How Did Yeast Evolve to Become Such a Good Wine Maker?
Clustering Algorithms

Table of Contents

BA Chapter XXX: Comparing Pairs of Sequences

Center of Gravity
Visualization of Gene Expression Matrices
Clustering and Corrupted Cliques

Charging Stations

Why Does the Lloyd Algorithm Converge?

Code Challenges
FFT: FarthestFirstTraversal
LA: Lloyd algorithm (LA)
SKMC : Soft k-Means Clustering
HC: **HierarchicalClustering**

An Evolutionary History of Wine Making

How long have we been addicted to alcohol? Yeast is one of the oldest organisms to be domesticated by humans. In 2007, excavating an old graveyard in an Armenian cave, scientists found a 6,000 year-old winery showcasing a wine press, fermentation vessels, and even drinking cups. This discovery was important, because winemaking was a big technological innovation that required harnessing an otherwise inedible wild grape and understanding how to control ***Saccharomyces***, the genus of yeast used in alcohol and bread production.

Although this finding is the earliest evidence of wine-making, there is little doubt that we have been interested in alcohol for much longer than 6,000 years. In 2008, scientists discovered that pen-tailed tree shrews, that are similar to the ancient ancestors of all primates, are alcoholics. Their beverage of choice? A "palm wine" produced by the flowers of Bertram palms and naturally fermented by *Saccharomyces* yeasts. This finding suggests that our own taste for alcohol might have a genetic basis that predate the discovery in Armenia by 55 million years!

Figure. The world's oldest winery (left) and a tree shrew (right).

Pound for pound, the amount of palm wine that tree shrews consume daily would be lethal to most mammals. Fortunately, tree shrews have more efficient ways of metabolizing alcohol than we do, and so they avoid inebriation, which would increase their risk of being killed by a predator. Because of tree shrews' alcohol tolerance, scientists believe that alcohol probably serves some evolutionary advantages for these animals, such as protection against heart attack. It is possible that our not-so-distant ancestors were also chronic drinkers (e.g., chimpanzees like naturally brewed nectar from fruits) and that we may have inherited an ancestral association of alcohol intake with caloric gain.

Diauxic shift. Yeasts *Saccharomyces cerevisiae* are able to brew wine because they convert the **glucose** found in fruits into **ethanol**. Because *S. cerevisiae* often lives on grapewines, should not simply crushing grapes be sufficient for wine making? Why should wine makers put these crushed grapes into tightly sealed containers?

Once its supply of glucose runs out, *S. cerevisiae* must do something to survive. In order to adjust to new conditions, it inverts its metabolism, and the ethanol that it just produced becomes its new food supply. This change in metabolism, known as the **diauxic shift**, can only occur in the presence of oxygen, which explains why wine makers must tightly seal their barrels. In the absence of oxygen, the yeast will hibernate until either glucose or oxygen become available.

The diauxic shift is a complex process that affects the expression of many genes and was an enormous genetic innovation. But how and when did this shift occur? And which genes are involved in it?

Which Genes Are Responsible for the Diauxic Shift?

Two hypotheses with different fates. Remember Susumu Ohno and his Random Breakage Model (RBM) from Chapter 6? In addition to this hypothesis, he also came with the **Whole Genome Duplication** (WGD) hypothesis postulating the existence of dramatic and rare evolutionary events that fully duplicate a genome. WGD results in a twice longer genome and doubling the number of genes.

Ohno proposed the WGD model in 1970, when there was absolutely no evidence to support it. However, he argued that WGD would be needed at the time of an evolutionary innovation when a species would need to come up with a revolutionary new function. For example, imagine the time, millions of years ago, when the first fruit-bearing plants had evolved, but no organisms could metabolize the glucose produced by these fruits. For this reason, the first species to metabolize glucose would have had an enormous evolutionary advantage. Yet this is not a simple evolutionary task. Rather than creating a new gene here or there, metabolizing glucose would have required implementing new metabolic pathways with many genes working together. Ohno argued that a WGD event would provide a platform for such revolutionary innovation. Indeed, immediately after WGD, every gene would have two copies, one of them will be working to support whatever previous function (so that the organism can survive) while another will be free to evolve whatever way is beneficial without compromising the previous function.

The Random Breakage Model and the Whole Genome Duplication Model had very different fates. From the very beginning, the RBM was embraced by biologists and turned into a widely accepted dogma until it was refuted in 2003. In contrast, the WGD model was met with skepticism because *S. cerevisia*e had only a small fraction of duplicated genes. As a result, the WGD model did not get any traction for 25 years until Wolfe and Shields provided the first computational arguments in favor of WGD in *S. cerevisia*e, despite the fact that only about 13% of *S. cerevisiae* genes have a duplicate in *S. cerevisiae*. Indeed, even if 100s of genes participate in an evolutionary innovation after WGD, most genes are not really needed for this innovation. Thus, the unneeded duplicate genes will be bombarded by mutations until they are turned into pseudogenes before eventually disappearing from the genome millions of years later.

STOP and Think: How would you argue that *S. cerevisiae* has undergone a whole-genome duplication rather than duplicating genes one gene at a time when needed?

In 2004, Manolis Kellis compared the genome of *S. cerevisiae* to that of a newly sequenced yeast called *Kluyveromyces waltii.* It turned out that for nearly every synteny block in *K. waltii*, there were two synteny blocks in *S. cerevisiae* (see DETOUR: Finding Synteny Blocks in Highly Duplicated Genomes). This fact implied that there was indeed a WGD during yeast evolution, which Kellis estimated to have occurred about 100 million years ago.

Which yeast genes drive the diauxic shift? In 2005, Michael Thomson tried to figure out how WGD has led to the "invention" of the diauxic shift. One of the many steps yeast performs during fermentation is the **acetaldehyde-to-ethanol** conversion. If oxygen becomes subsequently available, the accumulated ethanol is converted back to acetaldehyde. Both the acetaldehyde-to-ethanol and ethanol-to-acetaldehyde conversions are catalyzed by **alcohol dehydrogenase (Adh)**. In *S. cerevisiae*, Adh activity is encoded by two genes that arose from the duplication of a single gene. Although the expression of Adh_1 hardly changes over time, Adh_2 is expressed only when glucose concentration drops. The Adh_1 enzyme has an elevated ability of producing ethanol as compared to Adh_2, whereas Adh_2 has an elevated ability of feeding on ethanol.

Thomson used a multiple alignment of Adh genes from various yeast species to reconstruct an ancient *Saccharomyces* Adh ancestor gene (referred to as Adh_A). This ancient gene showed a preference to convert acetaldehyde to ethanol, thus resembling Adh_1. Therefore, in pre-duplicated *Saccharomyces*, alcohol dehydrogenase was mainly involved in the generation, and not consumption, of ethanol. After WGD, Adh_A along with its duplicate (as well as some other duplicated genes) formed the basis for the "make-accumulate-consume" strategy of ethanol production, which provided the ancestor of *Saccharomyces* yeast with an advantage over its competitors because ethanol is toxic to most bacteria and yeasts. *Saccharomyces* kills its competitors by producing ethanol but can also consume the generated ethanol later.

STOP and Think. How would you find the genes in *S. cerevisae* that work together to accomplish the diauxic shift?

Imagine that you are able to monitor all *n* yeast genes for *m* hours (before and after the diauxic shift) resulting in an *nxm* **gene expression matrix** ($E_{i,j}$), where $E_{i,j}$ is the expression level of gene *i* at moment *j*. Just looking at this matrix you will be able to see genes with various behavior, e.g., genes whose expression hardly changes, genes whose expression rapidly increases before the diauxic shift and decreases afterwards, genes whose expression suddenly increases after the diauxic shift, etc.

STOP and Think. Can you divide all yeast genes into clusters so that genes in the same cluster have similar behavior and genes in different clusters have different behavior?

While this chapter focuses on gene expression across various time points to study the diauxic shift, biologists often use the same expression analysis technologies to study gene expression in a myriad of applications, from analyzing various tissues before and after taking drugs, to studying cancerous versus non-cancerous cells. For example, expression analysis led to developing **MammaPrint**, a diagnostic test that determines the likelihood of breast cancer returning after treatment. MammaPrint is based on gene expression analysis of 70 genes that are associated with activation and suppression of tumors. But how did biologists identify the human genes used in MammaPrint or yeast genes responsible for the diauxic shift?

Gene Expression Analysis

In 1997, in the first massive gene expression experiment ever, Joseph DeRisi and colleagues sampled an *S. cerevisiae* culture 7 times at hours -6, -4, -2, 0, +2, +4, and +6, where 0 refers to the timing of the diauxic shift. Since there are ≈6400 genes in yeast, they generated 6400x7 gene expression matrix.

STOP and Think. What technology would you use to generate this matrix?

We have already discussed three technologies that can be used to generate this matrix (see DETOUR: How Biologists Measure Gene Expression?). While it looks like DeRisi and colleagues had many technologies to choose from, neither of them matured by 1997! That is why they invented a yet another **microarray** technology (different from DNA array technology) that you can read about in DETOUR: Microarrays.

Since biologists today have so many options on how to generate data for expression analysis, we will not discuss the specifics of each technology but rather assume that we are already given an n x m gene expression matrix, where n is the number of genes and m is the number of checkpoints. The value $E_{i,j}$ in the gene expression matrix represents the expression level (non-negative number) of gene i at the checkpoint j. The entire i-th row of the expression matrix is called the **expression vector** of gene i. Biologists often work with logarithms of the expression levels and below, when we refer to the gene expression expression matrices, we mean logarithms of gene expression values.

Expression levels of related genes may vary by several orders of magnitude and genes with low expression levels may be related to genes with high expression levels. To facilitate comparison of genes with low and high expression levels, we will assume that each row of the gene expression matrix is normalized by dividing all element in the row i by the **vector norm** of this row equal to $\sqrt{}\ (\sum_{j=1,m} E_{i,j}^2)$ (See DETOUR: Vector Norm).

Exercise Break: Prove that all expression vectors in the normalized matrix have vector norm equal to 1.

If the expression vectors of two genes are similar, there is a good chance that these genes are somehow related, that is, they either perform similar functions or are involved in the same biological process. Of course, we do not exclude the possibility that expression vectors of two genes may look similar simply by chance, but the probability of such a chance similarity decreases as we increase the number of checkpoints. Accordingly, if the expression vector of a newly sequenced gene is similar to the expression vector of a gene with known function, a biologist may suspect that these genes perform similar or related functions. Another important application of expression analysis is in the deciphering of regulatory pathways; similar expression vectors may imply co-regulation, e.g., two genes are co-regulated if their expression is controlled by the same transcription factor. This suggests a "guilt by association" strategy for genome annotation starting from few genes with known functions and potentially propagating the known functions of these genes to other genes in the same cluster.

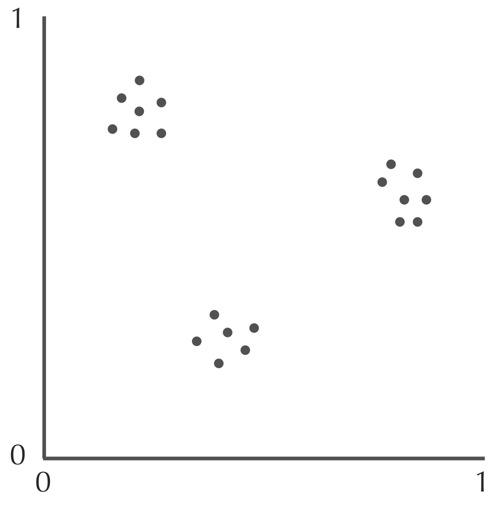
What is Clustering?

Homogeneity and separation. We would like to **partition** the set of all genes into clusters so that each gene belongs to a single cluster. Our clusters should satisfy two conditions, which are illustrated in Figure below:

- **Homogeneity**. Expression vectors of genes within a cluster should be similar to each other.

- **Separation**. Expression vectors of genes from different clusters should be dissimilar.

