April 23, 2018

CS 361: Probability & Statistics

Clustering

The curse of dimensionality

The curse

As the dimensionality of a dataset increases, our intuitions based on experience with 2d and 3d data become less and less reliable

We will use a simple model to show that many ideas we have about data depend explicitly on how many dimensions it has

The model

We will be assuming that our data lies within a cube, of edge length 2, centered at the origin. Which is to say that each component of a data item x_i is in the range [-1, 1]

We will assume that our data is uniformly distributed within this cube. This means that the probability density of the data is given by

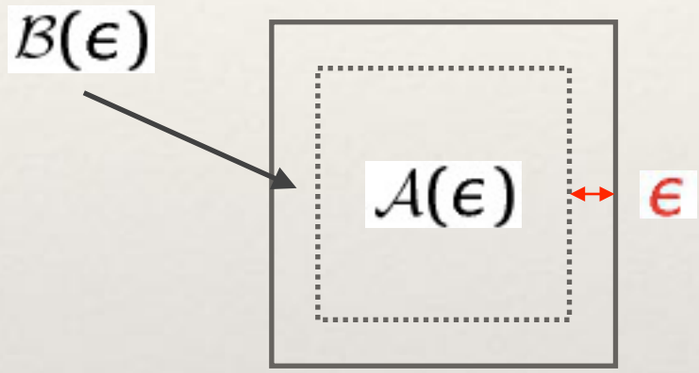
$$P(x) = \frac{1}{2^d}$$

where d is the number of dimensions of the data and x is a d-dimensional point

Like any uniform probability density, it is constant over the whole range of the data and is designed to have a total integral of 1

Data near the edge

The first phenomenon we will examine is how much of the data is near the edge



We can think of the inner box as being the fruit and the outer box as being the rind of the data, like an orange

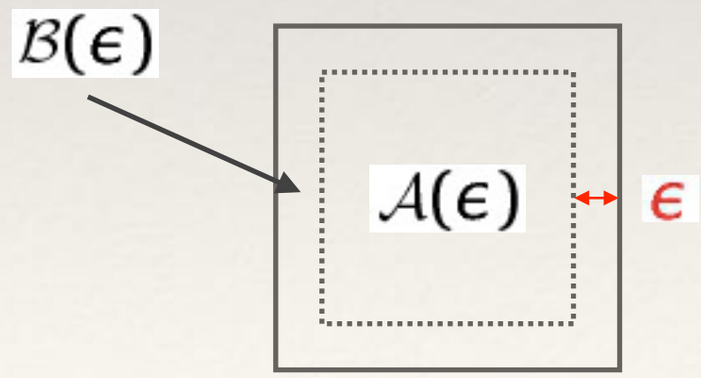
Let $\mathcal{A}(\epsilon)$ be the fruit. *A* is just the set of points where every component is in the range $[-(1-\epsilon),(1-\epsilon)]$. Let $\mathcal{B}(\epsilon)$ be the rind

Data near the edge

For a small epsilon, your intuition tells you that there should be more fruit than rind. And while this is true in, say, 3 dimensions it is not true in high dimensions

To show this, note that if we have a uniform probability density, we can compare the volume of two regions A and B in *d*-dimensional space by comparing P({x in A}) and P({x in B})

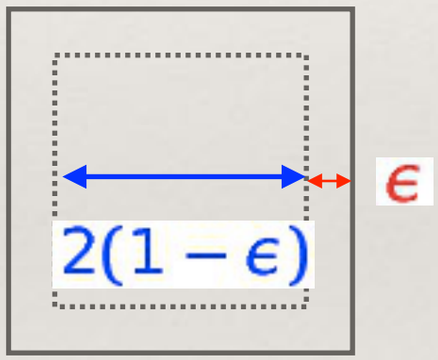
So we will compute, $P(x \in \mathcal{A}(\epsilon))$ where $\mathcal{A}(\epsilon)$ is the fruit or inner-box as before



Data near the edge

If we integrated to find $P(x \in \mathcal{A}(\epsilon))$ we would just be integrating the constant $\frac{1}{2^d}$ over the volume of $\mathcal{A}(\epsilon)$, so $P(x \in \mathcal{A}(\epsilon))$ is just the volume of $\mathcal{A}(\epsilon)$ multiplied by $\frac{1}{2^d}$

