


A decorative graphic on the left side of the slide. It features a solid red arrow pointing to the right, positioned horizontally. Behind the arrow and extending upwards and to the right are several thin, dark, curved lines that sweep across the frame, creating a sense of movement or flow.

CLUSTERING

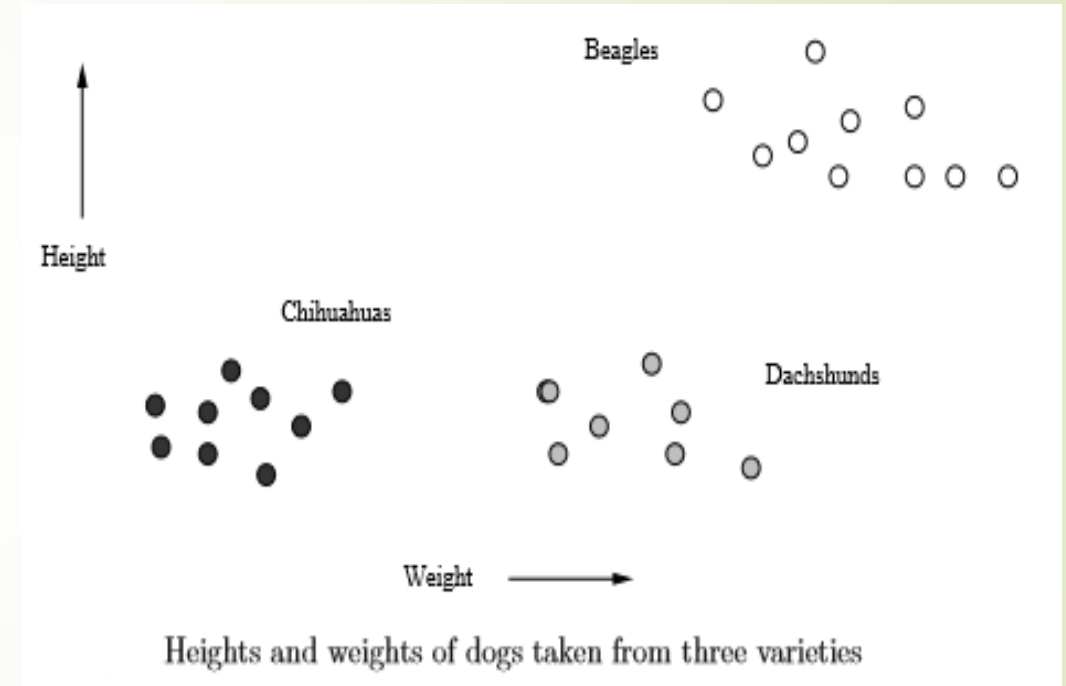


Introduction to Clustering

- Clustering is the process of examining a collection of “points,” and grouping the points into “clusters” according to some distance measure.
 - The goal is that points in the same cluster have a small distance from one another, while points in different clusters are at a large distance from one another.
- 

Classical examples of Clustering

- We have the height and weight measurements of dogs of several varieties.
- Without knowing which dog is of which variety, we can see just by looking at the diagram that the dogs fall into three clusters, and those clusters happen to correspond to three varieties.





Distance Measure

A set of points is said to be space. A distance between any two points is denoted by $d(x,y)$.

And it must satisfy the following properties---

i) $d(x,y) \geq 0$.

ii) $d(x,y) = 0$ iff $x = y$.

iii) $d(x,y) = d(y,x)$.

iv) $d(x,z) \leq d(x,y) + d(y,z)$.



Distances

- In Euclidean Space
 - Euclidean distance
 - Manhattan distance
 - L^∞ -distance
- In Non Euclidean Space
 - Jaccard distance
 - Cosine distance
 - Hamming distance
 - Edit distance

