Audience Segmentation

## Read fake data

This data is simulated for a client proposal

seg.raw <- read.csv("CRMData.csv")  
seg.df <- seg.raw[ , -7] # a copy without the known segment assignments  
  
summary(seg.df)

## age gender income kids ownHome   
## Min. :19.26 Female:157 Min. : -5183 Min. :0.00 ownNo :159   
## 1st Qu.:33.01 Male :143 1st Qu.: 39656 1st Qu.:0.00 ownYes:141   
## Median :39.49 Median : 52014 Median :1.00   
## Mean :41.20 Mean : 50937 Mean :1.27   
## 3rd Qu.:47.90 3rd Qu.: 61403 3rd Qu.:2.00   
## Max. :80.49 Max. :114278 Max. :7.00   
## subscribe   
## subNo :260   
## subYes: 40   
##   
##   
##   
##

str(seg.df)

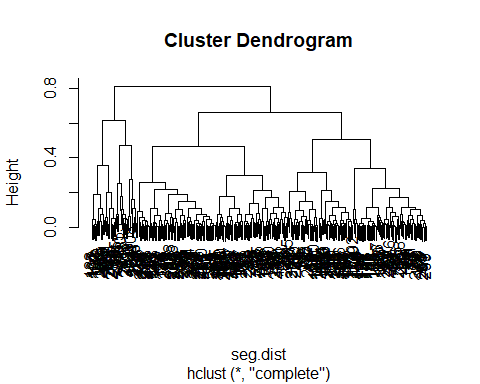
## 'data.frame': 300 obs. of 6 variables:  
## $ age : num 47.3 31.4 43.2 37.3 41 ...  
## $ gender : Factor w/ 2 levels "Female","Male": 2 2 2 1 1 2 2 2 1 1 ...  
## $ income : num 49483 35546 44169 81042 79353 ...  
## $ kids : int 2 1 0 1 3 4 3 0 1 0 ...  
## $ ownHome : Factor w/ 2 levels "ownNo","ownYes": 1 2 2 1 2 2 1 1 1 2 ...  
## $ subscribe: Factor w/ 2 levels "subNo","subYes": 1 1 1 1 1 1 1 1 1 1 ...

### Build clustering solution

# now the real hclust() work  
library(cluster) # daisy works with mixed data types  
seg.dist <- daisy(seg.df)  
# inspect some of the results  
as.matrix(seg.dist)[1:5, 1:5]

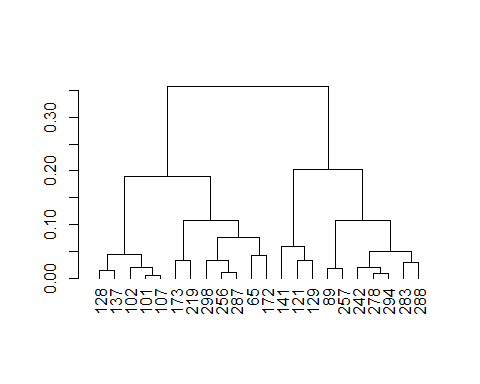
## 1 2 3 4 5  
## 1 0.0000000 0.2532815 0.2329028 0.2617250 0.4161338  
## 2 0.2532815 0.0000000 0.0679978 0.4129493 0.3014468  
## 3 0.2329028 0.0679978 0.0000000 0.4246012 0.2932957  
## 4 0.2617250 0.4129493 0.4246012 0.0000000 0.2265436  
## 5 0.4161338 0.3014468 0.2932957 0.2265436 0.0000000

seg.hc <- hclust(seg.dist, method="complete")  
  
plot(seg.hc)

 A hierarchical dendrogram is interpreted primarily by height and where observations are joined. The height represents the dissimilarity between elements that are joined.

Let us zoom into one section of the chart

plot(cut(as.dendrogram(seg.hc), h=0.5)$lower[[1]])

 Check the proposed similarities

# check some of the proposed similarities  
seg.df[c(101, 107), ] # similar

## age gender income kids ownHome subscribe  
## 101 24.73796 Male 18457.85 1 ownNo subYes  
## 107 23.19013 Male 17510.28 1 ownNo subYes

seg.df[c(278, 294), ] # similar

## age gender income kids ownHome subscribe  
## 278 36.23860 Female 46540.88 1 ownNo subYes  
## 294 35.79961 Female 52352.69 1 ownNo subYes

seg.df[c(173, 141), ] # less similar

## age gender income kids ownHome subscribe  
## 173 64.70641 Male 45517.15 0 ownNo subYes  
## 141 25.17703 Female 20125.80 2 ownNo subYes

As you can see, these segments seem quite similar.

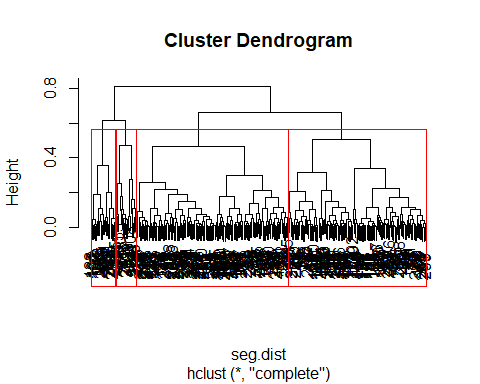
# examine cophenetic correlation  
cor(cophenetic(seg.hc), seg.dist)

## [1] 0.7682436

CPCC > 0.7 indicates a relatively strong fit, meaning that the hierarchical tree represents the distances between customers well.

Let us try to cut the dendrogram such that we get 4 clusters

plot(seg.hc)  
rect.hclust(seg.hc, k=4, border="red")



# actually get 4 groups  
seg.hc.segment <- cutree(seg.hc, k=4) # membership vector for 4 groups  
table(seg.hc.segment)

## seg.hc.segment  
## 1 2 3 4   
## 124 136 18 22

We see that groups 1 and 2 dominate the assignment. Note that the class labels (1, 2, 3, 4) are in arbitrary order and are not meaningful in themselves. seg.hc.segment is the vector of group assignments.

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

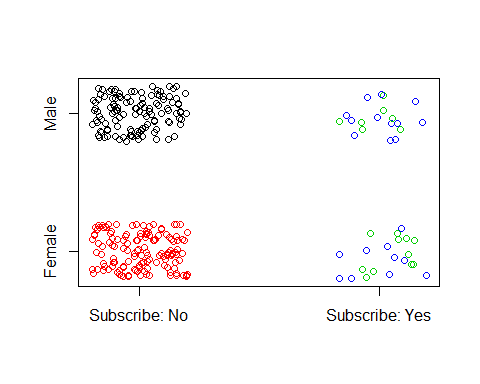
seg.df$seg.hc.segment = seg.hc.segment  
seg.summ <- function(data, groups) {  
 aggregate(data, list(groups), function(x) mean(as.numeric(x)))   
}  
  
numeric\_mean <- function(col){  
 return (mean(as.numeric(col)))  
}  
seg.df %>% group\_by(seg.hc.segment) %>% summarize\_each(funs(numeric\_mean))

## `summarise\_each()` is deprecated.  
## Use `summarise\_all()`, `summarise\_at()` or `summarise\_if()` instead.  
## To map `funs` over all variables, use `summarise\_all()`

## # A tibble: 4 x 7  
## seg.hc.segment age gender income kids ownHome subscribe  
## <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 40.8 2.00 49454. 1.31 1.47 1.  
## 2 2 42.0 1.00 53760. 1.24 1.48 1.  
## 3 3 44.3 1.39 52628. 1.39 2.00 2.  
## 4 4 35.8 1.55 40456. 1.14 1.00 2.

