DRAFT – WILL UNDERGO MANY FURTHER REVISIONS – PLEASE DO NOT DISTRIBUTE!

How Did Yeast Become a Wine Maker?

Clustering Algorithms

Table of Contents

Finding Synteny Blocks in Highly Duplicated Genomes
How Biologists Measure Gene Expression?
Microarrays
Vector Norm
Center of Gravity Theorem
Visualization of Gene Expression Matrices
Clustering and Corrupted Cliques
Why Does the Lloyd Algorithm Converge?

Code Challenges
9A: Implement **FarthestFirstTraversal**
9B: Solve Squared Error Distortion.
9C: Implement the Lloyd algorithm (LA)
9D : Implement the soft k-Means clustering algorithm
9E: Implement **HierarchicalClustering**

An Evolutionary History of Wine Making

How long have we been addicted to alcohol?

One of the oldest organisms to be domesticated by humans was yeast. In 2007, while excavating an old graveyard in an Armenian cave, scientists discovered 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 understanding how to control ***Saccharomyces***, the genus of yeast used in alcohol and bread production.

Although this discovery offered the earliest evidence of wine-making, there is little doubt that our interest in alcohol is much older. In 2008, scientists discovered that pen-tailed tree shrews, which 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* yeast. This finding suggests that our own taste for alcohol may have a genetic basis predating the Armenian wine cave by 55 million years!

Figure 1: 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 also possible that our not-so-distant ancestors were also chronic drinkers (e.g., chimpanzees like drinking naturally brewed nectar from fruits) and that we may have inherited an ancestral association of alcohol intake with caloric gain.

The diauxic shift

The species of yeast we will consider in this chapter is *Saccharomyces cerevisiae*. This organism can brew wine because it converts the **glucose** found in fruit into **ethanol**. Because S. cerevisiae often lives on grapevines, why is simply crushing grapes not sufficient for wine-making? Why must wine makers store 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 changing conditions, it inverts its metabolism, with the ethanol that it just produced becoming 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. Without 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 postulated the **Whole Genome Duplication (WGD) Model** postulating . rare evolutionary events that duplicate an entire genome. A WGD doubles the length of the genome and the number of its genes.

Ohno proposed the WGDM in 1970, when there was absolutely no evidence to support it. However, he argued that a WGD would be necessary at a time of critical evolutionary innovation when a species would need to implement 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 creating new metabolic pathways with many genes working together.

Ohno argued that a WGD would provide a platform for such revolutionary innovation. Indeed, every duplicated gene would have two copies, one serving its previous function and facilitating survival, and the other free to evolve into a new function.

The Random Breakage Model and the Whole Genome Duplication Model had very different fates. From its proposal, the RBM was embraced by biologists, and it became widely accepted dogma until its refutation in 2003. In contrast, the WGD Model was initially met with skepticism because only 13% of *S. cerevisia*e genes are duplicated. 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 a WGD in *S. cerevisia*e. They argued that the fact that only 13% of *S. cerevisia*e genes are duplicated is not surprising since even if hundreds of genes participate in an evolutionary innovation after a WGD, most genes are not needed for this innovation. Thus, unneeded duplicate genes will be bombarded by mutations until they are turned into pseudogenes and eventually disappear from the genome after millions of years.

STOP and Think: How would you argue that *S. cerevisiae* has undergone a WGD as opposed to a series of single-gene duplications?

In 2004, Manolis Kellis compared the genome of *S. cerevisiae* to that of a newly sequenced yeast species 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?

One of the many steps yeast performs during fermentation is a conversion of acetaldehyde into ethanol; 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.

In 2005, Michael 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 a WGD, Adh_A and its duplicate (as well as some other duplicated genes) formed the basis for the "make-accumulate-consume" strategy of ethanol production. This strategy equipped the ancestor of *Saccharomyces* yeast with an advantage over its competitors; *Saccharomyces* kills its competitors by producing ethanol, which is toxic to most bacteria and yeasts, and it can then consume this ethanol.

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 a period of m hours (before and after the diauxic shift), resulting in an $n \times m$ **gene expression matrix** E, where $E_{i,j}$ is the expression level of gene i at moment j. Just by looking at this matrix, you will be able to see genes having various behavior with respect to the diauxic shift, , 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?

Although this chapter focuses on gene expression across various time points to study the diauxic shift, biologists study gene expression in a myriad of applications, from analyzing tissues before and after taking drug, to studying cancerous versus non-cancerous cells. For example, expression analysis led to **MammaPrint**, a diagnostic test that determines the likelihood of breast cancer recurrence and is, based on expression analysis of 70 human genes associated with tumor activation and suppression. But how did biologists identify these genes?

Introduction to Clustering

Gene Expression Analysis

In 1997, Joseph DeRisi conducted the first massive gene expression experiment by sampling an *S. cerevisiae* culture seven times at hours -6, -4, -2, 0, +2, +4, and +6, where 0 refers to the timing of the diauxic shift. Since there are about 6400 genes in *S. cerevisiae*, this resulted in a 6400 x 7 gene expression matrix.

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

We have already discussed three technologies that could be used to generate this matrix (see **DETOUR:** How Do Biologists Measure Gene Expression?), but none of these technologies had matured by 1997! For this reason, DeRisi invented **microarrays**, which differ from the DNA arrays that we discussed in Chapter 3 (see **DETOUR:** Microarrays).

Since biologists today have many options for how to generate data for expression analysis, we will not discuss the specifics of each technology but rather assume that we are already given the $n \times m$ gene expression matrix E, where n is the number of genes and m is the number of checkpoints. The i-th row of E 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 expression vector is normalized by dividing all elements in row i by the **vector norm** of this row, which is equal to $\sqrt{(\sum_{j=1,m} E_{i,j}^2)}$.

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 either perform similar functions or are involved in the same biological process. We do not exclude the possibility that the expression vectors of two unrelated genes are similar simply by chance, but this possibility becomes less likely as we increase the number of checkpoints (m). 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.

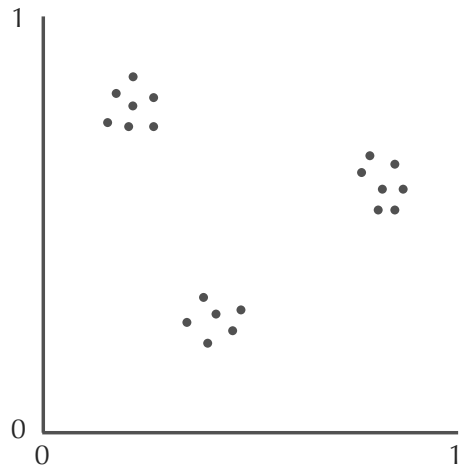
Another important application of expression analysis is in deciphering regulatory pathways; similar expression vectors may imply co-regulation, i.e., two genes are co-regulated if their expression is controlled by the same transcription factor. This suggests a "guilt by association" strategy for genome annotation by starting from a few genes with known functions and potentially propagating the known functions of these genes to other genes in the same cluster.

