Introduction

In lecture 7 we discussed clustering. This set of notes is meant as a supplement to the slides from that lecture, so instead of copying slides I will refer to them.

Clustering shouldn’t be confused with classification, where you already have pre-defined labels for some of the data points, and you use that information to devise a rule that assigns labels to new points. Clustering algorithms, on the other hand, don’t use pre-defined labels to group your data. Data is instead grouped by how “near” points are to each other.

From the Jones/Pevzner book: good clusters have two basic properties. We want our clusters to be homogenous, and separated. That is, we want the points within any given cluster to be very similar, and points in different clusters to be very different.

One example given in the slides for classification is labeling of samples as either representative of AML, acute myeloid leukemia or ALL, acute lymphoblastic leukemia. This example is right out of a seminal paper by Todd Golub et al. from Science in 1999. The paper and associated data is available at [http://www.broad.mit.edu/cgi-bin/cancer/publications/pub_paper.cgi?mode=view&paper_id=43](http://www.broad.mit.edu/cgi-bin/cancer/publications/pub_paper.cgi?mode=view&paper_id=43). In that paper, both classification and clustering were used to put unknown samples into AML and ALL groups, given training samples (classification), and to discover subtypes of AML and ALL (clustering). The paper is mentioned in lecture 8 as well.

In the future, clustering and classification should lead to a ton of other interesting medical diagnostic tests for heart diseases, kidney problems, just about any health problem you can think of.

Microarrays

mRNA expression levels are commonly used as features for clustering samples. To get a large data set of expression levels, microarray analysis is used.

In microarray analysis, many complementary probes for RNAs are stuck to a chip. We take the RNA of interest and reverse transcribe it into fluorescently labeled cDNA, then dump it on the plate, wash the excess off, and see what sticks due to the complementation. We can see the levels of expression by measuring the fluorescent probe. (The slide gives a better overview than I can give in words.)

There are of course problems with noise, but they are out of the scope of this lecture/lecture note. We’ll just assume we get a perfect signal to noise ratio. Briefly, the cDNA can non-specifically hybridize, the reverse transcription may not be perfect, the cDNA can get stuck to the plate, there may be fluorescence quenching, and so on.
Once we have all this data, we can either try to cluster genes or cluster experiments/samples. The data is often represented as a matrix of genes and experiments. That is, each row represents the expression levels of a gene, and each entry in that row is the expression level of that gene in a different experiment/sample, with columns representing the expression levels of many genes in one experiment. Then, clustering experiments is analogous to finding similar columns, and clustering genes is like finding similar rows.

One of the major companies providing microarrays is Affymetrix, mentioned in passing on the lecture slides (http://www.affymetrix.com). If you want more technical information on microarray technology, you can read the technical manuals on their site, which are ridiculously thorough. The link is at the bottom right of their home page, but you’ll need to register to view the manuals.

**K-means clustering**

This is an example of partitioning, where we assign each point to exactly one cluster. Basically, we are trying to minimize the total distance from points to their corresponding centers. Or, as stated in the lecture, we are trying to make the most compact clusters possible, where compact might be defined by Euclidean distance or some other cost criterion or metric.

The algorithm works as follows (the slides do a much better job of explaining this, so I recommend reading them instead of this):

We make the assumption that there will be K clusters, with centers $\mu_k$. We will classify the points $x_i$.

1. Randomly place these K centers in space.
2. Iterate:
   a. Calculate the distance between every $x_i$ and $\mu_k$.
   b. Using the distance information assign the label k corresponding to the closest center $\mu_k$ to each point.
   c. Before the next iteration we move $\mu_k$ to the centroid of the points with that label k, and repeat.

**Fuzzy k-means**

This version of k-means tries to deal with the problem where points are somewhat in between centers. This is done by replacing distance with probability (which of course could be some function of distance), and by using a weighted centroid based on those probabilities.

Fuzzy k-means reduces to K-means if the probability function used is simply 1 if the center is the closest center and 0 otherwise.
Generative model and relation to HMMs

The hidden Markov model (HMM) is another common generative model—we will try to draw some parallels here between HMMs and the K-means and fuzzy K-means clustering algorithms.

What are we doing when we cluster? We are assuming something about the underlying data. In the case of K-means and fuzzy K-means, we are assuming that K centers (parameters) generate the data points using a Gaussian distribution. In the case where $K = 2$, this can be thought of as flipping a coin to choose one of the two centers, then randomly placing a point, according to a Gaussian distribution, somewhere near the center.