#seg.summ(data = seg.df,groups = seg.hc.segment)

# plot this  
plot(jitter(as.numeric(seg.df$gender)) ~ jitter(as.numeric(seg.df$subscribe)),   
 col=seg.hc.segment, yaxt="n", xaxt="n", ylab="", xlab="")  
axis(1, at=c(1, 2), labels=c("Subscribe: No", "Subscribe: Yes"))  
axis(2, at=c(1, 2), labels=levels(seg.df$gender))



### Perform k-means clustering

# convert factor variables to numeric (kmeans requires). OK b/c all are binary.  
seg.df.num <- seg.df  
seg.df.num$gender <- ifelse(seg.df$gender=="Male", 0, 1)  
seg.df.num$ownHome <- ifelse(seg.df$ownHome=="ownNo", 0, 1)  
seg.df.num$subscribe <- ifelse(seg.df$subscribe=="subNo", 0, 1)  
summary(seg.df.num)

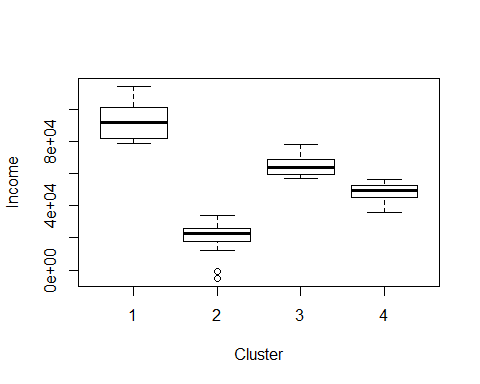
## age gender income kids   
## Min. :19.26 Min. :0.0000 Min. : -5183 Min. :0.00   
## 1st Qu.:33.01 1st Qu.:0.0000 1st Qu.: 39656 1st Qu.:0.00   
## Median :39.49 Median :1.0000 Median : 52014 Median :1.00   
## Mean :41.20 Mean :0.5233 Mean : 50937 Mean :1.27   
## 3rd Qu.:47.90 3rd Qu.:1.0000 3rd Qu.: 61403 3rd Qu.:2.00   
## Max. :80.49 Max. :1.0000 Max. :114278 Max. :7.00   
## ownHome subscribe seg.hc.segment   
## Min. :0.00 Min. :0.0000 Min. :1.000   
## 1st Qu.:0.00 1st Qu.:0.0000 1st Qu.:1.000   
## Median :0.00 Median :0.0000 Median :2.000   
## Mean :0.47 Mean :0.1333 Mean :1.793   
## 3rd Qu.:1.00 3rd Qu.:0.0000 3rd Qu.:2.000   
## Max. :1.00 Max. :1.0000 Max. :4.000

set.seed(96743)  
seg.k <- kmeans(seg.df.num, centers=4)

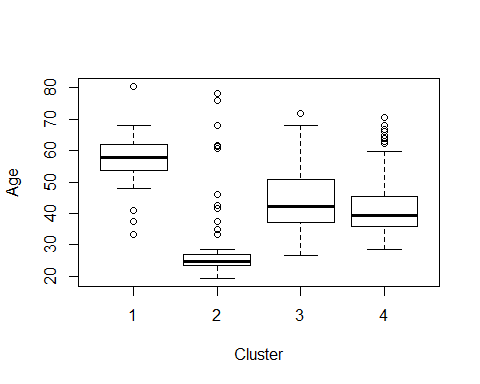
# inspect it  
seg.summ(seg.df, seg.k$cluster)

## Group.1 age gender income kids ownHome subscribe  
## 1 1 56.37245 1.428571 92287.07 0.4285714 1.857143 1.142857  
## 2 2 29.58704 1.571429 21631.79 1.0634921 1.301587 1.158730  
## 3 3 44.42051 1.452632 64703.76 1.2947368 1.421053 1.073684  
## 4 4 42.08381 1.454545 48208.86 1.5041322 1.528926 1.165289  
## seg.hc.segment  
## 1 1.809524  
## 2 1.809524  
## 3 1.694737  
## 4 1.859504

# plot one of the variables  
boxplot(seg.df.num$income ~ seg.k$cluster, ylab="Income", xlab="Cluster")



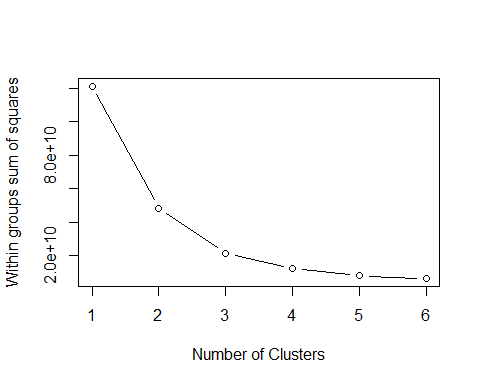
boxplot(seg.df.num$age ~ seg.k$cluster, ylab="Age", xlab="Cluster")



### Scree Plot

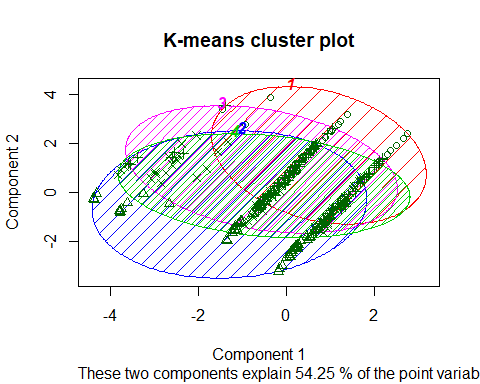
Draw a scree plot to determine the number of clusters.

wssplot <- function(data, nc=15, seed=1234){  
 wss <- (nrow(data)-1)\*sum(apply(data,2,var))  
 for (i in 2:nc){  
 set.seed(seed)  
 wss[i] <- sum(kmeans(data, centers=i)$withinss)}  
 plot(1:nc, wss, type="b", xlab="Number of Clusters",  
 ylab="Within groups sum of squares")}  
  
wssplot(seg.df.num, nc=6)



### Plot the results

# plot the result  
library(cluster)  
clusplot(seg.df, seg.k$cluster, color=TRUE, shade=TRUE,   
 labels=4, lines=0, main="K-means cluster plot")



Overall, this is a far more interesting cluster solution for our segmentation data than the hclust() proposal. The groups here are clearly differentiated on key variables such as age and income.With this information, an analyst might cross-reference the group membership with key variables (as we did using our seg.summ() function and then look at the relative differentiation of the groups.

This may suggest a business strategy. In the present case, for instance, we see that group 1 is modestly well differentiated, and has the highest average income. That may make it a good target for a potential campaign. Many other strategies are possible, too; the key point is that the analysis provides interesting options to consider. A limitation of k-means analysis is that it requires specifying the number of clusters, and it can be difficult to determine whether one solution is better than another. If we were to use k-means for the present problem, we would repeat the analysis for k =3, 4, 5, and so forth, and determine which solution gives the most useful result for our business goals. One might wonder whether the algorithm itself can suggest how many clusters are in the data. Yes! To see that, we turn next to model-based clustering.