Homogeneity and separation

We would like to **partition** the set of all yeast genes into clusters so that each gene belongs to a single cluster. Clusters should satisfy two conditions, which are illustrated in Figure 2 for a collection of points in 2-D:

- **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



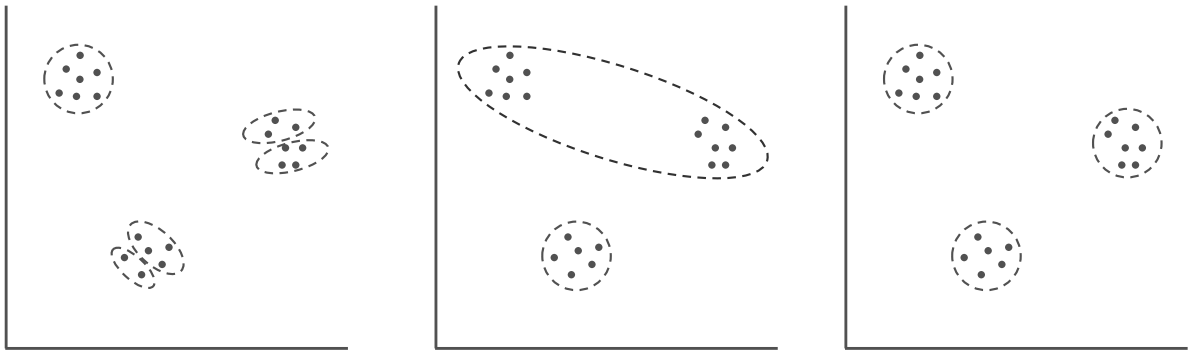


Figure 2: (Top) A 20 x 2 matrix along with a representation of each row as a point in 2-dimensional space. (Bottom) The clusters on the left exhibit homogeneity but not separation; the clusters in the middle exhibit separation but not homogeneity; the clusters on the right exhibit both homogeneity and separation.

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

Input: An *n x m* gene expression matrix *E*.

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

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

Clustering as an optimization problem

Motivated by Figure 2, we will view the n rows of the $n \times m$ gene expression matrix as a set *Data* of n points in m-dimensional space. In order to divide *Data* into k clusters, we will first introduce the problem of finding k points that will serve as the "centers" of the k clusters in *Data*. The value of k is not known *a priori*, 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

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

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:

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

We define the maximal distance between all data points *Data* and centers X, $MaxDistance(Data, X)$, as the maximum of $d(DataPoint, X)$ among all data points:

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

We can now state a well-defined computational problem: choosing centers X in order to minimize $MaxDistance(Data, X)$.

***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* of *k* centers that minimizes *MaxDistance(DataPoints, X)* over all possible choices of *X*.

Once we have chosen a set of centers, the points in *Data* can be divided into *k* clusters by assigning each data point to its nearest center.

