STScI Astrostatistics
R tutorials
Eric Feigelson (Penn State)
November 2011

SESSION 2
Multivariate clustering and classification

****
****

Unsupervised clustering of COMBO-17 galaxy photometry

We illustrate unsupervised clustering algorithms using a two-dimensional color-magnitude diagram constructed from the COMBO-17 (`Classifying Objects by Medium-Band Observations in 17 Filters') photometric survey of normal galaxies (Wolf et al. 2003). The R script below starts with the which function to filter the dataset, keeping only low-redshift galaxies with z < 0.3 and remove a few points with bad data values. Most of the original 65 variables are ignored, and we keep only the galaxy absolute magnitude in the blue band, M_B, and the ultraviolet-to-blue color index, M_{280 - M_B. The resulting color-magnitude diagram (left panel) shows the well-known concentrations of luminous red galaxies around (M_B,M_{280-M_B) \simeq (-16,-0.2) and fainter blue galaxies around (-13,-0.9).

The clusters are more clearly seen after smoothing the point process with a two-dimensional kernel density estimator using kde2d in R's MASS library. The blue galaxies are spirals and irregular galaxies that have experienced recent active star formation, while the red galaxies are mostly ellipticals that have only older stars formed early in the Universe's history. Note that many galaxies have properties distributed around the major concentrations. For example, a few extremely luminous and red galaxies are seen around (-20,1.5); these are nearby examples of the `luminous red galaxies' that are very important in cosmological studies (Eisenstein et al. 2005).

While the kernel density estimator provides a valuable visualization of the clustering pattern, it does not assign individual galaxies to specific clusters. We illustrate unsupervised clustering of the

dataset using three methods in the following R script. For the nonparametric procedures where we assume a Euclidean distances between points in the 2-space, we first standardize the variables by removing the means and dividing by the standard deviations.

```
# Unsupervised clustering of red and blue galaxies
# Color-magnitude diagram for low-redshift COMBO-17 galaxies

COMBO=read.table('http://astrostatistics.psu.edu/MSMA/datasets/
COMBO17_lowz.dat',header=T,fill=T)
dim(COMBO) ; names(COMBO)
par(mfrow=c(1,2))
plot(COMBO,pch=20,cex=0.5,xlim=c(-22,-7), ylim=c(-2,2.5),xlab='M_B
(mag)',ylab='M_280 - M_B (mag)',main='COMBO-17 galaxies (z<0.3)')

# Two-dimensional kernel-density estimator

library(MASS)
COMBO_sm=kde2d(COMBO[,1],COMBO[,2], h=c(1.6,0.4),lims = c
(-22,-7,-2,2.5), n=500)

image(COMBO_sm,col=grey(13:0/15),xlab='M_B (mag)',ylab='M_280 - M_B
(mag)',,xlim=c(-22,-7), ylim=c(-2,2.5),xaxp=c(-20,-10,2))
text(-18,-1.2,'Blue cloud', col='blue')
text(-18,-0.8,'Green valley', col='darkgreen')
text(-10,-0.2,'Red sequence', col='red')
text(-13,1.5,'Bright cluster galaxies', col='darkred')
par(mfrow=c(1,1))
```

R and CRAN have a variety of agglomerative hierarchical clustering algorithms. We start with the most commonly used procedure, function hclust in base-R. The procedure runs on the matrix of pairwise distances between points constructed using the function dist. As the structures in the smoothed distribution seem roughly spherical, we choose the `complete linkage' definition of group locations. The product of hclust is a dendrogram which can be displayed using plclust.

```
# Standardize variables
```

```
Mag_std=(COMBO[,1]-mean(COMBO[,1]))/sd(COMBO[,1])
Color_std=(COMBO[,2]-mean(COMBO[,2]))/sd(COMBO[,2])
COMBO_std=cbind(Mag_std,Color_std)
plot(COMBO_std)

# Hierarchical clustering

COMBO_dist=dist(COMBO_std)
COMBO_hc=hclust(COMBO_dist,method='complete')
 # Cutting the tree at k=5 clusters
plclust(COMBO_hc,label=F)
COMBO_hc5a=rect.hclust(COMBO_hc,k=5,border='black') ; str
(COMBO_hc5a)
COMBO_hc5b=cutree(COMBO_hc,k=5) ; str(COMBO_hc5b)
plot(COMBO,pch=(COMBO_hc5b+19),cex=1.0,xlab='M_B (mag)',ylab='M_280
- M_B (mag)',main='COMBO-17 hier clustering
(k=5)',cex.lab=1.3,cex.axis=1.3)
```

