Intro. to Snorkel: Extracting Spouse Relations from the News

Part III: Training an End Extraction Model

In this final section of the tutorial, we'll use the noisy training labels we generated in the last tutorial part to train our end extraction model.

For this tutorial, we will be training a Bi-LSTM, a state-of-the-art deep neural network implemented in TensorFlow.

In [1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import os

# TO USE A DATABASE OTHER THAN SQLITE, USE THIS LINE
# Note that this is necessary for parallel execution amongst other things...
# os.environ['SNORKELDB'] = 'postgres:///snorkel-intro'

from snorkel import SnorkelSession
session = SnorkelSession()
```

We repeat our definition of the `Spouse` `Candidate` subclass:

In [2]:
```
from snorkel.models import candidate_subclass

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

We reload the probabilistic training labels:

In [3]:
```
from snorkel.annotations import load_marginals

train_marginals = load_marginals(session, split=0)
```

We also reload the candidates:

In [4]:
```
train_cands = session.query(Spouse).filter(Spouse.split == 0).order_by(Spouse.id).all()
dev_cands   = session.query(Spouse).filter(Spouse.split == 1).order_by(Spouse.id).all()
test_cands  = session.query(Spouse).filter(Spouse.split == 2).order_by(Spouse.id).all()
```

Finally, we load gold labels for evaluation:

In [5]:
```
from snorkel.annotations import load_gold_labels

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

Now we can setup our discriminative model. Here we specify the model and learning hyperparameters.

They can also be set automatically using a search based on the dev set with a GridSearch object.

In [6]:
```
from snorkel.learning.pytorch import LSTM

train_kwargs = {
    'lr':            0.01,
    'embedding_dim': 50,
    'hidden_dim':    50,
    'n_epochs':      10,
    'dropout':       0.25,
    'seed':          1701
}

lstm = LSTM(n_threads=None)
lstm.train(train_cands, train_marginals, X_dev=dev_cands, Y_dev=L_gold_dev, **train_kwargs)
```

```
[LSTM] Training model
[LSTM] n_train=16004  #epochs=10  batch size=256
[LSTM] Epoch 1 (33.52s) Average loss=0.618173   Dev F1=0.00
[LSTM] Epoch 2 (70.08s) Average loss=0.602743   Dev F1=10.20
[LSTM] Epoch 3 (106.21s)        Average loss=0.595686   Dev F1=12.67
[LSTM] Epoch 4 (141.95s)        Average loss=0.590128   Dev F1=22.59
[LSTM] Epoch 5 (177.92s)        Average loss=0.586008   Dev F1=22.02
[LSTM] Epoch 6 (214.17s)        Average loss=0.585437   Dev F1=13.82
[LSTM] Epoch 7 (249.03s)        Average loss=0.584641   Dev F1=23.36
[LSTM] Epoch 8 (284.29s)        Average loss=0.581280   Dev F1=23.73
[LSTM] Epoch 9 (319.53s)        Average loss=0.579671   Dev F1=25.12
checkpoints/LSTM
[LSTM] Model saved as <LSTM>
[LSTM] Epoch 10 (355.27s)       Average loss=0.578497   Dev F1=25.00
[LSTM] Training done (357.58s)
[LSTM] Loaded model <LSTM>
```

Now, we get the precision, recall, and F1 score from the discriminative model:

In [7]:
```
p, r, f1 = lstm.score(test_cands, L_gold_test)
print("Prec: {0:.3f}, Recall: {1:.3f}, F1 Score: {2:.3f}".format(p, r, f1))
```

```
Prec: 0.197, Recall: 0.657, F1 Score: 0.303
```

We can also get the candidates returned in sets (true positives, false positives, true negatives, false negatives) as well as a more detailed score report:

In [8]:
```
tp, fp, tn, fn = lstm.error_analysis(session, test_cands, L_gold_test)
```

```
========================================
Scores (Un-adjusted)
========================================
Pos. class accuracy: 0.676
Neg. class accuracy: 0.883
Precision            0.213
Recall               0.676
F1                   0.324
----------------------------------------
TP: 73 | FP: 270 | TN: 2046 | FN: 35
========================================
```

Note that if this is the final test set that you will be reporting final numbers on, to avoid biasing results you should not inspect results. However you can run the model on your *development set* and, as we did in the previous part with the generative labeling function model, inspect examples to do error analysis.

You can also improve performance substantially by increasing the number of training epochs!

Finally, we can save the predictions of the model on the test set back to the database. (This also works for other candidate sets, such as unlabeled candidates.)

In [9]:
```
lstm.save_marginals(session, test_cands)
```

```
Saved 2424 marginals
```

More importantly, you completed the introduction to Snorkel! Give yourself a pat on the back!