INTRO TO MEASURES OF DISTANCE/SIMILARITY, DOCUMENT SIMILARITY, CLUSTERING AND R

Gates

TOPICS

This document covers many topics, including:

Intro to Clustering

Measure of Distance and Similarity

Examples in Document/Text Similarity

- TF-IDF
- Cosine Similarity

More on Clustering

- K – means
- Hierarchical
- Densisty

WHAT IS CLUSTERING?

1) **Clustering is "unsupervised".** This means that the data is not labeled or categorized.

2) Clustering is used to "discover" if groups/categories exist and if so – what they are.

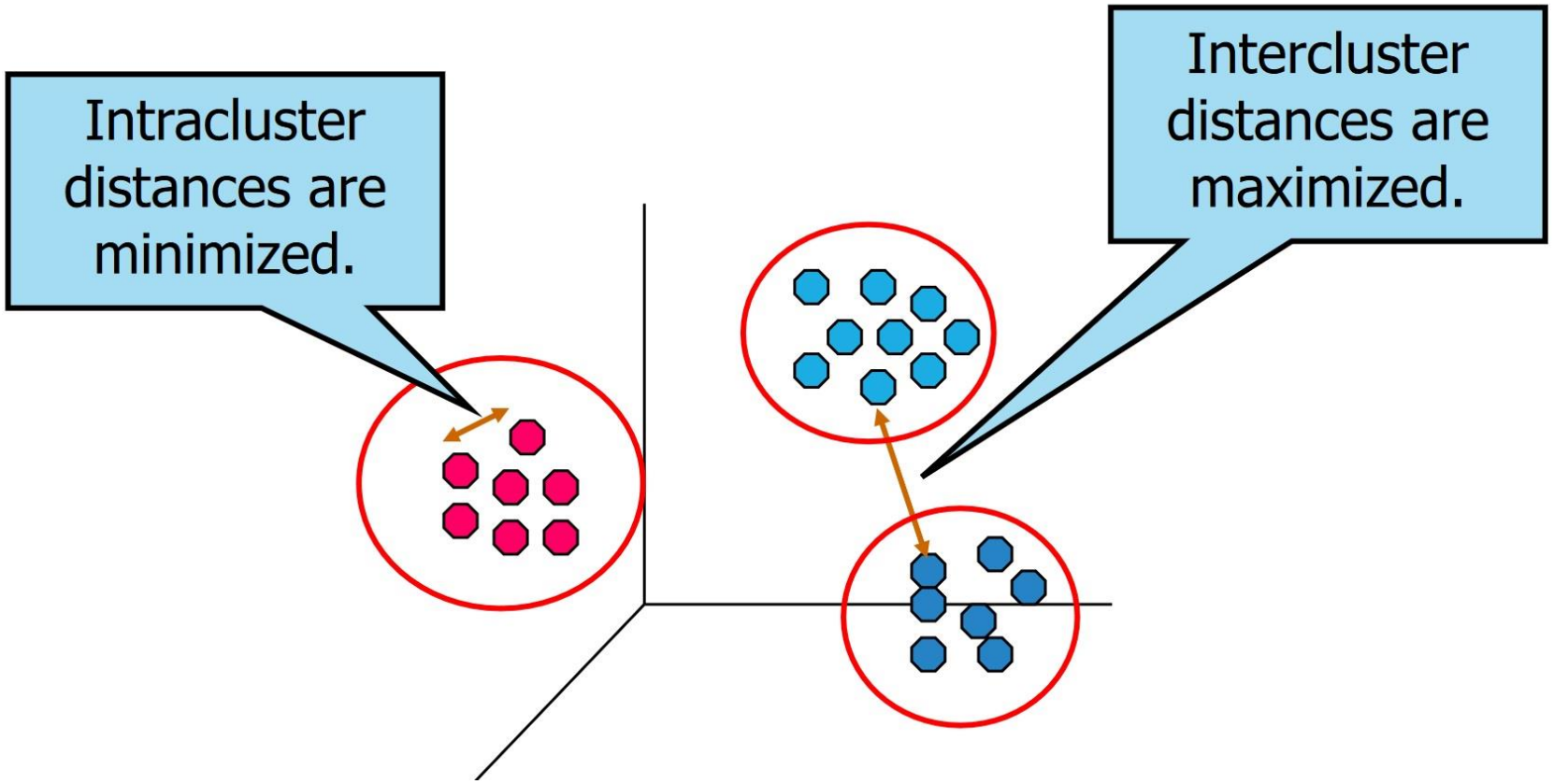
3) Clustering can be used to determine if the data in a dataset fits into any type of groups/categories/classes.

4) If categories can be identified, this information then be used to **classify** or **predict** other vectors.

5) In a very simple way, clustering should group data vectors (rows are data) that are "similar" and at the same time, maintain a dis-similarity *between* clusters.

- But – how can we measure "similarity"?

VERY GENERAL GOAL OF CLUSTERING



APPLICATIONS OF CLUSTERING

1) There are an enormous number of applications for clustering, as clustering is a method for discovering similarity (and differences) between data vectors/rows.

2) One can cluster books, articles, or documents by topics.

3) Once can cluster customers by common attributes – such as purchase similarities, location similarities, expenditure similarities, etc.

4) One can cluster social data by attributes such as common interests, common career areas, etc.

5) One can cluster radiation data collected from objects is space to determine if they are star, planets, galaxies, etc.

6) One can cluster music into categories – think about this for a second – how do you think Pandora does this?

Clustering can also be used for outlier detection. This is especially true for visual EDA (exploratory data analysis).

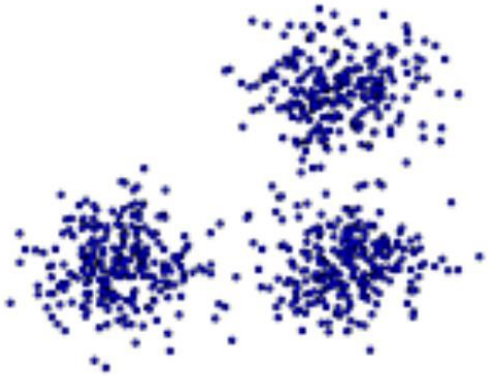
CHALLENGES OF CLUSTERING

1) Clustering is powerful and excellent for discovery. However, it also has many challenges associated with it:

- High Dimension

- When data vectors (each row of data) have many attributes (variable values), it is not possible to visualize this data.
- In addition, the higher the dimensionality, the less accurate some measure of similarity can be become. For example, the Euclidean distance loses accuracy. Why? Because at very high dimension, any two points are very far apart with respect to this type of distance.
- For higher D data, other similarity measures such as NN (nearest neighbor) or Cosine similarity.

- Clusters may be based on density or on distance.



MEASURES OF SIMILARITY/DISTANCE

1) The goal of clustering is to group together "similar" data vectors (also called data rows) while distinguishing between non-similar vectors.

2) To determine if two vectors are similarly, a **measure of similarity** is needed.

TWO GENERAL DISTANCE MEASURE TYPES:

(a) Euclidean distance – used to measure real-valued attributes and dense points.

- Euclidean distance is based on the location (as an x, y, z, .. coordinate) of each vector (point) in space.

(b) non-Euclidean

- This type of measure is based on the properties of the vectors rather than their location in a defined space.

SOME FUN MATH: THE AXIOMS OF A DISTANCE MEASURE

1) For any measure D to be a “distance measure” it must meet the following properties:

(a) Given any two vectors x and y, $D(x,y) >= 0$. In other words, distance cannot be negative.

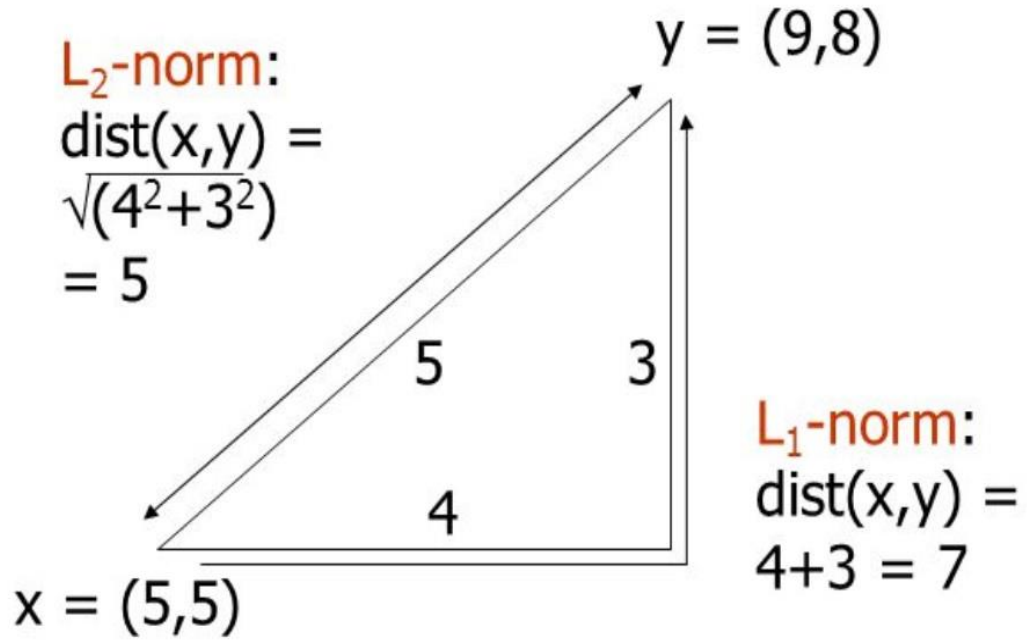
(b) $D(x, y) = 0$ iff $x = y$. In other words, the only way the distance between two vectors is 0 is if they are the same vector.

(c) $D(x,y) = D(y,x)$. Direction does not matter.

(d) $D(x, y) <= D(x,z) + D(z,y)$. This is called the Triangle Inequality.

