Machine Learning Final Project: Locally Linear Embedding for Visualizing High-Dimensional Data

Blake Shaw <bs2018@columbia.edu>

May 6th, 2004

1 Introduction

In many fields ranging from Genomics, to Sociology we are presented with large amounts of high-dimensional data. With the advent of being able to collect this wealth of information, a common question has emerged: How can we extract meaningful visual information from large high-dimensional data sets? Ideally, we want to represent significant relationships between N items in a D dimensional space using only 2 dimensions, allowing the data to be plotted, and easily visualized. In this paper, I show the results of using Locally Linear Embedding (LLE) to visualize information about the musical listening habits of the Columbia community.

1.1 Data

The data used for this project is a D x N table, where D is 2000 musical artists, and N is 68 musical listeners. Each entry of the table represents how many times a user has listened to a certain artist. This data was aggregated by the CUtunes project.

1.2 Goals for Analysis

Ideally we would like to visualize this data in two ways:

i. **Plotting maps of users**

Distances between users represents musical compatibility between the two users, and clusters in the map represent groups of people with very similar musical tastes

ii. **Plotting maps of artists**

Distances between artists represents some idea of how similar two artists are, and clusters in this case should correspond to some idea of "genre."

2 Implementation

See the Code (attached) for full implementation details.

2.1 A Brief Description of the Algorithm

LLE is comprised of 3 steps. First we compute the distance from each point to it's K nearest neighbors. We then create a matrix of reconstruction weights W minimizing the cost specified by this distance matrix. We can then use W to reduce the problem of finding a low dimensional embedding to a sparse eigenvalue problem. Note: The only free parameter in this algorithm (assuming we are embedding into a 2-dimensional space) is the value of K. Please see the Results section for more about picking the best K.

2.2 Estimating Distances

I have experimented with two different ways of comparing multi-dimensional probability distributions (representing either artists, or users), in order to create a NxN distance matrix where each entry represents the distance between two items in D dimensions. P is the probability distribution of the first item and Q is the distribution of the second.

i. **Euclidean Distance**

$$distance(P,Q) = \sqrt{\sum_{i=0}^{D}(P_i - Q_i)^2} \qquad (1)$$

ii. **Kullback-Leibler divergence**

$$distance(P,Q) = \sum_{i=0}^{D} P_i \log \frac{P_i}{Q_i} \quad (2)$$

3 Results

3.1 Discussion of Initial Trials

For my initial trials with LLE, I picked values for K which produced maps which made the most sense in terms of my own intuition about the data. Figures 1-4 (Attached) show maps for artists and users, using euclidean and KL divergence metrics. Lines are drawn between the 5 nearest neighbors for each item.

3.1.1 Artist Maps

Assessing the quality of these maps is a very subjective task. However upon inspecting the artist plots (see attached figure 5), we see a variety of features which make sense. For instance there is a close proximity of "The Shins" to "Shins, The". It is a good sign that an artist with two entries in the data still occupies the same general region of the space. Furthermore, certain areas appear to be populated with artists of a specific type or "genre" of music. Although many data points appear to not be correctly placed into a specific type, this "noise" could simply be attributed to a lack of sufficient data for those points. Upon a first examination of this problem, I believe that there simply isn't enough data to properly relate two artists to each other. Out of the 100 most popular artists shown on these maps, only approximately 20 percent have significant contributions by more then 10 users. It should be expected that many more dimensions are needed to properly define similarity between musical artists.

3.1.2 User Maps

It is easier to gauge the quality of these maps then the artist maps, because one can simply look at two users CUtunes data to see if they are in fact listening to many of the same artists. Furthermore, in this case the data is more sufficient (68 points in 2000 dimensions). Although the bottom section of the data is pretty sparse, it still contains valuable information for comparing two users. In general this map seems to make sense given the dataset.

An interesting feature to note is that by examining the connections drawn between the K nearest neighbors, we find distinct hubs. In most social networks (scale-free), the degree of connectivity typically obeys a power-law distribution. It makes sense that we would see a large, highly connected component, with groups of individuals scattered to the periphery based on terms which make them and their neighbors different from the masses.

Furthermore, it is interesting to compare how these two different distance metrics represent these hubs. It is not completely clear from figures 2 and 4. However, looking over a larger range of K, it appears that the KL Divergence metric groups highly connected items closer together then the euclidean metric; this fact probably comes from the relation of KL Divergence to entropy.

