```
# Author: Shashi Narayan
# Date: September 2016
# Project: Document Summarization
# H2020 Summa Project
#####################################

"""
Document Summarization Modules and Models
"""



from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np
import tensorflow as tf
import random
import os
import re
import os.path

from pyrouge import Rouge155
import json
from multiprocessing import Pool
from contextlib import closing

from my_flags import FLAGS
```

```
def _rouge(system_dir, gold_dir):
    # Run rouge
    r = Rouge155()
    r.system_dir = system_dir
    r.model_dir = gold_dir
    r.system_filename_pattern = '([a-zA-Z0-9]*).model'
    r.model_filename_pattern = '#ID#.gold'
    output = r.convert_and_evaluate(rouge_args="-e /address/to/rouge/data/directory/rouge/data -a -c 95 -m -n 4 -w 1.2")
    # print output
    output_dict = r.output_to_dict(output)
    # print output_dict

    # avg_rscore = 0
    # if FLAGS.rouge_reward_fscore:
    #     avg_rscore = (output_dict["rouge_1_f_score"]+output_dict["rouge_2_f_score"]+
    #                   output_dict["rouge_3_f_score"]+output_dict["rouge_4_f_score"]+
    #                   output_dict["rouge_l_f_score"])/5.0
    # else:
    #     avg_rscore = (output_dict["rouge_1_recall"]+output_dict["rouge_2_recall"]+
    #                   output_dict["rouge_3_recall"]+output_dict["rouge_4_recall"]+
    #                   output_dict["rouge_l_recall"])/5.0

    avg_rscore = (output_dict["rouge_1_f_score"]+output_dict["rouge_2_f_score"]+output_dict["rouge_l_f_score"])/3.0

    return avg_rscore
```

```
def _rouge_wrapper_traindata(docname, final_labels, final_labels_str):
    # Gold Summary Directory : Always use original sentences
    gold_summary_directory = FLAGS.gold_summary_directory + "/gold-"+FLAGS.data_mode+"-training-org"
    gold_summary_fileaddress = gold_summary_directory + "/" + docname + ".gold"

    # Prepare Gold Model File
    os.system("mkdir -p "+FLAGS.tmp_directory+"/gold-"+docname+"-"+final_labels_str)
    os.system("cp "+gold_summary_fileaddress+" "+FLAGS.tmp_directory+"/gold-"+docname+"-"+final_labels_str+"/")

    # Document Sentence: Always use original sentences to generate summaries
    doc_sent_fileaddress = FLAGS.doc_sentence_directory + "/" + FLAGS.data_mode + "/training-sent/"+docname+".summary.final.org_sents"
    doc_sents = open(doc_sent_fileaddress).readlines()

    # Prepare Model file
    os.system("mkdir -p "+FLAGS.tmp_directory+"/model-"+docname+"-"+final_labels_str)

    # Write selected sentences
    labels_ones = [idx for idx in range(len(final_labels[:len(doc_sents)])) if final_labels[idx]=="1"]
    model_highlights = [doc_sents[idx] for idx in labels_ones]
    foutput = open(FLAGS.tmp_directory+"/model-"+docname+"-"+final_labels_str+"/"+docname+".model" , "w")
    foutput.write("".join(model_highlights))
    foutput.close()

    return _rouge(FLAGS.tmp_directory+"/model-"+docname+"-"+final_labels_str, FLAGS.tmp_directory+"/gold-"+docname+"-"+final_labels_str)

def _multi_run_wrapper(args):
    return _rouge_wrapper_traindata(*args)
```

```
def _get_lcs(a, b):
    lengths = [[0 for j in range(len(b)+1)] for i in range(len(a)+1)]
    # row 0 and column 0 are initialized to 0 already
    for i, x in enumerate(a):
        for j, y in enumerate(b):
            if x == y:
                lengths[i+1][j+1] = lengths[i][j] + 1
            else:
                lengths[i+1][j+1] = max(lengths[i+1][j], lengths[i][j+1])
    # read the substring out from the matrix
    result = []
    x, y = len(a), len(b)
    while x != 0 and y != 0:
        if lengths[x][y] == lengths[x-1][y]:
            x -= 1
        elif lengths[x][y] == lengths[x][y-1]:
            y -= 1
        else:
            assert a[x-1] == b[y-1]