Euclidean Distance - Euclidean

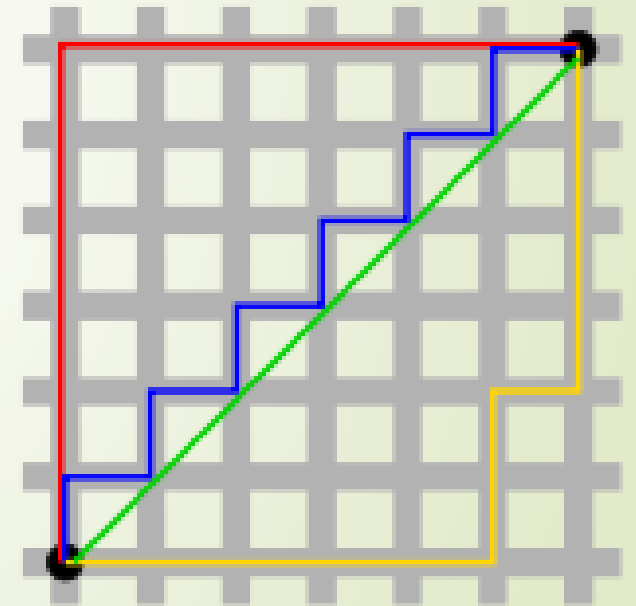
- **Euclidean distance** is the straight-line distance between two points in an Euclidean space.
- In Cartesian coordinates, if $\mathbf{p} = (p_1, p_2, \dots, p_n)$ and $\mathbf{q} = (q_1, q_2, \dots, q_n)$ are two points in Euclidean n -space, then the distance (d) from \mathbf{p} to \mathbf{q} , or from \mathbf{q} to \mathbf{p} is given by the Pythagorean formula:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.$$

Manhattan Distance - Euclidean

- ▶ The distance between two points in a grid based on a strictly horizontal and/or vertical path (i.e. along the grid lines).
- ▶ The Manhattan distance is the simple sum of the horizontal and vertical components
- ▶ In the diagram :
 - ▶ Blue, Red and Yellow lines : Manhattan distance
 - ▶ Green : Euclidean distance



L^∞ -distance - Euclidean

- **Chebyshev distance** or L_∞ metric is a metric defined on a vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension.
- The Chebyshev distance between two vectors or points p and q , with standard coordinates p_i and q_i , respectively, is

$$D_{\text{Chebyshev}}(p, q) := \max_i (|p_i - q_i|).$$

Jaccard Distance – Non Euclidean

- The **Jaccard index**, also known as the **Jaccard similarity coefficient** is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard Distance – Non Euclidean

- The **Jaccard distance**, which measures *dissimilarity* between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union:

$$d_J(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

Cosine Distance – Non Euclidean

- ▶ **Cosine similarity** is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.
- ▶ The cosine of two vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta$$

- ▶ Given two vectors of attributes, A and B , the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as:


$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Hamming Distance – Non Euclidean

- ▶ the **Hamming distance** between two strings of equal length is the number of positions at which the corresponding symbols are different.
- ▶ It measures the minimum number of *substitutions* required to change one string into the other, or the minimum number of *errors* that could have transformed one string into the other.
- ▶ Example : The Hamming distance between
 - ▶ "karolin" and "kathrin" is 3.
 - ▶ "karolin" and "kerstin" is 3.
 - ▶ 1011101 and 1001001 is 2.
 - ▶ 2173896 and 2233796 is 3.

Edit Distance – Non Euclidean

- ▶ **Edit distance** is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other.
- ▶ Given two strings a and b on an alphabet Σ (e.g. the set of ASCII characters, the set of bytes $[0..255]$, etc.), the edit distance $d(a, b)$ is the minimum-weight series of edit operations that transforms a into b .
- ▶ Simple set of Operations :
 - ▶ **Insertion** of a single symbol. If $a = uv$, then inserting the symbol x produces uxv . This can also be denoted $\epsilon \rightarrow x$, using ϵ to denote the empty string.
 - ▶ **Deletion** of a single symbol changes uxv to uv ($x \rightarrow \epsilon$).
 - ▶ **Substitution** of a single symbol x for a symbol $y \neq x$ changes uxv to uyv ($x \rightarrow y$).



Edit Distance – Non Euclidean

- Example :
- The edit distance between "kitten" and "sitting" is 3.
 - **k**itten → **s**itten (substitution of "s" for "k")
 - sitt**e**n → sitt**i**n (substitution of "i" for "e")
 - sittin → sittin**g** (insertion of "g" at the end).



Modern Clustering Problems

- Cluster documents by their topic, based on the occurrence of common, unusual words in the documents.
- Cluster moviegoers by the type or types of movies they like.



Classifications of Clustering

Strategies :

- ▀ Hierarchical
- ▀ Point-assignment

DataSets :

- ▀ Collection of points belonging to some space

Spaces :

- ▀ Euclidean
- ▀ Non Euclidean




Hierarchical Clustering

- Hierarchical or agglomerative algorithms start with each point in its own cluster. Clusters are combined based on their “closeness,” using one of many possible definitions of “close.” Combination stops when further combination leads to clusters that are undesirable for one of several reasons.
- For example, we may stop when we have a predetermined number of clusters, or we may use a measure of compactness for clusters, and refuse to construct a cluster by combining two smaller clusters if the resulting cluster has points that are spread out over too large a region.