There is no formal procedure to select branches of the dendrogram as physically valid clusters. The cophenetic correlation coefficient, a measure of the similarity of the hierarchical structure and the data, is 0.52 using functions cophenetic and cor, but this can not readily be converted to a probability. We investigated the tree by trial-and-error, and found that `cutting the tree' at $k=5$ clusters provides a useful result. Two procedures are shown here: {rect.hclust that shows rectangles in the dendrogram (top panel), and cutree which gives an output with individual galaxy memberships of the five clusters. These are shown as different symbols in the color-magnitude diagram of the bottom panel; the open triangles show the red galaxies and the open circles show the blue galaxies. These clusters include many outlying galaxies, and examination of smaller clusters in the hierarchy does not cleanly discriminate the cluster extents seen in the smoothed distribution seen in the earlier figure (left panel).

Our second clustering method attempts to alleviate this problem with the hierarchical clustering results by using the density as a starting point for the clustering algorithm. We use the DBSCAN (density-based cluster analysis) in CRAN package fpc (`fixed point clusters') which implements the procedure of Ester et al. (1996). DBSCAN is widely used, particularly for problems where compact clusters of interest are embedded in multiscale structure. The

{dbscan function requires user input of two parameters: the minimum number of points within a radius (or `reach') associated with the clusters of interest. By trial-and-error, we found that a minimum of 10 points within 0.3 standardized magnitude units provided a useful result as shown in the figure. Here only the fraction of galaxies lying within the regions satisfying this local density criterion are classified; red and blue galaxy groups are clearly discriminated, and intermediate galaxies are not classified.

```
# Density-based clustering algorithm

install.packages('fpc') ; library(fpc)
COMBO_dbs = dbscan(COMBO_std,eps=0.1,MinPts=5,method='raw')
print.dbscan(COMBO_dbs) ; COMBO_dbs$cluster
plot(COMBO[COMBO_dbs$cluster==0,], pch=20,cex=0.7,xlab='M_B
(mag)',ylab='M_280 - M_B (mag)',main='COMBO-17 density wt
clustering',cex.lab=1.3,cex.axis=1.3)
points(COMBO[COMBO_dbs$cluster==2,],pch=2,cex=1.0)
points(COMBO[COMBO_dbs$cluster==1 | COMBO_dbs
$cluster==3,],pch=1,cex=1.0)
```

Our third clustering method is the well-respected parametric mclust (`model-based clustering') package in CRAN that fits a multivariate normal (MVN) mixture model by maximum-likelihood estimation using the EM Algorithm with Bayesian regularization (Fraley & Raftery 2002, 2007). We run an unsupervised procedure, but the calculation can be initialized with the output of MVN hierarchical clustering and the user can specify conjugate priors for the means and variances. In function mclustBIC, the `VVV' model name specifies multivariate ellipsoidal Gaussians with arbitrary orientations. Model selection is performed by maximizing the Bayesian Information Criterion (BIC) for different number of clusters.

The model-based clustering algorithm is shown in the figure. The likelihood for the COMBO-17 color-magnitude diagram is maximized for three clusters, two of which distinguish the red and blue galaxy sequences. Detailed results are provided by the summary.mclustBIC function including the probabilities of cluster membership for each galaxy and the uncertainties to these probabilities.

