Gregory Brown https://github.com/toadSTL/CS5542_Lab
Class ID: 4
Big Data Analytics and Applications
CS 5542

Lab 2

1. Generate captions for dataset using Show and Tell model

It is worth noting from the outset that, since our project is on Visual Question Answering, the captions which we generate here will differ substantially from our reference samples since, for reference samples we will be using the "question"+" "+"answer" from our training set. Moreover, the bulk of the work which was done for this lab was in setting up and running the machine translation metrics which are used to evaluate how well the captions we generated match our reference samples. The code developed in the process of this lab is very lab to generate the accuracy scores will likely prove very useful for our project.

In order to generate captions for the images in our dataset, we move the images for which we wish to generate captions into a folder at <path_to_imgs>. Then we must modify 'inference.py' provided with the show and tell model, by adding the following snippet of code. This code generates

```
list_of_images = os.listdir(<path_to_imgs>)
img_files_paths = r""
for image in list_of_images:
   img_files_paths += <path_to_imgs>+image+","
img_files_paths = img_files_paths[:-1]
```

a string containing the filenames of the images for which we wish to generate captions, separated by commas, with directory information. We can then

```
tf.flags.DEFINE_string("model_path", r<path_of_model_file>, "Model graph def path")
tf.flags.DEFINE_string("vocab_file", r<path_of_vocab_file>, "Text file containing the
vocabulary.")
tf.flags.DEFINE_string("input_files", img_files_paths,
                       "File pattern or comma-separated list of file patterns "
                       "of image files.")
```

simply pass this string at the step in 'inference.py' where 'input_files' are defined.

Finally, we will output the generated captions in such a way that we can perform our evaluation metrics on them as easily as possible. To do this we modify highlighted lines from the main function of inference.py reproduced

```
model = ShowAndTellModel(FLAGS.model_path)
vocab = Vocabulary(FLAGS.vocab_file)
filenames = _load_filenames()
```

```
generator = CaptionGenerator(model, vocab)
output = ""
for filename in filenames:
   with tf.gfile.GFile(filename, "rb") as f:
       image = f.read()
   captions = generator.beam_search(image)
   print("Captions for image %s:" % os.path.basename(filename))
   for i, caption in enumerate(captions):
       # Ignore begin and end tokens <S> and </S>.
       sentence = [vocab.id_to_token(w) for w in caption.sentence[1:-1]]
       sentence = " ".join(sentence)
       output += os.path.basename(filename)+"\t"+sentence+"\n"
       print("  %d) %s (p=%f)" % (i, sentence, math.exp(caption.logprob)))
o = open("output.txt","w+")
o.write(output)
```

here. As the code shows, we load the model, vocab, and input files and then generate captions. All we have added is an accumulator output string which we write to 'output.txt' which takes the image filename and the caption generated for that filename.

The captions generated with beam_size = 1 are simply the 'best' captions generated for the images. When we generate 4 captions they will be the 'best' 4 captions as considered by the model. As such, we will discuss the captions generated here in the next section.

