Two Simple ML Algorithms [1]

Shaobo Li

University of Kansas

[1] Partially based on Hastie, et al. (2009) ESL, and James, et al. (2013) ISLR

Clustering – an unsupervised learning method

- Goal: find subgroups of a sample observations
  - Not based on any single variable (e.g. gender, race)
  - Based on all given variables

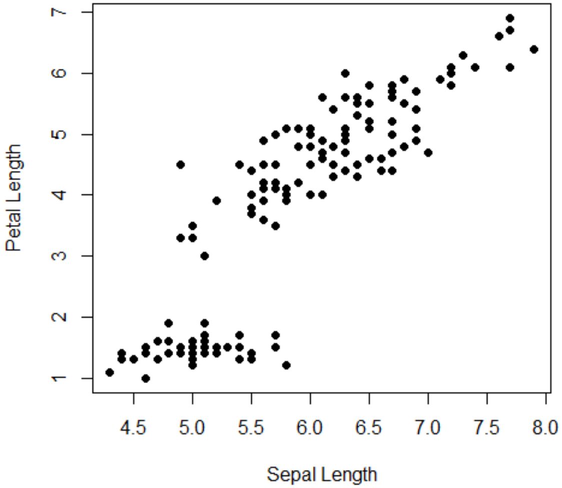
Clustering – an unsupervised learning method

- Goal: find subgroups of a sample observations
  - Not based on any single variable (e.g. gender, race)
  - Based on all given variables
- The number of subgroup is subjective

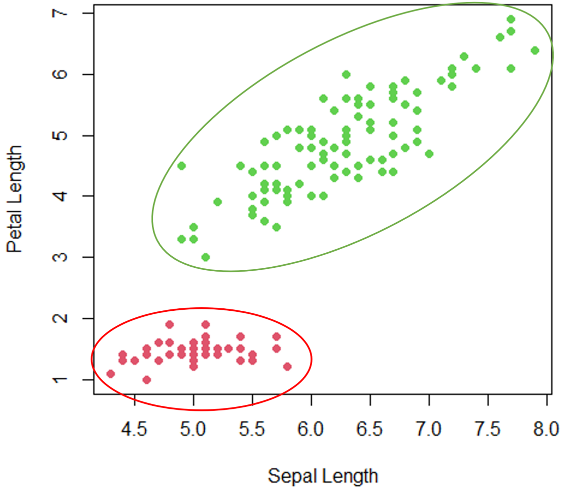
Clustering – an unsupervised learning method

- Goal: find subgroups of a sample observations
  - Not based on any single variable (e.g. gender, race)
  - Based on all given variables
- The number of subgroup is subjective
- Approaches:
  - K-means clustering
  - Hierarchical clustering
  - Model-based clustering