### Model Based Clustering (MCLUST)

The key idea for model-based clustering is that observations come from groups with different statistical distributions (such as different means and variances). The algorithms try to find the best set of such underlying distributions to explain the observed data. We use the mclust package to demonstrate this. Such models are also known as “mixture models” because it is assumed that the data reflect a mixture of observations drawn from different populations, although we don’t know which population each observation was drawn from. We are trying to estimate the underlying population parameters and the mixture proportion. mclust models such clusters as being drawn from a mixture of normal (also known as Gaussian) distributions. As you might guess, because mclust models data with normal distributions, it uses only numeric data. We use the numeric data frame seg.df.num that we adapted for kmeans(). The model is estimated with Mclust()

# do mclust for segments  
library(mclust)

## Package 'mclust' version 5.4  
## Type 'citation("mclust")' for citing this R package in publications.

###  
# convert factor variables to numeric (mclust requires). OK b/c all are binary.  
# these lines are the same as above for k-means [not repeated in book]  
seg.df.num <- seg.df  
seg.df.num$gender <- ifelse(seg.df$gender=="Male", 0, 1)  
seg.df.num$ownHome <- ifelse(seg.df$ownHome=="ownNo", 0, 1)  
seg.df.num$subscribe <- ifelse(seg.df$subscribe=="subNo", 0, 1)  
summary(seg.df.num)

## age gender income kids   
## Min. :19.26 Min. :0.0000 Min. : -5183 Min. :0.00   
## 1st Qu.:33.01 1st Qu.:0.0000 1st Qu.: 39656 1st Qu.:0.00   
## Median :39.49 Median :1.0000 Median : 52014 Median :1.00   
## Mean :41.20 Mean :0.5233 Mean : 50937 Mean :1.27   
## 3rd Qu.:47.90 3rd Qu.:1.0000 3rd Qu.: 61403 3rd Qu.:2.00   
## Max. :80.49 Max. :1.0000 Max. :114278 Max. :7.00   
## ownHome subscribe seg.hc.segment   
## Min. :0.00 Min. :0.0000 Min. :1.000   
## 1st Qu.:0.00 1st Qu.:0.0000 1st Qu.:1.000   
## Median :0.00 Median :0.0000 Median :2.000   
## Mean :0.47 Mean :0.1333 Mean :1.793   
## 3rd Qu.:1.00 3rd Qu.:0.0000 3rd Qu.:2.000   
## Max. :1.00 Max. :1.0000 Max. :4.000

###  
  
  
# fit the model  
seg.mc <- Mclust(seg.df.num)  
summary(seg.mc)

## ----------------------------------------------------  
## Gaussian finite mixture model fitted by EM algorithm   
## ----------------------------------------------------  
##   
## Mclust VEV (ellipsoidal, equal shape) model with 2 components:  
##   
## log.likelihood n df BIC ICL  
## -5079.071 300 65 -10528.89 -10528.89  
##   
## Clustering table:  
## 1 2   
## 124 176

# what if we estimate 4 clusters?  
seg.mc4 <- Mclust(seg.df.num, G=4)  
summary(seg.mc4)

## ----------------------------------------------------  
## Gaussian finite mixture model fitted by EM algorithm   
## ----------------------------------------------------  
##   
## Mclust VII (spherical, varying volume) model with 4 components:  
##   
## log.likelihood n df BIC ICL  
## -19420.54 300 35 -39040.71 -39043.4  
##   
## Clustering table:  
## 1 2 3 4   
## 86 89 59 66

# Develop a 3 cluster model

# fit the model  
seg.mc3 <- Mclust(seg.df.num, G=3)  
summary(seg.mc3)

## ----------------------------------------------------  
## Gaussian finite mixture model fitted by EM algorithm   
## ----------------------------------------------------  
##   
## Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3 components:  
##   
## log.likelihood n df BIC ICL  
## -5304.038 300 51 -10898.97 -10901.88  
##   
## Clustering table:  
## 1 2 3   
## 66 163 71

# Compare the two models

# compare the three models  
logLik(seg.mc, seg.mc3, seg.mc4)

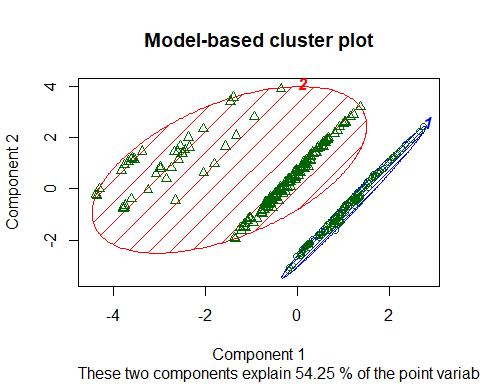
## 'log Lik.' -5079.071 (df=65)

# examine the 3-cluster model  
seg.summ(seg.df, seg.mc3$class)

## Group.1 age gender income kids ownHome subscribe  
## 1 1 36.02187 2.000000 45227.51 1.348485 1.000000 1.000000  
## 2 2 44.68018 1.472393 52980.52 1.171779 1.865031 1.245399  
## 3 3 38.02229 1.000000 51550.98 1.422535 1.000000 1.000000  
## seg.hc.segment  
## 1 1.00000  
## 2 2.02454  
## 3 2.00000

### Plot the 2-cluster model

# plot the 3-cluster model  
library(cluster)  
clusplot(seg.df, seg.mc$class, color=TRUE, shade=TRUE,   
 labels=4, lines=0, main="Model-based cluster plot")



### Latent Class Analysis: poLCA()

Latent class analysis (LCA) is similar to mixture modeling in the assumption that differences are attributable to unobserved groups that one wishes to uncover. In this section we take a look at the poLCA package for polytomous (i.e., categorical) LCA. Whereas mclust and kmeans() work with numeric data, and hclust() depends on the distance measure, poLCA uses only categorical variables. To demonstrate it here, we adopt an opposite strategy from our procedure with k-means and mclust and convert our data seg.df to be all categorical data before analyzing it.

There are several approaches to convert numeric data to factors, but for purposes here we simply recode everything as binary with regard to a specified cutting point (for instance, to recode as 1 for income below some cutoff and 2 above that). In the present case, we split each variable at the median() and recode using ifelse() and factor(). We use with() to save typing, and ~1 to specify a formula with intercepts only:

seg.df.cut <- seg.df  
seg.df.cut$age <- factor(ifelse(seg.df$age < median(seg.df$age), 1, 2))  
seg.df.cut$income <- factor(ifelse(seg.df$income < median(seg.df$income),  
 1, 2))  
seg.df.cut$kids <- factor(ifelse(seg.df$kids < median(seg.df$kids), 1, 2))  
summary(seg.df.cut)

## age gender income kids ownHome subscribe   
## 1:150 Female:157 1:150 1:121 ownNo :159 subNo :260   
## 2:150 Male :143 2:150 2:179 ownYes:141 subYes: 40   
##   
##   
##   
##   
## seg.hc.segment   
## Min. :1.000   
## 1st Qu.:1.000   
## Median :2.000   
## Mean :1.793   
## 3rd Qu.:2.000   
## Max. :4.000

# create a model formula  
seg.f <- with(seg.df.cut,   
 cbind(age, gender, income, kids, ownHome, subscribe)~1)

With the data in place, we specify the model that we want to fit. poLCA can estimate complex models with covariates, but for the present analysis we only wish Segmentation: Clustering and Classification to examine the effect of cluster membership alone. Thus, we model the dependent variables (all the observed columns) with respect to the model intercepts (i.e., the cluster positions).