Points lying between two clusters can be investigated using a

visualization tool known as `shadow' and `silhouette' plots coupled to centroid-based partitioning cluster analysis (Leisch 2009). Each data point has a shadow value equal to two-times the distance to the closest cluster centroid divided by the sum of distances to closest and second-closest centroids. Points with shadow values near unity lie equidistant from the two clusters. Silhouette values measure the difference between the average dissimilarity of a point to all points in its own cluster to the smallest average dissimilarity to the points of a different cluster. Small values again indicate points with ill-defined cluster memberships. These plots can be constructed using CRAN's flexclust package.

```
# Model-based clustering

library(mclust)
COMBO_mclus=mclustBIC(COMBO,modelNames='VVV')
plot(COMBO_mclus,col='black')
COMBO_sum_mclus=summary.mclustBIC(COMBO_mclus,COMBO,3)
COMBO_sum_mclus$parameters ; COMBO_sum_mclus$classification
COMBO_sum_mclus$z ; COMBO_sum_mclus$uncertainty
plot(COMBO,pch=(19+COMBO_sum_mclus$classification),cex=1.0,xlab='M_B
(mag)',ylab='M_280 - M_B (mag)',main='COMBO-17 MVN model clustering
(k=3)',cex.lab=1.3,cex.axis=1.3)
```

Altogether, none of the unsupervised clustering techniques showed the `blue cloud', `green valley', `red sequence' and `BCGs' as well as a simple kernel smoother. The results of unsupervised clustering, including the astronomers' favorite `friends-of-friends' algorithm, are often unreliable in the sense that reasonable alternative algorithms give very different scientific results.

++++++++++++++++++
++++++++++++++++++

Supervised classification of SDSS point sources

The Sloan Digital Sky Survey (SDSS) has produced some of the most impressive photometric catalogs in modern astronomy. A selection of 17,000 SDSS point sources, along with training sets for three spectroscopically confirmed classes (main sequence plus red giant stars, quasars, and white dwarfs). These are 4-dimensional datasets

