**Foundations of Artificial Intelligence**

**Lab Assignment 1**

Gardyan Priangga Akbar

7695883

I created the K-nearest neighbor algorithm by utilizing the Python libraries Pandas and Matplotlib. Pandas is used to read the dataset and create the training and test datasets. Matplotlib is used to provide visualization of how the algorithm performs using various values for k.

To create the best possible model, I needed to make sure the entries used for the training and test datasets are randomized from the source and that the amount of data for all classes needs to be balanced. This is achieved by shuffling the dataset with a particular seed (in which I chose to use the value of 69 for the seed). Pandas provides this functionality with its *sample()* function. After that, I spliced the DataFrame into 3 different DataFrames, with each only containing all samples pertaining to one particular class label (i.e one DataFrame for Iris-setosa, one for Iris-versicolor, and another one for Iris-virginica). Because every class has the same amount of samples (all 50) and it was required to do a 50% split for training and test dataset, I concatenated the first 25 samples from each class into the training dataset, and the last 25 from all classes for the test dataset. I do not need to worry about selecting the values at random because this was already done prior.

The next step is to implement the actual K-nearest neighbor algorithm itself. To achieve this, I wrote an algorithm to determine the Euclidean distance between a single data point to all other points. I used the formula of d(p, q)^2 = (p1 - q1)^2 + (p2 - q2)^2 + (p3 - q3)^2 + (p4 - q4)^2 for this algorithm. The resulting values from all data points are sorted as to get the nearest data points. According to the value of k, the first k neighbors are returned. Using this information, a majority vote is used to predict to what class does a sample from the test dataset belong to. For example, if 2 out of the 3 nearest neighbors are Iris-versicolor, then the test sample will be predicted to have that particular class label. To evaluate the suitability of the k-nearest neighbor algorithm, I ran the algorithm against all samples in the test dataset and counted how many times it predicted the class wrongly compared to all the samples.

After evaluating the algorithm using various values of k, I used matplotlib to visualize how the error value changes against different values of k. Based on this, it seems that the error will generally increase alongside k, but there might be a sweet spot where then error is at its lowest with a particular value for k. I also made the observation that despite setting the same seed value for the dataset shuffling, I still get different results for the error value every time I run the Python script. Turns out this is because of a quirk in Python’s *max()* function which was used to select the majority vote when 2 or more classes have the same amount of votes.