Farthest First Traversal

Although the *k*-Center Clustering Problem looks simple, it is NP-hard. Pseudocode for the **Farthest First Traversal** heuristic for the *k*-Center Clustering Problem is shown below. This approach begins by selecting an arbitrary point in *Data* as the first center; it then iteratively adds a new center as the point in *Data* farthest from 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
```

Exercise Break: Apply **FarthestFirstTraversal** to the data in Figure 2.

Code Challenge: Implement **FarthestFirstTraversal**. 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 approximates the optimal solution of the *k*-Center Clustering Problem, it is not often used for gene expression analysis. 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 devise 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

To model the *typical* derivation of each data point from its nearest center, we will introduce a new scoring function. Given a set of n data points *Data* and a set of k centers X, their **squared error distortion** is defined as the mean squared distance from each data point to its nearest center (Figure 3):

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

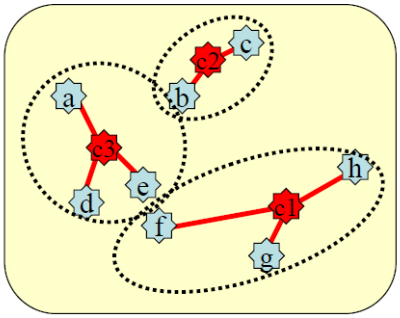


Figure 3: The squared error distortion is computed as the sum of the squares of the lengths of red segments divided by 8.

Exercise Break: Compute *Distortion*(*Data*, *X*) for the points *Data* in Figure 2 and the three centers $X = \{(0.20, 0.81), (0.39, 0.29), (0.80, 0.62)\}$. How would you change these centers to reduce the squared error distortion?

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.

Whereas before we attempted to minimize *MaxDistance*(*Data*, *X*), we can now find centers by minimizing *Distortion*(*Data*, *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.

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, it remains unclear how to find the center of these points minimizing the squared error distortion.

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*:

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

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*. It turns out this is the point minimizing $Distortion(Data, x)$ among all possible centers x.

Center of Gravity Theorem. The center of gravity of a set of points *Data* solves the k-Means Clustering Problem for $k = 1$.

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

Although the k-Means Clustering Problem turned out to be easy for $k = 1$, it is *NP*-hard even for $k = 2$ when clustering in a multi-dimensional space. However, in the case that we are clustering in a one-dimensional space (i.e., when all data points fall on a line), the 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 as centers. The Lloyd algorithm then iteratively performs the following two steps (Figure 4):

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

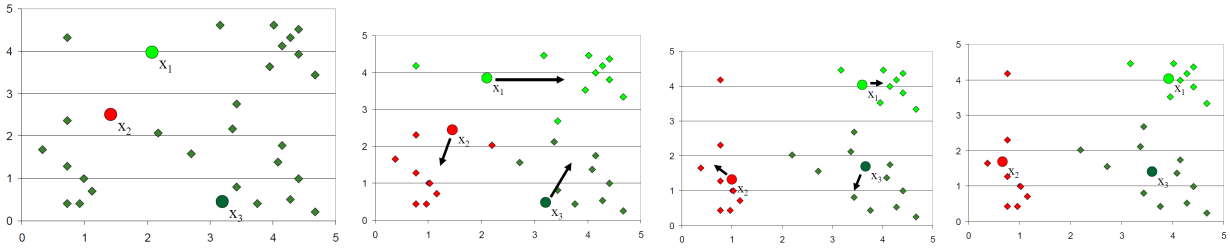


Figure 4: The Lloyd algorithm in action.

Code Challenge: Implement the Lloyd algorithm 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 the centers (and therefore their clusters) stop changing between iterations.

STOP and Think: 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 **DETOUR:** Why Does the Lloyd Algorithm Converge? The Lloyd algorithm often converges to a local minimum of the squared error distortion function rather than to a global minimum.

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

Initializing the Lloyd algorithm

Since the Lloyd algorithm may converge to a local minimum, our goal is to initialize it in such a way that it has a better chance of converging to a global minimum. Figure 5 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 5 (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.

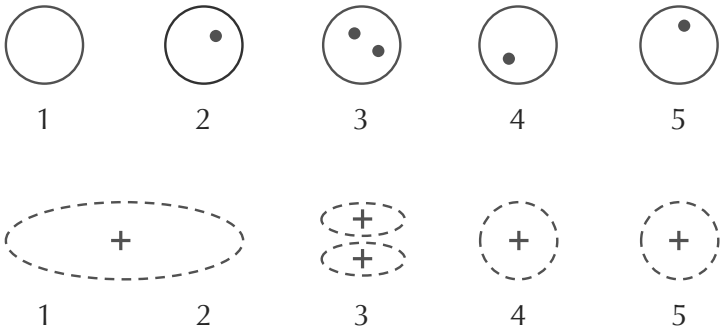


Figure 5: (Top) Five circles in two-dimensional space with each circle containing 100 points (not shown). The Lloyd algorithm was initialized so that circle 1 has no centers, circle 3 has two centers, and all other circles have one center, shown as black dots. (Bottom) The Lloyd algorithm erroneously combines the points in circles 1 and 2 into a single cluster and splits circle 3 into two clusters. Updated centers are shown as crosses.

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 nearby points. 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
    randomly select DataPoint from Data 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

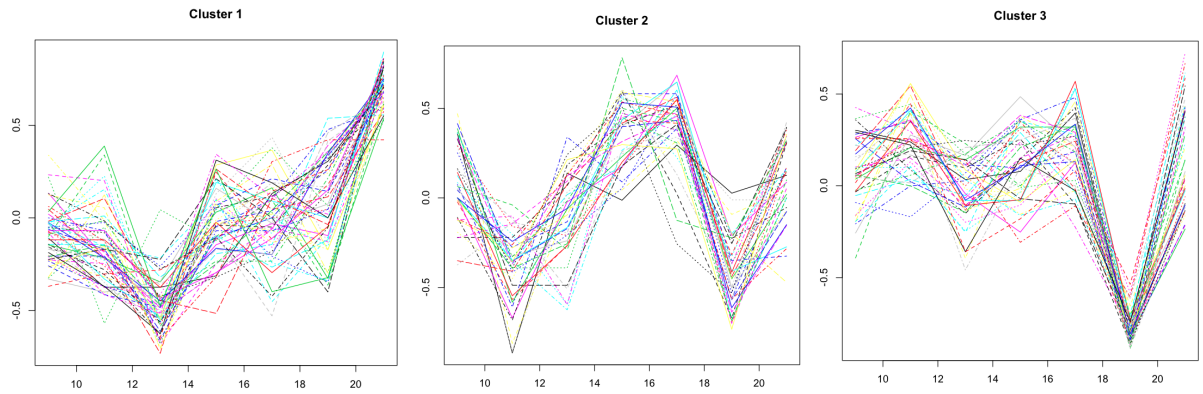
