Machine Learning Workshop - Unseupervised Learning

Brian Clark

Winter 2019

Contents

1 Overview

This vignette introduces some basic unsupervised learning techniques. Unsupervised learning esssentially means that you do not have a response variable or label. Generally speaking, we use unsupervised learning techniques to learn about the data. For example, how might subsets or communities of observations be best grouped, segmented, or clustered? Typical unsupervised learning techniques include clustering. Common applications arise in several areas including medial reserach, marketing, and social networking. In the finance area, we may want to segment a dataset of retail credit customers to find commoon themes. Alternatively, we may want to analyze the network underlying the financial system as a whole.

The examples in this vignette will largely follow Chapter 10 of James et al. (2013). It will not cover principal compnents analysis (PCA) becuase most standard econometric texts cover the topic so I assume that most folks in ECON are very much aware of the method. We will cover k-means clustering, which is a common technique for grouping data.

2 Clustering Methods

The goal of any clustering algorithm is to find patterns in the data. For exmaple we may want to find groups of financial institutions that behave in similar ways. This might prove useful for assessing systemic risk for example. Alternatively, we may want to detect patterns among individuals that might help us to better segment our data or understand the riskness of a portfolio.

In any clustering techniques, a few unique challenges arise that are generally not present in supervised learning models. For one, the nature of the problem is unsupervised meaning that we are not seeking to classify our data into known groups. Rather, any clustering method seeks to find groups who share similar traits, but belong to ex ante undefined groups. In other words, we are looking for patterns in the data and generally have no ex ante knowledge of the community structure of the data. This creates a few unqiue challenges:

1. How do we select the number of clusters?
2. Assuming we are able to clustour data, what do the clusters represent?

The answer to both questions is largely based on domain-specific knowledge. For example, you might have an idea of how many clusters are in the data - or at least how many clusters you want to split the data into. Some methods will have statistical criteria that suggest an "optimal" number of clusters based on some cost-complexity tradeoff (similar to what we saw for CART decision trees). However, in geenral it is up to the user to decide the appropriate number of clusters and it comes down to the specific application.

Assuming we are able to group the data into clusters, the next question is what do they mean? Remember, we don't have a set of response variables because the problem is unsupervised (if we did, we wouldn't need to cluster in the first place!). Again, the answer to his question boils down to some domain-specific knowledge. That is, once we have the clusters can we back out some information that would help us to understand our data? For example, can we group customers based on past purchase or borrowing behavior? Or can we group novel, unstrictured data into groups that human understadnable? In the end, the goal is to use the unsupervised learning to better understand our data - and it generally involves as much art as science.

2.1 k-Means Clustering

k-means clustering should not be confused with the k-nearest neighbor classification algorithm. Although they do share some common characteristics, they are very different: 1) k-means clustering is an unsupervised learning technique for segmenting or clusting data and 2) k-NN is a supervised classification algorithm.

The details of k-means clustering are explained more thoroughly in section 10.3.1 of James et al. (2013) but we will cover the basics here. First, let's define the problem as follows (again using the James et al. (2013) notation). Let $C_1, ..., C_k$ be the k sets containing the indices of the observaitons in each cluster. Our goal is to satisfy two properties:

1. Each observatiom must belong to at least one cluster.
2. Each observation be belong to only one cluster.

The underlying goal is to find a set of clusters such that the within-cluster variation is as small as possible - that is the observatinos are as similar as possible. This amounts to solving the following optimization problem:

$$\min_{C_1,...,C_k} \left[\sum_{i=1}^{k} W(C_k) \right] \tag{1}$$

where $W(C_k)$ is a measure of dis-similarity of the obervations within a cluster. We may want to define $W(C_k)$ as a measure of distance such as the squared Euclidian distance:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i',\in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2, \tag{2}$$

where $|C_k|$ is the number of observations within each cluster. Combing the above equations, we get the k-means clustering algorithm:

$$\min_{C_1,...,C_k} \left[\sum_{i=1}^{k} \frac{1}{|C_k|} \sum_{i,i',\in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right]. \tag{3}$$

This means that we want to find the set of clusters that minimize the mean differnence between observations within each group. The next step is to solve the algorithm. As it turns out, finding a global optimum is challenging, however solving for a local optimum is straight-forward. This is a common characteristic of many optimization problems. Often the most simplistic solution for the problem is to try different starting values - which is exactly what we will do for k-means clustering.