Some Euclidean Distances

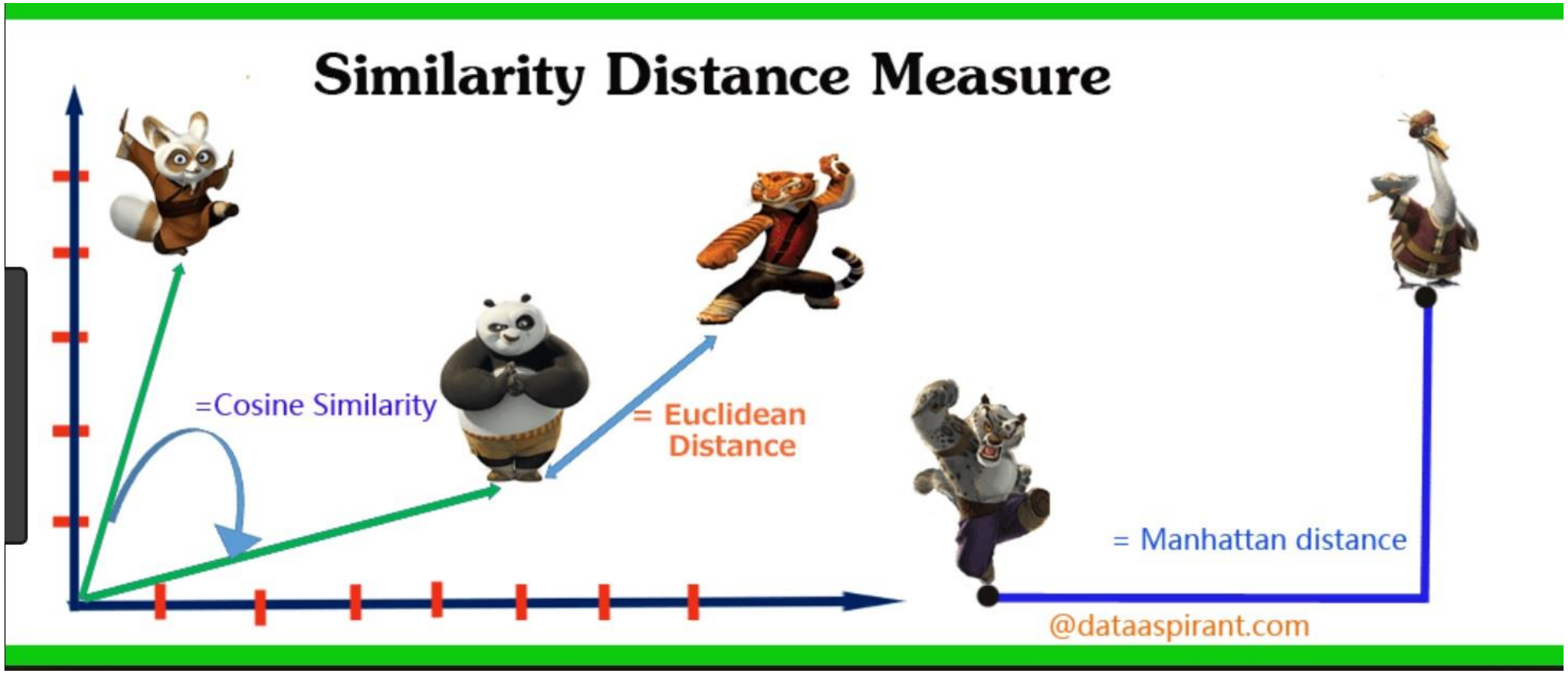
- L_2 *norm* : d(x,y) = square root of the sum of the squares of the differences between *x* and *y* in each dimension.
  - The most common notion of "distance."
- L_1 *norm* : sum of the differences in each dimension.
  - *Manhattan distance* = distance if you had to travel along coordinates only.



Note: The L3 norm, cubes the differences and so no.

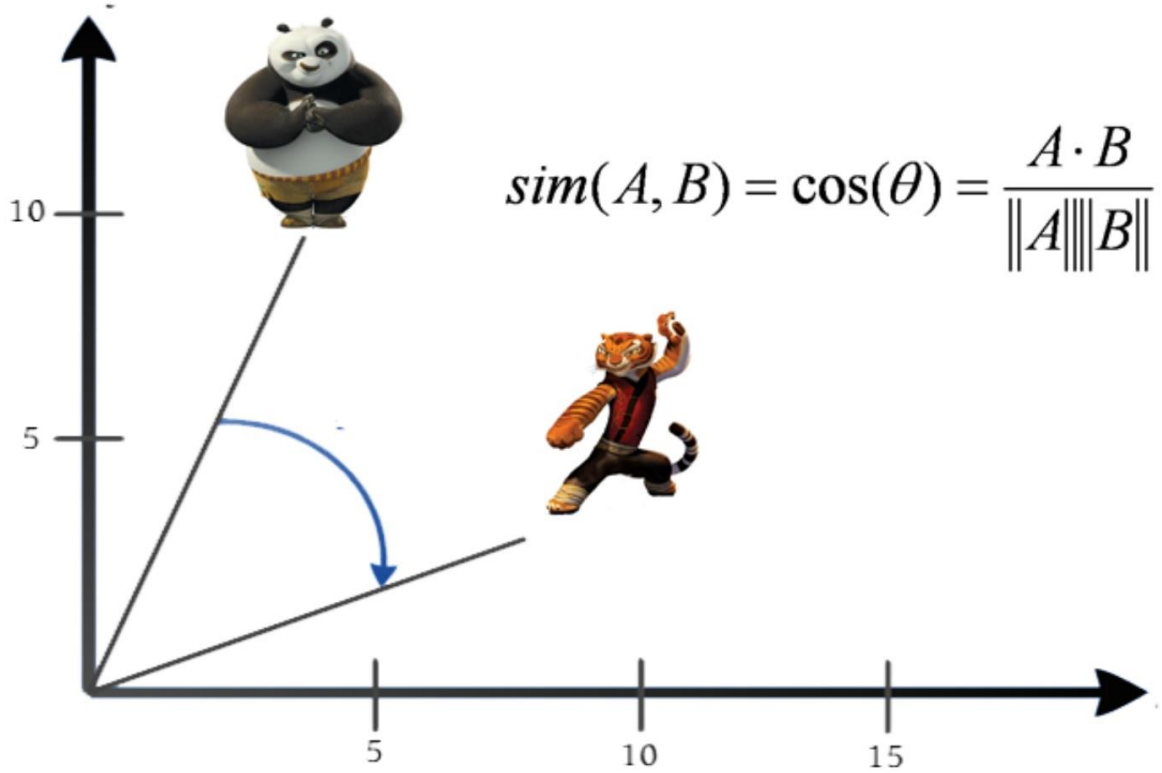
The L1 norm is often called the **Manhattan distance** as it is based on the idea of "block distance" rather than point to point direct distance.

A COMPARISON



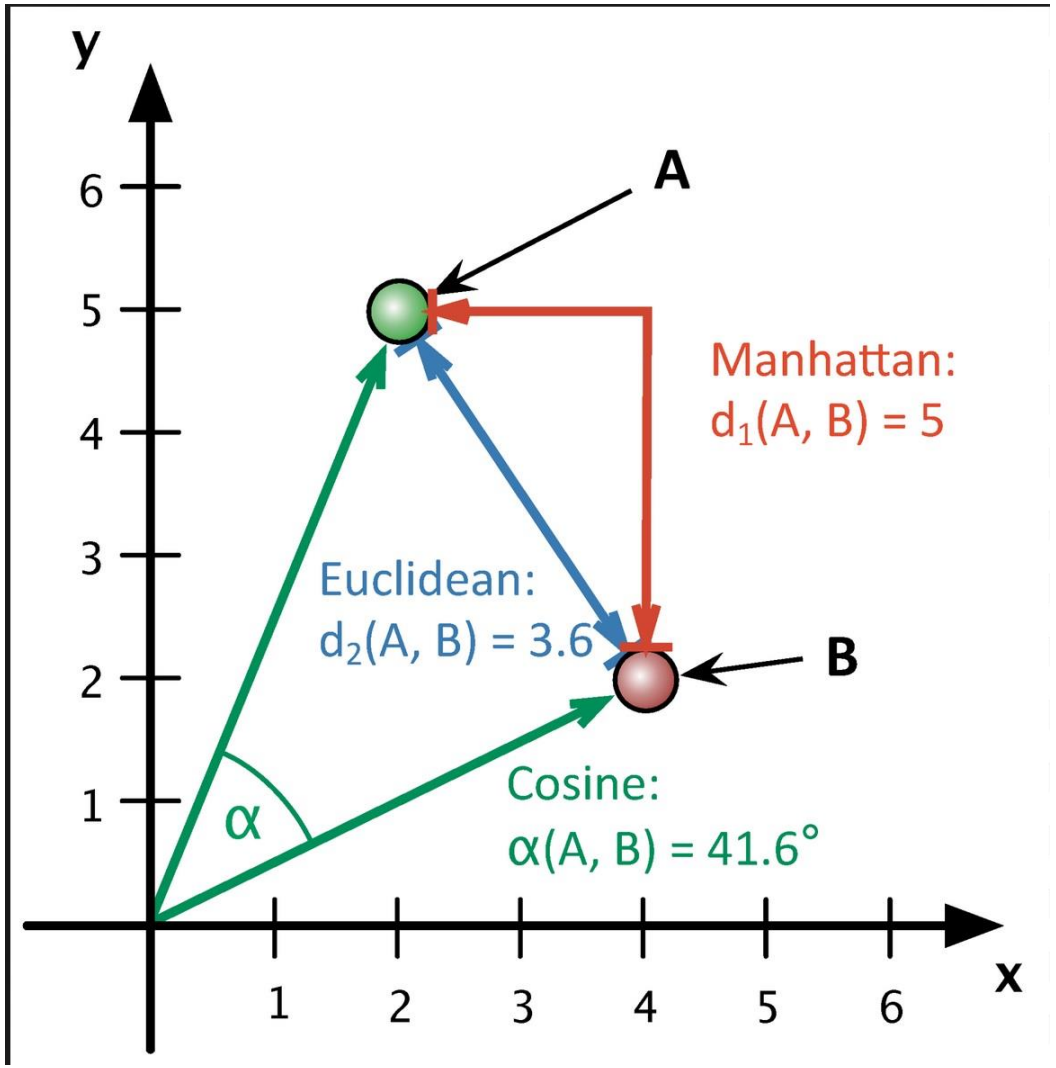
Cosine Similarity

http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/



$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Cosine Similarity is non-Euclidean.



NON-EUCLIDEAN DISTANCES

- *Jaccard distance* for sets = 1 minus ratio of sizes of intersection and union.
- *Cosine distance* = angle between vectors from the origin to the points in question.
- *Edit distance* = number of inserts and deletes to change one string into another.

http://infolab.stanford.edu/~ullman/mining/pdf/cs345-cl.pdf

Jaccard Distance

- Example: p_1 = 10111; p_2 = 10011.
  - Size of intersection = 3; size of union = 4, Jaccard measure (not distance) = 3/4.
- Need to make a distance function satisfying triangle inequality and other laws.
- d(x,y) = 1 – (Jaccard measure) works.

http://infolab.stanford.edu/~ullman/mining/pdf/cs345-cl.pdf

COSINE SIMILARITY

Good for high D data.

1) Imagine that each row or data vector is a numerical vector.

2) Next, no matter what dimension you are using, the origin is (0, 0, …0)

So in 2D the origin is (0,0) in 3D its (0,0,0) and so on.

3) Next, Any two vector points in any D space create an angle between them.

4) The COS of any angle is defined as the normalized dot product:

(V1 . V2) / |V1| |V2|

Example: Suppose vector 1 (V1) is [1, 3] and suppose vector 2 (V2) is [2,2]

