Charles Papandreou
Duke University – IoT Lab
26 July 2019

This document details each file created during my work on the DMA research project this summer. There are descriptions of what each file is used for as well as descriptions of each of the enclosed function in the case they were required to be modified.

Files:

- **robotControl.py**
  - **Description:** This file allows for the robot to draw three variations of each digit, as well as organizing functions with which to call each of these various digit drawing functions
  - **Functions:**
    - *draw_0a()*
      - draws variation #1 of digit 0
    - *draw_0b()*
      - draws variation #2 of digit 0
    - *draw_0c()*
      - draws variation #3 of digit 0
    - *print_Zero()*
      - generates a random number that selects between variation 1, 2, or 3 of digit 0
    - *…*
    - *draw_9c()*
    - *printNine()*
    - *draw_NUMBER(numToDraw)*
      - takes in a string input of which digit and variation should be drawn and calls the correct respective function. Used in testing
    - *draw_Sequence(inpt)*
      - takes in a string input of a sequence of digits to draw and calls the respective print functions, printing a random variation of each digit for each digit in the specified sequence. Could be used for eventual testing to see if learning algorithms can recognize a random sequence
    - *MainFunction*
      - When performing data collection in connection with the DMA machine, edit the numToDraw field on line ~741 to specify what digit to collect data for. Keep drawTimes set to 1 for single repetitions when calling the robotControl.py script from DMA collection file.

- **featureExtractionDMA.py**
  - Description: This file is used to create a csv holding all of the extracted features for digit.mat files. This format csv is used as an input in classificationDigits.py. Features being used are peak, wave, dispersion, and auto-correlation.
  - Functions:
    - *createFeatureDict(filename,nth,rOrI)*
      - filename specifies the name of the .mat file being read in for data. This should be a .mat file containing a (#samples x seriesLength x #masks x 2) matrix
      - nth specifies when extracting time series data from the .mat files to take every nth time measurement. (ie- 1 is every single step, 2 is every 2^{nd} step, 1000 is every 1000^{th} step)
      - rOrI specifies whether to read in the series values for real or imaginary components. This is in the format of a string saying "real" or otherwise
      - First - creates a dictionary where each key is a sample number and the value is a 2-dimensional array with each index being a mask number and the array stored at each being a new time series.
      - Second – creates a feature dictionary with each key as a sample num and the value being an array of 4 arrays each holding 50 features. Each sample will have 200 features (4 per mask).
      - Returns the created feature dictionary
    - *extractFeatures(everyNth,rOrI)*
      - calls createFeatureDict for each of the digit.mat files provided. Creates a list of each of the feature dictionaries being returned by createFeatureDict(). When adding more variations of digits later, will need to add more calls to createFeatureDict in this function.
      - Returns a list of feature dictionaries
    - *generateCsvOutput(outputFilename,everyNth,rOrI)*
      - calls extractFeatures() to get list of features. Then iterates through creating a csv file as output storing all features. Each row is a sample, with each sample holding 201 columns. 200 of these are features, and the last is that samples correct classification for future reference and testing.
    - *MainFunction*
      - Bottom of file calls generateCsvOutput specifying output file name. Edit these parameters to generate various types of csv outputs for different specifications.

- **classificationDigits.py**
  - Description: The purpose of this file is to test the accuracy of some basic learning algorithms with different specifications for the number of masks used. The learning algorithms used include K-Nearest Neighbors, Linear Support Vector Machines, Random Forest, and Naive Gaussian Bayes. 10-fold cross validation is used and each of these 10 results is averaged to give a single value for the masks being used. When 5 masks are specified, every sliding window of 5 masks is tested (ie – 0-5, 1-6, 2-7 etc). The results of each of these are returned in a list of accuracies, which are eventually averaged to provide an average accuracy at 1, 2, 5, 10, 20, and 50 masks for each algorithm type for both real and imaginary numbers.
  - Functions:
    - *knnCV(csvName, numMasks, folds, neighbors)*
      - performs knn for each of the specified number of folds acting as testing data. Neighbors specifies the number of neighbors being selected for knn.
    - *svmCV(csvName, numMasks, folds)*
      - performs linear svm for each of the specified number of folds acting as testing data.
    - *forestCV(csvName, numMasks, folds, trees)*
      - performs random forest for each of the specified number of folds acting as testing data. trees specifies the number of trees being created in each random forest
    - *bayesCV(csvName, numMasks, folds)*
      - performs naïve Gaussian bayes for each of the specified number of folds acting as testing data.
    - *plotAvgs(avgVals)*
      - Takes in a 2d list of average vals, with the lists at each index representing the average vals for a different algorithm and datatype
    - *generateAvgAccVals(filName1, filName2)*
      - Takes in names of files to read features from for both real (file 1) and imaginary (file 2) numbers. Average accuracy values are generated for each algorithm for real and imaginary numbers at 1,2,5,10,20, and 50 masks. The accuracy values for each algorithm are saved in a list which is appended to a larger list to be returned as an ouput
    - *MainFunction*
      - *Bottom of the file contains a call to the generateAvgAccVals() function whose return value is stored as an array in averageVals. plotAvgs is then called with averageVals below this, which results in a plot as an output for the file representing accuracies of each algorithm at different numbers of masks.*