3.2 Picking an Appropriate K Value

The task of picking a value for K which provides the best embedding is still a very open problem. One proposed method is to look at which values of K provide the clearest distinction between the eigenvalues used for the embedding and the eigenvalues excluded. If there is a large gap, it means LLE is not as sensitive as when there is a small gap. This method is tied to matrix stability issues. To roughly approximate the size of the eigen-gap, I chose to simply plot the difference between the d+1 lowest eigenvalue (used in the embedding), and the one directly next to it (not used). Figure 6 (attached) shows these results. Generally for K greater than 10, there is little change in the maps produced. It was good to see that my hunch of picking between 8-12 was correct; there does exist a high maximum there. However this method finds K = 20 to be optimal. Despite this fact, I believe, based on intuition, that a smaller value of K is more accurate.

4 Conclusion and Future Directions

There are two directions in which I would take this project provided with more time. My primary concern with the results is that I believe there was insufficient data. Some points seem like noise. It would be helpful to incorporate some sort of metric

for how much information is provided for each item, and whether that information is sufficient to properly classify it. Secondly, The data appears to follow patterns typical of other social networks. By modifying the distance function to incorporate that the average degree follows a power law distribution it may be possible to better visualize the data. In conclusion I look forward to running LLE on a larger data set, and further examining how the network of musical artists and listeners behaves like other social networks.

References

[1] Dick de Ridder and Robert P.W. Duin. Locally linear embedding for classification.

[2] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding.

[3] Sam T. Roweis and Lawrence K. Saul. Think globally, fit locally: Unsupervised learning of low dimensional manifolds.