Then, the dot product V1 . V2 = (1*2) + (3*2) = 8

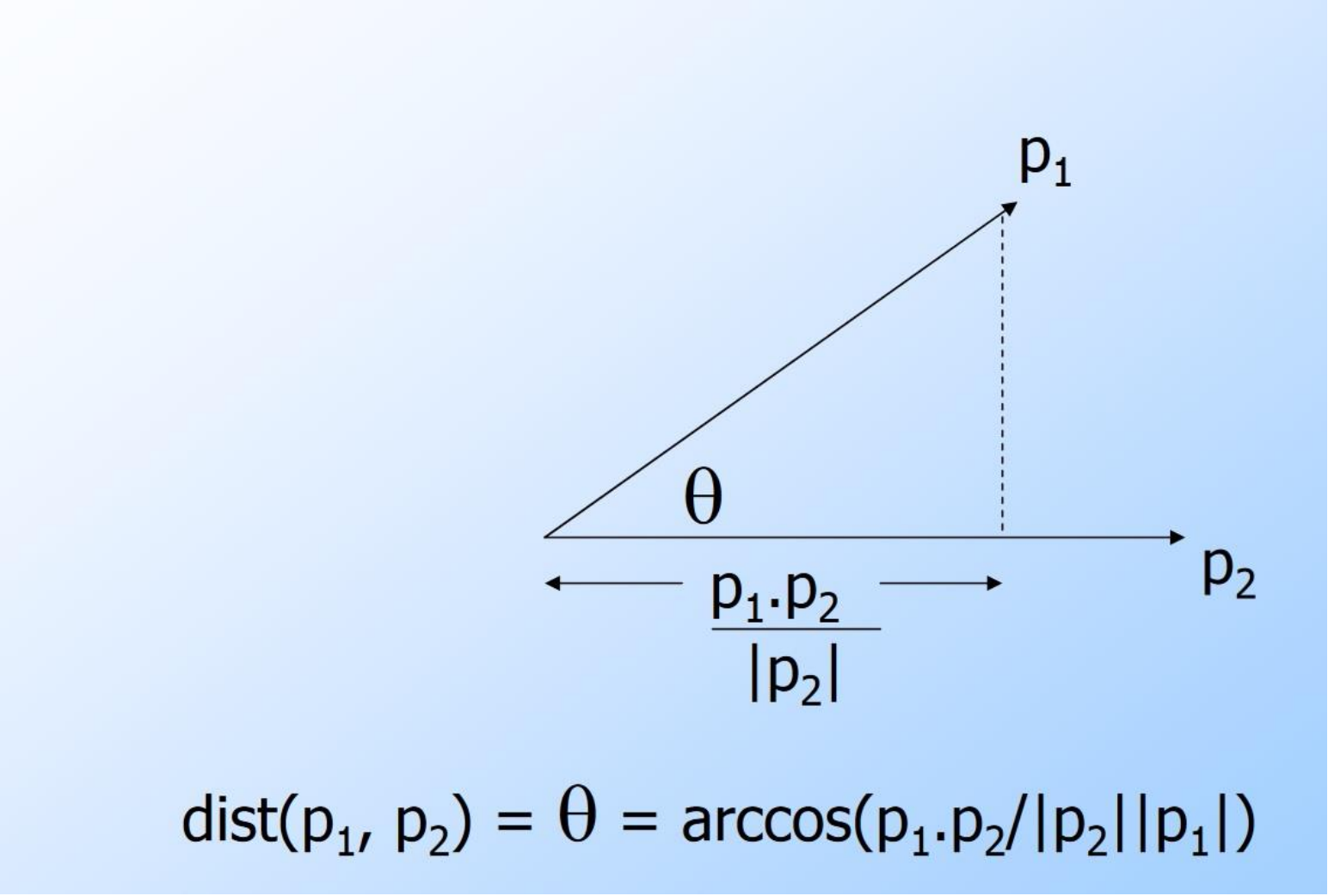
Next, |V1| = sqrt(1^2 + 3^2) = sqrt(10)

|V2| = sqrt(2^2 + 2^2) = sqrt(8)

So COS() = 8 / (sqrt(8)*sqrt(10)) = .894

To solve for the angle, find the arccos (.894) = 26.6 degrees

VISUAL OF COS MEASURE



EDIT DISTANCE

1) The Edit Distance between any two strings or sequences of characters is **the number of inserts and deletes of characters needed to make the strings identical.**

2) First, the **longest common subsequence (LCS)** is defined as the longest string obtained by deleting from x and y.

Then, $d(x,y) = |x| + |y| - 2|LCS(x,y)|$

- x = *abcde* ; y = *bcduve*.
- Turn x into y by deleting *a*, then inserting *u* and *v* after *d*.
  - Edit-distance = 3.
- Or, LCS(x,y) = *bcde*.
- $|x| + |y| - 2|LCS(x,y)| = 5 + 6 - 2*4 = 3$

DISTANCE MEASURES FOR NOMINAL DATA

Example: How similar are these two people?

i = (Refund = Yes, Married, Income = 120K)

j = (Refund = No, Divorced, Income = 90K)

Taxpayer	Refund	Marital Status	Income in Thousands
i	Yes	Married	120
j	No	Divorced	90

MEASURE BY MATCHES

Taxpayer	Refund	Marital Status	Income in Thousands
i	Yes	Married	120
j	No	Divorced	90

m: Number of matches; *p*: Total number of nominal variables

$$d(i,j) = \frac{p-m}{p}$$

D(I,j) = (3) - (0) / (3) = 1 (farthest possible distance)

However, we can categorize the numerical data.
Example: Salary Range 1: 0 – 30K, Range 2: 31K – 50K, Range 3: 51K – 89K, Range 4: 90K – 130K, etc.

Then, D(I,j) = (3) - (1) / (3) = 2/3

Pages 69-79 Kumar (2005)

Binary Attributes

- A contingency table for binary data

Object *i* \ Object *j*	1	0	sum
1	*q*	*r*	*q*+*r*
0	*s*	*t*	*s*+*t*
sum	*q*+*s*	*r*+*t*	*p*

- Distance measure for symmetric binary variables

$$d(i,j) = \frac{r+s}{q+r+s+t}$$

- Distance measure for asymmetric binary variables

$$d(i,j) = \frac{r+s}{q+r+s}$$

- Jaccard coefficient (*similarity* measure for *asymmetric* binary variables)

$$sim(i,j) = \frac{q}{q+r+s} = 1 - d(i,j)$$

DOCUMENT SIMILARITY COMPARISON (DISTANCE): AN EXAMPLE OF THE NEED AND VALUE OF NORMALIZATION

1) The **goal** is to utilize the word types and frequencies in documents **to determine how similar the documents are.**

2) Recall that similarity is a measure of distance.

3) First, think about the problem itself. Imagine three documents. Suppose the first has 100 words and the second has 5000 words, and the third has 10,000 words. Simply by the number of words, it will appear that Doc 2 and Doc 3 are more similar.

4) Therefore, before comparing documents, weighting and normalization must be used.

5) TF- IDF

- TF stands for TERM FREQUENCY
- IDF stands for INVERSE DOC FREQUENCY
- **TF-IDF is considered a "weight"**

TF - IDF

Compute idf_t using the formula: $\text{idf}_t = \log_{10} \frac{1{,}000{,}000}{\text{df}_t}$

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

TF-IDF EXAMPLE

Example:
Consider a document containing 100 words wherein the word cat appears 3 times.
TF for cat = (3 / 100) = 0.03.

Now, assume we have 10 million documents and the word cat appears in one thousand of these.
IDF = log(10,000,000 / 1,000) = 4.

Thus, the **TF-IDF weight** is the product of these quantities: **0.03 * 4 = 0.12**.

REPRESENTING DOCUMENTS AS VECTORS/MATRICES

The following slides will show difference methods for representing documents.

Suppose, for this example, that there are 6 documents. In this case, the documents are novels.

Next, suppose we are looking at 7 words and how they occur in each of the 6 documents.

Examples….

BINARY REPRESENTATION OF OCCURRENCE

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

REPRESENTATION BY ACTUAL COUNT

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

TF – IDF WEIGHT MATRIX

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0
MERCY	1.51	0.0	1.90	0.12	5.25	0.88
WORSER	1.37	0.0	0.11	4.15	0.25	1.95

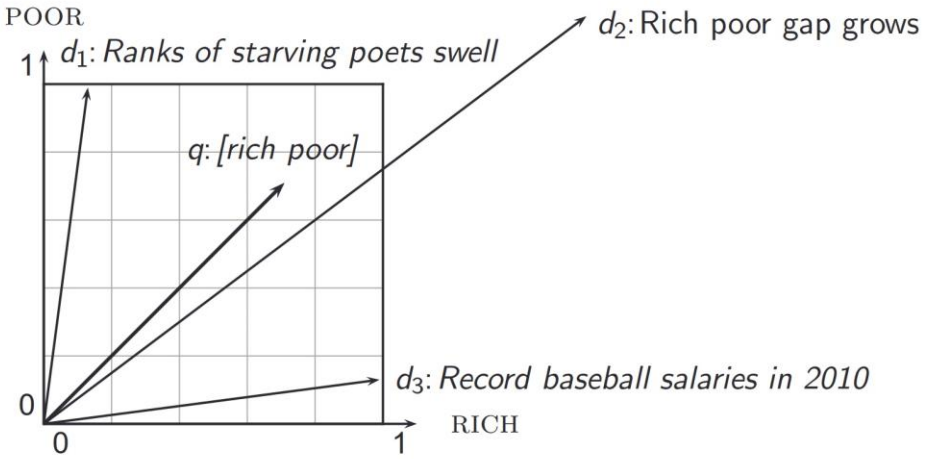
Documents are vectors (also called high D "points" in a D-space

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

SIMILARITY MEASURES: MEASURING THE "DISTANCE" BETWEEN DOCUMENT VECTORS IN D SPACE.

Options for measuring the Distance (or similarity) between documents:

1) Euclidean – this is not a good idea, as the greater the dimension, the less accurate the measure. (recall that the number of words you are looking at IS the dimension). Euclidean distance can be very large for vectors of different length.
2) **Angles between** vectors is a better measure… COSINE SIMILARITY



The Euclidean distance of $\vec{q}$ and $\vec{d}_2$ is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document d'. d' is twice as long as d.
- "Semantically" d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

LENGTH NORMALIZATION USING COSINE (L2 NORM)

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use the L_2 norm:
  $||x||_2 = \sqrt{\sum_i x_i^2}$
- This maps vectors onto the unit sphere . . .

  . . . since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

COSINE SIMILARITY EXAMPLE

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.
- $\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$
  - (if $\vec{q}$ and $\vec{d}$ are length-normalized).

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ . . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

How similar are the following novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

Term frequencies (raw counts)

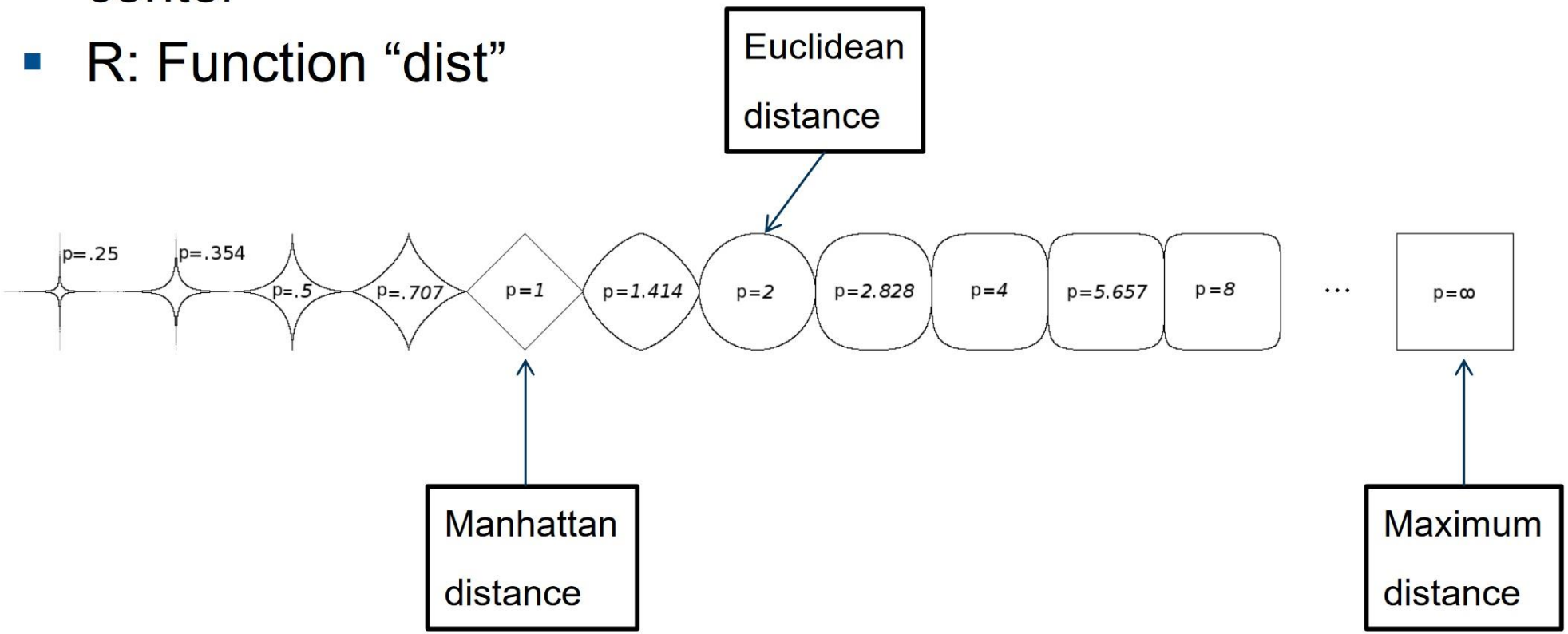
term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

	Term frequencies (raw counts)			Log frequency weighting			Log frequency weighting and cosine normalisation		
term	SaS	PaP	WH	SaS	PaP	WH	SaS	PaP	WH
AFFECTION	115	58	20	3.06	2.76	2.30	0.789	0.832	0.524
JEALOUS	10	7	11	2.0	1.85	2.04	0.515	0.555	0.465
GOSSIP	2	0	6	1.30	0.00	1.78	0.335	0.000	0.405
WUTHERING	0	0	38	0.00	0.00	2.58	0.000	0.000	0.588

- (To simplify this example, we don't do idf weighting.)
- cos(SaS,PaP) $\approx$
  $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69

Intuition for Minkowski Distance

- p: Index of Minkowski Distance
- Points on the line have equal Minkowski Distance from center
- R: Function “dist”



BACK TO CLUSTERING: TWO MAIN TYPES

1) **Partitioned** (also known as point assignment) – top down – divide objects into non-overlapping

k-means, k-medoids, CLARANS, EM

2) **Hierarchical** – bottom up – all points start as their own cluster – then merge and grow into larger clusters based on closeness/similarity.