Hierarchical Clustering in Euclidean Space

- ▶ We begin with every point in its own cluster. As time goes on, larger clusters will be constructed by combining two smaller clusters, and we have to decide in advance:
 - ▶ How will clusters be represented?
 - ▶ How will we choose which two clusters to merge?
 - ▶ When will we stop combining clusters?
- ▶ Algorithm:
 - WHILE it is not time to stop
 - DO pick the best two clusters to merge;
 - combine those two clusters into one cluster;
 - END;



Hierarchical Clustering in Non-Euclidean Spaces

- use some distance measure that is computed from points, such as Jaccard, cosine, or edit distance
- we cannot combine points in a cluster when the space is nonEuclidean, our only choice is to pick one of the points of the cluster itself to represent the cluster.
- Ideally, this point is close to all the points of the cluster, so it lies in the "center."
- We call the representative point the clustroid.



Selecting a clustroid

- We can select the clustroid in various ways by minimizing the distances between the clustroid and the other points in the cluster.
- Common choices include selecting as the clustroid the point that minimizes:
 - The sum of the distances to the other points in the cluster.
 - The maximum distance to another point in the cluster.
 - The sum of the squares of the distances to the other points in the cluster.

Example considering edit distances

Edit distances :

| | ecdab | abecb | aecdb |
|-------|-------|-------|-------|
| abcd | 5 | 3 | 3 |
| aecdb | 2 | 2 | |
| abecb | 4 | | |

Criteria for clustroid selection:

| Point | Sum | Max | Sum-Sq |
|-------|-----|-----|--------|
| abcd | 11 | 5 | 43 |
| aecdb | 7 | 3 | 17 |
| abecb | 9 | 4 | 29 |
| ecdab | 11 | 5 | 45 |

Here for every criteria aecdb will be selected as the clustroid. However different criteria could yield different clustroids.



Combining clusters

- merge the two clusters based on:
 - Proximity of clustroids.
 - Average or minimum distance between all pairs of points from the clusters.
 - Density of a cluster, based on the radius(if we take the clustroid to be the point with the smallest sum of squares of distances to the other nodes, then define the radius to be that sum of squares or its square root) or diameter (maximum distance between any two points in the cluster).