To compute the volume we need to know the side length of the inner box

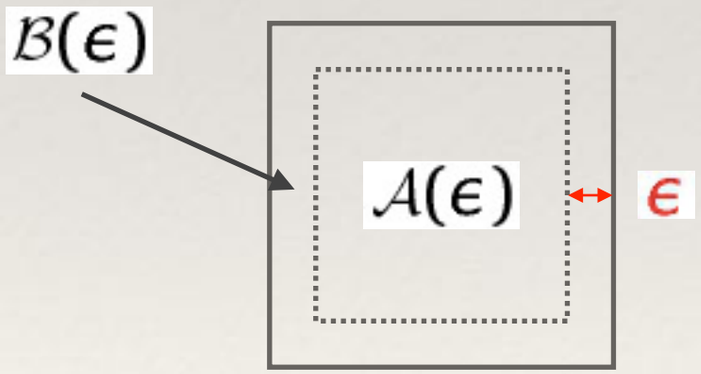


So the volume of $\mathcal{A}(\epsilon)$ is $2(1-\epsilon)^d$ which tells us that

$$P(x \in \mathcal{A}(\epsilon)) = (2(1-\epsilon))^d(\frac{1}{2^d}) = (1-\epsilon)^d$$ and $P(x \in \mathcal{B}(\epsilon)) = 1 - (1-\epsilon)^d$

Example

An epsilon of 0.05 would give us what seems like a fairly small rind, 90% of the side-length will be in the fruit. However, for d=100, we see that P(x in A) = (0.95)^100 which is approximately 0.0059. Or that 99.4% of the volume of the cube is near the edge



Conclusion: for large *d*, at least one component of most data items is near 1 or -1, i.e. the point is close to the boundary

Data far from the origin

The suggestion from the last result is that most of the data in a high dimensional space is far from the origin (even though the distribution we have been assuming has a mean at the origin)

Let's look at the expected value of the squared distance from the origin of points, under our uniform cube/box centered at the origin

$$\mathbb{E}[x^T x] = \int_{box} x^T x P(x)\, dx$$

$$\mathbb{E}[x^T x] = \int_{box} \sum_{i=1}^{d} (x^{(i)})^2 P(x)\, dx$$

Data far from the origin

$$\mathbb{E}[x^T x] = \int_{box} \sum_{i=1}^{d} (x^{(i)})^2 P(x)\, dx$$

Since the components of our data are independent from one another, we can rewrite this as

$$\mathbb{E}[x^T x] = \int_{-1}^{1} \sum_{i=1}^{d} (x^{(i)})^2 P(x^{(i)})\, dx^{(i)}$$

$$\mathbb{E}[x^T x] = \sum_{i=1}^{d} \int_{-1}^{1} (x^{(i)})^2 \frac{1}{2}\, dx^{(i)}$$

$$\mathbb{E}[x^T x] = \frac{d}{3}$$

Data far from each other

Consider the expected value of the squared distance between two points **u** and **v** distributed in our uniform cube

$$\mathbb{E}[d(\mathbf{u}, \mathbf{v})^2] = \int_{box} \int_{box} \sum_{i=1}^{d} (\mathbf{u}^{(i)} - \mathbf{v}^{(i)})^2 P(\mathbf{u}, \mathbf{v}) \, d\mathbf{u} \, d\mathbf{v}$$

Which if I rearrange and use the fact that **u** and **v** are independently sampled, I can write as

$$\mathbb{E}[d(\mathbf{u}, \mathbf{v})^2] = \mathbb{E}[\mathbf{u}^T\mathbf{u}] + \mathbb{E}[\mathbf{v}^T\mathbf{v}] - 2\mathbb{E}[\mathbf{u}^T\mathbf{v}]$$

Data far from each other

$$\mathbb{E}[d(\mathbf{u}, \mathbf{v})^2] = \mathbb{E}[\mathbf{u}^T\mathbf{u}] + \mathbb{E}[\mathbf{v}^T\mathbf{v}] - 2\mathbb{E}[\mathbf{u}^T\mathbf{v}]$$

This expression be simplified from my last result and the fact that since **u** and **v** are drawn independently I have $\mathbb{E}[\mathbf{u}^T\mathbf{v}] = 0$, giving a result of

$$\mathbb{E}[d(\mathbf{u}, \mathbf{v})^2] = 2\frac{d}{3}$$

The curse of dimensionality

Data isn't where we think it is. For high dimensional data, it is close to the boundary, far from the origin, and individual points are quite far from one another