# fit the model  
library(poLCA)

## Loading required package: scatterplot3d

## Loading required package: MASS

##   
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':  
##   
## select

set.seed(02807)  
seg.LCA3 <- poLCA(seg.f, data=seg.df.cut, nclass=3)

## Conditional item response (column) probabilities,  
## by outcome variable, for each class (row)   
##   
## $age  
## 1 2  
## class 1: 1.0000 0.0000  
## class 2: 0.0000 1.0000  
## class 3: 0.6555 0.3445  
##   
## $gender  
## Female Male  
## class 1: 0.4211 0.5789  
## class 2: 0.4681 0.5319  
## class 3: 0.6079 0.3921  
##   
## $income  
## 1 2  
## class 1: 1.0000 0.0000  
## class 2: 0.3803 0.6197  
## class 3: 0.3746 0.6254  
##   
## $kids  
## 1 2  
## class 1: 0.2818 0.7182  
## class 2: 0.8065 0.1935  
## class 3: 0.1575 0.8425  
##   
## $ownHome  
## ownNo ownYes  
## class 1: 0.7289 0.2711  
## class 2: 0.2338 0.7662  
## class 3: 0.6638 0.3362  
##   
## $subscribe  
## subNo subYes  
## class 1: 0.7496 0.2504  
## class 2: 0.8948 0.1052  
## class 3: 0.8960 0.1040  
##   
## Estimated class population shares   
## 0.1974 0.341 0.4616   
##   
## Predicted class memberships (by modal posterior prob.)   
## 0.2333 0.3467 0.42   
##   
## =========================================================   
## Fit for 3 latent classes:   
## =========================================================   
## number of observations: 300   
## number of estimated parameters: 20   
## residual degrees of freedom: 43   
## maximum log-likelihood: -1092.345   
##   
## AIC(3): 2224.691  
## BIC(3): 2298.767  
## G^2(3): 42.77441 (Likelihood ratio/deviance statistic)   
## X^2(3): 38.47647 (Chi-square goodness of fit)   
##

seg.LCA4 <- poLCA(seg.f, data=seg.df.cut, nclass=4)

## Conditional item response (column) probabilities,  
## by outcome variable, for each class (row)   
##   
## $age  
## 1 2  
## class 1: 0.6823 0.3177  
## class 2: 0.0000 1.0000  
## class 3: 1.0000 0.0000  
## class 4: 1.0000 0.0000  
##   
## $gender  
## Female Male  
## class 1: 0.5853 0.4147  
## class 2: 0.4810 0.5190  
## class 3: 0.8466 0.1534  
## class 4: 0.3277 0.6723  
##   
## $income  
## 1 2  
## class 1: 0.4137 0.5863  
## class 2: 0.3701 0.6299  
## class 3: 0.5850 0.4150  
## class 4: 1.0000 0.0000  
##   
## $kids  
## 1 2  
## class 1: 0.0000 1.0000  
## class 2: 0.8114 0.1886  
## class 3: 1.0000 0.0000  
## class 4: 0.2506 0.7494  
##   
## $ownHome  
## ownNo ownYes  
## class 1: 0.6540 0.3460  
## class 2: 0.2688 0.7312  
## class 3: 0.6537 0.3463  
## class 4: 0.7721 0.2279  
##   
## $subscribe  
## subNo subYes  
## class 1: 0.8746 0.1254  
## class 2: 0.8965 0.1035  
## class 3: 1.0000 0.0000  
## class 4: 0.7203 0.2797  
##   
## Estimated class population shares   
## 0.4101 0.3697 0.0643 0.1559   
##   
## Predicted class memberships (by modal posterior prob.)   
## 0.41 0.3733 0.0667 0.15   
##   
## =========================================================   
## Fit for 4 latent classes:   
## =========================================================   
## number of observations: 300   
## number of estimated parameters: 27   
## residual degrees of freedom: 36   
## maximum log-likelihood: -1088.021   
##   
## AIC(4): 2230.041  
## BIC(4): 2330.043  
## G^2(4): 34.12473 (Likelihood ratio/deviance statistic)   
## X^2(4): 31.50696 (Chi-square goodness of fit)   
##

seg.LCA4$bic

## [1] 2330.043

seg.LCA3$bic

## [1] 2298.767

The 3-cluster model shows a lower BIC by 32 and thus a substantially stronger fit to the data. As we’ve seen, that is not entirely conclusive as to business utility, so we also examine some other indicators such as the quick summary function and cluster plots:

# examine the solutions  
# 3 clusters  
seg.summ(seg.df, seg.LCA3$predclass)

## Group.1 age gender income kids ownHome subscribe  
## 1 1 28.22385 1.685714 30075.32 1.1285714 1.285714 1.271429  
## 2 2 54.44407 1.576923 60082.47 0.3846154 1.769231 1.105769  
## 3 3 37.47652 1.277778 54977.08 2.0793651 1.325397 1.079365  
## seg.hc.segment  
## 1 1.900000  
## 2 1.634615  
## 3 1.865079

seg.summ(seg.df, seg.LCA4$predclass)

## Group.1 age gender income kids ownHome subscribe  
## 1 1 36.62554 1.349593 52080.13 2.1951220 1.349593 1.113821  
## 2 2 53.64073 1.535714 60534.17 0.5178571 1.785714 1.098214  
## 3 3 30.22575 1.050000 41361.81 0.0000000 1.350000 1.000000  
## 4 4 27.61506 1.866667 28178.70 1.1777778 1.066667 1.333333  
## seg.hc.segment  
## 1 1.829268  
## 2 1.660714  
## 3 1.950000  
## 4 1.955556

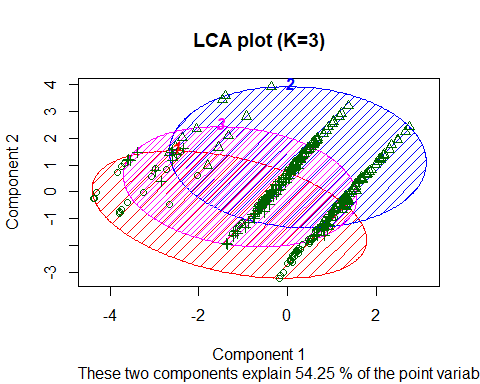
table(seg.LCA3$predclass)

##   
## 1 2 3   
## 70 104 126

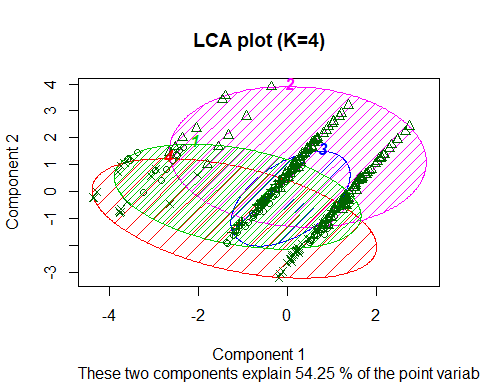
table(seg.LCA4$predclass)

##   
## 1 2 3 4   
## 123 112 20 45

clusplot(seg.df, seg.LCA3$predclass, color=TRUE, shade=TRUE,   
 labels=4, lines=0, main="LCA plot (K=3)")



clusplot(seg.df, seg.LCA4$predclass, color=TRUE, shade=TRUE,   
 labels=4, lines=0, main="LCA plot (K=4)")



At a high level, it appears that “Group 2” is similar in both solutions. The primary difference is that “Group 3” buried inside the overlapping ellipses in the 4-cluster solution could be viewed as being largely carved out of two larger groups (Groups “2” and “3” as labeled in the 3-cluster solution). This is an approximate interpretation of the data visualization, not a perfect correspondence.

Does the additional group in the 4-cluster solution add anything to our interpretation? Turning to the quick summary from seg.summ() in the code block, we see good differentiation of groups in both models. One argument in favor of the 4-cluster solution is that Group 3 has no subscribers (as shown by the mean in the seg.summ() results) and is relatively well identified (mostly younger women with no kids); that might make it an appealing group either for targeting or exclusion, depending on one’s strategy.

### Comparing Cluster Solutions

mapClass() solves the matching problem. It examines all permutations of how two sets of class assignments might be related and selects a mapping that maximizes agreement between the two assignment schemes. adjustedRandIndex() likewise matches two assignment schemes and then computes the degree of agreement over and above what might be attributed to “chance” by simply assigning all observations to the largest group [81, 131]. Its magnitude may be interpreted similarly to a standard r correlation coefficient.

