This is the documentation / user guide for all the codes that are put up in this repository.

This part is about pre-processing of the dataset.

PRE-PROCESSING

For the approaches listed below, the data provided has to be pre-processed before it is used to train the ML model. The method to pre-process data varies for training and testing.

TRAINING DATASET

The training data is one single large CSV file containing data from all the 600 CSV files in the input zip files. To obtain the dataset in this form, the following steps are to be followed:
1. Combine all the CSV files from each scenario folder into one single file using ***'Combiner.py'*** program to get combined CSV files for each scenario. Use ***'delete_empty.py'*** to remove all empty rows in dataset.
2. Extract all the *RSSI, SINR and Throughput* values from each scenario's output text file and store them in new text files.
3. Now use the ***'Convertor.py'*** program, in which paste the values from the new text files (created for each of the parameters) into the array *'a'*. Then change the name of column according to name of parameter to be added. Please see that the values must be appended to the right combined CSV scenario file.
4. After appending all the 3 parameters values to the respective files, combine the newly created CSV files again with ***'Combiner.py'***. Once again use ***'delete_empty.py'*** to remove all empty rows in dataset.
5. In the final large CSV file remove all the columns except the features and labels required.
6. The large CSV should be like the ***'Combined_rssi_corrected_proper_sinr_row_fixed.csv'*** present in the repository.

TESTING DATASET

The testing data is a folder containing all the pre-processed CSV files of all the 200 files provided as test dataset. To obtain these pre-processed CSV files, the following steps are to be followed:
1. Since the SINR and RSSI values are provided in the form of rows in the respective CSV files, they are transposed as column using ***'convertor_vertical.py'***. The output will be CSV files with the extracted values being appended to respective columns.
2. Use ***'Combiner.py'*** to combine all newly transposed RSSI and SINR files to the respective input files. Please note that RSSI, SINR and input files must be in the same location and they should belong to the respective scenario.
3. Use the ***'remover.py'*** to remove the first index column (redundant column) which has been created due to the previous step.
4. Now as the above steps have been performed for all the 4 test folders, it is recommended to copy paste the newly created CSV files all into one single directory.
5. In the newly created directory with the files, use the ***'cleaner.py'*** program to filter out the necessary feature columns only. The preprocessed CSVs are thus obtained.
6. The final files should be in the same form as available in the repository as a zip file named ***'All_combined_proper_clean_final.zip'.***
Note: During the above process, in the final files, SINR and RSSI values have header names as 0.1 and 0 respectively. The header names do not matter while feeding the test CSV files to model for prediction of Throughput. For convenience these header names are left as it is.

This part of documentation is for the ANN code.

Artificial Neural Network (ANN) Approach

This documentation is for the ANN approach to the problem statement. The code for this documentation is available in the ANN folder in the Github repository. Since the work on this approach was done primarily in Google Colab, the python code was implemented in the form of the Jupyter Notebook. The folder ANN contains two files, the Jupyter Notebook format (.ipynb) of the code and the converted script format of the same code.

Language and Packages

To use either of the above two formats of this code, it is necessary to have installed Python 3 language in the system, preferably version 3.6+, and the respective pip installer.

The main packages and libraries used in corresponding code are: **TensorFlow, Pandas, Numpy, Matplotlib, Keras, Scikit Learn, Regex, Glob**

This approach is mainly implemented using Keras and Tensorflow libraries. The data for training has been transformed using Standard Scaler. The training data is the large CSV obtained from combining all training CSVs after preprocessing. The testing data are preprocessed CSV files.

- A brief explanation of the model is explained. There is one Input layer that takes the size of the standard scaled input features array (NumPy). There are 6 sequential layers containing 1024 hidden nodes(units) and each of them is ReLU activated. The 7th hidden layer contains 512 nodes and is also ReLU activated. The last output layer has one output node to get the predicted value. While compiling the model the loss function considered is the mean squared error loss function. The optimizer chosen for training the model is the Adam optimizer. The metrics used for compilation are MSE (mean squared error) and MAE (Mean Absolute error). The test CSV files are fed into the trained model using the predictor function.
- Initially we had trained the model with 80% of available dataset and we used 20% of it for testing. And once we had got different dataset for testing, we used 100% of earlier dataset for training the model.

Using Script (.py) file:

After successfully installing the requirements, the Python script can be downloaded and run using:

python ANN_predictor.py

It must be noted that when running the script, the locations of the training data set (preprocessed large CSV file) and the location of the test files (preprocessed CSV files) must be clearly filled by giving the complete address of the location. A better approach is to place the script, training CSV, and the testing files in the same directory.

After the execution of the program, the directory of the test files will have new CSV files with the same name as each corresponding test file, but with *'_ANN_throughput'* as an extension to their names. These are the output CSV files with *Throughput* values being appended.

