Creating Gold Annotation Labels with BRAT

This is a short tutorial on how to use BRAT (Brat Rapid Annotation Tool), an online environment for collaborative text annotation.

http://brat.nlplab.org/

In [ ]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import os
import numpy as np

# Connect to the database backend and initalize a Snorkel session
from lib.init import *
```

Step 1: Define a `Candidate` Type

In [ ]:
```
Spouse = candidate_subclass('Spouse', ['person1', 'person2'])
```

a) Select an example `Candidate` and `Document`

Candidates are divided into 3 splits mapping to a unique integer id:

- 0: *training*
- 1: *development*
- 2: *testing*

In this tutorial, we'll load our training set candidates and create gold labels for a document using the BRAT interface

Step 2: Launching BRAT

BRAT runs as as seperate server application. When you first initialize this server, you need to provide your applications `Candidate` type. For this tutorial, we use the `Spouse` relation defined above, which consists of a pair of `PERSON` named entities connected by marriage.

Currently, we only support 1 relation type per-application.

In [ ]:
```
from snorkel.contrib.brat import BratAnnotator

brat = BratAnnotator(session, Spouse, encoding='utf-8')
```

a) Initialize our document collection

BRAT creates a local copy of all the documents and annotations found in a `split` set. We initialize or document collection by passing in a set of candidates via the `split` id. Annotations are stored as plain text files in standoff format.

After launching the BRAT annotator for the first time, you will need to login to begin editing annotations. Navigate your mouse to the upper right-hand corner of the BRAT interface (see Fig. 1) click 'login' and enter the following information:

- **login**: *brat*
- **password**: *brat*

Advanced BRAT users can setup multiple annotator accounts by adding USER/PASSWORD key pairs to the `USER_PASSWORD` dictionary found in `snokel/contrib/brat/brat-v1.3_Crunchy_Frog/config.py` . This is useful if you would like to keep track of multiple annotator judgements for later adjudication or use as labeling functions as per our tutorial on using Snorkel for Crowdsourcing.

In [ ]:
```
brat.init_collection("spouse/train", split=0)
```

We've already generated some BRAT annotations, so import and existing collection for purposes of this tutorial.

In [ ]:
```
brat.import_collection("data/brat-spouse.zip", overwrite=True)
```

b) Launch BRAT Interface in a New Window

Once our collection is initialized, we can view specific documents for annotation. The default mode is to generate a HTML link to a new BRAT browser window. Click this link to connect to launch the annotator editor.

In [ ]:
```
doc_name = '5ede8912-59c9-4ba9-93df-c58cebb542b7'
doc = session.query(Document).filter(Document.name==doc_name).one()

