**Home Assignment 2 8850 Machine Learning**

Problem. You are given a data set consisting of DNA sequences (the file is available here) of the same length. Each DNA sequence is a string of characters from the alphabet ‘A’,’C’,’T’,’G’. The sequences are feature vectors and characters are features. The data set is stored as a fasta file, which is essentially a text file which stores the data in the following form:

>Sequence1 AAGCACAGGATGTAATGGTGGGGCCGACCGCCTATTATTCTG >Sequence2 AAGCACAGGATGTAATGGTGGGGCCGACCGCCTATTATTCTG >Sequence3 AAGCACAGGATGTAATGGTGGGGCCGACCGCCTATTATTCTG  
etc.  
Each line starting with ‘>’ symbol contains the name of a sequence followed by

the sequence itself in the next line.  
I) Propose how to convert feature vectors of characters into feature vectors of

numbers. Justify your approach. Write a script which converts DNA sequences from our dataset into numerical format.

🡺 I have converted each character into its ASCII value as it is easier to use further. So, by doing this, my sequence got converted to list of ASCII values. For E.g. A got changed to 65 and B got changed to 66. And ASCII value is considered better for machines. As machine can easily identify ASCII value, therefore I am using ASCII values.

Code Snippet: -

**mylist = list()**

**for record in SeqIO.parse("HW2.fas", "fasta"):**

**mylist.append(record.seq)**

**print(record.seq)**

**myarray = np.asarray(mylist)**

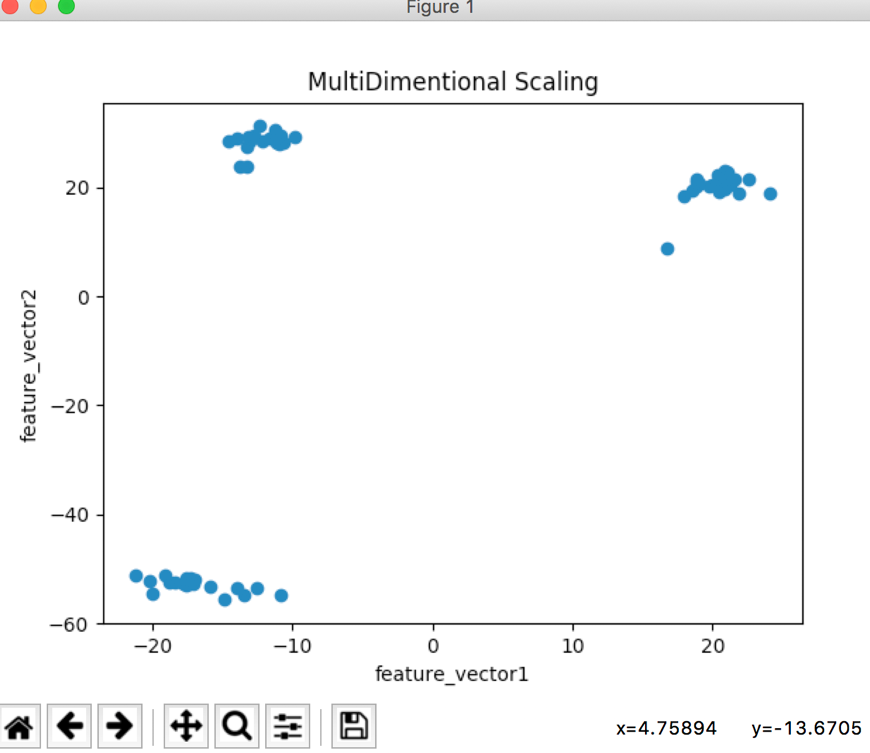
**# define a matrix**

**X\_std = myarray.view(np.uint8)**

II) Build a visualization of the DNA data using two methods:

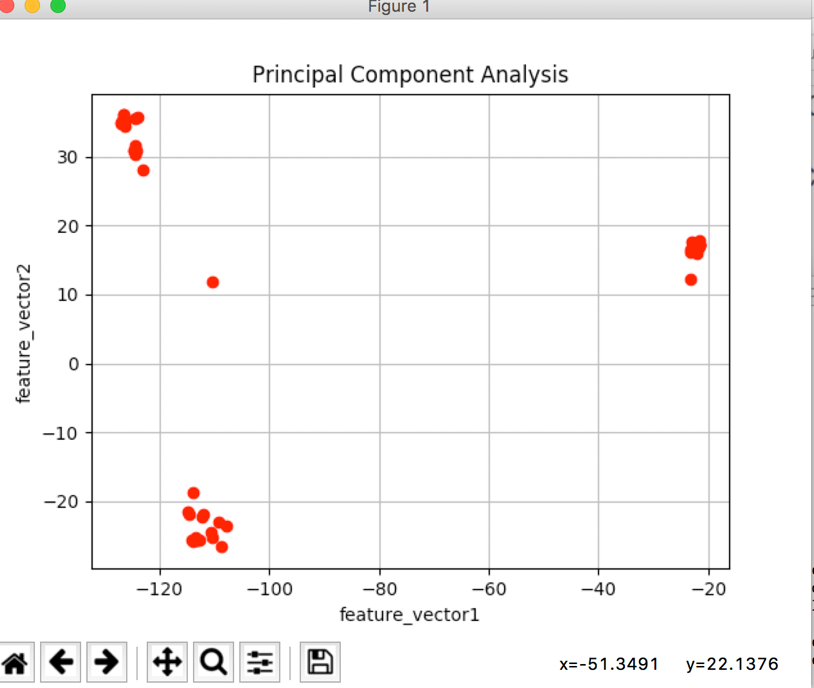
1. Multidimensional scaling with two dimensions. An input distance matrix should be the matrix of pairwise Hamming distances between sequences.

