CLUSTERING

Outline

- Clustering
  - Chapter 7 and 12 in Mining of Massive Datasets Textbook
  - Chapter 9 in Information Retrieval in Practice

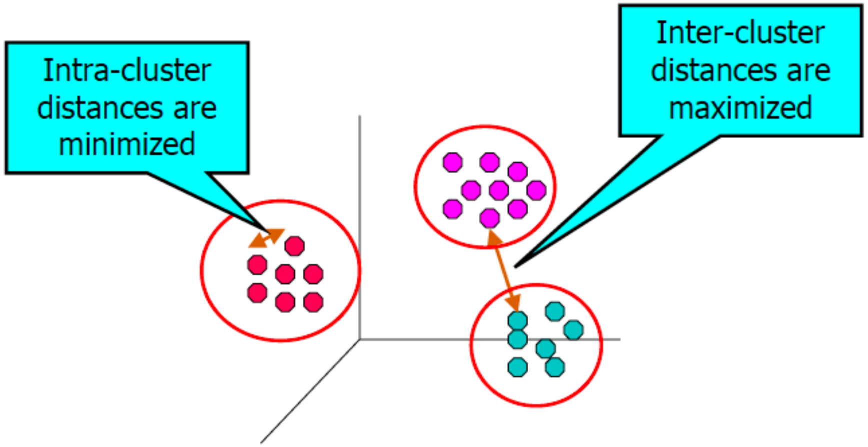
  - Today
    - General clustering (Kmeans)

  - Friday
    - LDA – Latent Dirichlet Allocation

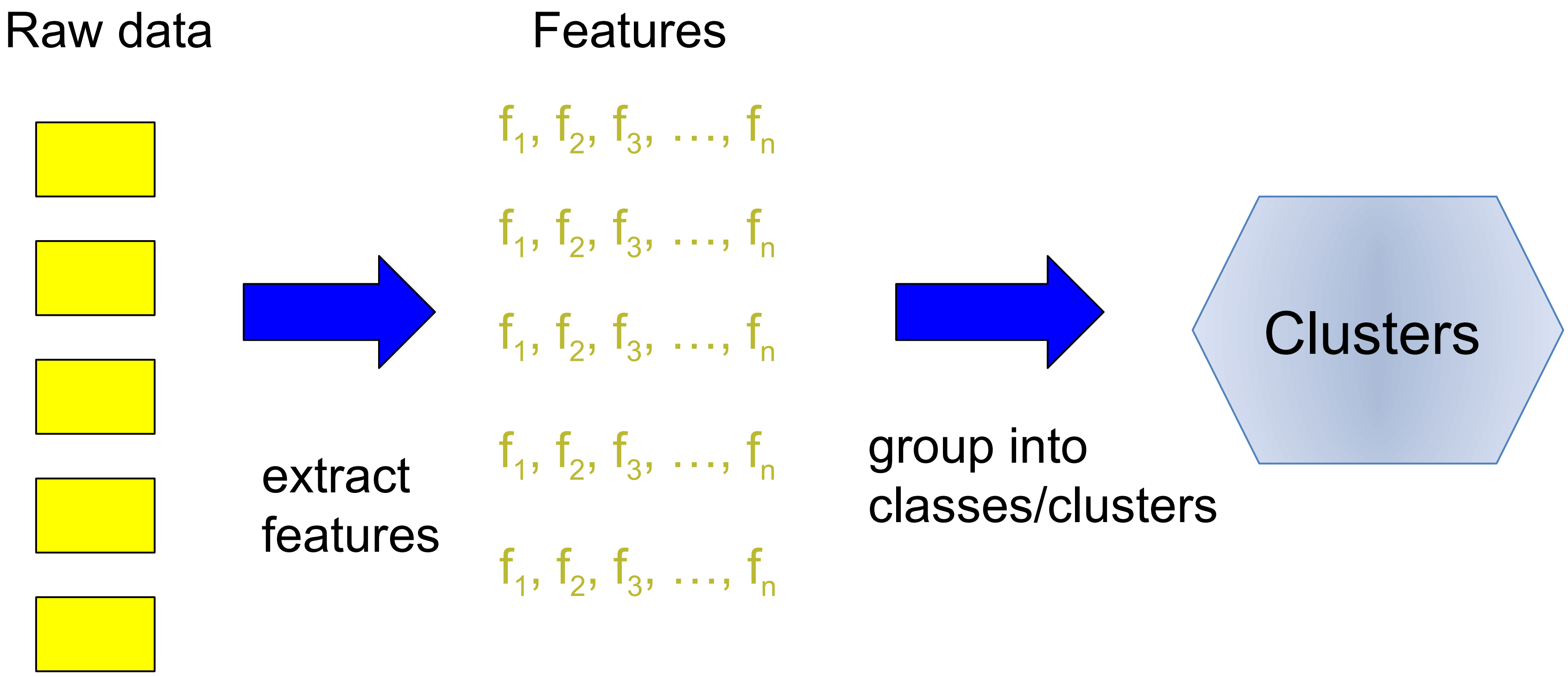
The Problem of Clustering

- Clustering is a technique for finding similarity groups in data, called clusters.
  - Groups data instances that are
    - Similar to (near) each other in one cluster, and
    - Very different (far away) from each other into different clusters.

- Clustering is an unsupervised learning task as no class values denoting an a priori grouping of the data instances are given, which is the case in supervised learning.



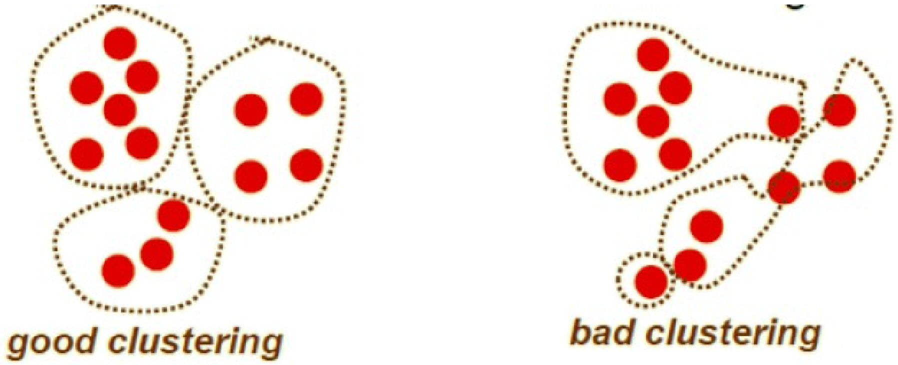
Unsupervised learning: clustering



No "supervision", we're only given data and want to find natural groupings

What do we need for clustering?

- Proximity measure, either
  - Similarity measure $s(x_1, x_2)$ - large if x_1,x_2 are similar
  - Dissimilarity (or distance) measure $d(x_1,x_2)$ – small if x_1,x_2 are similar
- Criterion function to evaluate a clustering



- Algorithm to compute clustering
  - For example, by optimizing the criterion function

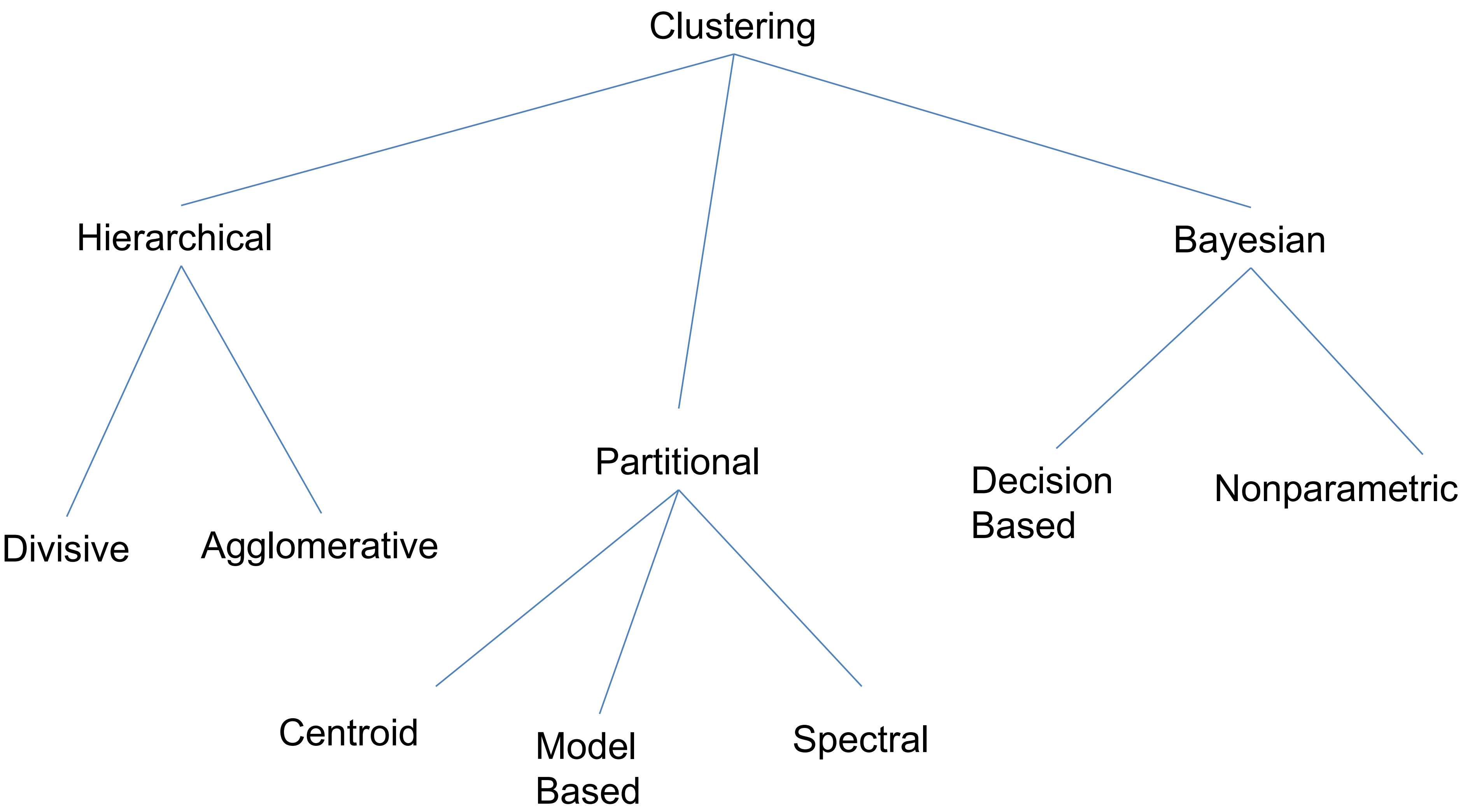
Distance (dissimilarity) measures

- Euclidean distance
  - $d(x_i, x_j) = \sqrt{\sum_{k=1}^{d}(x_i^{(k)} - x_i^{(k)})^2}$
- Manhattan (city block) distance
  - Approximation to Euclidean distance, cheaper to compute
  - $d(x_i, x_j) = \sum_{k=1}^{d} |x_i^{(k)} - x_j^{(k)}|$
- Minkowski distance (there are special cases)
  - \`$d_p(x_i, x_j)$=$(\sum_{k=1}^{m} |x_{ik} - x_{jk}|^p)^{\frac{1}{p}}$
  - P is a positive integer

Cluster evaluation (a hard problem)

- **Intra-cluster cohesion**(compactness):
  - Cohesion measures how near the data points in a cluster are to the cluster centroid.
  - Sum of squared error (SSE) is a commonly used measure.
- **Inter-cluster separation**(isolation):
  - Separation means that different cluster centroids should be far away from one another.
- In most applications, expert judgments are still the key

Clustering Techniques

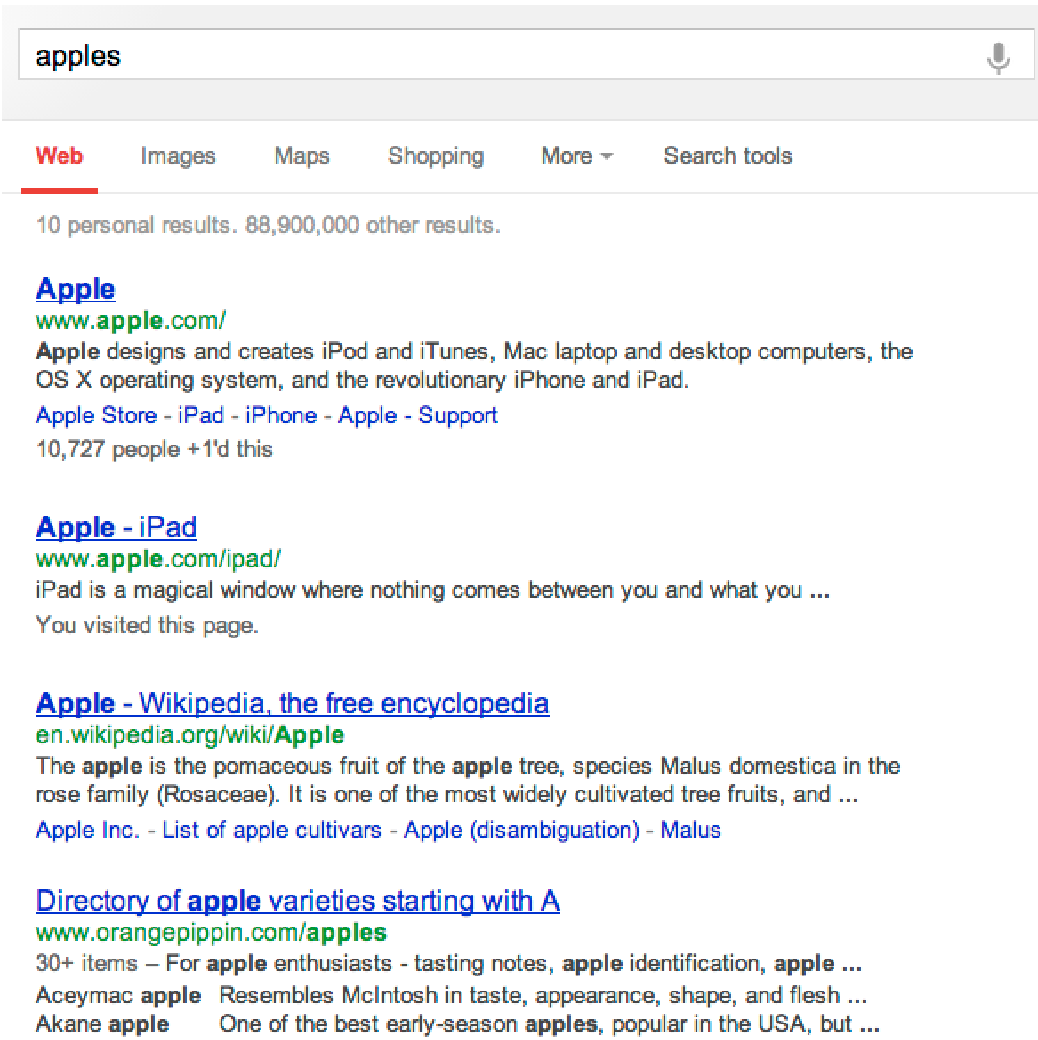


Applications of clustering

- *Understanding*
  - Grouping objects into conceptually meaningful classes is an important step in analysis.
  - EX: Group related documents for browsing, group genes and proteins that have similar functionality, or group stocks with similar price fluctuations.

