2019-09-21__explore-persona-model

September 27, 2019

```
In [7]: import os
        import glob
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        import re
        import pyperclip
        import numpy as np
```

1 Visualize Input Data

```
In [213]: import itertools
          import sys
          sys.path.append('../models/ACL2013_Personas/preprocess/')
          import coreproc
          from importlib import reload
          from IPython.display import clear_output
          from IPython.display import HTML, display
          import matplotlib
          reload(coreproc)
          color_wheel = list(matplotlib.colors.cnames.values())
```

```
In [349]: agent_v_color = color_wheel[12]
          patient_v_color = color_wheel[1]
          mod_color = color_wheel[2]
          entity_color = color_wheel[3]
          def html_replace_list(idx_list, sentence, sent_toks, color):
              for idx in idx_list:
                  word = sent_toks[idx]
                  sentence = sentence.replace(' %s ' % word, ' <span style="background-color: ' +  colo
              return sentence
```

```
In [63]: batch_rows = open('../models/ACL2013_Personas/preprocess/batch-0.ss').read().strip().split('\n
         split_rows = list(map(lambda x: x.strip().split('\t'), batch_rows))
```

```
In [260]: idx = 0
          for docid, rows in itertools.groupby(split_rows, key=lambda r: r[0]):
              rows = list(rows)
              if idx > 5:
                  break
              idx += 1
```

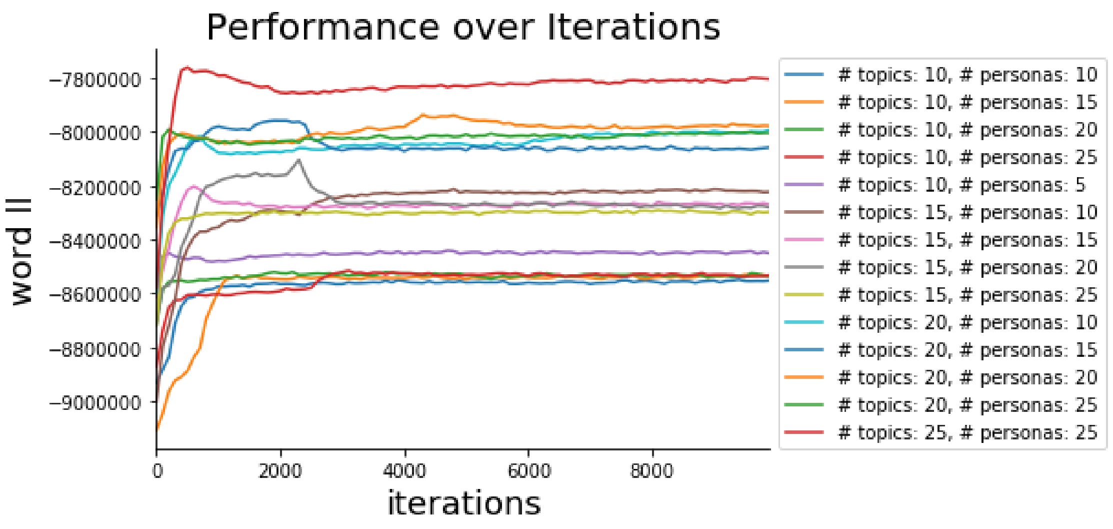
```
In [261]: processed = coreproc.process_full_doc(rows)
In [303]: agents_df = pd.DataFrame(processed['agents'])
          patients_df = pd.DataFrame(processed['patients'])
          mod_df = pd.DataFrame(processed['modifiers'])
          entities_df = pd.DataFrame(processed['entities'])
          sentence_df = pd.DataFrame(processed['sentences'])

In [352]: col_list = [("agent verb color", agent_v_color),
          ("patient verb color", patient_v_color),
          ("modifier color", mod_color),
          ("entity color", entity_color),]
          legend = []
          for name, col in col_list:
              line = name + ': <span style="background-color: ' +  col + '">&nbsp;&nbsp;&nbsp;&nbsp;</s
              legend.append(line)

In [397]: # patients_df.head()
          # mod_df.head()
          # agents_df.head()
          # entities_df.head()

In [393]: hex_to_rgb_tup = lambda h: tuple(int(h.replace('#','')[i:i+2], 16) for i in (0, 2, 4))
          make_rgb = lambda h, a: 'rgba' + str(hex_to_rgb_tup(h) + (a,))
          rand_colors = np.random.choice(color_wheel, len(entities_df['entity-id'].unique()), replace=F

In [395]: sentences = []
          for idx, sentence, sentence_id in sentence_df.itertuples():
              sent_toks = sentence.split()
              ent_id2_entidx = (entities_df
                .loc[lambda df: df['sentence-idx'] == int(sentence_id.replace('S', ''))]
                .groupby('entity-id')['entity-idx']
                .aggregate(list)
              )
              for ent_id, entidx_list in ent_id2_entidx.iteritems():
                  sentence = html_replace_list(entidx_list, sentence, sent_toks, make_rgb(rand_colors[e
              sentences.append(sentence)

          HTML('<h3>Co-referenced Entities</h3><br><br>' + ' '.join(sentences))

Out[395]: <IPython.core.display.HTML object>
```