The basic algorithm as outlined on page 388 of James et al. (2013) is as follows:

1. Randomly assign a number $1, 2, ..., k$ to each of the observations which serve as the initial cluster assignments.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the k clusters, compute the *centroid*. The k^{th} cluster centroid is the vector of p feature means for the observations in the k_{th} cluster.
   (b) Assign each observation to the cluster whose centroid is closest. Define closest as in Eq. (2) - the Eucldean distance.

The above algorithm is guranteed to converge to a local optimum. See pages 388-389 in James et al. (2013) for a description of why this is so.

However, even if the above algorithm works and finds a local optimum, we still face two issues: 1) how to choose k and 2) how to determine the global optimum. Selecting k is not a straight-forward problem and generally involves some art and science (and a lot of domain-specific knowledge). The answer to the second quesiton is a bit more straight-forward, we can generally gain comfort that our solution is a "global" optimimum by trying many different starting values; just as you would do for fitting a distribution using MLE, for example.

In practice, there are a few considerations. One, the clustring methods tend to be high-varaince. As such, it is best to run them on various samples of data if possible, or use some sort of cross-validation approach. You should also be carefult not to describe the results as "truth." After all, the problem is by definition unsupervised meaning we don't know the true outcome. However, even after considering all the ppotential limitations of clustering, it can prove to be a valuable tool for exploring data and reducing dimensions (e.g., you can actually apply k-means clustering to your feature space).

2.2 Hierarchical Clustering

An alternative to k-means clustering is hierarchical clustering. Whereas k-means forces the user to pre-define k, hierarchical clustring is a bottom-up approach that sequentially clusters the most similar observations until all observations are in a single cluster. The method is described in Section 10.3.2 of James et al. (2013). We will just touch on the basics of hierarchical clustering here.

The basic hierarchical clustering algorithm is as follows. Start by defining some measure of dissimilarity such as Euclidean distance. The algorithm starts by placing all n observations in to n distinct clusters. The two most similar observations are then grouped together resulting in $n - 1$ clusters. This process is repeated sequentially until all observations are in a single cluster.

One additional detail is that we need ot extend our simple measure of dissimilarity to be able to measure distance between groups of observations. We refer to these extensions as linkages. Table 10.2 of James et al. (2013) describes four common linkages:

1. Complete: Maximimal intercluster dissimilarity. This is the largest dissimilarity (distance) between any two points in the clusters.
2. Single: Minimal intercluster dissimilarity. This is the smallest dissimilarity (distance) between any two points in the clusters.
3. Average: Mean intercluster dissimilarity. This is the mean dissimilarity (distance) between all pairwise sets of points in the clusters.
4. Centroid: Distance between the centroids of the clusters.

3 Implementatin in R

We start by writing our own k means clustering algorithm. We will use the iris data to test the function. Let's start by loading the data and retaining only three features so we can plot the data (i.e., we will have a 3-D plot).

```
rm(list=ls())
dat <- iris

# set.seed(1234)

# Only use the first three variables so we can plot
dat.dep <- iris[,c(1:3)]
```

Next, we will define $k =$ for three clusters and pre-process the data a bit.

```
k <- 3
m <- dim(dat.dep)[1]
dat.dep.clus <- cbind(dat.dep,
                      matrix(NA,ncol=1,nrow=m))
names(dat.dep.clus) <- c("v1","v2","v3","c")
```

Now we can start to build the algorithm. Start by randomly assigning a cluster (C_1, C_2, C_3) to each observation.

```
# Initialize the process (using random method):
dat.dep.clus[,k+1] <- sample.int(k,m,replace=T)
```