0.14	0.76
0.16	0.83
0.19	0.75
0.20	0.87
0.20	0.80
0.25	0.82
0.25	0.75
0.33	0.28
0.37	0.34
0.38	0.23
0.40	0.30
0.44	0.26
0.46	0.31
0.75	0.64
0.77	0.68
0.79	0.55
0.80	0.60
0.83	0.55
0.83	0.66
0.85	0.60



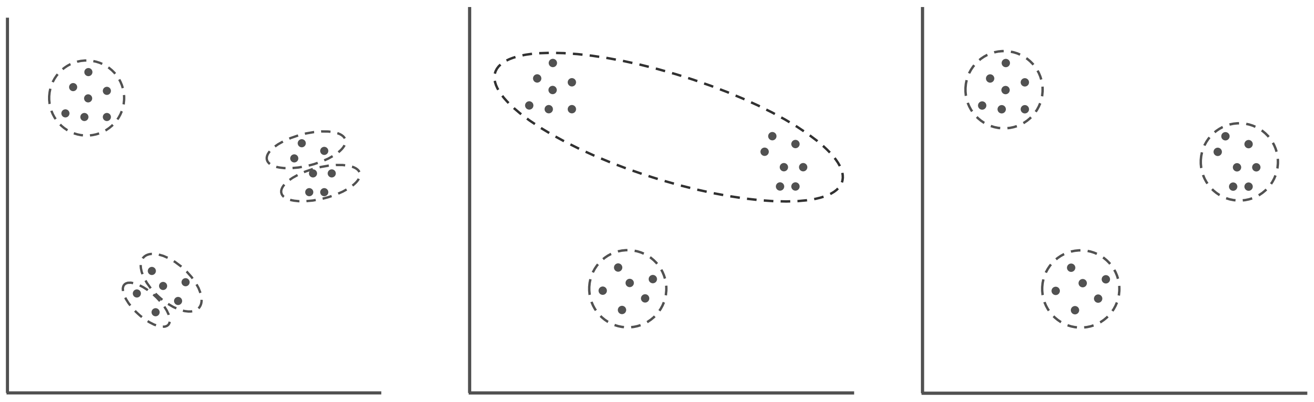


Figure. (Top) A 20 x 2 expression matrix along with a representation of each expression vector as a point in 2-dimensional space. Clusters on bottom left exhibit good homogeneity and separation, while the clusters on bottom right do not.

Clustering Problem. Partition a set of expression vectors into clusters.

Input. An *nxm* gene expression matrix *E*.

Output. Clusters of the *n* expression vectors from *E* satisfying the conditions of homogeneity and separation.

STOP and Think. You undoubtedly noticed that the Clustering Problem is ill-defined. How would you transform it into a well-defined computational problem?

Clustering as an optimization problem. As indicated in Figure above, one can view the *n* rows of the $n \times m$ gene expression matrix as a set *Data* of *n* points in *m*-dimensional space. The number of clusters, which we denote as *k*, is not known in advance, so biologists typically apply clustering algorithms to gene expression data for various values of *k* and then select the most conclusive value of *k*.

Given points $v=(v_1, \ldots, v_m)$ and $w=(w_1, \ldots, w_m)$ in multi-dimensional space, the **Euclidean distance** between them is defined as

$$d(v,w)=sqrt(\sum_{1\leq i\leq m} (v_j—w_j)^2).$$

Given a point *DataPoint* in multi-dimensional space and a set of *k* points *X,* called **centers**, the distance from *DataPoint* to *X*, denoted *d*(*DataPoint, X*), is defined as the Euclidean distance from *DataPoint* to the closest center in *X*:

$$d(DataPoint, X) = \min_{x \text{ in } X} d(DataPoint, x)$$

Given a set of points *Data*, we define the maximal distance between data points and centers, denoted *MaxDistance(Data,X)* as the maximum of *d(DataPoint,X)* among all data points:

$$\max_{\text{all points } DataPoint \text{ in } Data} d(DataPoint,X)$$

***k*-Center Clustering Problem.** Given a set of data points, find *k* center points minimizing the maximum distance between these data points and centers.

Input: A set of points *Data* and an integer *k*.

Output: A set *X* consisting of *k* points (centers) that minimize *MaxDistance(DataPoints,X)* over all possible choices of *X*.

The Farthest First Traversal. The pseudocode for the **Farthest First Traversal** heuristic for the *k*-Center Clustering Problem is shown below. This approach starts with an arbitrary point as the first center; it then iteratively adds a new center as the data point farthest from the centers chosen so far.

```
FarthestFirstTraversal(Data, k)
  DataPoint ← a point from Data
  X ← the set consisting of a single point DataPoint
  while |X| < k
    DataPoint ← a point in Data maximizing d(DataPoint,X) among all data points
    add DataPoint to X
  return X
```

Code Challenge. Implement **FarthestFirstTraversal** (FFT). Assume that you select the 1st point from *Data* at the initial step.

STOP and Think. What is the running time of **FarthestFirstTraversal?**

Exercise Break. Let *X* be a solution found by **FarthestFirstTraversal** and X_{opt} be the optimal solution of the k-Center Clustering Problem. Prove that $MaxDistance(Data,X) \leq 2*MaxDistance(Data,X_{opt})$.

Although **FarthestFirstTraversal** is fast and its solution often comes close to the optimal solution of the *k*-Center Clustering Problem, it is not used often for gene expression data analysis. The reason why is because in *k*-Center Clustering, we selected centers *X* so that they would minimize *MaxDistance(Data, X*), the maximum distance between any point in *Data* and its nearest center. But biologists are usually interested in analyzing *typical* rather than *maximum* deviations (since maximum deviations may represent outliers that often correspond to experimental errors).

STOP and Think. Can you come up with an alternative scoring function for evaluating the quality of clustering that is more biologically adequate than *MaxDistance(Data,X)*.

k-Means Clustering

The squared error distortion. The **squared error distortion** for a set of *n* data points *Data* and a set of *k* centers *X,* is defined as the mean squared distance from each data point to its nearest center:

$$Distortion(Data,X) = \sum_{\text{all points } DataPoint \text{ in } Data} d(DataPoint,X)^2 \,/\, n.$$

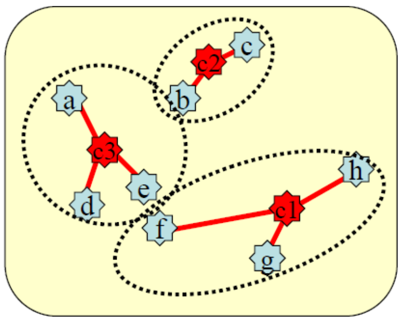


Figure. The squared error distortion is computed as 8 "red" squared distances divided by 8.

STOP and Think. Given a set of data points and a set of *k* centers, how would you define partitioning of the data points into *k* clusters imposed by *k* centers?

Squared Error Distortion Problem. Given data points and centers, compute the squared error distortion.

Input: A set of points *Data* and a set of centers *X*.

Output: The squared error distortion *Distortion(Data,X)*.

Code Challenge. Solve the Squared Error Distortion (SED) Problem.

We are interested in finding a set of centers *X* minimizing *Distortion(Data,X)* since the squared error distortion turned out to be a good measure of "quality" of clustering of the set *Data* "imposed" by the set of centers *X*.

k-Means Clustering Problem. Given a set of data points, find *k* center points minimizing the squared error distortion.

Input: A set of points *Data* and an integer *k*.

Output: A set *X* consisting of *k* centers that minimizes *Distortion(Data,X)* over all possible choices of *X*.

Clustering data points into *k* clusters can be obtained from a solution of k-Means Clustering Problem by simply assigning each data point to its closest center.

k-Means clustering and the center of gravity. It turns out that the *k*-Means Clustering Problem is *NP*-Hard when *k* > 1. Let us therefore consider the case *k* = 1. Although we acknowledge that partitioning a set of all data points into a single cluster is not exactly a complicated problem, finding the center of these points is less obvious and will soon help us design a heuristic for larger values of *k*.

First, note that when *k* = 1, the *k*-Means Clustering Problem is equivalent to finding a center point *x* that minimizes the sum of squared distances from *Data*:

$$Distortion(Data, x) = \sum_{\text{all points } DataPoint \text{ in } Data} d(DataPoint_i, x)^2/n$$

Next, we define the **center of gravity** of points *Data* as $(\sum_{\text{all points } DataPoint \text{ in } Data} DataPoint)/n$, i.e., the point whose *i*-th coordinate is the average of *i*-th coordinates of all points from *Data*.

Theorem. The center of gravity of a set of points *Data* solves the k-Means Clustering Problem for *k* = 1, i.e., it minimizes *Distortion*(*Data, x*) among all possible centers *x*.

For a proof of this theorem, see DETOUR: Center of Gravity.

Although the k-Means Clustering Problem turned out to be simple for *k*=1, it is NP-hard even for *k*=2 in the case of clustering in a multi-dimensional space. However, in the case when all data points are located on a line, k-means clustering problem can be solved in polynomial time.

Exercise Break. Develop an algorithm for solving the k-Means Clustering Problem in one dimension.

The Lloyd Algorithm

From centers to clusters and back. The **Lloyd algorithm** is one of the most popular clustering heuristics, and it often generates good approximate solutions for the *k*-Means Clustering Problem. In the beginning, this method chooses *k* arbitrary data points $X = \{x_1, . . . x_k\}$ as centers. The Lloyd algorithm then iteratively performs the following two steps (Figure below):

- **From Centers to Clusters**: Assign each data point to the cluster corresponding to its closest center (ties are broken arbitrarily).
- **From Clusters to Centers**. After the assignment of all data points to *k* clusters, compute new centers as each cluster's center of gravity.

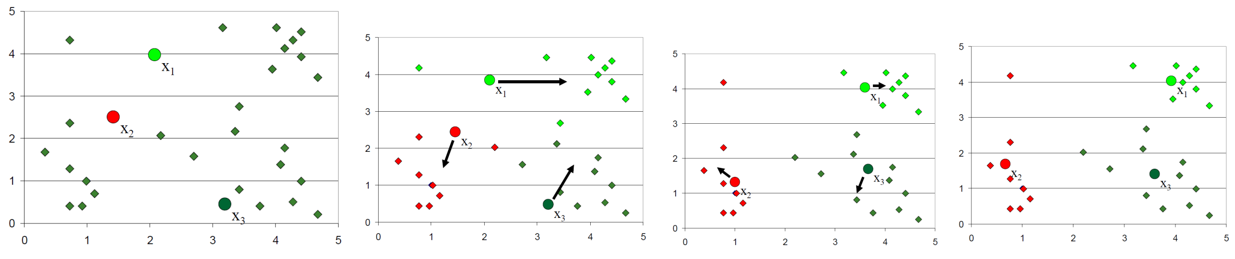


Figure. The Lloyd algorithm in action.

Code Challenge. Implement the Lloyd algorithm (LA) for *k*-means clustering.

As you may have noticed in Figure above, the centers appear to be moving less and less between iterations. We say that the Lloyd algorithm has **converged** if it the centers (and therefore the clusters) stop changing between iterations.

Exercise Break. Can you design a set of data points such that the Lloyd algorithm does not converge.

It is easy to see that the Lloyd algorithm converges. Indeed if the centers do not change between consecutive iterations of the Lloyd algorithm, we have converged. Otherwise:

- Whenever an assignment of data to clusters is changed during the "From Centers to Clusters" step, the squared error distortion is reduced.
- Whenever a cluster center is moved during the "From Clusters to Centers" step, the squared error distortion is reduced.

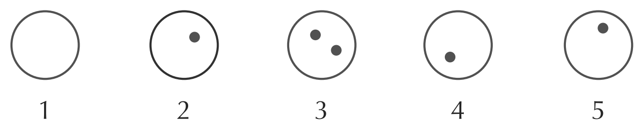
If you still wonder why the squared error distortion reduces at each step, see the Charging Station: Why does the k-means clustering algorithm converge? The Lloyd algorithm often converges to a local minimum of the squared error distortion function rather than the global minimum.

Exercise Break. Run the Lloyd algorithm 1000 times on thd diauxic shift dataset (each time initialized with a new set of 6 randomly chosen centers) and construct the histogram of the squared error distortions of the resulting 1000 solutions. How many times did you have to run the Lloyd algorithm before finding the highest-scoring solution among 1000 solutions?

Initializing the Lloyd algorithm. Since the Lloyd algorithm converges to a local minimum, our goal is to initialize it in such a way that it has a better chance to converge to a global minimum. Figure below illustrates that things could go horribly wrong if you do not pay attention to initialization. It shows five circles in 2-D with each circle (cluster) containing 100 points (not shown). If we initialize the Lloyd algorithm by choosing five centers at random from the data, there is a good chance that we will end up with no centers from circle 1, two centers from circle 3, and one center each from circles 2, 4, and 5.