1.0.1 Entites + Agent, Patient, Modifiers

```
In [379]: ## apply labels
          labels = []
          htmls = []
          for idx, sentence in enumerate(processed['sentences']):
              s_id, sent = sentence.values()
              sent_toks = sent.split()
              s_id = int(s_id.replace('S', ''))

              ## replace entities
              s_entities = entities_df.loc[lambda df: df['sentence-idx'] == s_id]
              sent = html_replace_list(s_entities['entity-idx'], sent, sent_toks, entity_color)

              ## replace agent
              s_agents = agents_df.loc[lambda df: df['sentence-idx'] == s_id]
              sent = html_replace_list(s_agents['verb-idx'], sent, sent_toks, agent_v_color)

              ## replace patient
              s_patients = patients_df.loc[lambda df: df['sentence-idx'] == s_id]
              sent = html_replace_list(s_patients['verb-idx'], sent, sent_toks, patient_v_color)

              ## replace modifier
              s_mod = mod_df.loc[lambda df: df['sentence-idx'] == s_id].loc[lambda df: df['relation'] !
              sent = html_replace_list(s_mod['mod-idx'], sent, sent_toks, mod_color)

              if idx ==0:
                  html = '<br>'.join(legend) + "<br><br>" + sent.replace('$', '')
              else:
                  html = sent.replace('$', '')

              htmls.append(html)
          display(HTML(' '.join(htmls)))
          #     label = input()
          #     labels.append(label)
          #     clear_output() ## clear ipython output

<IPython.core.display.HTML object>
```

2 Model Performance

```
In [418]: slurms = glob.glob('../models/ACL2013_Personas/output/slurm-4*')
          wlls = {}
          all_names = []
          for slurm in slurms:
              lines = open(slurm).read().split('\n')
              name = list(filter(lambda x: 'output dir' in x, lines))

              ###
              if name and 'runreg-false' in name[0]:
                  name = name[0].replace('output dir: ', '')
                  all_names.append(name)
                  params2lines[name] = lines
                  ##
                  wll = list(filter(lambda x: 'wordLL' in x, lines))
                  if len(wll) > 50:
                      wlls[name] = max(wlls.get(name, []), wll, key=lambda x: len(x))

In [424]: for key, lines in sorted(wlls.items(), key=lambda x: x[0]):
              wll = list(map(lambda x: x.split()[-1], lines))
              t = pd.Series(wll).astype(float)
              t.index = t.index * 100
              k, p = re.findall('numtopics-(\d+).numpersonas-(\d+)', key)[0]
              ax = t.plot(label='# topics: %s, # personas: %s' % (k, p))
              ax.spines['right'].set_visible(False)
              ax.spines['top'].set_visible(False)
              plt.legend(bbox_to_anchor=(1,1))
              plt.ylabel('word ll', fontdict={"size": 18})
              plt.xlabel('iterations', fontdict={"size": 18})
              plt.title("Performance over Iterations", fontdict={"size": 20})
```



3 Look at topic outputs

```
In [11]: import glob
         import pandas as pd
         fs = glob.glob('../models/ACL2013_Personas/output/newspapers.numtopics-20.numpersonas-10.runre

In [12]: ## K = num topics
         ## P = num personas
         ## C = num characters total
         ## D = num documents
         ## V = len vocab

         ## character conditional file = 3 * C x K
         ## character posterior file = C x P
         ## agents_file = P X K
         ## mod_file = P x K
         ## patients_file = P x K
         ## phi_file = K x V

         ## DONT USE:
         ## -------------------------------------------------------------------------------------
         ## featfile = unused
         ## persona_file = P x K * 3 ### each row: agents, patients, mod... equal to the agents_file, p


         characterConditionalFile, characterPosteriorFile, featfile, agents_file, mod_file, patients_fi

In [13]: t1 = open(characterConditionalFile, encoding='utf-8').read().split('\n')
         character_cond_posterior_df = pd.DataFrame(
             list(map(lambda x: x.split('\t'), t1)),
             columns=["type", "entity_id", "doc_id", "title", "charName", "fullName", "NumEvents", "max_
         )
```

```
In [14]: character_cond_posterior_df.head()
```

```
Out[14]:       type entity_id                                doc_id title    charName  \
         0    agent        e9  1dab5d59-c916-11e9-a6c2-b831b5755f6c   ???  Government
         1  patient        e9  1dab5d59-c916-11e9-a6c2-b831b5755f6c   ???  Government
         2      mod        e9  1dab5d59-c916-11e9-a6c2-b831b5755f6c   ???  Government
         3    agent       e10  1dab5d59-c916-11e9-a6c2-b831b5755f6c   ???    Congress
         4  patient       e10  1dab5d59-c916-11e9-a6c2-b831b5755f6c   ???    Congress

                  fullName NumEvents max_samples  \
         0  The Government         5          10
         1  The Government         5           0
         2  The Government         5          18
         3        Congress         6          10
         4        Congress         6           0

                                                    topic_dist
         0  0.00000 0.00000 0.00000 6.00000 0.00000 4.0000...
         1  0.00000 0.00000 0.00000 0.00000 0.00000 0.0000...
         2  0.00000 0.00000 0.00000 0.00000 0.00000 0.0000...
         3  0.00000 0.00000 0.00000 3.00000 0.00000 9.0000...
         4  0.00000 0.00000 0.00000 0.00000 0.00000 0.0000...
```

```
In [15]: t2 = open(characterPosteriorFile, encoding='utf-8').read().split('\n')
         characer_posterior_df = pd.DataFrame(
             list(map(lambda x: x.split('\t'), t2)),
             columns=["entity_id", "doc_id", "title", "charName", "fullName", "NumEvents", "max_finalPos
         )
```

