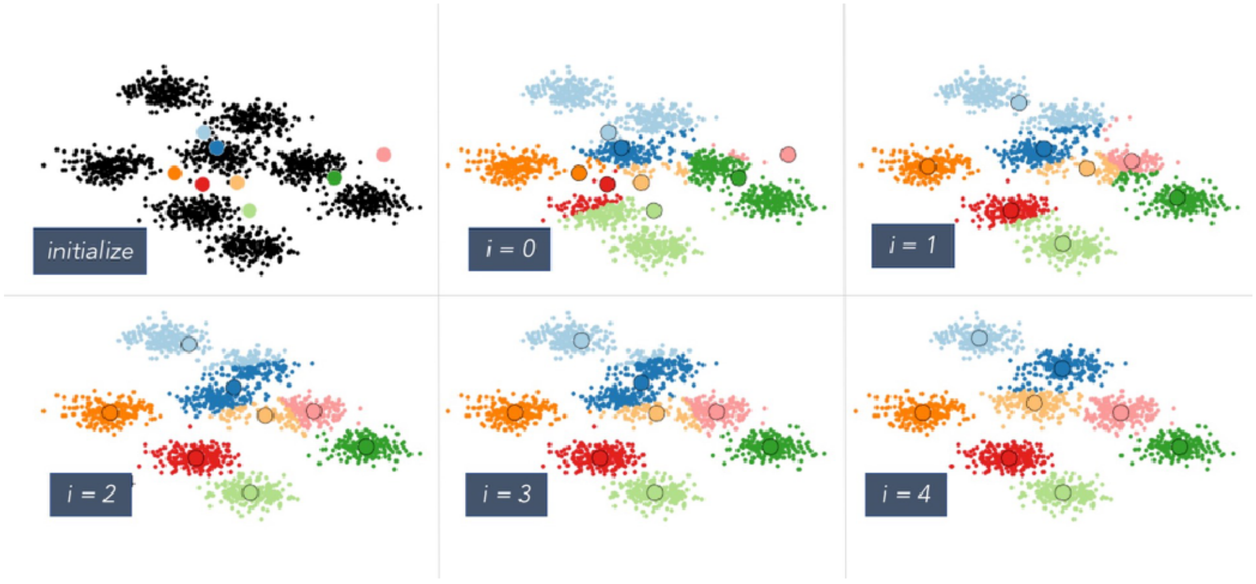

Figure 1: Illustration of k-means algorithm from initialization to convergence

K-Means

The *k-means* clustering is a technique to identify clusters of observations by treating features as coordinates in n-dimensional space. The k in k-means is specified by the analyst – it is the number of clusters that will be returned upon running the algorithm. k is not a known quantity and will need to be optimized by the analyst.

The technique is fairly straight forward to optimize and is one that is iterative as shown in the pseudocode below:

```
Initialize k centroids
Repeat following until convergence:
  Calculate distance between each record n and centroid k
  Assign points to nearest centroid
  Update centroid coordinates as average of each feature per cluster
```

The first step involves selecting k-number of random centroids from the feature space and giving each centroid a label. For each observation in the data, calculate the Euclidean distance $(d(x_1, x_2) = \sqrt{\sum_{i=1}^{n}|z_1 - z_2|^2})$ to all initial centroids, then assign each point to the closest centroid. This is known as the *assignment* step – all points take the label of its closest centroid. It is unlikely that this initial assignment is likely suboptimal, thus the algorithm will *update* the centroid coordinates by calculating the mean value of each feature within each cluster. Upon doing so, this assignment-update procedure is iteratively repeated until the centroid coordinates no longer change between iterations (see illustration below).

Central to algorithm is goal to find some set of k coordinates that minimize the within-cluster sum of squares (WSS):

$$argmin \sum_{j=1}^{k}\sum_{i=1}^{n}||x_{i,j} - \mu_j||^2$$

where the sum of the distance x of each point i in cluster j to its corresponding centroid of j. Distance is calculated in terms of all input features x and the j^{th} cluster centroid μ.