STOP and Think. What is the probability that one of the circles will have no centers?

After the first iteration of the Lloyd algorithm, all points in circles 1 and 2 will be assigned to the leftmost center, which will move approximately halfway between circles 1 and 2 (Figure below (bottom)). The two centers in circle 3 will end up dividing the points in that circle into two clusters. And the centers in clusters 4 and 5 will move toward the centers of those clusters. The Lloyd algorithm will then quickly converge, resulting in an incorrect clustering.



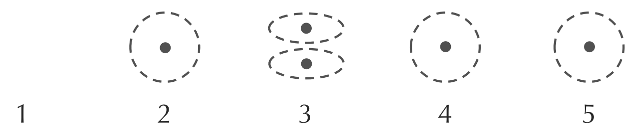


Figure. (Top) Five circles in two-dimensional space with each circle containing 100 points (not shown). The Lloyd algorithm was initialized so that circle 1 has no centers, circle 3 has two centers, and all other circles have one center. (Bottom) The Lloyd algorithm erroneously combines the points in circles 1 and 2 into a single cluster and splits circle 3 into two clusters.

STOP and Think. How would you change the Lloyd algorithm's initialization step to improve clustering?

Since initializing the Lloyd algorithm using **FarthestFirstTraversal** (to pick *k* centers that are far away from each other) is too sensitive to outliers, we will describe an alternative **k-means++ Initializer algorithm**. Similarly to **FarthestFirstTraversal, k-means++** picks the *k* centers one at a time, but instead of choosing the point farthest from those picked so far, it chooses each point at random in such a way that distant points are chosen with higher probability than points nearby. In particular, the probability of choosing a center *DataPoint* from *Data* is proportional to the squared distance of *DataPoint* from the centers already chosen, i.e., to $d(DataPoint, X)^2$.

```
k-means++Initializer(Data, k)
   DataPoint ← randomly chosen point from Data
   X ← set consisting of a single point DataPoint
  while |X| < k
    select a point DataPoint from Data at random with probability proportional to $d(DataPoint,X)^2$
    add DataPoint to X
  return X
```

Exercise Break: If you were disappointed with the results of your implementation of the Lloyd algorithm, power it up with **k-means++Initializer** – you may need it in the next section when we start analyzing gene expression data.

Clustering Yeast Genes Implicated in the Diauxic Shift

Although the diauxic shift is an important event, it has no bearing on most of the yeast's functions. We thus expect that most *S. cerevisiae* genes are not involved in the diauxic shift, and that their expression will hardly changes during the experiment. We will therefore exclude these genes from further consideration, thus reducing the size of the expression matrix we will be working with.

Given the expression vector $(E_1, \ldots, E_m)$ of a gene, one can compute its mean $\mu = 1/m * \sum_{i=1,m} E_i$ and variance $1/m * \sum_{i=1,m} (E_i-\mu)^2$. It turns out that most yeast genes have expression vectors with very small variance, meaning that their expression hardly changes before and after the diauxic shift. We will exclude these low-variance genes from our analysis, leaving us with only 264 genes whose expression significantly changes around the diauxic shift.

Although the variance of an expression vector tells us how much a gene's expression changes over time, it does not reveal the gene's *pattern* of change, which can vary greatly. Moreover, many genes share the same pattern of change over time, and thus appear to be co-regulated. Our goal is to determine how many such distinct patterns exist and reveal subsets of genes with similar patterns. As we mentioned, the choice of the number of clusters *k* often requires exploring many values of *k* and then deciding which value makes the most sense biologically. Because tweaking the value of *k* can be a difficult task, we will ask you, somewhat arbitrarily, to partition the yeast dataset into six clusters.

Exercise Break. Apply the Lloyd algorithm to the yeast dataset and partition this dataset into six clusters.

Figure below visualizes expression vectors for each cluster obtained after applying the Lloyd algorithm to the yeast dataset (see DETOUR: Visualization of Gene Expression Matrices to learn more about how biologists visualize clustering of gene expression data).

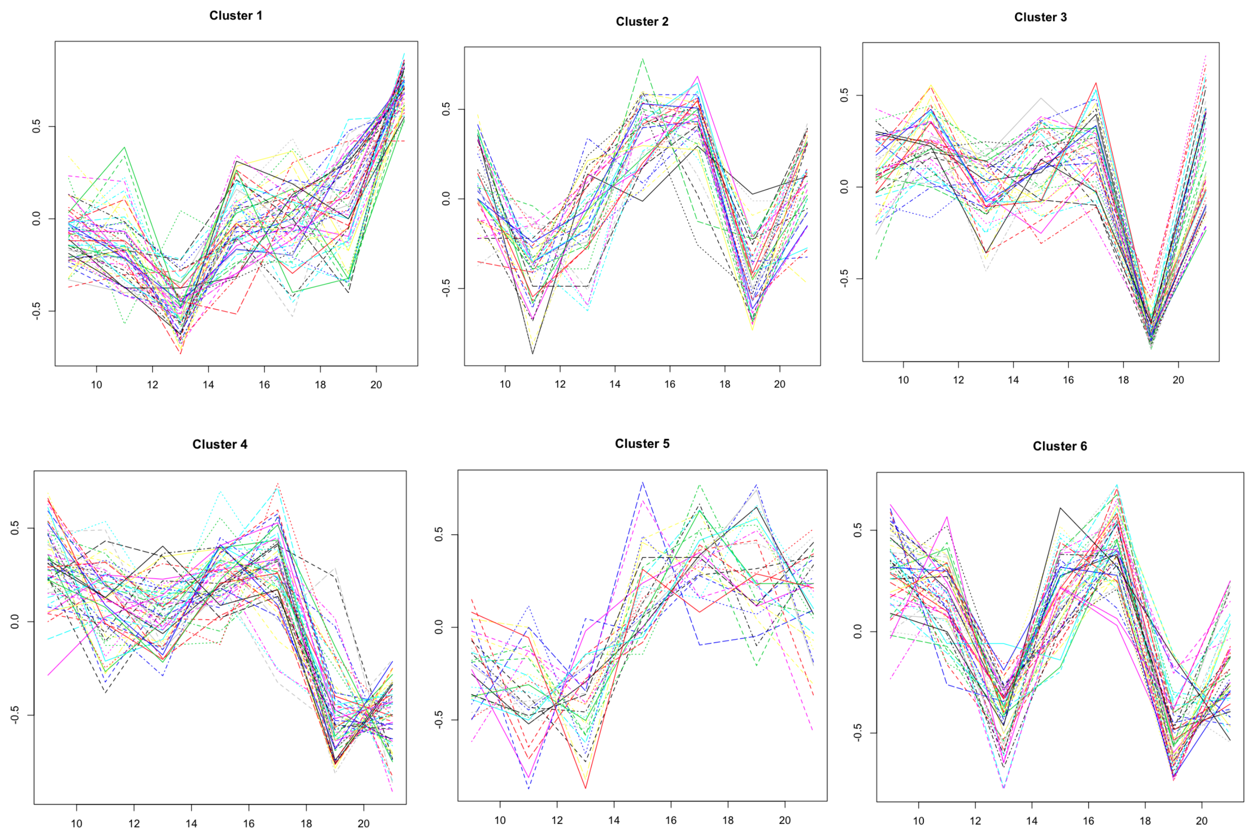


Figure. Applying the Lloyd algorithm (with six centers) to the yeast dataset of 264 genes results in clusters containing (from top left to bottom right) 54, 36, 42, 54, 28, and 50 genes. Expression vectors for all genes in each of these six clusters are visualized as six separate plots.

To reveal the patterns of the expression vectors in each cluster, we average them (Figure below).

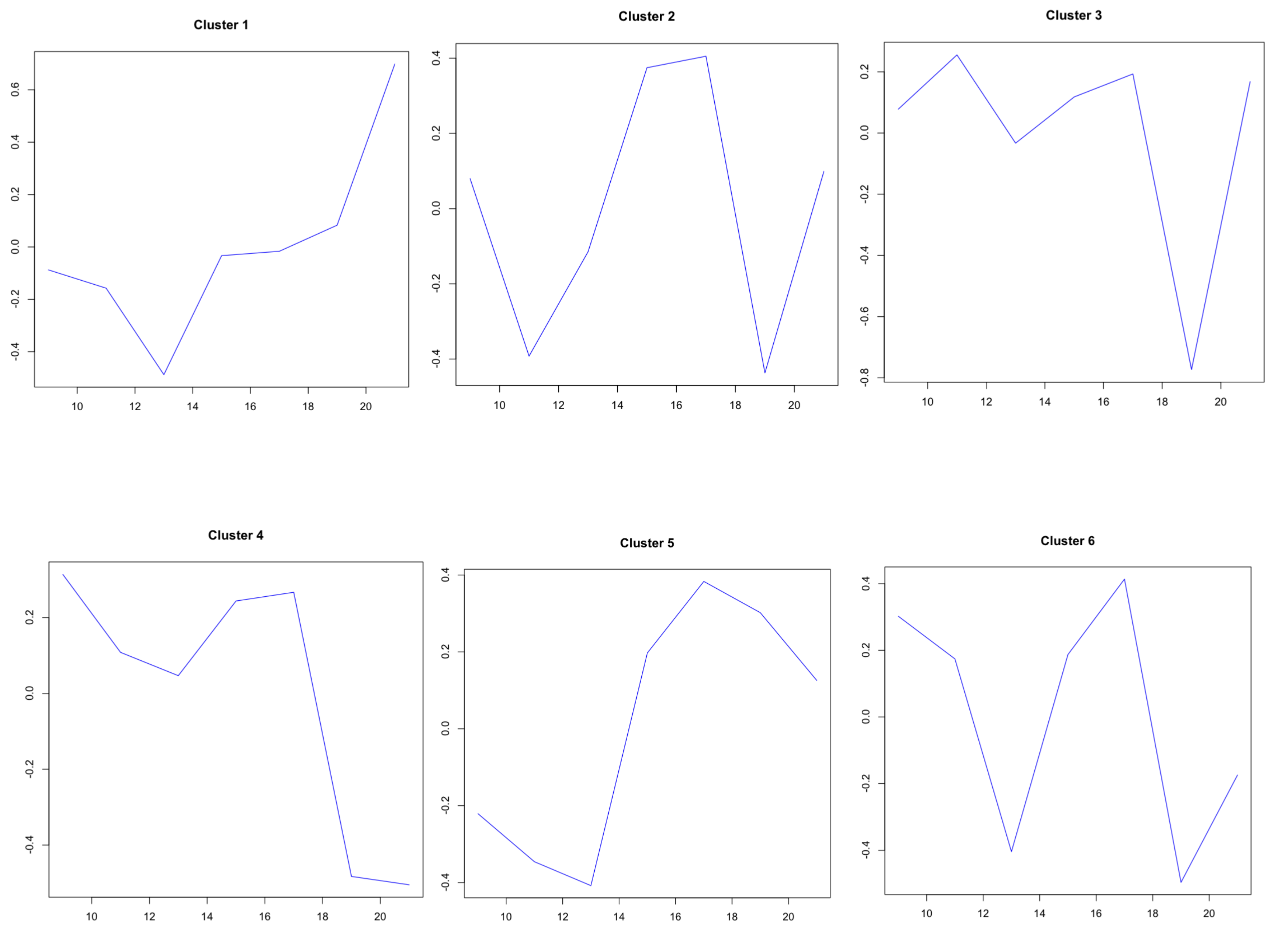


Figure. Averaging the expression vectors in each plot from Figure above reveals six types of regulatory behavior. **We will need to crop these figure (and previous one) from the bottom since the hours are not important.**

The six plots in Figure above reveal six different types of regulation of various genes involved in the diauxic shift. These different patterns of regulation raise various questions for further biological studies. For example, what regulatory mechanisms force genes in the first cluster to increase their expression? What causes genes in the third cluster to decrease their expression? And how do these changes contribute to the diauxic shift?

Limitations of k-means Clustering

How would you cluster the data in Figure below?