- *Summarization*
  - Reduce the size of large data sets
  - EX: PCA requires space complexity $O(m^2)$ so may not be practice for large datasets. We can cluster dataset and apply PCA on the cluster prototypes.

Applications (clustering for IR)



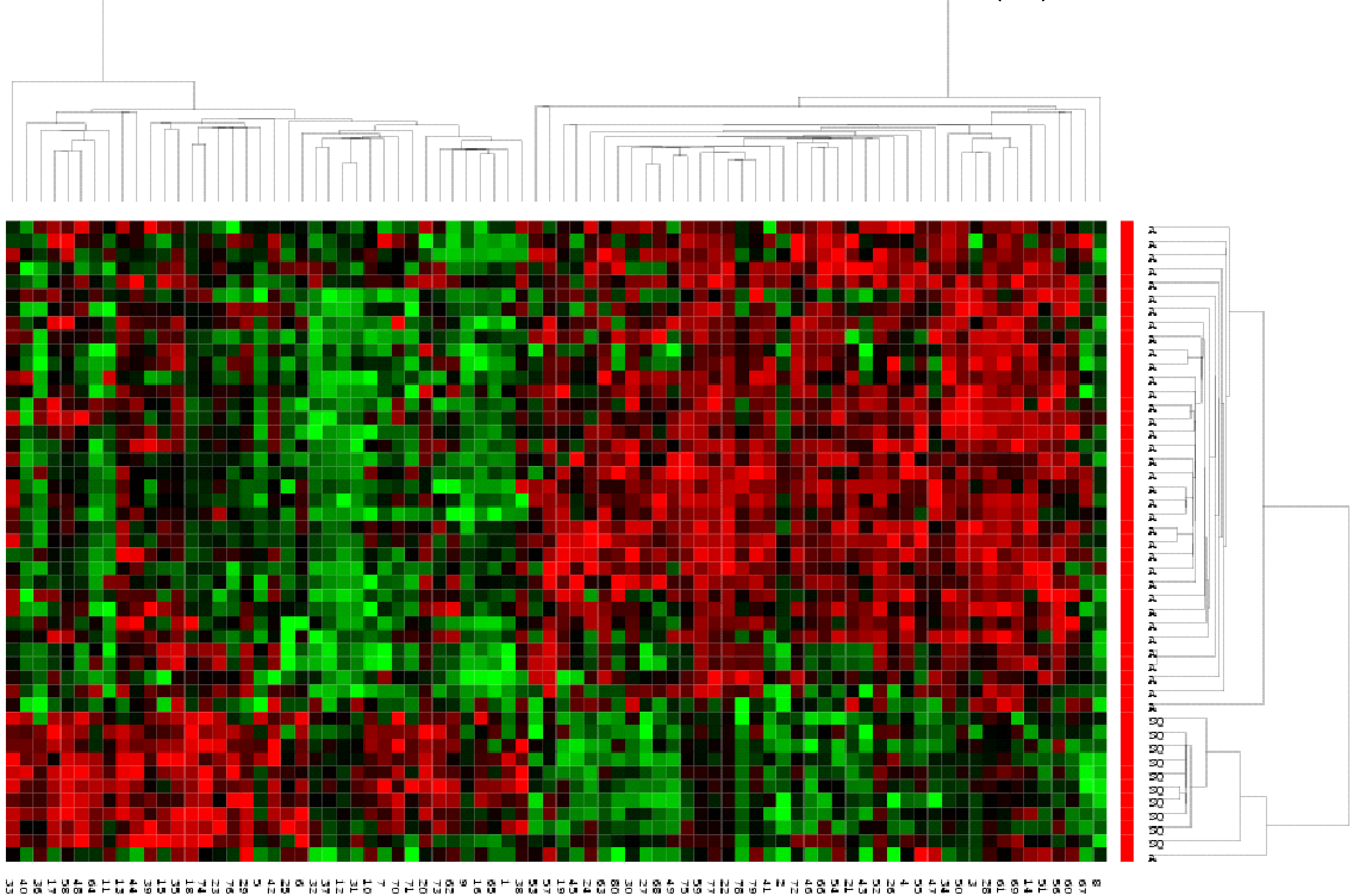
Documents or webpages in the same cluster are likely to be similar.

Application – Clustering for image search

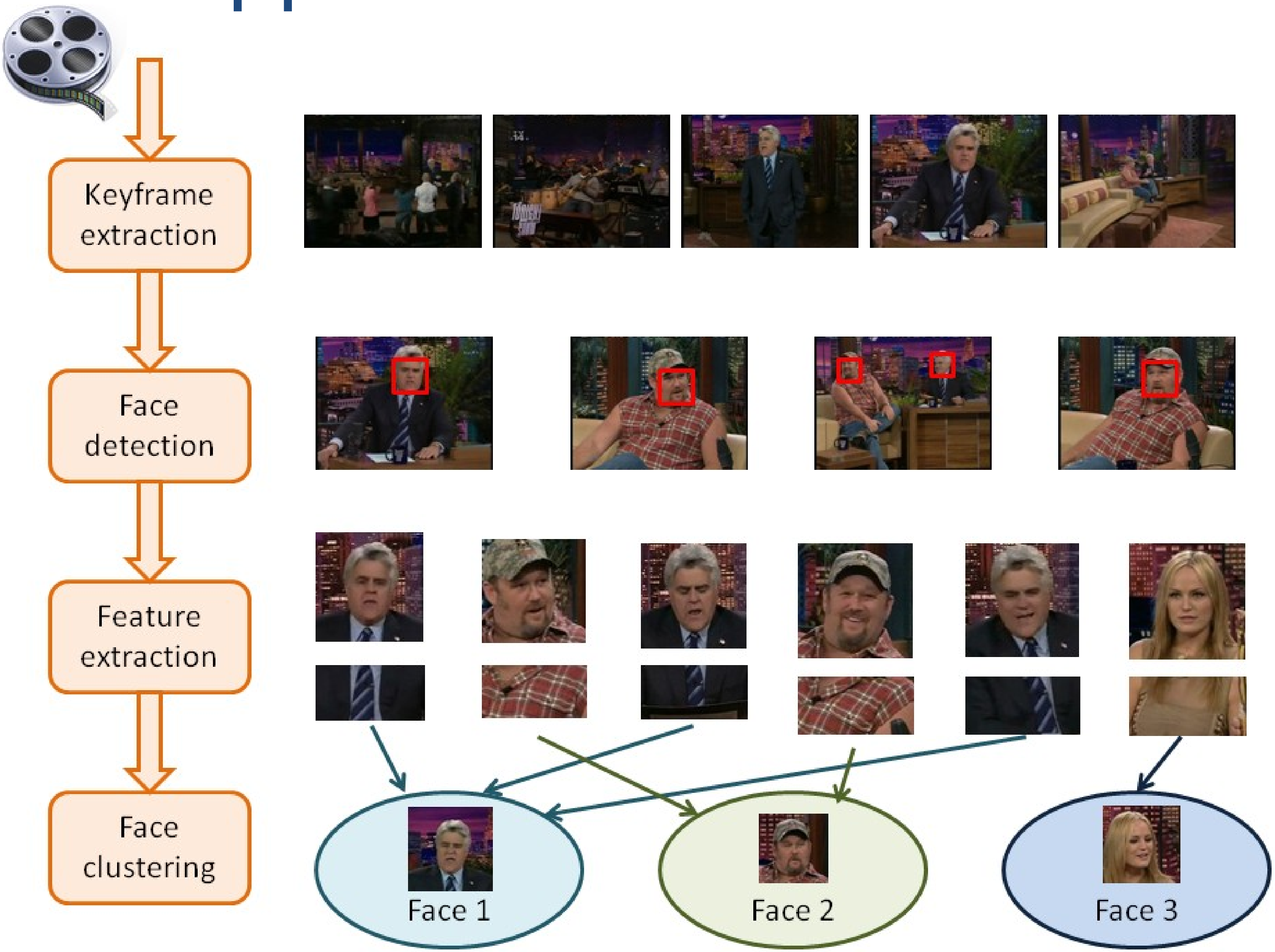
Given a collection of unlabeled objects(images), cluster them into groups.
So that given an image, we call pull similar images from the right cluster.

Gene expression data

Data from Garber et al.
PNAS (98), 2001.

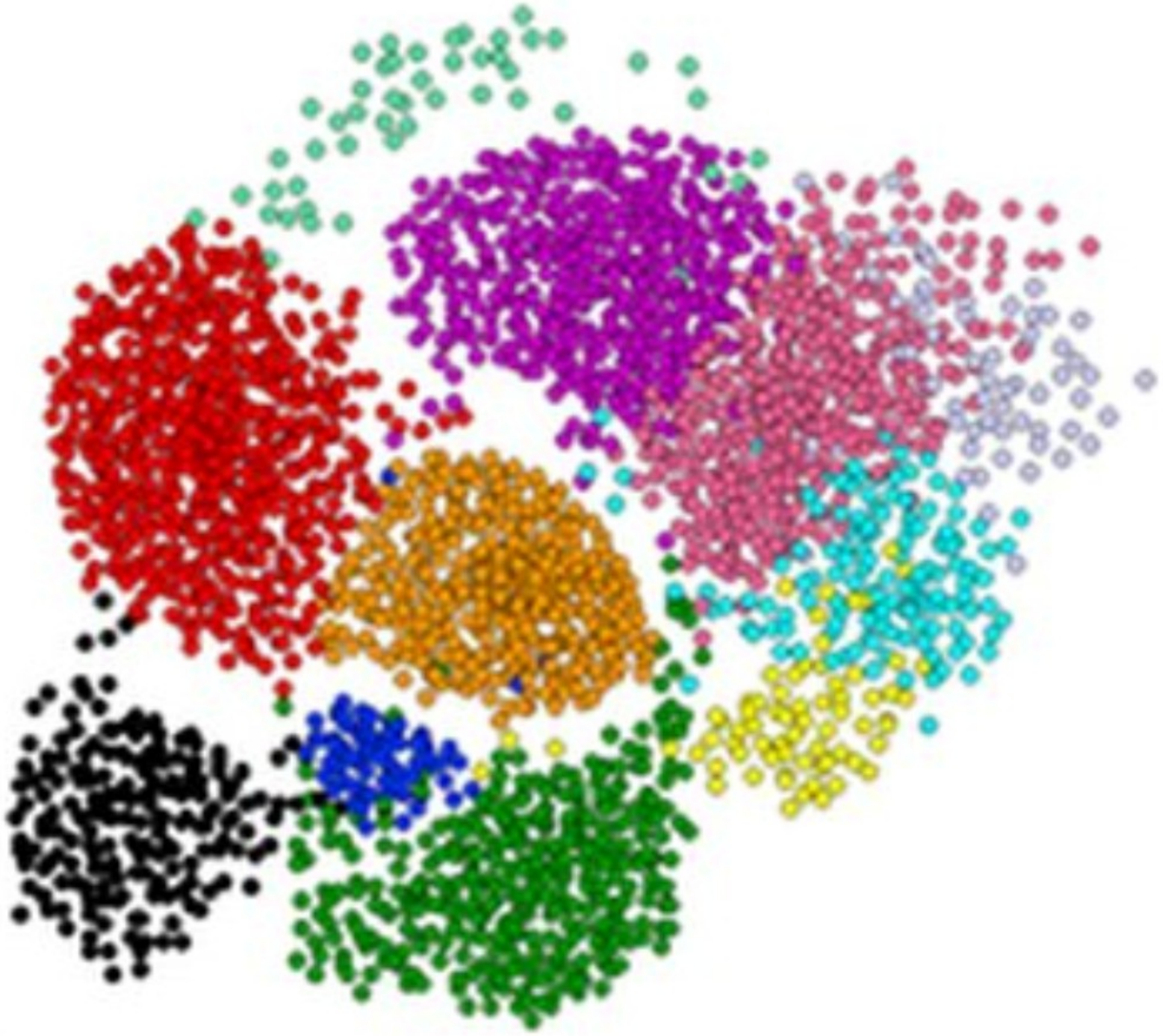


Applications - Face Clustering



Applications - Face clustering

Clustering is a hard problem!

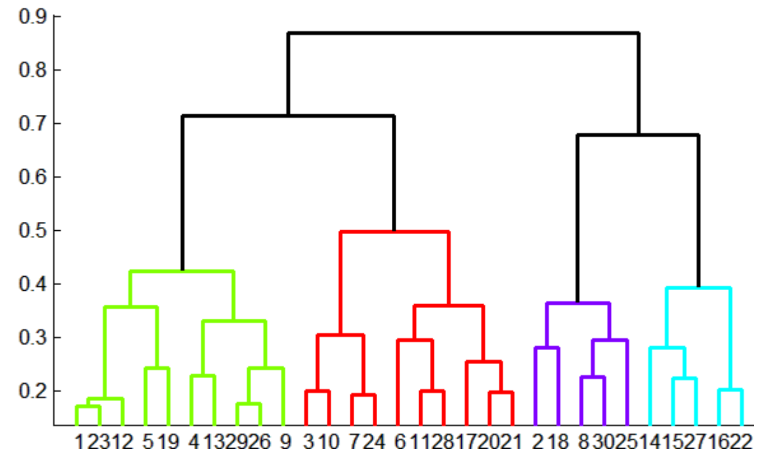


Why is it hard?

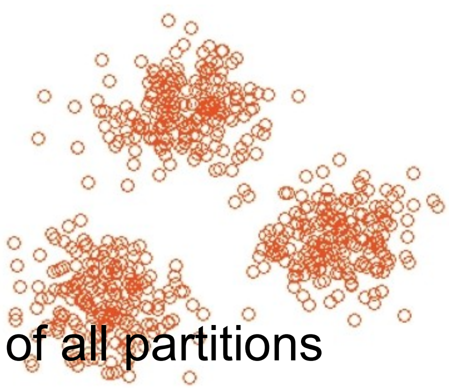
- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving

- Many applications involve not 2, but 10 or 10,000 dimensions
- High-dimensional spaces look different: Almost all pairs of points are at about the same distance

Overview: Methods of Clustering



- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it
- **Partitional**
  - Usually start with a random (partial) partitioning
  - Refine it iteratively
    - *K* means clustering
    - Model based clustering
- **Bayesian**
  - Try to generate a posteriori distribution over the collection of all partitions of the data.