            result = [a[x-1]] + result
            x -= 1
            y -= 1
    return len(result)
```

```
def _get_ngram_sets(highlights):
    set_1gram = set()
    set_2gram = set()
    set_3gram = set()
    set_4gram = set()
    fullen = len(highlights)
    for widx in range(fullen):
        # 1gram
        set_1gram.add(str(highlights[widx]))
        # 2gram
        if (widx+1) < fullen:
            set_2gram.add(str(highlights[widx])+"-"+str(highlights[widx+1]))
        # 3gram
        if (widx+2) < fullen:
            set_3gram.add(str(highlights[widx])+"-"+str(highlights[widx+1])+"-"+str(highlights[widx+2]))
        # 4gram
        if (widx+3) < fullen:
            set_4gram.add(str(highlights[widx])+"-"+str(highlights[widx+1])+"-"+str(highlights[widx+2])+"-"+str(highlights[widx+3]))
    return set_1gram, set_2gram, set_3gram, set_4gram

def _rouge_wrapper_traindata_nopyrouge(docname, final_labels_str, document, highlights):
    cand_highlights_full = []
    for sentidx in final_labels_str.split("-"):
        cand_highlights_full += [wordid for wordid in document[int(sentidx)] if wordid != 0]
        cand_highlights_full.append(0)
    highlights_full = []
    for sent in highlights:
        highlights_full += sent
        highlights_full.append(0)
    # print(cand_highlights_full,highlights_full)
```

```
    # Get sets
    cand_1gram, cand_2gram, cand_3gram, cand_4gram = _get_ngram_sets(cand_highlights_full)
    # print(cand_1gram, cand_2gram, cand_3gram, cand_4gram)
    gold_1gram, gold_2gram, gold_3gram, gold_4gram = _get_ngram_sets(highlights_full)
    # print(gold_1gram, gold_2gram, gold_3gram, gold_4gram)

    # Get ROUGE-N recalls
    rouge_recall_1 = 0
    if len(gold_1gram) != 0:
        rouge_recall_1 = float(len(gold_1gram.intersection(cand_1gram)))/float(len(gold_1gram))
    rouge_recall_2 = 0
    if len(gold_2gram) != 0:
        rouge_recall_2 = float(len(gold_2gram.intersection(cand_2gram)))/float(len(gold_2gram))
    rouge_recall_3 = 0
    if len(gold_3gram) != 0:
        rouge_recall_3 = float(len(gold_3gram.intersection(cand_3gram)))/float(len(gold_3gram))
    rouge_recall_4 = 0
    if len(gold_4gram) != 0:
        rouge_recall_4 = float(len(gold_4gram.intersection(cand_4gram)))/float(len(gold_4gram))

    # Get ROUGE-L
    len_lcs = _get_lcs(cand_highlights_full, highlights_full)
    r = 0 if (len_lcs == 0) else (float(len_lcs)/len(cand_highlights_full))
    p = 0 if (len_lcs == 0) else (float(len_lcs)/len(highlights_full))
    b = 0 if (r == 0) else (p / r)
    rouge_recall_l = 0 if (len_lcs == 0) else (((1+(b*b))*r*p)/(r+(b*b*p)))

    rouge_recall_average = (rouge_recall_1+rouge_recall_2+rouge_recall_3+rouge_recall_4+rouge_recall_l)/5.0
    # print(rouge_recall_1, rouge_recall_2, rouge_recall_3, rouge_recall_4, rouge_recall_l, rouge_recall_average)

    # Get final labels
    final_labels = [[1, 0] if (str(sentidx) in final_labels_str.split("-")) else [0, 1] for sentidx in range(FLAGS.max_doc_length)]  # [max_doc_length,
target_label_size]

    return rouge_recall_average, final_labels
```

```
def _multi_run_wrapper_nopyrouge(args):
    return _rouge_wrapper_traindata_nopyrouge(*args)

class Reward_Generator:
    def __init__(self):
        self.rouge_dict = {}

        # Start a pool
        self.pool = Pool(10)

    def save_rouge_dict(self):
        with open(FLAGS.train_dir+"/rouge-dict.json", 'w') as outfile:
            json.dump(self.rouge_dict, outfile)

    def restore_rouge_dict(self):
        self.rouge_dict = {}
        if os.path.isfile(FLAGS.train_dir+"/rouge-dict.json"):
            with open(FLAGS.train_dir+"/rouge-dict.json") as data_file:
                self.rouge_dict = json.load(data_file)

    def get_full_rouge(self, system_dir, datatype):