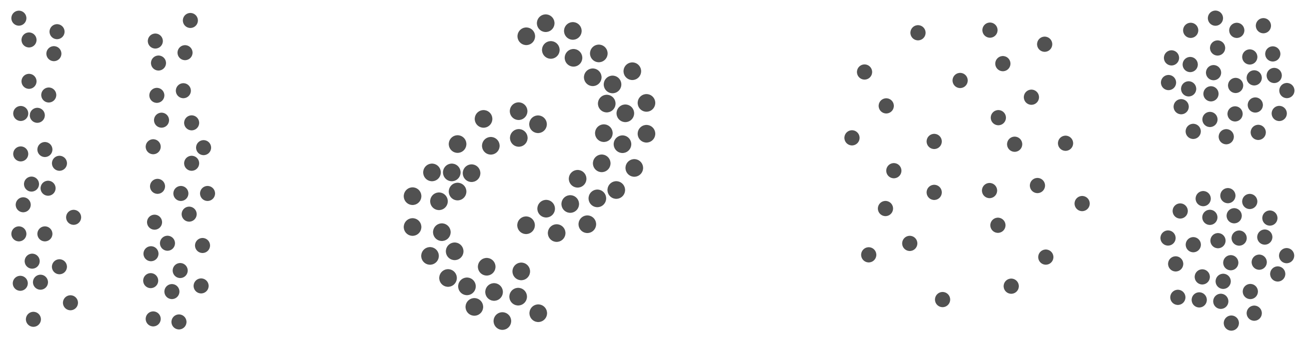


Figure. Difficult clustering problems for *k* = 2 (left and middle) and *k* = 3 (right). **THE FIGURE NEEDS TO BE REDRAWN WITHOUT COLORS**

It turns out that, in the case of challenging clustering problems, the human eye and the Lloyd algorithm may disagree (Figure below).

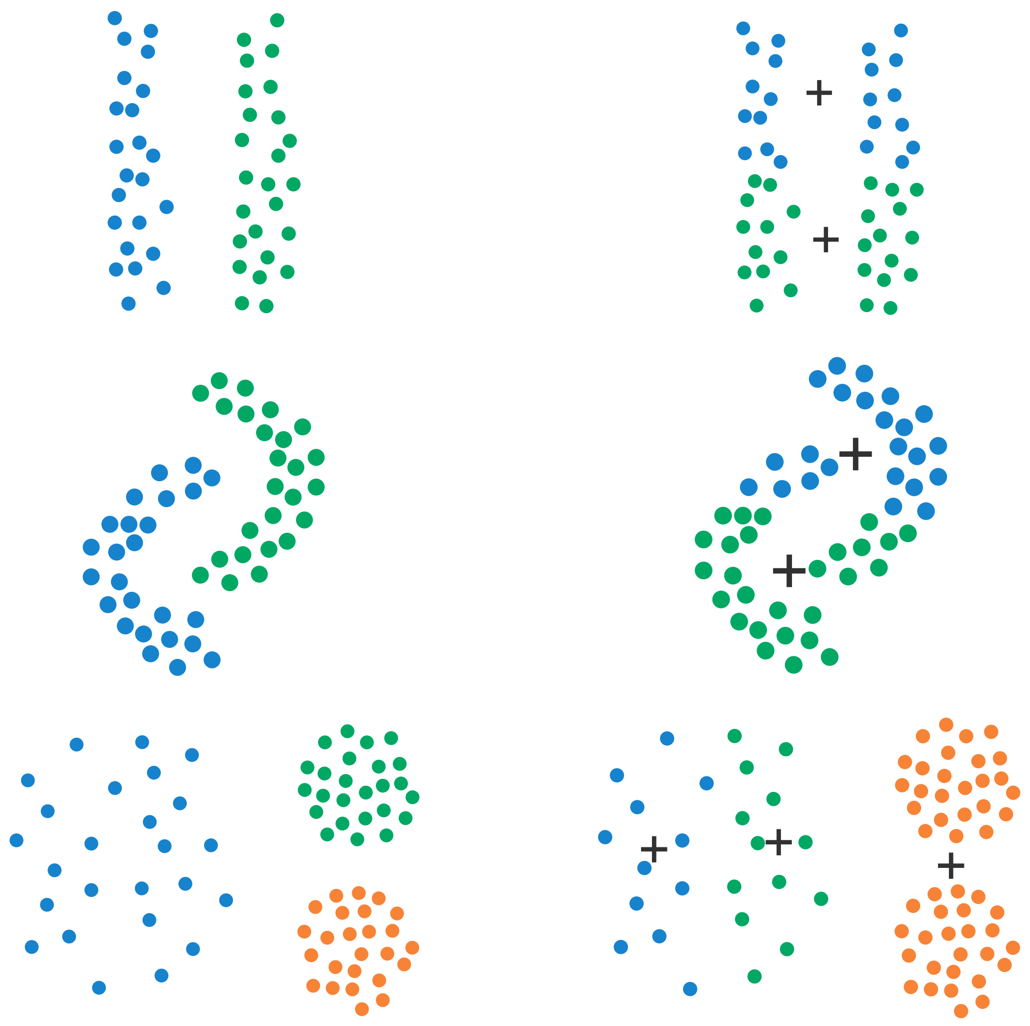


Figure. The human eye (left) and the Lloyd algorithm (right) often provide different solutions in the case of elongated clusters (top), clusters with non-globular shapes (middle), and clusters with widely different densities (bottom). Centers of gravity of clusters on the right are shown by crosses.

STOP and Think: The Lloyd algorithm assign each data point to the cluster corresponding to a closest center with ties broken arbitrarily. What are the negative effects of such arbitrary decisions?

Given a set of centers, we call a data point a **midpoint** if it is equidistant (or nearly equidistant) from two centers (Figure below). The Lloyd algorithm assigns a midpoint to a cluster corresponding to either one or another closest center (ties are broken arbitrarily). However, a "hard" assignment of a midpoint to a specific cluster may be problematic since there is a good reason to assign it to another cluster. In such cases, the Lloyd algorithm may perform poorly since midpoints play the same role as all other data points in assignment of clusters (and thus in assignments of positions of the centers). A better solution would be to modify the Lloyd algorithm so that the points on the border of two clusters should be less important in deciding on the positions of the centers of these clusters.

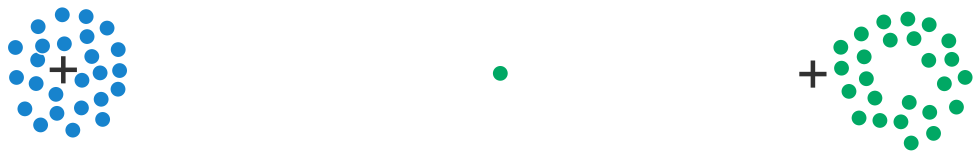


Figure. A point approximately halfway between two clusters skews the center of gravity of the cluster to which it belongs (centers of clusters are shown by "+" sign). A modification to the Lloyd algorithm would allow this point equal influence over both clusters.

Thus, our goal is to find an algorithm that can make "soft" decisions about cluster membership by allowing data points to belong to more than one cluster. To this end, we will take what seems to be a completely unrelated historical detour.

Expectation Maximization

Flipping biased coins. To address some limitations of the Lloyd algorithm, **soft k-means algorithm** makes soft rather than hard decisions about cluster membership by allowing data points to belong to more than one cluster and taking into account the magnitudes of distances between points and centers. To introduce soft k-means clustering, we first describe the idea of the **expectation maximization** algorithm by using an example from a review article by Batzoglou and Do.

Imagine that you walk into a crooked casino where a dealer flips a coin with an unknown bias θ, i.e., on any flip, the coin will land on heads with probability θ. Your goal is to estimate θ. The dealer flips the coin *n* times and it lands on heads *i* out of *n* times. For each θ, we can compute the probability of the resulting sequence of flips and our goal is to find θ that maximizes this probability.

STOP and Think. Given the sequence of coin flips, what is the best estimate of θ?

The probability of generating a given sequence of *n* flips containing *i* heads is $f(\theta) = \theta^i * (1-\theta)^{n-i}$. Because we are searching for a value of θ maximizing this expression, we will take the derivative of *f*(θ), which is

$$f'(\theta) = i * \theta^{i-1} * (1-\theta)^{n-i} - \theta^{i} * (n-i) * (1-\theta)^{n-i-1} = (i * (1-\theta) - \theta * (n-i)) * \theta^{i-1} * (1-\theta)^{n-i-1}.$$

We are searching for value of θ where this derivative turns into 0:

$$f'(\theta) = i * (1-\theta) - \theta * (n-i) = 0$$

The solution is given by $\theta = i/n$ (as you have probably expected from the very beginning) implying that the fraction of heads out of all flips provides the best estimate for θ, i.e.,

$$\theta_A = \text{\#heads generated in all flips/total \# of flips}$$

<u>*A crooked dealer with two coins.*</u> Next, imagine that the dealer has two identical looking coins *A* and *B*, with (unknown) respective probabilities θ_A and θ_B of resulting in heads. At five different times during the night, you observe the dealer selects one of these coins (but you don't know which one) and flips it *n* = 10 times, resulting in the following sequences of heads ("H") and tails ("T").

```
H T T T H H T H T H
H H H H T H H H H H
H T H H H H H T H H
H T H T T T H H T T
T H H H T H H H T H
```

You know the number of heads in each of these experiments, represented as a vector *Data*=(*Data*$_1$, *Data*$_2$, *Data*$_3$, *Data*$_4$, *Data*$_5$)=(5, 9, 8, 4, 7) but you do not know which coin was used in each of these 5 experiments.

STOP and Think. How would you estimate the probabilities θ_A and θ_B?

If you were told what coin was used in each experiment, e.g., if you were given a vector *HiddenVector*=(0, 1, 1, 0, 1) where 1 stands for coin A and 0 stands for coin B, then you would estimate *Parameters*=(θ_A,θ_B) as before (Figure below):

$$\theta_A = \text{\#heads generated in all flips with coin } A\text{/total \# of flips with coin } A$$

$$\theta_B = \text{\#heads generated in all flips with coin } B\text{/total \# of flips with coin } B$$

	Data	*HiddenVector*		*Parameters*
H T T T H H T H T H	5	0		
H H H H T H H H H H	9	1		
H T H H H H H T H H	8	1	➔	θ_A=(5+4)/20=0.45
H T H T T T H H T T	4	0		θ_B=(9+8+7)/30=0.80
T H H H T H H H T H	7	1		

Figure. Computing *Parameters* = (θ_A, θ_B) from *Data* and *HiddenVector*.

STOP and Think: We just saw that given *Data* and *HiddenVector*, we can find *Parameters*. If you were given *Data* and *Parameters*, can you find the most likely choice of *HiddenVector*?

If *Parameters* is known, we can decide whether coin *A* or coin *B* was more likely to have generated the *n* observed flips in each of the five coin flipping experiments by simply comparing the probabilities of generating each sequence with coin A or coin B. Indeed, If $\theta_A^{xi}\ (1-\theta_A)^{n-xi} > \theta_B^{xi}\ (1-\theta_B)^{n-xi}$, we guess that experiment *i* used *A*; otherwise, it used *B* (ties are broken arbitrarily).

	Data	*Parameters*		*HiddenVector*
`H T T T H H T H T H`	5			0
`H H H H T H H H H H`	9			1
`H T H H H H H T H H`	8	θ_A=0.45	➔	1
`H T H T T T H H T T`	4	θ_B=0.80		0
`T H H H T H H H T H`	7			1

Figure. Computing *HiddenVector* from *Data* and *Parameters*. For example, used coin A (the first element of *HiddenVector* is 0) because $0.45^5\,(1\text{-}0.45)^5 > 0.80^5\,(1\text{-}0.80)^5$.

In summary, if *HiddenVector* is known and *Parameters* is unknown, we can reconstruct *Parameters:*

$$(Data,\ HiddenVector,\ ?) \longrightarrow Parameters$$

Likewise, if *Parameters* is known and *HiddenVector* is unknown, we can reconstruct *HiddenVector*:

$$(Data,\ ?\ ,\ Parameters) \rightarrow HiddenVector$$

Our problem, however, is that both *HiddenVector* and *Parameters* are unknown:

$$(Data,\ ?\ ,?) \longrightarrow\ ???$$

This problem may appear hopeless, but recall Chapter 3, where we learned that starting from a random guess is not necessarily a bad idea. Thus, we will start from an arbitrary choice of *Parameters*=(θ_A,θ_B) and immediately reconstruct their most likely *HiddenVector*:

$$(Data,\ ?,\ Parameters) \rightarrow HiddenVector$$

As soon as we know *HiddenVector*, we will question our wisdom in our initial choice of *Parameters* and re-estimate them:

$$(Data,\ HiddenVector,\ ?) \longrightarrow Parameters'$$

Finally, we repeat these two steps and watch how *Parameters* and *HiddenVector* are getting closer and closer to the hopefully correct ones:

$$(Data,?,Parameters) \rightarrow (Data,\ HiddenVector,\ Parameters) \rightarrow (Data,\ HiddenVector,?) \rightarrow$$
$$(Data,\ HiddenVector,\ Parameters') \rightarrow (Data,?,Parameters') \rightarrow (Data,\ HiddenVector',\ Parameters')\ldots \rightarrow\ldots$$

Exercise Break. Prove that this process converges.