Hard vs. soft clustering

Hard clustering: Each example belongs to exactly one cluster

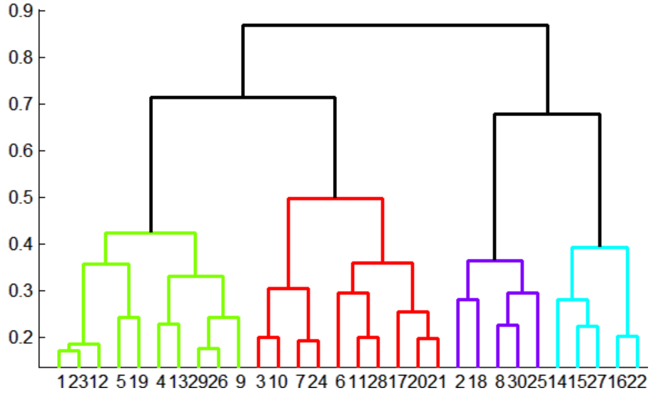
Soft clustering: An example can belong to more than one cluster (probabilistic)

- Makes more sense for applications like creating browsable hierarchies
- You may want to put a pair of sneakers in two clusters: (i) sports apparel and (ii) shoes

Hierarchical Clustering

- Key operation: Repeatedly combine two nearest clusters

- Three important questions:
  - 1) How do you represent a cluster of more than one point?
  - 2) How do you determine the “nearness” of clusters?
  - 3) When to stop combining clusters?



Hierarchical Clustering

- Key operation: Repeatedly combine two nearest clusters

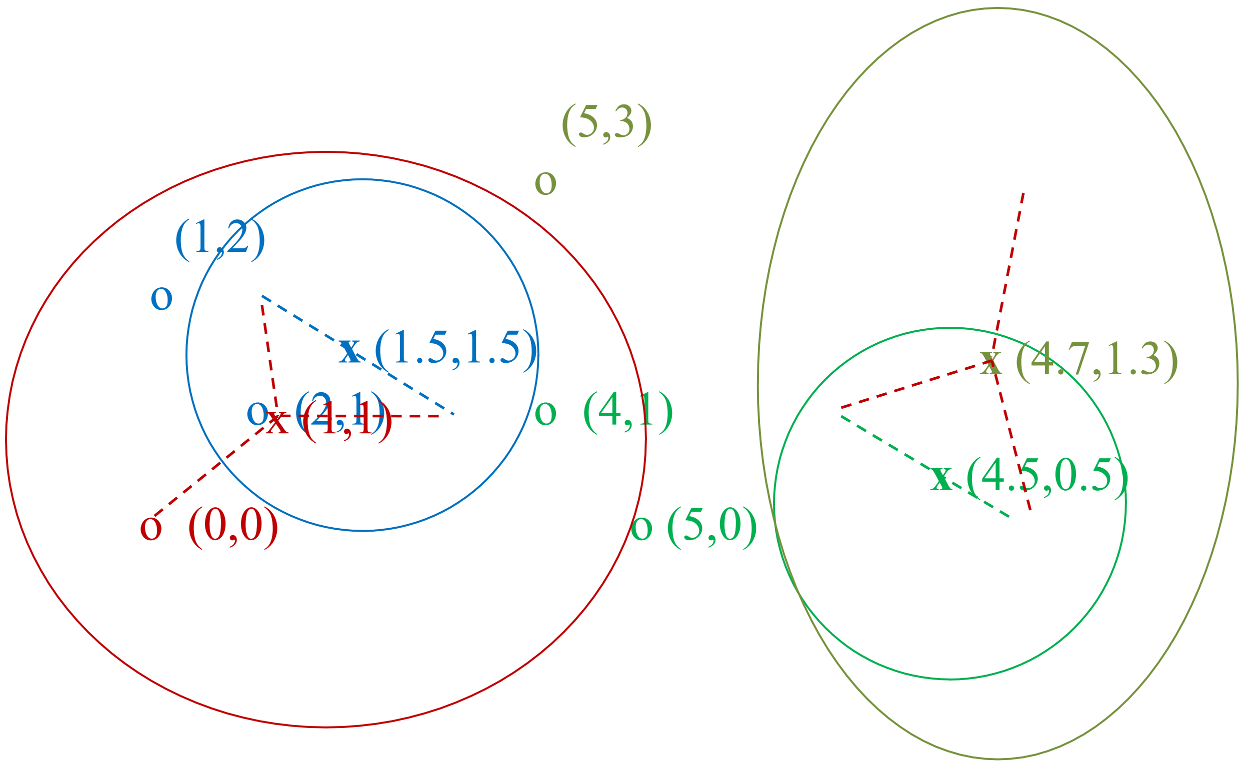
(1) How to represent a cluster of many points?

- Key problem: As you merge clusters, how do you represent the "location" of each cluster, to tell which pair of clusters is closest
- Euclidean case: each cluster has a centroid = average of its (data)points

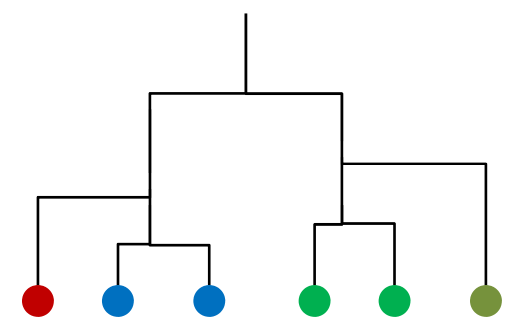
(2) How to determine "nearness" of clusters?

- Measure cluster distances by distances of centroids

Example: Hierarchical clustering



Data:
o … data point
x … centroid



Dendrogram

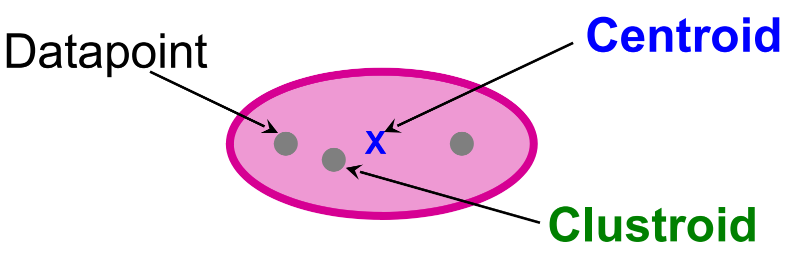
And in the Non-Euclidean Case?

- What about the Non-Euclidean case?
  - The only "locations" we can talk about are the points themselves
    - i.e., there is no "average" of two points
- Approach 1:
  - (1) How to represent a cluster of many points?
    clustroid = (data)point "closest" to other points
  - (2) How do you determine the "nearness" of clusters? Treat clustroid as if it were centroid, when computing inter-cluster distances

"Closest" Point?

- (1) How to represent a cluster of many points?
  clustroid = point "closest" to other points
- Possible meanings of "closest":
  - Smallest maximum distance to other points
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points
    - For distance metric d clustroid c of cluster C is:

$$\min_{c} \sum_{x \in C} d(x,c)^2$$



Cluster on
3 datapoints

Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.
Clustroid is an **existing** (data)point that is "closest" to all other points in the cluster.

Implementation

- Naïve implementation of hierarchical clustering:
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - $O(N^3)$
- Careful implementation using priority queue can reduce time to $O(N^2 \log N)$
  - Still too expensive for really big datasets that do not fit in memory

K-MEANS CLUSTERING

Demo of K-means clustering :
http://www.onmyphd.com/?p=kmeans.clustering&ckattempt=2

K-means

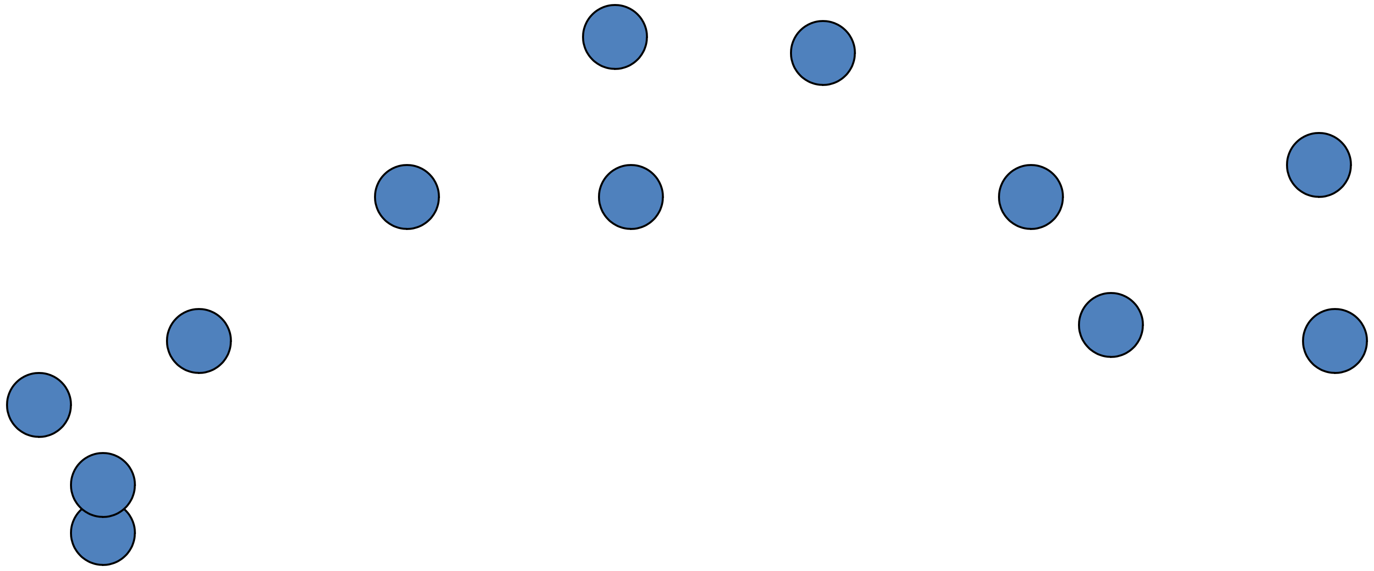
- Given *k*, the *k-means* algorithm works as follows:

1. Choose *k*(random) data points (seeds) to be the initial centroids, cluster centers
2. Assign each data point to the closest centroid
3. Re-compute the centroids using the current cluster memberships
4. If a convergence criterion is not met, repeat steps 2 and 3

K-means convergence (stopping) criterion

- No (or minimum) re-assignments of data points to different clusters, *or*
- No (or minimum) change of centroids, or
- Minimum decrease in the **sum of squared error**(SSE),
  - $SSE = \sum_{j=1}^{k} \sum_{x \in C_j} d(x, m_j)^2$
  - C_j is the j^{th} cluster,
  - $\mathbf{m}_j$ is the centroid of cluster C_j (the mean vector of all the data points in C_j),
  - $d(\mathbf{x}, \mathbf{m}_j)$ is the (Euclidian) distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$

