Feedback sheet: Assignment

This feedback sheet is meant to help you to improve your assignment reports, and to understand the grading criteria.

Part of the work of an applied mathematician is to communicate with experts that may not be mathematicians. Therefore, it is important to develop one's reporting skills. In this course, the quality of the reporting is an important element in grading.

Below, a list of important points concerning the reporting are given, and it also serves as a rubric for feedback.

1. The report should be self-explanatory, not an answer to a question. This means that you have to explain what the problem is that you are addressing. Preferably, do not just copy the assignment but explain in your own words what the assignment asks you to do. **A printed code is not a sufficient answer!**

   Your grade:

   ☐ Excellent: No need to improve
   ☐ Good: Acceptable, leaving space for improvement
   ☐ Below standard : Requires more work

2. The report should be carefully typeset, paying attention to the layout. Sloppy reporting gives a negative impression and raises questions about the correctness of the results, so one needs to pay attention on the visual quality.

   Your grade:

   ☐ Excellent: No need to improve
   ☐ Good: Acceptable, leaving space for improvement
   ☐ Below standard : Requires more work

3. Well planned graphs and plots are an integral part of good reporting. When you include a plot in the report, pay attention to the following:

   (a) Make sure that the fonts are of readable size. The default font in Matlab is not acceptable, so increase the font size, e.g., using the command `set(gca,'FontSize',15)`.

   (b) Lines and curves need to be thick enough to be readable, and markers must be large enough.

   (c) Pay attention to axes. For instance, if you plot a circle, make sure that it does not appear as an ellipse, using commands like `axis('square')` or `axis('equal')`. Also, think about the scaling of the axis so that they convey the information you want. Sometimes a logarithmic scale is more informative.

   (d) Select the graphical output carefully. Is a histogram better than a continuous curve? Discrete quantities may be better represented by dots or bars than by continuous curves etc.

(e) Consider marking the axes with a label, using commands `xlabel, ylabel`.

Your grade:

- ☐ Excellent: No need to improve
- ☐ Good: Acceptable, leaving space for improvement
- ☐ Below standard : Requires more work

4. Technical correctness. Each assignment will be checked for correctness. If you get a grade less than excellent, go and check the comments on the report.

   Your grade:

   - ☐ Excellent: No need to improve
   - ☐ Good: Acceptable, leaving space for improvement
   - ☐ Below standard : Requires more work

5. Code: Including a code, or a piece of it is recommended when applicable. Pay attention on your code: A code with no comments is below standard. Well documented code is a great asset for yourself, too!

   Your grade:

   - ☐ Excellent: No need to improve
   - ☐ Good: Acceptable, leaving space for improvement
   - ☐ Below standard : Requires more work

Assignment 3
MATH 444: Mathematics of Data Science

David Frost

Problem 1: Linear Discriminant Analysis

Linear discriminant analysis (LDA) is used to reduce dimensionality of data for visualization purposes as well as a heuristic to test the effectiveness of other dimensionality reduction tools or clustering algorithms. To use LDA, we must have annotated data, which means that we must have a vector that lists the true clustering or classification of each data vector. Therefore, unlike a method such as k-means clustering, we must actually know a lot about the data without trying to make our own guesses.

LDA works by first splitting the data matrix into k matrices, one for each cluster, and then finding the average data vector of each cluster and finally the average data vector of the overall data set matrix. Scatter matrices have to be calculated for each cluster matrix. A within-cluster and between-cluster scatter matrix must also be calculated. After finding the largest eigenvalue of the within-cluster scatter matrix, we then scale that eigenvalue by a tiny coefficient τ and the result is called ϵ. Afterwards, find the Cholesky factorization of $S_w + \epsilon I_n = K^T K$. We then want to find the k - 1 largest eigenvalues and corresponding eigenvectors $w^{(j)}$ of the matrix $A = K^{-T} S_b K^{-1}$ where S_b is the between-cluster scatter matrix where j ranged from 1 to k - 1. We then need to solve the linear system $Kq^{(j)} = w^{(j)}$ for j from 1 to k - 1. We can then form a matrix Q from $[q^{(1)}q^{(2)}...q^{(k-1)}]$. Each LDA-reduced matrix $Z_j = Q^T X_j$ for j from 1 to k. The description of this algorithm is based on Professor Daniela Calvetti and Professor Erkki Somersalo's book *Mathematics of Data Science: A Computational Approach to Clustering and Classification.*

The summary of the purpose of LDA is to find an ideal subspace which maximizes the distance of the data vectors. For example, if we have 13-dimensional data that we want to visualize in a 2-dimensional scatter plot, then we want to find a 2-dimensional subspace that maximizes the distance between all of the data vectors. In other words, we want to find the direction that maximizes spread.