        # Gold Directory: Always use original files
        gold_summary_directory = FLAGS.gold_summary_directory + "/gold-"+FLAGS.data_mode+"-"+datatype+"-orgcase"

        rouge_score = _rouge(system_dir, gold_summary_directory)

        # Delete any tmp file
        os.system("rm -r "+FLAGS.tmp_directory+"/tmp*")

        return rouge_score
```

```
# def get_batch_rouge(self, batch_docnames, batch_predicted_labels):

#     # Numpy dtype
#     dtype = np.float16 if FLAGS.use_fp16 else np.float32

#     # Batch Size
#     batch_size = len(batch_docnames)

#     # batch_rouge
#     batch_rouge = np.empty(batch_size, dtype=dtype)

#     # Estimate list of arguments to run pool
#     didx_list = []
#     docname_labels_list = []
#     for docindex in range(batch_size):
#         docname = batch_docnames[docindex]
#         predicted_labels = batch_predicted_labels[docindex]

#         # Prepare final labels for summary generation
#         final_labels = [str(int(predicted_labels[sentidx][0])) for sentidx in range(FLAGS.max_doc_length)]
#         # print(final_labels)

#         isfound = False
#         rougescore = 0.0
#         if docname in self.rouge_dict:
#             final_labels_string = "".join(final_labels)
#             if final_labels_string in self.rouge_dict[docname]:
#                 rougescore = self.rouge_dict[docname][final_labels_string]
#                 isfound = True

#         if isfound:
#             # Update batch_rouge
#             batch_rouge[docindex] = rougescore
#         else:
#             didx_list.append(docindex)
#             docname_labels_list.append((docname, final_labels))
```

```
#     # Run parallel pool
#     if(len(didx_list) > 0):
#         # Run in parallel
#         rougescore_list = self.pool.map(_multi_run_wrapper,docname_labels_list)
#         # Process results
#         for didx, rougescore, docname_labels in zip(didx_list, rougescore_list, docname_labels_list):
#             # Update batch_rouge
#             batch_rouge[didx] = rougescore

#             # Update rouge dict
#             docname = docname_labels[0]
#             final_labels_string = "".join(docname_labels[1])
#             if docname not in self.rouge_dict:
#                 self.rouge_dict[docname] = {final_labels_string:rougescore}
#             else:
#                 self.rouge_dict[docname][final_labels_string] = rougescore
#         # Delete any tmp file
#         os.system("rm -r "+ FLAGS.tmp_directory+"/tmp* " + FLAGS.tmp_directory+"/gold-* " + FLAGS.tmp_directory+"/model-*")
#     # print(self.rouge_dict)
#     return batch_rouge
```

```
def get_batch_rouge_withmultisample(self, batch_docnames, batch_predicted_labels_multisample):
    """
    Args:
    batch_docnames: [batch_size]
    batch_predicted_labels_multisample: [batch_size, rollout_count, FLAGS.max_doc_length, FLAGS.target_label_size]
    Return:
    rougescore: [batch_size, FLAGS.num_sample_rollout]
    """

    # Numpy dtype
    dtype = np.float16 if FLAGS.use_fp16 else np.float32

    # Batch Size and sample rollout count
    batch_size = len(batch_docnames)
    rollout_count = batch_predicted_labels_multisample.shape[1]

    # batch_rouge
    batch_rouge_multisample = np.empty((batch_size, rollout_count), dtype=dtype)

    # Prepare of all rollout labels dict and prepare docname_labels_list to run
    docname_labels_rollout_dict = {}
    docname_labels_list = []
    for docindex in range(batch_size):
        docname = batch_docnames[docindex]
        # print(docname)

        for rolloutidx in range(rollout_count):
            predicted_labels = batch_predicted_labels_multisample[docindex][rolloutidx] # [FLAGS.max_doc_length, FLAGS.target_label_size]
            # Prepare final labels for summary generation
            final_labels = []