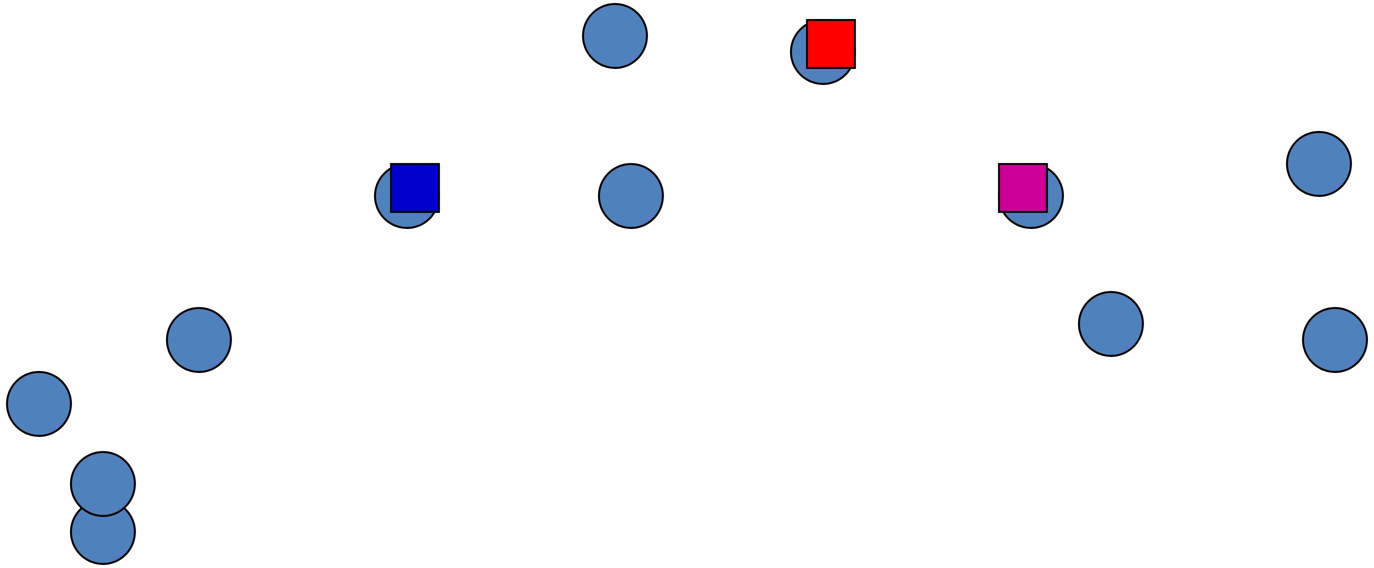
K-means: an example



Randomly Initialize the cluster centers

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster



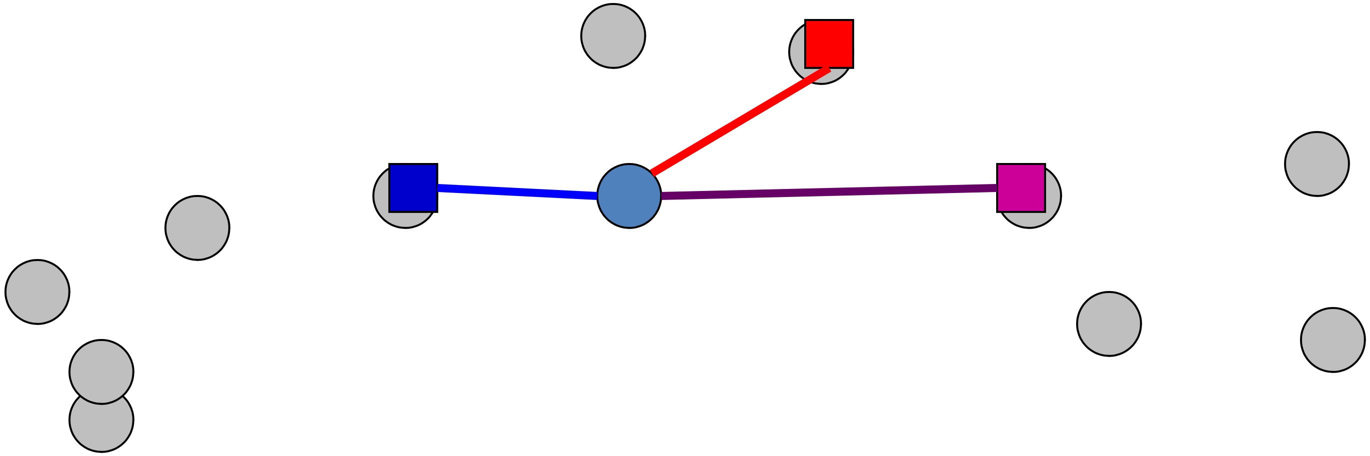
Iteration # 1

- **Assign each item to closest cluster center**

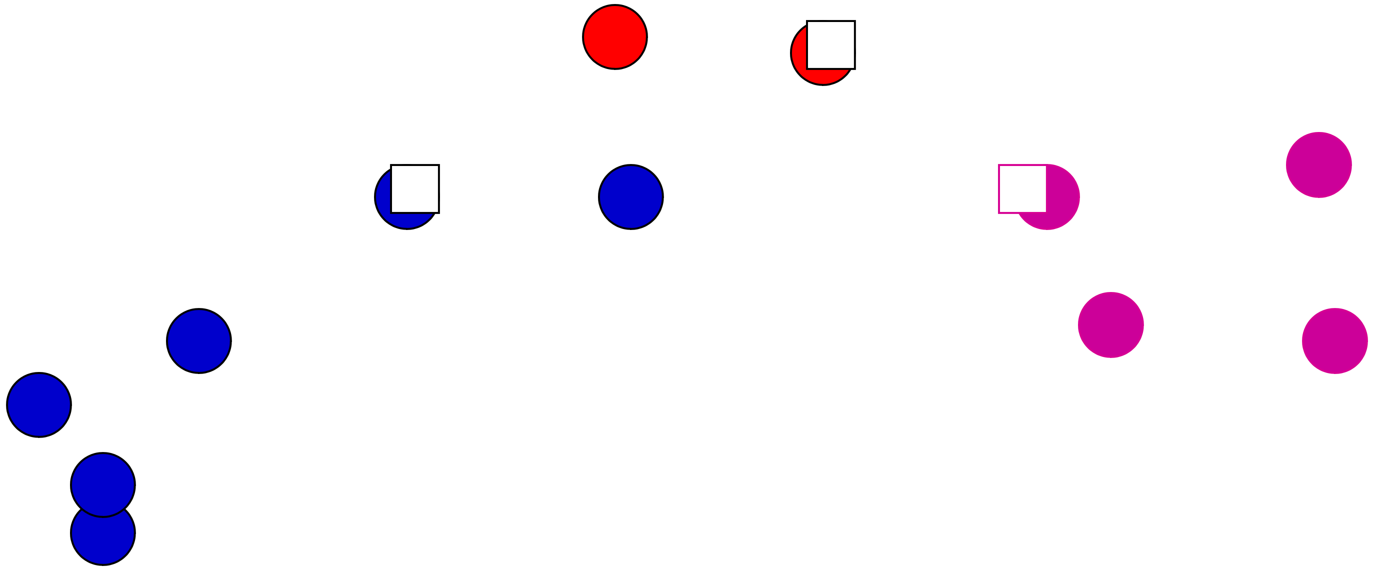
  iterate over each point:

  - get distance to each cluster center
  - assign to closest center (hard cluster)

- Recalculate centers as the mean of the points in a cluster

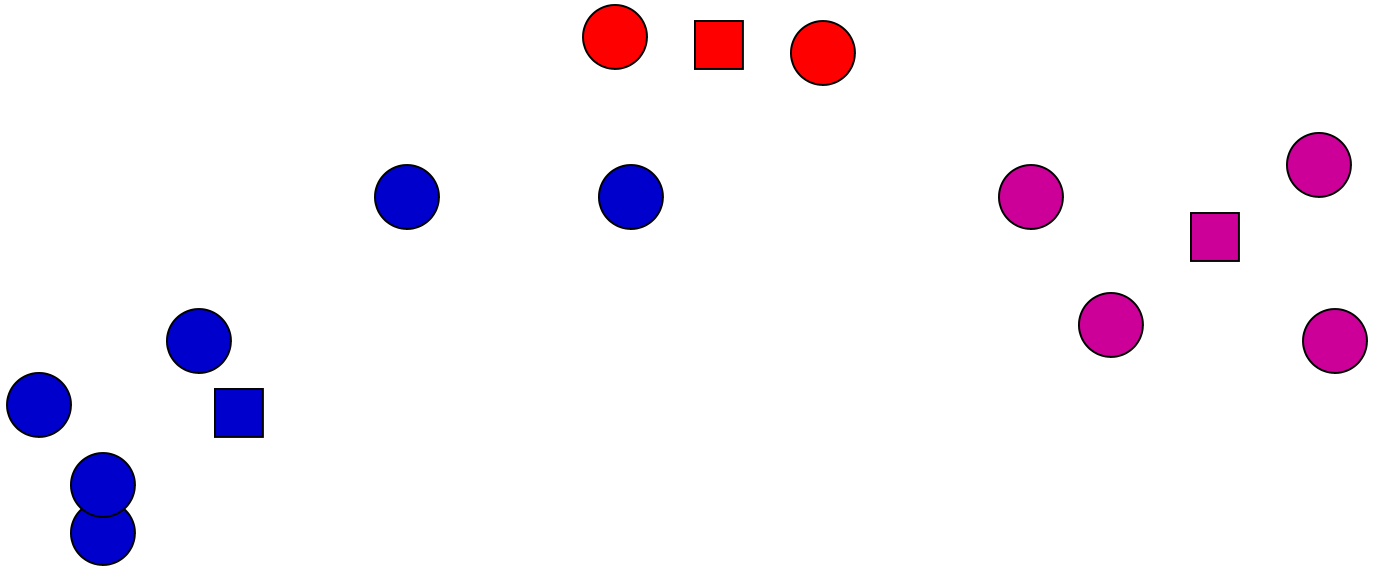


After assigning points to nearest center



Iteration # 1

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster



Distance measures

Euclidean:

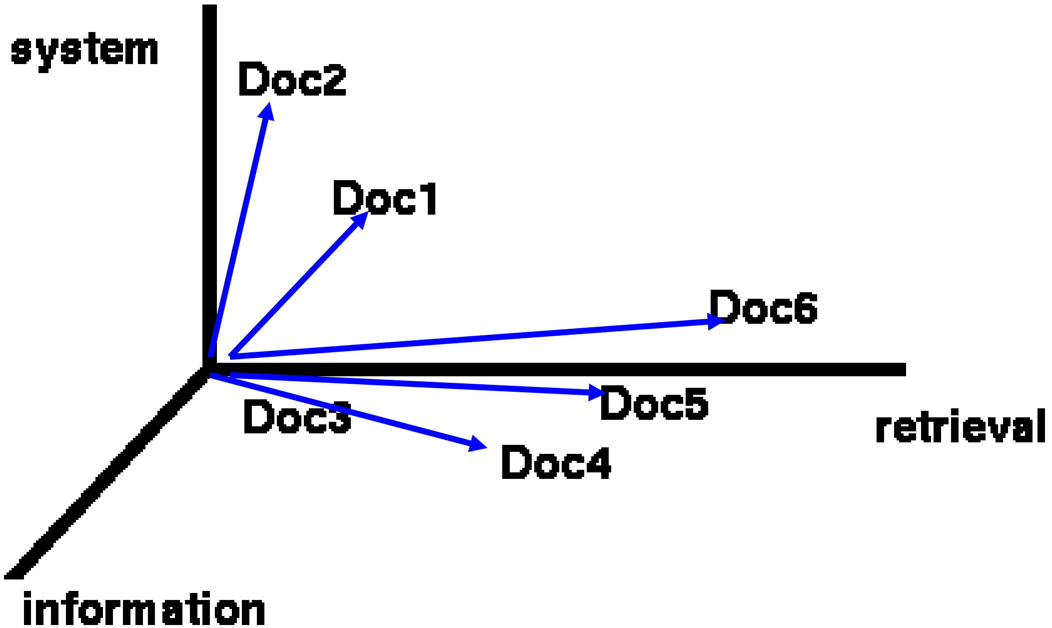
$$d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

good for spatial data

Clustering documents

One feature for each word. The value is the number of times that word occurs.

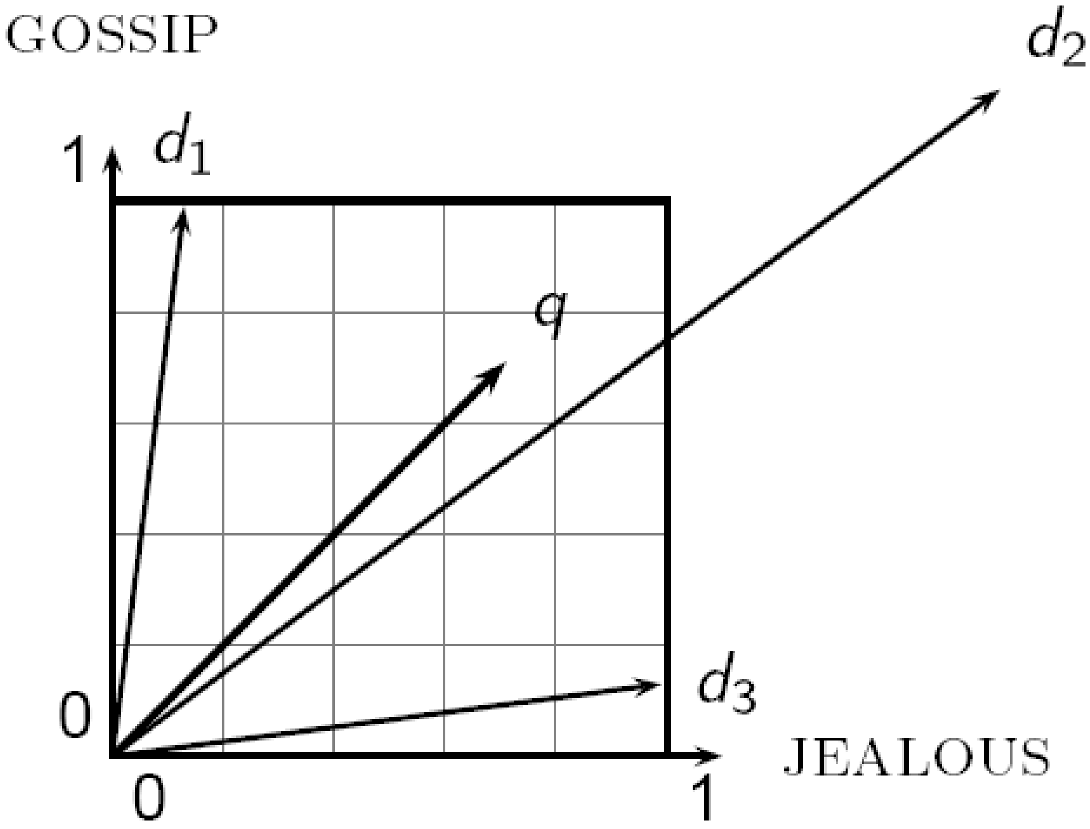
Documents are points or vectors in this space



When Euclidean distance doesn't work

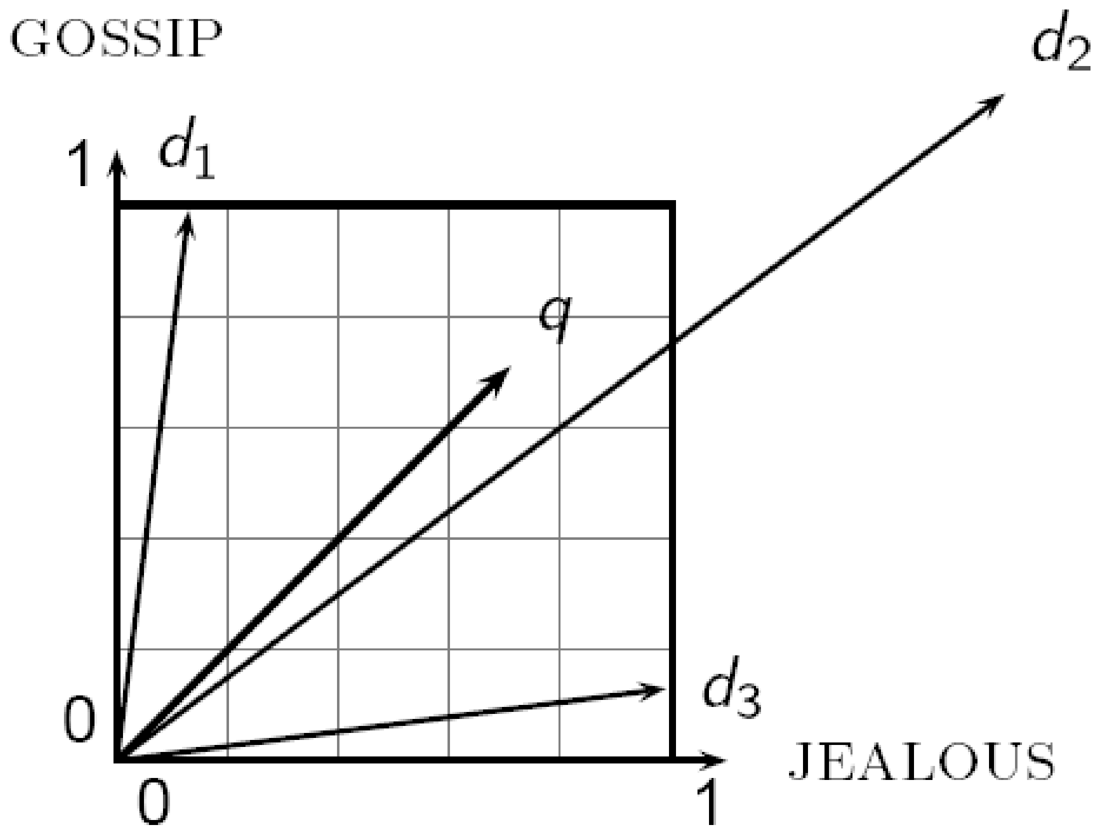
Which document is closest to q using Euclidian distance?

Which do you think should be closer?