Next, we need a measure of dissimilarity and a function to compute the centroids since our dissimilarity measures are based on centroids. A centroid is essentially the mean value of a set of observations - think of it like the center of mass in multiple dimensions.

```
# Define centroids:
mycentroid <- function(datin,tmp.cluster){
  meanMat <- matrix(NA,ncol=k,nrow=k)
  for (i in 1:k){
    for (j in 1:k){
      meanMat[i,j] <- mean(datin[,j][tmp.cluster==i])
    }
  }
  return(meanMat)
}
```

Now we can define a function that assigns each observation to a cluster to minimize the distance.

```
# Assign each observation to a cluster that minimizes the Euc distance:
myAssign <- function(datin){
  new.cluster <- matrix(NA,nrow=m,ncol=1)
  for (i in 1:m){
    tmp <- datin[i,c(1:k)]
    tmp.c <- rep(NA,k)
    for (c in 1:k){
      tmp.c[c] <- norm(tmp - m.i[c,],type="2")
    }
    new.cluster[i] <- which.min(tmp.c)
  }
  # print(new.cluster)
```

```
  # print("------------------------")
  return(new.cluster)
}
```

The above two functions (`myAssign` and `mycentroid`) form the basis of the model. For this simplified example, we are simply going to run our algorithm for a fixed number of iterations. In practice you would run it until some stopping criteria is met - typically the point at which observations were no longer changing clusters. But for simplicity, we'll just choose a fixed number, say 20.

```
# Run the method for several iterations:
N <- 20 # number of iterations

tmp.cluster <- sample.int(k,m,replace=T)
soln <- matrix(NA,ncol=N,nrow=m)
for (iter in 1:N){
  m.i <- mycentroid(dat.dep.clus,tmp.cluster)
  tmp.cluster <- myAssign(dat.dep.clus)
  soln[,iter] <- tmp.cluster
}


myClusters <- soln[,dim(soln)[2]]
dat.plot <- cbind(dat.dep,myClusters)
```

Finally, let's plot the data. We will use `plot_ly` for this example. You can also do this in `ggplot`.

```
# Plot the results:
library("ggplot2")
library("scatterplot3d")
library("plotly")
```

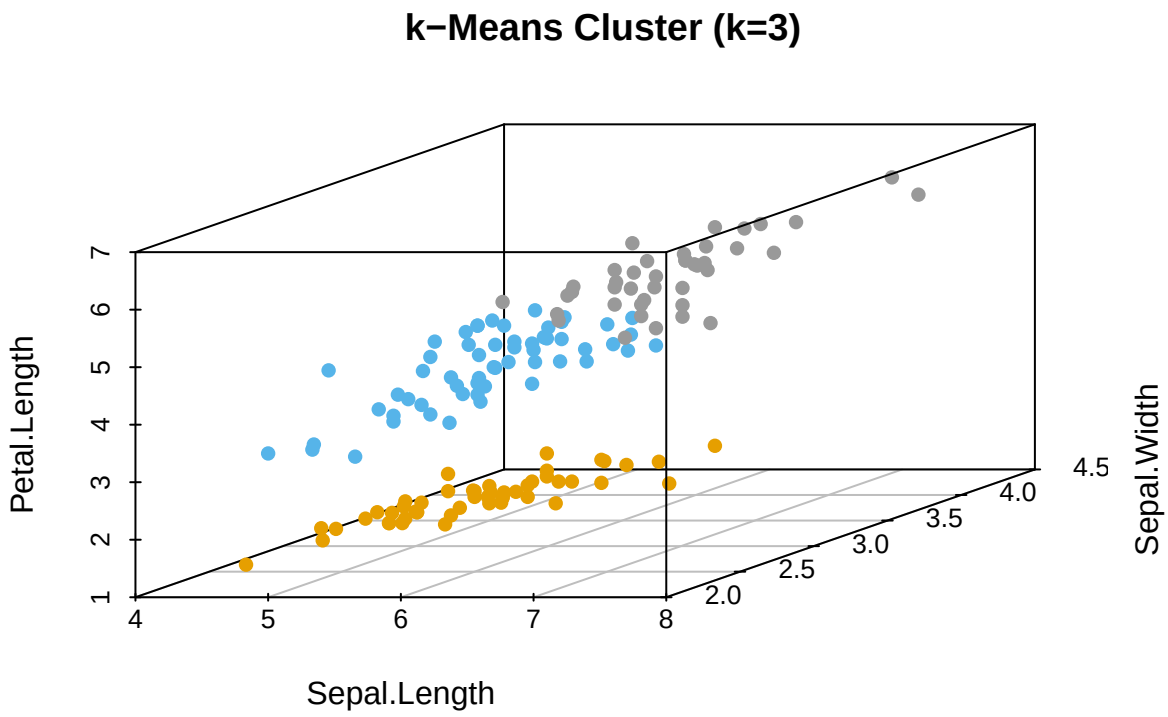
```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
colors <- c("#999999", "#E69F00", "#56B4E9")
colors <- colors[as.numeric(dat.plot[,4])]
scatterplot3d(dat.plot[,1:3], pch = 16, color=colors,
              main="k-Means Cluster (k=3)")
```




```
# Define the plotting object:
p <- plot_ly(dat.plot, x = ~Sepal.Length, y = ~Sepal.Width, z = ~Petal.Length,
             color = ~myClusters, colors = c("#999999", "#E69F00", "#56B4E9")) %>%
  add_markers() %>%
  layout(scene = list(xaxis = list(title = 'v1'),
                      yaxis = list(title = 'v2'),
                      zaxis = list(title = 'v3')))
# Print the plot
print(p)
```

```
## Warning: `as_dictionary()` is soft-deprecated as of rlang 0.3.0.
## Please use `as_data_pronoun()` instead
## This warning is displayed once per session.
```