with variables representing the ratios of brightness in the five SDSS photometric bands (u-g, g-r, r-i, and i-z). The resulting color-color scatterplots show distributions that cannot be well-modeled by multinormal distributions, and distributions that are distinct in some variables but overlapping in others. The analysis here starts with the SDSS_train and SDSS_test obtained using the following R script.

```
# SDSS point sources dataset, N=17,000 (mag<21, point sources, hi-qual)

SDSS=read.csv('http://astrostatistics.psu.edu/MSMA/datasets/SDSS_test.csv',header=T)
dim(SDSS) ; summary(SDSS)
SDSS_test=data.frame(cbind((SDSS[,1]-SDSS[,2]),(SDSS[,2]-SDSS[,3]),(SDSS[,3]-SDSS[,4]),(SDSS[,4]-SDSS[,5])))
names(SDSS_test)=c('u_g','g_r','r_i','i_z')
str(SDSS_test)

par(mfrow=c(1,3))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c(-0.7,1.8),pch=20, cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r (mag)')

plot(SDSS_test[,2],SDSS_test[,3],xlim=c(-0.7,1.8),ylim=c(-0.7,1.8),pch=20, cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='g-r (mag)',ylab='r-i (mag)')

plot(SDSS_test[,3],SDSS_test[,4],xlim=c(-0.7,1.8),ylim=c(-1.1,1.3),pch=20, cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='r-i (mag)',ylab='i-z (mag)')
par(mfrow=c(1,1))

# Quasar training set, N=2000 (Class 1)

temp1 = read.table('http://astrostatistics.psu.edu/MSMA/datasets/SDSS_QSO.dat', header=T)
dim(temp1) ; summary(temp1)
qso = cbind(temp1[,c(5,7,9,11,13,6,8,10,12,14,2,3)]) # set same variables in both datasets
```

```
bad_phot_qso = which(qso[,1:6] > 21.0 | qso[,9]==0)
qso1 = qso[-bad_phot_qso,]
qso2 = qso1[1:2000,]
qso3=cbind((qso2[,1]-qso2[,2]),(qso2[,2]-qso2[,3]),(qso2[,3]-qso2[,
4]),(qso2[,4]-qso2[,5]))
qso_train=data.frame(cbind(qso3,rep(1,length(qso2[,1]))))
names(qso_train)=c('u_g','g_r','r_i','i_z','Class')
dim(qso_train) ; summary(qso_train)

# Star training set, N=5000 (Class 2)

temp2 = read.csv('http://astrostatistics.psu.edu/MSMA/datasets/
SDSS_stars.csv', header=T)
dim(temp2) ; summary(temp2)
star=cbind((temp2[,1]-temp2[,2]),(temp2[,2]-temp2[,3]),(temp2[,3]-
temp2[,4]),(temp2[,4]-temp2[,5]))
star_train=data.frame(cbind(star,rep(2,length(star[,1]))))
names(star_train)=c('u_g','g_r','r_i','i_z','Class')
dim(star_train) ; summary(star_train)

# White dwarf training set, N=2000 (Class 3)

temp3 = read.csv('http://astrostatistics.psu.edu/MSMA/datasets/
SDSS_wd.csv', header=T)
dim(temp3) ; summary(temp3)
temp3=na.omit(temp3)
wd =cbind((temp3[1:2000,2]-temp3[1:2000,3]),(temp3[1:2000,3]-temp3
[1:2000,4]),(temp3[1:2000,4]-temp3[1:2000,5]),(temp3[1:2000,5]-temp3
[1:2000,6]))
wd_train=data.frame(cbind(wd,rep(3,length(wd[,1]))))
names(wd_train)=c('u_g','g_r','r_i','i_z','Class')
dim(wd_train) ; summary(wd_train)

# Combined training set (9000 objects)

SDSS_train=data.frame(rbind(qso_train,star_train,wd_train))
names(SDSS_train)=c('u_g','g_r','r_i','i_z','Class')
str(SDSS_train)

par(mfrow=c(1,3))
plot(SDSS_train[,1],SDSS_train[,2],xlim=c(-0.7,3),ylim=c
(-0.7,1.8),pch=20, col=SDSS_train[,
```

```
5],cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g
(mag)',ylab='g-r (mag)')

legend(-0.5,1.7,c('QSO','MS + RG','WD'), pch=20,col=c
('black','red','green'),cex=1.8)

plot(SDSS_train[,2],SDSS_train[,3],xlim=c(-0.7,1.8),ylim=c
(-0.7,1.8),pch=20, col=SDSS_train[,
5],cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='g-r
(mag)',ylab='r-i (mag)')

plot(SDSS_train[,3],SDSS_train[,4],xlim=c(-0.7,1.8),ylim=c
(-1.1,1.3),pch=20, col=SDSS_train[,
5],cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='r-i
(mag)',ylab='i-z (mag)')
par(mfrow=c(1,1))
```

Unsupervised clustering fails to recover the known distributions in the SDSS photometric distribution. We show, for example, the result of a k-means partitioning in the figure. The k-means partitioning with divides the main sequence into segments, even though there are no gaps in the distribution. Even a supervised k-means partitioning with three initial cluster roughly centered on the three training classes does not lead to a correct result. Similar problems arise when unsupervised hierarchical clustering (R function hclust) or model-based clustering (function Mclust in package mclust) are applied to the SDSS distributions.

```
# Unsupervised k-means partitioning

SDSS_kmean=kmeans(SDSS_test,6)
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.5,3),ylim=c
(-0.5,2),pch=20,col=SDSS_kmean$cluster, cex=0.6,
cex.lab=1.3,cex.axis=1.3,cex.main=1.3,main='SDSS unsupervised k-
means',xlab='u-g (mag)',ylab='g-r (mag)')
```

Discrimination analysis and k-nn classification

The classification of SDSS objects is much improved when the training set used. We show here the result of linear discriminant analysis (LDA) using function lda in base-R's MASS package. The LDA

classification from the training set is applied to the test set using R's predict function. The figure (top left panel) shows the result for the test sample; a similar plot can be inspected for the training sample.