We use table() to look at the cross-tabs between the LCA 3-cluster and 4-cluster solutions found above:

# compare 3-cluster and 4-cluster solutions  
table(seg.LCA3$predclass, seg.LCA4$predclass)

##   
## 1 2 3 4  
## 1 13 0 12 45  
## 2 0 104 0 0  
## 3 110 8 8 0

It would appear that observations assigned to “Group 1” in the 3-cluster solution are split between Groups 1, 3, and 4 in the 4-cluster solution, while “Group 3” maps closely to “Group 1” (in the 4 class solution) and “Group 2” is predominantly the same in both. However, matching groups manually is sometimes unclear and generally error-prone. Instead, we use mapClass (a, b) and adjustedRandIndex(a, b) to compare agreement between the two solutions:

library(mclust)  
mapClass(seg.LCA3$predclass, seg.LCA4$predclass)

## $aTOb  
## $aTOb$`1`  
## [1] 4  
##   
## $aTOb$`2`  
## [1] 2  
##   
## $aTOb$`3`  
## [1] 1  
##   
##   
## $bTOa  
## $bTOa$`1`  
## [1] 3  
##   
## $bTOa$`2`  
## [1] 2  
##   
## $bTOa$`3`  
## [1] 1  
##   
## $bTOa$`4`  
## [1] 1

adjustedRandIndex(seg.LCA3$predclass, seg.LCA4$predclass)

## [1] 0.7288822

This tells us that “1” in the LCA3 model (a) maps best to “4” in the LCA4 model (b), and so forth. The adjusted Rand index of 0.729 indicates that the match between the two assignment lists is much better than chance. From a business perspective, it also tells us that the 3-cluster and 4-cluster differ modestly from one another, which provides another perspective on choosing between them.

# compare random assignment to LCA4  
set.seed(11021)  
random.data <- sample(4, length(seg.LCA4$predclass), replace=TRUE)  
adjustedRandIndex(random.data, seg.LCA4$predclass)

## [1] 0.002292031

In this case, the adjusted Rand index is near zero, because the match between the clusters is no better than random chance.

Finally we compare the LCA 4-cluster solution to the true segments in seg.raw:

# compare to known segments  
table(seg.raw$Segment, seg.LCA4$predclass)

##   
## 1 2 3 4  
## Moving up 50 4 8 8  
## Suburb mix 62 29 2 7  
## Travelers 0 79 1 0  
## Urban hip 11 0 9 30

adjustedRandIndex(seg.raw$Segment, seg.LCA4$predclass)

## [1] 0.3513031

With a Rand index of 0.35, the LCA solution matches the true segment assignments moderately better than chance alone. In many cases, of course, one would not have identified clusters for comparison; but when they are available from other projects or previous efforts, it is helpful to examine correspondence in this way.

#### Using CLASSIFICATION

First, we will use Naive Bayes.

set.seed(04625)  
train.prop <- 0.65  
train.cases <- sample(nrow(seg.raw), nrow(seg.raw)\*train.prop)  
seg.df.train <- seg.raw[train.cases, ]  
seg.df.test <- seg.raw[-train.cases, ]  
library(e1071)  
(seg.nb <- naiveBayes(Segment ~ ., data=seg.df.train))

##   
## Naive Bayes Classifier for Discrete Predictors  
##   
## Call:  
## naiveBayes.default(x = X, y = Y, laplace = laplace)  
##   
## A-priori probabilities:  
## Y  
## Moving up Suburb mix Travelers Urban hip   
## 0.2512821 0.3025641 0.2615385 0.1846154   
##   
## Conditional probabilities:  
## age  
## Y [,1] [,2]  
## Moving up 36.09168 4.167010  
## Suburb mix 40.14240 5.173803  
## Travelers 57.47194 8.126370  
## Urban hip 23.95040 1.798332  
##   
## gender  
## Y Female Male  
## Moving up 0.6530612 0.3469388  
## Suburb mix 0.4576271 0.5423729  
## Travelers 0.4705882 0.5294118  
## Urban hip 0.3333333 0.6666667  
##   
## income  
## Y [,1] [,2]  
## Moving up 52880.45 9836.682  
## Suburb mix 54124.75 11429.940  
## Travelers 63547.20 23862.123  
## Urban hip 21285.99 5141.259  
##   
## kids  
## Y [,1] [,2]  
## Moving up 2.102041 1.489476  
## Suburb mix 1.694915 1.249196  
## Travelers 0.000000 0.000000  
## Urban hip 1.166667 1.108409  
##   
## ownHome  
## Y ownNo ownYes  
## Moving up 0.6734694 0.3265306  
## Suburb mix 0.5932203 0.4067797  
## Travelers 0.2156863 0.7843137  
## Urban hip 0.8611111 0.1388889  
##   
## subscribe  
## Y subNo subYes  
## Moving up 0.79591837 0.20408163  
## Suburb mix 0.93220339 0.06779661  
## Travelers 0.92156863 0.07843137  
## Urban hip 0.77777778 0.22222222

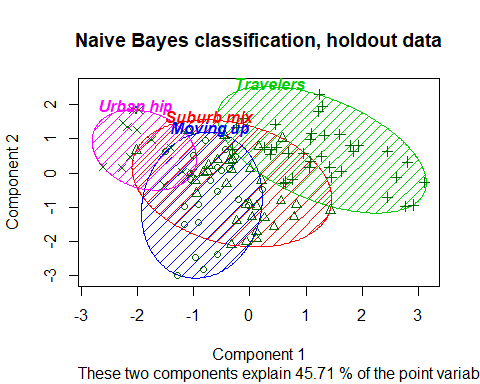
(seg.nb.class <- predict(seg.nb, seg.df.test))

## [1] Suburb mix Travelers Suburb mix Suburb mix Suburb mix Suburb mix  
## [7] Moving up Suburb mix Suburb mix Suburb mix Travelers Moving up   
## [13] Moving up Moving up Suburb mix Moving up Moving up Suburb mix  
## [19] Suburb mix Suburb mix Moving up Suburb mix Suburb mix Moving up   
## [25] Suburb mix Suburb mix Moving up Suburb mix Suburb mix Suburb mix  
## [31] Suburb mix Suburb mix Suburb mix Moving up Suburb mix Suburb mix  
## [37] Suburb mix Suburb mix Suburb mix Suburb mix Suburb mix Urban hip   
## [43] Urban hip Urban hip Urban hip Urban hip Urban hip Urban hip   
## [49] Urban hip Urban hip Urban hip Moving up Urban hip Urban hip   
## [55] Urban hip Travelers Travelers Travelers Travelers Travelers   
## [61] Travelers Travelers Travelers Travelers Travelers Travelers   
## [67] Travelers Travelers Travelers Travelers Travelers Travelers   
## [73] Travelers Travelers Travelers Travelers Travelers Travelers   
## [79] Travelers Travelers Travelers Travelers Travelers Travelers   
## [85] Suburb mix Moving up Moving up Travelers Moving up Travelers   
## [91] Suburb mix Moving up Suburb mix Travelers Moving up Travelers   
## [97] Moving up Moving up Moving up Moving up Moving up Moving up   
## [103] Moving up Travelers Moving up   
## Levels: Moving up Suburb mix Travelers Urban hip

# frequencies in predicted data  
prop.table(table(seg.nb.class))

## seg.nb.class  
## Moving up Suburb mix Travelers Urban hip   
## 0.2285714 0.3047619 0.3428571 0.1238095

# plot it  
clusplot(seg.df.test[, -7], seg.nb.class, color=TRUE, shade=TRUE,   
 labels=4, lines=0,   
 main="Naive Bayes classification, holdout data")



# compare to known segments (which we can do with this test data)  
mean(seg.df.test$Segment==seg.nb.class)

## [1] 0.8

# adjusted for chance  
library(mclust)  
adjustedRandIndex(seg.nb.class, seg.df.test$Segment)

## [1] 0.5626787

table(seg.nb.class, seg.df.test$Segment)

##   
## seg.nb.class Moving up Suburb mix Travelers Urban hip  
## Moving up 13 10 0 1  
## Suburb mix 3 29 0 0  
## Travelers 5 2 29 0  
## Urban hip 0 0 0 13