Although the diauxic shift is an important event in the yeast's life, it has no bearing on most of the yeast's functions. We thus expect that most *S. cerevisiae* genes are not involved in the diauxic shift, and that their expression will hardly changes during the experiment. We will therefore exclude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

Given the expression vector (E_1, …, 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 therefore 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 (i.e, to the selected 264 genes from the yeast dataset) and partition this dataset into six clusters.

Figure 6 visualizes expression vectors for each cluster obtained after applying the Lloyd algorithm to the 264 selected genes from the yeast dataset. See **DETOUR:** Visualization of Gene Expression Matrices to learn more about how biologists visualize gene expression data.



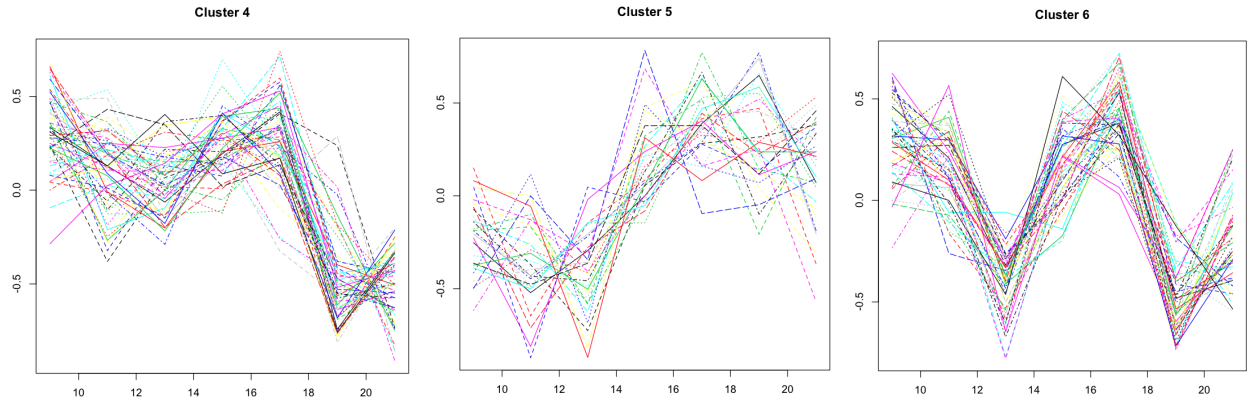


Figure 6: Applying the Lloyd algorithm (with six centers) to the yeast dataset of 264 genes results in clusters containing (from top left to bottom right) 54, 36, 42, 54, 28, and 50 genes. Expression vectors for all genes in each of these six clusters are visualized as six separate plots. Each expression vector (E_1, …, E_m) is represented as a line graph connecting point (i, E_i) to ($i + 1$, E_{i+1}) for each i between 1 and $m - 1$.

To reveal the patterns of the expression vectors in each cluster, we average them (Figure 7).

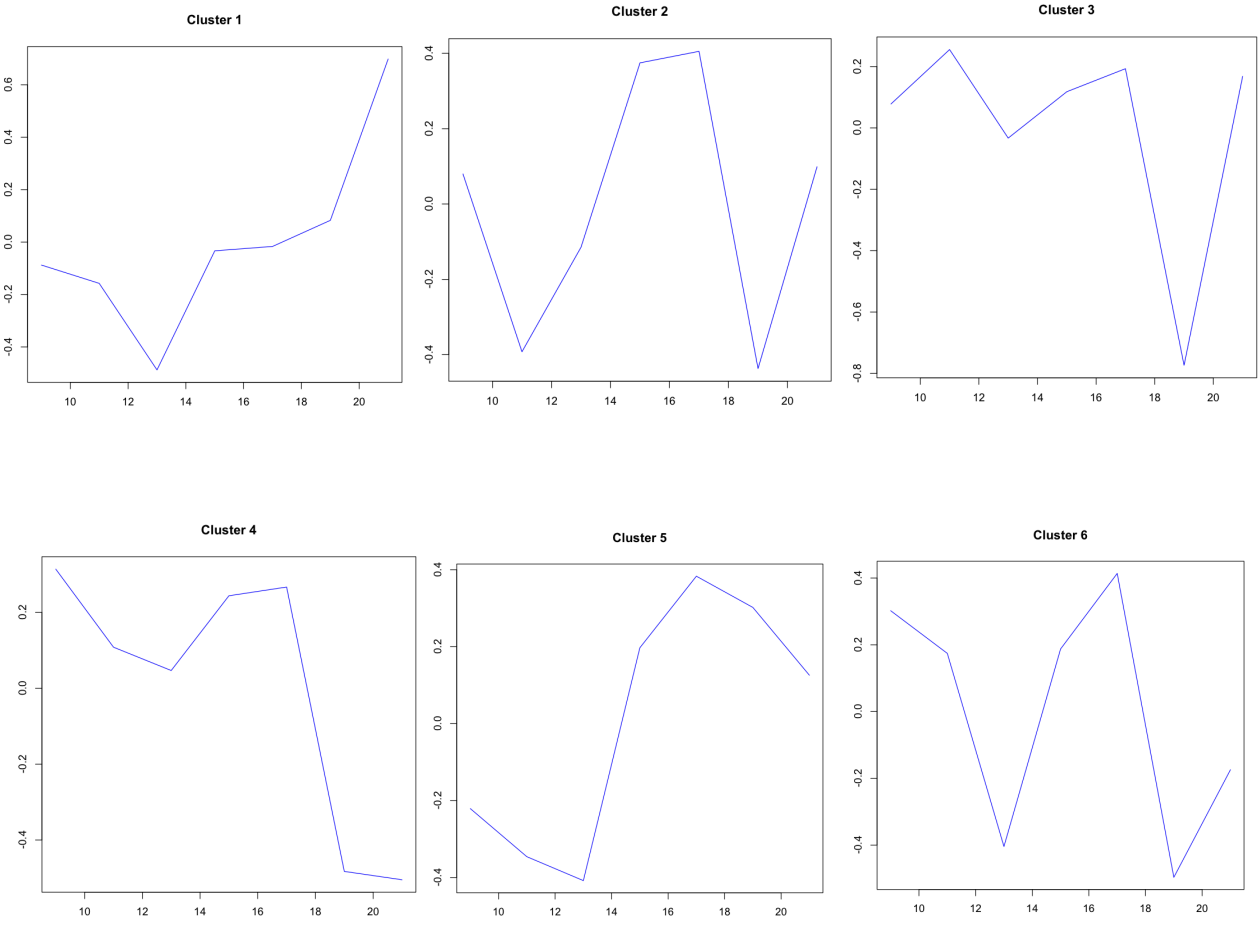


Figure 7: Averaging the expression vectors in each plot from Figure 6 reveals six types of regulatory behavior.

The six plots in Figure 7 reveal six different regulation patterns of various genes involved in the diauxic shift. These patterns 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?

Exercise Break: Apply the Lloyd algorithm to the entire yeast dataset (i.e, to all 6400 genes from the yeast dataset) and partition this dataset into six clusters. This clustering of the entire yeast dataset imposes clustering of the selected 264 genes from the yeast dataset. How does this clusteting differ from clustering shown in Figure 6?

Limitations of *k*-means Clustering

Before we continue our discussion of finding genes associated with the diauxic shift, consider: how would you cluster the data in Figure 8?

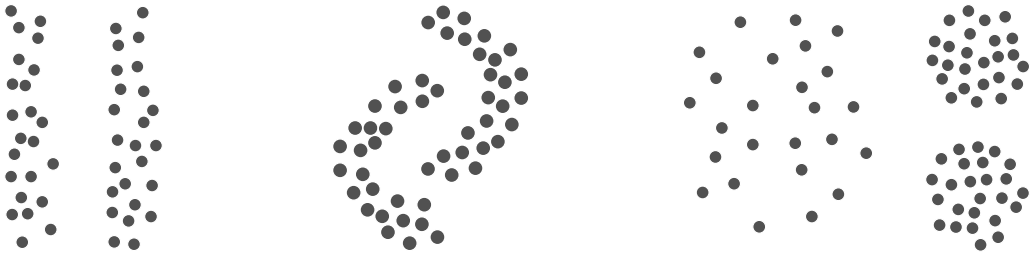


Figure 8: Difficult clustering problems for $k = 2$ (left and middle) and $k = 3$ (right).

It turns out that, in the case of challenging clustering problems, the human eye and the Lloyd algorithm will disagree (Figure 9).

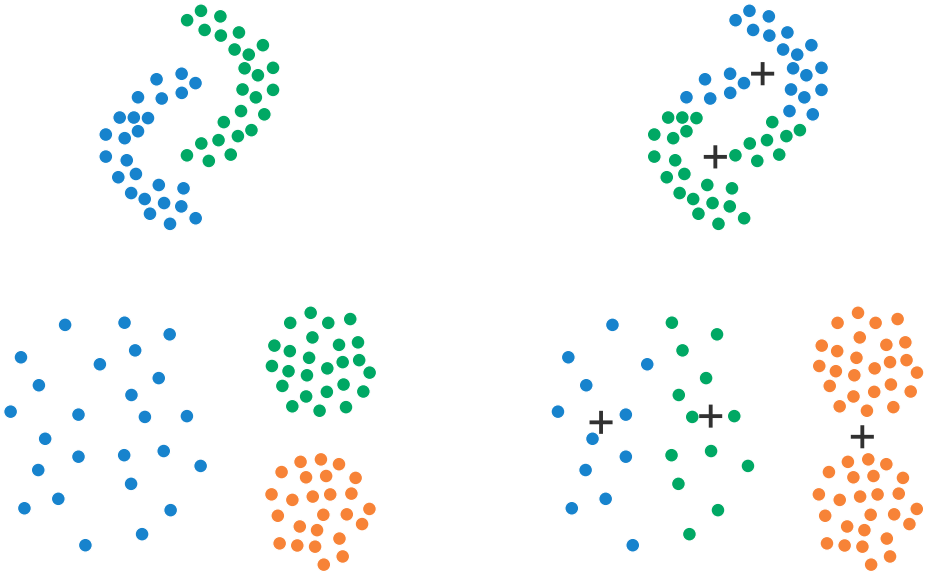


Figure 9: The human eye (left) and the Lloyd algorithm (right) often provide different solutions in the case of elongated clusters (top), clusters with non-globular shapes (middle), and clusters with widely different densities (bottom). Cluster centers of gravity on the right are shown by crosses.

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?

We call a data point a **midpoint** if it is approximately equidistant from two centers (Figure 10). The Lloyd algorithm makes a "hard" assignment of a midpoint to a specific cluster, which does not make sense, since there is just as good of a reason to assign it to another cluster. As a result, the Lloyd algorithm may perform poorly. A better solution would be to modify the Lloyd algorithm so that midpoints are less important in deciding on the positions of the centers of their corresponding clusters.

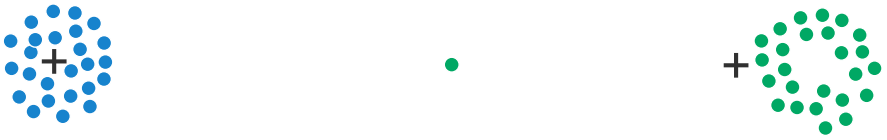


Figure 10: A point approximately halfway between two clusters skews the center of gravity of the green cluster to which it belongs (centers of clusters are shown by crosses). A modification to the Lloyd algorithm would allow this point equal influence over both clusters.

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 may seem like a completely unrelated detour.

Expectation Maximization

Flipping biased coins

To address some limitations of the Lloyd algorithm, our goal is to allow data points to belong to more than one cluster by 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 the value of θ that maximizes this probability.

STOP and Think: Given a 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 $f(\theta)$, 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 is equal to zero:

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

The solution is given by $\theta = i/n$ (as you have may have expected), implying that the observed fraction of heads provides the best estimate for θ, i.e.,

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

A crooked dealer with two coins

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 your night at the casino, you observe the dealer selecting one of these coins (but you don't know which one) and flipping 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
```