Using Jupyter Notebook (.ipynb) file:

In the case of running the *Jupyter Notebook* format*(.ipynb)*, the *Jupyter* notebook software has to be downloaded into the system.

Change directory to the path containing the downloaded *.ipynb* file. It is better to have the *.ipynb* file in the same location as where the training and testing preprocessed CSV files are present.

Run all the cells one by one to get the respective outputs below each cell.

After running all the cells in the notebook, the directory of the test files will be filled with new CSV files with the same name as each corresponding test file, but with *'_ANN_throughput'* as an extension to their names. These are the output throughput files.

Using Google Colab (Recommended):

As mentioned in the beginning, this program was done in *Google Colab (cloud)*. This is recommended if the installation of python, required packages, and other requirements are to be skipped. It is necessary to have a google account to use *Colab* and it is also recommended to give permission to connect to *Google Drive*, which serves as the storage.

The link to the *Colab notebook* for this program is as below:

https://colab.research.google.com/drive/1QmvV5HU9v4KHV8W2JNuwzhRH5Y18AnsP?usp=sharing

Google Drive is recommended to be used as it is easier to upload the training CSV and testing CSV files and access it in *Colab*.

The alternate way is to upload the files to session storage in *Colab*. This is not recommended as it takes a long time to upload and mainly because the uploaded files are available only during runtime.

In the notebook, it must be seen that the locations of the training CSV files and testing CSV files in the drive must be properly mentioned. Also, it is recommended to upload the training preprocessed CSV in *.csv* format and the testing preprocessed CSV files as a zip file.

Run all the cells one by one to see respective outputs after each cell. After running all the cells there will be a zip file named *'ANN_throughput_results.zip'* in the Files tab. This zip contains both the test files as well as the new CSV files with the same name as each corresponding test files, but with *'_ANN_throughput'* as an extension to their names. These are the output throughput files.

Download the zip to access the output throughput files.

It must be noted that in all the formats of the code, there is a set of commented lines that can be used to save the model in *HDF5* format. This model can be downloaded and loaded in a different program for experimentation. This method is recommended in order to avoid running the training part of the Neural network over and over each time.

The final output in all the formats is a set of output CSV files with *Throughput* values being appended.

This part of documentation is for the KNN_predictor_func.py script.

The prerequisite packages to run this python script are as follows: **Scikit Learn, Numpy, Pandas, Regex, Glob**

This method is mainly implemented using scikit-learn library. The training data has been transformed using Standard Scaler. And it is the CSV obtained from combining all the pre-processed training CSVs (from different deployments in different scenarios). The testing data is pre-processed in the same way as training data.

It is recommended to have training dataset, testing dataset and python script in the same directory, otherwise please provide the correct path while reading the CSV files.
In this approach *sklearn KNeighborsRegressor* is used considering *n_neigbhors* to be 10. Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are used as metrics to find the accuracy of the Regressor.
Initially we had trained the model with 80% of available dataset and we used 20% of it for testing. And once we had got different dataset for testing, we used 100% of earlier dataset for training the model.

In the ***predictor*** function all the test CSV files are fed to the model for predicting the throughput.

And finally, the python script is executed using the command:

python KNN_predictor_func.py

After the execution of the program, the directory of the test files will have new CSV files with the same name as that of corresponding test file, but with *'_KNN_throughput'* as an extension to their names. These are the output throughput files. *'_KNN_throughput'* as an extension to their names. These are the output CSV files with predicted *Throughput* values being appended.

This part of documentation is for the Random_forest_predictor_func.py script.

The prerequisite packages to run this python script are as follows: **Scikit Learn, Pandas, Numpy, Glob**
This method is mainly implemented using scikit-learn library. The training data has been transformed using Standard Scaler. And it is the CSV obtained from combining all the pre-processed training CSVs (from different deployments in different scenarios). The testing data is pre-processed in the same way as training data.

It is recommended to have training dataset, testing dataset and python script in the same directory, otherwise please provide the correct path while reading the CSV files.

In this approach we have used max depth of 10 as one of the parameters for the random forest regressor. Next the model is trained using the transformed data, and the predictions are made. Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are used as metrics to find the accuracy of the Regressor.
Initially we had trained the model with 80% of available dataset and we used 20% of it for testing. And once we had got different dataset for testing, we used 100% of earlier dataset for training the model.

In the ***predictor*** function all the test CSV files are fed to the model for predicting the throughput.

And finally, the python script is executed using the command:

python Random_forest_predictor_func.py

After the execution of the program, the directory of the test files will have new CSV files with the same name as that of corresponding test file, but with *'_throughput'* as an extension to their names. These are the output throughput files. *'_throughput'* as an extension to their names. These are the output CSV files with predicted *Throughput* values being appended.