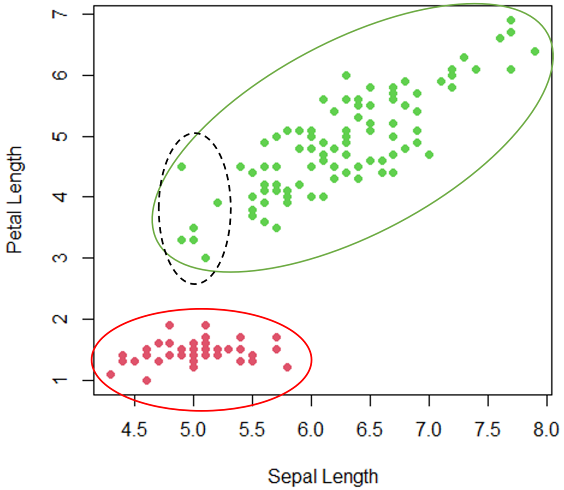
An example – Iris data



An example – Iris data



An example – Iris data

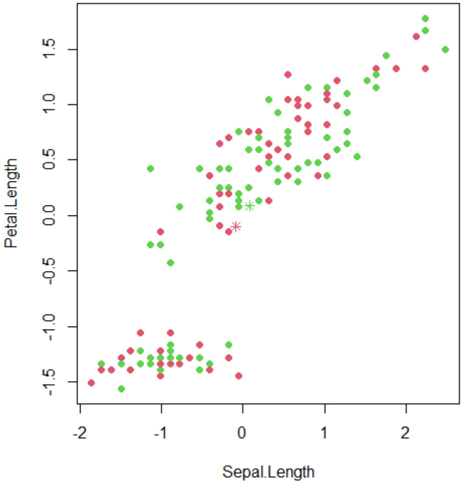


K-means clustering – step-by-step

Run the following R code, and see what it does.

```
iris1 <- scale(iris[,-c(2,4,5)])
n <- nrow(iris1)
index <- sample(2, n, replace = T)
iris.sub1 <- iris1[index==1,]
iris.sub2 <- iris1[index==2,]
mean.sub1 <- apply(iris.sub1, 2, mean)
mean.sub2 <- apply(iris.sub2, 2, mean)

plot(iris1, col=index+1, pch=16)
points(x=mean.sub1[1], y=mean.sub1[2], col=2, pch=8)
points(x=mean.sub2[1], y=mean.sub2[2], col=3, pch=8)
```

This is a random grouping (first step)

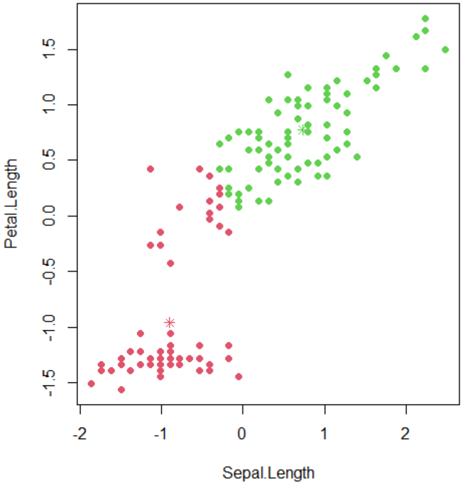


The next step

Run the following R code, and see what it does.

```
Eudist <- function(x, y) sqrt(sum((x-y)^2))

d1<-sapply(1:n,function(i) Eudist(mean.sub1,iris1[i,]))
d2<-sapply(1:n,function(i) Eudist(mean.sub2,iris1[i,]))
index.new <- apply(cbind(d1, d2), 1, which.min)
iris.sub1 <- iris1[index.new==1,]
iris.sub2 <- iris1[index.new==2,]
mean.sub1 <- apply(iris.sub1, 2, mean)
mean.sub2 <- apply(iris.sub2, 2, mean)

plot(iris1, col=index.new+1, pch=16)
points(x=mean.sub1[1], y=mean.sub1[2], col=2, pch=8)
points(x=mean.sub2[1], y=mean.sub2[2], col=3, pch=8)
```

Data points are regrouped (second step)



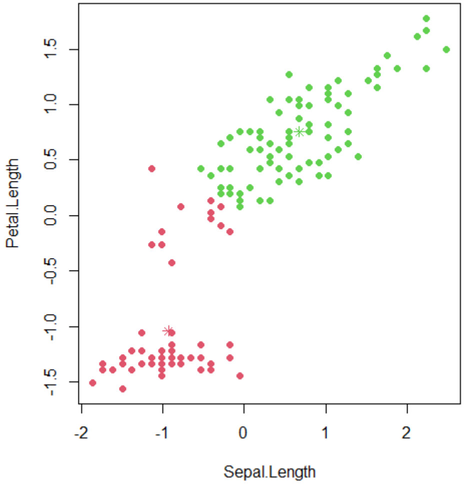
How does this happen?