Now, you know the number of heads in each of the five sequences, 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 to generate each sequence.

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

If you were told which 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 11):

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

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

	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 11: 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 are 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, as shown in Figure 12, if $\theta_A^{xi}(1-\theta_A)^{n-xi} > \theta_B^{xi}(1-\theta_B)^{n-xi}$, we guess that experiment *i* used coin *A*; otherwise, it used coin *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 12: Computing *HiddenVector* from *Data* and *Parameters*. For example, used coin A (the first element of *HiddenVector* is 0) because $0.45^5(1-0.45)^5 > 0.80^5(1-0.80)^5$.

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

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

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:

$$(Data, ?, ?) \rightarrow \textbf{???}$$

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:

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

Finally, we repeat these two steps and watch how *Parameters* and *HiddenVector* are getting closer and closer to – 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 (i.e., that the hidden vector and parameters eventually stop changing).

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

For example, Figure 13 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

Figure 13: Starting with *Parameters* = (0.65, 0.70) results in *HiddenVector* = (0, 1, 1, 0, 1), which forces us to update *Parameters* to *Parameters'* = (0.45, 0.8).

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 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 $Data = (Data_1, \ldots, Data_n)$, we will represent their assignment to k clusters as an n-dimensional vector $HiddenVector = (HiddenVector_1, \ldots, HiddenVector_n)$, where each $HiddenVector_i$ can take integer values from 1 to k. We will then represent the k cluster centers as $Parameters=(\theta_1, \ldots, \theta_k)$. 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:

- **Centers to Clusters**: $(Data, ?, Parameters) \rightarrow HiddenVector$
- **Clusters to Centers**: $(Data, HiddenVector, ?) \rightarrow Parameters$

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 $\theta_A^{xi} (1-\theta_A)^{n- xi} = \theta_B^{xi} (1-\theta_B)^{n- xi}$? Is it fair to always select A if $\theta_A^{xi} (1-\theta_A)^{n- xi}$ is only slightly larger than $\theta_B^{xi} (1-\theta_B)^{n- xi}$?
2. If a data point is equally distant from two centers, what cluster should 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

Given parameters $Parameters = (\theta_A, \theta_B)$, 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 $Parameters = (\theta_A, \theta_B) = (0.6, 0.5)$, we will select B for the coin flipping sequence "HHHHTHHHHH" because $0.6^9 * 0.4^1 = 0.0040310784$ is larger than $0.5^9 * 0.5^1 = 0.0009765625$. But how can we make soft decisions about which coin generated the sequence, such as assigning A has "responsibility" p and B "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 compute responsibilities 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 **responsibility profile** *HiddenMatrix* that can be constructed from *Data* and *Parameters*:

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

Figure 14 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*:

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

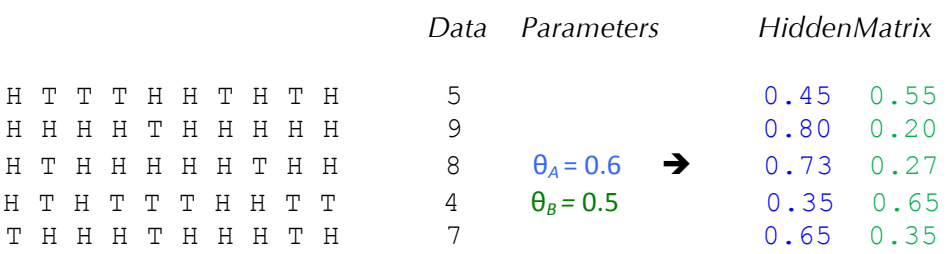


Figure 14: Computing *HiddenMatrix* from *Data* and *Parameters*. Compare with Figure XXX for the same choice of *Data* and *Parameters*.

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:*

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

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

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

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:

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

Similarly,

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

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

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

$$\theta_B = (1*5+0*9+0*8+1*4+0*7) / \ (1*10+0*10+0*10+1*10+0*10) = 9/20 = 0.45$$

Hard assignments correspond to the following responsibility profile, where the first row corresponds to *A* and the second row corresponds to *B*:

$$HiddenMatrix = \begin{matrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{matrix}$$

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

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

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

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

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

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

In the case of soft choices and the following 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*:

$\theta_A = (0.45*5+0.80*9+0.73*8+0.35*4+0.65*7) / \ (0.45*10+0.80*10+0.73*10+0.35*10+0.65*10) = 21.3/29.8 \approx 0.71$
$\theta_B = (0.55*5+0.20*9+0.27*8+0.65*4+0.35*7) / \ (0.55*10+0.20*10+0.27*10+0.65*10+0.35*10) = 11.7/20.1 \approx 0.58$

Soft *k*-Means Clustering

We are now ready to modify the Lloyd algorithm into the **soft *k*-means clustering algorithm** inspired by the expectation maximization algorithm from the preceding section. Given *n* points

Data = {$Data_1$, . . . ,$Data_n$} and k centers X = {x_1, . . . x_k}, this algorithm first constructs an n x k responsibility matrix *HiddenMatrix*. The closer data point i is to center x_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 $Data_i$ is a planet and that center x_j is a star, and it computes responsibility according to the Newtonian inverse-square law of gravitation.

Unfortunately for Newton fans, however, it turns out that the following formula from 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 15 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., β →∞? How would soft k-means clustering perform when β = 0 and as β → -∞?

4/5	1/5	0.453	0.547	0.408	0.592
9/10	1/10	0.291	0.709	0.144	0.856
1/2	1/2	0.500	0.500	0.500	0.500
1/10	9/10	0.709	0.291	0.856	0.144
1/5	4/5	0.547	0.453	0.592	0.408

Figure 15: *HiddenMatrix* constructed for the five (one-dimensional) points *Data* = {-3, -2, 0, +2, +3} and the two centers X = {-1, +1} for $HiddenMatrix_{i,j} = [1/d(Data_i,x_j)^2]/[\sum_{all\ centers\ j} 1/d(Data_i,x_j)^2]$ (left), $HiddenMatrix_{i,j} = e^{-\beta \cdot d(Datai,xj)}/\sum_{all\ centers\ j} e^{-\beta \cdot d(Datai,xj)}$ for β = 1 (middle) and β = 2 (right).

Figure 16 illustrates how *HiddenMatrix* changes with respect to the stiffness parameter.

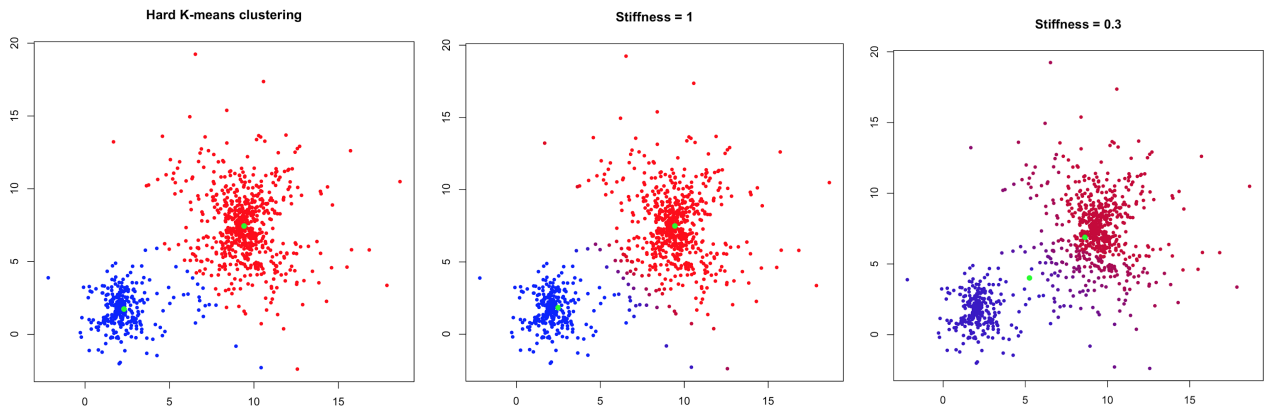


Figure 16: The results of the Lloyd algorithm Hard *k*-means clustering (left) and soft *k*-means clustering algorithms for β = 1 (middle) and β = 03 (right). Responsibility profiles for "red" and "blue" centers and two choices of the stiffness parameter, β=1 (middle), and β = 0.3 (right). The colors 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).