STOP and Think. Note that we have never properly formulated the problem we have been trying to solve and have never described what the "correct solution" means! Can you formulate the computational problem we have been trying to solve in this section?

For example, Figure below illustrates the above process for an initial choice of *Parameters*=(0.65, 0.70).

	Data	*Parameters* ➔	*HiddenVector* ➔	*Parameters'* ➔	*HiddenVector'*
`H T T T H H T H T H`	5		0		0
`H H H H T H H H H H`	9		1		1
`H T H H H H H T H H`	8	θ_A=0.65	1	θ_A=0.45	1
`H T H T T T H H T T`	4	θ_B=0.70	0	θ_B=0.8	0
`T H H H T H H H T H`	7		1		1

Vu, can you generate the data above for Parameters=(0.6, 0.5)?

Exercise Break: If *HiddenVector* consists of all zeroes, then the total number of flips with coin *A* equals 0, and the formula for computing θ_A does not work. Change the described formula for computing *Parameters* to address this complication **.**

From coin flipping to k-means clustering. You may think that the coin flipping in the crooked casino has nothing to do with *k*-means clustering, but it does!

STOP and Think: What are *Data, HiddenVector*, and *Parameters* in the Lloyd algorithm?

Given data points *Data=(Data*$_1$*,…,Data*$_n$*),* we will represent their assignment to *k* clusters as an *n*-dimensional vector *HiddenVector = (HiddenVector*$_1$*,…, HiddenVector*$_n$*),* where each *HiddenVector*$_i$ can take integer values from 1 to *k*. We will then represent the *k* cluster centers as *Parameters=*(θ_1*, … ,* θ_k*).* In *k*-means clustering, similarly to the coin flipping challenge, we are given *Data*, but *HiddenVector* and *Parameters* are unknown. The Lloyd algorithm starts from randomly chosen *Parameters* and iteratively perform two steps:

- **From Centers to Clusters**: (*Data, ?, Parameters*) → *HiddenVector*
- **From Clusters to Centers**: (*Data, HiddenVector,?*) → *Parameters*

STOP and Think: Consider the following related questions, the first regarding the crooked casino and the second regarding clustering.

1. How would you select between coins *A* and *B* if $p_A^{x_i}(1-p_A)^{n-x_i} = p_B^{x_i}(1-p_B)^{n-x_i}$? Is it "fair" to always select A if $p_A^{x_i}(1-p_A)^{n-x_i}$ is only slightly larger than $p_B^{x_i}(1-p_B)^{n-x_i}$?
2. If a data point is equally distant from two centers, what cluster will you assign it to? Is it "fair" to always assign it to one center if it is only slightly closer to this data point than another center?

STOP and Think. If a data point is equally distant from two centers, what cluster will you assign it to? Is it "fair" to always assign it to one center if it is only slightly closer to this data point than another center?

From hard to soft choices. The expectation maximization algorithm further develops the idea of the algorithm that we just described. Given parameters *Parameters=*(θ_A, θ_B), we now know how to make hard decisions by selecting a single most likely coin *A* or *B* in each coin flipping experiment:

$$(Data,\ ?,\ Parameters) \rightarrow HiddenVector$$

For example, when *Parameters=*(θ_A, θ_B)*=*(0.6, 0.5), we will select *B* for the coin flipping experiment HHHHTHHHHH because $0.6^9 * 0.4^1 = 0.0040310784$ is larger than $0.5^9 * 0.5^1 = 0.0009765625$. But how about making "soft" decisions about which coin generated the sequence: e.g., deciding that A has a "responsibility" *p* and B has a "responsibility" *1-p* for this sequence of flips?

In terms of clustering, if a data point is exactly halfway between two centers, each of these centers should have the same responsibility for attracting it to their clusters. We demand that the responsibilities of all centers for a given data point should sum to 1.

STOP and Think: Given *Parameters* = (0.6, 0.5) and the sequence of coin flips "HHHHTHHHHH", how would you come up with responsibilities (probabilities) for *A* and *B*?

To answer the preceding question, note that since the probability that coin *A* generated this outcome ($0.6^9 * 0.4^1 = 0.0040310784$) is approximately four times larger than the probability that coin *B* generated this outcome, ($0.5^9 * 0.5^1 = 0.0009765625$), we can assign the following responsibilities to coins A and B:

$$0.0040310784/(0.0040310784+0.0009765625) \approx 0.80$$
$$0.0009765625/(0.0040310784+0.0009765625) \approx 0.20$$

As a result, instead of a vector *HiddenVector*, we now have a 2 x 5*x5* **responsibility profile** *HiddenMatrix* that can be constructed from *Data* and *Parameters*:

$$(Data,\ ?,\ Parameters) \rightarrow HiddenMatrix$$

Figure below illustrates the computation of *HiddenMatrix* for *Data*=(5,9,8,4,7) and *Parameters* = (0.6,0.5). However, the question remains how to compute *Parameters* from *Data* and *HiddenMatrix*:

$$(Data,\ HiddenMatrix,\ ?) \rightarrow Parameters$$

	Data	*Parameters*		*HiddenMatrix*	
H T T T H H T H T H	5			0.45	0.55
H H H H T H H H H H	9			0.80	0.20
H T H H H H H T H H	8	$\theta_A = 0.6$	➔	0.73	0.27
H T H T T T H H T T	4	$\theta_B = 0.5$		0.35	0.65
T H H H T H H H T H	7			0.65	0.35

Figure. Computing *HiddenMatrix* from *Data* and *Parameters*. Compare with Figure XXX for the same choice of *Data* and *Parameters*.

Soft choices in parameter estimation. The **expectation maximization** algorithm alternates between an **E-step**, in which we guess a responsibility matrix *HiddenMatrix* for *Data* given *Parameters:*

(*Data*, ?, *P*arameters) → *HiddenMatrix*

and the **M-step**, in which we re-estimate *Parameters* using *HiddenMatrix*:

(*Data*, *HiddenMatrix*, ?) → *Parameters*

We have seen how to carry out the E-step for the crooked casino. As for the M-step, let us return to making hard choices, where we will take the liberty to rewrite the formula for computing *Parameters*=(θ_A, θ_B) from *Data* and *HiddenVector* (our goal is to show that a similar formula can be used for the M-step).

θ_A = #heads generated in all flips with coin A/total # of flips with coin A=
($\sum_{\text{all experiments } i \text{ generated with coin A}}$#heads in experiment *i*)/($\sum_{\text{all experiments } i \text{ generated with coin A}}$ # of flips in experiment *i*)=
($\sum_{\text{all experiments } i}$ $HiddenVector_i$*#heads in experiment *i*)/($\sum_{\text{all experiments } i}$ $HiddenVector_i$* # of flips in experiment *i*)=
($\sum_{\text{all experiments } i}$ $HiddenVector_i * Data_i$) /($\sum_{\text{all experiments } i}$ $HiddenVector_i * n$)=

Similarly,

θ_B = ($\sum_{\text{all experiments } i}$ (1- $HiddenVector_i$)*$Data_i$) /($\sum_{\text{all experiments } i}$ (1- $HiddenVector_i$)*n).

Thus, for *Data*=(5, 9, 8, 4, 7) and *HiddenVector*=(0, 1, 1, 0 1), θ_A and θ_B are estimated as

θ_A = (0*5+1*9+1*8+0*4+1*7)/ (0*10+1*10+1*10+0*10+1*10)=24/30=0.8

θ_B = (1*5+0*9+0*8+1*4+0*7)/ (1*10+0*10+0*10+1*10+0*10)=9/20=0.0.45

Since hard assignments correspond to the responsibility profile

$$HiddenMatrix = \begin{pmatrix} HiddenMatrix_{A,1} & HiddenMatrix_{A,2} & HiddenMatrix_{A,3} & HiddenMatrix_{A,4} & HiddenMatrix_{A,5} \\ HiddenMatrix_{B,1} & HiddenMatrix_{B,2} & HiddenMatrix_{B,3} & HiddenMatrix_{B,4} & HiddenMatrix_{B,5} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

the first (second) row of *HiddenMatrix* is equal to *HiddenVector (1-HiddenVector).* We rewrite the formulas above in terms of *HiddenMatrix*:

θ_A =($\sum_{\text{all experiments } i}$ $HiddenMatrix_{A,i} * Data_i$) /($\sum_{\text{all experiments } i}$ $HiddenMatrix_{A,i} * n$)=
θ_A =($\sum_{\text{all experiments } i}$ $HiddenMatrix_{B,i} * Data_i$) /($\sum_{\text{all experiments } i}$ $HiddenMatrix_{B,i} * n$)=

We will be using exactly the same formula for soft choices in the M-step!

We illustrate the similarities and differences between making hard and soft choices using the example of *Data*=(5, 9, 8, 4, 7). In the case of hard choices and *HiddenVector*=(0, 1, 1, 0 1), θ_A and θ_B are estimated as

θ_A = (0*5+1*9+1*8+0*4+1*7)/ (0*10+1*10+1*10+0*10+1*10)=24/30=0.8

θ_B = (1*5+0*9+0*8+1*4+0*7)/ (1*10+0*10+0*10+1*10+0*10)=9/20=0.0.45

In the case of soft choices and the responsibility profile

$$HiddenMatrix = \begin{matrix} 0.45 & 0.80 & 0.73 & 0.35 & 0.65 \\ 0.55 & 0.20 & 0.27 & 0.65 & 0.35 \end{matrix}$$

the M-step results in the following estimate for *Parameters*:

θ_A =(0.45*5+0.80*9+0.73*8+0.35*4+0.65*7)/ (0.45*10+0.80*10+0.73*10+0.35*10+0.65*10)=21.3/29.8≈0.71
θ_B =(0.55*5+0.20*9+0.27*8+0.65*4+0.35*7)/ (0.55*10+0.20*10+0.27*10+0.65*10+0.35*10)=11.7/20.1≈0.58

Soft *k*-means clustering

We are now ready to modify the Lloyd algorithm to make soft decisions about cluster membership. Given *n* points $Data = \{Data_1, \ldots, Data_n\}$ and *k* centers $X = \{x_1, \ldots x_k\}$, the **Soft *k*-means Clustering Algorithm** is inspired by the Expectation Maximization Algorithm from the preceding section. This algorithm first constructs an *n* x *k* responsibility matrix *HiddenMatrix*. The closer the data point *i* is to a center *j* (in comparison to other centers), the larger the responsibility $HiddenMatrix_{i,j}$. For example, this responsibility can be defined as

$$HiddenMatrix_{i,j} = 1/d(Data_i,x_j)^2 / \sum_{\text{all centers } j} 1/d(Data_i, x_j)^2$$

where $d(Data_i,x_j)$ is the distance between data point $Data_i$ and center x_j. The construction of the responsibility matrix is based on the implicit assumption that a point $Data_i$ is a "planet" and the center x_j is a "star", and computes responsibility according to the Newtonian inverse-square law of gravitation. Unfortunately for Newton fans, it turns out that the following formula motivated by statistical physics often works better in practice:

$$HiddenMatrix_{i,j} = e^{-\beta \cdot d(Datai,xj)} / \sum_{\text{all centers } j} e^{-\beta \cdot d(Datai,xj)}$$

In this formula, β is a constant called the **stiffness parameter**. Figure below illustrates various approaches for computing *HiddenMatrix* in the case when *Data* represents points in one-dimensional space.

STOP and Think: What is the difference between the Lloyd algorithm and the Soft *k*-means Clustering Algorithm when the stiffness parameter is very large, i.e., $\beta \to \infty$? How would the Soft *k*-means Algorithm perform when $\beta = 0$ and when $\beta \to -\infty$?