This makes perhaps more sense if we think contrapositively:

For a given data point to be far from the boundary in high dimensions it has to have none of its many components close to -1 or 1

Each new dimension provides another opportunity to be close to -1 or 1 which is all we need for it to be at a boundary

For a given data point to be close to the origin, all of its many components have to be close to 0

For two data points to be anything but far from each other, in all of their many components they need to be relatively close

Multiple blobs

We will work with a relatively simple model of data that works well even in light of the strangeness of high dimensional data

This new type of model we will begin to explore allows us to think of data in a high-dimensional space as belonging to multiple more or less distinct blobs. This presents a couple of challenges that we will learn to solve

1) We will need to figure out a good probability distribution for high-dimensional blobs and then how to determine its parameters from the data

2) We will need to think about how to assign each data point to one of the blobs

Multivariate normal distribution

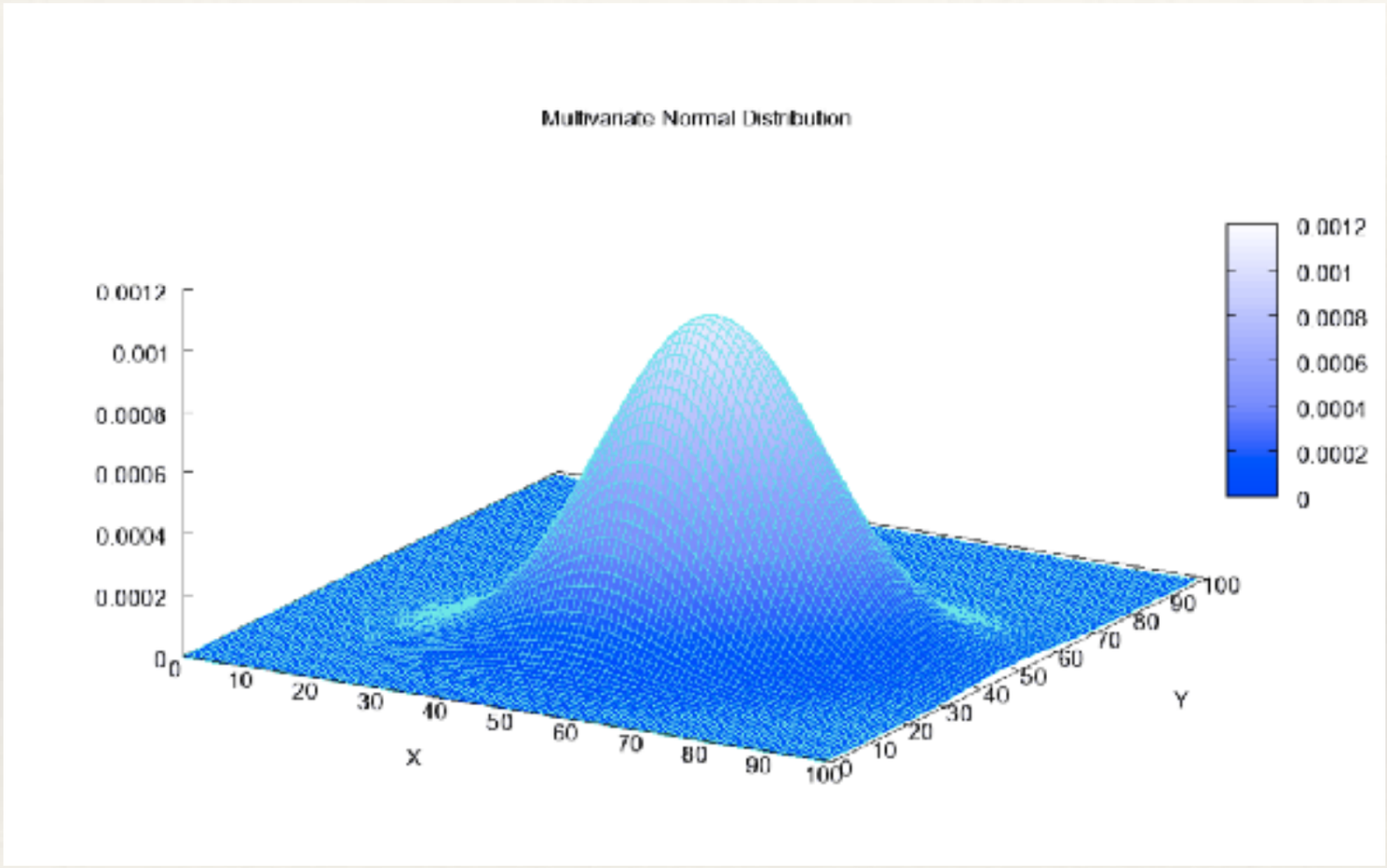
Multivariate normal

A simple probability distribution that works well for describing a high-dimensional blob is the multivariate normal distribution

