**PROBLEM 1** **– Clustering**

Clustering is an important bioinformatic tool that helps uncover patterns and groupings in multi-dimensional data. In this problem you will explore several common clustering techniques used by bioengineers that assist in data interpretation.

1. **Hierarchical clustering** is an unsupervised clustering technique for finding a hierarchy of groupings in data. There are two forms of hierarchical clustering: **agglomerative**, in which all data points are initialized as their own cluster and a hierarchy is built up from these (“bottom up”), and **divisive**, where all observations start in one cluster and are further sub-divided recursively (“top down”). In this problem you will perform **agglomerative**, or “bottom up” clustering from a set of data points.

Two important metrics in hierarchical clustering are **distance** and **linkage**. The term “distance” is usually used to refer loosely to any criteria that defines the similarity/dissimilarity between two sets of points (e.g. Euclidean distance), even when those measures do not satisfy the mathematical criteria for a metric; linkage refers to a criteria that defines the similarity/dissimilarity between groups of points (i.e. between clusters, e.g. average linkage). The choice of distance and linkage metrics influence how the clusters are formed and should be chosen appropriately for the type of data.

1. Hierarchically cluster the 2D points below manually using **Euclidean distance and complete (maximum) linkage**. Show the steps for each round of the algorithm run and draw a dendgrogram representing your results.

(0,0), (3,0), (0,6), (21,2), (23,2)

1. Use the Matlab clustergram function to check your by-hand calculations using the same distance and linkage metrics. (See documentation on function to adjust metrics; Note: this Matlab function standardizes [z-scores] data by default; you should set the ‘standardize’ flag to ‘none’ to see the true clustering of the raw data).

C:\Users\Anthony\Documents\MIT\20.440 Spring 2016 Psets\Mar30_PSet\simple_2D_clustergram.emf

1. **K-means clustering** is another clustering technique that partitions *N* observations, where each observation is a D-dimensional vector, into *k* clusters, with each cluster defined by a center or “centroid”; each observation is assigned to one of the *k* clusters by finding the closest centroid based on some distance metric (e.g. Euclidean distance). The centroid is defined as the mean of the points in a given cluster. The overall aim of the algorithm is to reduce the within cluster sum of squares.

A simple iterative procedure is most commonly used to implement the algorithm:

1. Set the parameter *k*, for the number of clusters.
2. Initialization: Assign *k* initial guesses for the centroids (typically chosen as *k* random observations among the *N* total).
3. Classification: Assign each observation to one of the *k* clusters based on the minimum distance centroid.
4. Re-calculate centroids: use the observations contained in each cluster to re-calculate the *k* centroids.
5. Termination: Repeat steps (iii) – (iv) until the centroids no longer change (or change within some small epsilon threshold).
6. Manually cluster the following 2D data points with the k-means algorithm. Use **Euclidean** **distance** and ***k* = 2**. Do this procedure twice, using as initial guesses for the centroids:
   1. Points C and H
   2. Points B and I

Points:

A (0, 2)

B (0, 1)

C (1, 2)

D (2.5, 5)

E (3.1, 5)

F (3.1, 6)

G (4, 2)

H (5, 2)

I (5, 0)

1. Comment on how the choice of initial conditions affects the partitioning of the data points into *k=*2 clusters.
2. **Implement the k-means clustering algorithm for *k* ≥ 2 in Matlab.** Skeleton code is provided in the file problem1B\_kmeans.m. You should edit the function kmeans\_cluster, which takes three inputs: 1) a data matrix X (where each row is an individual n-D vector observation), 2) the value of *k*, and 3) a distance function. The code returns: 1) the cluster membership (a 1-D vector of cluster assignments for each observation), 2) the centroids, and 3) the distance of each observation to its calculated centroid. Edit the included function distance, which already contains code for Euclidean distance, to include code for the ‘cityblock’ (or the L1) distance metric:

The output of this function is an “anonymous” function, which implements the specified distance metric and can be passed as an argument to your k-means code (e.g. distFun = distance(‘Euclidean’); d = distFun(x,y) to define the distance function as Euclidean and to compute this distance between two vectors x and y).