3.0.1 map topics

```
In [18]: import numpy as np
         from scipy.stats import entropy
         from IPython.display import HTML
```

```
In [19]: ## read in vocabulary
         vocab_text = open(phi).read()
         vocab_mat = list(filter(lambda x: x!=[], map(lambda x: x.split(), vocab_text.split('\n'))))
         vocab = vocab_mat[0]
         vocab_dat = vocab_mat[1:]
         word_topics = pd.DataFrame(vocab_dat, columns=vocab).astype(float)

         ## get top words by topic
         top_words_by_topic = []
         for i in range(word_topics.shape[0]):
             sorted_topics = word_topics.loc[i].sort_values(ascending=False).head(10)
             top_words_by_topic.append(list(sorted_topics.index))

         ## reformat into an HTML table for easier reading
         ncols = 10
         table = []
         row = []
```

```
        for topic, words in enumerate(top_words_by_topic):
            subtable_header = '<th>topic: %s</th>' % topic
            subtable_body = ''.join(list(map(lambda elem: '<tr><td>%s</td></tr>' % elem, words)))
            subtable = '<table><tr>%s</tr>%s</table>' % (subtable_header, subtable_body)
            row.append(subtable)
            if (topic % ncols == (ncols - 1)) or (topic == len(top_words_by_topic) - 1) :
                table.append('\n'.join(list(map(lambda elem: '<td>%s</td>' % elem, row))))
                row = []
```

```
In [20]: HTML('<table>' + ''.join(list(map(lambda row: '<tr>%s</tr>' % row, table))) + '</table>')
```

```
Out[20]: <IPython.core.display.HTML object>
```

```
In [21]: mod_output = open(mod_file).read().strip().split('\n')
         mod_mat = list(map(lambda x: x.split(), mod_output))
         mod_header = list(map(lambda elem: 'topic %s' % elem, mod_mat[0]))
         mod_dat = mod_mat[1:]

         mod_df = pd.DataFrame(mod_dat, columns=mod_header).astype(float)
         mod_df.index = list(map(lambda x: 'persona %d' % x, mod_df.index))
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [44]: # mainparts = input_data[0].strip().split('\t')
         # movieID = mainparts[0].strip();
         # agentLine = mainparts[1].strip();
         # patientLine = mainparts[2].strip();
         # modifierLine = mainparts[3].strip();
         # entityNameString = mainparts[4].strip();
         # entityFullNameString = mainparts[5].strip();

         # entitiesByEID = {}

         # doc = {}
         # doc['id'] = movieID;

         # entitiesByEID = {}
         # # Read entity head lemmas
         # head_lemmas = json.loads(entityNameString)
         # for e, name in head_lemmas.items():
         #     e = e.lower()
         #     entity = entitiesByEID.get(e, {})
         #     entity['name'] = name
         #     entitiesByEID[e] = entity

         # # Read entity full text mentions
         # full_text_mention = json.loads(entityFullNameString)
         # for e, name in full_text_mention.items():
         #     e = e.lower()
         #     entity = entitiesByEID.get(e, {})
         #     entity['full_name'] = name
         #     entitiesByEID[e] = entity

         # # Read the tuple predarg fragments
         # eventsByTID = {}
         # readTupleArgs("AGENT", agentLine, entitiesByEID, eventsByTID)
         # readTupleArgs("PATIENT", patientLine, entitiesByEID, eventsByTID)
         # readTupleArgs("MODIFIER", modifierLine, entitiesByEID, eventsByTID)

         # def readTupleArgs(role, argLine, entitiesByEID, eventsByTID):
         #     parts = argLine.split(" ")
         #     for word in parts:
         #         wparts = word.split(":")
         #         if len(wparts) > 3:
         #             eKey = wparts[0].lower()
         #             tupleID = wparts[1]
         #             supertag = wparts[2]
         #             verb = wparts[3]

         #             entitiesByEID[eKey][role] = entitiesByEID[eKey].get(role, []) + [verb]
         #             if tupleID not in eventsByTID:
         #                 eventsByTID[tupleID] = (tupleID, verb)
```