In the Lloyd algorithm, we updated the center of a cluster as the center of gravity of all points in the cluster. In soft *k*-means clustering, 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*, meaning that center x_j should be updated as the **weighted center of gravity** of all data points:

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

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 algorithm.

Exercise Break: Apply soft *k*-means clustering to the yeast diauxic shift gene expression data and compare the results with those 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. 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

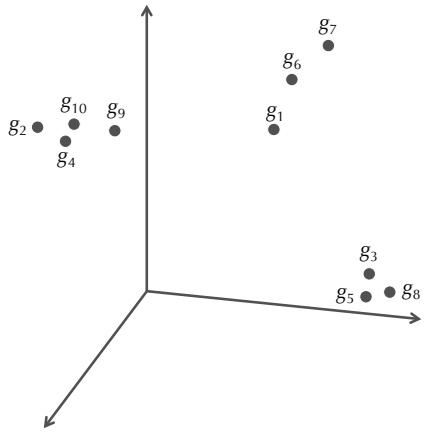


Figure 17: A toy gene expression matrix of ten genes measured at three time points (left), the distance matrix based on Euclidean distance (right), and the gene expression vectors as points in three-dimensional space (bottom).

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 18 shows a tree representing the data in Figure 17 and describes a family of ten different ways of dividing the data into clusters. Specifically, as shown in Figure 18, a horizontal line through the crossing the tree at i points corresponds to a division of the ten genes into i clusters.

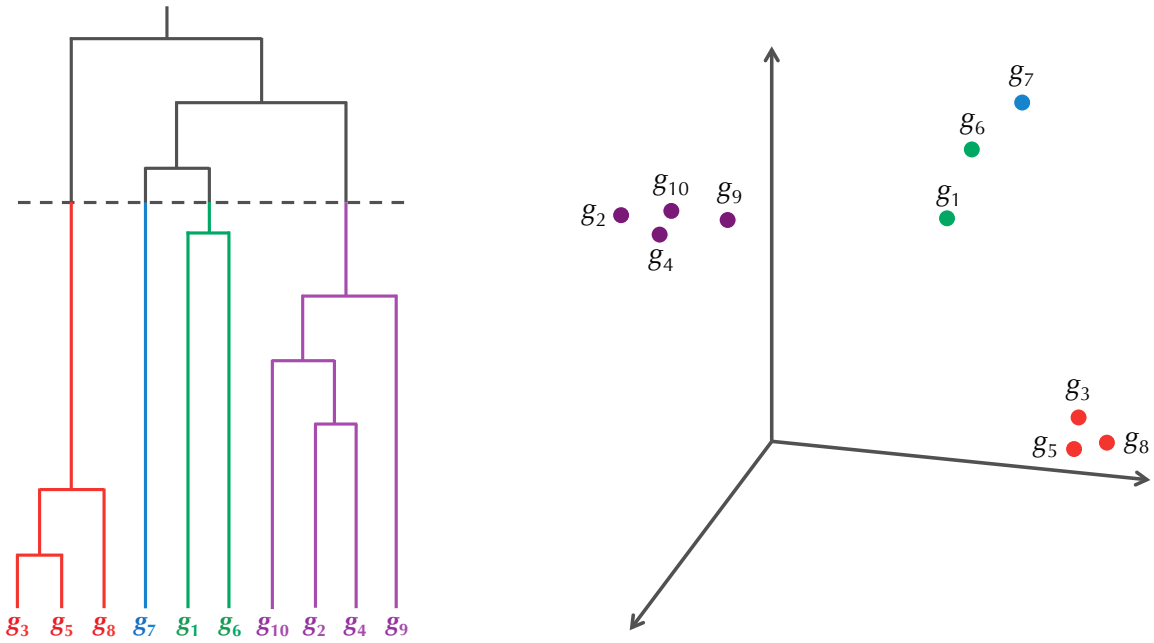


Figure 18: (Left) A hierarchical clustering of the data in Figure above. The horizontal line crosses the data in four places and divides the data into four colored clusters. (Right) The colored points shown in three-dimensional space.