Now look back at the clustering algorithms. Given a point $x$ and a list of possible centers, we want to choose the most probable center (parameter) for $x$. Well, this just involves the argmax function. (Look at slides for formula.) After that, we want to find the best new parameter, the maximum likelihood $\mu_k$ for the next iteration of the algorithm. Well, this is also just argmax (it gets turned into an argmin when you simplify the math).

Notably, both K-means and fuzzy K-means are examples of EM (expectation-maximization algorithms). We find the expected missing labels, and then we update the model to maximize the likelihood of the parameters given those expected labels. EM is guaranteed to converge and guaranteed to find the best possible answer (at least from an algorithm point of view).

With that said, K-means can be thought of as similar to Viterbi learning, and fuzzy K-means is like the Baum Welch algorithm used with HMMs.

Changing the model, improving K-means

The example given in class where K-means fails is the two moons problem. Other methods (spectral clustering, for example) can solve this problem. K-means fails in the two moons problem because of the underlying assumptions.

So, to change our clusters, we can change the underlying assumptions—perhaps some centers are favored over others, or points are not distributed around centers according to a Gaussian distribution.

A second problem in K-means clustering is choosing $K$, the number of clusters. $K$ needs to be reasonable. If we select $K$ to be too large, we are potentially overfitting the data. If $K$ is the number of data points, then we have perfect clustering—just put the centers right on top of the data points, and voilà! You get a perfect clustering. However, those clusters wouldn’t tell you anything useful from a biological standpoint.

To get around this problem of selecting $K$, we can use hierarchical clustering.
Hierarchical clustering

Hierarchical clustering is the most widely used algorithm for mRNA expression data. The algorithm starts by putting each data point in its own cluster. At each step, the closest clusters are merged. This produces a tree, with clusters at every level. To get the number of clusters you want, just “cut” the tree at a certain level. The algorithm is given in the Jones/Pevzner book in section 10.2.

Of course, we will still need some sort of metric to determine which clusters are closest. Metrics are covered in the tables from the lecture slides.

Evaluating cluster performance

A few ways of evaluating cluster performance were discussed. The first is robustness. In other words, if the clusters we obtain are correct, we should be able to arrive at similar clusters using variations in the algorithm or data. We’d expect subsets of the points to cluster in similar ways. We’d also expect the points to cluster similarly given a (reasonably large) subset of their features. In the case of K-means clustering, we’d also expect a similar clustering starting with different centers.

The second way is to measure category enrichment, or to look for over-representation of genes related to the same biological function in a given cluster. Imagine we were clustering 100 genes, of which 10 were kinases. If 9 of the 10 kinases end up in a single cluster, intuitively, that would seem very significant and unlikely to have happened by chance. This makes our clusters “better”.

Now imagine the same experiment if 90 of the 100 genes were kinases. Then, the chance of getting 9 kinases in one cluster is much higher. We can use the hypergeometric distribution to quantify these probabilities.

A third way would be to use other (outside) biological data (a previous classification from a database, for example) and see if the clusters match up in some reasonable manner. However, this isn’t always possible. Perhaps your samples haven’t been tested/classified before, or maybe there are too many to test.

Self-organizing feature maps

Although they were mentioned earlier in the lecture, I put this section at the end since it was somewhat glossed over, and because SOMs were not really part of the main focus of the lecture.

I found it a little easier to follow in words, rather than the way it was presented on the lecture slide. The algorithm for using SOMs to cluster points is as follows:

1. Select centers \( m_i \) randomly (like k-means)
2. Iterate:
   a. Select a point \( x \) (still like k-means)
   b. Find the closest center \( m \) to \( x \) (still like k-means)
c. Find the centers within a certain “neighborhood” of \( m \) (the \( h \) function from slide 27 “Self-Organizing Feature Maps”) and move them closer to \( x \), based on their distance from \( m \) (the \( \eta \) from the slides).
   (This is slightly different from the lecture—on the lecture slide, the centers in a neighborhood of the point were moved instead of the centers in a neighborhood of the nearest center.)

d. Shrink the neighborhood and repeat, until the neighborhood is tiny and the result converges.

The further away the other neighborhood centers are from \( m \), the less they should move. This is one parameter you can adjust. The second parameter you can adjust is the speed at which the neighborhood shrinks. However, it is difficult to really come up with guidelines on setting either parameter for good clusters.

In general, shrink the neighborhood too fast, and the algorithm approaches k-means (since, if the neighborhoods are small, fewer centers will be affected.) Too slow, and the centers tend to move towards the centroid of the points (since many of the centers will be affected). I think the adjustability of these parameters is the advantage of SOMs over simple k-means clustering.

Some references for SOMs:
http://www.cs.princeton.edu/courses/archive/fall06/cos436/Duda/C/SOFM.htm
http://www.ai-junkie.com/ann/som/som1.html