🡺



1. Principal component analysis. Reduce the numerical data matrix built in I) to two dimensions and plot sequences using two obtained features as coordinates.

🡺



Which method do you think produced the better result? Please, explain your answer.

🡺 Multidimensional scaling (MDS) and principal component analysis (PCA) are both dimensionality reduction techniques, but they optimize for different things. Also, PCA minimizes dimensions, preserving covariance of data. MDS minimizes dimensions, preserving distance between data points. Both are same, if covariance in data is equal to Euclidean distance between data points in high dimension.  
And both MDS and PCA are different, if distance measure is different.

I feel MDS produced better results as the visualization is better. Proper clusters are formed in the graphs. And properly data is extracted in less no. of steps.

You may use library functions to read data from the file and perform multidi- mensional scaling. For principal component analysis, you are not allowed to use library functions but you may use them to calculate eigenvalues and eigenvectors. Your submission should contain:

􏰄 The code of your scripts/programs;  
􏰄 Two visualization plots  
􏰄 Pdf or word file with your answers to the questions.

**Code: -**

**#!/usr/bin/python**

**from \_\_future\_\_ import division**

**import numpy as np**

**#import Bio**

**from Bio import SeqIO**

**from numpy import array**

**from numpy import mean**

**from numpy import cov**

**import numpy.linalg as linalg**

**from matplotlib import pyplot as plt**

**from sklearn.manifold import MDS**

**#Loading file and converting sequence into ASCII**

**mylist = list()**

**for record in SeqIO.parse("HW2.fas", "fasta"):**

**mylist.append(record.seq)**

**print(record.seq)**

**myarray = np.asarray(mylist)**

**print(myarray.shape)**

**# define a matrix**

**X\_Matrix = myarray.view(np.uint8)**

**print(X\_Matrix.shape)**

**#Principal Component Analysis**

**mean\_vector = np.mean(X\_Matrix, axis=0)**

**cov\_matrix = (X\_Matrix - mean\_vector).T.dot((X\_Matrix - mean\_vector)) / (X\_Matrix.shape[0]-1)**

**print('Covariance matrix \n%s' %cov\_matrix)**

**eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)**

**print('Eigenvectors \n%s' %eigenvectors)**

**print('\nEigenvalues \n%s' %eigenvalues)**

**for ev in eigenvectors.T:**

**np.testing.assert\_array\_almost\_equal(1.0, np.linalg.norm(ev))**

**#Listing (eigenvalue, eigenvector) tuples**

**eigenpairs = [(np.abs(eigenvalues[i]), eigenvectors[:,i]) for i in range(len(eigenvalues))]**

**# Sorting Eigen values and Eigen Vectors**

**eigenpairs.sort(key=lambda x: x[0], reverse=True)**

**print('Eigenvalues in descending order:')**

**for i in eigenpairs:**

**print(i[0])**

**matrix\_w = eigenvectors[:,:2]**

**print(matrix\_w.shape)**

**print('Matrix W:\n', matrix\_w)**

**#Final Matrix**

**Y = X\_Matrix.dot(matrix\_w)**

**print(Y.shape)**

**feature\_vector1 = Y[:,0]**

**feature\_vector2 = Y[:,1]**

**print(feature\_vector1)**

**print(feature\_vector1.shape)**

**print(feature\_vector2)**

**print(feature\_vector2.shape)**

**plt.xlabel('feature\_vector1')**

**plt.ylabel('feature\_vector2')**

**plt.title('Principal Component Analysis')**

**plt.grid('True')**

**plt.plot(feature\_vector1,feature\_vector2,'ro')**

**plt.show()**

**#Multidimentional Scaling**

**def findhamdist(str1, str2):**

**diffs = 0**

**for k in xrange(len(str1)):**

**if str1[k] != str2[k]:**

**diffs += 1**

**return diffs**

**rowscolumn = 62**

**MDS\_Matrix = np.zeros((62, 62))**

**for i in xrange(rowscolumn):**

**for j in xrange(rowscolumn):**

**MDS\_Matrix[i][j] = findhamdist(X\_Matrix[i,:], X\_Matrix[j,:])**

**print(MDS\_Matrix)**

**model = MDS(n\_components=2, dissimilarity='precomputed', random\_state=6)**

**out = model.fit\_transform(MDS\_Matrix)**

**print(out)**

**plt.title('MultiDimentional Scaling')**

**plt.xlabel('feature\_vector1')**

**plt.ylabel('feature\_vector2')**

**plt.scatter(out[:, 0], out[:, 1])**

**plt.show()**