Issues with Euclidian distance

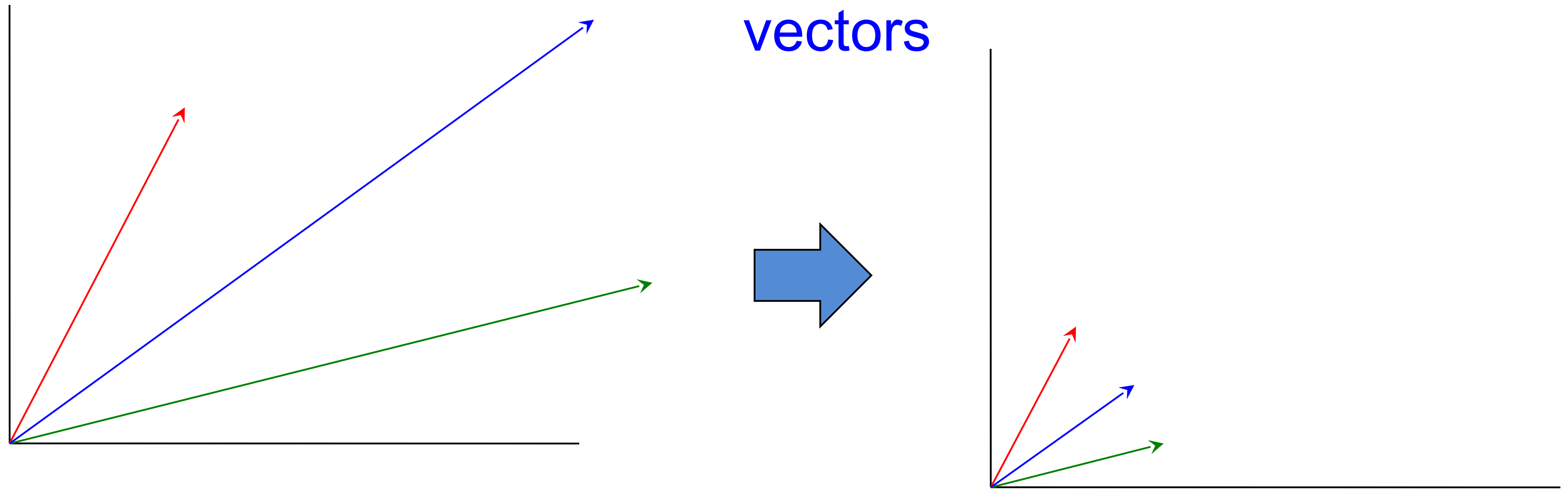
- the Euclidean distance between *q* and d_2 is large
- but, the distribution of terms in the query *q* and the distribution of terms in the document d_2 are very similar
- This is not what we want!



cosine similarity

$$sim(x,y) = \frac{x \bullet y}{|x||y|} = \frac{x}{|x|} \bullet \frac{y}{|y|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}}$$

correlated with the angle between two vectors



cosine distance

cosine similarity is a similarity between 0 and 1, with things that are similar 1 and not 0

We want a distance measure, cosine distance:

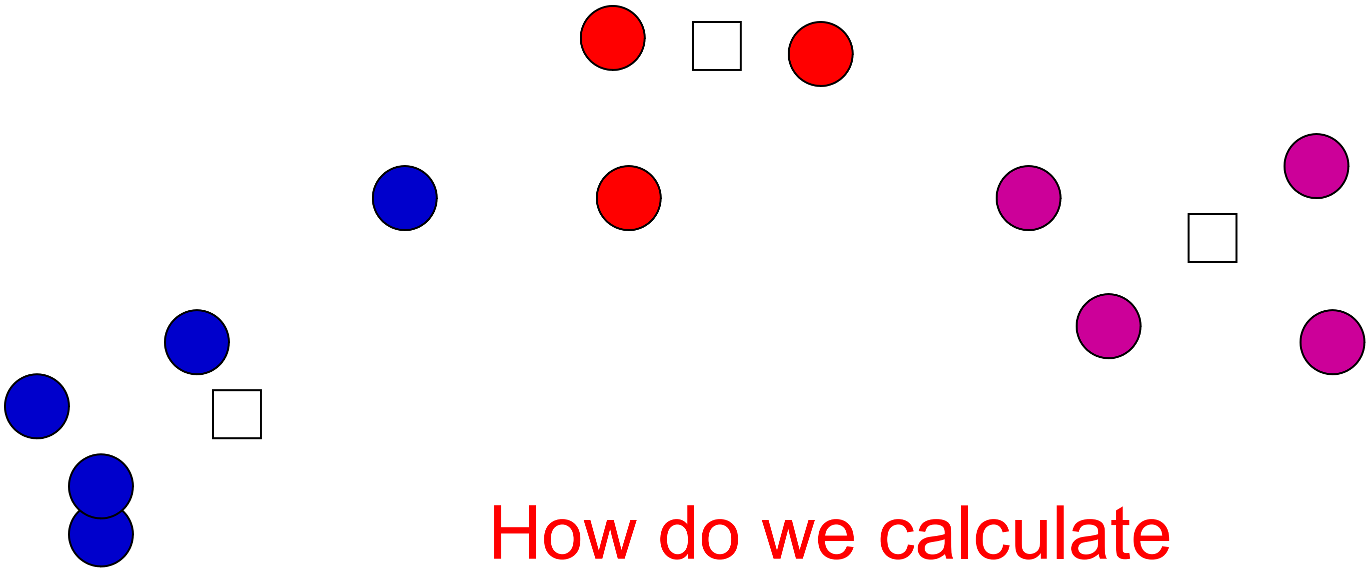
$$d(x,y) = 1 - sim(x,y)$$

- good for text data and many other “real world” data sets
- is computationally friendly since we only need to consider features that have non-zero values in **both** examples

K-means

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster

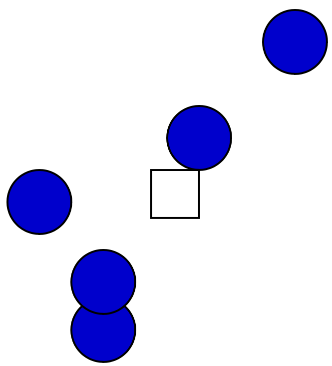


How do we calculate these?

K-means

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster



Mean of the points in the cluster:

$$\mu(\mathrm{C}) = \frac{1}{|C|} \sum_{x \in C} x$$

where:

$$x + y = \sum_{i=1}^{n} x_i + y_i \qquad \frac{x}{|C|} = \sum_{i=1}^{n} \frac{x_i}{|C|}$$

Comments on the K-Means Method

- Strength
  - Relatively efficient: O(tkn),
    - where n is # objects,
    - k is # clusters, and
    - t is # iterations.
    - Normally, k, t << n.
  - Often terminates at a local optimum.
- Weakness
  - Applicable only when mean is defined, then what about categorical data?
  - Need to specify k, the number of clusters, in advance
  - Unable to handle noisy data and outliers
  - Not suitable to discover clusters with non-convex shapes

K-means loss function

K-means tries to minimize what is called the "k-means" loss function:

$$loss = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i$$

that is, the <u>sum of the squared distances </u>from each point to the associated cluster center

Minimizing k-means loss

Iterate:

1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

$$loss = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i$$

Does each step of k-means move towards reducing this loss function (or at least not increasing)?

Minimizing k-means loss

Iterate:

1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

$$loss = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i$$

Intuition:

1. Any other assignment would end up in a larger loss
2. The mean of a set of values minimizes the squared error

Minimizing k-means loss

Iterate:

1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

$$loss = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i$$

Does this mean that k-means will always find the minimum loss/clustering?

Minimizing k-means loss

Iterate:

1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

$$loss = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i$$

NO! It will find *a minimum*.

Unfortunately, the k-means loss function is generally not convex and for most problems has many, many minimums

We're only guaranteed to find one of them

K-means variations/parameters

Start with some initial cluster centers

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster

What are some other variations/parameters we haven’t specified?

K-means variations/parameters

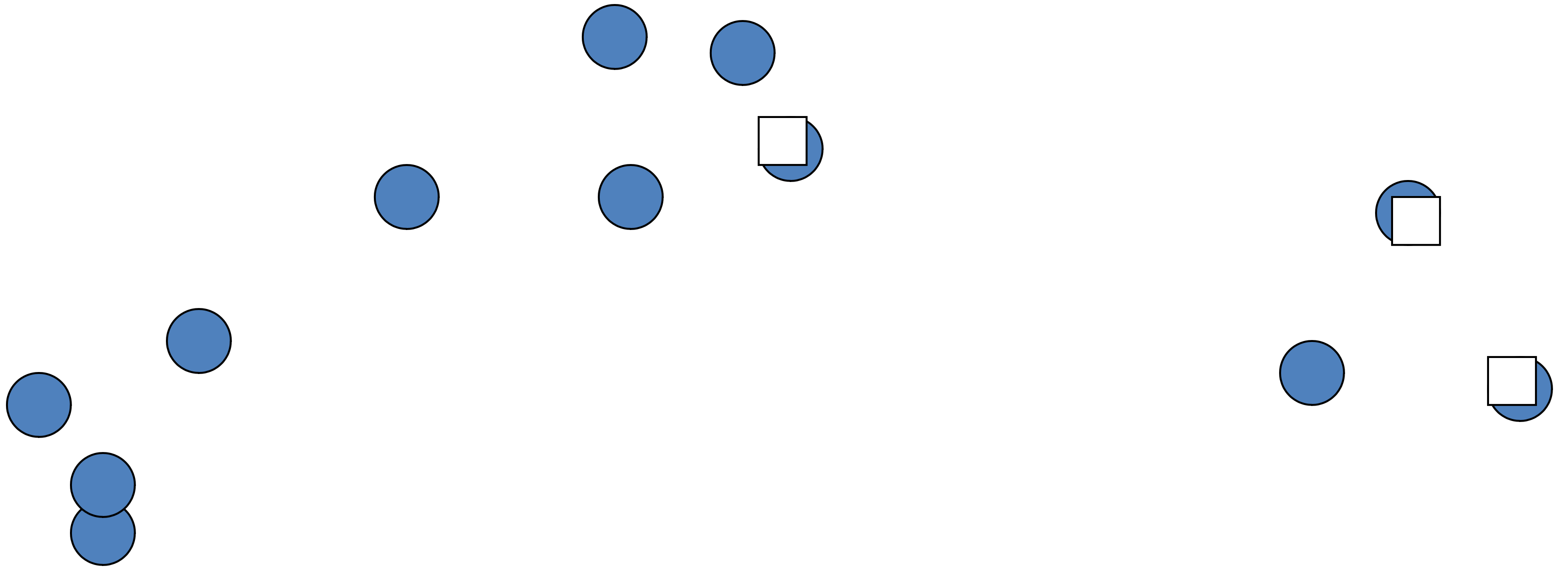
Initial (seed) cluster centers

Convergence

- A fixed number of iterations
- partitions unchanged
- Cluster centers don't change

K!

K-means: Initialize centers randomly



What would happen here?

Seed selection ideas?

Seed choice

Results can vary drastically based on random seed selection

Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings

Common heuristics

- Random centers in the space
- Randomly pick examples
- Points least similar to any existing center (furthest centers heuristic)
- **Try out multiple starting points**
- Initialize with the results of another clustering method

Furthest centers heuristic

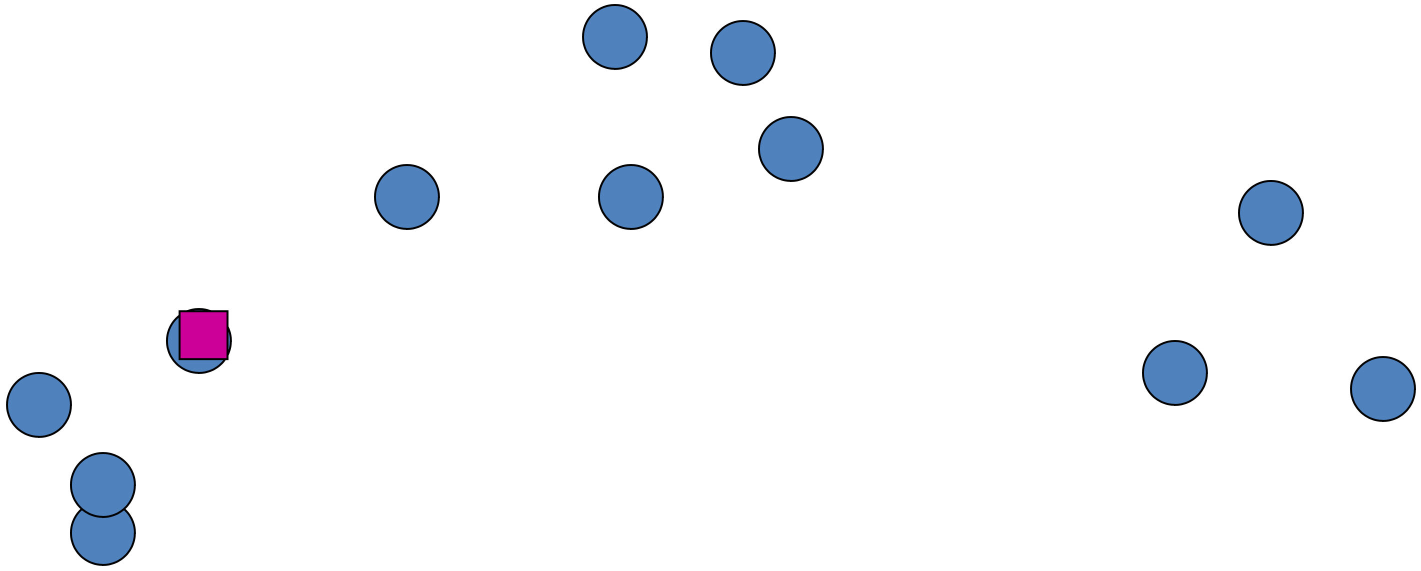
μ_1 = pick random point

for i = 2 to K:

μ_i = point that is furthest from **any** previous centers

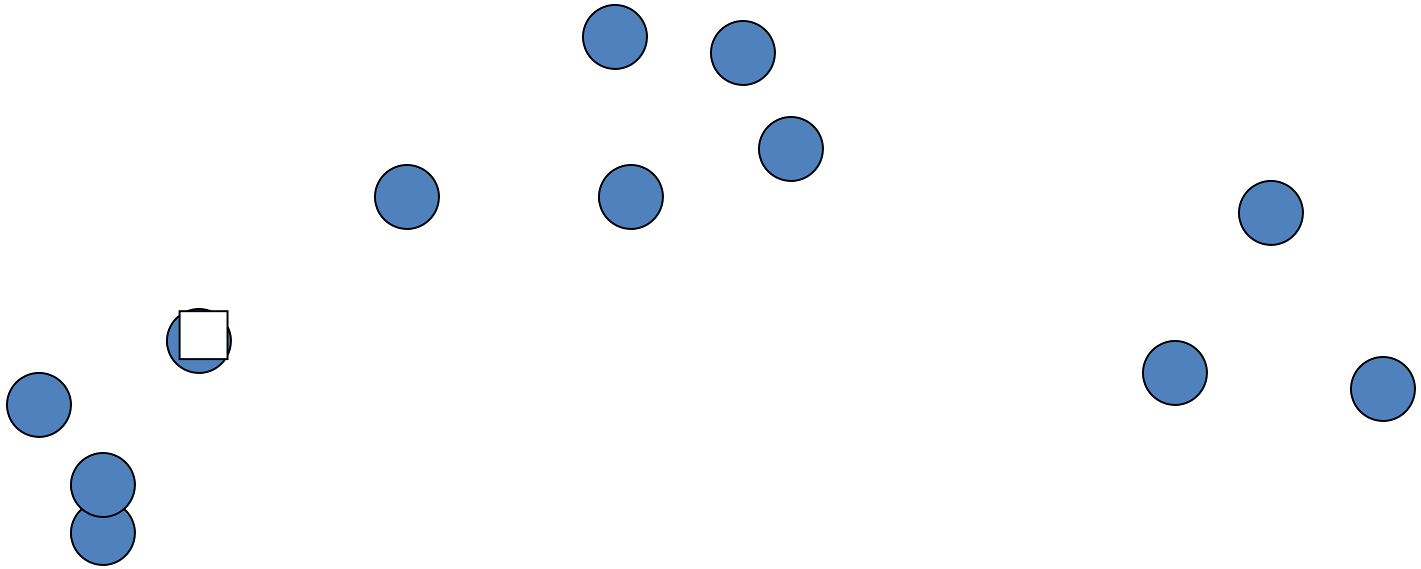
$$\mu_i = \underbrace{\underset{x}{\operatorname{argmax}}}_{\text{point with the largest distance to any previous center}} \underbrace{\min_{\mu_j : 1 < j < i} d(x, \mu_j)}_{\text{smallest distance from x to any previous center}}$$

K-means: Initialize furthest from centers



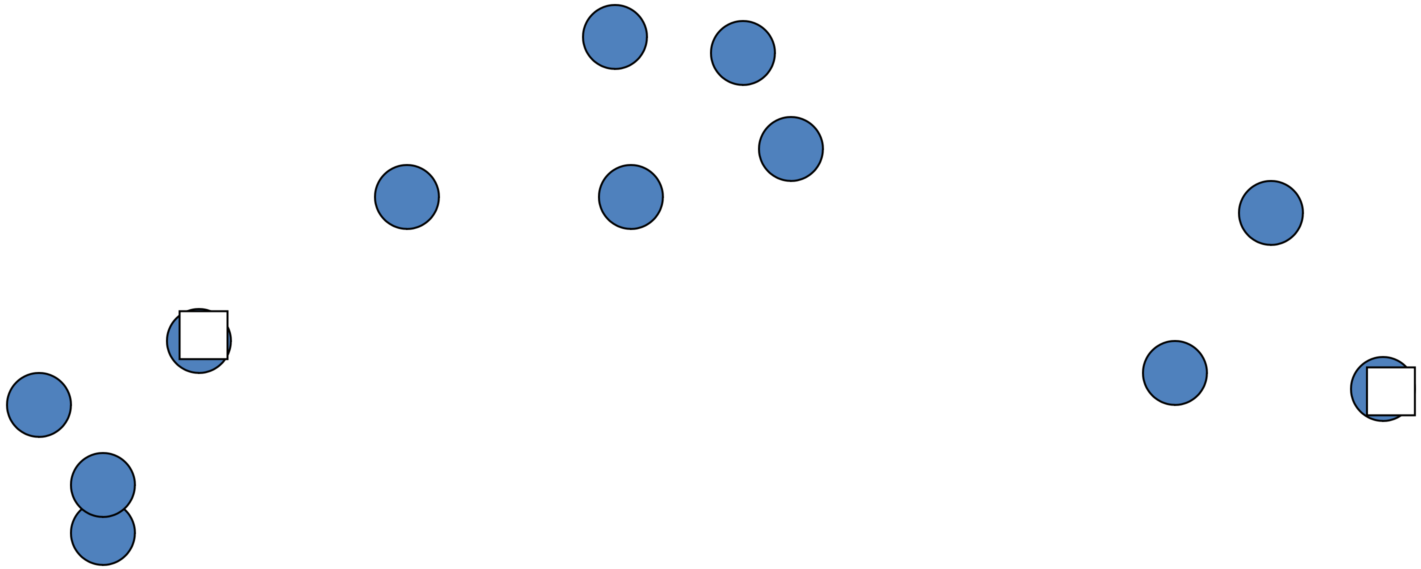
Pick a random point for the first center

K-means: Initialize furthest from centers



What point will be chosen next?

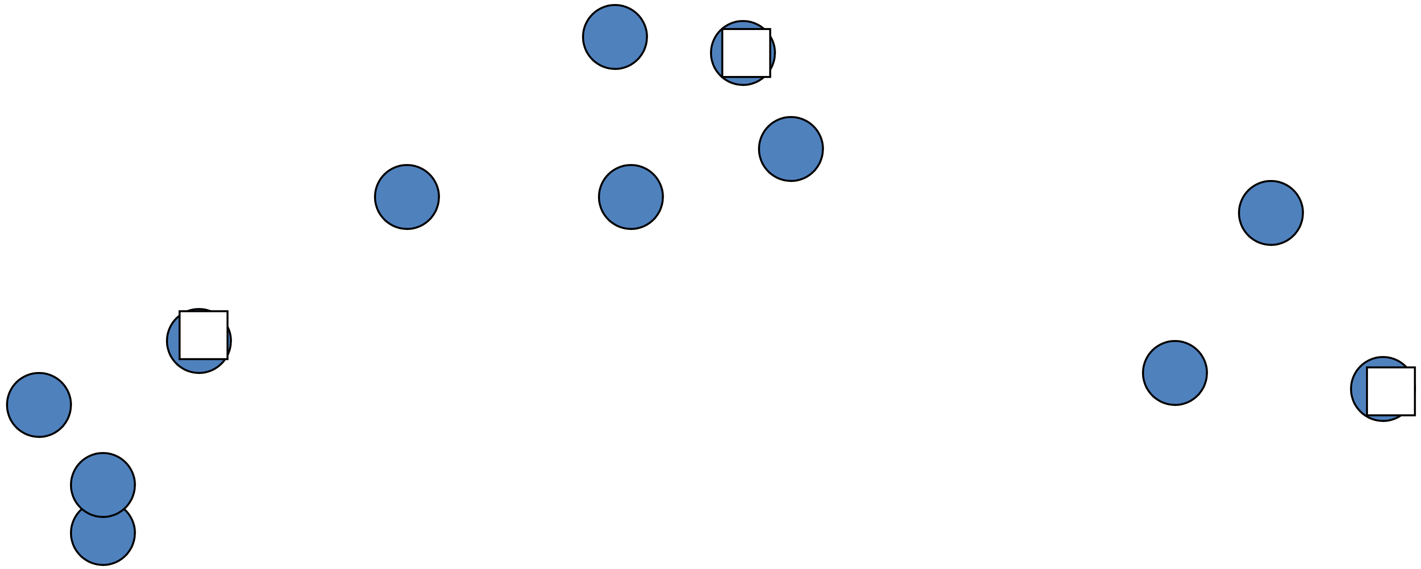
K-means: Initialize furthest from centers



Furthest point from center

What point will be chosen next?

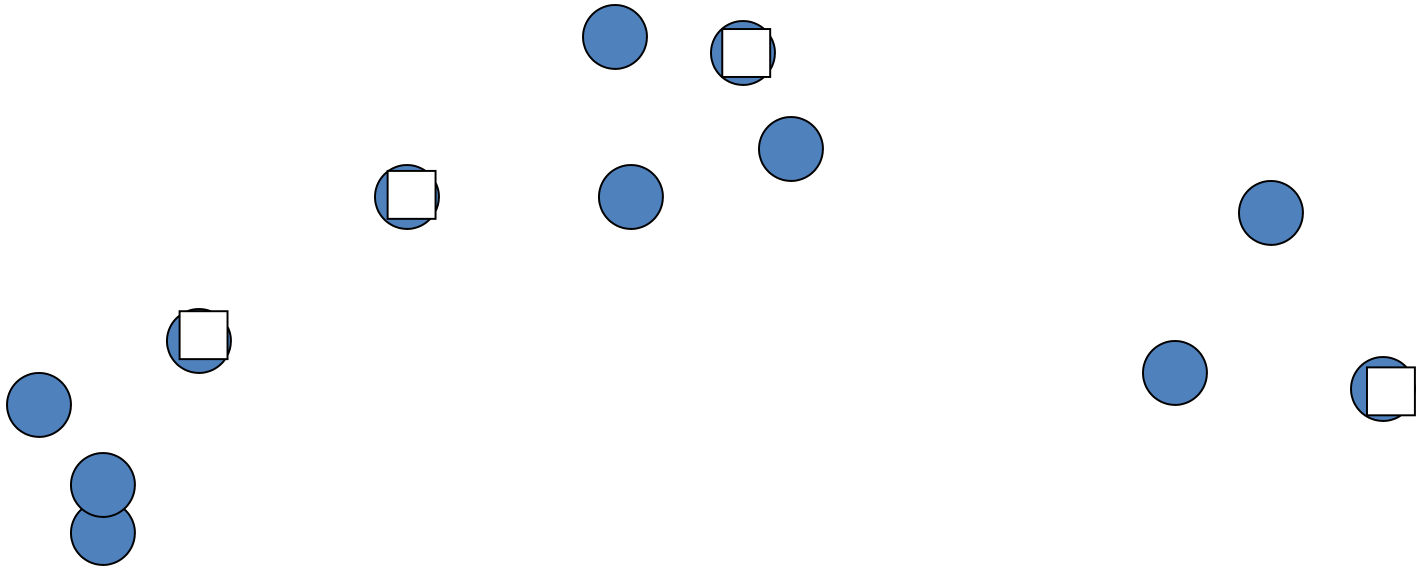
K-means: Initialize furthest from centers



Furthest point from center

What point will be chosen next?

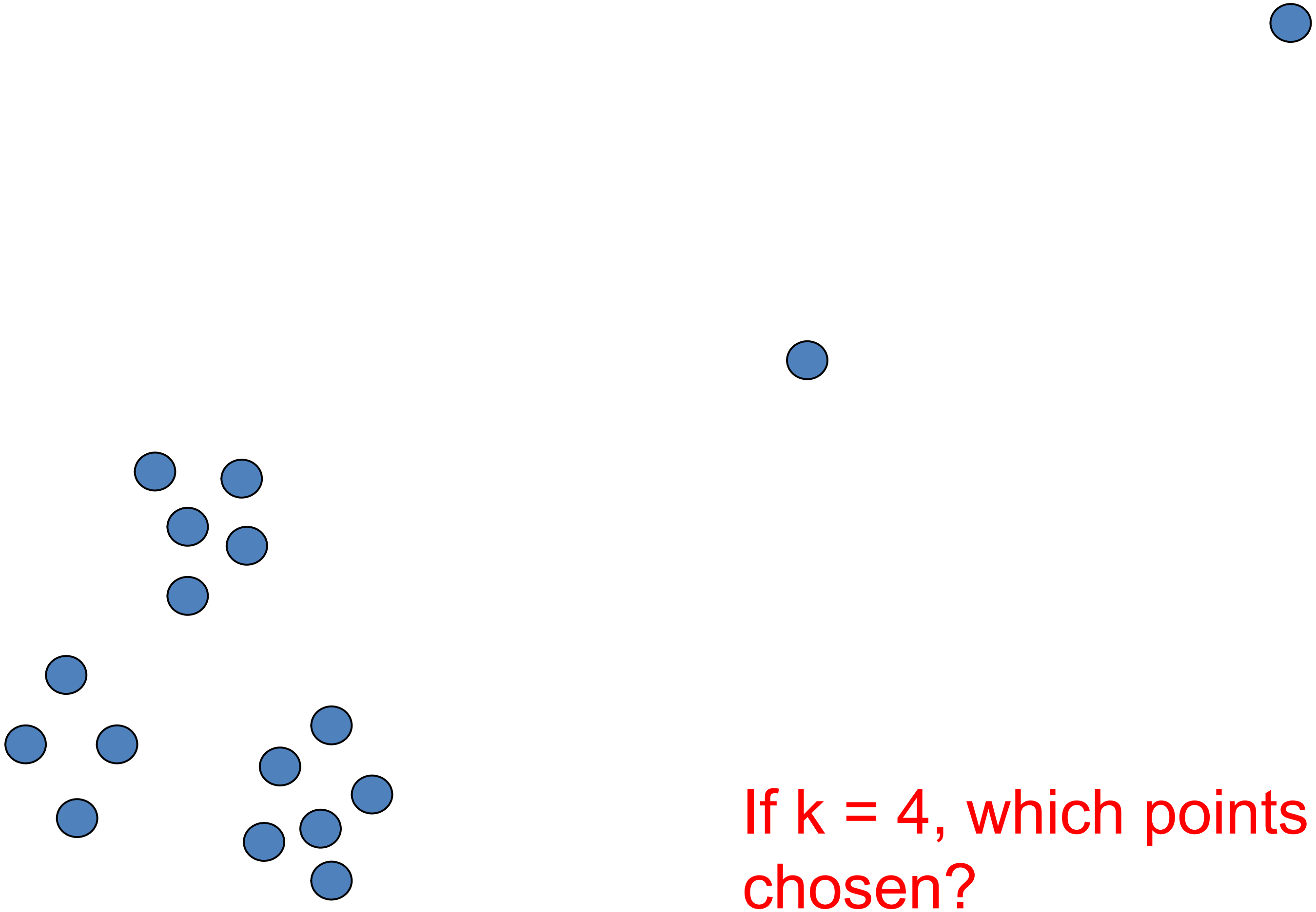
K-means: Initialize furthest from centers