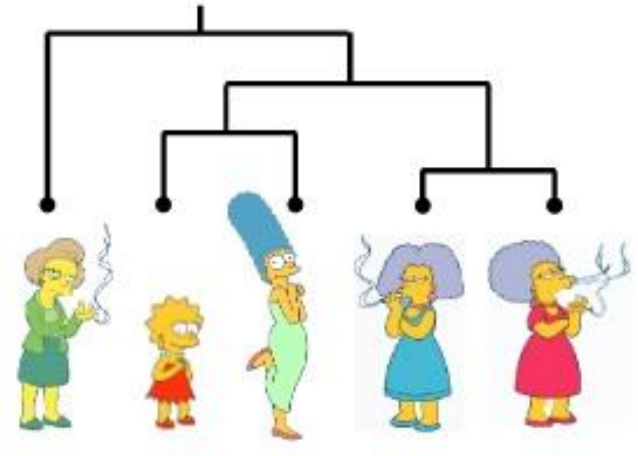
DIANA (DIvisive ANAlysis Clustering), AGNES (Agglomerative Nesting), BIRCH (Balanced, Iterative, Reductive), ROCK (Robust CLustering), CHAMELEON

3) **DBSCAN**

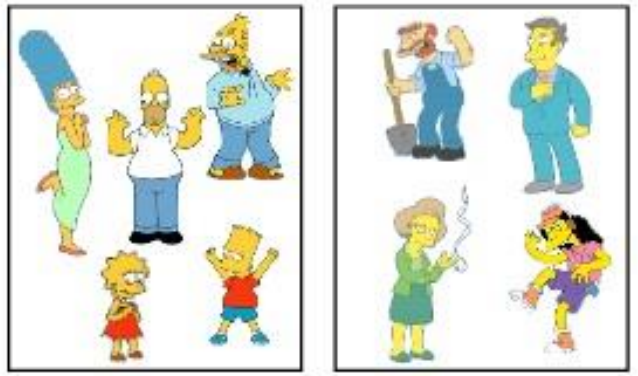
PARTITION VS. HIERARCHICAL

- **Partitional algorithms:** Construct various partitions and then evaluate them by some criterion (we will see an example called BIRCH)
- **Hierarchical algorithms:** Create a hierarchical decomposition of the set of objects using some criterion

Hierarchical



Partitional



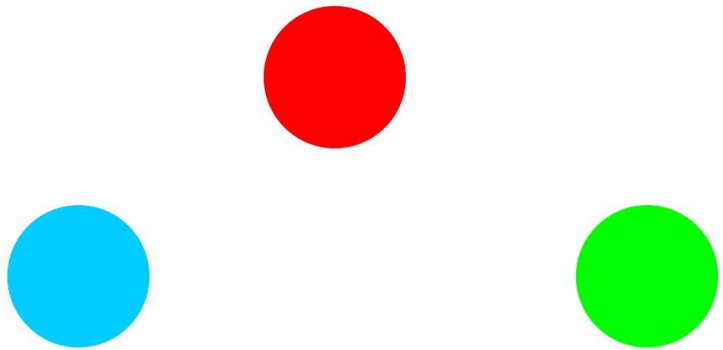
OTHER SPECIAL CLUSTERING TYPES

- Exclusive (or non-overlapping) versus non-exclusive (or overlapping)
  - In non-exclusive clusterings, points may belong to multiple clusters.
    - Points that belong to multiple classes, or 'border' points
- Fuzzy (or soft) versus non-fuzzy (or hard)
  - In fuzzy clustering, a point belongs to every cluster with some weight between 0 and 1
    - Weights usually must sum to 1 (often interpreted as probabilities)
- Partial versus complete
  - In some cases, we only want to cluster some of the data

WELL SEPARATED CLUSTERS (EASIER TO MANAGE) AND CENTER-BASED

- Center-based
  - A cluster is a set of objects such that an object in a cluster is closer (more similar) to the "center" of a cluster, than to the center of any other cluster
  - The center of a cluster is often a centroid, the minimizer of distances from all the points in the cluster, or a medoid, the most "representative" point of a cluster

- Well-Separated Clusters:
  - A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster.

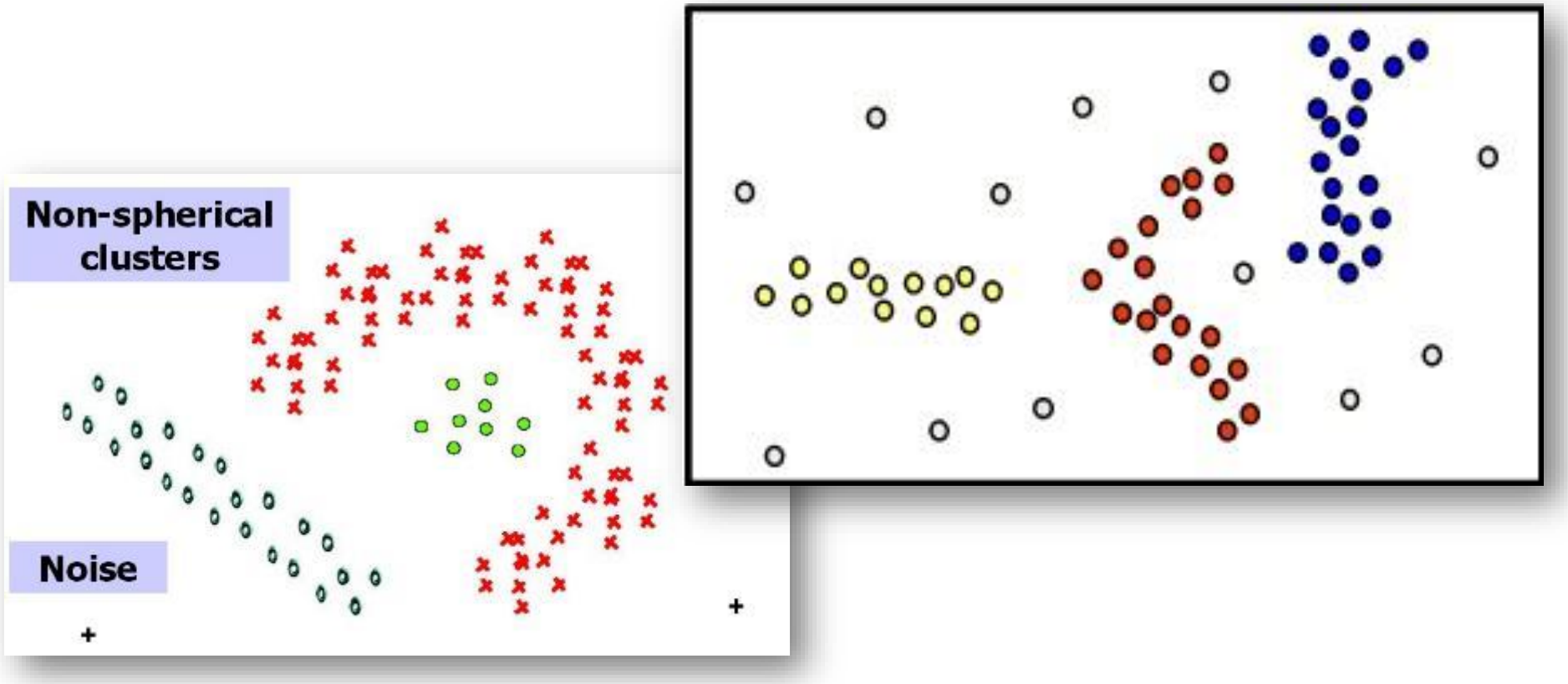


3 well-separated clusters

TYPES OF CLUSTERS: DENSITY-BASED

Density-based

- A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.
- Used when the **clusters are irregular** or **intertwined**, and when noise and outliers are present.



K-means Clustering

- Partitional clustering approach
- Each cluster is associated with a centroid (center point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters, K, must be specified
- The objective is to minimize the sum of distances of the points to their respective centroid

K MEANS CLUSTERING EXAMPLE BY HAND AND IN R

A	-1	3.2			Distances between each point		Euclidean				
B	3.9	3.8			AB	4.94	Choose initial centers at A (-1,3.2) and B (3.9,3.8)				
C	4.8	4.5			AC	5.94					
D	3.1	-3.5			AD	7.85					
E	-5	3.7			AE	**4.03**	1) place E in same cluster as A				
					BC	**1.14**	1) place C in same cluster as B				
					BD	7.34	Recalculate centers of clusters				
					BE	8.90					
					CD	8.18	Now we have AE and BC				
					CE	9.83	Center of AE is	-3	3.45		
					DE	10.84	Center of BC is	4.35	4.15		
							There is one more point to place.				
							Determine which cluster to place it into				
							Recall that Cluster 1 has A and E				
			Final Clusters:				Cluster 2 has B and C				

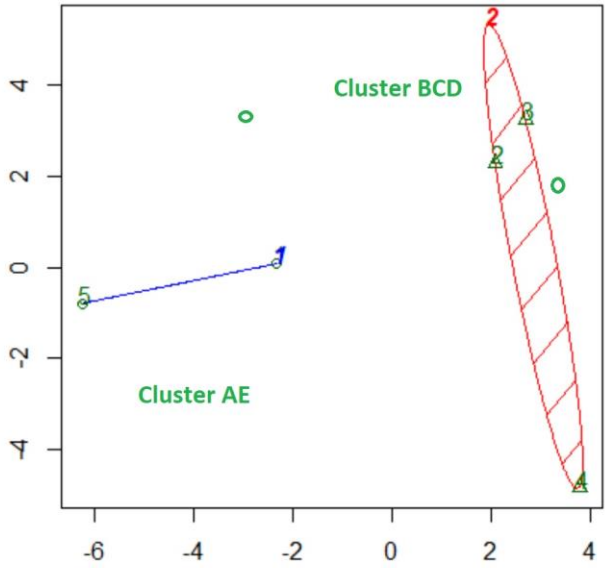
RESULTS BY HAND

Final Clusters:				
AE and BCD			D has not been placed.	
			Distance from D to other two cluster centers	
Final Centers:			Distance from D to AE is	9.247297
Center of AE is	-3	3.45	Distance from D to BC is	7.751451
Center of BCD is	3.933333	1.6		
			D is closer to Cluster 2	