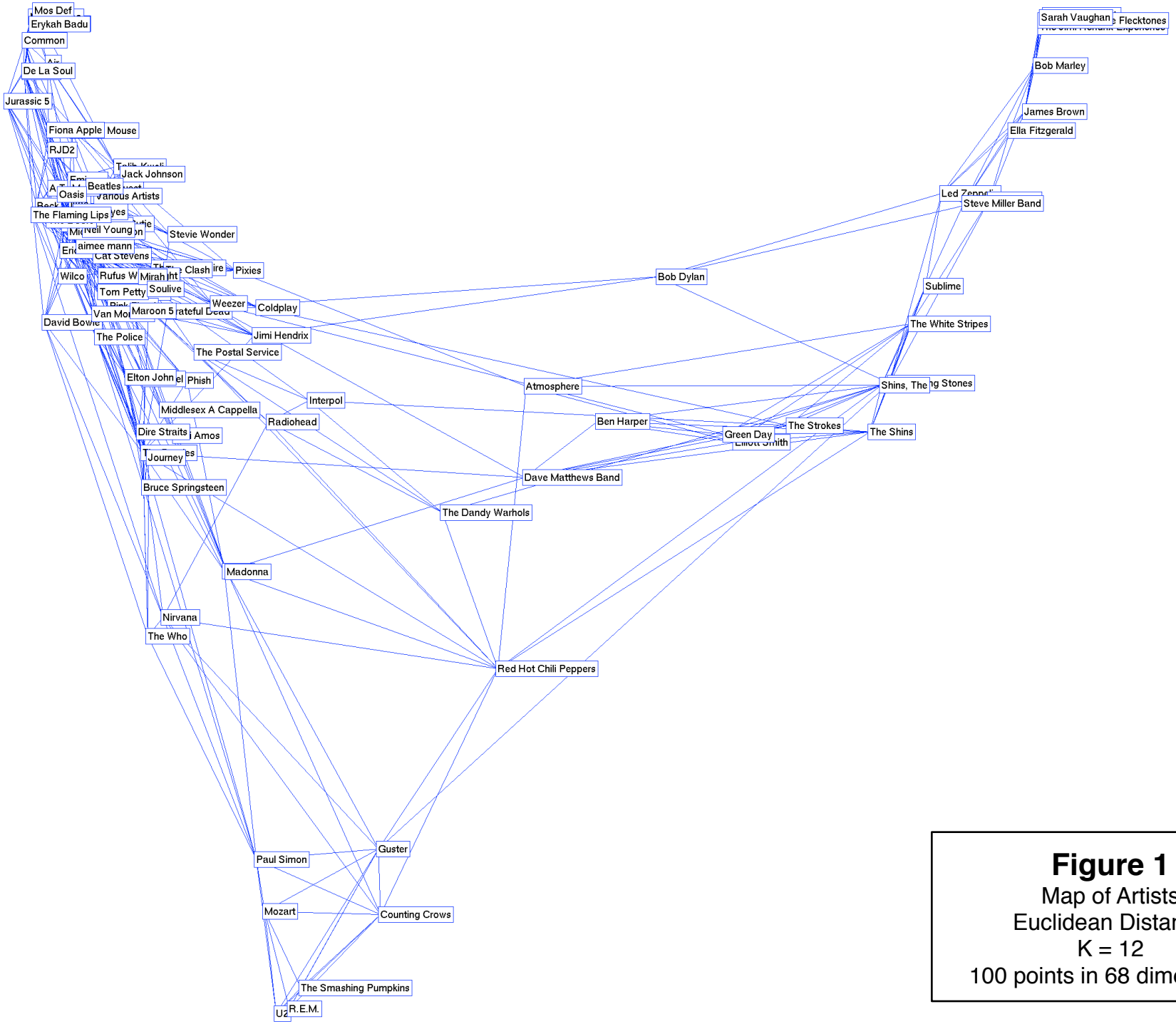


Figure 1

Map of Artists
Euclidean Distance
K = 12
100 points in 68 dimensions