This is just another type of random variable/probability model. Except the random quantities in this case are d dimensional vectors rather than a single real number

Luckily, a lot of what we developed earlier still applies. In particular, to analyze this distribution all we need is a probability density function that assigns some non-negative value to each point and that integrates to 1

Multivariate density



Source: wikipedia

To get densities that look like this—basically normal, but in more than one dimension—they will have the form we see on the next slide

Multivariate normal density

For a *d*-dimensional multivariate normal the parameters of the distribution are the *d* dimensional vector μ called the mean, and a *dxd* symmetric (and positive definite) matrix Σ called the covariance matrix. Those names sound familiar and we will see why in a minute

Given these parameters the density of the distribution is

$$p(\mathbf{x};\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

Multivariate expectations

If we used the density on the last slide to compute some expected values we would get

Useful Facts: 12.1 *Parameters of a Multivariate Normal Distribution*

Assuming a multivariate normal distribution, we have

- $\mathbb{E}[\mathbf{x}] = \mu$, meaning that the mean of the distribution is μ.
- $\mathbb{E}\left[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T\right] = \Sigma$, meaning that the entries in Σ represent covariances.

Multivariate MLE

If we had a *d* dimensional dataset that we believed followed a multivariate normal distribution, we would have the following MLE values for the parameters, which we show without proof (mainly because we don't want to differentiate a determinant!)

$$\hat{\mu} = \frac{\sum_i \mathbf{x}_i}{N}$$

and

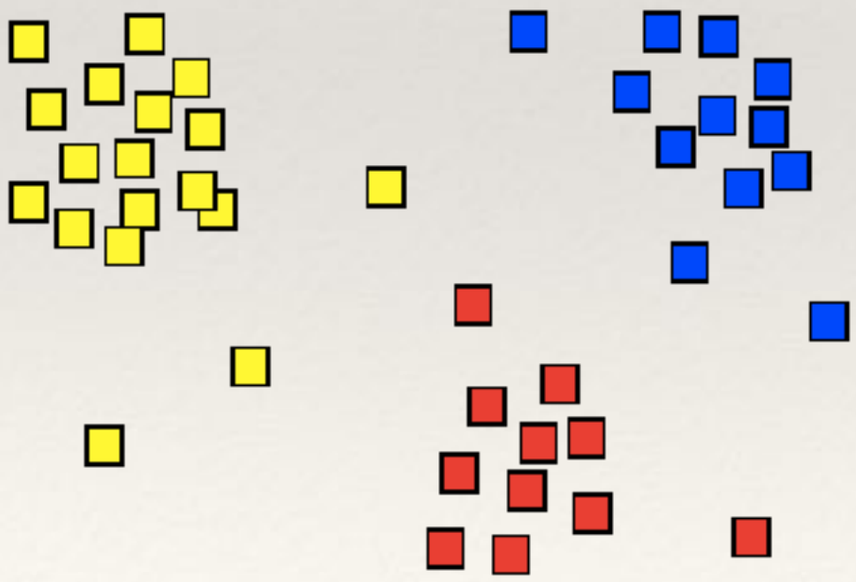
$$\hat{\Sigma} = \frac{\sum_i (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T}{N}$$

Clustering

Clustering

Setting aside entirely the discussion on multivariate normal distributions, we will turn to the problem of multiple blobs of data

The problem we are solving with clustering is: given a set of data points with some notion of distance between them, can we divide the data into groups such that items within the groups are close to one another and items in different groups aren't?