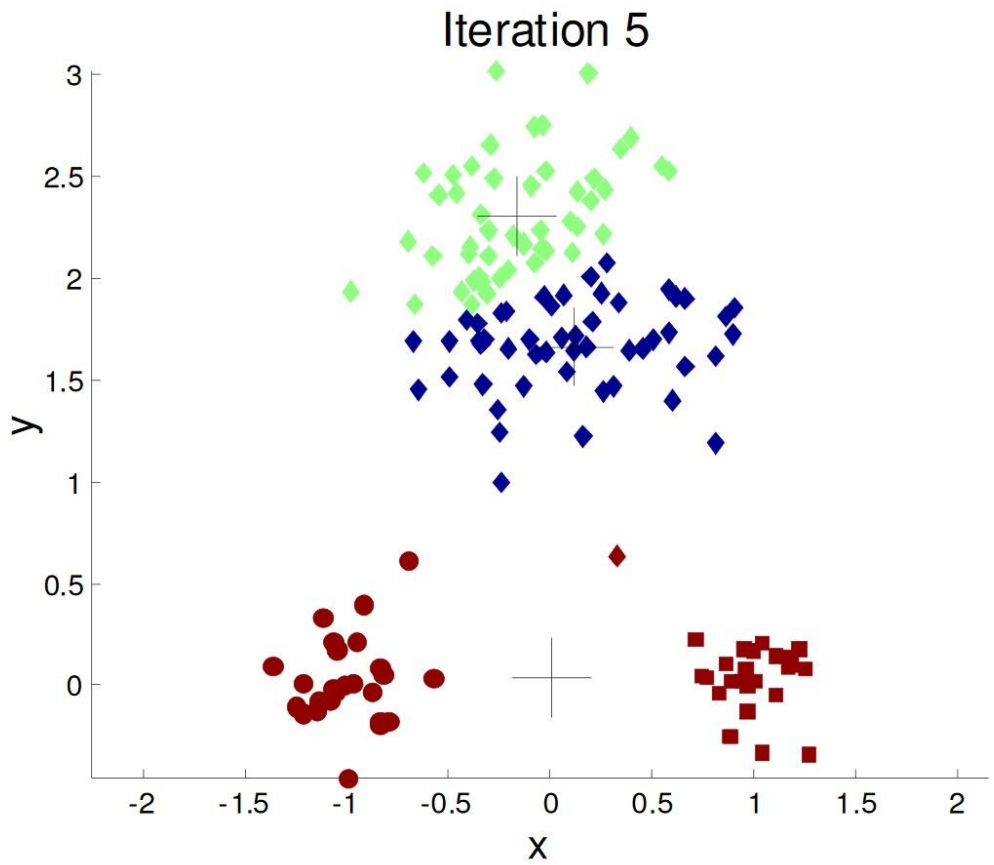
```r
## k - means
#install.packages("cluster")
library(cluster)

## When learning any new method - it is best to
## do it by hand using a very small dataset.
## Then, use a langauge like R or Python to check the
## results.

## This will help you to see if you did it right AND to see
## what R (or pYthon) is doing.

## Using methods blindly is very dangerous!

(SmallExample <- read.csv("SmallClusterDataset.csv", header=FALSE))
str(SmallExample)
(DataNoLabels <- SmallExample[,c(2:3)])

k = 2  ## choose the number of clusters
(kmeansSmallModel <- kmeans(DataNoLabels, k, centers = DataNoLabels[c(1,2),],
  algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                "MacQueen")))
(kmeansSmallModel[1])
### VIEW
clusterGroups <- data.frame(SmallExample,kmeansSmallModel$cluster)

## NOTE: the "V" in view must be cap (not lowercase)
View(clusterGroups)

## plot the clusters

## use PCA (prin comp analysis) to reduce the Dim
## to view the clusters
clusplot(DataNoLabels, kmeansSmallModel$cluster,color=TRUE,
         shade=TRUE, labels=2, lines=0)
```

RESULTS FROM R ARE THE SAME AS BY HAND

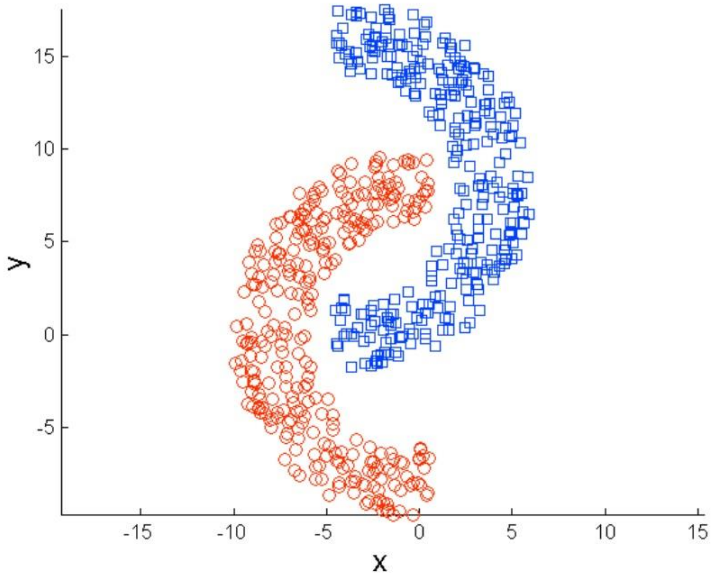
	V1	V2	V3	kmeansSmallModel.cluster
1	A	-1.0	3.2	1
2	B	3.9	3.8	2
3	C	4.8	4.5	2
4	D	3.1	-3.5	2
5	E	-5.0	3.7	1



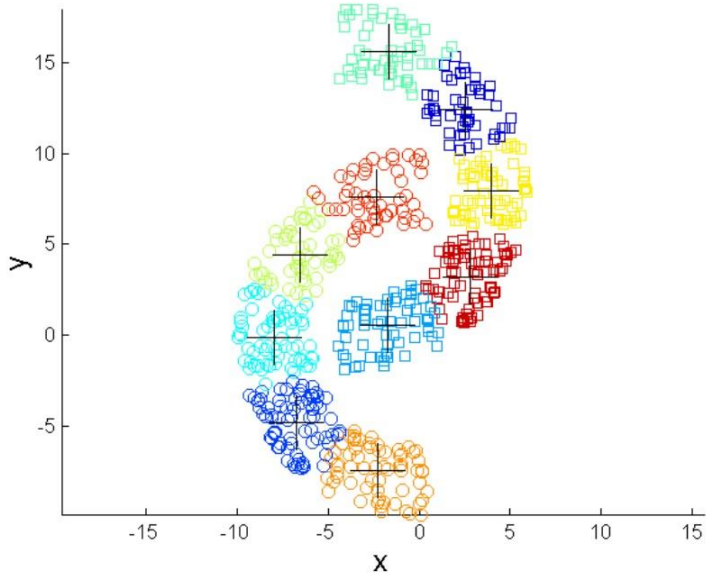
Importance of Choosing Initial Centroids



LIMITATION OF K-MEANS



Original Points



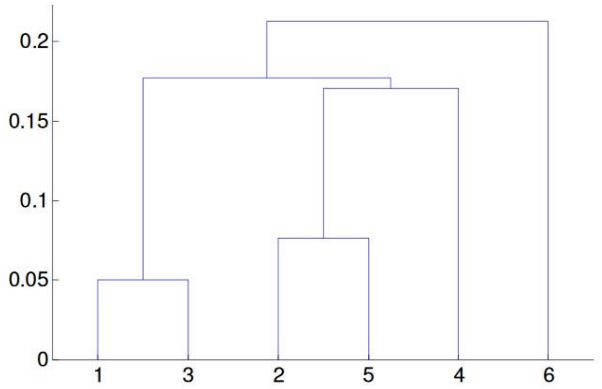
K-means Clusters

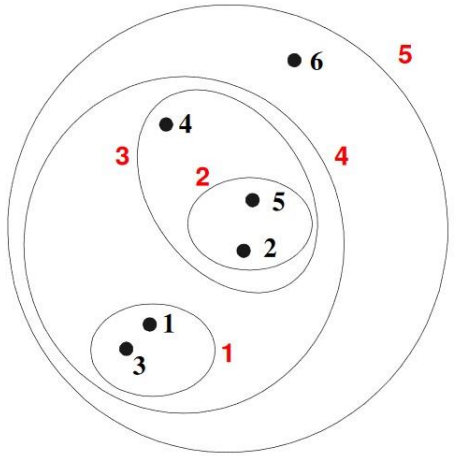
Hierarchical Clustering

- Two main types of hierarchical clustering
  - Agglomerative:
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - Divisive:
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains a point (or there are k clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  - A tree like diagram that records the sequences of merges or splits





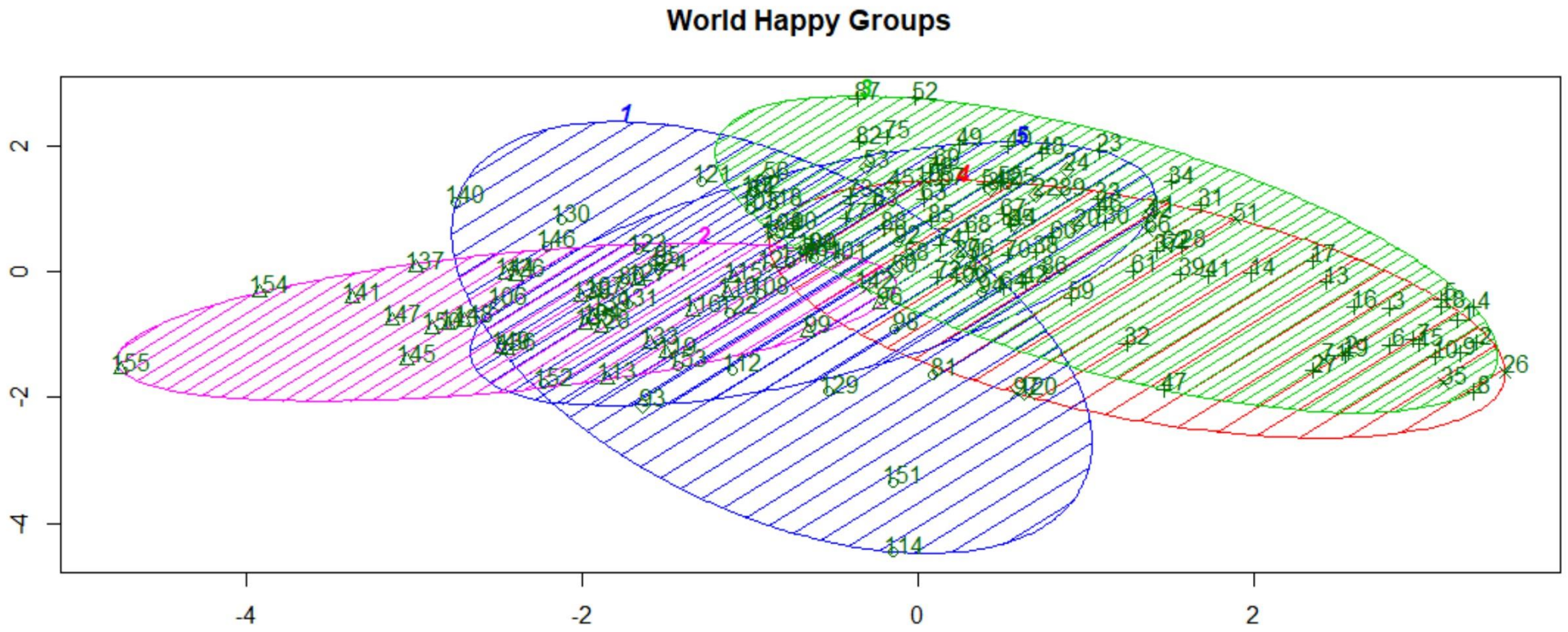
1) It is not necessary to choose or preselect the number of clusters.
2) Hierarchical clusterings may correspond well to taxonomies – such as the animal kingdom.