# summary data for proposed segments in the test data  
seg.summ(seg.df.test, seg.nb.class)

## Group.1 age gender income kids ownHome subscribe  
## 1 Moving up 34.29258 1.125000 51369.52 2.2916667 1.416667 1.250000  
## 2 Suburb mix 41.24653 1.562500 58095.10 2.1875000 1.562500 1.000000  
## 3 Travelers 55.08669 1.444444 58634.10 0.0000000 1.666667 1.166667  
## 4 Urban hip 23.36047 1.461538 22039.69 0.8461538 1.307692 1.153846  
## Segment  
## 1 1.541667  
## 2 1.906250  
## 3 2.666667  
## 4 4.000000

# summary data for the known segments in the test data  
seg.summ(seg.df.test, seg.df.test$Segment)

## Group.1 age gender income kids ownHome subscribe  
## 1 Moving up 36.88989 1.190476 53582.16 1.4761905 1.333333 1.190476  
## 2 Suburb mix 39.61984 1.487805 56341.99 2.2439024 1.585366 1.048780  
## 3 Travelers 58.57245 1.448276 59869.24 0.0000000 1.689655 1.206897  
## 4 Urban hip 23.71537 1.428571 22700.06 0.9285714 1.357143 1.142857  
## Segment  
## 1 1  
## 2 2  
## 3 3  
## 4 4

# predict raw probabilities  
predict(seg.nb, seg.df.test, type="raw")

## Moving up Suburb mix Travelers Urban hip  
## [1,] 4.070780e-01 5.928052e-01 4.848358e-05 6.832328e-05  
## [2,] 2.715183e-04 2.422066e-03 9.973064e-01 6.143554e-32  
## [3,] 2.671393e-01 7.326897e-01 1.710510e-04 2.844967e-40  
## [4,] 2.237216e-01 7.746457e-01 1.632613e-03 7.568258e-37  
## [5,] 2.255663e-01 7.740280e-01 4.057610e-04 9.030641e-11  
## [6,] 2.051011e-01 7.948047e-01 9.422156e-05 1.554167e-33  
## [7,] 6.226983e-01 3.772938e-01 7.952188e-06 1.250892e-24  
## [8,] 1.429048e-01 8.565200e-01 5.751660e-04 2.667936e-36  
## [9,] 2.282427e-01 7.716462e-01 1.111296e-04 6.476270e-22  
## [10,] 1.881606e-01 8.114632e-01 3.762108e-04 2.155829e-33  
## [11,] 1.409837e-03 3.908209e-03 9.946820e-01 4.248292e-33  
## [12,] 8.812766e-01 1.187179e-01 5.530903e-06 1.315135e-13  
## [13,] 9.363446e-01 6.188859e-02 1.766807e-03 2.521926e-19  
## [14,] 5.935200e-01 4.064759e-01 2.932670e-06 1.210546e-06  
## [15,] 4.725188e-01 5.274249e-01 5.631135e-05 5.870878e-25  
## [16,] 7.963638e-01 2.035722e-01 6.404062e-05 1.030205e-20  
## [17,] 6.477085e-01 3.522871e-01 4.456907e-06 9.790789e-19  
## [18,] 1.715141e-03 9.840527e-01 1.423214e-02 2.993688e-73  
## [19,] 1.179409e-01 8.812472e-01 8.119052e-04 1.001008e-39  
## [20,] 2.893329e-01 7.106172e-01 4.987617e-05 3.384453e-30  
## [21,] 5.585618e-01 4.413705e-01 6.533428e-05 2.374933e-06  
## [22,] 3.337822e-01 6.660742e-01 1.436054e-04 9.311995e-32  
## [23,] 8.249975e-02 9.172824e-01 2.178072e-04 3.649129e-44  
## [24,] 5.008053e-01 4.991899e-01 4.757632e-06 1.371772e-15  
## [25,] 4.442623e-01 5.555204e-01 2.172838e-04 3.746701e-33  
## [26,] 3.586398e-01 6.411569e-01 2.032695e-04 4.374358e-34  
## [27,] 9.864036e-01 1.331108e-02 2.852760e-04 7.920841e-20  
## [28,] 4.791363e-01 5.201146e-01 7.491095e-04 6.306239e-32  
## [29,] 1.719093e-01 8.274102e-01 6.805144e-04 4.384449e-39  
## [30,] 2.792319e-01 7.207143e-01 5.377341e-05 3.945087e-32  
## [31,] 4.444809e-01 5.555000e-01 1.902410e-05 6.186405e-23  
## [32,] 4.317279e-01 5.682500e-01 2.207835e-05 4.243505e-31  
## [33,] 1.988075e-01 8.011554e-01 3.716906e-05 3.770192e-28  
## [34,] 6.826468e-01 3.172864e-01 6.680345e-05 6.263200e-15  
## [35,] 3.656168e-01 6.343525e-01 3.075380e-05 8.154636e-29  
## [36,] 1.409983e-02 9.597235e-01 2.617663e-02 1.705685e-65  
## [37,] 1.474668e-01 8.518188e-01 7.143783e-04 6.036608e-42  
## [38,] 4.107278e-01 5.892610e-01 1.121548e-05 8.009665e-12  
## [39,] 1.097242e-01 8.895641e-01 7.116990e-04 3.640361e-41  
## [40,] 1.584013e-01 7.947219e-01 4.687685e-02 4.328512e-50  
## [41,] 3.505440e-02 9.606926e-01 4.252956e-03 6.290196e-55  
## [42,] 1.122555e-05 2.586008e-05 2.070450e-09 9.999629e-01  
## [43,] 2.708565e-06 5.002437e-06 8.813820e-09 9.999923e-01  
## [44,] 2.012864e-06 6.344218e-06 8.237494e-04 9.991679e-01  
## [45,] 2.618790e-05 5.683791e-05 6.829390e-08 9.999169e-01  
## [46,] 5.290098e-05 8.357094e-05 4.181535e-02 9.580482e-01  
## [47,] 5.528166e-06 1.352741e-05 1.234177e-03 9.987468e-01  
## [48,] 1.505269e-04 1.307232e-04 1.121059e-08 9.997187e-01  
## [49,] 9.840399e-05 1.124898e-04 7.933819e-09 9.997891e-01  
## [50,] 4.141680e-05 2.133260e-05 1.691168e-09 9.999372e-01  
## [51,] 4.846737e-06 9.321986e-06 3.083240e-09 9.999858e-01  
## [52,] 5.073755e-01 3.176407e-01 3.414710e-05 1.749497e-01  
## [53,] 6.285033e-06 1.992342e-05 4.122520e-09 9.999738e-01  
## [54,] 5.617880e-04 6.110798e-04 1.523376e-01 8.464896e-01  
## [55,] 1.585677e-04 1.260877e-04 5.471531e-03 9.942438e-01  
## [56,] 3.848736e-12 4.426424e-08 1.000000e+00 2.158109e-114  
## [57,] 7.589320e-16 2.151421e-09 1.000000e+00 6.591408e-136  
## [58,] 2.180764e-03 1.399840e-02 9.838208e-01 1.973439e-34  
## [59,] 1.453951e-04 1.172650e-03 9.986820e-01 1.997596e-41  
## [60,] 5.165052e-13 5.590014e-08 9.999999e-01 9.545166e-118  
## [61,] 1.036906e-17 4.096307e-10 1.000000e+00 7.010506e-154  
## [62,] 3.486228e-07 7.350485e-05 9.999261e-01 1.723402e-66  
## [63,] 1.779764e-12 7.979504e-08 9.999999e-01 1.151915e-108  
## [64,] 6.556751e-10 1.532494e-06 9.999985e-01 6.536369e-81  
## [65,] 8.515440e-09 6.959245e-06 9.999930e-01 2.837008e-81  
## [66,] 2.052778e-15 3.492683e-10 1.000000e+00 2.393598e-138  
## [67,] 5.688381e-05 1.289988e-03 9.986531e-01 1.425667e-55  
## [68,] 1.838440e-15 4.026445e-09 1.000000e+00 3.379407e-140  
## [69,] 1.798857e-06 8.038892e-05 9.999178e-01 5.958004e-55  
## [70,] 2.076097e-06 1.326329e-04 9.998653e-01 6.508037e-62  
## [71,] 4.656512e-11 2.093391e-07 9.999998e-01 1.881985e-106  
## [72,] 3.626703e-14 3.847436e-09 1.000000e+00 5.646379e-97  
## [73,] 1.605414e-11 7.846866e-08 9.999999e-01 2.853606e-113  
## [74,] 4.409435e-16 1.604309e-09 1.000000e+00 2.010348e-151  
## [75,] 5.821964e-19 1.476104e-11 1.000000e+00 1.988381e-169  
## [76,] 1.681580e-11 3.428651e-07 9.999997e-01 1.122044e-90  
## [77,] 1.204774e-06 1.381948e-04 9.998606e-01 1.868316e-73  
## [78,] 8.299835e-13 3.707382e-08 1.000000e+00 4.592253e-99  
## [79,] 5.965582e-26 3.845277e-15 1.000000e+00 1.662621e-199  
## [80,] 6.230834e-05 1.976800e-03 9.979609e-01 1.249634e-53  
## [81,] 2.172137e-09 2.501661e-06 9.999975e-01 1.545993e-83  
## [82,] 2.948970e-18 8.801701e-12 1.000000e+00 1.907936e-155  
## [83,] 4.223935e-14 1.605263e-09 1.000000e+00 7.259006e-127  
## [84,] 2.087288e-14 2.251596e-08 1.000000e+00 1.373288e-130  
## [85,] 2.799951e-01 7.195106e-01 4.942965e-04 8.053514e-37  
## [86,] 6.455032e-01 3.544805e-01 1.631816e-05 3.333431e-12  
## [87,] 8.770655e-01 1.229329e-01 1.643295e-06 1.032652e-17  
## [88,] 9.124913e-02 6.023881e-02 8.485121e-01 3.476815e-21  
## [89,] 8.125672e-01 1.874132e-01 1.963252e-05 2.173069e-23  
## [90,] 1.880513e-03 7.330164e-03 9.907893e-01 2.953075e-43  
## [91,] 4.978976e-01 5.020053e-01 9.703109e-05 2.455250e-31  
## [92,] 6.512389e-01 3.487564e-01 4.661974e-06 5.535521e-20  
## [93,] 4.071302e-01 5.928361e-01 3.377476e-05 3.364233e-12  
## [94,] 3.936946e-02 5.920812e-02 9.014224e-01 5.525953e-24  
## [95,] 7.131613e-01 2.868359e-01 2.722473e-06 2.914527e-14  
## [96,] 1.981632e-04 1.915729e-03 9.978861e-01 2.559204e-30  
## [97,] 5.536556e-01 4.463331e-01 1.132570e-05 2.656886e-17  
## [98,] 8.656092e-01 1.343837e-01 7.053256e-06 2.362430e-13  
## [99,] 8.003381e-01 1.988422e-01 8.196091e-04 6.861077e-22  
## [100,] 5.230366e-01 4.769470e-01 1.638791e-05 7.947511e-14  
## [101,] 8.540672e-01 1.459302e-01 2.573149e-06 1.778876e-15  
## [102,] 5.469917e-01 4.529987e-01 9.673663e-06 1.779996e-24  
## [103,] 5.725079e-01 4.274841e-01 8.013238e-06 3.155921e-23  
## [104,] 3.270015e-03 6.079785e-03 9.906502e-01 5.581108e-30  
## [105,] 8.618555e-01 1.381424e-01 2.119364e-06 1.540710e-17