A sample clustering of 2d points. Note this isn't the same as classifying the points into 3 classes because we have no training data, we have no labels to learn a classifier from

Hierarchical clustering

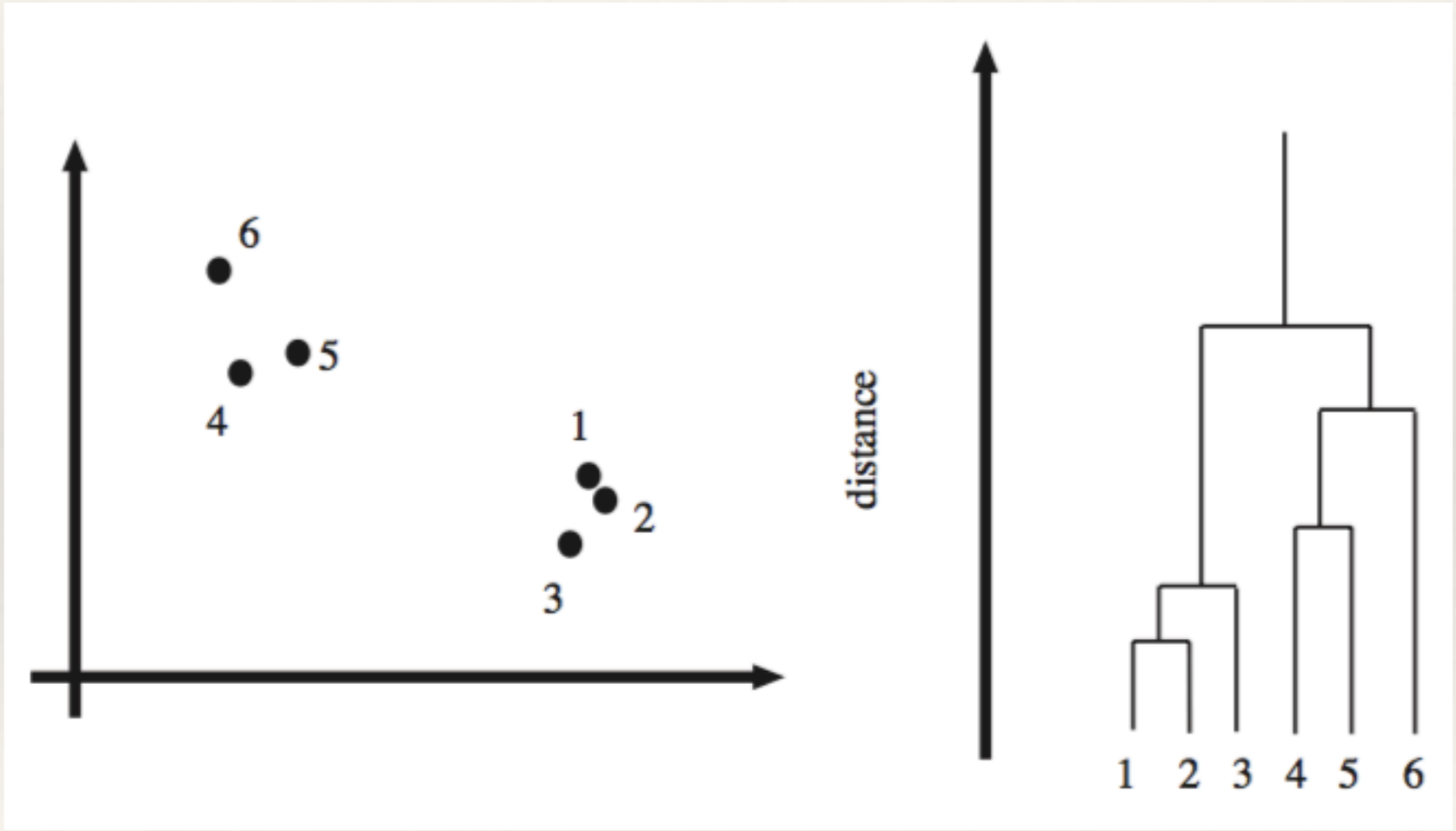
First we will talk about a couple of very natural algorithms for clustering that fall under the category of **hierarchical clustering**

In **divisive clustering** we regard the whole dataset as a cluster and then split the dataset recursively until we get a "satisfactory clustering"

In **agglomerative clustering**, each point is initially in its own cluster and we then merge clusters until we get a "satisfactory clustering"

These aren't the only two approaches to clustering. We will see another very important approach later

