Introduction to Machine Learning (CS 135)
Assignment 01 (25 points)
Due on Gradescope by 11:59 PM, Monday, 19 September

This assignment will get you started using NumPy. If you are new to using that library (or Python in general), you probably want to start with the tutorial provided at:

https://numpy.org/devdocs/user/quickstart.html

You have been supplied with some starter code, including the prototype for two functions that you need to complete. The code also contains detailed documentation of how the functions should operate, and some examples of their use. You can (and should) test your code to confirm that it does what is asked.

1. (*5 pts.*) Along with your code, submit a completed version of the `COLLABORATORS.txt` file. An example has been provided, which you should edit appropriately to include:

   - Your name.
   - The time it took you to compete the assignment.
   - Any resources you used to complete the assignment, including discussions with the instructor, TA's, or fellow students, and any online or offline resources consulted. If you did not need to consult any outside resources, you can say so.
   - A brief description of what parts, if any, of the assignment caused you to seek help.

2. (*10 pts.*) The first part of the code asks you to write a function to split data into a training set and a testing set (a very common task in ML). Your code will use basic NumPy array indexing and random number generation to take an input array containing L instances of F-dimensional data, and divide it into two mutually exclusive arrays of size M (for training) and N (for testing).

   As part of its input, the function takes parameter `frac_test`, specifying the overall fraction of the data-set to use for testing purposes. It will use this fraction to determine the size N, **rounding up** to the nearest whole number: $N = \lceil \texttt{frac\_test} * L \rceil$

   The function will also use NumPy functions like `shuffle` or `permutation` for doing random sampling of the data, so that the test/train instances are uniformly selected from the data-set:

   https://numpy.org/doc/stable/reference/random/legacy.html

   (Note that this links to a library that has been generally replaced by something a bit newer in the latest version of NumPy, but is still supported. Your code will function correctly if you use the legacy version, or the newer NumPy features.) Furthermore, we want the results of the function to be *reproducible*, for scientific purposes. That means that there should be a way to specify a source of randomness (a ***seed***) such that it is possible to duplicate any random selection. In NumPy, this is generally handled by using a `RandomState` instance, or via an integer seed. See the linked discussion from the source code comments for insight into using such seeds in your code.

   Notes: read the documentation of the function supplied with the starter code carefully. Ensure that your code meets the requirements as given. In addition, be sure that your code

uses ***only*** basic Python and functions from NumPy. ***Do not*** call functions for data-set generation and manipulation from libraries like `sklearn`.

3. (*10 pts.*) The other function you are to complete finds *nearest neighbors* of given data-instances. That is, given a set of F-dimensional data of size N, and Q query instances (also F-dimensional), we want to compute the K closest vectors found in the data, for some integer value K, and for each query instance (for a total of $Q \times K$ neighbors).

   In computing "closeness," we will use the *Euclidean distance* between vectors. For two vectors $\mathbf{x}^1 = (x_1^1, x_2^1, \ldots, x_F^1)$ and $\mathbf{x}^2 = (x_1^2, x_2^2, \ldots, x_F^2)$, this distance is given by:

$$\delta_E(x^1, x^2) = \sqrt{\sum_{i=1}^{F} (x_i^1 - x_i^2)^2}$$

   Your function will take in a data-set (a 2-dimensional array of size $N \times F$) and a query-set (a 2-dimensional array of size $Q \times F$), and return a 3-dimensional array (of size $Q \times K \times F$), where each row (indexed by Q) consists of the K nearest neighbors of the corresponding query vector. These neighbors should appear ***in order***, closest to least-close.

   Notes: it is possible that there will be ties among neighbors. If this occurs, such ties can be broken however you like (randomly or not). Again, be sure that your code uses ***only*** basic Python and functions from NumPy. ***Do not*** call functions from libraries like `sklearn`.

Submission details: Your work should satisfy the following requirements:

- All elements of your submission should be uploaded in a single upload, not one after the other. The easiest way to do this is really just to drag and drop all the files into the Gradescope submission window and submit once. (If you use a ZIP archive, make sure it contains no folder structure, just the requisite files.)
- Your code must work properly with the versions of Python and other libraries that are part of the 135 standard environment. The class website contains instructions for installing and using this environment. (On Canvas, you can find this in the **Modules** list, under **Start Here: Install Python**.) You can write the Python code using whatever tools you like, but you should ensure that the code executes properly. (See next page for an example of how you might test your code.)
- When you submit your code, only submit the completed functions. Your own testing code should be commented out or otherwise removed.
- All materials for this assignment should be submitted as part of a single ZIP archive. Submit your work to the Gradescope site for the course:

  https://www.gradescope.com/courses/438698

- After submission, the Gradescope site will run an auto-grader utility and give you a numerical score on the coding part of the assignment.* If there are errors in any tests, these will be indicated. It is recommended that you submit your work well before the deadline in order to help you determine if there are bugs or missing components that need to be corrected.

*__Note__: the text-file with collaboration information will be read by human eyes and graded later on, so you won't receive that part of the grade right away.

Testing your code: There are a variety of ways of testing your code. Here, we describe one. You may want to test your code in some other way; this is fine, but you should test it in some way.

- Start by activating the COMP 135 environment, from the directory containing your `hw1.py` file. Once the environment is active, you can load your file into the Python interactive shell (doing this makes the functions written in that file available from the shell):

```
$python -i hw1.py
```

- After you have loaded it, you can call functions from your file (and from NumPy, since `hw1.py` starts by importing that library as `np`). For instance, you can run the first few lines of testing code described in the documentation for the first function as follows, to generate an identity matrix of size (10×10), and then split it into a (3×10) testing set and a (7×10) training set:

```
>>> x_LF = np.eye(10)
>>> train_MF, test_NF = split_into_train_and_test(x_LF, frac_test=0.3,
                                  random_state=np.random.RandomState(0))
```

- The rest of the testing code can also be run in this fashion, to test the correct behavior of the function. Similar tests can be run on the second function you wrote as well.

- Again, the code you submit should only contain the two functions asked for (and any supporting functions you write to implement them). Your submission ***should not*** include any testing code itself.