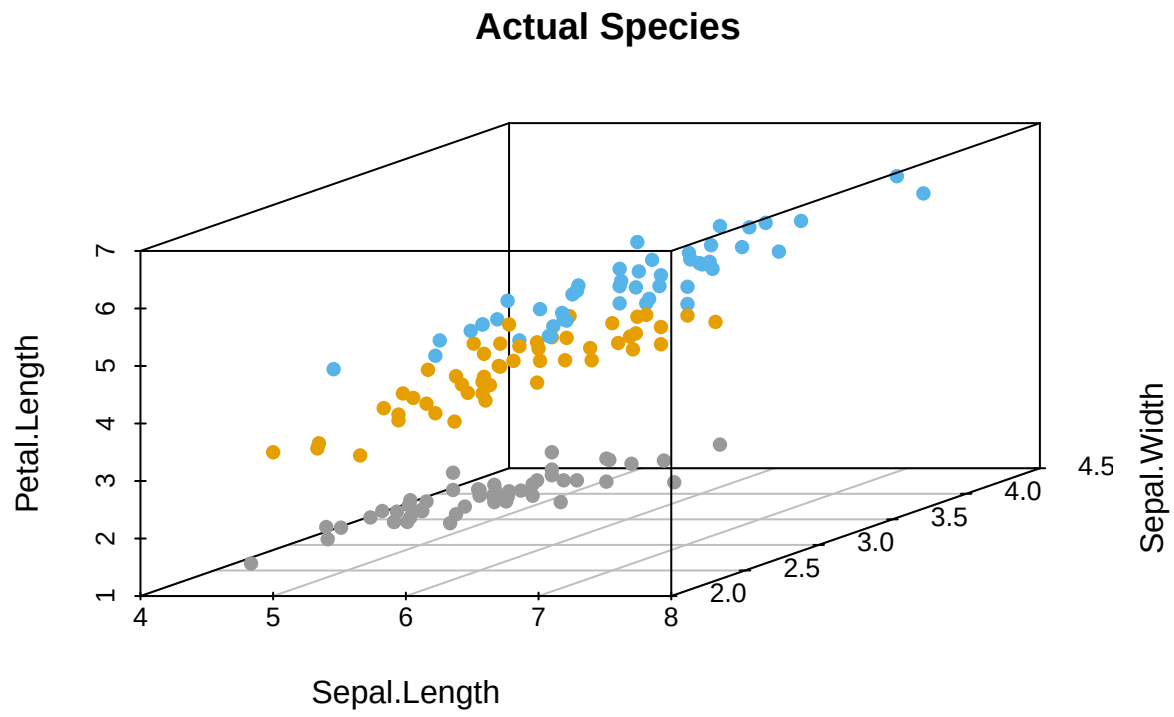
```
## Warning: `new_overscope()` is soft-deprecated as of rlang 0.2.0.
## Please use `new_data_mask()` instead
## This warning is displayed once per session.
```

```
## Warning: The `parent` argument of `new_data_mask()` is deprecated.
## The parent of the data mask is determined from either:
##
##   * The `env` argument of `eval_tidy()`
##   * Quosure environments when applicable
## This warning is displayed once per session.
```

```
## Warning: `overscope_clean()` is soft-deprecated as of rlang 0.2.0.
## This warning is displayed once per session.
```