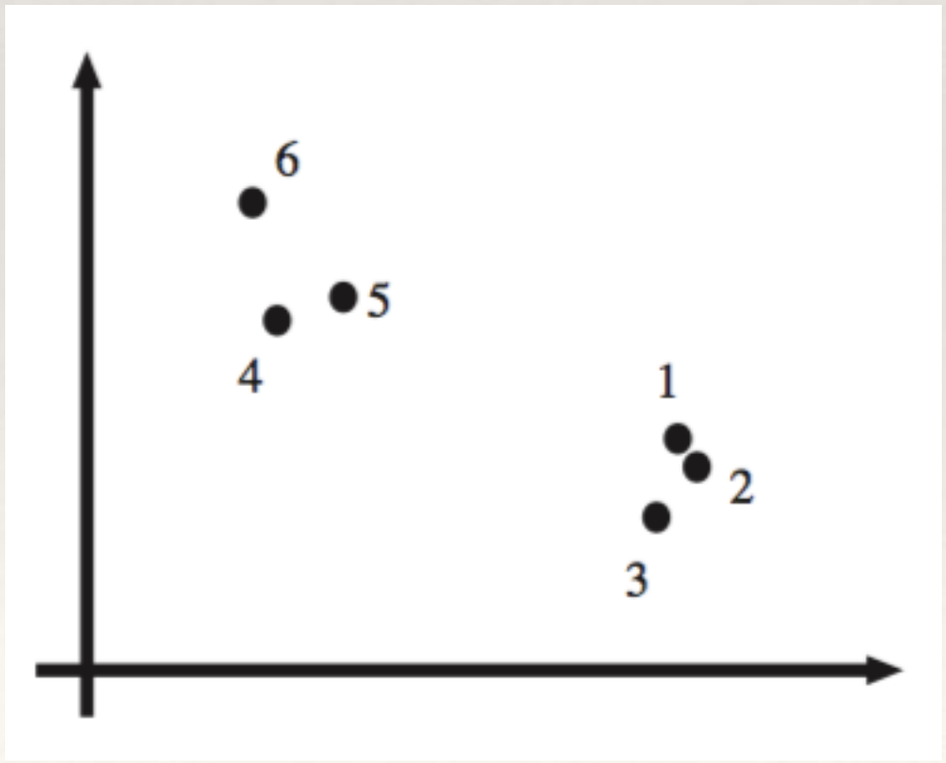
Hierarchical clustering



Inter-cluster distance

Two natural design questions arise. How should we decide which sets of clusters to merge/divide and what is a satisfactory set of clusters?

For the first question, we probably want to merge clusters with the smallest **inter-cluster distance**



If we were doing agglomerative clustering and had three clusters with c1 = {1,2,3}, c2 ={4,5} and c3={6}, we probably want to merge clusters c2 and c3

Inter-cluster distance

As we consider candidate pairs of clusters to merge, there are a number of ways we could measure inter-cluster distance to find a pair of clusters to merge

We might define the inter-cluster distance as the smallest distance between a point from cluster 1 and cluster 2. This is called **single-link clustering**

Another natural choice is to define the inter-cluster distance as the maximum distance between a point from cluster 1 and cluster 2. This is called **complete-link clustering**

Or we might choose to define the inter-cluster distance as the average distance between points from clusters 1 and 2. This is called **group average clustering**

Example

Suppose we have the following points to cluster: p1 = (0,0); p2=(1,0); p3=(-2, -3); p4=(0,-2); and p5=(1,1). Suppose further that the current clustering is c1=p1,p2; c2=p3, p4; and c3=p5. Which two clusters should we merge using the single-link cluster distance?

Evaluating merging c1 and c2

The single-link cluster distance is 2 (points p1 and p4 are the closest pair of points between c1 and c2)

Evaluating merging c1 and c3

The single-link cluster distance is 1 (points p2 and p5 are the closest pair of points between c1 and c3)

Evaluating merging c2 and c3

The single-link cluster distance is 3 (points p4 and p5 are the closest pair of points between c2 and c3)

So we would merge c1 and c3

Choosing a satisfactory clustering

One approach to deciding when to stop hierarchical clustering is the notion of the **cluster diameter**

For a given cluster its diameter is just the maximum distance between any two points in the cluster

For a stopping strategy, after each merger in agglomerative clustering, we might keep track of the average diameter of all the clusters. If we are merging clusters that *should* be merged, then this average diameter should increase gradually. If we see a sudden increase in the average radius, perhaps we should stop clustering just before that point

k-means clustering

Another approach

A completely different approach to clustering is given results from the following observations