There is an Italian wine data set with dimensions 13 x 187. We have data about 13 different chemical properties for 187 types of wine. This data is accompanied with an annotation vector that tells use which cultivar truly the wine truly belongs to. All of these wines come from the same region of Italy. Other algorithms frequently have trouble trying to classify or cluster the wine based on cultivar, so trying to use LDA to separate the different wines based on cultivar is a good experiment to test LDA's effectiveness.

After running linear discriminant analysis on the wine data, we can see that it performs extraordinarily well. In the past, I used k-medoids and k-means algorithms to try and cluster this same wine data set and ran into issues where it would keep misclassifying some of the data vectors. LDA, however, separates everything very well; there is still some overlap if you imagine reducing it into one dimension, but on the two-dimensional scatter plot, all of the clusters are very distinct and it only "misclassifies" a few vectors by putting them a bit into the wrong pile. Therefore, LDA is an excellent way to visualize the wine data in two dimensions and avoids some of the pitfalls associated with PCA, particularly putting some wines into the ”wrong piles”.

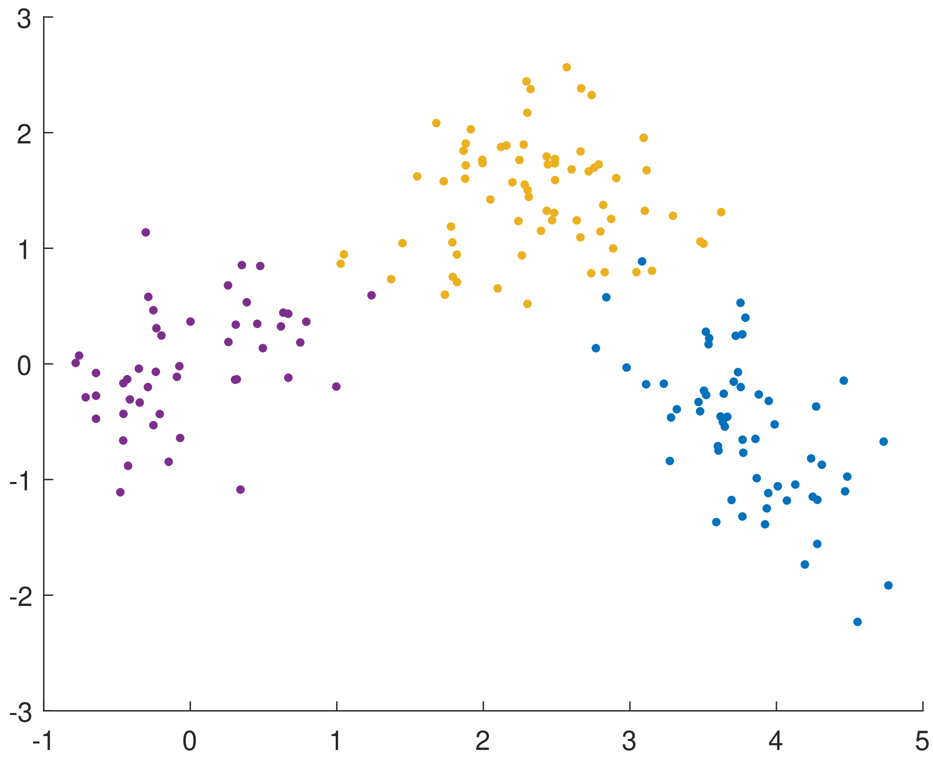

Figure 1: The result of plotting the two principal LDA directions of the wine data set.

Problem 2: Self-organizing maps

Self-organizing maps (SOM) are used to reduce dimensionality of data while still preserving topology.

The self-organizing maps algorithm returns two matrices: the 2D lattice of size N x N and the matrix of prototypes of size n x K, where n (lowercase n) refers to the dimensionality of the data matrix and K is the number of prototypes. Therefore there is one column for each prototype, and each prototype has as many elements as each data vector.