1. Run and test your k-means function on the 2D data points from (1) above. How do the output values for the cluster assignments and centroids change when running the algorithm across several different random start points?
2. You have likely wondered how one picks the appropriate value of *k* for a given clustering problem and noticed that the results of the algorithm are sensitive to initial conditions. To assist in choosing *k*, run the algorithm multiple times across several values for this parameter. For both distance metrics coded, plot the mean total error function (i.e. sum of squared distances from the centroids for points in a cluster) as a function of *k* and select a seemingly appropriate value for *k* based on the “elbow method”, where one finds the value of *k* where the reduction in error with additional complexity (i.e. clusters) appears to plateau. Justify your value of *k*, comparing the results from the two distance metrics, and re-run the algorithm multiple times at the “optimal” value of *k* for each, saving the cluster assignments and calculated centroids for the runs that minimize the global error. Plot the 2D points and cluster centroids as scatter plots, coloring the points by cluster membership. Comment on your selection of *k* and the overall clustering results. Also comment on what effect (if any) the choice of distance metric has on the final cluster assignments.
3. **Fuzzy k-means (c-means) clustering** is an extension of k-means clustering that allows for “soft” assignment of data points to clusters, as opposed to the algorithm you implemented above which performs “hard” assignment. In fuzzy k-means, every point belongs to every cluster with varying degrees of membership. The output of this algorithm is an *N*x*k* membership matrix, where each element is a fractional value representing how much a given data point “belongs” in a cluster (the rows of the matrix sum to one). Fuzzy k-means introduces an additional parameter *m* (>1), called the “fuzzifier”, that determines how hard or soft the assignments are to the *k* centroids.

The steps for implementing this algorithm on data matrix *X* are:

1. Choose the number of clusters *k*
2. Initialize *k* centroids
3. Calculate a distance matrix *D*, where *D* is an *Nxk* matrix of distances (according to some distance function) of each point from each centroid.
4. Update a partition matrix *U* (*Nxk*)according to:

Where *dij*is an individual element of *D* and *dic* is the distance for data point *i* to the centroid of cluster *c*.

1. Update the centroids C according to:
2. Terminate when the change in partition matrix is less than a small value epsilon, e.g. |Unew-Uold| < ε.
3. **Implement the fuzzy k-means algorithm for *k* ≥ 2 in Matlab.** Skeleton code is provided in the file problem1C\_fuzzy\_kmeans.m. You should edit the function fuzzy\_kmeans, which takes the same three inputs as the k-means function from earlier, along with an additional fourth input for the fuzzifier parameter. You can also use the distance function you created for k-means here. Note that the initial selection of centroids is done as before, though Gaussian noise is added to avoid divide by zero errors in the initial *U* matrix calculation.
4. Load the 2D data in data1C.mat, which is 100x2 data matrix, and run the fuzzy k-means algorithm on these values with ***k=*2 and Euclidean distance**. Use values of **1.1, 2, 4, and 8** for the fuzzifier parameter *m*. In a single figure, plot the calculated centroids along with data points as a 2D scatter plot. Color points with “strong” membership to a given cluster (i.e. a membership values > 0.6) with one of two colors and mark with an ‘x’ points with weak membership to clusters. How does the fuzzifier parameter affect the assignment of points to clusters?
5. Cluster the data contained in the file Reddy\_100nM\_EGF.mat using **hierarchical and k-means** approaches. This file contains recent data from experiments in which MCF-10A cells were treated with EGF (100 nM here) and phosphotyrosine peptides were measured in short (10 second) intervals post-stimulation (Reddy et al., *PNAS*, 2016). The data structure contains four elements: 1) ‘data’, which contains the phosphotyrosine measurements (rows are proteins/residues, columns are times), 2) times, string labels for the time points, 3) proteins, string labels for the proteins/gene symbols, and 4) sites, with string labels for the phospho residues.
6. Normalize the data row-wise (peptide-wise) as standard scores, or z-scores.
7. Use the Matlab clustergram function to hierarchically cluster the peptides (rows) of the data matrix. Describe any patterns that seem to emerge.
8. Use your k-means code to determine a reasonable value for *k* (search over 2-10) and cluster the peptides according to this value. Plot the profiles for the peptides contained within each cluster, along with the centroids, for each cluster. Describe the types of trajectories observed in response to the 100 nM EGF stimulus.

**PROBLEM 2** – **Gene Set Enrichment Analysis (GSEA)**

GSEA is a tool designed for interpretive analysis of expression datasets (genes, proteins, etc.) that searches for enriched categories of genes (i.e. gene sets) at the top or bottom of a ranked list of data. There are several choices for the ranking metric, though the purpose is to rank genes according to their correlation with a phenotype (e.g. control versus disease). The method works by moving down a ranked list of species and updating a running sum statistic: if it encounters a species in the current gene set, the statistic is increased; if not, the statistic is decreased. The method as described in Subramanian et al. (*PNAS*, 2005) calculates this running enrichment score (ES) according to:

1. Rank order your *N* genes by some metric against the phenotype, giving ***r****(gj) = rj* for the rank (correlation) scores for the sorted gene list.
2. Walk down the sorted list of genes and increase a running score by *Phit* if the current gene *g* is in the gene set *S*, or decrease by *Pmiss* otherwise, where:

And where *NH* is the number of genes in the gene set *S* and *p* is an exponent that influences the behavior of the running score when a gene is encountered (e.g. if *p =* 0, it reduces to the standard Kolmogorov-Smirnov statistic; if *p* = 1, the genes in *S* are weighted by their correlation with the phenotype normalized by the sum of all correlations).