Let’s repeat the second step

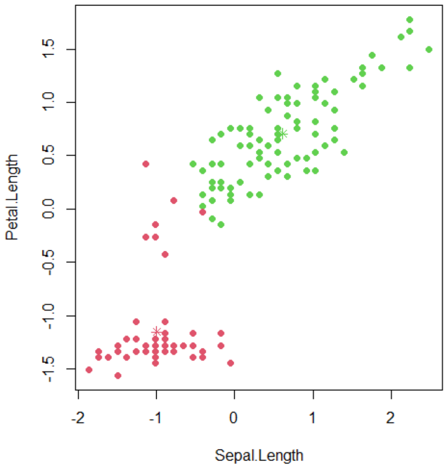
```
d1<-sapply(1:n,function(i) Eudist(mean.sub1,iris1[i,]))
d2<-sapply(1:n,function(i) Eudist(mean.sub2,iris1[i,]))
index.new <- apply(cbind(d1, d2), 1, which.min)
iris.sub1 <- iris1[index.new==1,]
iris.sub2 <- iris1[index.new==2,]
mean.sub1 <- apply(iris.sub1, 2, mean)
mean.sub2 <- apply(iris.sub2, 2, mean)

plot(iris1, col=index.new+1, pch=16)
points(x=mean.sub1[1], y=mean.sub1[2], col=2, pch=8)
points(x=mean.sub2[1], y=mean.sub2[2], col=3, pch=8)
```

Note that the code does not change at all. Why?

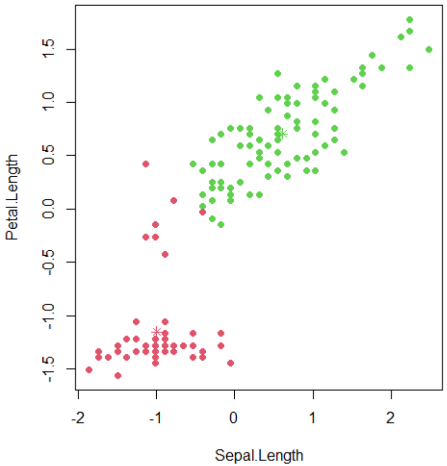
Data points are regrouped again



We can keep repeating this step, until...



We can keep repeating this step, until...



Members in each cluster do not change, which means the algorithm converges. How can we translate it into some numeric scores?

Statistics behind k-means clustering

The algorithm attempts to

- Minimize variance within clusters
- Maximize variance between clusters
- How about total variance?

Statistics behind k-means clustering

The algorithm attempts to

- Minimize variance within clusters
- Maximize variance between clusters
- How about total variance?

Instead of computing variance, we compute sum squared error (SSE).

- For a single variable X, SSE is defined as

$$SSE(X) = \sum_{i=1}^{n} (X_i - \bar{X})^2$$

Statistics behind k-means clustering

The algorithm attempts to

- Minimize variance within clusters
- Maximize variance between clusters
- How about total variance?

Instead of computing variance, we compute sum squared error (SSE).

- For a single variable X, SSE is defined as

$$SSE(X) = \sum_{i=1}^{n} (X_i - \bar{X})^2$$

- For multiple variables $\mathbf{X} = (X_1, ..., X_p)$, SSE is defined as

Statistics behind k-means clustering

The algorithm attempts to

- Minimize variance within clusters
- Maximize variance between clusters
- How about total variance?

Instead of computing variance, we compute sum squared error (SSE).

- For a single variable X, SSE is defined as

$$SSE(X) = \sum_{i=1}^{n} (X_i - \bar{X})^2$$

- For multiple variables $\mathbf{X} = (X_1, ..., X_p)$, SSE is defined as

$$SSE(\mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{p} (X_{ij} - \bar{X}_j)^2$$

Statistics behind k-means clustering

- $SSE(\mathbf{X})$ for entire sample: total sum squared error (SST).

Statistics behind k-means clustering

- $SSE(\mathbf{X})$ for entire sample: total sum squared error (SST).
- $SSE(\mathbf{X})$ for each cluster: within-group sum squared error (WSS).