K MEANS AND HIERARCHICAL CLUSTERING R CODE

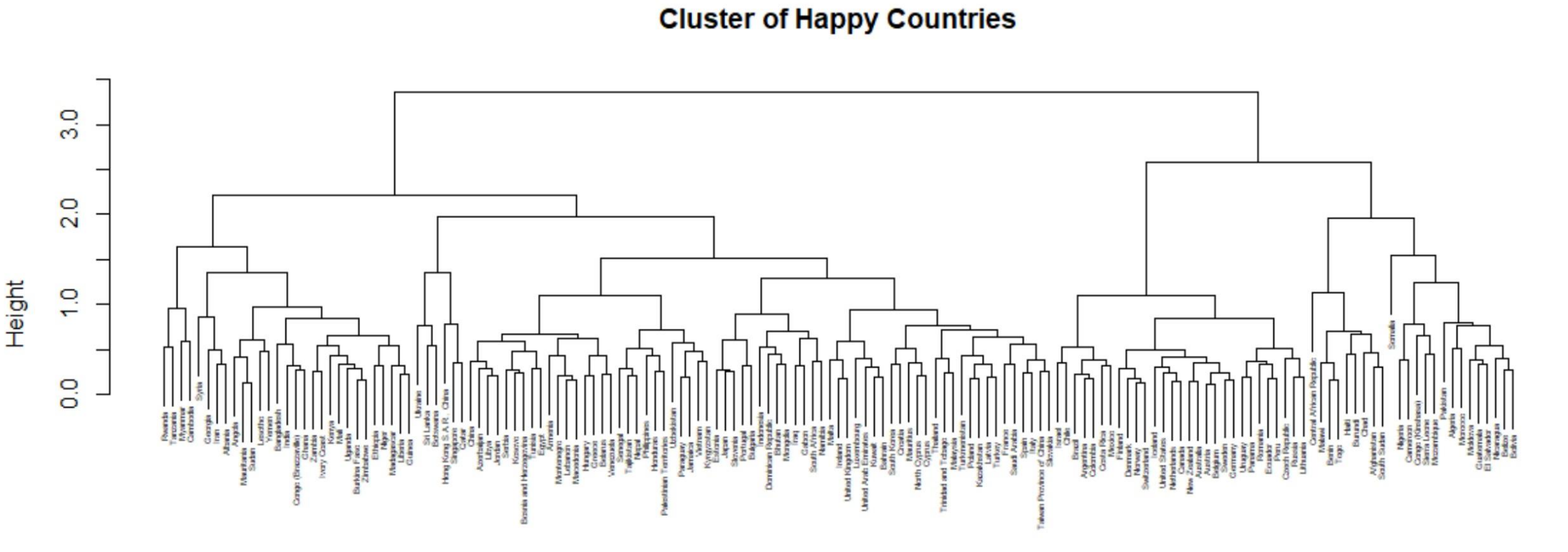
EXAMPLE OF HAPPINESS DATA CLUSTERING WITH CLUSTERS AND DENDROGRAM VIEWS

```
## Gates

setwd("C:\Users\profa\Documents\R\RStudioFolder_1\DrGExamples")

## k - means
#install.packages("cluster")
library(cluster)

(HappyData <- read.csv("KaggleHappinessWorldData_raw.csv", header=TRUE))
str(HappyData )
##drop columns 2  - 5 as they give a measure of rank
HappyDataWithoutRanks <- HappyData[,-c(1:5)]
(head(HappyDataWithoutRanks, n = 10))
HappyDataWithNamesButNotRanks <- HappyData[,-c(2:5)]

k = 5  ## choose the number of clusters
(kmeansHappy_5 <- kmeans(HappyDataWithoutRanks, k))

## list of cluster assignments
o=order(kmeansHappy_5$cluster)
CountryHappyGroup <- data.frame(HappyData$Country[o],kmeansHappy_5$cluster[o])
(head(CountryHappyGroup,n=40))

## Vizualize
View(CountryHappyGroup)
clusplot(HappyDataWithoutRanks, kmeansHappy_5$cluster,color=TRUE,
         shade=TRUE, labels=2, lines=0,main= 'World Happy Groups')

## Hierarchical Clustering and Dendrograms
d <- dist(as.matrix(HappyDataWithNamesButNotRanks))   # find distance matrix
hc <- hclust(d)                # apply hirarchical clustering
plot(hc, labels = HappyData$Country, hang = 0.2, cex=.4,
     main = "Cluster of Happy Countries")
```

CLUSTER RESULTS



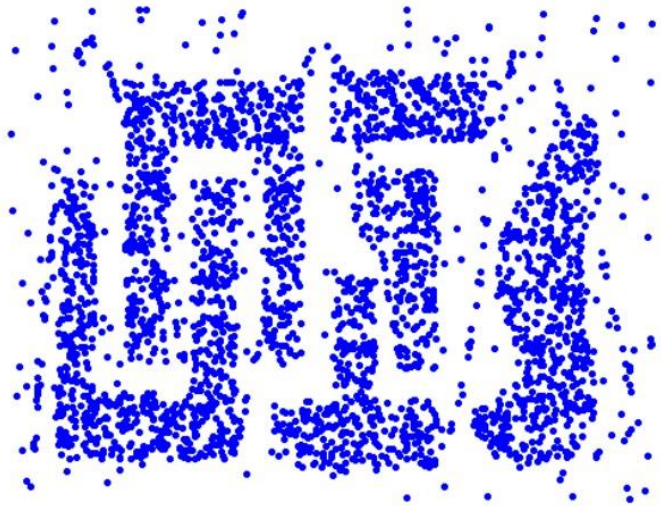
DENDROGRAM FROM R CODE



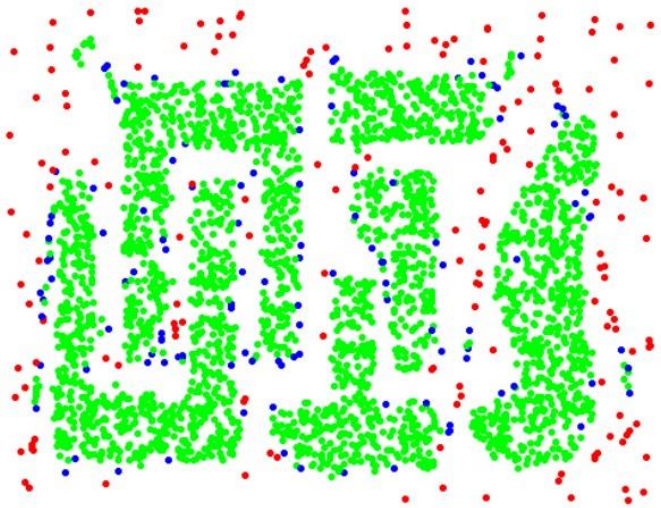
DBSCAN: Density-Based Clustering

- DBSCAN is a Density-Based Clustering algorithm
- Reminder: In density based clustering we partition points into dense regions separated by not-so-dense regions.
- Important Questions:
  - How do we measure density?
  - What is a dense region?
- DBSCAN:
  - Density at point p: number of points within a circle of radius Eps
  - Dense Region: A circle of radius Eps that contains at least MinPts points

DBSCAN: Core, Border and Noise Points



Original Points

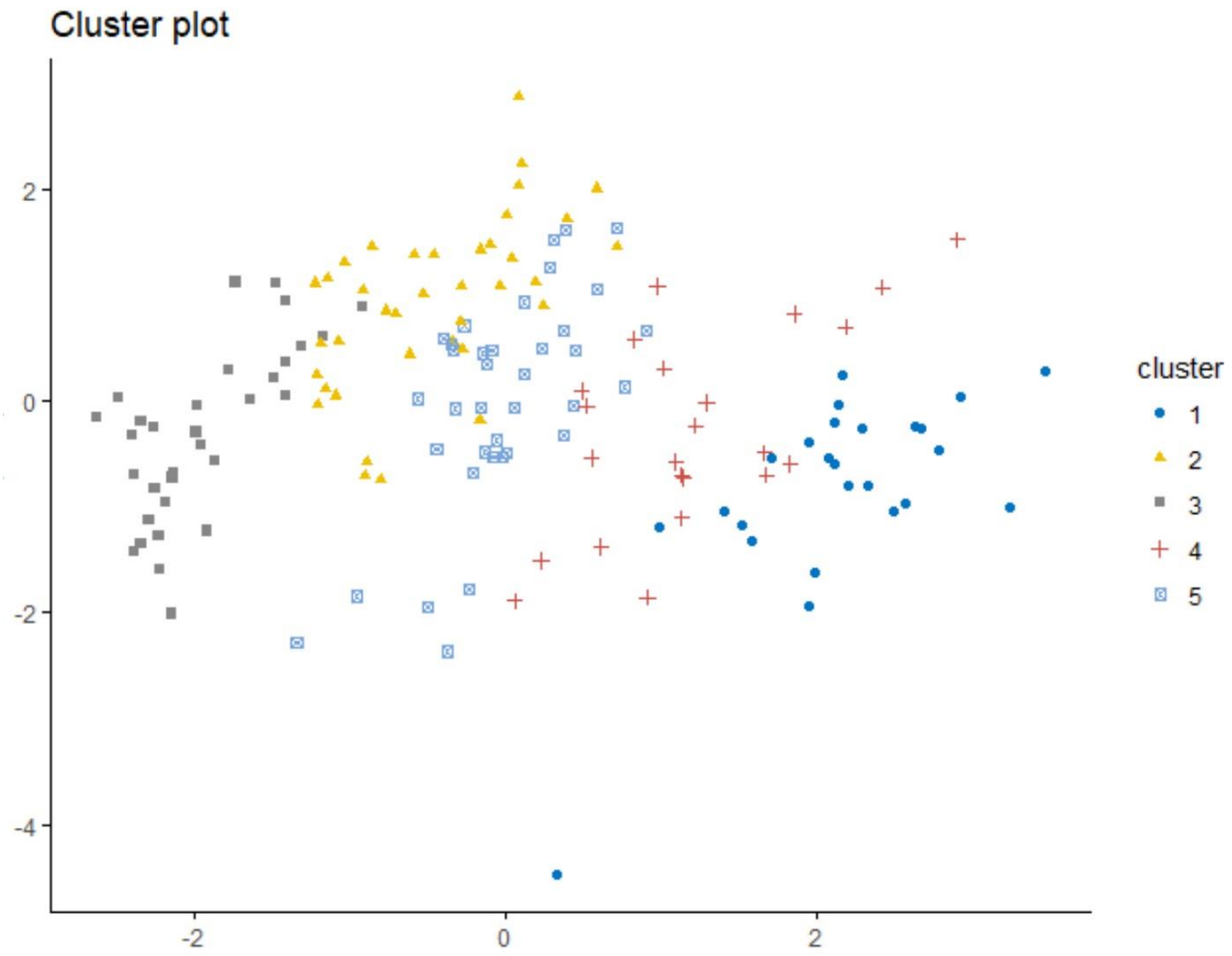


Point types: core, border and noise

Eps = 10, MinPts = 4

R CODE FOR DBSCAN

```
##library(factoextra) ## for DBSCAN

## install.packages("factoextra")

## DBSCAN Density-based clustering with columns

(HappyData <-
read.csv("KaggleHappinessWorldData_raw.csv",
header=TRUE))

df <- HappyData[, c(6,8,9,10)]

set.seed(123)

km.res <- kmeans(df, 5, nstart = 25)

fviz_cluster(km.res, df,  geom = "point",

             ellipse= FALSE, show.clust.cent = FALSE,

             palette = "jco", ggtheme = theme_classic())
```