Furthest point from center

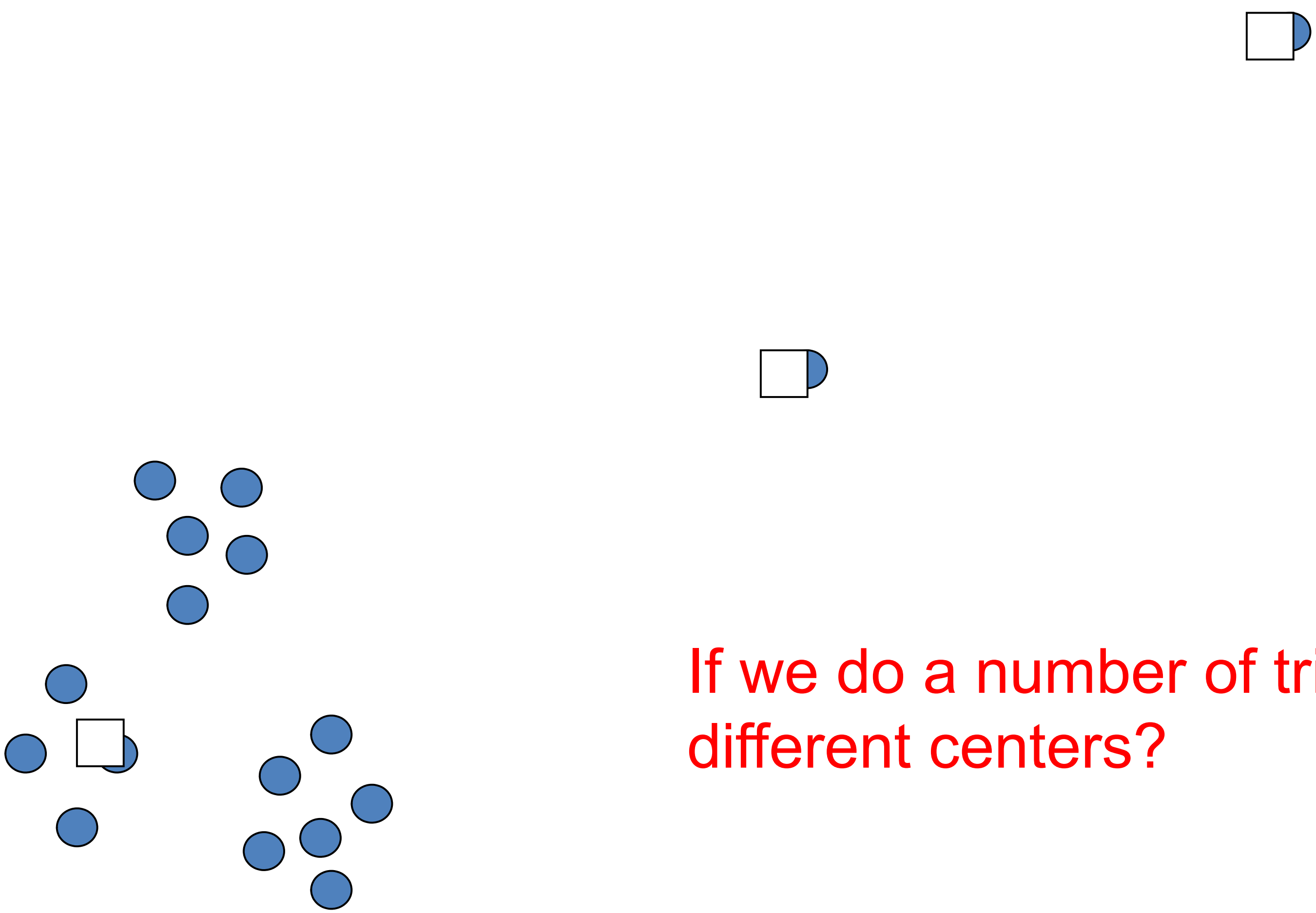
Any issues/concerns with this approach?

Furthest points concerns



If k = 4, which points will get chosen?

Furthest points concerns



If we do a number of trials, will we get different centers?

Furthest points concerns

Doesn't deal well with outliers

K-means++

μ_1 = pick random point

for k = 2 to **K**:

for i = 1 to **N**:

$s_i = \min d(x_i, \mu_{1\ldots k-1})$ // smallest distance to any center

μ_k = randomly pick point ***proportionate*** to ***s***

How does this help?

K-means++

μ_1 = pick random point

for k = 2 to **K**:

for i = 1 to **N**:

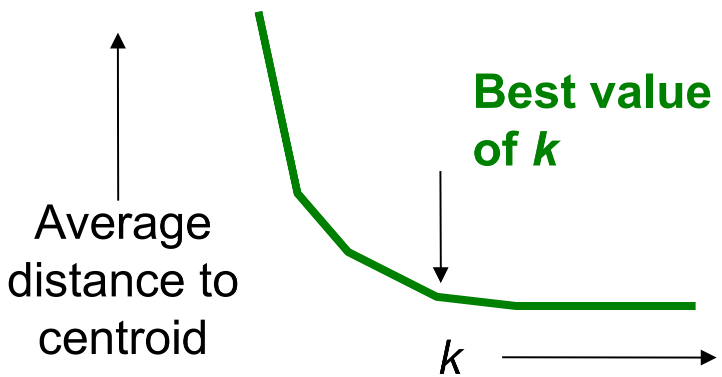
$s_i = \min d(x_i, \mu_{1\ldots k-1})$ // smallest distance to any center

μ_k = randomly pick point ***proportionate*** to ***s***

- Makes it possible to select other points
  - if #points >> #outliers, we will pick good points
- Makes it non-deterministic, which will help with random runs
- Nice theoretical guarantees!

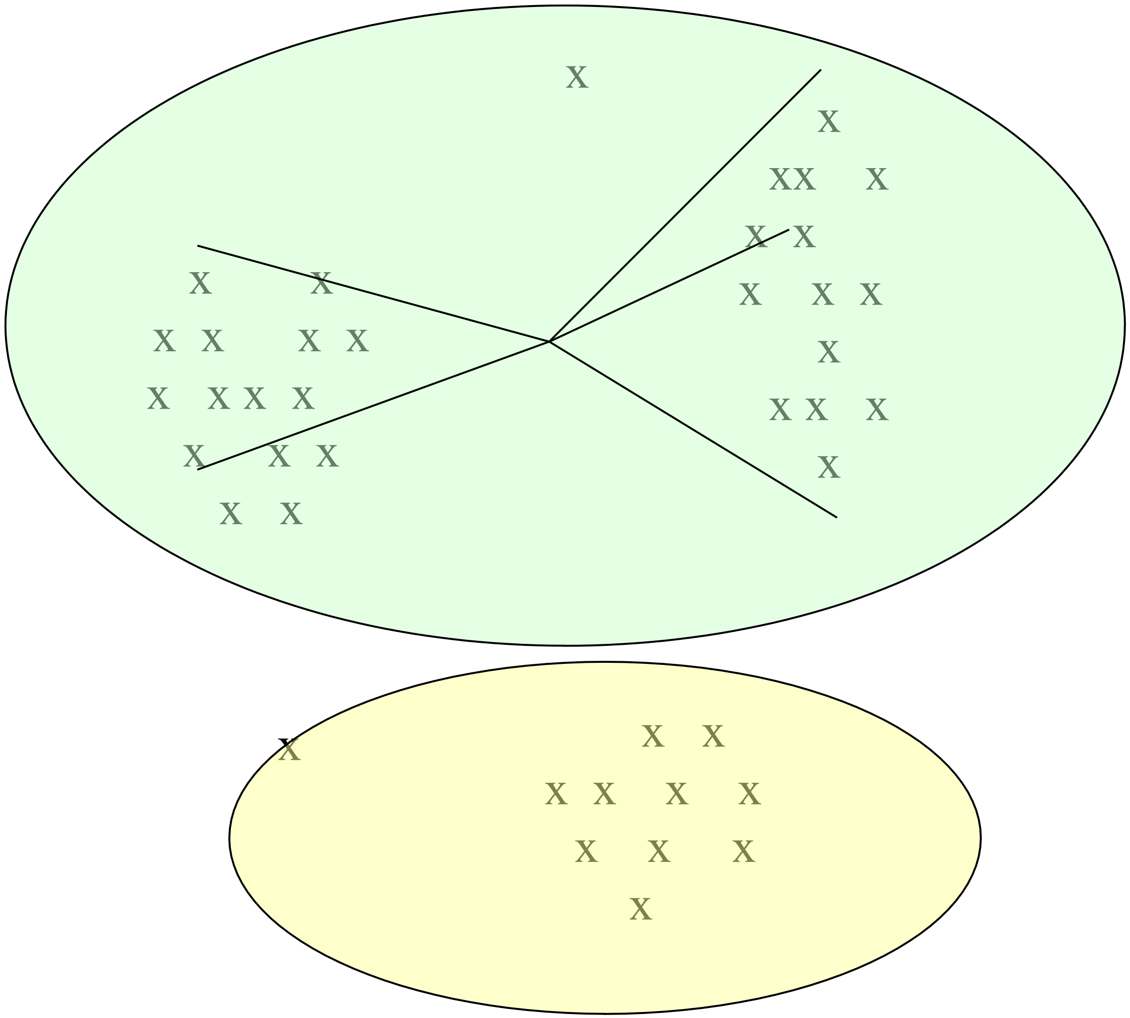
Getting the k right

- How to select k?
- Try different k, looking at the change in the average distance to centroid as k increases
- Average falls rapidly until right k, then changes little



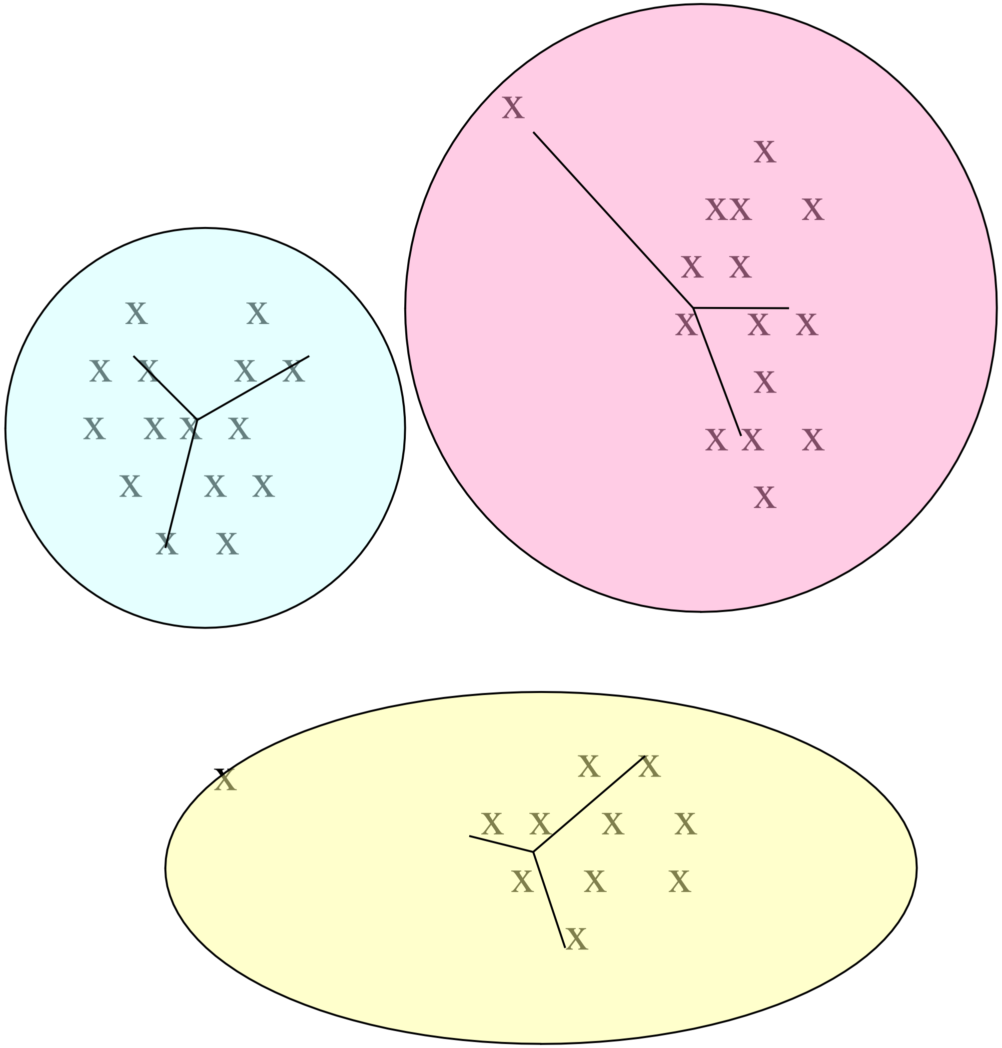
Example: Picking *k*

Too few;
many long distances to centroid.



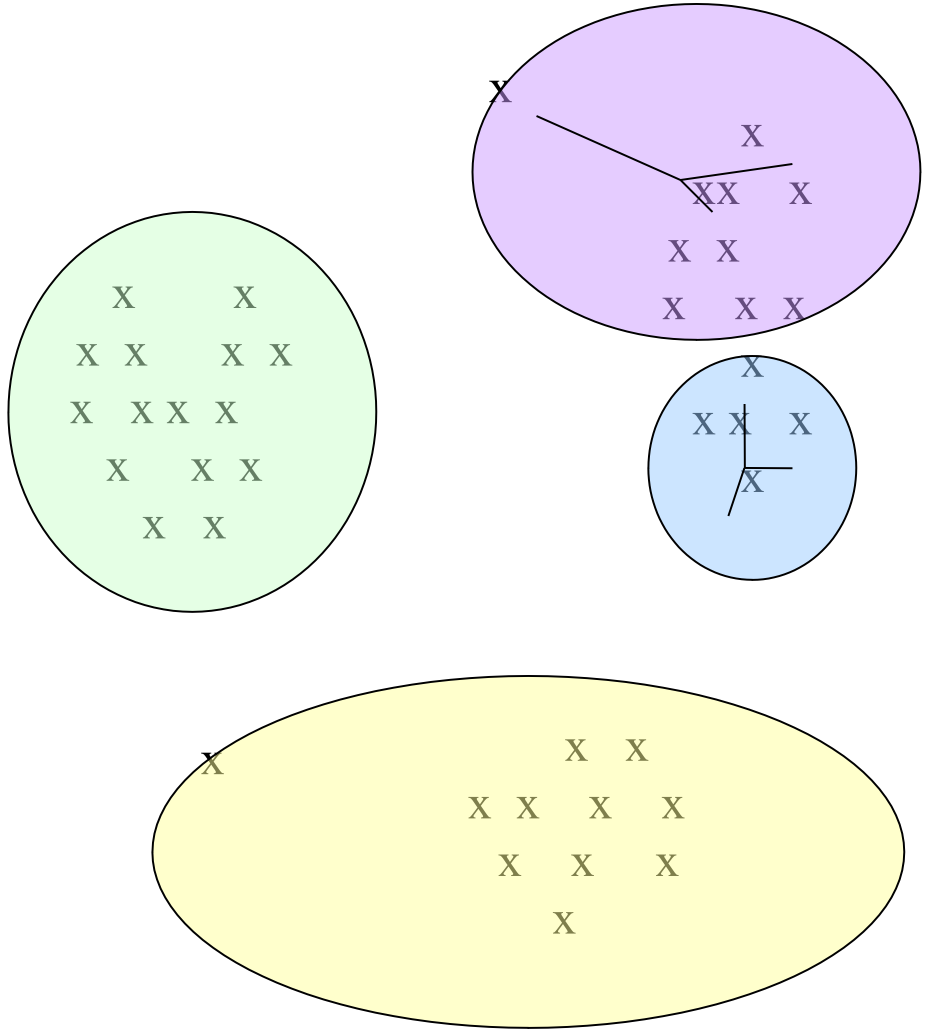
Example: Picking *k*

Just right;
distances
rather short.