2. Generate 4 captions for each image

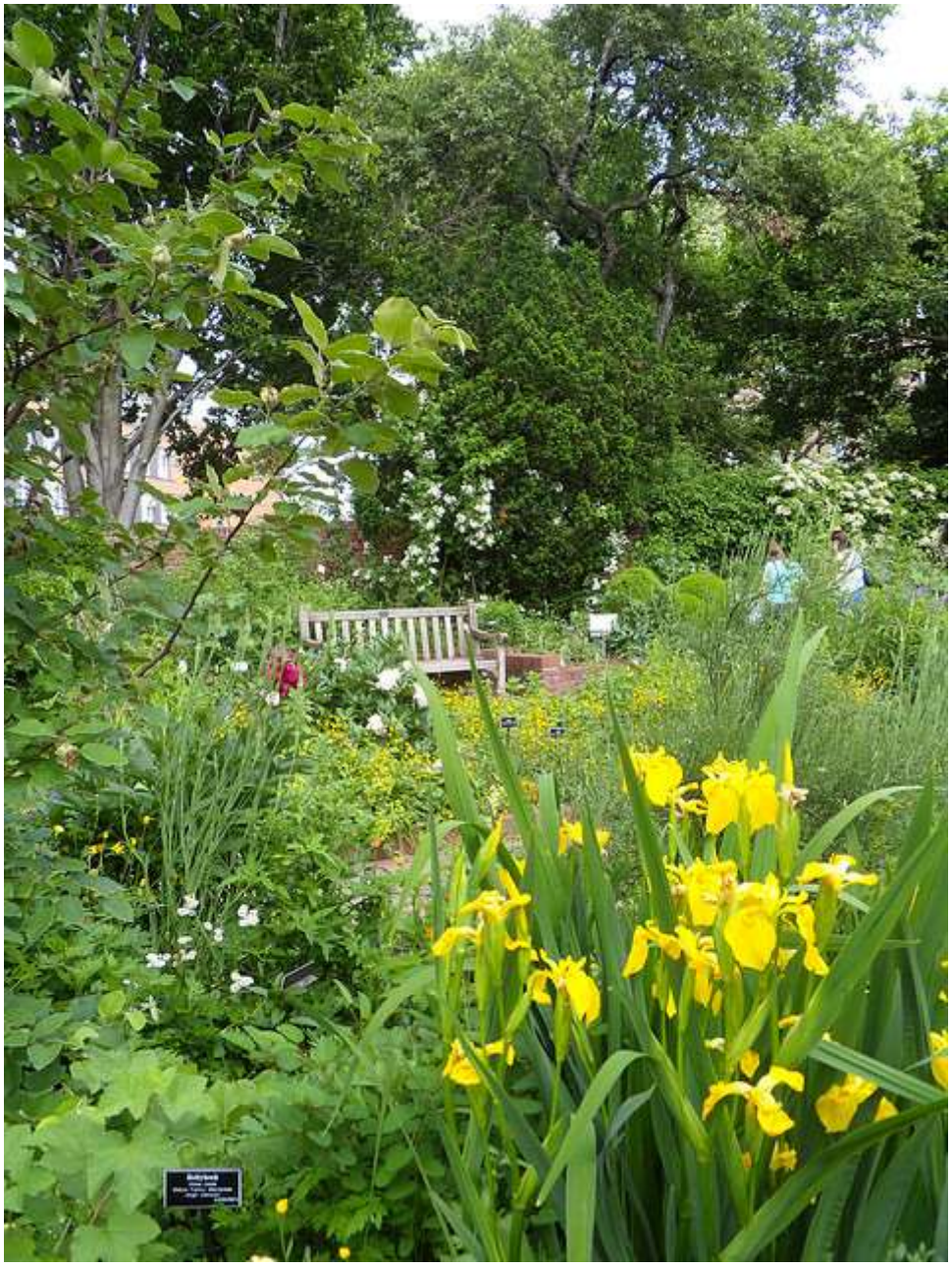
In order to update this scheme so that 4 captions are generated for each image we only need to update 'caption_generator.py' from the show and

```
def __init__(self,
             model,
             vocab,
             beam_size=4,
             max_caption_length=20,
             length_normalization_factor=0.0):
```

tell model as shown. Below we will consider a few examples of the captions generated for the provided images. In section 3 we will will provide sample scores corresponding to the same images and captions as well as summary scores whole dataset and, of course, the code explanation.

References:
000000030429.jpg what is the color of the leaves orange
000000030429.jpg what did the large leaf fill sitting in a park tree
Hypotheses:
000000030429.jpg a park bench sitting in the middle of a park
000000030429.jpg a park bench sitting in the middle of a forest

000000030429.jpg a park bench sitting in the middle of a lush green park
000000030429.jpg a park bench sitting in the middle of a field
000000030429.jpg



For this sample the

3. Report accuracy in BLEU, CIDER, METEOR and ROGUE

In the process of attempting to compute the scores for these metrics using the code found at the following repositories I encountered many errors:

- https://github.com/Maluuba/nlg-eval
- https://github.com/salaniz/pycocoevalcap
- https://github.com/tylin/coco-caption/tree/master/pycocoevalcap
- https://github.com/gcunhase/NLPMetrics

Most commonly I encountered the below error in associate with the calculation of Meteor scores:

```
   self.meteor_p.stdin.flush()
OSError: [Errno 22] Invalid argument
```

I did eventually find a work-around to use the code found in these repositories calculating all but Meteor scores. The workaround is:

```
from nlgeval import NLGEval
...
   nlgeval = NLGEval(metrics_to_omit={'METEOR'})
```

However in hopes a more robust scheme for calculating the scores, and in hopes of being able to calculate the meteor score as well, I set out to calculate these scores individually.