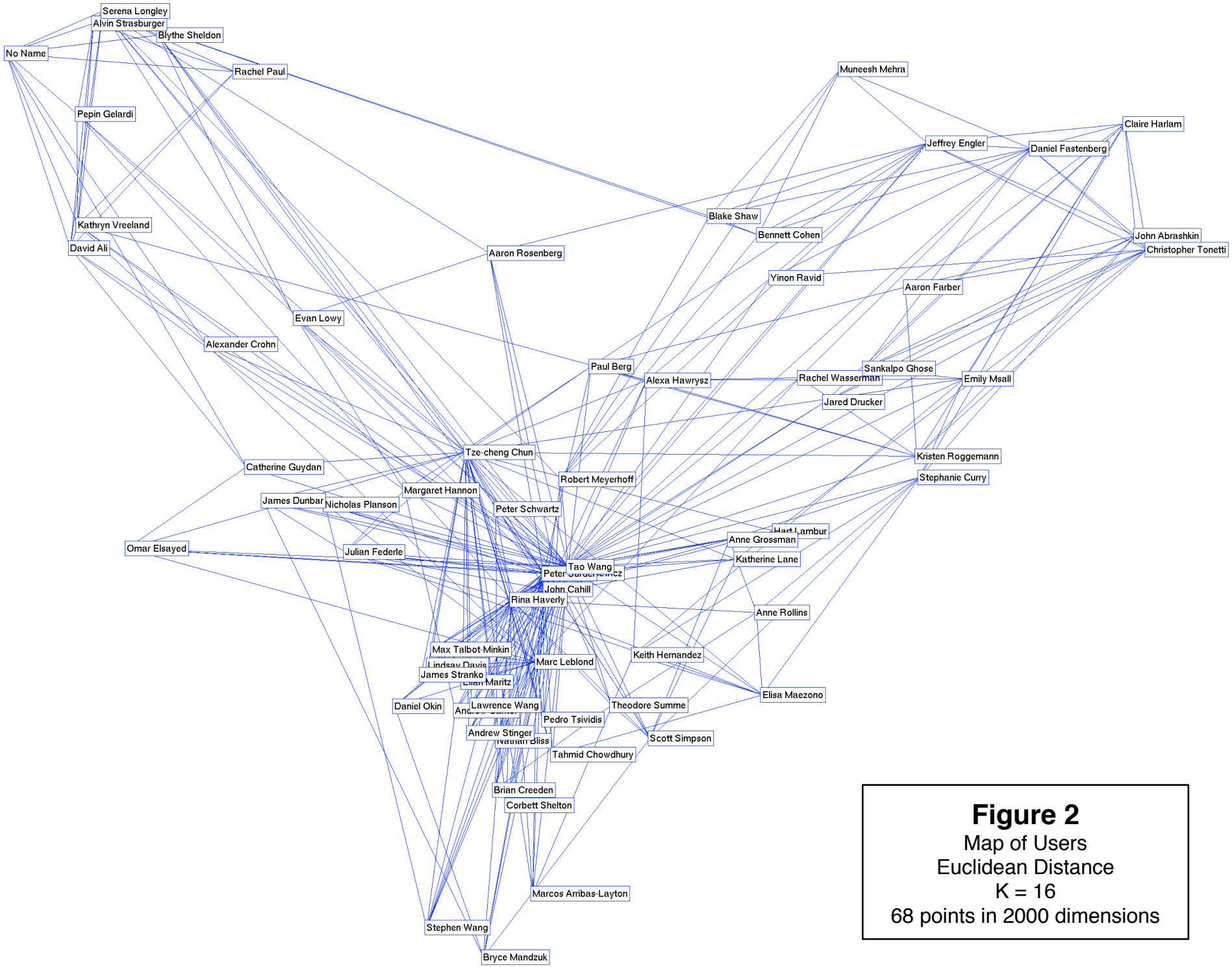


Figure 2
Map of Users
Euclidean Distance
K = 16