There are several steps to the SOM algorithm. First, the 2D lattice must be created. Second, the prototypes must be initialized. There are many different ways to initialize the prototypes and this step depends significantly on what data set is being used as input. One method that frequently works well is data vectors that are based on the mean of the data vectors but with some randomness added. Step 3 is keep iterating in a loop until we reach T_{max}, which is the maximum number of iterations. This loop involves several steps. The first is to pick a data vector from the data set uniformly at random and then to find the best matching unit (BMU) among the prototypes. Afterwards, every prototype must be updated. The degree to which each prototype is changed depends on several factors. One is the distance in the 2D lattice from the BMU. Another are the parameters used for learning. These are used to determine how much each prototype should change. Additonally, we have different parameters depending on whether the algorithm is still in the learning phase or done with that phase. $\alpha_0, \alpha_1, \gamma_0$, and γ_1 are the parameters used for this. I used the recommended parameters from the book *Mathematics of Data Science: A Computational Approach to Clustering and Classification* by Professor Daniela Calvetti and Professor Erkki Somersalo. These are $\alpha_0 = 0.9, \alpha_1 = 0.01, \gamma_0 = K/3$, and $\gamma_1 = 0.01$. α_0 and γ_0 are used in the learning phase to determine how much the prototypes should be updated in every iteration. Note that these are much greater than α_1 and γ_1; this is because the prototypes are changing very quickly during the learning phase and change very slowly afterwards. After T_0 iterations of the loop, the learning phase ends and we use α_1 and γ_1 to determine how quickly prototypes should update. $T_0 \geq 1000$ and $T_{max} \geq 1000 * K$ are also recommended by Professors Calvetti and Somersalo.

The full code of the SOM algorithm is attached in the appendix.

Problem 3: Handwritten Digits Data

There is a data set of handwritten digits that has dimensions 256 x 1707. These digits represent the integers from 0 to 9. Each data vector has 256 elements which actually represents a 16 X 16 image after being reshaped. We have 1707 data vectors, each of which is an image of a handwritten digit. Most of the digits are clearly legible and can be distinguished as one particular number, but there are some edge cases. Additionally, we want to analyze the topological structure of these digits to see which ones are "most" like their types; i.e., if we only look at digits 2, 4, and 7, then we want to know which 4 is most like a 4, in that it is nearly impossible to confuse it for another digit. We restrict ourselves to only a few digits to make the data easier to analyze and to keep the size of the lattice small enough to be able to comprehend easily.

Self-organizing maps (SOM) are a great tool to use to analyze this data set. This is because they can both reduce dimensionality from 256 dimensions to 2 dimensions for imaging purposes and also because they can preserve topological structure of the data. The 2D lattice of a self-organizing map allows us to place prototypes for the digits on a grid, which truly allows us to see the "distance" between one data vector and another. For example, if we try analyzing the digits 2, 4, and 7 using a self-organizing map with a lattice size of 10 X 10, we would expect that a 4 surrounded only by other 4s is unambiguously a 4. However, a 4 that is surrounded by 2s, 4s, and 7s could possibly be reasonably mistaken for either a 2 or 7. The same applies to 2s and 7s.

After running the SOM algorithm, we then find the closest data vector (one image of a digit) to each prototype in the lattice and then show the 16 x 16 image of that data vector at that place in the lattice. Since we have one hundred 16 X 16 images, the final image of these data vectors overlaid on the lattice has dimensions 160 X 160.

We can see that SOM work very well for this data set. 2s, 4s, and 7s are all kept together, with only a couple noticeable errors. We conclude that SOM are an excellent way to reduce the dimensionality of the handwritten digits data set while also preserving the topological structure, as the algorithm was able to keep all of the relevant digits together without significantly mixing them into the other clusters.

We conclude that the k-means and k-medoids algorithms did not do that well in clustering the wines. By looking at the plots of the actual classifications of the wine data and then the attempted clustering by the k-means and k-medoids algorithms, we can see that the two clusterings aren't that close to what the true clustering is. This isn't really the fault of the algorithms, however; looking at the true classification of the data, we can see that the yellow and light blue cultivars have significant overlap, so unsupervised clustering inherently is weak for this type of problem.

