Smart To-Do – Documentation

**1. Generating HitApp Task data:**

(a) JSON from MSR data

The MSR commitment data file is stored 'SmartToDo/Avo\_commitment\_data/avocado\_commitment.csv'

This file contains 7730 raw avocado sentences along with information about the sentence index, email-id, body, subject, Sent-to, Sent-from and a label of whether it is a ground truth commitment or not.

*Python scripts in ‘SmartToDo/CodeRepo/genCommitDataSet/’:*

**genSQLQuery\_reply\_Info.py** – Reads each data point in MSR commitment data file and stores the original email-ids of data points which have a commitment sentence. It then generates a SQL query to obtain the reply\_to IDs of those original emails-ids.

The original email id and reply\_to email ids are output to Commit\_reply\_Info.csv after SQL server query.

**genSQLQuery\_email\_sentence\_Info.py** – Obtains the set of all original and reply\_to email-ids and then generates SQL query to extract Sentence Information of all those ids. Also generates another SQL to extract Subject, Sent-to, Sender Name, Body of all the email-ids.

The information is output to Commit\_email\_Info.csv and Commit\_sentence\_Info.csv from SQL server query.

**genDataSet\_json.py** – Combines the information from Commit\_reply\_Info.csv, Commit\_email\_Info.csv and Commit\_sentence\_Info.csv for all the positive commitment labelled points in MSR commitment data file. It then creates a dictionary storing all the sentences of replied\_to email, current email, Subjects, Sent-To, Sent-From information, Commitment sentence. All these information will be used in the HitApp Task file creating.

Dumps the Python dictionary to a json file – ‘./CodeRepo/data/commit\_data/Commit\_dataset.json’

(b) JSON from TEE Classifier output

*Python Scripts in ‘./CodeRepo/genAugmentDataSet’:*

**genSQLQuery\_extract\_raw\_sentences.py** – Generates a SQL query to extract Avocado sentences from entire corpus, excluding sentences from duplicated email-ids, or those which are already in MSR commitment data. We also choose sentences which has >2 and <=50 tokens.

Outputs the raw sentences to ‘./CodeRepo/data/augment\_data/raw\_Highlight\_Candidates.csv’

**genAEther\_Classifier\_input.py** – Converts the raw sentences to a format that can be input to AEther Classifier. Mainly removal of tabs, tagging along the sentence id, sentence index, email id info with the raw sentences.

Outputs to ‘./CodeRepo/data/augment\_data/augment\_Aether\_Input.tsv’

**process\_Commitment\_output.py** – Takes TEE Classifier output and labels sentences > 0.90 probability as Commitment sentence (Label 1). Else, labels it as 0. Creates the file ‘CodeRepo/data/augment\_data/highlight\_info.tsv’

A similar JSON file creation using SQL queries is used again for the Aether output where highlighted sentences are first chosen using the TEE classifier.

The analogous files are:

'SmartToDo/Avo\_commitment\_data/avocado\_commitment.csv' ⬄ ‘CodeRepo/data/augment\_data/highlight\_info.tsv’

‘./genCommitDataSet/genSQLQuery\_reply\_Info.py’ ⬄ ‘./genAugmentDataSet/genSQLQuery\_reply\_Info.py’

‘./genCommitDataSet/genSQLQuery\_email\_Info.py’ ⬄ ‘./genAugmentDataSet/genSQLQuery\_email\_Info.py’

‘./genCommitDataSet/genSQLQuery\_sentence\_Info.py’⬄ ./genAugmentDataSet/genSQLQuery\_sentence\_Info.py’

Dumps the Python dictionary to a json file – ‘./CodeRepo/data/augment\_data/Augment\_dataset.json’

(c) HitApp TSV from JSON Dumps

*Python scripts in ‘SmartToDo/CodeRepo/HitApp\_process’:*

**genHitApp\_tsv\_v2.py** – Creates HitApp Task file from JSON Dump ‘./CodeRepo/data/commit\_data/Commit\_dataset.json’

The outfile file path has to be changed appropriately.