68 points in 2000 dimensions

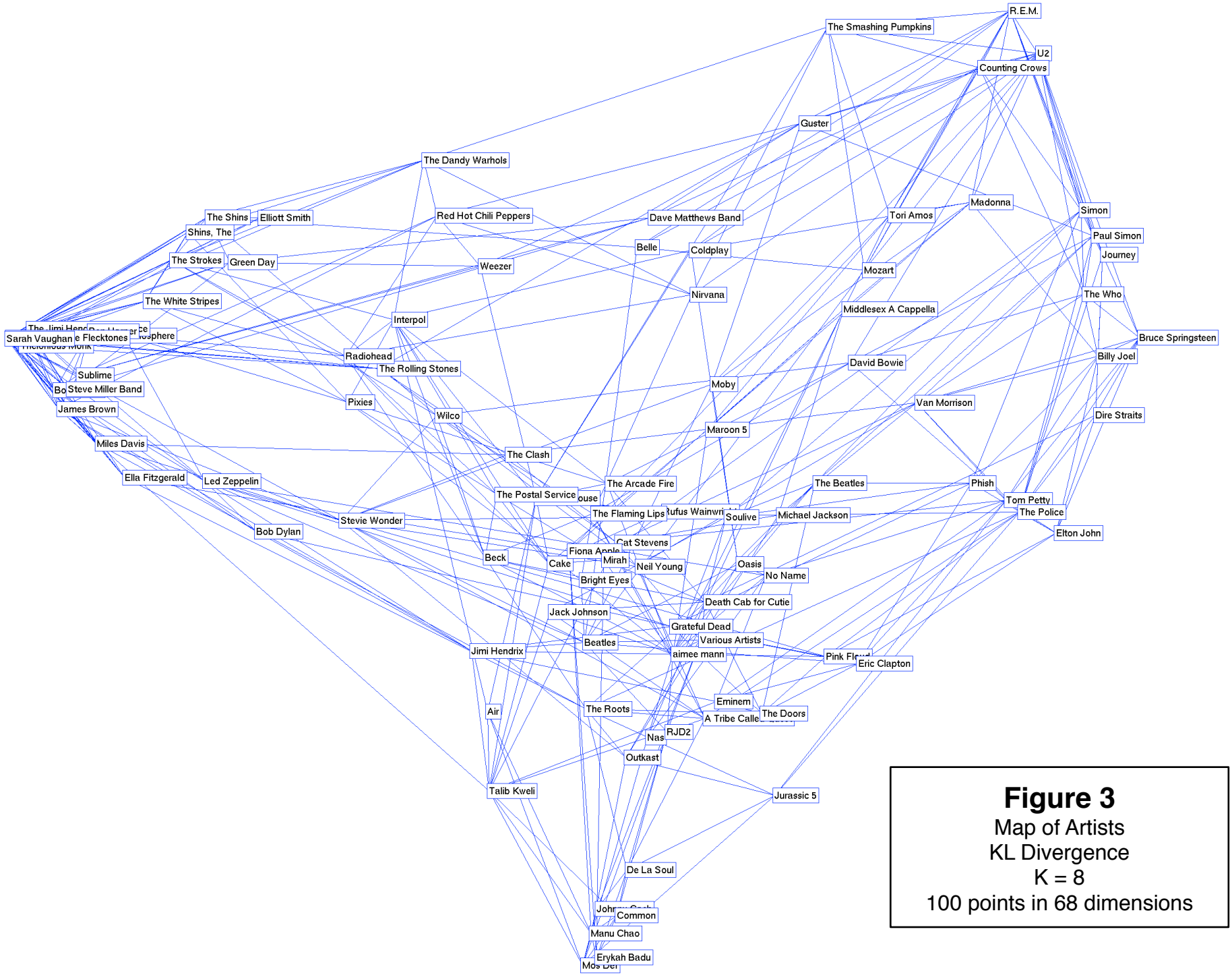
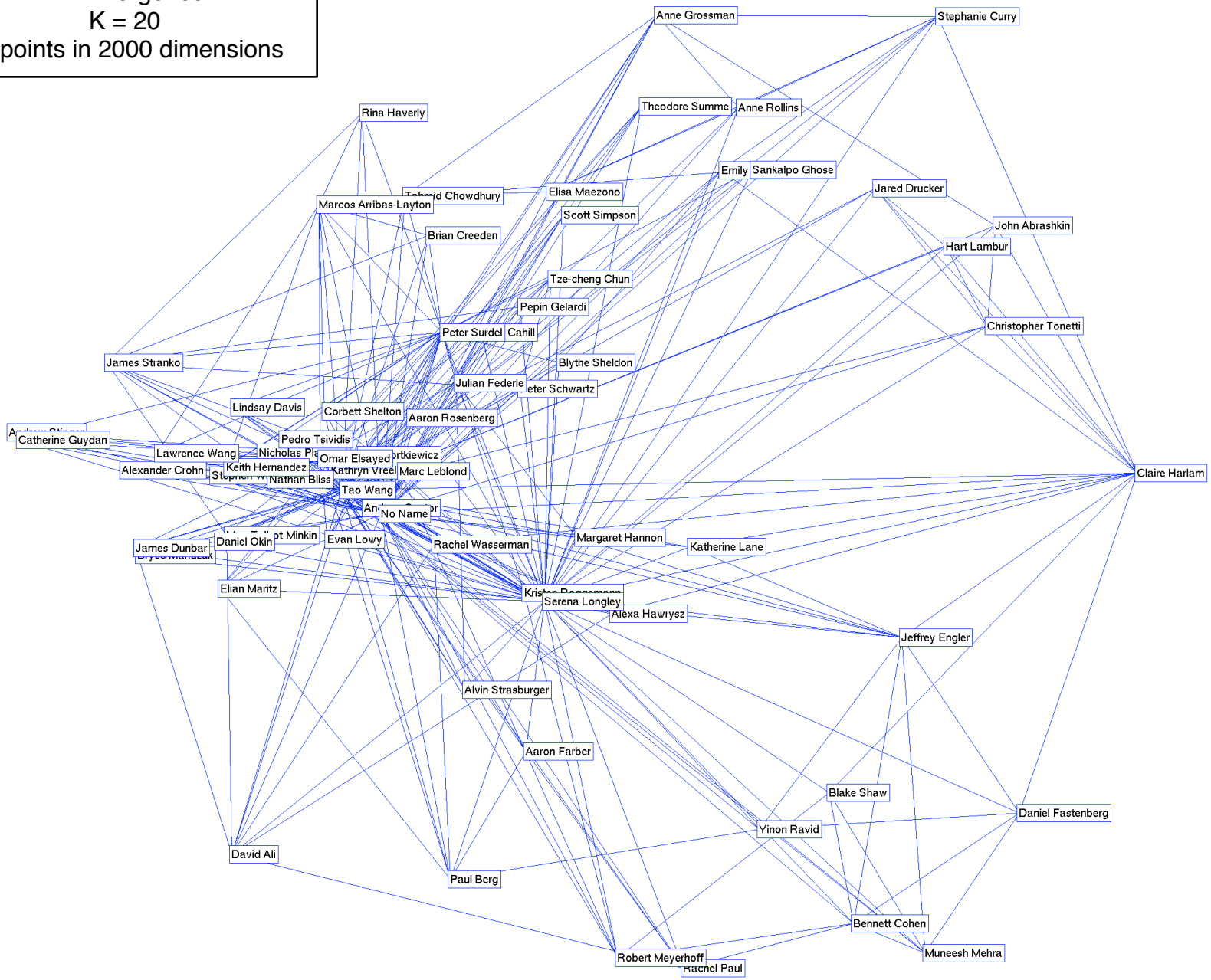