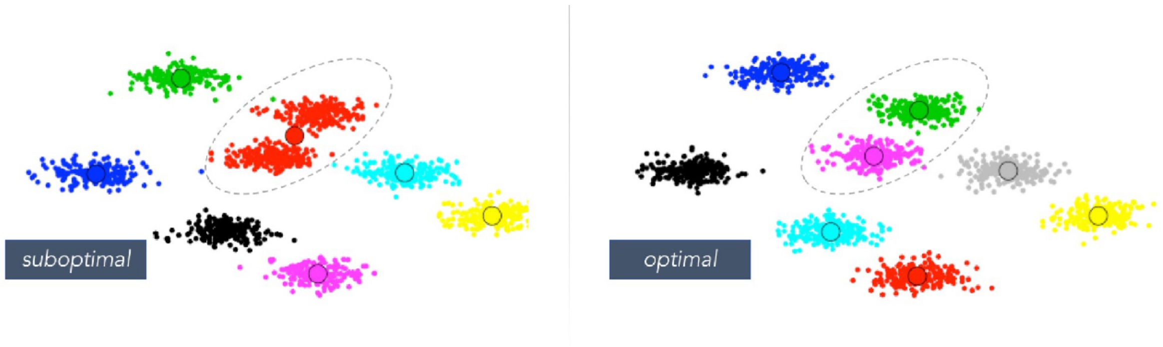

Figure 2: Comparison of a suboptimal result and optimal result

Assumptions

While k-means is a simple algorithm, its performance and effectiveness is guided by a number of key assumptions at each step of computation.

- *Scale.* As k-means treats features as coordinates, each feature is assumed to have equal importance, which in turn means that results may be inadvertently biased simply by the scale and variances of underlying features. To remedy this problem, input features should be mean-centered standardized ($\frac{x_i - \mu}{\sigma}$) or otherwise transformed to reduce scaling effects. Note, however, that the influence of scaling may not always be removed. For example, a data set containing both continuous and binary features would likely perform quite poorly as Euclidean distances are not well-suited for binary. Thus, where possible, apply k-means when the formats are homogeneous, doing so using Euclidean L2-distances for continuous and binary distances for matrices of discrete features.
- *Missing Values.* K-Means do not handle missing values as each data point is essentially a coordinate. Thus, often times k-means models are usually reserved for complete data sets.
- *Stability of Clusters.* The initialization step of the algorithm chooses k initial centroids at random. The initial random selection is known to lead to suboptimal and unstable clusters. The instability in the results can be observed when running the algorithm for some value of k multiple times, sometimes leading to different cluster composition: holding k constant between model runs, a record $i = 1$ may be in the same cluster as $i = 10, 23, 40$ in one set of results, but only with $i = 23$ in another model run. The stability of clusters may be due to a number of things, such as a suboptimal choice of k, a high number features that add noise to the optimization process, among others.
- *Choice of K.* Selecting the best value of k is arguably a subjective affair: there is a lack of consensus regarding how to identify k. One method known as the *Elbow method* chooses k at the inflection point where an additional cluster does not significantly reduce the variance explained or reduction of error. The simplest method of identifying the inflection point can be seen by plotting the percent WSS over all values of k that were tested. This approach is deceptively simple as the inflection point might not manifest itself in some data sets.

