance, on the `USArrests` data set, we can access these standard deviations as follows:

```
> pr.out$sdev
[1] 1.575 0.995 0.597 0.416
```

The variance explained by each principal component is obtained by squaring these:

```
> pr.var=pr.out$sdev^2
> pr.var
[1] 2.480 0.990 0.357 0.173
```

To compute the proportion of variance explained by each principal component, we simply divide the variance explained by each principal component by the total variance explained by all four principal components:

```
> pve=pr.var/sum(pr.var)
> pve
[1] 0.6201 0.2474 0.0891 0.0434
```

We see that the first principal component explains 62.0 % of the variance in the data, the next principal component explains 24.7 % of the variance, and so forth. We can plot the PVE explained by each component, as well as the cumulative PVE, as follows:

```
> plot(pve, xlab="Principal Component", ylab="Proportion of
    Variance Explained", ylim=c(0,1),type='b')
> plot(cumsum(pve), xlab="Principal Component", ylab="
    Cumulative Proportion of Variance Explained", ylim=c(0,1),
    type='b')
```

The result is shown in Figure 10.4. Note that the function `cumsum()` computes the cumulative sum of the elements of a numeric vector. For instance: `cumsum()`

```
> a=c(1,2,8,-3)
> cumsum(a)
[1]  1  3 11  8
```

10.5 Lab 2: Clustering

10.5.1 K-Means Clustering

The function `kmeans()` performs K-means clustering in `R`. We begin with a simple simulated example in which there truly are two clusters in the data: the first 25 observations have a mean shift relative to the next 25 observations. `kmeans()`

```
> set.seed(2)
> x=matrix(rnorm(50*2), ncol=2)
> x[1:25,1]=x[1:25,1]+3
> x[1:25,2]=x[1:25,2]-4
```

We now perform K-means clustering with $K = 2$.

```
> km.out=kmeans(x,2,nstart=20)
```

The cluster assignments of the 50 observations are contained in `km.out$cluster`.

```
> km.out$cluster
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1
[30] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The K-means clustering perfectly separated the observations into two clusters even though we did not supply any group information to `kmeans()`. We can plot the data, with each observation colored according to its cluster assignment.

```
> plot(x, col=(km.out$cluster+1), main="K-Means Clustering
    Results with K=2", xlab="", ylab="", pch=20, cex=2)
```

Here the observations can be easily plotted because they are two-dimensional. If there were more than two variables then we could instead perform PCA and plot the first two principal components score vectors.

In this example, we knew that there really were two clusters because we generated the data. However, for real data, in general we do not know the true number of clusters. We could instead have performed K-means clustering on this example with $K = 3$.

```
> set.seed(4)
> km.out=kmeans(x,3,nstart=20)
> km.out
K-means clustering with 3 clusters of sizes 10, 23, 17
```

```
Cluster means:
        [,1]        [,2]
1  2.3001545 -2.69622023
2 -0.3820397 -0.08740753
3  3.7789567 -4.56200798

Clustering vector:
 [1] 3 1 3 1 3 3 3 1 3 1 3 1 3 1 3 1 3 3 3 3 3 1 3 3 3 2 2 2 2
     2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2

Within cluster sum of squares by cluster:
[1] 19.56137 52.67700 25.74089
 (between_SS / total_SS =  79.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
    "tot.withinss" "betweenss"    "size"
> plot(x, col=(km.out$cluster+1), main="K-Means Clustering
    Results with K=3", xlab="", ylab="", pch=20, cex=2)
```

When $K = 3$, K-means clustering splits up the two clusters.

To run the `kmeans()` function in `R` with multiple initial cluster assignments, we use the `nstart` argument. If a value of `nstart` greater than one is used, then K-means clustering will be performed using multiple random assignments in Step 1 of Algorithm 10.1, and the `kmeans()` function will report only the best results. Here we compare using `nstart=1` to `nstart=20`.