1. Determine the maximum ES as the maximum deviation from zero of *P­hit* – *Pmiss*.
2. On the data in “mock\_expression\_data.txt”, which contains measurements for 26 “genes” A through Z across three samples of each of two phenotypes “red” and “green”, use the **signal-to-noise ratio (SNR)** metric to correlate the genes with the two phenotypes:

Where *μ* is the mean expression level for the gene in the red/green phenotype samples and *σ* is the standard deviation. Rank the genes according to the calculated SNR values (high to low). You may do these calculations in Matlab, Excel, or by-hand. Report the ranked list of genes along with the SNR values.

1. Use as gene set *S* the genes [A, B, C] and exponent *p* = 1 to generate a running ES plot and to calculate the max ES for this gene set. Does your result suggest that this gene set is enriched in the data?
2. **Write a Matlab function that implements the GSEA running ES score calculation**. The function should take a ranked vector of phenotype correlation values (e.g. SNR values), the corresponding genes, a gene set, and the *Phit* exponent value *p*. The function should return the running scores (for eventual plotting) and the maximum ES.
3. Use your code to plot running ES statistics and report the maximum ES for the following gene sets:
4. [A, B, Y]
5. [X, Y, Z]
6. [A, H, O, V]
7. [L, M, N, O]
8. A proposed method for significance testing on the maximum ES is to permute the input data, compute maximum ES values for the permuted data against the gene sets, generating a null distribution of scores (ESNull), and to test how extreme the true ES value is against this background (i.e. the fraction of times you observe a value in ESNull greater than or equal to the true value). When comparing the true ES to ESNull, only use the portion of the distribution with the same sign as the true value. The default permutation method recommended for GSEA is to randomly re-assign the phenotype labels, thereby preserving gene-gene correlations in the original data; however, as is often an issue in high-throughput studies, small sample sizes per condition (i.e. < 7 samples) limit the utility of this method. The alternative method is to permute the gene labels and perform the significance calculations on these samples. Sine you have only three samples per condition in your mock data, use the **gene label re-sampling method and run 1000 permutations** for the four gene sets in (D), reporting the calculated significance values. Which gene sets appear to be significant and how do these correlate with phenotype?
9. Now you will run GSEA on a real set of gene expression measurements against a small set of KEGG pathways. The expression measurements are from a study in which samples from patients with acute lymphoblastic leukemia (ALL) or acute myelogenous leukemia (AML) were profiled (Armstrong et al., *Nature Genetics*, 2002).
10. Load the expression data contained in Leukemia\_data.mat, which includes:

* Lekemia\_data.data – a 10,056 x 48 matrix of ALL and AML expression measuremnets
* Lekemia\_data.genes – a 10,056 element vector of gene names
* Lekemia\_data.phenotypes – a 48 element vector of sample phenotypes

1. Load the pathway data in human\_kegg\_path\_to\_gene\_small.mat, which is a mapping container (similar to a python dictionary) keyed by pathway names and containing cell objects of genes in the pathway. The code below shows how you can access this data:

% Load the pathways container and extract pathway names

load(‘human\_kegg\_path\_to\_gene\_small.mat’);

pathway\_names = p2g.keys();

% Access the genes in one of the pathways

gene\_set = p2g(char(pathway\_names(1)));

The gene\_set variable is a cell array of gene names contained in the first pathway.

1. Compute SNR values for the ALL vs. AML samples and rank accordingly.
2. For the six pathways, run your GSEA code for the maximum ES value and run 1000 additional random permutations to compute the p-values for each of these. You may either permute according to the gene labels or according to phenotype labels. Report running ES plots, maximum ES values, and p-values for each of the six pathways. Which pathways appear significantly enriched in the genes associated with differences in the ALL and AML phenotype labels?

\*Note that the GSEA software available from the Broad (and described in Subramanina et al.) also implements an FDR correction when testing for multiple hypotheses. You are not responsible for implementing this procedure. The details of the FDR procedure are described in the GSEA paper as well as in the user manual for the GSEA code.

(<http://software.broadinstitute.org/gsea/doc/GSEAUserGuideFrame.html>)