An alternative, but far more computationally intensive approach involves calculating the *silhouette*, which is compares estimates the similarity of a given observation i as compared to observations within and outside the cluster. The silhouette $s(i)$ is defined as:

$$s(i) = \frac{b_i - a_i}{max(a_i, b_i)}$$

where a_i is the Euclidean distance between a point i to other points in the same cluster, b_i is the minimum distance between i and any other cluster the sample. The values of $s(i)$ fall between -1 and 1, where 1 indicates that an observation is well-matched with its cluster and -1 indicates that fewer or more clusters

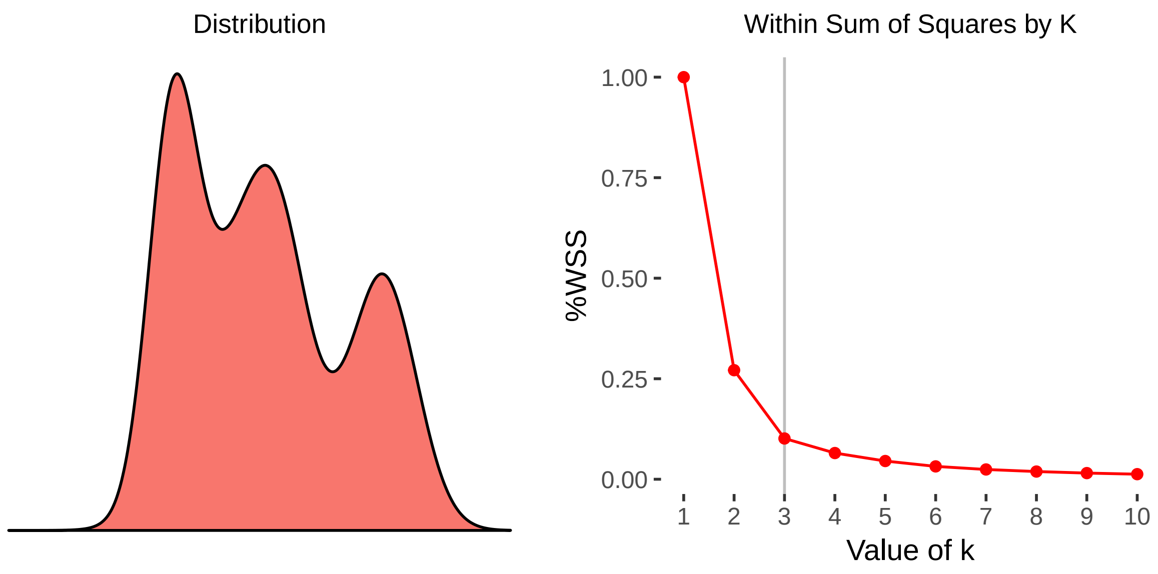

Figure 3: Elbow method: Choose k at the inflection point

may be required to achieve a better match. Note that silhouettes do not scale well with very large data sets as a $n \times n$ similarity matrix (e.g. distance between all points to all points). Often times, a smaller sample should be used to enable the use of this method.

For a step-by-step walkthrough of the application of the k-means algorithm, see *How much of the ground is covered in [vegetation/buildings/economic activity]?* in the DIY section of this chapter.

DIY: How much of the ground is covered in [vegetation/buildings/economic activity]?

Satellite imagery is an increasingly available data resource that provides full coverage of the Earth's surface. Free satellite imagery from Aqua and Terra satellites (NASA), Suomi NPP (NOAA/NASA), and Landsat Operational Land Imager (NASA/USGS), among others help earth scientists and remote sensing researchers better understand our blue marble. Publicly available imagery tends to be relatively coarse, designed for environmental and climatological inferences. Private firms such as Digital Globe and Planet operate high resolution satellites that support commercial and intelligence purposes.

Other than making for a pretty picture, what can we do with satellite imagery? Suppose the following question were asked:

> How much of a piece of land is covered in healthy vegetation?