If we had to produce a clustering of a set of points into k clusters, and we knew where the center of each cluster was, it would be easy to assign points to clusters. We would just assign a point to the cluster corresponding to the cluster center it is nearest to

If instead we knew which points belonged to which clusters, it is quite easy to calculate the center of the cluster

An algorithm

Taking these two observations into account a natural approach would be to try and minimize the following function by finding an optimal allocation of clusters

$$\Phi(\delta, \mathbf{c}) = \sum_{i,j} \delta_{i,j} \left[(\mathbf{x}_i - \mathbf{c}_j)^T (\mathbf{x}_i - \mathbf{c}_j) \right]$$

where

$$\delta_{i,j} = \begin{cases} 1 \text{ if } \mathbf{x}_i \text{ belongs to cluster } j \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$

where c_j is the cluster center of the j-th cluster and x_i is the i-th data point

Unfortunately there are an exponential number of different potential assignments of points to clusters, so we will need a different approach than to directly minimize this function

k-means

The method of **k-means clustering** takes our two observations into account in the following way.

First we assume we know that there are k clusters

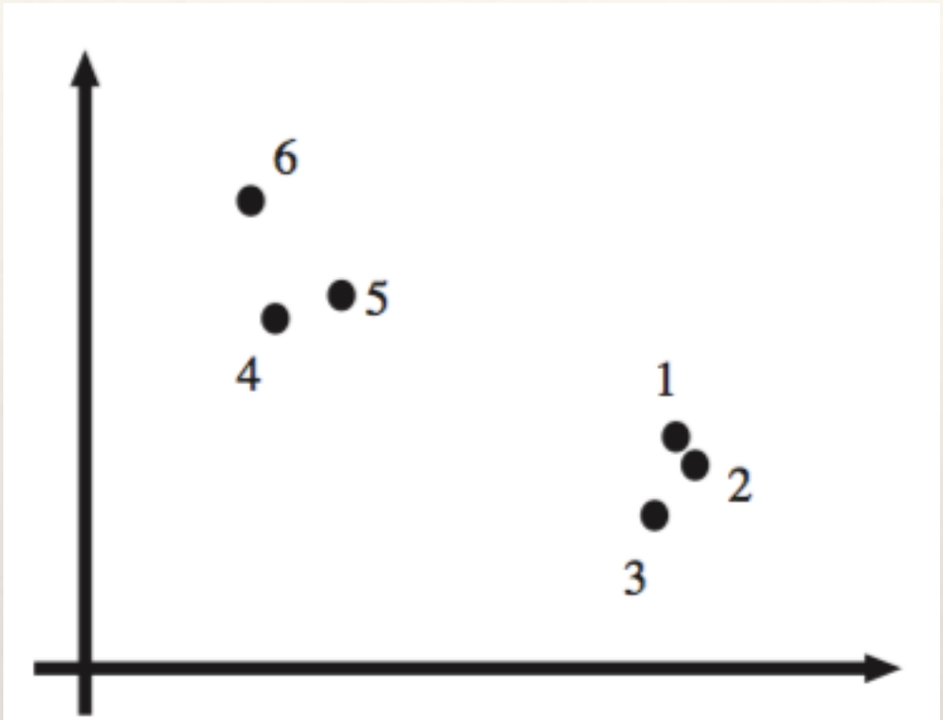
We initialize by picking k random points in our dataset and calling them cluster centers