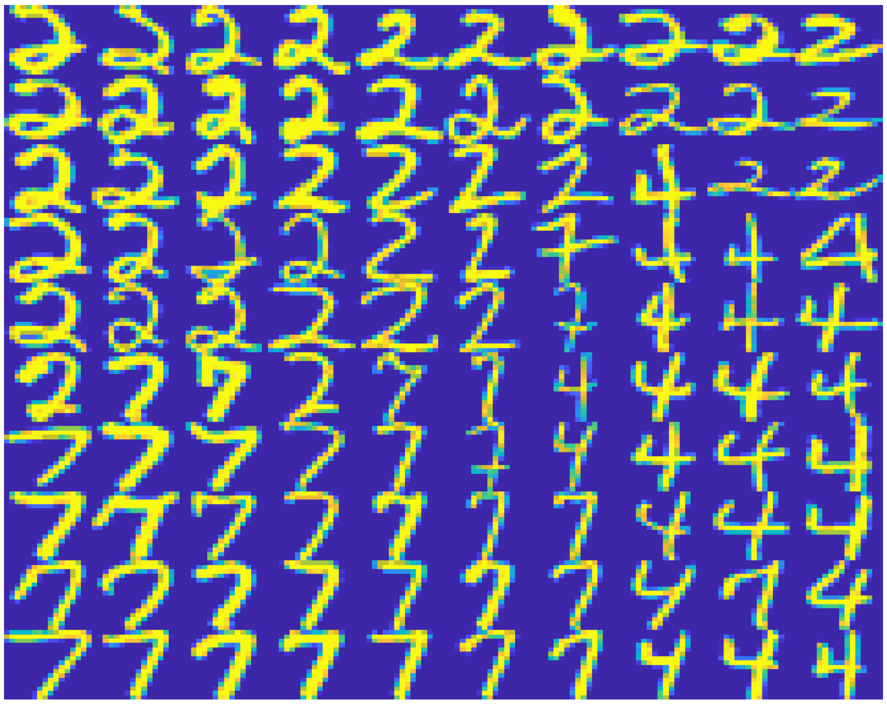

Figure 2: The result of using self-organizing maps on the handwritten digits dataset restricted to just 2s, 4s, and 7s. The topological structure is preserved. At each point in that lattice is the image of the data vector closest to the respective prototype.

Appendix

Code for Problem 1: Linear discriminant analysis

```
load WineData
% dimensions of data
n = size(X, 1);
p = size(X, 2);
tau = 10^(-10);
Xbar = 1/p * sum(X, 2);
X_c = X - Xbar;
% there are three clusters in the data
% the three matrices, one for each cluster
X1 = X(:, find(I == 1));
X2 = X(:, find(I == 2));
X3 = X(:, find(I == 3));
% the centroids of each cluster
C1 = 1/size(X1, 2) * sum(X1, 2);
C2 = 1/size(X2, 2) * sum(X2, 2);
C3 = 1/size(X3, 3) * sum(X3, 2);
```

```
% global centroid
C = 1/size(X, 2) * sum(X, 2);
% center the clusters
X1 = X1 - C1*ones(1, size(X1, 2));
X2 = X2 - C2*ones(1, size(X2, 2));
X3 = X3 - C3*ones(1, size(X3, 2));
% within-cluster centered matrix
X_w = [X1 X2 X3];
% within-cluster scatter matrix
S_w = X_w * X_w';
% initialize matrices
X1_bar = [];
X2_bar = [];
X3_bar = [];
% 3 for loops
for a = 1:size(X1, 2)
    X1_bar = [X1_bar, C1];
end
for a = 1:size(X2, 2)
    X2_bar = [X2_bar, C2];
end
for a = 1:size(X3, 2)
    X3_bar = [X3_bar, C3];
end

X_bar = [X1_bar X2_bar X3_bar];
X_bar_centered = X_bar - C*ones(1,p);

S_b = X_bar_centered * X_bar_centered';
% largest eigenvalue of within-cluster scatter matrix
largest_eigenvalue = eigs(S_w, 1);
epsilon = tau * largest_eigenvalue;
% cholesky
K = chol(S_w + (epsilon * eye(n)));
K_inverse = inv(K);
K_inverse_transpose = K_inverse';
A = K_inverse_transpose * S_b * K_inverse;
two_eigs_of_A = eigs(A, 2);
[W, two_eigenvalues] = eigs(A, 2);
Q = K \ W;
for a = 1:2
    Q(:,a) = Q(:,a)/norm(Q(:,a));
end
Z_1 = Q' * X1;
Z_2 = Q' * X2;
Z_3 = Q' * X3;
% Plot the two principal LDA directions
Z = Q' * X;
colormap("lines");
figure(1);
scatter(Z(1,:),Z(2,:),10,I,'filled');
```