Satellite imagery can easily be used to support this task. Let's take an example of crop fields in Kansas as seen from imagery captured by the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) instrument on the Terra satellite. The image below captures a 37.2-km x 38.8-km area where green areas indicate healthy vegetation and other colors may indicate other types of activity.[1] While our eyes are able to pick out the colors, summarizing the vegetation patterns in a scalable manner may be best left to a clustering algorithm. We can apply k-means clustering to conduct *color quantization* in which we reduce the number of colors into fewer distinct colors that summarize patterns in the data.

```r
#Library
  library(raster)
  library(digIt)
```

[1]https://www.nasa.gov/topics/earth/earthmonth/earthmonth_2013_01.html

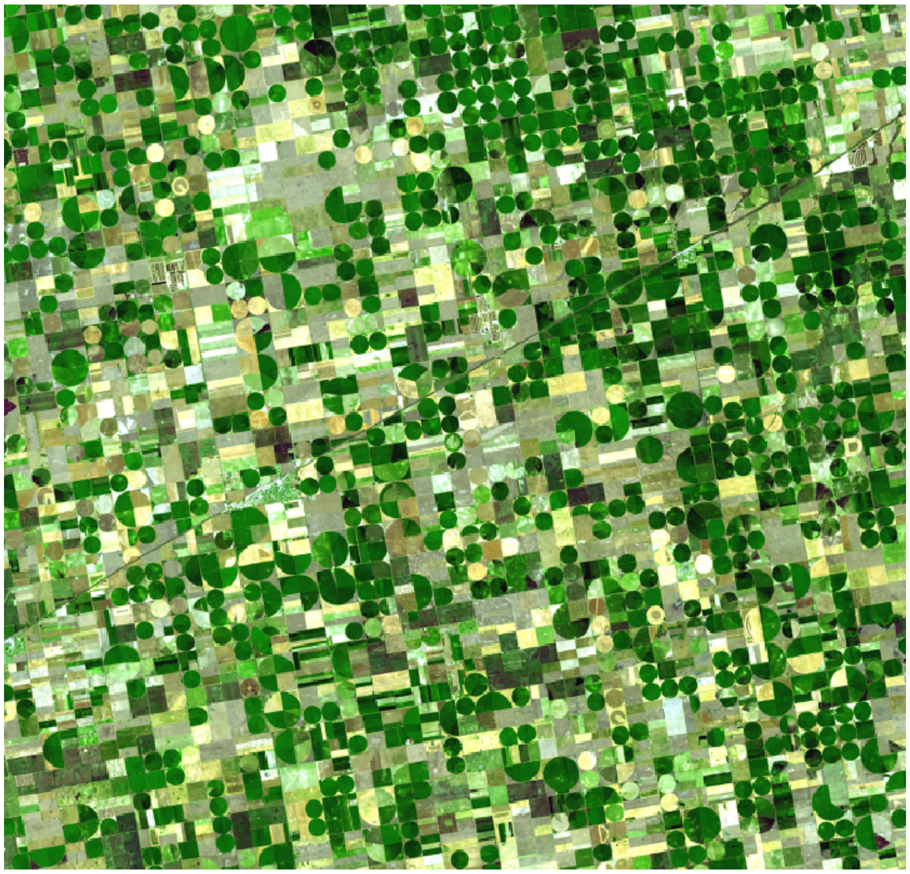

Figure 4: Crops in Finney County KS. Via NASA/GSFC/METI/Japan Space Systems, and U.S./Japan ASTER Science Team

```r
  img <- digIt("color_segment_kansas")
  plotRGB(img)
```

Before we do so, it is worth describing how imagery can be used as data. Digital photographs are comprised of a three-dimensional array that essentially resembles three matrices sandwiched together. Each matrix is an $n \times m$ matrix for each red-green-blue (RGB). The goal is to cluster on the colors, which requires the each of the $n \times m$ matrices to be transformed into a two dimensional matrix with 3 columns (one for each color) and of length nm. K-means is applied to this matrix to obtain groups of similar colors.