            final_labels_sindices = []
            for sentidx in range(FLAGS.max_doc_length):
                final_labels.append(str(int(predicted_labels[sentidx][0])))
                if int(predicted_labels[sentidx][0]) == 1:
                    final_labels_sindices.append(str(sentidx+1))
            final_labels_string = "-".join(final_labels_sindices)
```

```
        # print(final_labels,final_labels_string)

        isfound = False
        rougescore = 0.0
        if docname in self.rouge_dict:
            if final_labels_string in self.rouge_dict[docname]:
                rougescore = self.rouge_dict[docname][final_labels_string]
                isfound = True

        if isfound:
            # Update batch_rouge
            batch_rouge_multisample[docindex][rolloutidx] = rougescore
        else:
            if docname not in docname_labels_rollout_dict:
                docname_labels_rollout_dict[docname] = [docindex, {final_labels_string:[rolloutidx]}]
                docname_labels_list.append((docname, final_labels, final_labels_string))
            else:
                if final_labels_string not in docname_labels_rollout_dict[docname][1]:
                    docname_labels_rollout_dict[docname][1][final_labels_string] = [rolloutidx]
                    docname_labels_list.append((docname, final_labels, final_labels_string))
                else:
                    docname_labels_rollout_dict[docname][1][final_labels_string].append(rolloutidx)
                    # no need to add to docname_labels_list

# print(docname_labels_list)
# Run parallel pool
if(len(docname_labels_list) > 0):
    # Run in parallel
    with closing(Pool(10)) as mypool:
        rougescore_list = mypool.map(_multi_run_wrapper,docname_labels_list)
    # rougescore_list = self.pool.map(_multi_run_wrapper,docname_labels_list)
```

```
    # Process results
    for rougescore, docname_labels in zip(rougescore_list, docname_labels_list):
        docname = docname_labels[0]
        final_labels = docname_labels[1]
        final_labels_string = docname_labels[2]

        # Update batch_rouge
        docindex = docname_labels_rollout_dict[docname][0]
        for rolloutidx in docname_labels_rollout_dict[docname][1][final_labels_string]:
            batch_rouge_multisample[docindex][rolloutidx] = rougescore

        # Update rouge dict
        if docname not in self.rouge_dict:
            self.rouge_dict[docname] = {final_labels_string:rougescore}
        else:
            self.rouge_dict[docname][final_labels_string] = rougescore
    # Delete any tmp file
    os.system("rm -r "+ FLAGS.tmp_directory+"/tmp* " + FLAGS.tmp_directory+"/gold-* " + FLAGS.tmp_directory+"/model-*")
# print(self.rouge_dict)

return batch_rouge_multisample
```

```python
def get_batch_rouge_withmultisample_nopyrouge(self, batch_docnames, batch_predicted_labels_multisample_str, batch_docs, batch_highlights_nonnumpy):
    """
    Args:
    batch_docnames: [batch_size]
    batch_predicted_labels_multisample_str: [batch_size, rollout_count]
    batch_docs: [batch_size, FLAGS.max_doc_length, FLAGS.max_sent_length]
    batch_highlights_nonnumpy: [batch_size, highlights_lengths, each_highlights]
    Return:
    rougescore: [batch_size, FLAGS.num_sample_rollout]
    batch_gold_sampled_label_multisample: [batch_size, FLAGS.num_sample_rollout, FLAGS.max_doc_length, FLAGS.target_label_size]
    """

    # Numpy dtype
    dtype = np.float16 if FLAGS.use_fp16 else np.float32

    # Batch Size and sample rollout count
    batch_size = len(batch_docnames)
    rollout_count = batch_predicted_labels_multisample_str.shape[1]

    # batch_rouge
    batch_rouge_multisample = np.empty((batch_size, rollout_count), dtype=dtype)