- **saveTrained.py**
    - Description: Used to save pre-trained models that can be loaded onto other devices. Trains a model for knn, svm, random forest, and naïve bayes. Allows for specification of number of masks used in dataset when training. It is important to note that when using these models on a machine, oftentimes it will require that they have been initially trained on that machine or another compatible machine.
    - Functions:
        - *createTrainedModels(numMasks)*
            - reads in dmaFeaturesReal.csv and selects specified number of masks. Then trains four types of models and saves them in .sav files to be used for prediction later in other programs.
        - *MainFunction*
            - Calls createTrainedModels for varying numbers of masks. In this case currently using 1, 2, 5,10,20 and 50 masks when training models. Can add or adjust by editing lines at bottom of code

- **clientSideTime.py**
  - Description: Generates average time outputs to a csv for different aspects of feature extraction and prediction processes while interacting with a server. Performs process with feature extraction locally versus all on server. Gives time outputs for total times, feature extraction, classification, model loading time, encoding of data, communication/transmission of data, and decoding of data. Only run this code after server is up and running.
  - Functions:
    - *extractSignals(digitNum)*
      - takes in an integer value that represents which digit to extract signals from. Iterates through returning a list with each index representing a sample and holding a 50x4096 list.
    - *localFeatureServerClass(digitD,numMasks)*
      - takes in a 2d array with each index representing a mask number and the arrays stored at each index being a time series.
      - takes in the number of masks being used when extracting features which helps pick the correct pre-trained model for classification.
      - Starts by extracting features and storing them in a array of length 200 then encoding using jsonpickle, and sending to server which eventually receives a response back
    - *serverFeatureClass(digitD,numMasks)*
      - takes in a 2d array with each index representing a mask number and the arrays stored at each index being a time series.
      - takes in the number of masks being used when extracting features which helps pick the correct pre-trained model for classification.
      - Starts by encoding the 50x4096 array holding all the time series data using jsonpickle then sending it to a server to perform classification and feature extraction before receiving a response
    - *timeSingleDigit(signals,numMasks)*
      - takes in all time series for each sample of a single digit. Gets times for each piece of the process by calling both serverFeatureClass() and localFeatureServerClass(). Repeats each of these processes 50 times for each sample and averages the final results to receive average values for each part of the process
    - *generateCsvOutput()*
      - Calls extract signals then proceeds to call timeSingleDigit() for each of 1,2,5,10,20, and 50 masks. Stores the resulting time arrays in a list which is then used to create a csv output.

- **serverPi.py**
  - Description: Performs the parts of the feature extraction and classification process not performed locally on the client side by the edge device. In order to use properly, must be in same folder as the pre-trained models on the device that is running the server. Must start this program before using the clientSideTime.py program for data collection
  - Functions:
    - *test()*
      - starts by decoding the transferred data by using jsonpickle decode, after which the first index of the read in array is used to specify the number of masks and the second index tells whether or not feature extraction will be required. This second index stores “extract” if needed and a normal data point otherwise. If extraction is required, it performs feature extraction to create the list of features. It then uses this information as a test case with the pre-trained models to perform classification. It takes all the timed values and returns them to the client side after finish

- **timeTestEdge.py**
  - Description: This program allows for testing of the feature extraction and classification process solely performed on single device.
  - Functions:
    - *extractSignals(digitNum)*
      - takes in an integer value that represents which digit to extract signals from. Iterates through returning a list with each index representing a sample and holding a 50x4096 list.
    - *timeTestSingleSampleMasks(digitD, numMasks)*
      - performs and times each step of the process including model loading time, feature extraction, classification, and total times
      - Takes in the number of masks to use as well as an array holding time series values for each of the 50masks
    - *timeTestSingleDigit(signals, numMasks)*
      - calls timeTestSingleSampleMasks() 50 times for each sample of a digit and averages the results adding the final results to a list and returning it
    - *MainFunction*
      - Calls extract signals to get the data, then uses it to call timeTestSingleDigit with 1,2,5,10,20, and 50 masks being considered. Accumulates all the results and outputs them in a useable format to a csv