Statistics behind k-means clustering

- $SSE(\mathbf{X})$ for entire sample: total sum squared error (SST).
- $SSE(\mathbf{X})$ for each cluster: within-group sum squared error (WSS).
- Sum of WSS across all clusters: total within-group sum squared error (TWSS)

Statistics behind k-means clustering

- $SSE(\mathbf{X})$ for entire sample: total sum squared error (SST).
- $SSE(\mathbf{X})$ for each cluster: within-group sum squared error (WSS).
- Sum of WSS across all clusters: total within-group sum squared error (TWSS)
- $SST - TWSS$: Between-group sum square

Statistics behind k-means clustering

- $SSE(\mathbf{X})$ for entire sample: total sum squared error (SST).
- $SSE(\mathbf{X})$ for each cluster: within-group sum squared error (WSS).
- Sum of WSS across all clusters: total within-group sum squared error (TWSS)
- $SST - TWSS$: Between-group sum square
- Exercise:

  Revisit the algorithm we just performed. Compute the above three measures at the end of each step. How are they changing over iterations?

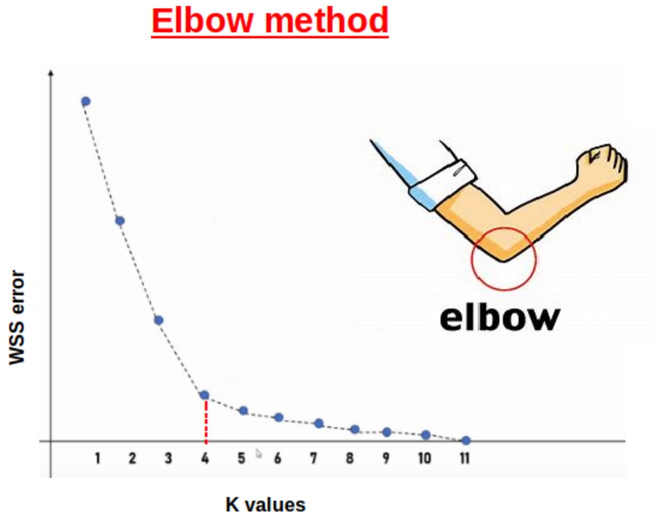
Choosing K

- Balancing the tradeoff between model complexity versus goodness of fit
  - Larger $K \implies$ higher complexity (poor interpretability) and better fit (maybe overfitting)
  - Smaller $K \implies$ lower complexity (better interpretability) and less precise
- A common approach to find optimal K: Elbow Method

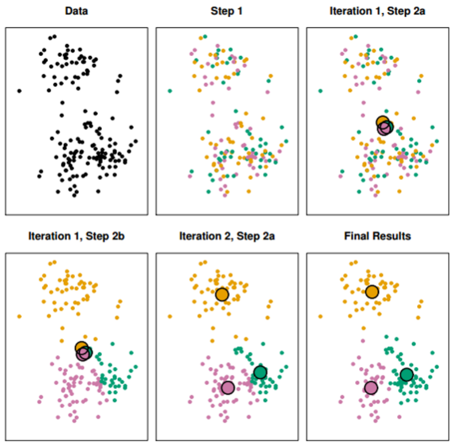
Choosing K

- Balancing the tradeoff between model complexity versus goodness of fit
  - Larger $K \implies$ higher complexity (poor interpretability) and better fit (maybe overfitting)
  - Smaller $K \implies$ lower complexity (better interpretability) and less precise
- A common approach to find optimal K: Elbow Method
  Plot TWSS vs. K and look for the "elbow" where additional clusters provide diminishing returns in variance reduction.