COMPARING DOCUMENTS AND R

Gates

EXAMPLE CODE

You can always locate all of my example code and data files, etc. HERE:

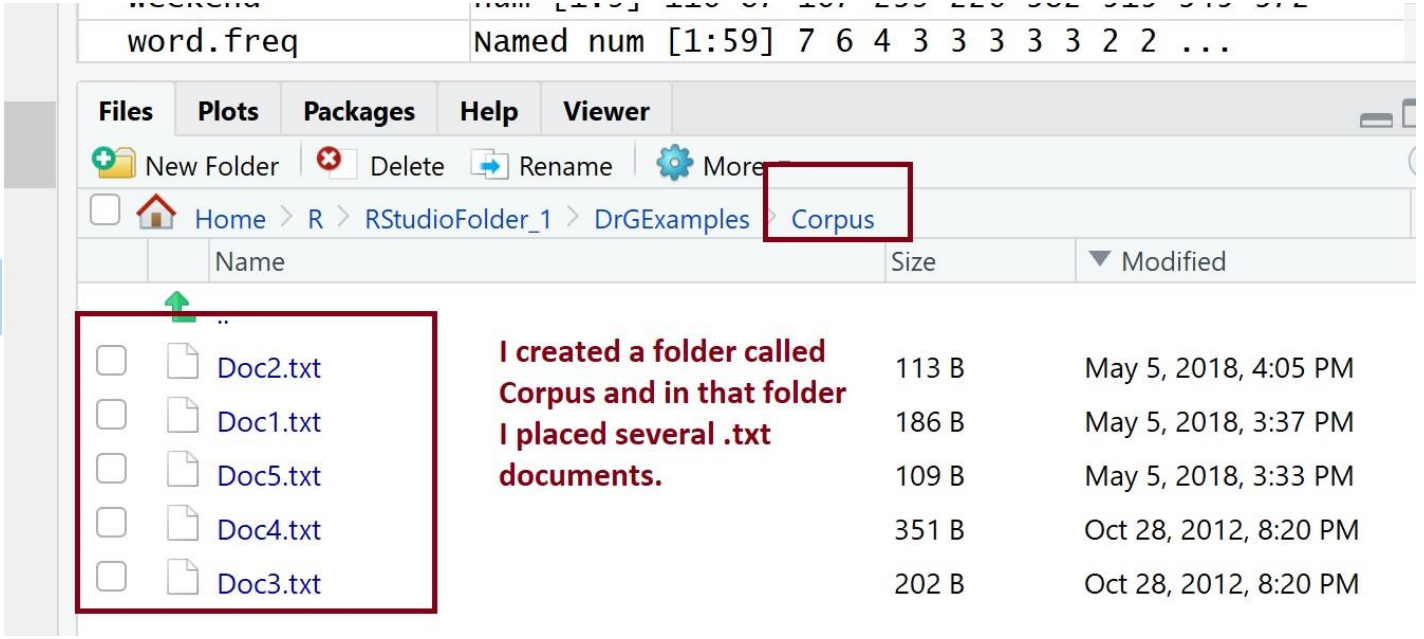
https://drive.google.com/drive/folders/1rXm4jTHMTTjFvHfJ3daCCWNmOdF9tNPl?usp=sharing

LIBRARIES

```
library(wordcloud)
## ONCE: install.packages("tm")
library(tm)
# ONCE: install.packages("Snowball")  ## Does not work in R v. 3.5.x as of 5/5/18. check back.
## ONCE: install.packages("slam")
library(slam)
library(quanteda)
## ONCE: install.packages("quanteda") ## Note - this includes SnowballC
library(SnowballC)
library(arules)
##ONCE: install.packages('proxy')
library(proxy)
```

THE CORPUS: A COLLECTION OF TEXT DOCUMENTS IN A FOLDER

```
TheCorpus <-
Corpus(DirSource("Corpus"))

##The following will show you that
##you read in 5 documents

(TheCorpus)
```



START TO PREPARE THE CORPUS FOR EVALUATION: CLEAN AND PREP IT PART 1

```
## The tm_map function allows you to perform the same
## transformations on all of your texts at once
CleanCorpus <- tm_map(TheCorpus, removePunctuation)
## Remove all Stop Words
CleanCorpus <- tm_map(CleanCorpus, removeWords, stopwords("english"))
## You can also remove words that you do not want
MyStopWords <- c("like", "very", "can", "I", "also", "lot")
CleanCorpus <- tm_map(CleanCorpus, removeWords, MyStopWords)
## NOTE: If you have many words that you do not want to include
## you can create a file/list
## MyList <- unlist(read.table("PATH TO YOUR STOPWORD FILE", stringsAsFactors=FALSE)
## MyStopWords <- c(MyList)
##Make everything lowercase
CleanCorpus <- tm_map(CleanCorpus, content_transformer(tolower))
```

START TO PREPARE THE CORPUS FOR EVALUATION: CLEAN AND PREP IT PART 2

```
## Let's see where we are so far...

inspect(CleanCorpus)

## Next, I will write all cleaned docs  - ##the entire cleaned and prepped corpus

## to a file - in case I want to use it for ##something else.

(Cdataframe <- data.frame(text=sapply(CleanCorpus, identity),  stringsAsFactors=F))

write.csv(Cdataframe, "Corpusoutput.csv")
```

```
[1] chocolate  yummy    eat chocolate every day dark chocolate   best it   good  almonds almonds  go
od  everything almonds  chocolate make  good cake

[2] coffee  chocolate go  well together one  eat chocolate cake  coffee   soy milk   coffee


[3] dogs   run  jump  eat things dogs  fun   protect   danger  run  help   fall   well dogs   stinky
     pick   poop   time

[4] cats  different  dogs  dont run  jump  much  rather  sleep    fairness dogs  sleep    unlike dog
s    lots  playing   cat   sit   couch      intruder   home   probably better    dog since felines
usually just run   bed  hide
[5] mountain climbing  fun    mountain  switerland    colorado hiking  iceland  fun
```

VIEW CORPUS (ALL DOCS) AS A MATRIX

View corpus as a document matrix

TMD stands for Term Document Matrix

(MyTDM <- TermDocumentMatrix(CleanCorpus))

inspect(MyTDM)

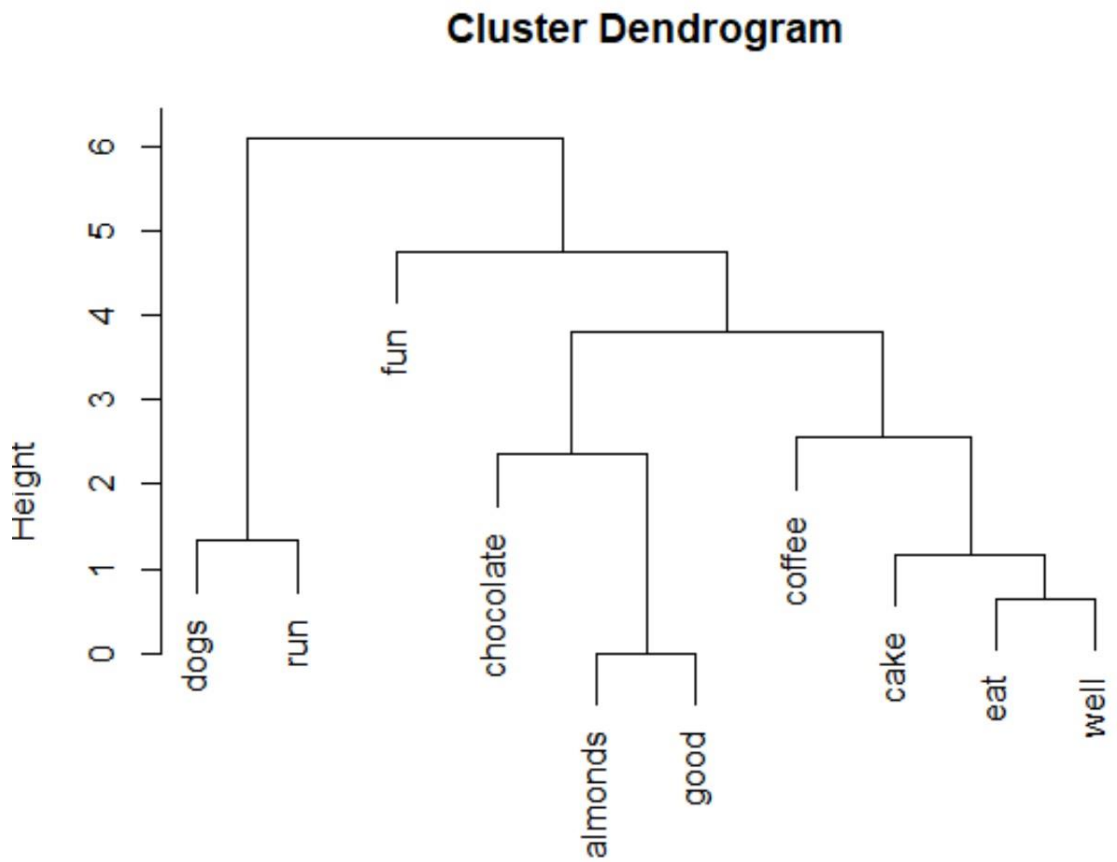
```
> inspect(MyTDM)
<<TermDocumentMatrix (terms: 61, documents: 5)>>
Non-/sparse entries: 70/235
Sparsity           : 77%
Maximal term length: 10
Weighting          : term frequency (tf)
Sample             :
           Docs
Terms       1 2 3 4 5
  almonds   3 0 0 0 0
  cake      1 1 0 0 0
  chocolate 4 2 0 0 0
  coffee    0 3 0 0 0
  dogs      0 0 3 3 0
  eat       1 1 1 0 0
  fun       0 0 1 0 2
  good      3 0 0 0 0
  run       0 0 2 2 0
  well      0 1 1 0 0
```

LOOK AT FREQUENCIES AND PLOT

```
## This will find words that occur more than 3
##times in the entire corpus

findFreqTerms(MyTDM, 1)

## Find assocations with aselected conf

findAssocs(MyTDM, 'coffee', 0.20)

## VISUALIZE

CleanDF <- as.data.frame(inspect(MyTDM))

(CleanDF)

CleanDFScale <- scale(CleanDF)

d <- dist(CleanDFScale,method="euclidean")

fit <- hclust(d, method="ward.D2")

plot(fit)
```

```
> findFreqTerms(MyTDM, 1)
 [1] "almonds"    "best"       "cake"       "chocolate"  "dark"       "day"        "eat"
 [8] "every"      "everything" "good"       "make"       "yummy"      "coffee"     "milk"
[15] "one"        "soy"        "together"   "well"       "danger"     "dogs"       "fall"
[22] "fun"        "help"       "jump"       "pick"       "poop"       "protect"    "run"
[29] "stinky"     "things"     "time"       "bed"        "better"     "cat"        "cats"
[36] "couch"      "different"  "dog"        "dont"       "fairness"   "felines"    "hide"
[43] "home"       "intruder"   "just"       "lots"       "much"       "playing"    "probably"
[50] "rather"     "since"      "sit"        "sleep"      "unlike"     "usually"    "climbing"
[57] "colorado"   "hiking"     "iceland"    "mountain"   "switerland"
> ## Find assocations with aselected conf
> findAssocs(MyTDM, 'coffee', 0.20)
$`coffee`
     milk       one       soy  together      cake      well       eat chocolate
     1.00      1.00      1.00      1.00      0.61      0.61      0.41      0.25
```