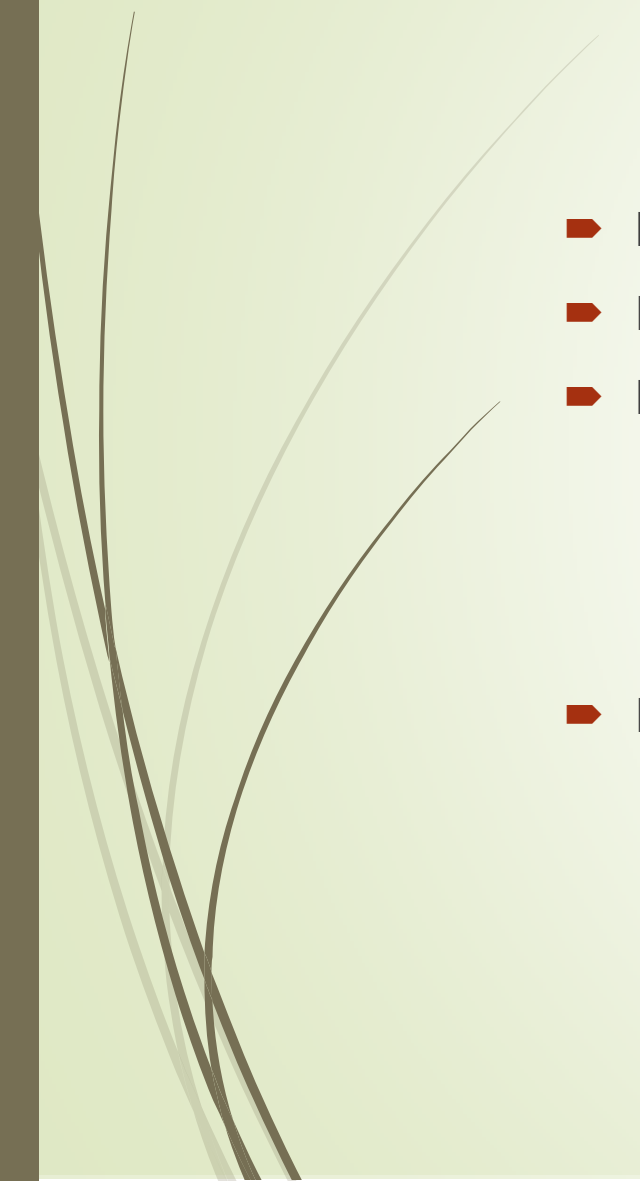


Point Assignment

- Points are considered in some order, and each one is assigned to the cluster into which it best fits. This process is normally preceded by a short phase in which initial clusters are estimated.
- Variations allow occasional combining or splitting of clusters, or may allow points to be unassigned if they are outliers (points too far from any of the current clusters).



K-Means Algorithm

- Initially choose k points that are likely to be in different clusters;
 - Make these points the centroids of their clusters;
 - FOR each remaining point p
 - DO find the centroid to which p is closest;
 - Add p to the cluster of that centroid;
 - Adjust the centroid of that cluster to account for p ;
 - END;
- 



Initializing Clusters

- We want to pick points that have a good chance of lying in different clusters.
- Approaches
 - Pick points that are as far away from one another as possible.
 - Cluster a sample of the data, perhaps hierarchically, so there are k clusters. Pick a point from each cluster, perhaps that point closest to the centroid of the cluster

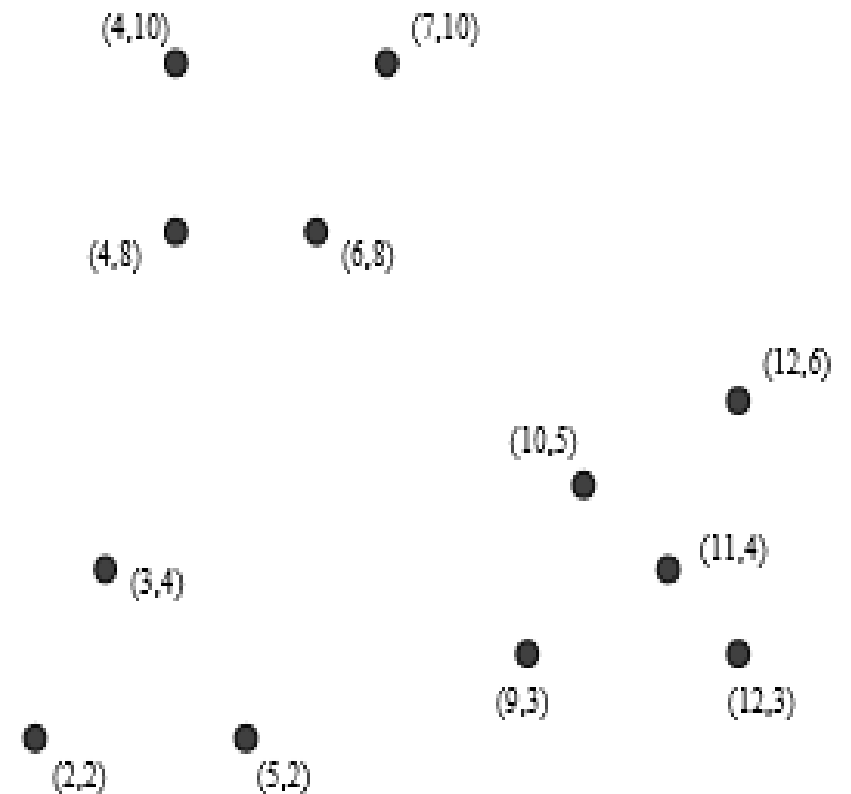


Selecting Points

- Pick the first point at random;
- WHILE there are fewer than k points
 - DO Add the point whose minimum distance from the selected points is as large as possible;
- END;


Example

- Our initial choice of a point is near the center, say (6,8).
- The furthest point from (6,8) is (12,3), so that point is chosen next.
- Among the remaining ten points, the one whose minimum distance to either (6,8) or (12,3) is a maximum is (2,2). That point has distance $\sqrt{52} = 7.21$ from (6,8) and distance $\sqrt{101} = 10.05$ to (12,3); thus its "score" is 7.21.
- Notice that these three belong to different clusters. Had we started with a different point, say (10,5), we would get a different set of three initial points. In this case, the starting points would be (10,5), (2,2), and (4,10). Again, these points belong to the three different clusters.





