Homework Assignment 3
Due by end of the day Dec 5th via canvas

SDS 385 Statistical Models for Big Data

1. **Theory of LSH** Recall that a locality sensitive hashing scheme is a set $\mathcal{F}$ of hashing functions that operate on a set S of objects, such that for two objects $x, y \in S$,

$$P(h(x) = h(y)) = sim(x, y),$$

where $sim(x, y) : S \times S \to [0, 1]$ is a similarity function.

   (a) Let $d(x, y) = 1 - sim(x, y)$. Prove that sim(.) to have a locality sensitive hashing scheme, $d(x, y)$ should satisfy the triangle inequality.

$$d(x, y) + d(y, z) \geq d(x, z) \quad \text{for all } x, y, z \in S$$

   (b) Consider the following two similarity functions for two sets A and B. The overlap similarity function:

$$sim_{over}(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)}$$

   and the Dice similarity function

$$sim_{dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

   Is there a locality sensitive scheme for these? Prove or give a counter example.

   (c) Sometimes it is more convenient to have a hash function family that maps objects to $\{0, 1\}$. In that case, the output of T different hash functions can simply be concatenated to obtain a T-bit binary hash string for an object. Here you will show that we can always obtain such a binary hash function family with a slight change in the similarity measure. Given a locality sensitive hash function family $\mathcal{F}$ corresponding to a similarity function $sim(x, y)$, give a mechanism to obtain a locality sensitive hash function family $\mathcal{F}'$ that maps objects to $\{0, 1\}$ and corresponds to the similarity function $\frac{1+sim(x,y)}{2}$.

2. **Spectral Clustering**[1] In class we have seen k-means and EM for estimating GMM's. Since your professor will not get a chance to actually cover her favorite clustering method aka spectral clustering, methinks this is a great opportunity to introduce you to it. There is a class of clustering algorithms, called spectral clustering algorithms, which has recently become quite popular. Many of these algorithms are quite easy to implement and perform well on certain clustering problems compared to more traditional methods like k-means.

[1] This problem was designed in collaboration with Jon Huang

In this problem, we will try to develop some intuition about why these approaches make sense and implement one of these algorithms.

Before beginning, we'll review a few basic linear algebra concepts you may find useful for some of the problems.

- If A is a matrix, it has an v with eigenvalue λ if $Av = \lambda v$.
- For any $m \times m$ symmetric matrix A, the *Singular Value Decomposition* of A yields a factorization of A into
$$A = USU^T$$
where U is an $m \times m$ orthogonal matrix (meaning that the columns are pairwise orthogonal). and $S = diag(|\lambda_1|, |\lambda_2|, \ldots, |\lambda_m|)$ where the λ_i are the eigenvalues of A.

Given a set of m datapoints $x_1, \ldots, x_m$, the input to a spectral clustering algorithm typically consists of a matrix, A, of pairwise similarities between datapoints often called the *affinity matrix*. The choice of how to measure similarity between points is one which is often left to the practitioner. A very simple affinity matrix can be constructed as follows:

$$A(i,j) = A(j,i) = \begin{cases} 1 & \text{if } d(x_i, x_j) < \Theta \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $d(x_i, x_j)$ denotes the Euclidean distance between points x_i and x_j.

The general idea of spectral clustering is to construct a mapping of the datapoints to an eigenspace of A with the hope that points are well separated in this eigenspace so that something simple like k-means applied to these new points will perform well.

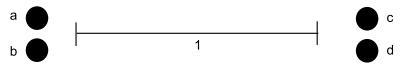

Figure 1: Simple dataset

As an example, consider forming the affinity matrix for the dataset in Figure 1 using Equation 1 with $\Theta = 1$. We have that

$$A = \left[\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 0 \\ b & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 1 & 1 \\ d & 0 & 0 & 1 & 1 \end{array}\right]$$

Now for this particular example, the clusters $\{a, b\}$ and $\{c, d\}$ show up as nonzero blocks in the affinity matrix. This is, of course, artificial, since we could have constructed the matrix A using any ordering of $\{a, b, c, d\}$. For example, another possibile affinity matrix for A could have been:

$$\tilde{A} = \left[\begin{array}{c|cccc} & a & c & b & d \\ \hline a & 1 & 0 & 1 & 0 \\ c & 0 & 1 & 0 & 1 \\ b & 1 & 0 & 1 & 0 \\ d & 0 & 1 & 0 & 1 \end{array}\right]$$