Note that for some reason, the `plot_ly` graphic does not render to the PDF but the `scatterplot3d()` one will. The plot_ly one is interactive. We can also plot the actual species.

```
dat.plot.act <- iris[,c(1:3,5)]
colors <- c("#999999", "#E69F00", "#56B4E9")
colors <- colors[as.numeric(dat.plot.act[,4])]
scatterplot3d(dat.plot.act[,1:3], pch = 16, color=colors,
              main="Actual Species")
```



Essentially, our clustering method here reveals the three species. However, if we were to choose $k \neq 3$ the results would obviously not be as good.

3.1 R's Built in K Means Function

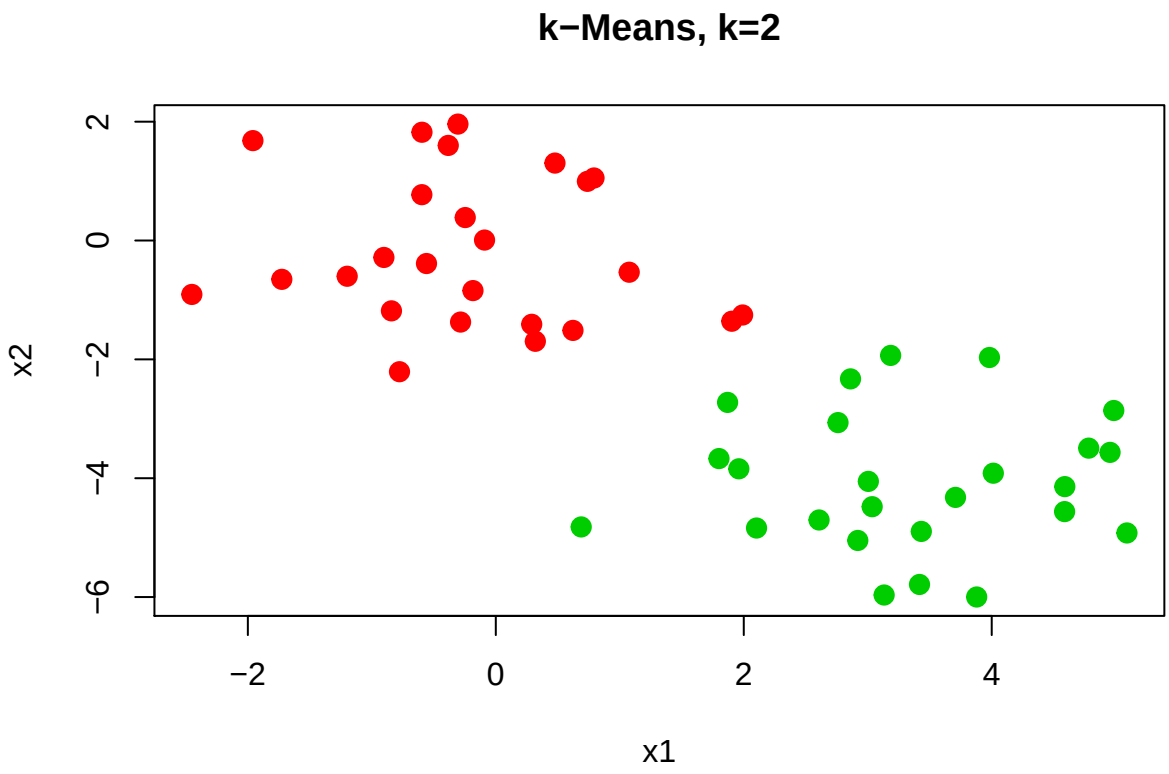
This is the example from Lab 10 in James et al. (2013).

```
set.seed(2)
x<-matrix(rnorm(100),ncol=2)
x[1:25,1] <- x[1:25,1] + 3
x[1:25,2] <- x[1:25,2] - 4

km.out <- kmeans(x,2,nstart=20)
names(km.out)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
# Plot the results:
plot(x, col=(km.out$cluster+1), main="k-Means, k=2",
     xlab="x1",ylab="x2",pch=20,cex=2)
```