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

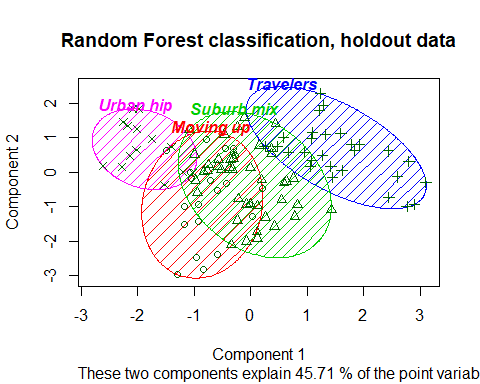
##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

set.seed(98040)  
  
(seg.rf <- randomForest(Segment ~ ., data=seg.df.train))

##   
## Call:  
## randomForest(formula = Segment ~ ., data = seg.df.train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 23.08%  
## Confusion matrix:  
## Moving up Suburb mix Travelers Urban hip class.error  
## Moving up 30 17 0 2 0.38775510  
## Suburb mix 18 36 4 1 0.38983051  
## Travelers 0 3 48 0 0.05882353  
## Urban hip 0 0 0 36 0.00000000

# predict the test data for random forest  
seg.rf.class <- predict(seg.rf, seg.df.test)  
  
# plot the solution  
library(cluster)  
  
clusplot(seg.df.test[, -7], seg.rf.class, color=TRUE, shade=TRUE,   
 labels=4, lines=0, main="Random Forest classification, holdout data")



# get the individual prediction distribution  
seg.rf.class.all <- predict(seg.rf, seg.df.test, predict.all=TRUE)  
  
# look at the distribution for the first 5 test data cases  
apply(seg.rf.class.all$individual[1:5, ], 1, table)

## $`2`  
##   
## Moving up Suburb mix Travelers Urban hip   
## 197 226 24 53   
##   
## $`3`  
##   
## Moving up Suburb mix Travelers   
## 44 245 211   
##   
## $`4`  
##   
## Moving up Suburb mix Travelers Urban hip   
## 91 350 58 1   
##   
## $`6`  
##   
## Moving up Suburb mix Travelers   
## 55 437 8   
##   
## $`7`  
##   
## Moving up Suburb mix Travelers Urban hip   
## 105 167 24 204

# summaries for the proposed and actual segments  
seg.summ(seg.df.test, seg.rf.class)

## Group.1 age gender income kids ownHome subscribe  
## 1 Moving up 34.63061 1.136364 51885.37 2.45454545 1.454545 1.272727  
## 2 Suburb mix 40.66557 1.487805 57737.61 1.63414634 1.536585 1.000000  
## 3 Travelers 59.26118 1.464286 59812.04 0.03571429 1.714286 1.214286  
## 4 Urban hip 24.37450 1.500000 21842.73 1.00000000 1.285714 1.142857  
## Segment  
## 1 1.636364  
## 2 1.829268  
## 3 2.892857  
## 4 3.857143

seg.summ(seg.df.test, seg.df.test$Segment)

## Group.1 age gender income kids ownHome subscribe  
## 1 Moving up 36.88989 1.190476 53582.16 1.4761905 1.333333 1.190476  
## 2 Suburb mix 39.61984 1.487805 56341.99 2.2439024 1.585366 1.048780  
## 3 Travelers 58.57245 1.448276 59869.24 0.0000000 1.689655 1.206897  
## 4 Urban hip 23.71537 1.428571 22700.06 0.9285714 1.357143 1.142857  
## Segment  
## 1 1  
## 2 2  
## 3 3  
## 4 4

# confusion matrix in test data  
mean(seg.df.test$Segment==seg.rf.class)

## [1] 0.7333333

table(seg.df.test$Segment, seg.rf.class)