```
> set.seed(3)
> km.out=kmeans(x,3,nstart=1)
> km.out$tot.withinss
[1] 104.3319
> km.out=kmeans(x,3,nstart=20)
> km.out$tot.withinss
[1] 97.9793
```

Note that `km.out$tot.withinss` is the total within-cluster sum of squares, which we seek to minimize by performing K-means clustering (Equation 10.11). The individual within-cluster sum-of-squares are contained in the vector `km.out$withinss`.

We *strongly* recommend always running K-means clustering with a large value of `nstart`, such as 20 or 50, since otherwise an undesirable local optimum may be obtained.

When performing K-means clustering, in addition to using multiple initial cluster assignments, it is also important to set a random seed using the `set.seed()` function. This way, the initial cluster assignments in Step 1 can be replicated, and the K-means output will be fully reproducible.

10.5.2 Hierarchical Clustering

The `hclust()` function implements hierarchical clustering in `R`. In the following example we use the data from Section 10.5.1 to plot the hierarchical clustering dendrogram using complete, single, and average linkage clustering, with Euclidean distance as the dissimilarity measure. We begin by clustering observations using complete linkage. The `dist()` function is used to compute the 50×50 inter-observation Euclidean distance matrix.

`hclust()`

`dist()`

```
> hc.complete=hclust(dist(x), method="complete")
```

We could just as easily perform hierarchical clustering with average or single linkage instead:

```
> hc.average=hclust(dist(x), method="average")
> hc.single=hclust(dist(x), method="single")
```

We can now plot the dendrograms obtained using the usual `plot()` function. The numbers at the bottom of the plot identify each observation.

```
> par(mfrow=c(1,3))
> plot(hc.complete,main="Complete Linkage", xlab="", sub="",
    cex=.9)
> plot(hc.average, main="Average Linkage", xlab="", sub="",
    cex=.9)
> plot(hc.single, main="Single Linkage", xlab="", sub="",
    cex=.9)
```

To determine the cluster labels for each observation associated with a given cut of the dendrogram, we can use the `cutree()` function:

`cutree()`

```
> cutree(hc.complete, 2)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
[30] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
> cutree(hc.average, 2)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
[30] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2
> cutree(hc.single, 2)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
[30] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

For this data, complete and average linkage generally separate the observations into their correct groups. However, single linkage identifies one point as belonging to its own cluster. A more sensible answer is obtained when four clusters are selected, although there are still two singletons.

```
> cutree(hc.single, 4)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 3 3 3 3
[30] 3 3 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3
```

To scale the variables before performing hierarchical clustering of the observations, we use the `scale()` function:

`scale()`

```
> xsc=scale(x)
> plot(hclust(dist(xsc), method="complete"), main="Hierarchical
    Clustering with Scaled Features")
```

Correlation-based distance can be computed using the `as.dist()` function, which converts an arbitrary square symmetric matrix into a form that the `hclust()` function recognizes as a distance matrix. However, this only makes sense for data with at least three features since the absolute correlation between any two observations with measurements on two features is always 1. Hence, we will cluster a three-dimensional data set.

as.dist()

```
> x=matrix(rnorm(30*3), ncol=3)
> dd=as.dist(1-cor(t(x)))
> plot(hclust(dd, method="complete"), main="Complete Linkage
    with Correlation-Based Distance", xlab="", sub="")
```

10.6 Lab 3: NCI60 Data Example

Unsupervised techniques are often used in the analysis of genomic data. In particular, PCA and hierarchical clustering are popular tools. We illustrate these techniques on the `NCI60` cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines.

```
> library(ISLR)
> nci.labs=NCI60$labs
> nci.data=NCI60$data
```

Each cell line is labeled with a cancer type. We do not make use of the cancer types in performing PCA and clustering, as these are unsupervised techniques. But after performing PCA and clustering, we will check to see the extent to which these cancer types agree with the results of these unsupervised techniques.

The data has 64 rows and 6,830 columns.

```
> dim(nci.data)
[1]   64 6830
```

We begin by examining the cancer types for the cell lines.

```
> nci.labs[1:4]
[1] "CNS"   "CNS"   "CNS"   "RENAL"
> table(nci.labs)
nci.labs
     BREAST          CNS        COLON K562A-repro K562B-repro
          7            5            7            1            1
   LEUKEMIA MCF7A-repro MCF7D-repro     MELANOMA        NSCLC
          6            1            1            8            9
    OVARIAN     PROSTATE        RENAL      UNKNOWN
          6            2            9            1
```