Example –

path\_to\_hitapp\_task = **'Pilot3\_batch\_start\_{}\_size\_{}.tsv'**.format(start\_index, max\_points)

**genHitApp\_augment\_tsv.py** – Creates HitApp Task file from JSON Dump ‘./CodeRepo/data/augment\_data/Augment\_dataset.json’

The outfile file path has to be changed appropriately.

Example –

path\_to\_hitapp\_task = **'Augment\_batch\_start\_{}\_size\_{}.tsv'**.format(start\_index, max\_points)

Examples of HitApp Task Files:

Augment\_batch\_start\_0\_size\_2000.tsv

Pilot3\_batch\_start\_100\_size\_100.tsv

JSON Dump

(Commit\_dataset.json)

start\_index, max\_points,

path\_to\_hitapp\_task

HitApp Task File

**genHitApp\_tsv\_v2.py**

**2. Generating Input for Seq2Seq models:**

Run the code **‘./CodeRepo/extractiveSummary/gen\_sent\_ranking.py’.**

We have kept the UHRS judgements both in ‘./CodeRepo/data’ and in ‘./CodeRepo/logs’ to prevent any accidental alteration.

UHRS Judgements

(List of Paths)

./CodeRepo/data/sent\_ranked\_fasttext.txt

(Ranking to determine “Useful” sentence)

Fast-text Embeddings

trained on Avocado

**gen\_sent\_ranking.py**

*Detour - Training Fast-text Embeddings:*

The code for training Fast-text embeddings on entire Avocado corpus is available in the folder **‘./CodeRepo/wordEmbed’**. After training the model, the embeddings are stored in:

**‘./CodeRepo/logs/checkpoint\_wordEmbed/Avocado/corpus800k’**. The above pipeline looks up the embeddings from this checkpoint.

It contains both 50 and 300 dimensional embedding vectors stored as Gensim model file. The model was trained using Gensim with spacy tokenization. The main script for training Fast-text on Avocado is in **‘./CodeRepo/wordEmbed/Embed\_Vocab.py’**

Once the ranking is obtained, each Seq2Seq model has its own input, output requirements. The data also needs to be split into train-validation-test. These scripts are available in:

‘./**CodeRepo/dataUtil’**

UHRS Judgements

(List of Paths)

Tokenizer

(Spacy or bert)

./CodeRepo/data/Orig\_seq2seq\_final\_data/

(Outputs tokenized src, tgt data from all filtered judgements)

**gen\_Orig\_seq2seq\_data.py**

./CodeRepo/data/Orig\_seq2seq\_final\_data/avocado.spacy\_tokenized/src-all.txt, tgt-all.txt

src-valid, tgt-valid

src-test, tgt-test

src-train, tgt-train

**gen\_Orig\_train\_val\_test\_split.py**

**3. Training Seq2Seq models:**

**\* We first transfer the data generated for each Seq2Seq model from ‘./CodeRepo/data/’ to their respective directories (viz. BiFocal\_NMT, QueryFocused\_NMT, OpenNMT-py) \***

The folders OpenNMT-py, BiFocal\_NMT and QueryFocused\_NMT contain a README\_<alg>.txt file (README\_Orig,txt, README\_BiFocal.txt, README\_QuerySent.txt) , which provides details on how to run them on the server.

There is also a file Template\_command\_line\_runs.txt which provides examples of running OpenNMT based codes. Additional details in OpenNMT official documentation:

[**http://opennmt.net/OpenNMT-py/**](http://opennmt.net/OpenNMT-py/)

**4. Computing Performance metrics:**

**‘./CodeRepo/myCode\_seq2seq.py’ –** Provide the path to the gold and predictions and it will compute Rouge-L, Rouge-1, Rouge-2, Blue4 and Meteor scores. You need to install a Python package for rouge calculation:

pip install py-rouge

**5. Hyper-parameter tuning:**

**‘./CodeRepo/HyperTune’ –** Contains code to generate config files and bash scripts. These are then run on the server. Note: Run the Python files to create the bash scripts in Linux and not in Windows. Otherwise, there are new-line incompatibility issues.