In order to calculate the scores we must first load the data in a format which can be passed to the functions which do the calculation. We are reading in the data from files output by caption generation and in the process of filtering images which take the following format:

hyp.txt:

```
...
000000030429.jpg      a park bench sitting in the middle of a park
000000030429.jpg      a park bench sitting in the middle of a forest
000000030429.jpg      a park bench sitting in the middle of a lush green park
000000030429.jpg      a park bench sitting in the middle of a field
...
```

ref.txt:

```
...
000000030429.jpg      what is the color of the leaves orange
000000030429.jpg      what did the large leaf fill sitting in a park tree
...
```

Here hyp.txt contains the hypotheses (captions) we generated, and ref.txt contains the reference samples. Note, as mentioned in section 1, the reference samples are of the form “question”+“ ”+“answer” and hence are unlikely to be very similar to the hypotheses. We want this data ultimately to take the format of three python lists. In order to show the structure we wish these lists to have consider the below example:

Given the following hyp.txt and ref.txt:
hyp.txt:

```
001.jpg       a b c
002.jpg       e f g
002.jpg       e f g g
003.jpg       i j k i
```

ref.txt:

```
001.jpg       a b c
002.jpg       e f g h
003.jpg       i j k l
003.jpg       i j k m n
```

We want ‘ids’, ‘hyp’, and ‘ref’ as follows:
‘ids’:

```
[1, 2, 2, 3, 3]
```

‘hyp’:

```
['a b c', 'e f g', 'e f g g', 'i j k i', 'i j k i']
```

‘ref’:

```
['a b c', 'e f g h', 'e f g h', 'i j k l', 'i j k n m']
```

Essentially what we are doing here is for every id, we want to compare each hypothesis with each reference. With the lists we have constructed here we can do that by simply calling ‘compute_scores(hyp[i], ref[i])’, because each row of ‘hyp’ and ‘ref’ correspond to one another insofar as they are pair which we want to compare. The idiosyncrasies of the metrics complicate this process a little bit, but having our data in lists of following this format is very helpful. Actually constructing lists of following this format is fairly complicated; the code to do so is in ‘Compute_Scores_All.py’, lines 13 - 163. Below is a pseudocod summary:

```
hyp_lines = open(hyp_path, "r").readlines()
hypotheses = [""] * len(hyp_lines)                          # Store Hypotheses
hyp_id = [""] * len(hyp_lines)                              # and corresponding ids
for index, line in enumerate(hyp_lines):
   split_line = line.replace("\n","").split("\t")
   hyp_id[index] = filename_to_id(split_line[0]
   hypotheses[index] = split_line[1]
ref_lines = open(ref_path, "r").readlines()
```

```
references = [""] * len(ref_lines)                          # Store references
ref_id = [""] * len(ref_lines)                              # and corresponding ids
for index, line in enumerate(ref_lines):
   split_line = line.replace("\n","").split("\t")
   ref_id[index] = filename_to_id(split_line[0]

unique_ids = integer_sort(list(set(hyp_id) & set(ref_id))
num_uids = len(unique_ids )

# Calculate and store the multiplicities of unique ids in 'hyp_id'
mult_hyps = [0]*num_uids
for i, id in enumerate(unique_ids):
   num_hyps_per_id = 0
   for j in range(num_hyps):
       if id == hyp_id[j]:
           num_hyps_per_id += 1
   mult_hyps[i] = num_hyps_per_id

# Calculate and store the multiplicities of unique ids in 'ref_id'
mult_refs = [0]*num_uids
for i, id in enumerate(unique_ids):
   num_refs_per_id = 0
   for j in range(num_refs):
       if id == ref_id[j]:
           num_refs_per_id += 1
   mult_refs[i] = num_refs_per_id

# Using the above arrays we calculate the multiplicity of each
# unique id in our final 'ids' array
nums_ids = [mult_cur_ref*mult_cur_hyp for mult_cur_ref, mult_cur_hyp in
zip(mult_hyps,mult_refs)]
num_pairs = sum(nums_ids)
```