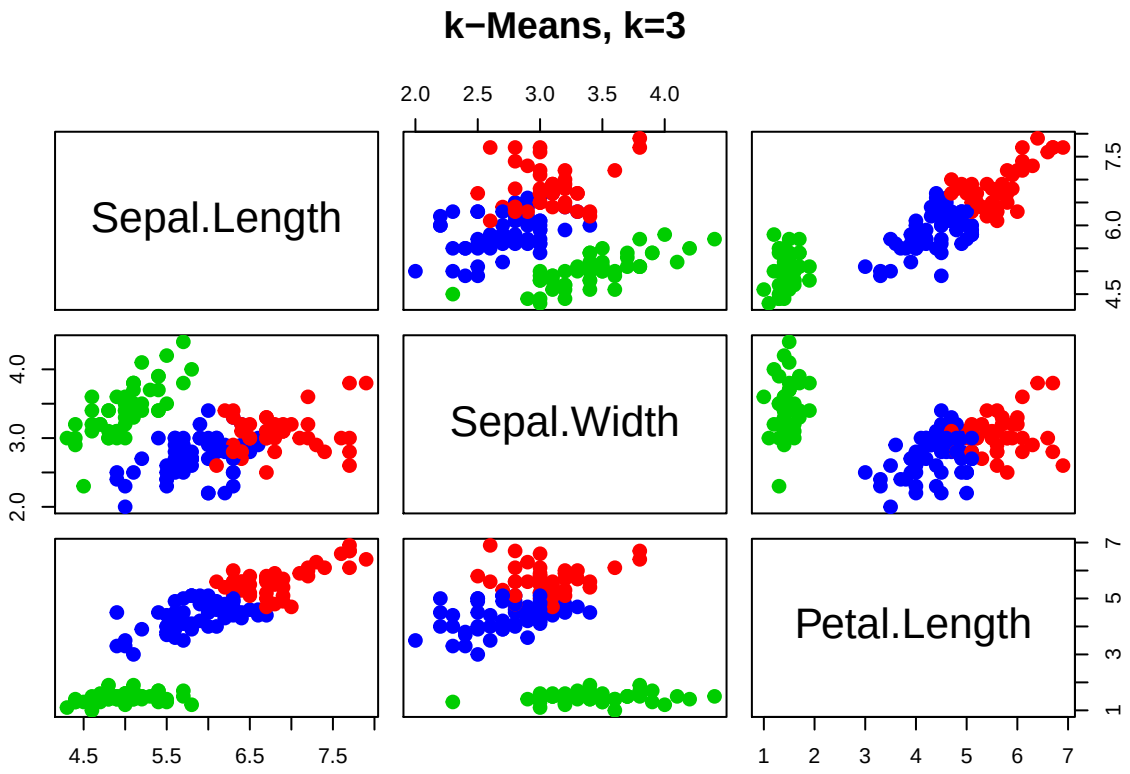


We could also try the same for the iris data.

```
km.out <- kmeans(dat.dep,3,nstart=20)
names(km.out)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
# Plot the results:
plot(dat.dep, col=(km.out$cluster+1), main="k-Means, k=3",
     pch=20,cex=2)
```


k−Means, k=3
Sepal.Length
Sepal.Width
Petal.Length
2.0
2.5
3.0
3.5
4.0
4.5
6.0
7.5
2.0
3.0
4.0
4.5
5.5
6.5
7.5
1
2
3
4
5
6
7
1
3
5
7

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning.* Vol. 112. Springer.