Discriminant analysis gives a reasonable classification of stars, quasars and white dwarfs, with no difficulty following the elongated and curved distributions in 4-space. However, some classification errors are evident: a few main sequence stars are mislabeled as quasars (black dots), and the white dwarf class (green dots) is truncated by the quasar distribution. The closely related quadratic discriminate analysis using function qda has additional problems, classifying some main sequence and red giant stars as white dwarfs.

CRAN package class implements a k-nearest neighbors classifier where a grid of classifications is constructed from the training set. The application to the SDSS test set is shown in the figure (top right panel) and shows good performance. Here we use k=4 neighbors, but the result is not sensitive to a range of k values.

We consider two ways to examine the reliability of the these classifiers by applying it to the training set. First, a cross-validation experiment can be made (e.g., using function knn.cv) where leave-one-out resamples of the test dataset give posterior probabilities for the classification of each object. Second, the class obtained by the classified can be plotted against the true class known for the training set objects. We show this in the bottom panels of the figure; the LDA clearly makes more misclassifications than the k-nn algorithm. For k-nn, misclassification of stars (Class 2) is rare (0.1%) but confusion between quasars (Class 1) and white dwarfs (Class 3) occurs in about 2% of cases. This is understandable given the overlap in their distributions in color-color plots. Note the use of R's jitter function to facilitate visualization of categorical data for scatterplots.

```
# Linear discriminant analysis

library(MASS)
SDSS_lda=lda(SDSS_train[,1:4],as.factor(SDSS_train[,5]))
SDSS_train_lda=predict(SDSS_lda,SDSS_train[,1:4])
SDSS_test_lda=predict(SDSS_lda,SDSS_test[,1:4])
```

```
par(mfrow=c(1,2))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c
(-0.7,1.8),pch=20, col=SDSS_test_lda$class,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r
(mag)')


# k-nn classification

install.packages('class') ; library(class)
SDSS_knn4 = knn(SDSS_train[,1:4],SDSS_test,SDSS_train[,
5],k=4,prob=T)
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c
(-0.7,1.8),pch=20, col=SDSS_knn4,
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r
(mag)')
par(mfrow=c(1,1))

# Validation of k-nn classification

SDSS_train_lda_cv=lda(SDSS_train[,1:4],as.factor(SDSS_train[,
5]),CV=T)

SDSS_train_lda=lda(SDSS_train[,1:4],as.factor(SDSS_train[,5]))
SDSS_train_knn4 = knn(SDSS_train[,1:4],SDSS_train[,1:4],SDSS_train[,
5],k=4)
par(mfrow=c(1,2))
plot(jitter(as.numeric(SDSS_train_lda$class),factor=0.5),jitter
(as.numeric(SDSS_train[,5]),factor=0.5),pch=20,cex=1.0,xlab='LDA
class',ylab='True class',cex.lab=1.3,cex.axis=1.3, xaxp=c
(1,3,2),yaxp=c(1,3,2))
plot(jitter(as.numeric(SDSS_train_knn4),factor=0.5),jitter
(as.numeric(SDSS_train[,5]),factor=0.5),pch=20,cex=1.0,xlab='k-nn
class',ylab='True class',cex.lab=1.3,cex.axis=1.3, xaxp=c
(1,3,2),yaxp=c(1,3,2))
par(mfrow=c(1,1))

# Single layer neutral network

install.packages('nnet') ; library(nnet)
options(size=100, maxit=1000)
SDSS_nnet <- multinom(as.factor(SDSS_train[,5]) ~ SDSS_train[,1] +
```

```
SDSS_train[,2] +
SDSS_train[,3] + SDSS_train[,4], data=SDSS_train)
SDSS_train_nnet <- predict(SDSS_nnet,SDSS_train[,1:4])
plot(jitter(as.numeric(SDSS_train_nnet), factor=0.5), jitter
(as.numeric(SDSS_train[,5]), factor=0.5), pch=20, cex=0.5,
xlab='nnet class', ylab='True class', xaxp=c(1,3,2),yaxp=c(1,3,2))
```