Figure. *HiddenMatrix* constructed for the five points *Data*={-2, -1, 0, +1, +2} and the two centers *X*=(-1,+1) for $HiddenMatrix_{i,j} = 1/d(Data_i,x_j)^2 / \sum_{\text{all centers } j} 1/d(Data_i,x_j)^2$ (top) and $HiddenMatrix_{i,j} = e^{-\beta \cdot d(Datai,xj)} / \sum_{\text{all centers } j} e^{-\beta \cdot d(Datai,xj)}$ for β=1 (middle) and β=2 (bottom). **MISSING FIGURE**

Figure below illustrates how *HiddenMatrix* changes with respect to the stiffness parameter.

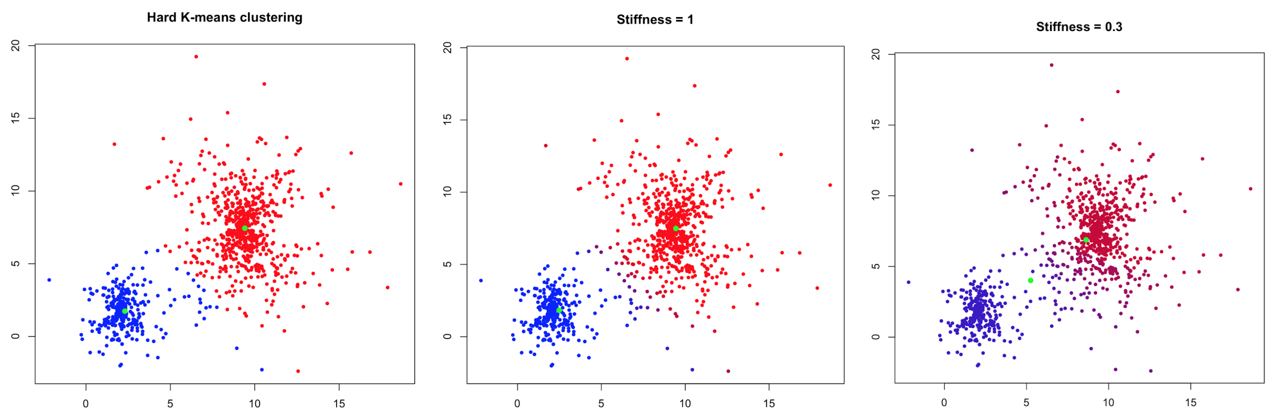


Figure. The results of the Lloyd algorithm Hard *k*-means clustering (left) and soft *k*-means clustering algorithms for β = 1 (middle) and β = 03 (right). Responsibility profiles for “red” and “blue” centers and two choices of the stiffness parameter, β=1 (middle), and β = 0.3 (right). The colors of the points vary across a spectrum from “pure red” to “pure blue” in such a way that the fraction of blue and red used for coloring a point is equal to the ratio of the corresponding responsibilities. Increasing the stiffness parameter increases the “contrast” by altering the ratio of blue and red used for coloring each data point. Cluster centers are shown as green points and are positioned at (2.30, 1.73) and (9.41, 7.44) (left), (2.48, 1.83) and (9.44, 7.49) (middle), and (5.25, 4.00), (8.62, 6.88) (right).

The centers of clusters are then updated as the **weighted center of gravity** of all data points:

$$x_j = \sum_{\text{all data points } i} HiddenMatrix_{i,j} * Data_i \; / \sum_{\text{all data points } i} HiddenMatrix_{i,j}$$

The intuition behind this formula is that center x_j is responsible only for a fraction of data point $Data_i$ corresponding to $HiddenMatrix_{i,j}$. Thus, each data point should only contribute this fraction to the center of gravity of all points in cluster *j*.

STOP and Think: Does the Soft *k*-means Clustering algorithm converge? If not, how would you modify it to ensure that it does not run forever?

Code Challenge: Implement the Soft *k*-means Clustering (SKMC) algorithm.

Exercise Break: Apply Soft *k*-means Clustering to the yeast diauxic shift gene expression data and compare the results with the results of the Lloyd algorithm.

Hierarchical Clustering

In Chapter X, we covered two types of approaches to evolutionary tree reconstruction with different strengths and weaknesses: alignment-based algorithms (including the Sankoff algorithm for the Small Parsimony Problem), and distance-based algorithms (including the Neighbor-Joining algorithm). Likewise, gene expression studies often use algorithms based on analyzing the gene expression matrix (like the Lloyd algorithm for k-means clustering) and the similarity-based algorithm that we will cover below.

Instead of analyzing the $n \times m$ expression matrix directlly, biologists sometimes first transform it into an $n \times n$ **similarity matrix** R, where $R_{i,j}$ indicates the similarity of the expression vectors for genes i and j (Figure below). Given a similarity matrix, we still would like to group genes into clusters satisfying the homogeneity and separation conditions discussed above.

There are many ways to quantify the similarity between expression vectors $x=(x_1,\ldots,x_m)$ and $y=(y,\ldots,y_m)$. One possiblility is the **dot product**, $\sum_{i=1,m} x_{i*} y_{i,}$; another is the **Pearson correlation coefficient** $r(x, y)$,

$$r(x, y) = \sum_{i=1,m} (x_i-x^*)\ (y_i-y^*)\ /\ sqrt\ (\sum_{i=1,m} (x_i-x^*)^2\ (y_i-y^*)^2),$$

where x^* and y^* denote the means of all coordinates of x and y, respectively.

STOP and Think: Given a vector x, which vectors y maximize and minimize $r(x,y)$?

The Pearson correlation coefficient varies between -1 and +1, where -1 is total negative correlation, 0 is no correlation, and +1 is total positive correlation. Based on the Pearson correlation coefficient, one can define the **Pearson distance** between expression vectors x and y as *PearsonDistance*$(x,y)=1-r(x,y)$.

Exercise Break: Compute the Pearson correlation coefficient for the following pairs of vectors:

1. $(\cos \alpha, \sin \alpha)$ and $(\sin \alpha, -\cos \alpha)$;
2. $(\sqrt{0.75}, 0.5)$ and $(-\sqrt{0.75}, 0.5)$.

	1 hr	2 hr	3 hr
g_1	10.0	8.0	10.0
g_2	10.0	0.0	9.0
g_3	4.0	8.5	3.0
g_4	9.5	0.5	8.5
g_5	4.5	8.5	2.5
g_6	10.5	9.0	12.0
g_7	5.0	8.5	11.0
g_8	2.7	8.7	2.0
g_9	9.7	2.0	9.0
g_{10}	10.2	1.0	9.2

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	0.7	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

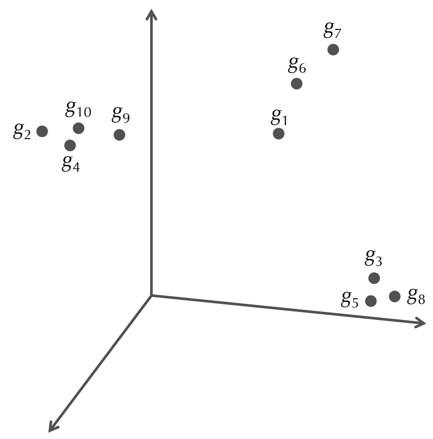


Figure. A toy gene expression matrix of ten genes measured at three time points (left), the distance matrix based on Euclidean distance (right), and the gene expression vectors as points in three-dimensional space (bottom).

In previous sections, we have assumed that we were working with a fixed number of clusters *k*. But in practice, clusters often have subclusters, which have subsubclusters, and so on. To capture clustering information on a variety of different layers, the **Hierarchical Clustering algorithm** first organizes elements into a tree. Figure below shows a tree representing the data in Figure above and describes a family of ten different ways of dividing the data into clusters. Specifically, as shown in Figure below, a horizontal line through the crossing the tree at *i* points corresponds to a division of the ten genes into *i* clusters.

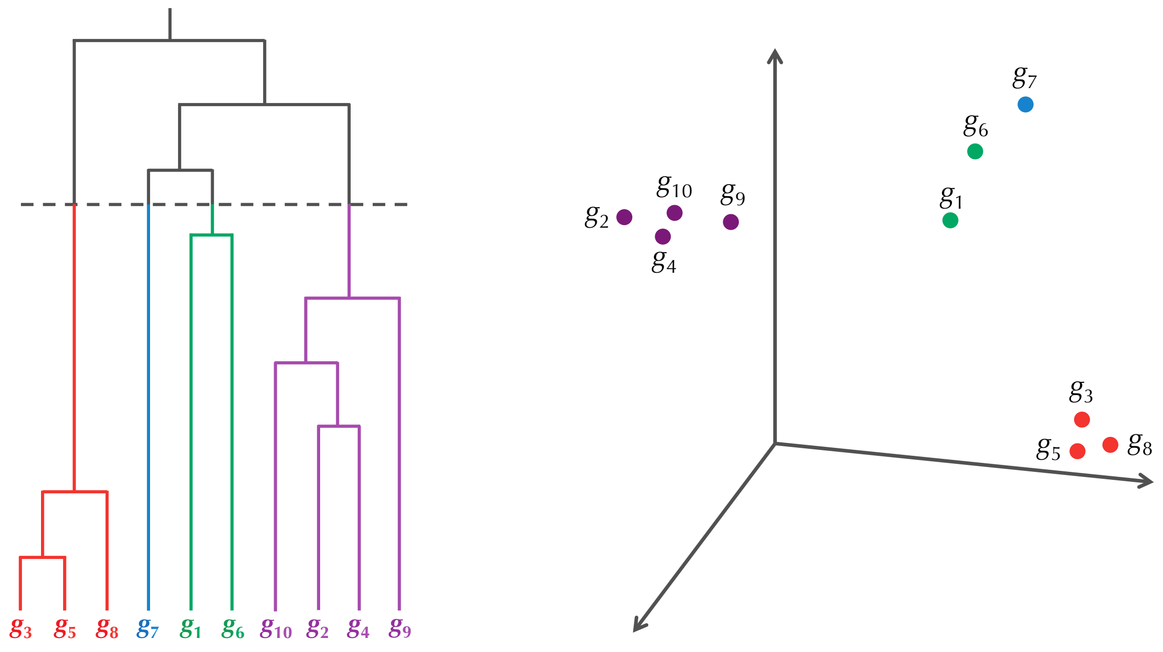

g_3 g_5 g_8 g_7 g_1 g_6 g_{10} g_2 g_4 g_9
g_7
g_6
g_{10}
g_9
g_1
g_2
g_4
g_3
g_5
g_8

Figure. (Left) A hierarchical clustering of the data in Figure above. The horizontal line crosses the data in four places and divides the data into four colored clusters. (Right) The colored points shown in three-dimensional space.

The **HierarchicalClustering** algorithm below starts from an $n \times n$ distance matrix D that is derived from the gene expression matrix. (e.g., by computing Pearson distances). Afterwards, it progressively generates n different partitions of the data into clusters, all represented by a tree in which each node is labeled by a cluster of genes. The first partition has n single-element clusters represented by leaves in the tree, with every element forming its own cluster. The second partition combines the two "closest" clusters, In general, the i-th partition combines the two closest clusters from the $(i-1)$-th partition and has $n-i+1$ clusters.

```
HierarchicalClustering(D, n)
  Clusters ← n single-element clusters labeled 1, …, n
  construct a graph T with n isolated nodes labeled by single elements 1, …, n
  while there is more than one cluster
    find the two closest clusters $C_1$ and $C_2$
    merge $C_1$ and $C_2$ into a new cluster C with $|C_1| + |C_2|$ elements
    add a new node to T, connect it to nodes $C_1$ and $C_2$ by directed edges, and label it by cluster C
    remove rows and columns of D corresponding to $C_1$ and $C_2$
    remove $C_1$ and $C_2$ from Clusters
    add a row and column to D for the new cluster C by defining $D(C,C^*)$ for each cluster $C^*$ in Clusters
    add C to Clusters
  assign root in T as a node with no incoming edges
  return rooted labeled tree T
```

The pseudocode above does not specify how exactly to compute the distances from a newly formed cluster to old ones and clustering algorithms vary in how they compute these distances. One commonly used metric defines the distance between two clusters as the smallest distance between any pair of elements from these clusters,

$$D_{min}(C_1,C_2) = min_{\text{all point } i \text{ in cluster } C1, \text{ all point } j \text{ in cluster } C2} D_{ij} .$$

Another distance function takes the average distance between elements in two clusters,

$$D_{avg}(C_1,C_2) = (\sum_{\text{all point } i \text{ in cluster } C1,} \sum_{\text{all point } j \text{ in cluster } C1,\ C2} D_{ij}) / (|C_1| * |C_2|).$$

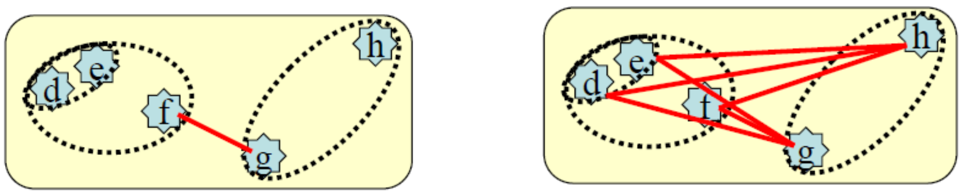


Figure. Defining the distance between cluster as the smallest distance (left) and the average distance (right).

