USER MANUAL

This manual describes the sequence of scripts to be run, gives a broad outline of what it does, and the results/files it produces. For knowing how to implement these (the required changes, etc.), please refer to the ***programmer manual*** *(describes the "how" part in the same order as below)*

It is highly recommended that you read and follow both the manuals simultaneously.

To implement rule-based extractor –

1. Run **rule_weighting.py** -> it will store the individual performance of each of the 9 rules (used as weights later on). It generates 3 files at a time (for partial match, exact match, dice score) for a single rule at a time. It ultimately produces 12 files – weighting_results_<tweets/quotes>_<normal/conditional>_<partial/exact/dice>.txt
2. Run **rule_based_app.py** -> Returns 4 lists at a time – candidate targets for majority approach, candidate targets for weighted majority (1 each for partial match, exact match, dice score – utilizing the 3 different weights for the 3 different metrics). These lists are stored – total 8 files – <tweets/quotes>_maj_list.txt , <tweets/quotes>_weighted_<partial/exact/dice>_list.txt
3. Run **rule_based_app_evaluation.py** -> Reads in the 4 lists (tweets, snippets one at a time)of candidate targets, compares with actual targets and give the results (for the 3 metrics for majority as well as weighted majority approach). Only prints these results.

To implement statistical extractor –

1. Run **generate.py** -> will produce the training and test files (17 each) in format for SVM HMM tool.
2. Run **convert_for_svm_labels.py** - > will change the above generated files (train and test) to the format for SVM perf tool for statistical extractor (Do not run this step to implement sequence labeling. Rest all instructions are same!)
3. Run **generate_sents.py** -> generates file for test sentences and actual targets, used for evaluation purposes.
4. **Run_SVM.txt** -> use the commands and follow instructions to implement the tools.
5. **classification_acc.py** -> do not run. Just some changes required for different results (refer programmer manual) before the next step.
6. Run **produce_classi_results.py** -> runs above script for all folds for all feature combinations and produce the results across all the four folds.
7. Run **final_results.py** -> averages the results produced above across the four folds and produces the final results for statistical extractor. These results are stored in apt files.

To implement hybrid framework –

1. Run steps 1-2 for rule-based extractor
2. Run steps 1-4 for statistical extractor

3. Now, we have all the required files. Do not run **integ_34.py** -> set <snippets/tweets> in required file names, and to set up AND/OR – a) change file names b) change intersection/union for sets in 3 places (refer programmer manual).
4. Implement **produce_classi_results.py** - > set it for integ_34.py, 4 runs to get 12 types of file – <PARTIAL/EXACT/DICE >_i34res_<AND/OR>_t(1-17) (6 for snippets, 6 for tweets) across all folds
5. Run **final_results.py** -> averages the results produced above across the four folds and produces the final results for hybrid. These results are stored in apt files.

NOTE :

For implementation of system, it is necessary to keep the code files, results files and dataset files in one single folder! (or do the necessary changes in the file names of various codes).