Machine learning classifiers

Machine learning classifiers perform well for this problem. In the following R script, we apply CART using rpart (acronym for `recursive partitioning and regression trees') in base-R's rpart library, and the Support Vector Machine {svm implemented in CRAN's e1071 package. The procedure for running these and similar classifiers is straightforward. The `model' is produced by rpart or svm with a formula like `Known_classes ~ .' to the training set. Examining the model using summary and str shows that the classifier output can be quite complicated; e.g., CART will give details on the decision tree nodes while SVM will give details on the support vectors. But the model predictions can be automatically applied to the training and test datasets using R's predict function without understanding these details.

We plot the predicted classes against the known classes for the training set in the figure. CART does not perform as well as the k-nn shown above, but the SVM classifier does a better job. The figure shows the CART tree with the splits labeled, and the next figure shows how much of the variance is reduced by each split of the data.

Considering the SVM classification as the best available, we show the final classifications of the test SDSS sample in the figure, and write them to an ASCII output file SDSS_test_svm.out. Note that R's write function produces tables that are difficult to read; we use the format function and other options in write to improve the appearance of the ASCII output.

```
# Classification And Regression Tree model, prediction and
validation

library('rpart')
```

```
SDSS_rpart_mod = rpart(SDSS_train[,5] ~.,data=SDSS_train[,1:4])
SDSS_rpart_test_pred = predict(SDSS_rpart_mod, SDSS_test)
SDSS_rpart_train_pred = predict(SDSS_rpart_mod, SDSS_train)
summary(SDSS_rpart_mod) ; str(SDSS_rpart_mod)
par(mfrow=c(1,2))
plot(jitter(SDSS_rpart_train_pred,factor=5),jitter(SDSS_train[,
5]),pch=20,cex=0.3,cex.axis=1.5, cex.lab=1.5,xlab='CART
class',ylab='True class',yaxp=c(1,3,2))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c
(-0.7,1.8),pch=20, col=round(SDSS_rpart_test_pred),
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r
(mag)')
plot(SDSS_rpart_mod) ; text(SDSS_rpart_mod)
plotcp(SDSS_rpart_mod,lwd=2,cex.axis=1.3,cex.lab=1.3)

# Support Vector Machine model, prediction and validation

install.packages('e1071') ; library(e1071)
SDSS_svm_mod = svm(SDSS_train[,5] ~.,data=SDSS_train[,1:4],cost =
100, gamma = 1)
summary(SDSS_svm_mod) ; str(SDSS_svm_mod)
SDSS_svm_test_pred = predict(SDSS_svm_mod, SDSS_test)
SDSS_svm_train_pred = predict(SDSS_svm_mod, SDSS_train)
par(mfrow=c(1,2))
plot(SDSS_svm_train_pred,jitter(SDSS_train[,
5]),pch=20,cex=0.3,cex.axis=1.5, cex.lab=1.5,xlab='SVM
class',ylab='True class',yaxp=c(1,3,2))
plot(SDSS_test[,1],SDSS_test[,2],xlim=c(-0.7,3),ylim=c
(-0.7,1.8),pch=20, col=round(SDSS_svm_test_pred),
cex=0.6,cex.lab=1.5,cex.axis=1.5,main='',xlab='u-g (mag)',ylab='g-r
(mag)')


# 23 new R functions are used in this tutorial:
# text, cbind, dist, hclust, plclust, rect.hclust, dbscan,
print.dbscan,
# points, mclustBIC, read.csv, data.frame, which, kmeans, lda,
predict,
# knn, as.numeric, jitter, multinom, as.factor, rpart, svm
```