Figure 3
Map of Artists
KL Divergence
K = 8
100 points in 68 dimensions

Figure 4
Map of Users
KL Divergence
K = 20
68 points in 2000 dimensions



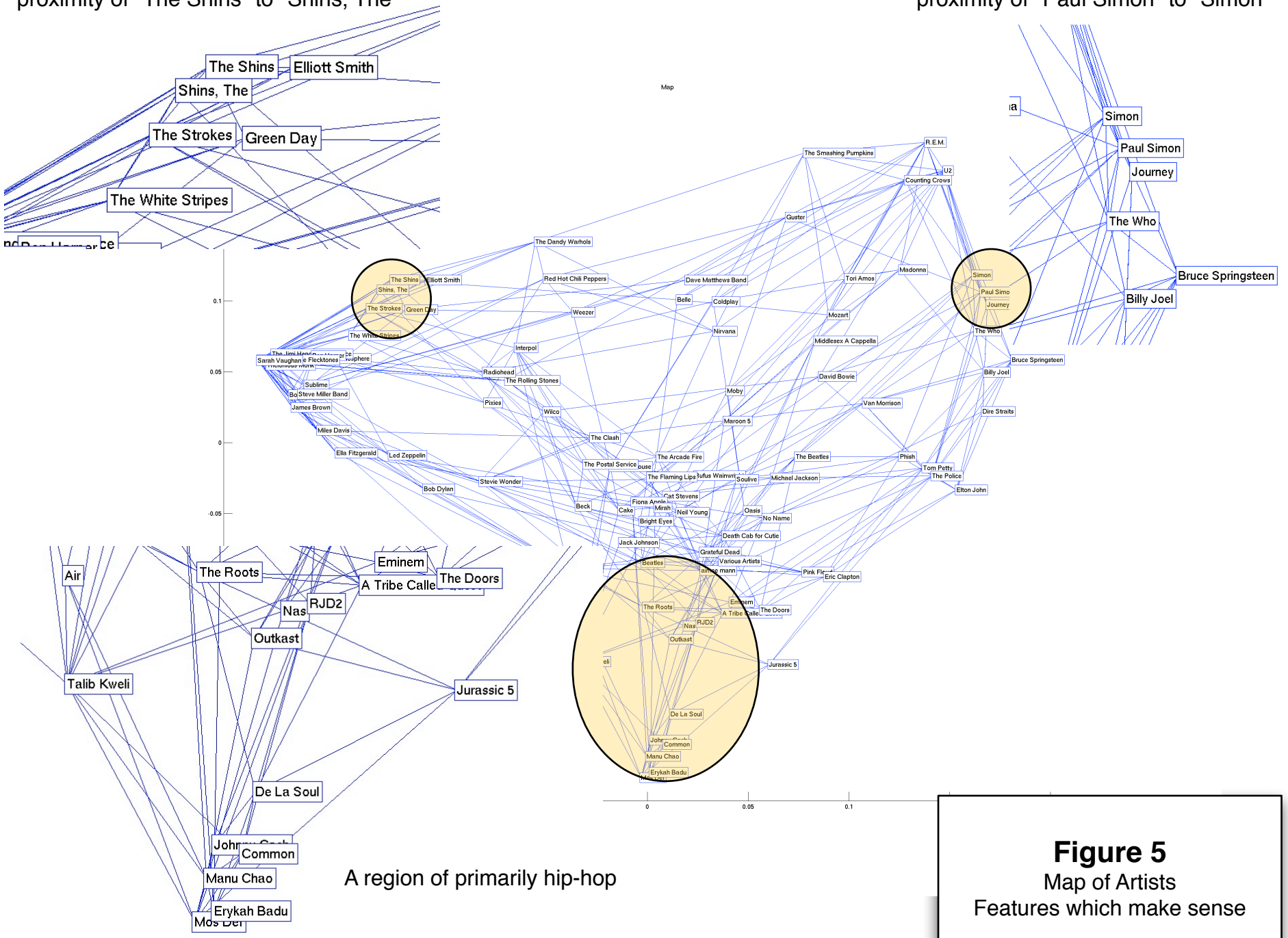
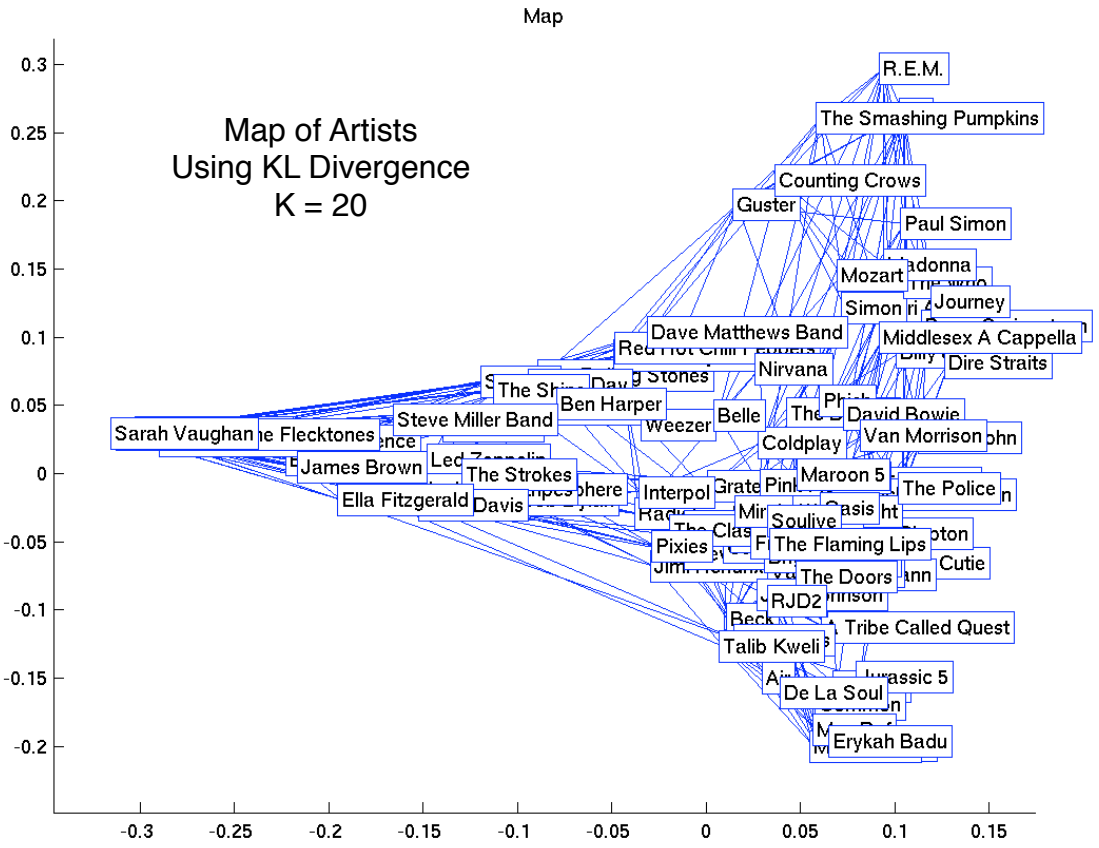
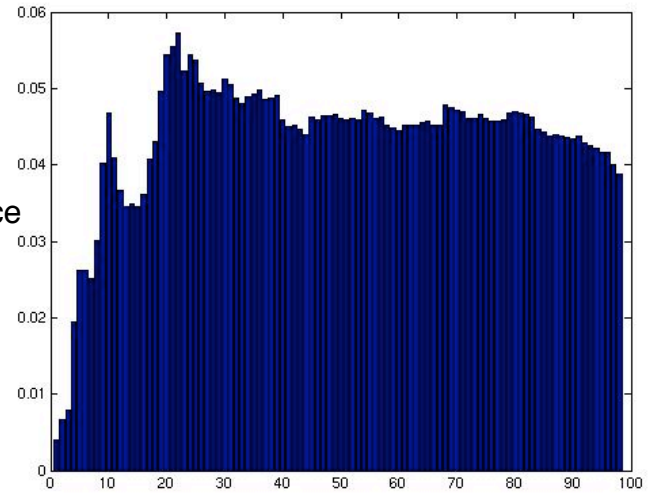