FREQUENCY DENDROGRAM



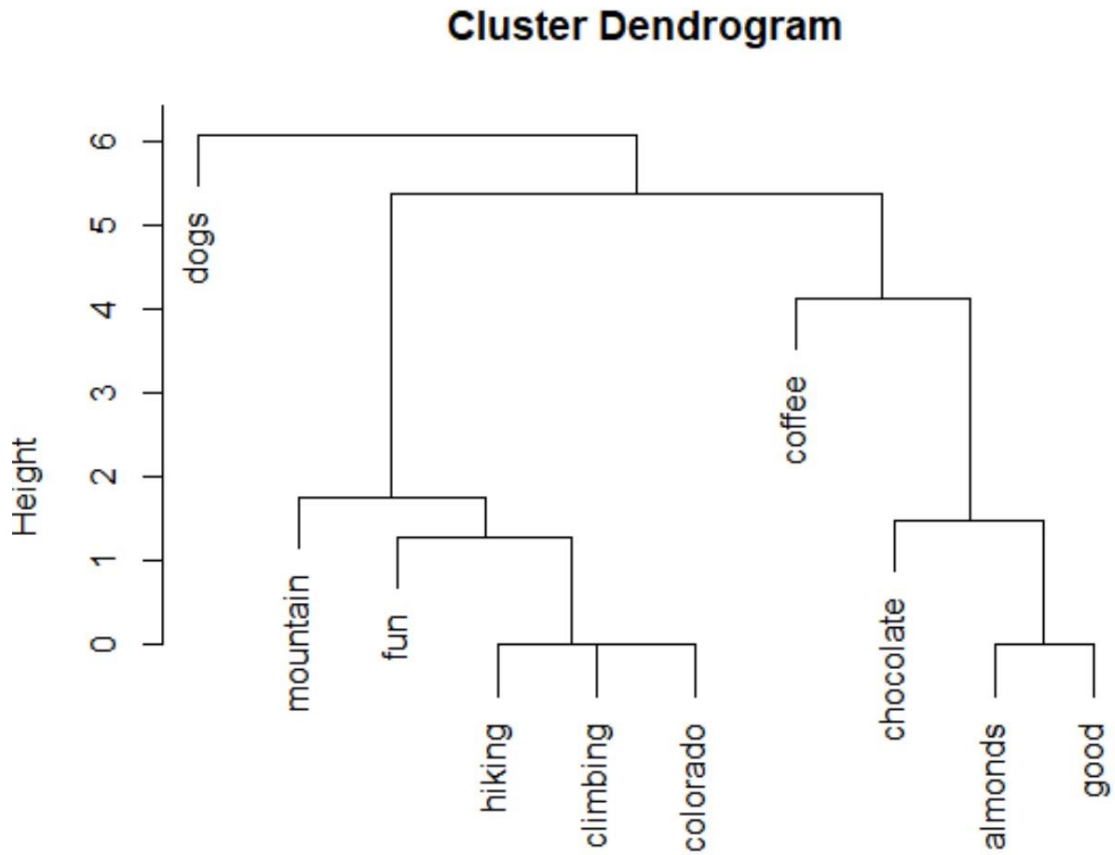
NORMALIZE AND RE-VISUALIZE

```
<<TermDocumentMatrix (terms: 61, documents: 5)>>
Non-/sparse entries: 70/235
Sparsity           : 77%
Maximal term length: 10
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
           Docs
Terms                 1         2          3         4         5
  almonds    0.3666202 0.0000000 0.00000000 0.0000000 0.0000000
  chocolate  0.2783007 0.2203213 0.00000000 0.0000000 0.0000000
  climbing   0.0000000 0.0000000 0.00000000 0.0000000 0.2579920
  coffee     0.0000000 0.5804820 0.00000000 0.0000000 0.0000000
  colorado   0.0000000 0.0000000 0.00000000 0.0000000 0.2579920
  dogs       0.0000000 0.0000000 0.22032135 0.1279285 0.0000000
  fun        0.0000000 0.0000000 0.07344045 0.0000000 0.2937618
  good       0.3666202 0.0000000 0.00000000 0.0000000 0.0000000
  hiking     0.0000000 0.0000000 0.00000000 0.0000000 0.2579920
  mountain   0.0000000 0.0000000 0.00000000 0.0000000 0.5159840
```

```
(MyDTM <- DocumentTermMatrix(CleanCorpus))

inspect(MyDTM)

## NOrmalize the Term Doc Matrix from above
##and then visualize it again

## Warning!! It is easy to mix up the DTM and
##the TDM- be carefull

NormalizedTDM <-
TermDocumentMatrix(CleanCorpus, control =
list(weighting = weightTfIdf, stopwords = TRUE))

inspect(NormalizedTDM)
```

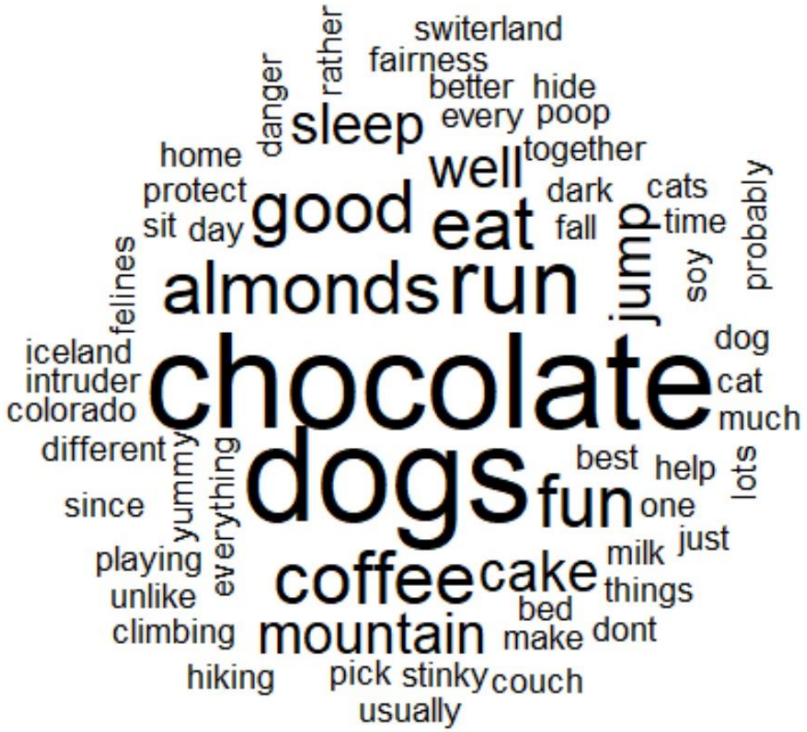
VISUALIZE THE NORMALIZED CORPUS

```
CleanDF_N <-
as.data.frame(inspect(NormalizedTDM))

CleanDFScale_N <- scale(CleanDF_N )

d <-
dist(CleanDFScale_N,method="euclidean")

fit <- hclust(d, method="ward.D2")

rect.hclust(fit, k = 4) # cut tree into 4
clusters

plot(fit)
```



THE WORDCLOUD

```
## Wordcloud

inspect(MyTDM)

m <- as.matrix(MyTDM)

(m)

#m <- as.matrix(CleanDF_N)

# calculate the frequency of words and sort it by frequency

word.freq <- sort(rowSums(m), decreasing = T)

wordcloud(words = names(word.freq), freq = word.freq*2,
min.freq = 2, random.order = F)
```



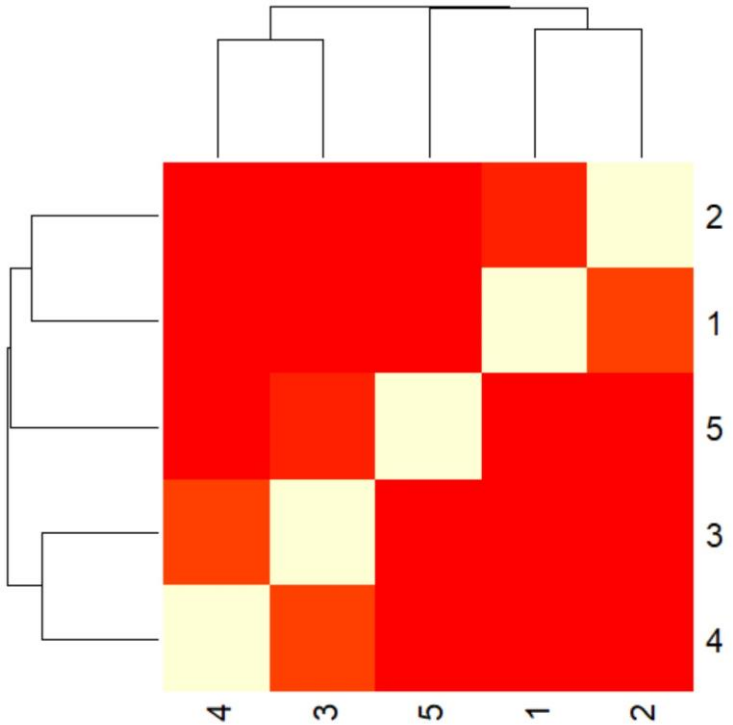
K MEANS CLUSTERING

```
## Use kmeans to cluster the documents
ClusterM <- t(m)
# transpose the matrix to cluster documents
(ClusterM)
set.seed(100) # set a fixed random seed
k <- 3 # number of clusters
kmeansResult <- kmeans(ClusterM, k)
(kmeansResult$cluster)
```

COSINE SIMILARITY

```
> (cos_sim_matrix <-(1 - cosine_dist_mat))
     Docs
Docs           1          2           3         4           5
   1 1.000000000 0.14062523 0.004817159 0.0000000 0.00000000
   2 0.140625230 1.00000000 0.028978970 0.0000000 0.00000000
   3 0.004817159 0.02897897 1.000000000 0.2137779 0.05344848
   4 0.000000000 0.00000000 0.213777918 1.0000000 0.00000000
   5 0.000000000 0.00000000 0.053448484 0.0000000 1.00000000
```

```
m2<-NormalizedTDM
inspect(NormalizedTDM)
str(m2)
cosine_dist_mat <-
  1 - crossprod_simple_triplet_matrix(m2)/
  (sqrt(col_sums(m2^2) %*% t(col_sums(m2^2))))
(cosine_dist_mat)
(cos_sim_matrix <-(1 - cosine_dist_mat))
heatmap(cos_sim_matrix)
## Most similar: 1 & 2,  and 3& 4
```



REFERENCES

1) My GU Slides

2) US Slides

3) Umich: https://cs.wmich.edu/alfuqaha/summer14/cs6530/lectures/ClusteringAnalysis.pdf

4) Kumar Data Mining Book

5) Stanford Infolab

http://infolab.stanford.edu/~ullman/mining/pdf/cs345-cl.pdf

6) https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v4.2.pdf

7)https://pdfs.semanticscholar.org/954f/ddb0c3d022cda2a7c3324e1391a4a3239d85.pdf

8) https://www.cl.cam.ac.uk/teaching/1314/InfoRtrv/lecture4.pdf

9) https://rstudio-pubs-static.s3.amazonaws.com/33876_1d7794d9a86647ca90c4f182df93f0e8.html

10) http://www.sthda.com/english/articles/30-advanced-clustering/105-dbscan-density-based-clustering-essentials/