Choosing a proper value for 'k'

- We may not know the correct value of k to use in a k -means clustering. However, if we can measure the quality of the clustering for various values of k , we can usually guess what the right value of k is.
 - If we take a measure of appropriateness for clusters, such as average radius or diameter, that value will grow slowly, as long as the number of clusters we assume remains at or above the true number of clusters. However, as soon as we try to form fewer clusters than there really are, the measure will rise precipitously.
- 



Choosing a proper value for 'k'

- ▶ If we have no idea what the correct value of k is, we can find a good value in a number of clustering operations that grows only logarithmically with the true number.
- ▶ Begin by running the k -means algorithm for $k = 1, 2, 4, 8, \dots$.
- ▶ Eventually, we will find two values v and $2v$ between which there is very little decrease in the average diameter.
- ▶ We may conclude that the value of k that is justified by the data lies between $v/2$ and v .
- ▶ If we use a binary search in that range, we can find the best value for k in another $\log_2 v$ clustering operations, for a total of $2\log_2 v$ clusterings.
- ▶ Since the true value of k is at least $v/2$, we have used a number of clusterings that is logarithmic in k .




Conducting the binary search

- We know that there is too much change between $v/2$ and v , or else we would not have gone on to run a clustering for $2v$ clusters.
- Suppose at some point we have narrowed the range of k to between x and y .
- Let $z = (x + y)/2$.
- Run a clustering with z as the target number of clusters. If there is not too much change between z and y , then the true value of k lies between x and z .
- So recursively narrow that range to find the correct value of k .
- On the other hand, if there is too much change between z and y , then use binary search in the range between z and y instead.



K-Means Using MapReduce

- The **map** function performs the procedure of assigning each sample to the closest center
- The **reduce** function performs the procedure of updating the new centers.
- Continue iteratively until change in cluster centers is negligible




K-Means Using MapReduce : Map Function

- **Map-function** : The input dataset is stored on HDFS as a sequence file of points, each of which represents a record in the dataset. The dataset is split and globally broadcast to all mappers. Consequently, the distance computations are parallel executed. For each map task, PKMeans construct a global variable *centers* which is an array containing the information about centers of the clusters. Given the information, a mapper can compute the closest center point for each sample. The intermediate values are then composed of two parts: the index of the closest center point and the sample information.

K-Means Using MapReduce : Mapper Algorithm

- **Input:** Global variable *centers*, the samples
- **Output:** *<key', value' >* pair, where the *key'* is the index of the closest center point and *value'* is a string comprising of sample information
 - 1. Construct the sample *instance* from *value*;
 - 2. *minDis* = *Double.MAX VALUE*;
 - 3. *index* = -1;
 - 4. For *i*=0 to *centers.length* do
 - *dis*= *ComputeDist(instance, centers[i])*;
 - If *dis* < *minDis* {*minDis* = *dis*; *index* = *i*;}
 - 5. End For
 - 6. Take *index* as *key'*;
 - 7. Construct *value'* as a string comprising of the values of different dimensions;
 - 8. output *< key, value>* pair;
 - 9. End



K-Means Using MapReduce : Reduce Function

- **Reduce-function.** The input of the reduce function is the data obtained from the map function of each host. The data includes the index of the closest center point and the sample information. In reduce function, we can sum all the samples and compute the total number of samples assigned to the same cluster. Therefore, we can compute the new centers from the assigned samples which are used for next iteration.

K-Means Using MapReduce : Reducer Algorithm

- **Input:** key is the index of the cluster center, *values(V)* are the list of the samples from different host
- **Output:** < key , value > pair, where the *key'* is the index of the cluster, *value'* is a string representing the new center
- 1. Initialize one array record the sum of value of each dimensions of the samples contained in the same cluster, e.g. the samples in the list *V*;
- 2. Initialize a counter *NUM* as 0 to record the sum of sample number in the same cluster;
- 3. while(*V.hasNext()*){
 - Construct the sample *instance* from *V.next()*;
 - Add the values of different dimensions of *instance* to the array
 - *NUM* += *num*;
- 4. Divide the entries of the array by *NUM* to get the new center's coordinates;
- 5. Take *key* as *key'*;
- 6. Construct *value'* as a string comprising of the *center's* coordinates;
- 7. output < *key*, *value* > pair;
- 8. End