brat.view("spouse/train", doc)
```

If you do not have a specific document to edit, you can optionally launch BRAT and use their file browser to navigate through all files found in the target collection.

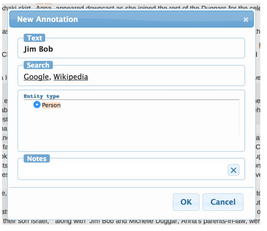
In [ ]:
```
brat.view("spouse/train")
```

Step 3: Creating Gold Label Annotations

a) Annotating Named Entities

`Spouse` relations consist of 2 `PERSON` named entities. When annotating our validation documents, the first task is to identify our target entities. In this tutorial, we will annotate all `PERSON` mentions found in our example document, though for your application you may choose to only label those that particpate in a true relation.

Begin by selecting and highlighting the text corresponding to a `PERSON` entity. Once highlighted, an annotation dialog will appear on your screen (see image of the BRAT Annotation Dialog Window to the right). If this is correct, click ok. Repeat this for every entity you find in the document.



Annotation Guidelines

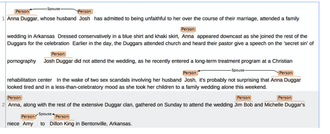
When developing gold label annotations, you should always discuss and agree on a set of *annotator guidelines* to share with human labelers. These are the guidelines we used to label the `Spouse` relation:

- **Do not** include formal titles associated with professional roles e.g., ***Pastor** Jeff*, ***Prime Minister** Prayut Chan-O-Cha*
- Do include English honorifics unrelated to a professional role, e.g., ***Mr.** John Cleese*.
- **Do not** include family names/surnames that do not reference a single individual, e.g., *the Duggar family*.
- Do include informal titles, stage names, fictional characters, and nicknames, e.g., ***Dog the Bounty Hunter***
- Include possessive's, e.g., *Anna**'s***.

b) Annotating Relations

To annotate `Spouse` relations, we look through all pairs of `PERSON` entities found within a single sentence. BRAT identifies the bounds of each sentence and renders a numbered row in the annotation window (see the left-most column in the image below).

Annotating relations is done through simple drag and drop. Begin by clicking and holding on a single `PERSON` entity and then drag that entity to its corresponding spouse entity. That is it!



Annotation Guidelines

- Restrict `PERSON` pairs to those found in the same sentence.
- The order of `PERSON` arguments does not matter in this application.
- **Do not** include relations where a `PERSON` argument is wrong or otherwise incomplete.

Step 4: Scoring Models using BRAT Labels

a) Evaluating System Recall

Creating gold validation data with BRAT is a critical evaluation step because it allows us to compute an estimate of our model's *true recall*. When we create labeled data over a candidate set created by Snorkel, we miss mentions of relations that our candidate extraction step misses. This causes us to overestimate the system's true recall.

In the code below, we show how to map BRAT annotations to an existing set of Snorkel candidates and compute some associated metrics.

In [ ]:
```
train_cands = session.query(Candidate).filter(Candidate.split==0).all()
```

b) Mapping BRAT Annotations to Snorkel Candidates

We annotated a single document using BRAT to illustrate the difference in scores when we factor in the effects of candidate generation.

In [ ]:
```
%time brat.import_gold_labels(session, "spouse/train", train_cands)
```

Our candidate extractor only captures 7/14 (50%) of true mentions in this document. Our real system's recall is likely even worse, since we won't correctly predict the label for all true candidates.

c) Re-loading the Trained LSTM

We'll load the LSTM model we trained in Workshop_4_Discriminative_Model_Training.ipynb and use to to predict marginals for our test candidates.

In [ ]:
```
test_cands = session.query(Spouse).filter(Spouse.split == 2).order_by(Spouse.id).all()
```

In [ ]:
```
from snorkel.learning.disc_models.rnn import reRNN

lstm = reRNN(seed=1701, n_threads=None)
lstm.load("spouse.lstm")
```

In [ ]:
```
marginals = lstm.marginals(test_cands)
```

d) Create a Subset of Test for Evaluation

Our measures assume BRAT annotations are complete for the given set of documents! Rather than manually annotating the entire test set, we define a small subset of 10 test documents for hand lableing. We'll then compute the full, recall-corrected metrics for this subset.

First, let's build a query to initalize this candidate collection.

In [ ]:
```
doc_ids = set(open("data/brat_test_docs.tsv","rb").read().splitlines())
cid_query = [c.id for c in test_cands if c.get_parent().document.name in doc_ids]

brat.init_collection("spouse/test-subset", cid_query=cid_query)
```

In [ ]:
```
brat.view("spouse/test-subset")
```

e) Comparing Unadjusted vs. Adjusted Scores

In [ ]:
```
import matplotlib.pyplot as plt
plt.hist(marginals, bins=20)
plt.show()
```

In [ ]:
```
from snorkel.annotations import load_gold_labels

L_gold_dev  = load_gold_labels(session, annotator_name='gold', split=1, load_as_array=True, zero_one=True)
L_gold_test = load_gold_labels(session, annotator_name='gold', split=2, zero_one=True)
```

In [ ]:
```
tp, fp, tn, fn = lstm.error_analysis(session, test_cands, L_gold_test)
```

In [ ]:
```
brat.score(session, test_cands, marginals, "spouse/test-subset")
```

In [ ]:
```
brat.score(session, test_cands, marginals, "spouse/test-subset", recall_correction=False)
```

In [ ]:
```

```