Code for Problem 2: Self-organizing maps

```
function [M,Q] = my_som(K,T_max, data, init_func)
% function my_som uses self-organizing maps to reduce
% dimensionality in data while still preserving topology and order.
% Input: K = number of prototypes. Should be a perfect square
% since the lattice should be a square
% Input: T_max: Number of iterations
% Input: data: the data matrix
% Input: init_func: the initialization function
% Output: M: the prototypes. Size is p x K
% Output: Q: the 2D lattice. Size is 2 x K

% n = number of elements in each data vector
n = size(data, 1);
% p = number of data vectors
p = size(data, 2);
% Parameters for learning for SOM
alpha_zero = 0.9;
alpha_one = 0.01;
gamma_zero = K/3;
gamma_one = 0.01;
% initial counter
t = 0;
% length of learning phase
T_zero = 10000;
% Q is a 2 x K matrix. Theoretically it should be N x N where N = sqrt(K),
% but it's more convenient to represent the 2D lattice as a 1 x K vector
% but where we store i and j values for each position in the first and
% second rows.
Q = ones(2, K);
% row for i coordinates
Q_top = [];
for i = 1:sqrt(K)
    Q_top = [Q_top, i*ones(1,sqrt(K))];
end
% row for j coordinates
Q_bottom = [];
for i = 1:sqrt(K)
    for j = 1:sqrt(K)
        Q_bottom = [Q_bottom, j];
    end
end

% Create lattice by combining
Q = [Q_top; Q_bottom];

% Initialize the prototypes
% We use init_func to make this program
% more modular so we can use whatever initialization
% is given as an input.
M = init_func(n, K, data);
% iteration
```

```
while t < T_max
    %find best matching unit from current prototypes
    % initialize distance from BMU to vector as Inf and index as 1
    vector_to_BMU_distance = Inf;
    j = 1;
    % pick a random data vector
    data_vector = data(:,randi(p));
    % check every prototype's distance to vector to find BMU
    for a = 1:K
        vector_and_prototype_distance = norm(M(:,a) - data_vector);
        % we have found a better matching unit
        if vector_and_prototype_distance < vector_to_BMU_distance
            vector_to_BMU_distance = vector_and_prototype_distance;
            % index of BMU
            j = a;
        end
    end
    % update every prototype now that we found a new BMU for that data
    % vector
    for ell = 1:K
        % calculating alpha(t) (learning rate)
        alpha_of_t = alpha_one;
        if alpha_zero*(1 - (t/T_zero)) > alpha_of_t
            alpha_of_t = alpha_zero*(1 - (t/T_zero));
        end
        % calculating gamma(t)
        gamma_of_t = gamma_one;
        if gamma_zero*(1 - (t/T_zero)) > gamma_of_t
            gamma_of_t =  gamma_zero*(1 - (t/T_zero));
        end
        % distance between particular prototype and BMU
        d_ell_j = norm(Q(:, ell) - Q(:, j));
        neighborhood = exp(( (-1/2) * (1/(gamma_of_t) * (1/gamma_of_t) )) * d_ell_j * d_ell_j);
        % updating prototype vectors
        M(:,ell) = M(:,ell) + alpha_of_t*neighborhood*(data_vector - M(:,ell));
    end
    % update counter
    t = t + 1;
end
```

Code for Problem 3: Handwritten Digits

```
load HandwrittenDigits.mat X I;
% number of iterations for SOM
T_max = 100000;
% number of prototypes. Should be perfect square since we're using
% a 2D lattice.
K = 100;
% finding the digits we want
I1 = find(I == 7);
I2 = find(I == 2);
I3 = find(I == 4);
```

```
% filtering the data to only include those digits
data = X(:, [I1 I2 I3]);
% dimensions of data. useful to prevent usage of magic numbers later
n = size(data, 1);
p = size(data, 2);
% run SOM
[M, Q] = my_som(K, T_max, data, @initialize_prototypes);
% for each prototype we run this loop
for a = 1:K
    closest_data_point = [];
    closest_data_point_distance = Inf;
    % find the closest data vector to this prototype
    for ell = 1:p
        if norm(M(:,a) - data(:,ell)) < closest_data_point_distance
            closest_data_point_distance = norm(M(:,a) - data(:,ell));
            closest_data_point = ell;
        end
    end
    % find coordinates on lattice of prototype
    j = Q(1, a);
    k = Q(2, a);
    cdp = closest_data_point;
    % plot the closest data vector onto the prototype's place in the
    % lattice
    V((j - 1)*16 + 1: j * 16, (k - 1) * 16 + 1: k * 16) = reshape(data(:, cdp), 16, 16)';

end
% show the 100 prototypes
colormap("gray");
imagesc(V);
axis("off");
```

Code for initialize_prototypes function used in SOM

```
function M = initialize_prototypes(n, K, data)
% function that is used to initialize prototypes for SOM algorithm
% Input: n = dimensionality of data
% Input: K = number of prototypes
% Input: data = data set matrix
% Output: M = initialized prototypes
M = ones(n, K);
% computes the mean of the data vectors in the matrix and adds some uniform
% randomness to generate each data vector. The odds of two vectors being
% identical is astronomically low
for a = 1:100
    meanvalue = (ones(256, 256) - rand(256,256) + rand(256,256)) * 1/size(data, 2) * sum(data, 2);
    M(:, a) = meanvalue;
end
end
```