The key insight here is that the eigenvectors of matrices A and $\tilde{A}$ have the same entries (just permuted). The eigenvectors with nonzero eigenvalue of A are: $e_1 = (.7, .7, 0, 0)^T$, $e_2 = (0, 0, .7, .7)$. And the nonzero eigenvectors of $\tilde{A}$ are: $\tilde{e}_1 = (.7, 0, .7, 0)^T$, $\tilde{e}_2 = (0, .7, 0, .7)$. Spectral clustering embeds the original datapoints in a new space by using the coordinates of these eigenvectors. Specifically, it maps the point x_i to the point $(e_1(i), e_2(i), \ldots, e_k(i))$ where $e_1, \ldots, e_k$ are the top k eigenvectors of A. We refer to this mapping as the *spectral embedding*.

Algorithm description

Frequently, the affinity matrix is constructed as

$$A_{ij} = \begin{cases} 1 & \text{If } i, j \text{ are amongst } k \text{ nearest neighbors of each other} \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

The best that we can hope for in practice is a near block-diagonal affinity matrix. It can be shown in this case, that after projecting to the space spanned by the top k eigenvectors, points which belong to the same block are close to each other in a euclidean sense. We won't try to prove this, but using this intuition, you will implement one (of many) possible spectral clustering algorithms. This particular algorithm is described in

`On Spectral Clustering: Analysis and an algorithm`
`Andrew Y. Ng, Michael I. Jordan, Yair Weiss (2001)`

We won't try to justify every step, but see the paper if you are interested. The steps are as follows:

- Construct an affinity matrix A using Equation 2.
- Symmetrically 'normalize' the rows and columns of A to get a matrix N: such that $N(i, j) = \frac{A(i,j)}{\sqrt{d(i)d(j)}}$, where $d(i) = \sum_k A(i, k)$.
- Construct a matrix Y whose columns are the first k eigenvectors of N.
- Normalize each row of Y such that it is of unit length.
- Cluster the dataset by running k-means on the set of spectrally embedded points, where each row of Y is a datapoint.

(a) Generate a dataset with 10000 points, with 5000 coming from a Gaussian centered at $\mu_1 = 3/\sqrt{p}$ with $\Sigma_1 = I_p$ and the rest from a mean $\mu_2 = -3/\sqrt{p}$ gaussian with $\Sigma_2 = I_p$. Create a k-nearest neighbor graph from this dataset and do Spectral clustering. For $p \in \{250, 500, 1000, 2000\}$, plot the time taken and the clustering accuracy of the following algorithms, averaged over 10 randomly generated datasets. You just have to write your own code for Spectral clustering, you can use available software like E2LSH or packages for kdtree for the rest. There are four algorithms, *Exact*, *JL+Exact*, *JL+KDtrees* and *E2LSH*. Use $k = 5$. You can also try out different k, its interesting to try this because too small or too large k can lead to a bad clustering accuracy. Remember that you can calculate the clustering accuracy because you generated the data and hence know the "latent" ground truth memberships.

i. (Exact) Use the brute force k-nearest neighbor algorithm. If this is taking too long, you may omit the results, but please write explicitly how long it took, e.g. "my k-nn graph took over yyy hours to build when $p = xxx$ and so I gave up."

ii. (JL+Exact, JL+KDtrees) Use the Johnson Lindenstrauss lemma to reduce the dimensionality of the data. Remember, for n points JL lets you project the datapoints into a $O(\log n/\epsilon^2)$ dimensional space with a multiplicative distortion of ϵ of the distances. Try out different ϵ values to generate the curves. For small ϵ, you would have higher accuracy and longer processing time, whereas for larger ϵ you would have lower accuracy and less computation time. Now use the KDtree algorithm and exact k-nearest neighbors to build the knn tree. There is a builtin function in Matlab using knnsearch which has an option of using the KDtree.

iii. (E2LSH) Use E2LSH to implement Locality sensitive hashing to obtain the nearest neighbor graph. You can find a matlab package here: `http://ttic.uchicago.edu/~gregory/download.html`.