Coding Assignment 1

Due Date: *Monday, September 8 (11.59PM)*

Submission: on Gradescope

Instructions:

The coding assignments may be completed in groups up to 4 students. If you do so, please make sure that you include everyone's full name, and that you *also select everyone's name when submitting the assignment on Gradescope*. This ensures that each group member will get a grade assigned and have access to the comments from the graders.

For this assignment, the following items need to be submitted:
(1) the **code** in R or *Python* (a Markdown file is ok)
(2) a **pdf** file with your code and results (Markdown-style) with all necessary plots and comments.

Problem 1: *[50 points]*

In this coding assignment, we are going to use the **UCI Pen-Based Recognition of Handwritten Digits Data set** which can be found and **downloada** from the *UCI Data Repository*: https://doi.org/10.24432/C5MG6K. You can also read the details of the data collection process.

This data set cointains digit trajectories, collected via a pen-tablet, using sequences of (x, y) coordinates producing **16** integer-valued **features** *per sample* (8 points $\times$ 2 coordinates) in the range $0 - 100$. There are $10,992$ samples.

- If you use Python, you can directly download the data from the UCI website above.
- If you use R, you can find the data set in the `PPCI` package - it is called `pendigits`.
- You can also download the provided files from the repository above directly.

For the questions below, please use the provided *training* (`pendigits.tra`) and *testing* (`pendigits.tes`) data sets.

To simplify the classification task, we will **only** consider the digits that correspond to the last two digits of your UIN – e.g. if your UIN is XXXXX56 then you will work with 5's and 6's; if your UIN ends in 766 then will work with 6's and 7's. If you work in a group, feel free to use only one member's UIN.

Our goal is to compare the performance of a **simple linear regression** and a **k-nearest neighbor** classification approach on the data. Use a naive approach for the linear regression in which you treat the response as continuous which you then convert to a binary classifier[1]. For the kNN approach, use $k = 1 : 20$.

(a) *[35 points]* Plot the two errors against the corresponding k values. Make sure that you annotate the plots (e.g. using different colors or types of line) and add legends as needed.

(b) *[15 points]* Does the plot match (approximately) your intuition of the bias-variance trade-off in terms of having a U-shaped error? What is the optimal k value based on this result? For the optimal k, what are the corresponding degrees-of-freedom and its error?

Problem 2: *[50 points]*

The goal in this problem is to write your very own kNN function!

(a) *[5 points]* Generate **4** *independent* standard Normal variables X_1, X_2, X_3, X_4 of $n = 1000$ independent observations. You can then generate a response Y as follows:

$$Y = X_1 + 2 \cdot X_2 - X_3 + \varepsilon$$

with IID $N(0, 1)$ errors ε. Set the *random seed to 598* for reproducibility.

(b) *[10 points]* Use the appropriate kNN function in R or Python and report the mean squared error (MSE) for your prediction with $k = 4$. Use the first 500 observations as the training data and the rest as testing data. Predict the response using the built-in kNN function with $k = 5$.

(c) *[35 points]* For this question, you cannot load any additional packages. Write your own kNN function, `mykNN (xtrain, ytrain, xtest, k)`, that fits a kNN model and predicts multiple target points `xtest`. The function should return a variable `ytest`.

Notes:

- `xtrain` is the training data set feature
- `ytrain` is the training data set response

[1]Logistic or other types of regression will not be given credit at this stage. The goal of this coding assignment is to get started with simpler tasks before diving into more sophisticated algorithms and methods

- `xtest` is the testing data set feature
- `ytest` is the testing data set prediction

You can use the Euclidean distance to calculate the closeness between two points.
Test your code by reporting the means square error on the testing data. It should be close to the one you computed in (b).

Recall that the MSE is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$