## seg.rf.class  
## Moving up Suburb mix Travelers Urban hip  
## Moving up 10 10 1 0  
## Suburb mix 11 28 1 1  
## Travelers 0 3 26 0  
## Urban hip 1 0 0 13

library(mclust)  
adjustedRandIndex(seg.df.test$Segment, seg.rf.class)

## [1] 0.4587587

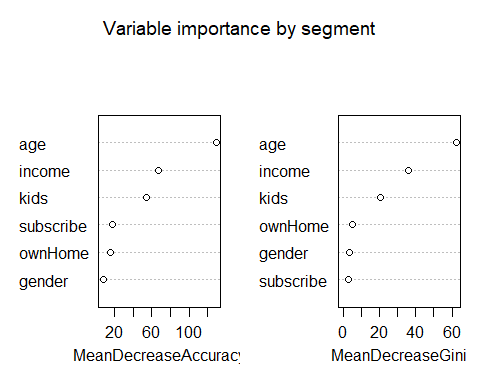
### random forest variable importance  
set.seed(98040)  
(seg.rf <- randomForest(Segment ~ ., data=seg.df.train, ntree=3000,  
 importance=TRUE))

##   
## Call:  
## randomForest(formula = Segment ~ ., data = seg.df.train, ntree = 3000, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 3000  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 23.59%  
## Confusion matrix:  
## Moving up Suburb mix Travelers Urban hip class.error  
## Moving up 29 19 0 1 0.40816327  
## Suburb mix 19 36 3 1 0.38983051  
## Travelers 0 3 48 0 0.05882353  
## Urban hip 0 0 0 36 0.00000000

importance(seg.rf)

## Moving up Suburb mix Travelers Urban hip MeanDecreaseAccuracy  
## age 59.926151 44.013275 122.6900323 86.496891 129.354271  
## gender 13.161197 -3.690088 -3.6665717 9.174039 7.757333  
## income 22.259442 17.992316 15.8721495 78.262846 67.554326  
## kids 18.263661 14.264086 55.5604028 6.410428 53.827310  
## ownHome 4.124127 -9.036638 22.6148866 19.842501 15.964989  
## subscribe 18.588573 9.460176 0.4312472 -4.130187 17.858784  
## MeanDecreaseGini  
## age 62.321942  
## gender 3.356217  
## income 36.212893  
## kids 20.634224  
## ownHome 4.941645  
## subscribe 3.010284

varImpPlot(seg.rf, main="Variable importance by segment")

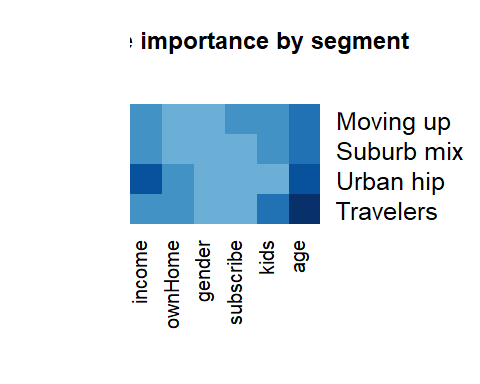


library(gplots)

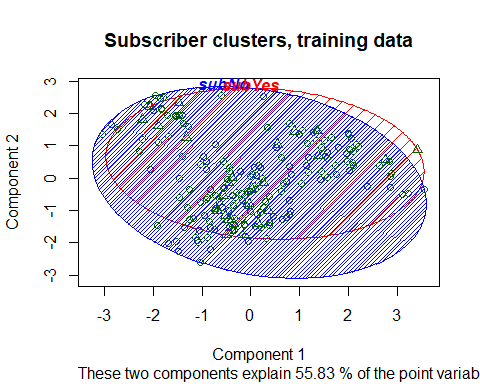
##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

library(RColorBrewer)  
  
  
heatmap.2(t(importance(seg.rf)[ , 1:4]),   
 col=brewer.pal(9, "Blues"),   
 dend="none", trace="none", key=FALSE,  
 margins=c(10, 10),  
 main="Variable importance by segment"  
 )



#### predict subscription status  
  
#### using random forest  
  
set.seed(92118)  
train.prop <- 0.65  
train.cases <- sample(nrow(seg.df), nrow(seg.df)\*train.prop)  
sub.df.train <- seg.raw[train.cases, ]  
sub.df.test <- seg.raw[-train.cases, ]  
  
  
# see how differentiated the subscribers are, in the training data  
  
clusplot(sub.df.train[, -6], sub.df.train$subscribe, color=TRUE, shade=TRUE,   
 labels=4, lines=0, main="Subscriber clusters, training data")



### Prediction: Identifying potential Customers\*

We now turn to another use for classification: to predict potential customers. An important business question-especially in high-churn categories such as mobile subscriptions-is how to reach new customers. If we have data on past prospects that includes potential predictors such as demographics, and an outcome such as purchase, we can develop a model to identify customers for whom the outcome is most likely among new prospects. In this section, we use a random forest model and attempt to predict subscription status from our data set seg.df. As usual with classification problems, we split the data into a training sample and a test sample:

library(randomForest)  
set.seed(11954)  
(sub.rf <- randomForest(subscribe ~ ., data=sub.df.train, ntree=3000))

##   
## Call:  
## randomForest(formula = subscribe ~ ., data = sub.df.train, ntree = 3000)   
## Type of random forest: classification  
## Number of trees: 3000  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 14.87%  
## Confusion matrix:  
## subNo subYes class.error  
## subNo 166 4 0.02352941  
## subYes 25 0 1.00000000

# try again with more trees, and balanced classes using sampsize  
set.seed(11954)  
(sub.rf <- randomForest(subscribe ~ ., data=sub.df.train, ntree=3000,   
 sampsize=c(25, 25)) )

##   
## Call:  
## randomForest(formula = subscribe ~ ., data = sub.df.train, ntree = 3000, sampsize = c(25, 25))   
## Type of random forest: classification  
## Number of trees: 3000  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 29.74%  
## Confusion matrix:  
## subNo subYes class.error  
## subNo 128 42 0.2470588  
## subYes 16 9 0.6400000

# predict the holdout data  
sub.rf.sub <- predict(sub.rf, sub.df.test)  
# confusion matrix  
table(sub.rf.sub, sub.df.test$subscribe)

##   
## sub.rf.sub subNo subYes  
## subNo 79 9  
## subYes 11 6

library(mclust)  
adjustedRandIndex(sub.rf.sub, sub.df.test$subscribe)

## [1] 0.1928668

library(psych)

##   
## Attaching package: 'psych'

## The following object is masked from 'package:randomForest':  
##   
## outlier

## The following object is masked from 'package:mclust':  
##   
## sim

cohen.kappa(cbind(sub.rf.sub, sub.df.test$subscribe))

## Call: cohen.kappa1(x = x, w = w, n.obs = n.obs, alpha = alpha, levels = levels)  
##   
## Cohen Kappa and Weighted Kappa correlation coefficients and confidence boundaries   
## lower estimate upper  
## unweighted kappa 0.025 0.26 0.5  
## weighted kappa 0.025 0.26 0.5  
##   
## Number of subjects = 105

With an adjusted Rand Index = 0.19 and Cohen’s kappa = 0.26 (confidence interval 0.025-0.50), the model identifies subscribers in the test data modestly better than chance.

How could we further improve prediction? We would expect to improve predictive ability if we had more data: additional observations of the subscriber group and additional predictor variables. We have described prediction using a random forest model, but there are many other approaches such as logistic regression and other machine learning algorithms

With a difficult problem-predicting a low incidence group, in data where the groups are not well-differentiated, and with a small sample-the random forest model performs modestly yet perhaps surprisingly well. There are no magic bullets in predictive modeling, but if you use the many tools available in R, avoid pitfalls such as class imbalance, and interpret results in terms of the business action, you will have good odds to achieve positive results.