The **HierarchicalClustering** algorithm, whose pseudocode is shown below, starts from an $n \times n$ distance matrix D derived from the gene expression matrix. (e.g., by computing Euclidean 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 C1 and C2
    merge C1 and C2 into a new cluster C with |C1| + |C2| elements
    add a new node to T, connect it to nodes C1 and C2 by directed edges, and label it by cluster C
    remove rows and columns of D corresponding to C1 and C2
    remove C1 and C2 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
```

This pseudocode does not specify how exactly to compute the distances from a newly formed cluster to old clusters, 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,

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

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

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

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 19 visualizes expression vectors for each of the six clusters obtained after applying **HierarchicalClustering** to the yeast dataset; their averages are shown in Figure 20.

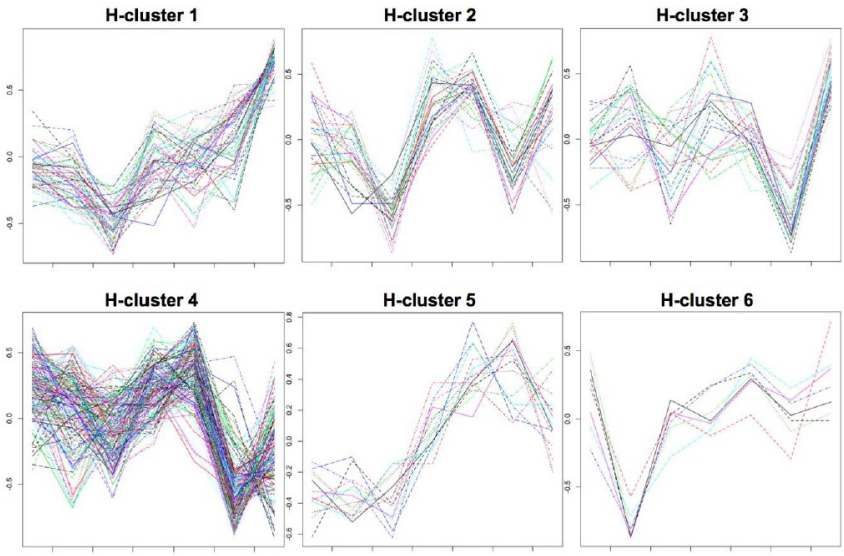


Figure 19: Applying **HierarchicalClustering** to the yeast dataset results in six clusters with (from top left to bottom right) 48, 27, 23, 145, 14, and 7 genes. Expression vectors for all genes in each of these six clusters are visualized as six separate plots.

As you can see, the clusters obtained by the **HierarchicalClustering** and the Lloyd algorithm (Figure 6) are rather different. You may be concerned that different clustering approaches have produced different clusters. The truth is that there are many different ways of formulating clustering as a computational problem (see **DETOUR:** Clustering and Corrupted Cliques for yet another approach). Furthermore, individual gene expression studies often represent the beginning of a larger experimental work to confirm that derived clusters make sense biologically. Once clusters have been generated, research often focuses on specific genes within these clusters.

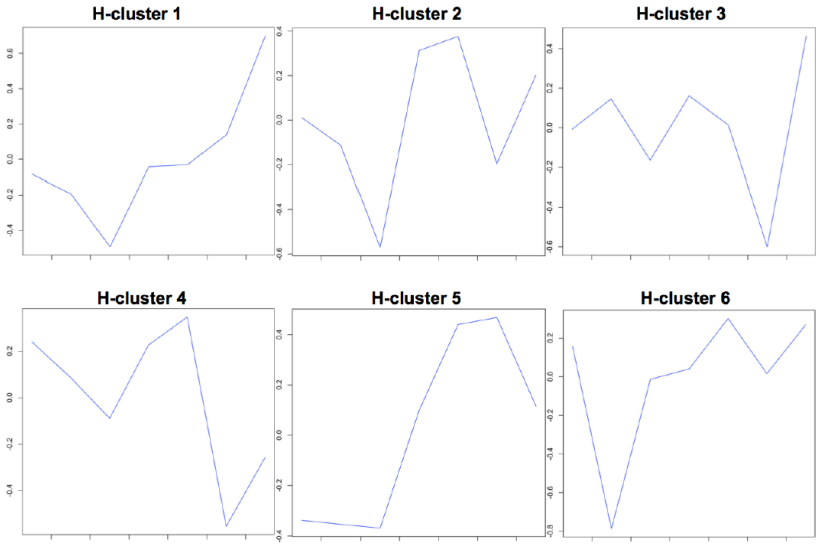


Figure 20: Averaging the expression vectors in each cluster reveals six types of regulatory patterns of various genes involved in the diauxic shift.

Epilogue: Clustering Tumor Samples

As we mentioned in the introduction, gene expression analysis has a wide variety of applications, including cancer studies. In 1999, Uri Alon analyzed gene expression data for 2000 genes from various colon tumor tissues and compared them with colon tissues from healthy individuals. We represent his data as 2000 x 60 gene expression matrix, where the first 40 columns describe tumor samples and the last 20 columns describe normal samples.

Now, suppose you performed a gene expression experiment with a colon sample from a new patient, representing a 61^{st} row 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, it makes sense to construct a 60 x 60 similarity matrix between tissues and cluster the 60 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 Alon's 2000 x 60 gene expression matrix and gene data from a new patient, determine whether this patient is likely to have a colon tumor.

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

Cristianini N., Hahn M.W. Introduction to Computational Genomics. Cambridge University Press, 2007

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

Bibliography Notes

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 that we borrowed in our discussion of coin flipping. The Lloyd algorithm for *k*-means clustering was introduced by Stuart Lloyd in 1957. The soft *k*-means clustering algorithm 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 (see Cristianini and Hahn, 2007 for an excellent analysis of this experiment) Eisen, Spellman, Brown and Botstein, 1998 described the 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 proposed 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 ancient alcohol dehydrogenases from yeast in 2005.

DETOURS

Finding Synteny Blocks in Highly Duplicated Genomes

WGDs are quickly followed by massive gene loss and rearrangements, making it difficult to reconstruct the architecture of the pre-duplicated genome. Indeed, although all genes in ancestral *S. cerevisiae* had duplicates immediately after its WGD approximately 100 million years ago (Figure 21(a)), many of these duplicates were lost (Figure 21(b)), leaving only 13% of genes with duplicates in modern-day *S. cerevisiae* (parts (c-d) in Figure below). How can we conclude, then, 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 synteny block of *K. waltii* corresponds to two regions of *S. cerevisiae*, as expected for WGD (part (e) in Figure below).

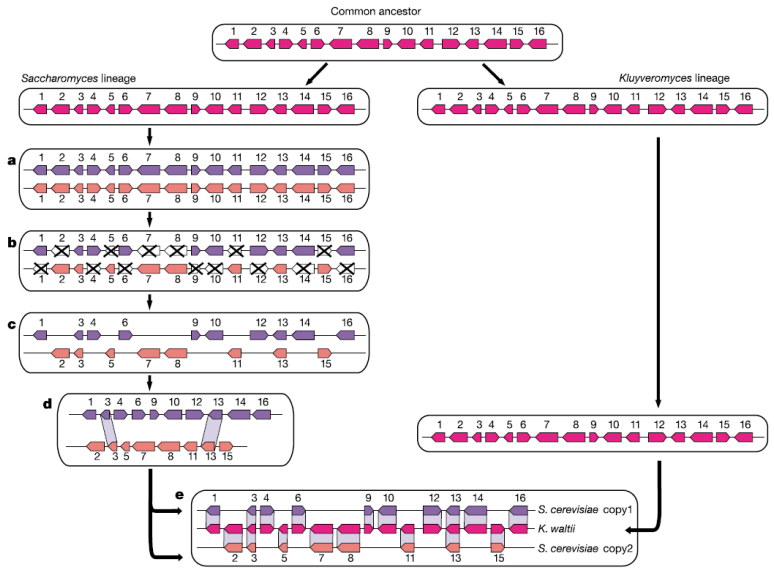


Figure 21: *K. waltii* helps find paired synteny blocks in *S. cerevisiae*. a) After divergence from *K. waltii*, *S. cerevisiae* underwent a WGD, creating two identical copies of every chromosome. b) The vast majority of duplicated genes underwent mutations, resulting in gene losses. c) Sister segments 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 conclude that the synteny blocks shown (formed by genes formed by genes (1,3 4, 6, 9, 10, 12, 13, 14, 16) and (2, 3, 5, 7, 8, 11, 13, 15) 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 because most synteny blocks in *K. waltii* have two corresponding synteny blocks in *S. cerevisiae*.

How Do Biologists Measure Gene Expression?

In a proteomics experiment (chapter X), a bioinformatician generates a set of spectra and matches them against a proteome. The number of spectra matching peptides from a given protein offers a proxy for the expression level of this protein. To estimate the protein expression from this proxy, bioinformaticians must 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 the **transcriptome** (all RNA transcripts present in the cell) rather than genome using an approach known as **RNA-Seq**. By estimating the quantity of each protein-coding RNA transcript in a sample, we obtain a proxy for the expression of the resulting protein. However, although 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 protein production in the cell in addition to transcription, such as translation, post-translational modifications, and protein degradation. These additional factors can muddle the correlation between the quantity of a transcript and protein expression.