Then, until our clustering changes very little, we run the following loop

(Re-)Assign each point to the cluster with the most nearby center

Calculate the center of all of the clusters

Example

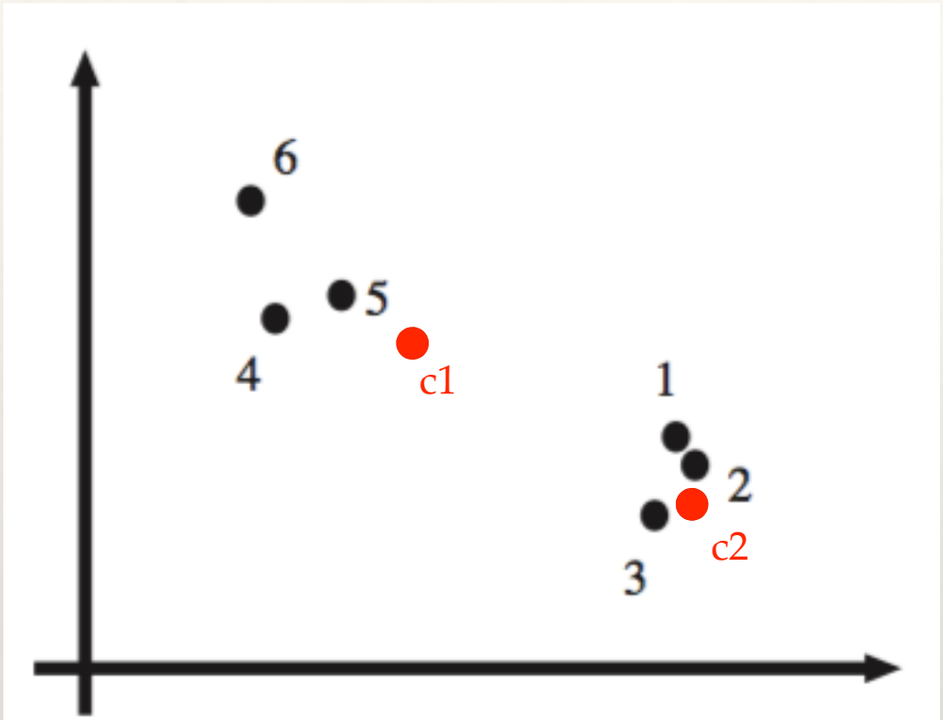


Let's say that k=2

And we randomly choose point 1 to be the center of cluster 1 and point 2 to be the center of cluster 2

Then we will have
c1 = {1,4,5,6} and c2={2,3}

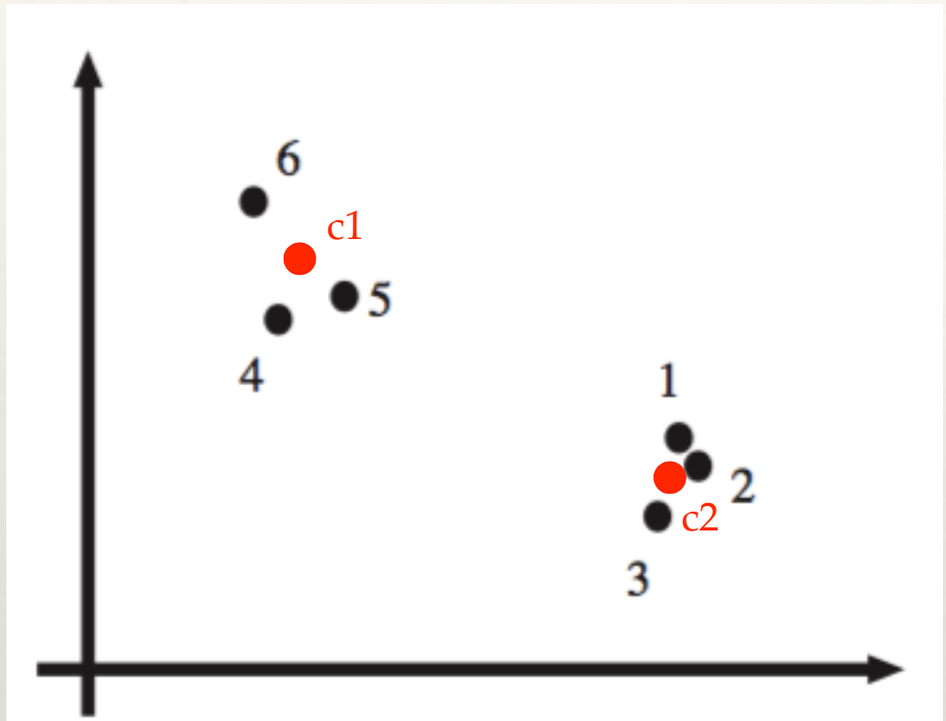
Example



With c1 = {1,4,5,6} and c2={2,3}, we re-calculate the cluster centers as just the mean of all the points in the cluster getting

We then re-assign points to their nearest cluster centers getting
c1={4,5,6} and c2={1,2,3}

Example



With c1={4,5,6} and c2={1,2,3} we recalculate cluster centers getting

We then re-assign points to the nearest cluster centers getting
c1={4,5,6} and c2={1,2,3}

This represents no change from the last iteration, so we stop

A couple of caveats

We need to figure out how to choose k

Doesn't this algorithm potentially produce a different clustering depending on which start points we use?