Figure 5
Map of Artists
Features which make sense

Figure 6
Picking the best K value





Plot of differences between the last eigenvalue in the embedding and the eigenvalue directly next to it. We see a distinct maximum at K = 20, and a peak at K = 9.

```
% Locally Linear Embedding
% Blake Shaw
% Machine Learning
% Columbia University
%
% [Y, distance, eigStrength] = lle(X, labels, K,dOutput, plotType)
%
% X = D x N matrix where N is the number of points in a D dimensional↙
space
% K = number of neighbors
% labels = labels for points
% dOutput = number of dimensions for output Y
% Y = points in a dOutput dimensional space
% distance = N x N matrix representing distances between points in N
% dimensions

function [Y, distance, eigStrength] = lle(X, labels, K,dOutput, plotType,↙
distanceType)

[D,N] = size(X);
disp(sprintf('LLE for %d points in %d dimensions',N,D));

disp(sprintf('Calculating Distance Matrix'));
distance = ones(N, N);


%normalize?
for i=1:N
    theSum = sum(X(:, i)) + D;
    for j=1:D
        X(j, i) = (X(j, i) + 1) / theSum;
    end
end

if distanceType == 1
    disp(sprintf('Using Euclidean Distance'));
    for i=1:N
        for j=1:N
            dist = 0;
            for k=1:D
                if(i ~= j)
                    %euclidean distance
                    A = X(k, i);
                    B = X(k, j);
                    dist = dist + (A - B)^2;
                end
            end
            distance(i, j) = dist^0.5;
        end
    end
elseif distanceType == 2
    disp(sprintf('Using KL Divergence'));
    for i=1:N
        for j=1:N
            dist = 0;
            for k=1:D
```

```
                if(i ~= j)
                    %kl divergence
                    P = X(k, i);
                    Q = X(k, j);
                    dist = dist + P * log(P / Q);
                end
            end
            distance(i, j) = dist;
        end
    end
end

disp(sprintf('Finding %d nearest neighbors', K));
[sortedDistances,indexes] = sort(distance);


% disp(sprintf('Printing Neighbors to file...'));
% fid = fopen('UserDistances.txt', 'w');
% for printUser=1:N
%     fprintf(fid, 'Neighbors for %s\n', labels(printUser, :));
%     for(i=1:N)
%         fprintf(fid,'     --->%s -- %6f\n', labels(indexes(i,↙
printUser), :), distance(printUser, indexes(i, printUser)));
%     end
% end
%fclose(fid);

nearestNeighbors = indexes(2:(1+K),:); %index 1 should be the point↙
itself


disp(sprintf('Solving for weights'));

z = zeros(D, K);
W = zeros(K,N);
for i=1:N

   %subtract Xi from every column of Z
   for j=1:K
       z(:, j) = X(:,nearestNeighbors(j,i)) - X(:,i);
   end

   C = z'*z;
   C = C + eye(K,K); %help numerical instabilities
   W(:,i) = inv(C) * ones(K, 1);
   W(:,i) = W(:,i)/sum(W(:,i));
end

disp(sprintf('Computing embedding coordinates using weights'));


M = sparse(1:N,1:N,ones(1,N),N,N,4*K*N); %sparse matrix
for i=1:N
   w = W(:,i);
   j = nearestNeighbors(:,i);
   M(i,j) = M(i,j) - w';
```

```
   M(j,i) = M(j,i) - w;
   M(j,j) = M(j,j) + w*w';
end

%M2 = ((eye(K, N) - W)' * (eye(K, N) - W)) % why doesnt this work?

%options.disp = 0;
%[Y,eigenvals] = eigs(M,N-1);
%eigenvals(N-2-dOutput:N-2, :);
numEigens = dOutput + 4;
[Y, eigenvals] = jdqr(M, numEigens, 0);
eigValues = diag(eigenvals);


eigStrength = eigValues(dOutput+1+1) - eigValues(dOutput+1);
Y = Y(:,2:dOutput+1)';

if plotType == 1
    disp(sprintf('Plotting'));
    figure(1);
    clf;
    % plot(Y(1, :), Y(2, :), 'r+');
    % axis([-1, 1, -1, 1]);

    if dOutput == 3
        for i=1:N
           for j=1:min(5, K)
              line( [Y(1, i), Y(1, nearestNeighbors(j,  i))], [ Y(2, i),↙
Y(2, nearestNeighbors(j, i))], [ Y(3, i), Y(3, nearestNeighbors(j, i))]↙
);
           end
        end

        for i=1:N
           text(Y(1, i), Y(2, i), Y(3, i), labels(i, :), 'EdgeColor',↙
'blue', 'BackgroundColor', 'white' );
        end
        axis([-1, 1, -1, 1, -1, 1]);
    else
       for i=1:N
           for j=1:min(5, K)
              line( [Y(1, i), Y(1, nearestNeighbors(j,  i))], [ Y(2, i),↙
Y(2, nearestNeighbors(j, i))]);
           end
        end

        for i=1:N
           text(Y(1, i), Y(2, i), labels(i, :), 'EdgeColor', 'blue',↙
'BackgroundColor', 'white' );
        end
        axis([-1, 1, -1, 1]);
    end
    title('Map');

    figure(2);
    clf;
```

```
    bar(eigValues(2:dOutput+1+2));
    axis([0, length(eigValues) + 1, 0, max(eigValues) + 0.1]);
    title('Eigenvalues');
end
```

```
%lleDemo
% a script for running lle tests

%set this variable to the path which contains the data folder
dataFolder = '/Users/blake/Desktop/ML-final-project/data/';

%1 -- artists
%2 -- users
%3 -- k test artists
%3 -- k test users
demo = 1;

if demo==1
    load([dataFolder 'UserSongs.data']);
    UserSongs = UserSongs(1:100, :);
    itemLabels = readtextfile([dataFolder 'ItemLabels.txt']);
    [Y, distances] = lle(UserSongs', itemLabels, 20, 2, 1, 2);
elseif demo==2
    load([dataFolder 'UserSongs.data']);
    UserSongs = UserSongs(:, :);
    itemLabels = readtextfile([dataFolder 'UserLabels.txt']);
    [Y, distances] = lle(UserSongs, itemLabels, 20, 2, 1, 1);
elseif demo==3
    load([dataFolder 'UserSongs.data']);
    UserSongs = UserSongs(1:100, :);
    itemLabels = readtextfile([dataFolder 'ItemLabels.txt']);

    eStrengths = [];
    for i=2:99
        [Y, distances, eigStrength] = lle(UserSongs', itemLabels, i, 2,↙
1, 2);
        eStrengths = [eStrengths eigStrength]

        figure(3);
        title('EigenStrength vs. K');
        clf;
        bar(eStrengths);
    end
elseif demo==4
    load([dataFolder 'UserSongs.data']);
    UserSongs = UserSongs(:, :);
    itemLabels = readtextfile([dataFolder 'UserLabels.txt']);

    eStrengths = [];
    for i=2:55
        [Y, distances, eigStrength] = lle(UserSongs, itemLabels, i, 2, 1,↙
2);
        eStrengths = [eStrengths eigStrength]

        figure(3);
        clf;
        bar(eStrengths);
        title('EigenStrength vs. K');
    end

end
```