Finally, one can use customized DNA arrays (Chapter 3) carrying probes aimed at each gene in a species of interest. Each probe is characterized by an intensity, which offers a proxy for the number of transcripts of a given gene present in the sample. A deficiency of DNA arrays is that these 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. This deficiency of DNA arrays makes RNA-seq more attractive in practice.

Microarrays

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

The amount of cDNA printed on each spot of the microarray can vary greatly, a complication that DeRisi had to work around in order to ensure that fluorescence intensities can be compared across spots and across arrays. To achieve this goal, cDNA arrays are used by hybridizing two samples (e.g., samples corresponding to two different timestamps) to each array (Figure 22). The two samples are labeled with different fluorescent dyes (green and red), that can be read by image-processing software.

Expression values obtained from a microarray are represented as the ratio of fluorescent intensities of the two samples. Thus, expression is measured as relative changes in the expression of individual genes between samples and time points.

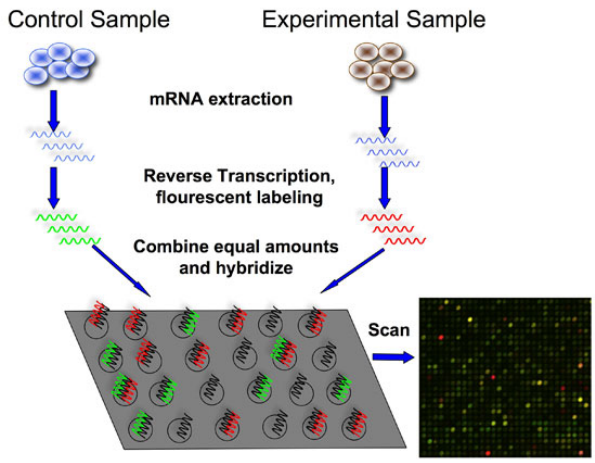


Figure 22: A microarray.

Center of Gravity Proof

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

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

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

The last line of this formula implies that we can independently minimize $d(Data, x)$ in each of m dimensions by minimizing each of the m expressions $\sum_{\text{all points DataPoint in Data}} (DataPoint^j - x^j)^2$. Each of these expressions is a concave up quadratic function of a single variable x^j. Thus, we can find the minimum of this function by finding where the derivative of each such function $(-\sum_{\text{all points DataPoint in Data}} 2*(DataPoint^j - x^j))$ is equal to zero. For each j, this process amounts to solving the equation

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

This equation's solution is given by

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

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 that $k = 1$ is simply the center of gravity of all data points.

□

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 a clustering of n data points into clusters $C = \{C_1, \ldots, C_k\}$ and a set of centers X, we extend the definition of squared error distortion between the clusters and the centers as follows:

$$Distortion(C, X) = \sum_{\text{all centers } x \text{ in } X} Distortion(C_i, x)/n$$

STOP and Think: Consider the following two questions.

1. 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?
2. 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 in every "Clusters to Centers" step and every "Centers to Clusters" step of the Lloyd algorithm.

STOP and Think: Does this mean that the Lloyd algorithm must converge?

It is not true that the Lloyd algorithm must converge just because the squared error distortion decreases at each step; for example, it could be the case that subsequent decreases in squared error distortion could become smaller and smaller, leading to an infinite process. The following exercise shows that this is not the case.

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

Visualization of Gene Expression Matrices

To make sense of a gene expression matrix, biologists often rearrange its genes so that similar expression vectors correspond to consecutive rows in the gene expression matrix. Then, biologists often assign colors to each element of the gene expression matrix according to their expression levels, resulting in a **heatmap**. Figure 23 shows the heatmap of the yeast gene expression matrix.

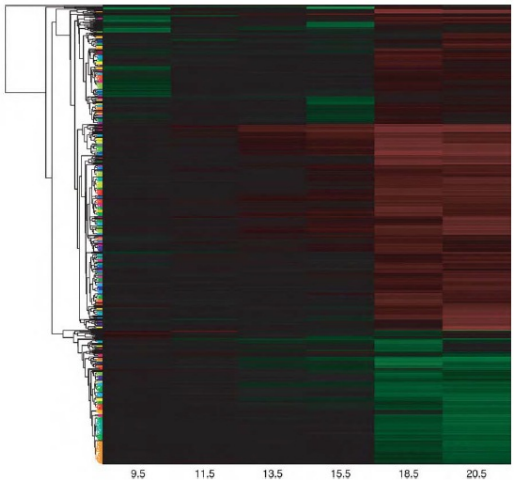


Figure 23: The heatmap of a gene expression matrix.

Clustering and Corrupted Cliques

In expression analysis studies, the similarity matrix R is often transformed into a **similarity graph** $G(\theta)$. The nodes of this graph represent genes, and an edge connects genes i and j if and only if the similarity $R_{i,j}$ between them exceeds a threshold value 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?

Intuitively, if a clustering satisfies the principles of homogeneity and separation, then there should be some value of θ such that each connected component of $G(\theta)$ is a **clique**, in which every two nodes are connected by an edge (Figure 24). In general, a graph whose connected components are all cliques is called a **clique graph**.

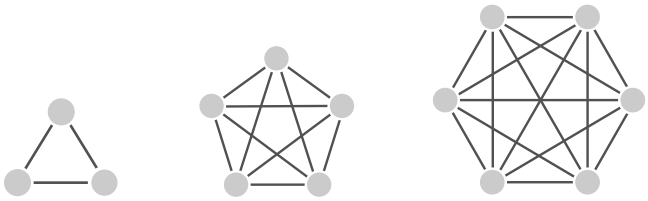


Figure 24:. A clique graph consisting of three cliques.

Errors in expression data and the absence of a universally accepted threshold θ often result in corrupted similarity graphs, whose connected components are not cliques (Figure 25). If the principle of homogeneity is violated, then genes from the same cluster will have a similarity value falling below θ, thus removing edges from a clique. If the principle of separation is violated, then genes from different clusters will have a similarity value exceeding θ, thus adding edges between

different cliques. The question therefore arises of how to transform a corrupted 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 edge additions and removals that transform this graph into a clique graph.

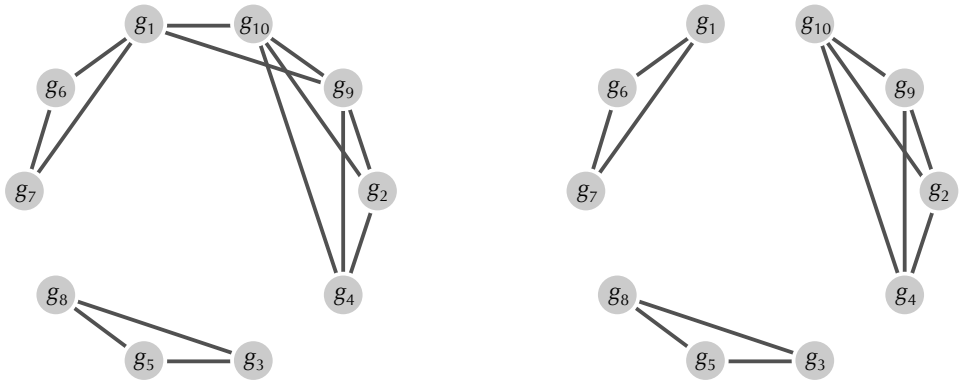


Figure 25: A similarity graph for the genes from Figure 17 (left) can be transformed into a clique graph (right) by removing edges (g_1, g_{10}) and (g_1, g_9).

It turns out that the Corrupted Cliques Problem is difficult to solve exactly, so some heuristics have been proposed to find approximate solutions. The **Cluster Affinity Search Technique (CAST) algorithm** described below performs remarkably well at clustering gene expression data.

We will define the similarity between gene i and cluster C as the average similarity between i and all genes in C:

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

Given a threshold value 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 (homogeneity) and all genes not in C are distant to C (separation). 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 in cluster C are removed from the similarity graph and CAST iterates over the resulting smaller graph.

```
CAST(G,α)
Clusters ← empty set
while G is nonempty
  C ← a single-node cluster consisting of a node of maximal degree in 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 C to the set Clusters
```

remove the nodes of C from G
return *Clusters*

Exercise Break: Implement **CAST**.

Exercise Break: Cluster the *S. cerevisiae* gene expression data using **CAST**.