Getting Started

In the wild, the `digIt()` function is not available. An image would normally need to be downloaded and loaded as a `brick()` using the `raster` package. For simplicity, we use the `digIt` library to download and load the ASTER data.

```r
#Library
  library(raster)
  library(digIt)

  img <- digIt("color_segment_kansas")
```

The image is converted into a matrix containing three vectors of equal length: one for each RGB value.

```
#Dimensions
  dim(img)
```

```
## [1] 2481 2589    3
```

```
#Convert image into columns
  data <- cbind(as.vector(img[[1]]),
                as.vector(img[[2]]),
                as.vector(img[[3]]))
```

With the data in the right shape, k-means can be applied. In this example, we use the `kmeans()` function that is built into R:

`kmeans(x, k)`

where:

- `x` is a data frame or matrix of numerical values
- `k` is the number of clusters

The result of the `kmeans()` function contains a number of attributes such as the cluster assignment of each observation. To evaluate the fitness of the cluster, a silhouette statistic can be calculated using the `silhouette()` function in the `cluster` library:

`silhouette(cluster, distance)`

where:

- `cluster` is the cluster assignment.
- `distance` is a dissimilarity matrix of input features produced by `dist()`.

Given the size of the input matrix ($n = 2481 \times 2589 = 6423309$), the dissimilarity matrix is produced on a sample of $n = 20000$.

Below, k-means is tested for values of $k = 2$ to $k = 10$ using a random sample of $n = 20000$. Before the loop, the sample is taken, then the dissimilarity matrix is calculated using the `dist()` function. Within the loop, k-means results are assigned to the object `res` from which the cluster assignments are extracted. The silhouette is then calculated and assigned to the `sil` object, from which the mean silhouette is estimated from observation level silhouettes (third column).

```
#Load cluster library
  library(cluster)

#Calculate distance object using sample of n = 10000
  set.seed(10)
  subdata <- data[sample(data, 20000),]
  d <- dist(subdata)

#Set up placeholder for silhouette values
  sil.out <- data.frame()

#Loop through values of k
  for(k in 2:10){

    set.seed(20)

    #Run k-means, save to o
    res <- kmeans(subdata, k)

    #Get silhouette
```