Up to this point we have loaded the hypothesis and references and calculated the multiplicity with which samples with the same ids appear among these hypothese and reference. These multiplicies are used in creating the lists. ‘ids’:

```
ids = [""]*num_pairs
j = 0
for i, id in enumerate(unique_ids):
   for k in range(nums_ids[i]):
       ids[j+k] = id
   j += nums_ids[i]

print(ids)
```

‘hyp’:

```
# Create list 'hyp' to store hypotheses to be used for score calculation
hyp = [""]*num_pairs
```

```
cur_index = 0
j = 0
prevID = hyp_id[0]
for i, cur_hyp in enumerate(hypotheses):
   if(hyp_id[i]!=prevID):
       j+=1
   cur_itr_mlt = nums_ids[j]
   if (cur_itr_mlt > 0):
       if mult_refs[j] > 1:  # We know that mult_refs[j] >= 1
           for k in range(mult_refs[j]):  # If mult_refs[j] > 1 add the current reference
               hyp[cur_index] = cur_hyp
               cur_index += 1
       else:
           hyp[cur_index] += cur_hyp  # Otherwise add the current hypothesis once
           cur_index += 1
       prevID = hyp_id[i]
       cur_itr_mlt -= mult_refs[j]
```

‘ref’:

```
# Create list 'ref' to store references to be used for score calculation
ref = [""]*num_pairs
cur_index = 0
j = 0
prevID = ref_id[0]
for i, cur_ref in enumerate(references):
   if (ref_id[i] != prevID):
       j += 1
   cur_itr_mlt = nums_ids[j]
   if (cur_itr_mlt > 0):
       if mult_hyps[j] > 1:  # We know that mult_hyps[j] >= 1
           for k in range(mult_hyps[j]):  # If mult_hyps[j] > 1 add the current reference
               ref[cur_index] = cur_ref  # k = mult_hyps[j] times
               cur_index += 1
       else:
           ref[cur_index] += cur_ref  # Otherwise add the current reference once
           cur_index += 1
       prevID = ref_id[i]
       cur_itr_mlt -= mult_hyps[j]
```

Finally we can begin calculating the scores.

For the Bleu scores there is a module in python which can be used. In order to do so we simply import the module and call the functions provided:

```
from nltk.translate.bleu_score import sentence_bleu
...
num_compares = len(ids)
for i in range(num_compares):
   cur_ref = [ref[i].split(" ")]
   cur_hyp = hyp[i].split(" ")
       bleu_1_scores[i] = sentence_bleu(cur_ref, cur_hyp, weights=(1, 0, 0, 0))
       bleu_2_scores[i] = sentence_bleu(cur_ref, cur_hyp, weights=(0, 1, 0, 0))
```

```
        bleu_3_scores[i] = sentence_bleu(cur_ref, cur_hyp, weights=(0, 0, 1, 0))
        bleu_4_scores[i] = sentence_bleu(cur_ref, cur_hyp, weights=(0, 0, 0, 1))
        bleu_C_scores[i] = sentence_bleu(cur_ref, cur_hyp)
```

I include the definition of 'num_compares' as it will be used to create score lists for subsequent metrics as well. The result of this code is that we have 5 lists, one for each bleu score

For CIDEr scores we use the following code:

```
from cider.cider import Cider
...
hyp_dict = {}
ref_dict = {}

for i, id in enumerate(ids):
   hyp_dict[id+str(i)] = [hyp[i]]
   ref_dict[id+str(i)] = [ref[i]]

cider_eval = Cider()
cider_score = cider_eval.compute_score(ref_dict, hyp_dict)
cider_scores_indiv = cider_score[1].tolist()
```

Notably we are utilizing the code about which I earlier complained, found at the linked repositories. For CIDEr score calculation this code executes without issue, so in my code I have used it.

