COMP219 Assignment 1

- ✓ F1
- ✓ F2
- ✓ F3
- ✓ F4
- ✓ F5

Software Dependencies:

- Python 3.x
- Anaconda and Spyder
- The following Python packages:
  - numpy
  - sklearn
  - math
  - pickle

How to run the program:

Run the file named sourceCode.py. You will be presented with a menu displaying the numbers 1 to 5, representing the required functionalities f1 to f5. The program also considers the additional requirements, in that running f1, f4, and f5 does not call the training functionalities f2 and f3.

To run the saved models, upon the display of the menu, select either 2 or 3 (corresponding to f2 and f3). The indirect and direct models are saved as model2.sav and model1.sav, respectively.

How the functionalities and additional requirements are implemented:

- **F1)**
  - Using sklearn datasets, I loaded the dataset (digits) using load_digits() and split into the training set (80%) and testing set (20%) using sklearn train_test_split. The total number of samples is defined by the length of the datasets images, hence len(digits.images). The number of classes here would be the number of target names, hence len(digits.target_names). To calculate the minimum and maximum values of each feature, while simultaneously counting the number of entries per class, I iterated through the images which is in range 0-10. For each image with index (X), I incremented by 1 as the number of entries for that class incremented by 1. The minimum and maximum values of said feature (X) is calculated and appended to the list minMax. To display this to the user, a print statement was used to for each of the variables that were desired.

- **F2)**
  - To successfully call library functions to train and save a model, I used sklearn.neighbors to import KNeighborsClassifier. Firstly, I created a new routine called getK, where the value for k in knn is calculated using the formula √totalDataEntires. If the answer is even, add 1, as an odd value of k is better for classifications. KNeighborsClassifier is then used to create a model that can later be called. This object is saved using pickle using .dump as “model1.sav”.

- **F3)**
  - The training data, XTrain and yTrain, are transformed to be used in my own implementation of knn. I first created a method called getDistance, where the transformed X and y training data, as well as the testing data, are passed as parameters. For every piece of test data, it is compared to every piece of train data, each of these only being the x value, not the label. For each value of test data and training data, the distance is $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$

calculated using another method implementation named jaccardIndex. The Jaccard Index is a statistic using in understanding similarities between sets, therefore making it easy to calculate the difference/distance between each training to the test points. Once all distances have been calculated for the one piece of test data, of which produces a list of distances, [distance, expected label, training data]. The list is then sorted from highest to lowest, as a jaccard index closer to 1 is more identical. The list is then passed into another method named getClass where the classification of said training data piece is calculated. getClass receives 4 parameters: distances – the list of distances from test data to each training data; realLabel – the expected label for the test data, totalPredictions – each prediction of each test data; totalTrueLabels – actual label for each test data. The method, getK, is used again to calculate the value of k in knn to be used in my implementation. The distances array is iterated through for k number of times to collect the k nearest neighbours. The most frequent class in the nearest neighbours list is calculated and appended to the list of predicted classes. The actual label is appended to a different list of classes. If there is a tie in the number of predicted classes, then if the real label is in those predicted classes, it is assumed to be correct. The list totalPredictions and totalTrueLabels are returned to getDistance where the iteration through each test and training data is continued. Once the test data has been completely iterated through, the model is saved using the method saveModel, where totalPredictions and totalTrueLabels are passed in as parameters. The method saveModel uses pickle to save the two lists as a model cannot be saved.

- **F4)**
  - To output the train and test errors for both models, the models are first loaded and returned using the method loadModels. To calculate the error percentage of the training and test data for the direct implementation of knn, the built in .score method is used. Error% = 1 – accuracy. For my own implementation, sklearn's accuracy_score is used, where the accuracy between the prediction label list and real label list is calculated. This gives the % error for the test data.

- **F5)**
  - To allow the users to query the models using a variable to query the index of the test dataset, I created a method called queryIndex, taking no parameters. A try except is used as the user may not input a valid index, being out of range or not an integer. Within the try statement, the user is asked for an input, if not valid they are told it is not a valid index and asked again for an input. Otherwise, the models are loaded and returned using loadModels. If the index is not within the length of the test dataset, then they are told that the index must be within the size of the dataset and asked for an input again. Otherwise, if the input is valid, for my own implementation the predictions list will be queried with the index given. For the direct implementation, the built in .predict method is used with the index given. These are then output to the user as the predicted classes for the given test data.

- **Additional requirements**
  - Each of the above functionalities are within their own methods. To run each of these, a menu is provided. As the functionalities are in their own methods, f1, f4, and f5 can be run by loading the saved models without calling the training functionalities f2 and f3.
  - To train the models, select either 2 or 3 from the menu provided. Nothing will be output as the models are only trained then saved. The indirect implementation may take 30 seconds to run. Choose 99 to quit the menu.

Meaning of parameters and variables

- **F1)**
  - nSamples – number of data entries
  - nClasses – number of classes
  - minMin – minimum and maximum values for each feature
  - classesTotal – number of entries per each classification

- **F2)**
  - k = k value for knn
  - knn = classifier

- **F3)**
  - JaccardIndex
    - xTrain – training data
    - xTest – testing data
    - intersect – the intersection of the two sets
    - union – the union of the two sets

  - getDistance
    - XTrainStd – transformed training data
    - XTestStd – transformed test data
    - yTest – label of the test data
    - yTrain – labels of the training data
    - distance – jaccard distance of test and training data
    - pointDistance – sorted list of distances
    - realLabel – real label of test data
    - realLabelIndex – index of label incremented for every test data

  - getClass
    - distances – list of distances from one test data piece to all other training data
    - realLabel – real label of the test data
    - totalPredictions – list of all predictions for classifications of the testing data
    - totalTrueLabels – all real labels of the testing data
    - k – k value for knn
    - kNearest – training data of nearest neighbours
    - predictedLabel – predicted label for testing data
    - newCount – counting the number of occurrences of each classification
    - maxClass = maximum count for classification
    - maxCount – the maximum number possible for classifications

  - saveModel
    - totalPredictions – list of all predictions for classification of the testing data
    - totalTrueLabels – all real labels of the testing data
    - combined – tuple of totalPredictions and totalTrueLabels

- **F4)**
  - compareErrors
    - impl – saved model of indirect implementation
    - classif – saved model of direct classification

        - testAcc = testing accuracy of indirect implementation

- **F5)**
    - queryIndex
        - index = users input
        - totalPredictions – list of all predictions for classification of the testing data
        - impl – saved model of indirect implementation
        - classif – saved model of direct classification

Idea of algorithm

- The idea of the algorithm was that each of the test data entries would be compared to each and every single training data entry. For each training data entry, the distance would be calculated from the test data entry and appended to a list. Once the testing data entry has been compared to all training data, the list is sorted and the k nearest neighbours are found. Based upon the most frequent occurrence of a classification in said list, that would be the predicted classification for that single testing data entry. This is completed for all testing data entries.
- The idea for the entire program was that the user could input a number from 1 to 5, corresponding to f1 to f5, and the implementation for said functionality would run successfully. This would mean that each functionality would need to be within its own methods and only be called when its specific method was called from the menu.