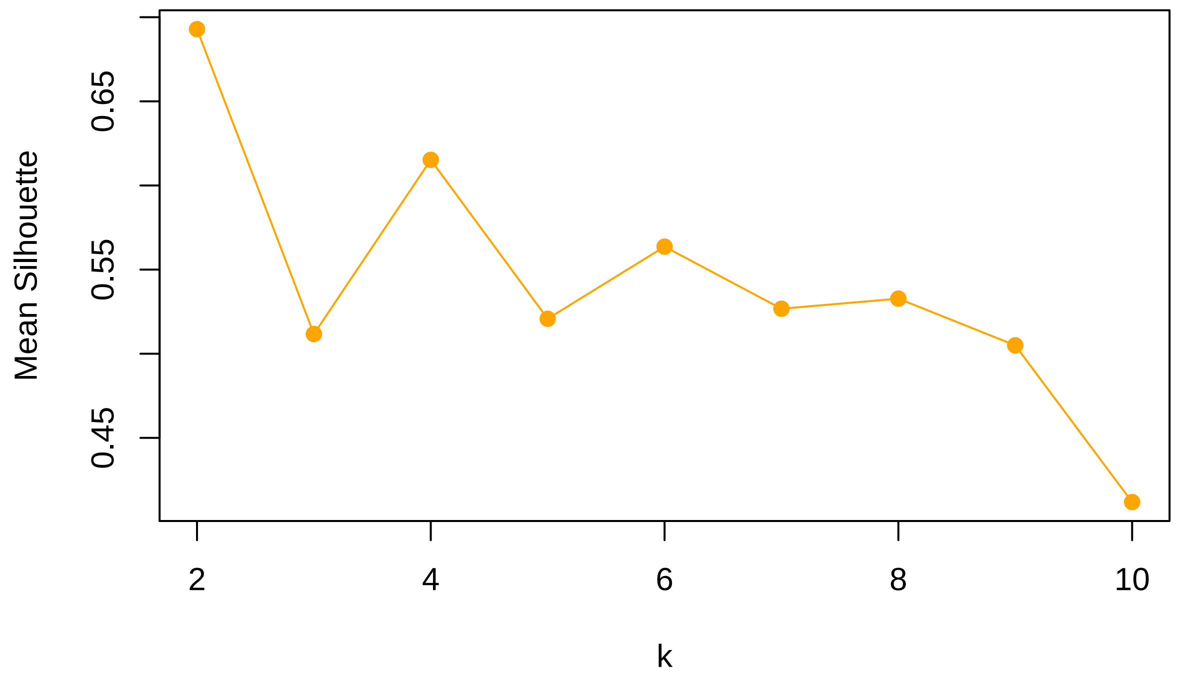

Figure 5: Mean silhouette by k

```
    sil <- silhouette(res$cluster, d)

    #Get summary values of silhouette
    temp <- data.frame(k.level = k,
                       avg = mean(sil[,3]))
    sil.out <- rbind(sil.out, temp)
  }
```

In color quantization exercises, lower values of k should be used. In the case below, the grid search suggests that $k = 2$ provides the most favorable cluster results.

```
#Plot result
  plot(sil.out[, c("k.level", "avg")], type = "l", col = "orange",
       ylab = "Mean Silhouette", xlab = "k")
  points(sil.out[, c("k.level", "avg")], pch = 19, col = "orange")
```

The k-means model is then estimated on the entire data set for $k = 2$. To visually check our results, we need to convert the vector of cluster assignments to a matrix with the same dimensions as the original image `img`. The matrix contains all the same information as an image, but is not in the right data class. Using `raster()`, the matrix can be converted into an raster image format containing cluster assignments.

```
#K values
  set.seed(123)
  res <- kmeans(data, 2)

#Convert cluster labels into matrix
  mat <- matrix(res$cluster,
                ncol = ncol(img),
                nrow = nrow(img),
                byrow = TRUE)
  img2 <- raster(mat)
```

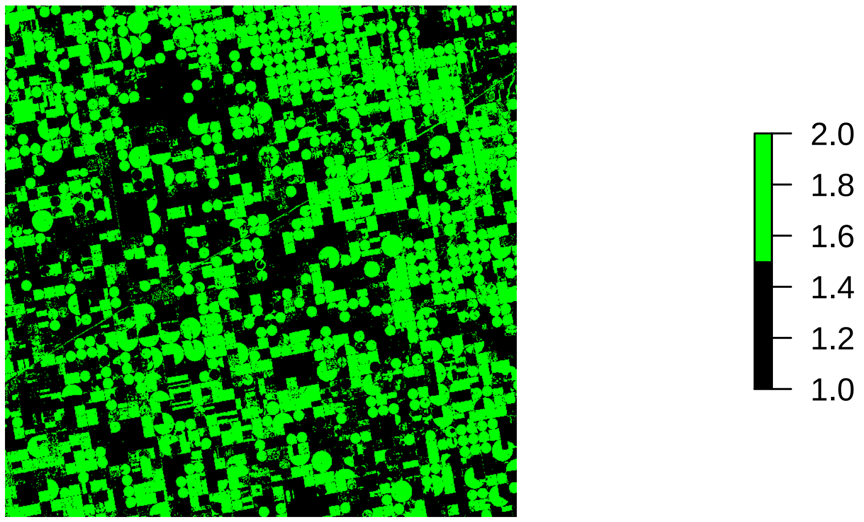

Figure 6: Color quantization for k = 3

With the data in the right form, the cluster assignments are rendered as an image. Notice that the healthy green areas are coded in green, which corresponds with cluster #2.

```
plot(img2, box=FALSE, yaxt = "n",  xaxt = "n",
     frame.plot = FALSE, col = c("black", "green"))
```

To calculate the proportion of the land that is covered in healthy vegetation as well as approximate land area, we can use the following calculation:

```
prop <- mean(mat == 2)
print(paste0("%Area = ", prop))
```

```
## [1] "%Area = 0.482713349147612"
```

```
print(paste0("km2 = ", 37.2 * 38.8 * prop))
```

```
## [1] "km2 = 696.729139625697"
```