For Meteor scores, we have had to get creative. Specifically we define the following java class and then call its functions using 'py4j':

```
import edu.cmu.meteor.scorer.MeteorConfiguration;
import edu.cmu.meteor.scorer.MeteorScorer;
import edu.cmu.meteor.util.Constants;
import py4j.GatewayServer;

public class Meteor {

   MeteorScorer scorer;
   MeteorConfiguration config;

   public Meteor(){
       config = new MeteorConfiguration();
       config.setLanguage("en");
       config.setNormalization(Constants.NORMALIZE_KEEP_PUNCT);
       scorer = new MeteorScorer(config);
   }

   public double compute_score(String hyp, String ref){
       return scorer.getMeteorStats(hyp, ref).score;
   }

   public static void main(String[] args) {
       Meteor app = new Meteor();
```

```
        GatewayServer server = new GatewayServer(app);
        server.start();
    }
```

The segment of Compute_Scores_All.py which calls this function is as follows:

```
from py4j.java_gateway import JavaGateway
...
gateway = JavaGateway()
meteor_eval = gateway.entry_point

meteor_scores = [0.0]*num_compares
for i in range(num_compares):
    meteor_scores[i] = meteor_eval.compute_score(hyp[i],ref[i])
```

In order for this to work we must first run the main function of the java class and then run our 'Compute_Scores_All.py' script.

Finally, and thankfully, Rouge scores can also be calculated as shown:

```
import rouge
...
rouge_eval = rouge.Rouge()
rouge_scores_dict_list = rouge_eval.get_scores(hyp,ref)
print(rouge_scores_dict_list)
rouge_1_scores = [0.0]*num_compares
rouge_2_scores = [0.0]*num_compares
rouge_l_scores = [0.0]*num_compares

for i, s in enumerate(rouge_scores_dict_list):
    if isinstance(s, dict):
        for metric, vals in s.items():
            print(metric)
            print(vals)
            if metric == 'rouge-1':
                rouge_1_scores[i] = vals.get('f')
            if metric == 'rouge-2':
                rouge_2_scores[i] = vals.get('f')
            if metric == 'rouge-l':
                rouge_l_scores[i] = vals.get('f')
```

Again, for Rouge there is a python module, though it must be installed.
Finally, after computing all these scores, we output them to 'output.csv' using the following code segment:

```
out_data = zip(ids, hyp, ref, bleu_C_scores, bleu_1_scores, bleu_2_scores, bleu_3_scores,
bleu_4_scores, cider_scores_indiv, meteor_scores, rouge_1_scores, rouge_2_scores,
rouge_l_scores)
output = list(out_data)
out_head = ["id", "hypothesis", "reference","bleu-C", "bleu-1", "bleu-2", "bleu-3",
"bleu-4", "CIDEr", "Meteor", "Rouge-1", "Rouge-2", "Rouge-l"]
with open('output.csv', 'w+', newline='') as out_file:
```

```
    csvWriter = csv.writer(out_file, delimiter = ',')
    csvWriter.writerow(out_head)
    csvWriter.writerows(output)
```

Using the resulting csv file I have calculated average scores for each metric. Additionally I have included some simplified rows of that 'output.csv' as examples. NB: CIDEr scores range 0 to 10 (others are 0 to 1)

bleu-C	bleu-1	bleu-2	bleu-3	bleu-4	CIDEr	Meteor	Rouge-1	Rouge-2	Rouge-l
0.01061	0.23454	0.04735	0.01522	0.00655	0.49026	0.12184	0.28136	0.05282	0.21703

For the example we considered in section 2 there are 8 rows of our final 'output.csv' file. The below two tables give three of the 8 reference pairs as well as a selection of the scores for those 3 pairs:

hypothesis	reference
a park bench sitting in the middle of a field	what did the large leaf fill sitting in a park tree
a park bench sitting in the middle of a lush green park	what did the large leaf fill sitting in a park tree
a park bench sitting in the middle of a forest	what is the color of the leaves orange

Considering the major difference between the form of our reference samples and the form of our hypotheses, namely that the former are questions with answers, and the latter are statements, it is unsurprising that the average scores and individual scores are fairly low.

bleu-1	bleu-2	CIDEr	Meteor	Rouge-1	Rouge-2	Rouge-l
0.45241	0.20107	1.26038	0.17008	0.5	0.21052	0.29417
0.41666	0.18181	0.93866	0.16399	0.47619	0.19047	0.37923
0.2	2.23E-308	0.06672	0.05369	0.25	0	0.24253