    batch_gold_sampled_label_multisample = np.empty((batch_size, rollout_count, FLAGS.max_doc_length, FLAGS.target_label_size), dtype=dtype)

    # Prepare of all rollout labels dict and prepare docname_labels_list to run
    docname_labels_rollout_dict = {}
    docname_labels_list = []
    for docindex in range(batch_size):
        docname = batch_docnames[docindex]
        document = batch_docs[docindex]
        highlights = batch_highlights_nonnumpy[docindex]
        # print(docname)
```

```
for rolloutidx in range(rollout_count):
    final_labels_string = batch_predicted_labels_multisample_str[docindex][rolloutidx]
    # print(final_labels_string)

    if docname not in docname_labels_rollout_dict:
        docname_labels_rollout_dict[docname] = [docindex, {final_labels_string:[rolloutidx]}]
        docname_labels_list.append((docname, final_labels_string, document, highlights))
    else:
        if final_labels_string not in docname_labels_rollout_dict[docname][1]:
            docname_labels_rollout_dict[docname][1][final_labels_string] = [rolloutidx]
            docname_labels_list.append((docname, final_labels_string, document, highlights))
        else:
            docname_labels_rollout_dict[docname][1][final_labels_string].append(rolloutidx)
            # no need to add to docname_labels_list

    # isfound = False
    # rougescore = 0.0
    # if docname in self.rouge_dict:
    #     if final_labels_string in self.rouge_dict[docname]:
    #         rougescore = self.rouge_dict[docname][final_labels_string]
    #         isfound = True

    # if isfound:
    #     # Update batch_rouge
    #     batch_rouge_multisample[docindex][rolloutidx] = rougescore
    # else:
    #     if docname not in docname_labels_rollout_dict:
    #         docname_labels_rollout_dict[docname] = [docindex, {final_labels_string:[rolloutidx]}]
    #         docname_labels_list.append((docname, final_labels_string, document, highlights))
    #     else:
    #         if final_labels_string not in docname_labels_rollout_dict[docname][1]:
    #             docname_labels_rollout_dict[docname][1][final_labels_string] = [rolloutidx]
    #             docname_labels_list.append((docname, final_labels_string, document, highlights))
    #         else:
    #             docname_labels_rollout_dict[docname][1][final_labels_string].append(rolloutidx)
    #             # no need to add to docname_labels_list
```

```
# print(docname_labels_rollout_dict )
# print(docname_labels_list)

# Run parallel pool
if(len(docname_labels_list) > 0):
    # Run in parallel
    # with closing(Pool(10)) as mypool:
    #     rougescore_finallabels_list = mypool.map(_multi_run_wrapper_nopyrouge,docname_labels_list)
    rougescore_finallabels_list = self.pool.map(_multi_run_wrapper_nopyrouge,docname_labels_list)

    # Process results
    for rougescore_finallabels, docname_labels in zip(rougescore_finallabels_list, docname_labels_list):
        rougescore = rougescore_finallabels[0]
        finallabels = rougescore_finallabels[1]
        docname = docname_labels[0]
        final_labels_string = docname_labels[1]

        # Update batch_rouge
        docindex = docname_labels_rollout_dict[docname][0]
        for rolloutidx in docname_labels_rollout_dict[docname][1][final_labels_string]:
            batch_rouge_multisample[docindex][rolloutidx] = rougescore
            batch_gold_sampled_label_multisample[docindex][rolloutidx] = np.array(finallabels[:], dtype=dtype)

        # # Update rouge dict
        # if docname not in self.rouge_dict:
        #     self.rouge_dict[docname] = {final_labels_string:rougescore}
        # else:
        #     self.rouge_dict[docname][final_labels_string] = rougescore

# print(self.rouge_dict)

return batch_rouge_multisample, batch_gold_sampled_label_multisample
```