Code Challenge: Implement **HierarchicalClustering** (HC) using the average distance between clusters.

Exercise Break: Apply **HierarchicalClustering** to the yeast dataset, and partition this dataset into six clusters. Do you expect these clusters to be of roughly the same size or of vastly different sizes?

Figure below visualizes expression vectors for each of the six clusters obtained after applying **HierarchicalClustering** to the yeast dataset.

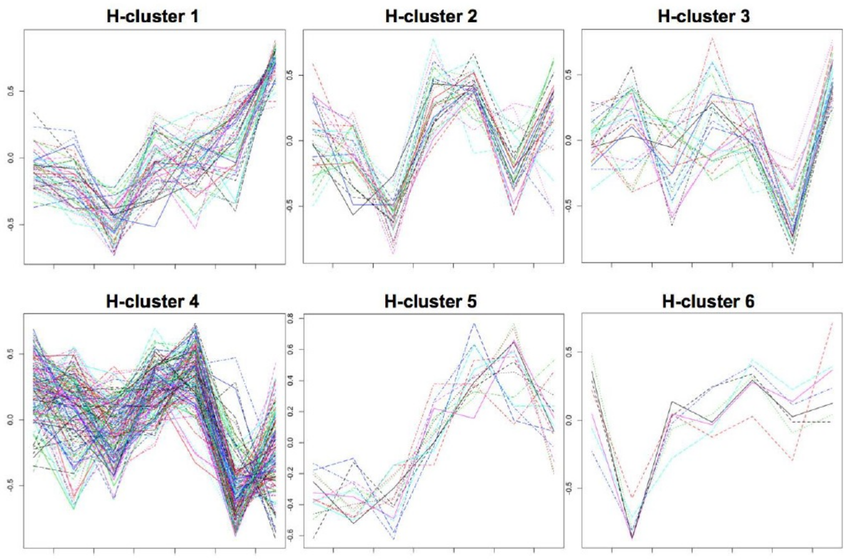


Figure. Applying **HierarchicalClustering** to the yeast dataset results in six clusters with (from top left to bottom right) 48, 27, 23, 145, 14, and 7 genes. Expression vectors for all genes in each of these six clusters are visualized as six separate plots.

To reveal the patterns of the expression vectors in each cluster we average them as shown in Figure. As you can see, the clusters obtained by the **HierarchicalClustering** and the Lloyd algorithm are rather different. You may be concerned that two clustering approaches produced different clusters but this is the fact of life in gene expression studies that often represent the beginning of a large experimental work to confirm that derived cluster make sense biologically and often focus on specific genes within some clusters.

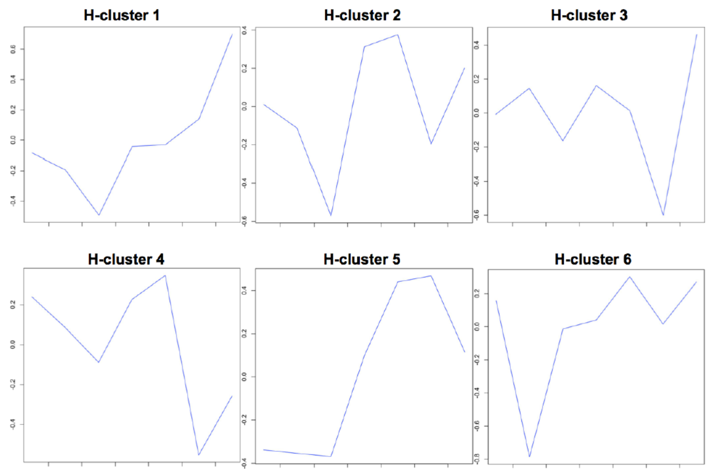


Figure. Averaging the expression vectors in each cluster reveals six types of regulatory patterns of various genes involved in the diauxic shift.

Although this chapter has focused on clustering, we have only just begun exploring the multitude of clustering algorithms that are used in modern bioinformatics. See DETOUR: Clustering and Corrupted Cliques to learn about yet another approach to clustering.

Epilogue: Clustering Tumor Samples

Although this chapter has focused on using gene expression analysis to study the diauxic shift, biologists often use the same expression analysis technologies to study gene expression in a myriad of applications, from analyzing various tissues before and after taking drugs, to studying cancerous versus non-cancerous cells. For example, expression analysis led to developing **MammaPrint**, a diagnostic test that determines the likelihood of breast cancer returning after treatment. MammaPrint is based on gene expression analysis of 70 genes that are associated with activation and suppression of tumors.

In 1999, Uri Alon analyzed gene expression data for 2000 genes from various colon tumor tissues (40 samples) and compared them with colon tissues from healthy individuals (22 samples). The data is represented by a 2000 x 62 gene expression matrix, where the first 40 columns describe tumor samples and the last 22 columns describe normal samples.

Now, suppose you performed a gene expression experiment with a colon sample from a new patient (representing a 63rd column in an augmented gene expression matrix), with the goal of determining whether this patient has a colon tumor. Since we are interested in finding differences between tumor tissues (rather than expression vectors of various genes), it makes sense to construct the 62 x 62 similarity matrix between tissues and cluster the 62 tissues based on this matrix. If we obtain a cluster consisting predominantly of cancer tissues, this cluster may help us diagnose colon cancer.

STOP and Think: It may seem that the best way to proceed is to cluster tissues into just two clusters (tumor versus healthy). Why is this not necessarily a good idea?

Challenge Problem: Given the 2000 x 62 gene expression matrix and gene data from a new patient, how would you evaluate whether this patient is likely to have a colon tumor?

Vu has acquired this data

Bibliography

U. Alon, N. Barkai, D. A. Notterman, G. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. PNAS, 96:6745–6750, 1999.

Arthur, D. and Vassilvitskii, S. *k*-means++: the advantages of careful seeding". Proceedings of the Eighteenth ACM-SIAM Symposium on Discrete Algorithms. 1027–1035, 2007

Batzoglou S. and Do C. B. What is the expectation maximization algorithm? Nature Biotechnology. 26:897-899, 2008.

A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression vectors. Journal of Computational Biology, 6:281–297, 1999.

Bezdek, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Springer, 1981

Ceppellini, R., Siniscalco, M. & Smith, C.A. The estimation of gene frequencies in a random-mating population. Ann. Hum. Genet. 20, 97–115, 1955

M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression vectors. Proceedings of the National Academy ofSciences of the United States of America, 95:14863–14868, 1998.

Kellis M, Birren BW, Lander ES. Proof and evolutionary analysis of ancient genome duplication in the yeast *Saccharomyces cerevisiae*. *Nature*, 428, 617–624, 2004

S. P. Lloyd. Least squares quantization in PCM. IEEE Transactions on Information Theory, 28:129–137, 1982.

Ohno S. Evolution by Gene Duplication. Allen and Unwin, London, 1970

J. L. DeRisi, V. R. Iyer, P. O. Brown**.** Exploring the Metabolic and Genetic Control of Gene Expression on a Genomic Scale**.** Science, 278:680-686, 1997

Thomson, J.M. et al. Resurrecting ancestral alcohol dehydrogenases from yeast. Nat. Genet. 37, 630–635, 2005

Wolfe, K.H. and Shields, D.C. Molecular evidence for an ancient duplication of the entire yeast genome. Nature 387, 708–713, 1997

The expectation maximization algorithm was first proposed by Ceppellini, Siniscalco, and Smith in 1955 and was rediscovered many times over by various researchers. Batzoglou and Do provided an excellent primer on expectation maximization. The k-means clustering algorithm was introduced by Stuart Lloyd in 1957. Soft k-means clustering approach was developed by James Bezdek in 1981. David Arthur and Sergey Vassilvitskii developed k-means++ initialization for k-means clustering. The CAST algorithm was developed by Ben-Dor, Shamir, and Yakhini in 1999.

DeRisi, Iyer, and Brown performed the first large-scale gene expression experiment to analyze the diauxic shift in 1997. Eisen, Spellman, Brown and Botstein, 1998 described first applications of hierarchical clustering to gene expression analyses. Uri Alon and colleagues analyzed patterns of gene expression in colon tumors in 1999.

Susumu Ohno came up with the Whole Genome Duplication hypothesis in 1970. Wolfe and Shields provided the first convincing arguments in favor of the Whole Genome Duplication in yeast in 1997. Kellis, Birren, and Lander provided further evidence for Whole Genome Duplication by analyzing various yeast species in 2004. Thomson and colleagues resurrected the sequence of the ancient alcohol dehydrogenases from yeast in 2005.

DETOURS

Finding Synteny Blocks in Highly Duplicated Genomes

Whole Genome Duplications (WGDs) quickly follow by a massive gene loss and rearrangements making it difficult to reconstruct the genomic architecture of the pre-duplicated genomes. Indeed, while all genes in ancestral *S. cerevisiae* had duplicates immediately after WGD ≈100 million years ago (part (a) in Figure below), many of them got lost (part (b) in Figure below), and only 13% of genes have duplicates in the modern-day *S. cerevisiae* (parts (c-d) in Figure below). The question thus arises how one can prove that *S. cerevisiae* has indeed undergone a WGD.

Manolis Kellis solved this puzzle by analyzing *K. waltii*, a related yeast species that has never undergone WGD. He constructed sequence alignments and found that nearly each region (synteny block) of *K. waltii* corresponds to two regions of *S. cerevisiae*, as expected for WGD (part (e) in Figure below).

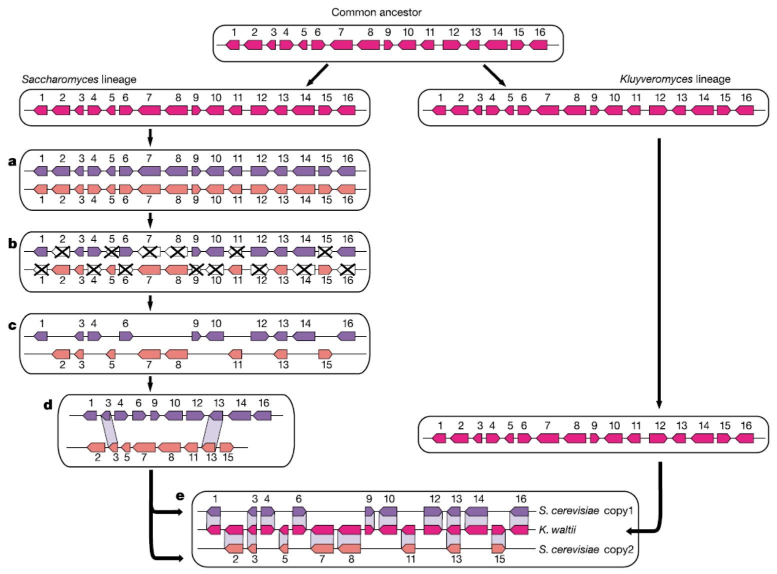


Figure. *K. waltii* "helps" to find paired synteny blocks in *S. cerevisiae.* a) After divergence from *K. waltii*, *S. cerevisiae* underwent a WGD, creating two identical copies of every chromosome. b) The vast majority of duplicated genes underwent mutations resulting in gene losses. c) Sister segments retained different subsets of the original genes, keeping both copies for only a minority of duplicated genes. d) Within *S. cerevisiae*, only a small fraction of genes (2 and 13 in the example above) have duplicates making it nearly impossible to figure out that the shown segments evolved from a single segment in the pre-duplicated genomes. e) However, comparison with *K. waltii* reveals the duplicated nature of the *S. cerevisiae* genome since most synteny blocks in *K. waltii* have two corresponding synteny blocks in *S. cerevisiae*.