Elbow method



K-means algorithm recap



- Here is a very good animation to illustrate k-means clustering algorithm. [link]

K-means algorithm recap

1. Randomly find k data points (observations) as the initial centers
2. For each data point, find the closest center and label it (e.g., using different colors). Now you have k clusters
3. Re-calculate the centers of current clusters
4. Repeat step 2 and 3 until the centers do not change

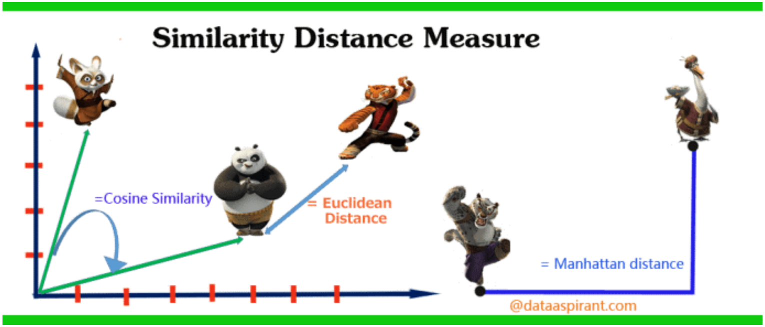
K-nearest neighbor (KNN)

- Model:

$$\hat{Y} = \hat{f}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i$$

  where $N_k(\mathbf{x})$ is the neighborhood of $\mathbf{x}$ defined by the k closest points $\mathbf{x}_i$.
- The prediction $\hat{Y}$ is typically the majority vote (or average if Y is numeric) of the outcomes in the neighbor.
- k: size of the neighbor
  - smaller or bigger neighbor is better?
- The key computation: find the neighbor of a given observation $\mathbf{x}$

Distance-based Similarity Measures



KNN algorithm illustration

Data preparation

```
iris.X <- scale(iris[,-5])
iris.Y <- iris[,5]
iris.train.X <- iris.X[1:148,]
iris.train.Y <- iris.Y[1:148]
iris.test.X <- iris.X[149:150,]
iris.test.Y <- iris.Y[149:150]
```

KNN algorithm illustration

Key component – computing distance

```
Eudist <- function(x, y){
  sqrt (sum((x-y)^2))
}

dist.vec <- rep(NA, nrow(iris.train.X))
for(i in 1:nrow(iris.train.X)){
  dist.vec[i] <- Eudist(iris.test.X[1,],
                        iris.train.X[i,])
}
```

KNN algorithm illustration

Find neighbor and make prediction

```
K <- 10
index.neighbor <- order(dist.vec)[1:K]
Y.neighbor <- iris.train.Y[index.neighbor]
freqY.neighbor <- table(Y.neighbor)
sort(freqY.neighbor, decreasing = T)
names(sort(freqY.neighbor, decreasing = T))[1]
```

KNN algorithm illustration

Find neighbor and make prediction

```
K <- 10
index.neighbor <- order(dist.vec)[1:K]
Y.neighbor <- iris.train.Y[index.neighbor]
freqY.neighbor <- table(Y.neighbor)
sort(freqY.neighbor, decreasing = T)
names(sort(freqY.neighbor, decreasing = T))[1]
```

Can you define a function, let's call it `majority_vote()`, to integrate the majority vote step? Then you can call the function to get the output in one step. Like the following:

```
majority_vote(Y.neighbor)
```

What is the impact of k – an example[2]

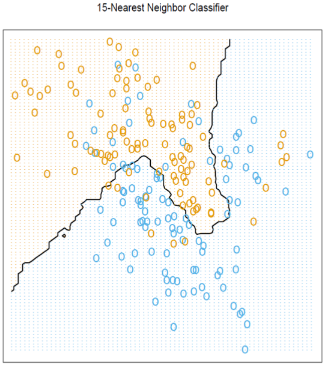


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.*

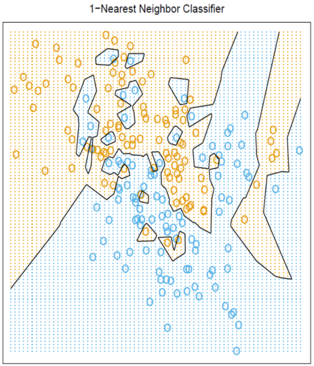


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$), *and then predicted by 1-nearest-neighbor classification.*

[2]Source: ESL pp.15-16