Example: Picking *k*

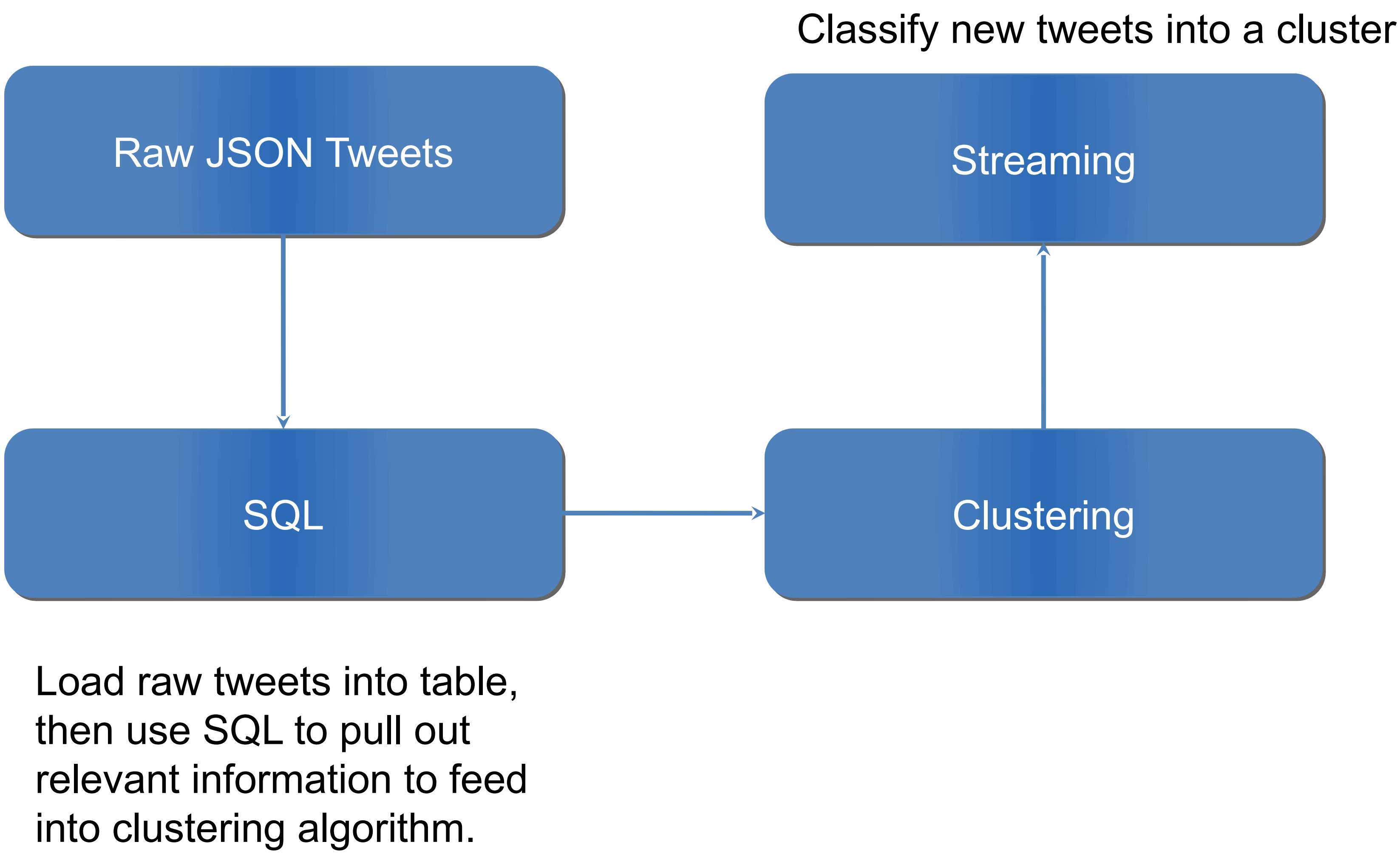
Too many;
little improvement
in average
distance.



Tweet Clustering by Language

- Tutorial
  - http://zdatainc.com/2014/08/real-time-streaming-apache-spark-streaming/
- Code is on github:
  - https://github.com/databricks/reference-apps
  - Uses Scala instead of Python (note, Twitter Stream is not yet natively supported for Python).

Cluster Tweets



Steps

- The problem is broken up into 3 parts :
- **Collect a Dataset of Tweets** - Spark Streaming is used to collect a dataset of tweets and write them out to files.
- **Examine the Tweets and Train a Model** - Spark SQL is used to examine the dataset of Tweets. Then Spark MLLib is used to apply the K-Means algorithm to train a model on the data.
- **Apply the Model in Real-time** - Spark Streaming and Spark MLLib are used to filter a live stream of Tweets for those that match the specified cluster.

Collecting Dataset

- Step 1 – collect dataset

```
val tweetStream = TwitterUtils.createStream(ssc, Utils.getAuth)
       .map(gson.toJson(_))

tweetStream.foreachRDD((rdd, time) => {
  val count = rdd.count()
  if (count > 0) {
    val outputRDD = rdd.repartition(partitionsEachInterval)
    outputRDD.saveAsTextFile(
      outputDirectory + "/tweets_" + time.milliseconds.toString)
    numTweetsCollected += count
    if (numTweetsCollected > numTweetsToCollect) {
      System.exit(0)
    }
  }
})
```

Cluster Tweets

Step 2 – Cluster tweets

```
val vectors = texts.map(Utils.featurize).cache()
    vectors.count()  // Calls an action on the RDD to populate the vectors cache.
    val model = KMeans.train(vectors, numClusters, numIterations)
    sc.makeRDD(model.clusterCenters,
numClusters).saveAsObjectFile(outputModelDir)

    val some_tweets = texts.take(100)
    println("----Example tweets from the clusters")
    for (i <- 0 until numClusters) {
      println(s"\nCLUSTER $i:")
      some_tweets.foreach { t =>
        if (model.predict(Utils.featurize(t)) == i) {
          println(t)
        }
      }
    }
```

Classify New Tweets

- Step 3 – Classify new tweets

```
println("Initializing Streaming Spark Context...")
val conf = new SparkConf().setAppName(this.getClass.getSimpleName)
val ssc = new StreamingContext(conf, Seconds(5))

println("Initializing Twitter stream...")
val tweets = TwitterUtils.createStream(ssc, Utils.getAuth)
val statuses = tweets.map(_.getText)

println("Initializing the KMeans model...")
val model = new KMeansModel(ssc.sparkContext.objectFile[Vector](
      modelFile.toString).collect())

val filteredTweets = statuses
      .filter(t => model.predict(Utils.featurize(t)) == clusterNumber)
filteredTweets.print()
```

Latent Dirichlet Allocation (LDA)

- As more information becomes available, it becomes more difficult to find and discover what we need.

- We need tools to help us organize, search and understand these vast amount of information.

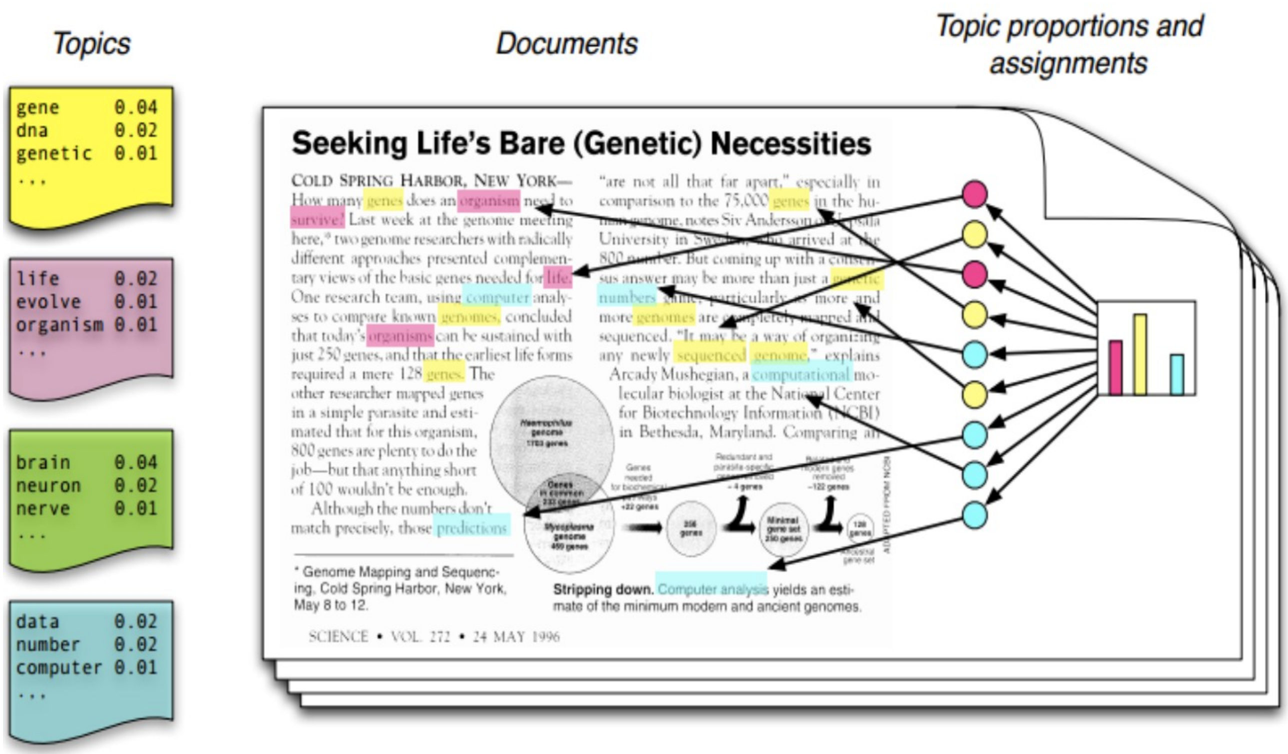
- Topic modeling provides methods for automatically organizing, understanding, searching, and summarizing large electronic archives:
-

  1. Discover the hidden themes in the collection
  2. Annotate the documents according to these themes
  3. Use annotations to organize, summarize, search, and form predictions

Some Assumptions

- We have a set of documents D_1, D_2, …, D_n.
- Each document is just a collection of words or a "bag of words". Thus, the order of the words and the grammatical role of the words (subject, object, verbs, …) are not considered in the model.
- Stop words like (am, is, the, this…) can be eliminated from the documents as a preprocessing step since they don't carry any information about the "topics".
- In fact, we can eliminate words that occur in at least %80 ~ %90 of the documents!
- Each document is composed of N "important" or "effective" words, and we want K topics.

Model Definition

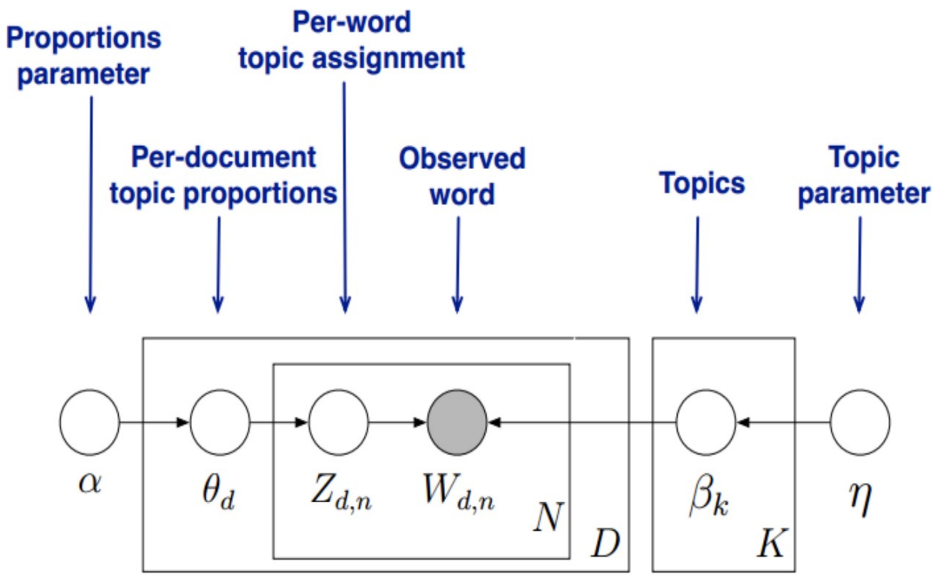


- Each topic is a distribution over words
- Each document is a mixture of corpus-wide topics
- Each word is drawn from one of these topics
- We only observe the words within the documents and the other structure are hidden variables.

Model Definition

- Our goal is to infer or estimate the hidden variables, i.e. computing their distribution conditioned on the documents.
  - P(topics, proportions, assignments | documents)

- Plate notation is a way of visually representing the dependencies among the model parameters.



- Shaded nodes are observed, and unshaded nodes are hidden.
- $\boldsymbol{\alpha}$ **–** is the parameter of Dirichlet prior on the per-document topic distribution
- $\boldsymbol{\beta}$ **–** is the parameter of the Dirichlet prior on the per-topic word distribution
- $\mathbf{W_{d,n}}$ – a specific word
- $\mathbf{Z_{d,n}}$ **–** is the topic of the nth word in the document
- $\boldsymbol{\theta_d}$ Is the topic distribution for document d

Generative Process

- LDA assumes that new documents are created in the following way:
  - Determine number of words in the document
  - Choose a topic mixture for the document over a fixed set of topics (i.e. 20% topic A, 30% topic B, 50% topic C)
  - Generate the words in the documents by:
    - First pick a topic based on the documents' multinomial distribution above.
    - Next pick a word based on the topic's multinomial distribution

Generative Process Example

- Say we have a group of articles that can be characterized by three topics: Animals, Cooking and Politics.

- Each of those topics can be described by the following words:
  - Animals: dog, chicken, cat, zoo, giraff
  - Cooking: Oven, food, restaurant, plates, taste, delicious
  - Politics: Republican, Democrat, Congress, Divisive, Ineffective

- Say we want to generate a new document that is 80% about animals and 20% about cooking.
  - We choose the length of the article (say, 1000 words)
  - We choose a topic based on our specified mixture (so, out of our 1000 words, 800 will be from topic ‘animals’)
  - We choose a word based on the word distribution of each topic

Working Backwards

- Suppose you have a corpus of documents
- You want LDA to learn the topic representation of K topics in each document and the word distribution of each topic
- LDA backtracks from the document level to identify topics that are likely to have generated the corpus.

Working Backwards (Cont.)

- Randomly assign each word in each document to one of the K topics.

- For each document d:
  - Assume that all topic assignments except for the current one are correct.
  - Calculate two proportions:
    - 1. Proportion of words in document d that are currently assigned to topic t = p(topic t | document d)
    - 2. Proportion of assignments to topic t over all document that come from this word w = p(word w | topic t)
  - Multiply those two proportions and assign w a new topic based on that probability. P(topic t | document d) * p(word w | topic t)