How Biologists Measure Gene Expression?

In a proteomics experiment (chapter X), one generates a set of spectra and matches them against the proteome. The number of spectra matching to peptides from a given protein represents a proxy for the expression level of this protein. To estimate the protein expression from this proxy, bioinformaticians should take into account the varying length of proteins, poor fragmentation of some peptides leading to difficulties in their identification, and many other factors.

In an RNA sequencing experiment, one generates reads from **transcriptome** (rather than genome) using an approach known as **RNA-Seq**. An estimate of the number of each protein-coding transcript provides a proxy for the expression of the resulting protein. While many expression analysis studies implicitly assume that the amount of each protein-coding transcript correlates well with the amount of protein encoded by this transcript, this assumption is often incorrect. Indeed, a number of processes affect the production of proteins in the cell beyond transcription (translation, post-translational modifications, protein degradation, etc.) and therefore this correlation is not straightforward.

Finally, one can use customized DNA arrays (chapter 4) that carry probes aimed at each gene in a species of interest. Each probe will be characterized by intensity that is a proxy for the number of transcripts of a given gene present in the sample. One deficiency of DNA arrays (that makes RNA-Seq more attractive) is that such arrays target the identification of known transcripts and often fail to evaluate unknown transcripts. For example, many cancers are caused by rare mutations and would go undetected.

Microarrays

The first microarrays that de Risi and colleagues used to study the diauxic shift were manufactured by spotting pre-made **complementary DNAs** (**cDNA**) on a glass slide. After capturing many mRNA transcripts expressed in yeast cells, they have made a collection of these transcripts and spotted them as cDNAs on a glass as probes. Afterwards, they hybridized fluorescently labeled RNA from a sample of interest to measure expression levels of various yeast genes.

Since there can be large variation in how much cDNA is printed on each spot of the microarray, de Risi and colleagues had to find a way to get around this complication and to make sure that fluorescence intensities are comparable across spots and across arrays. To achieve this goal, cDNA arrays are used by hybridizing two samples to each array. The two samples are labeled with different fluorescent dyes (green and red), that can be read by image-processing software.

The data from a microarray are represented as the ratio of fluorescent intensities of the two samples. These experiments are not meant to measure the absolute level of gene expression, but rather relative changes in the expression of individual genes between samples and time points.

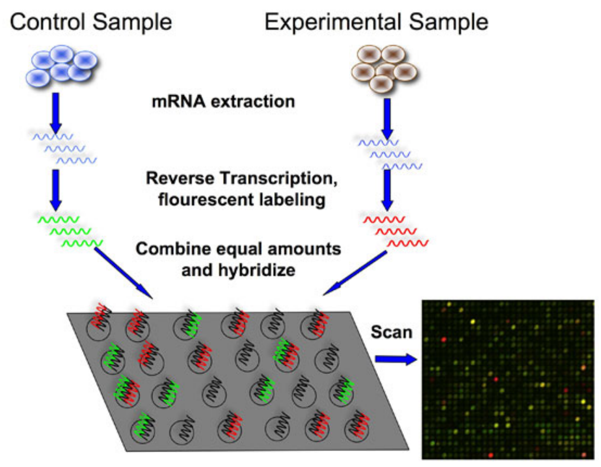

Control Sample
Experimental Sample
mRNA extraction
Reverse Transcription, flourescent labeling
Combine equal amounts and hybridize
Scan

Figure. Microarray.

Vector Norm

Phillip, please add this Detour

Center of Gravity

Theorem. The center of gravity of a set of points *Data* solves the k-Means Clustering Problem for $k = 1$, i.e., it minimizes *Distortion*(*Data, x*) among all possible centers *x*.

Proof. Since the squared distance between $DataPoint=(DataPoint^1,\dots DataPoint^m)$ and center $x=(x^1,\dots,x^m)$ is $\sum_{1\le j\le m} (DataPoint^j - x^j)^2$, we have

$$d(Data,x) = \sum_{\text{all points } DataPoint \text{ in } Data} d(DataPoint,x)^2/n=$$
$$\sum_{\text{all points } DataPoint \text{ in } Data} \sum_{1\le j\le m} (DataPoint^j - x^j)^2/n =$$
$$\sum_{1\le j\le m} \sum_{\text{all points } DataPoint \text{ in } Data} (DataPoint^j - x^j)^2/n$$

The last expression implies that we can independently minimize *d(Data,x)* in each of *m* dimensions by minimizing each of *m* expressions $\sum_{\text{all points } DataPoint \text{ in } Data} (DataPoint^j - x^j)^2$. Since each of these expressions is merely a quadratic function with a single variable x^j, we can find its minimum by computing the derivative of each such function (equal to $-\sum_{\text{all points } DataPoint \text{ in } Data} 2*(DataPoint^j - x^j)$) and finding where it turns into zero. It amounts to solving the equation:

$$\sum_{\text{all points } DataPoint \text{ in } Data} (DataPoint^j - x^j)=0$$

The solution is given by

$$x^j = (\sum_{\text{all points } DataPoint \text{ in } Data} DataPoint^j)/n$$

implying that the *j*-th coordinate of the center is the mean value of the *j*-th coordinates of the data points. Therefore, the solution of the k-Means Clustering Problem in the case of a single center is simply the center of gravity of all data points:

$$x = (\sum_{1\le i\le n} Data_i^j)/n \qquad \square$$

Visualization of Gene Expression Matrices

To make sense of the gene expression matrix biologists often rearrange the genes so that the similar expression vectors become consecutive in the gene expression matrix. For example, given a tree obtained by hierarchical clustering, one can order genes (leaves in the tree) in such a way that neighboring leaves correspond to neighboring rows.

After the ordering of genes is chosen, biologists often assign colors to each element of the gene expression matrix resulting in a **heatmap** of a microarray experiment. For example, black/green/red/ colors may represent entries for which expression is average/high/low. Figure below shows the heatmap of the yeast gene expression matrix along with a tree constructed using the hierarchical clustering approach.

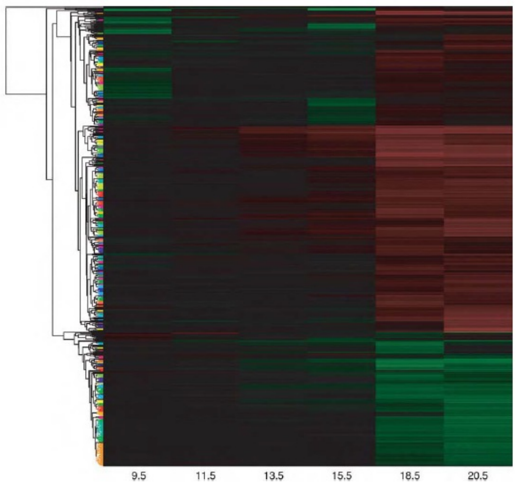


Figure. The heatmap of a gene expression matrix.

Clustering and Corrupted Cliques

In expression analysis studies, the similarity matrix is often transformed into a **similarity graph** *G = G*(θ*),* where the nodes are genes and there is an edge between genes *i* and *j* if and only if the similarity $R_{i,j}$ between them exceed a threshold θ.

STOP and Think. Consider clustering of genes that satisfies the homogeneity (similarity between any two genes within the same cluster exceeds θ) and separation (similarity between any two genes in different clusters does not exceed θ) principles. How the similarity graph *G*(θ*)* looks like for this dataset?

If clustering satisfies the homogeneity and separation principles, then each connected component of *G*(θ*)* is a **complete graph**, i.e., an undirected graph where every two nodes are connected by an edge. A **clique graph** is a graph in which every connected component is a complete graph (Figure below). Every partition of *n* elements into *k* clusters can be represented by a clique graph on *n* nodes with *k* connected components.

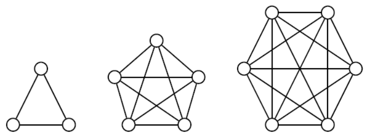


Figure. A clique graph consisting of three connected components.

While clustering satisfying the homogeneity and separation principles results in a clique graph, errors in expression data and the absence of a "universally good" threshold θ often result in "corrupted" similarity graphs (Figure below). Some elements of the similarity matrix may exceed the threshold for unrelated genes (adding edges between different clusters), while other elements may fall below the similarity threshold for related genes (removing edges within clusters). Such erroneous edges corrupt the clique graph, raising the question of how to transform the similarity graph into a clique graph using the smallest number of edge removals and additions.

Corrupted Cliques Problem. Find the smallest number of edges that need to be added or removed to transform a graph into a clique graph.

Input: A graph.

Output: The smallest number of additions and removals of edges that transforms this graph into a clique graph.

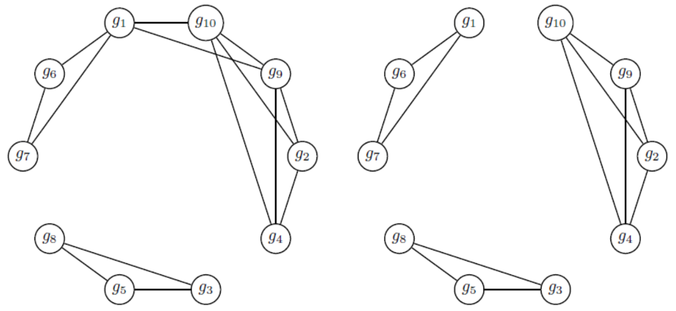


Figure. The similarity graph (left) can be transformed into a clique graph (right) by removing edges (1, 10) and (1, 9). **Phillip, No need to label nodes as g1, g10, 1-10 is sufficient**

It turns out that the Corrupted Cliques problem is difficult, so some heuristics have been proposed to approximately solve it. **CAST (Cluster Affinity Search Technique)** algorithm described below performs remarkably well clustering gene expression data.

Define the similarity between gene *i* and cluster *C* as the average similarity between *i* and all genes in the cluster *C*:

$$R_{i,C} = \sum_{\text{all elements } j \text{ in cluster } C} R_{i,j} / |C|$$

Given a threshold α, a gene *i* is close to cluster *C* if $R_{i,C} > \alpha$ and distant otherwise. A cluster is called **consistent** if all genes in *C* are close to *C* and all genes not in *C* are distant to *C*. The CAST algorithm uses the similarity graph *G* and the threshold α to iteratively find consistent clusters. After a consistent cluster is found, all nodes corresponding to elements of the cluster *C* are removed from the similarity graph and CAST iterates over a smaller graph.

```
CAST(G,α)
Clusters ← empty set
while the similarity graph G is non-empty
  C ← a single-node cluster consisting of a node of maximal degree in the similarity graph G
  while there exists a close gene i not in C or a distant gene i in C
    find the nearest close gene i not in C and add it to C
    find the farthest distant gene i in C and remove it from C
  add cluster C to the set Clusters
  remove nodes of cluster C from the similarity graph G
return Clusters
```

Exercise Break. Implement CAST algorithm.

Exercise Break. Cluster yeast gene expression data using CAST algorithm.

Charging Stations

Why Does the Lloyd Algorithm Converge?

Given a cluster *C* of data points and a center *x,* we define the distance between the cluster and the center as

$$d(C,x) = \sum_{\text{all points DataPoint in cluster } C} d(DataPoint,x)^2$$

Given clustering of *n* data point into clusters $\mathcal{C}=\{C_1, \ldots ,C_k\}$ and the set of centers $X = \{x_1, \ldots x_k\}$, we define the squared error distortion between the clusters and the centers as:

$$d(\mathcal{C},X)= \sum_{1\leq i\leq k} d(C_i,x_i)/m$$

STOP and Think. Given *k* centers, how would you partition the set of data points into *k* clusters to minimize the squared error distortion between the clusters and the centers?

STOP and Think. Given *k* clusters, how would you select *k* centers (one for each cluster) to minimize the squared error distortion between the clusters and the centers?

The answers to the previous two questions imply that the squared error distortion reduces with every iteration of the Lloyd algorithm. But can the Lloyd algorithm continue for an infinite number of iterations with smaller and smaller reductions in the squared error distortion at each iteration?

Exercise Break. Prove that the number of iterations of the Lloyd algorithm does not exceed the number of all possible partitions of the data set into *k* clusters. How would you estimate the number of such partitions?