Snorkel Workshop: Extracting Spouse Relations from the News

Part 3: Training the Generative Model

Now, we'll train a model of the LFs to estimate their accuracies. Once the model is trained, we can combine the outputs of the LFs into a single, noise-aware training label set for our extractor. Intuitively, we'll model the LFs by observing how they overlap and conflict with each other.

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

# Connect to the database backend and initalize a Snorkel session
from lib.init import *
from snorkel.models import candidate_subclass
from snorkel.annotations import load_gold_labels

from snorkel.lf_helpers import (
    get_left_tokens, get_right_tokens, get_between_tokens,
    get_text_between, get_tagged_text,
)

# initialize our candidate type definition
Spouse = candidate_subclass('Spouse', ['person1', 'person2'])

# gold (human-labeled) development set labels
L_gold_dev = load_gold_labels(session, annotator_name='gold', split=1)
```

I. Loading Labeling Matricies

First we'll load our label matrices from notebook 2

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

labeler = LabelAnnotator(lfs=[])
L_train = labeler.load_matrix(session, split=0)
L_dev   = labeler.load_matrix(session, split=1)
```

Now we set up and run the hyperparameter search, training our model with different hyperparamters and picking the best model configuration to keep. We'll set the random seed to maintain reproducibility.

Note that we are fitting our model's parameters to the training set generated by our labeling functions, while we are picking hyperparamters with respect to score over the development set labels which we created by hand.

II: Unifying supervision

A. Majority Vote bad!

The most simple way to unify the output of all your LFs is by computed the unweighted majority vote

```
from lib.scoring import *
majority_vote_score(L_dev,L_gold_dev)
```

```
pos/neg 190:2621 6.8%/93.2%
precisiion  31.96
recall  32.63
f1  32.29
```

Generative Model (job: de-noise)

In data programming, we use a more sophisitcated model to unify our labeling functions. We know that these labeling functions will not be perfect, and some may be quite low-quality, so we will *model* their accuracies with a generative model, which Snorkel will help us easily apply.

This will ultimately produce a single set of **noise-aware training labels**, which we will then use to train an end extraction model in the next notebook. For more technical details of this overall approach, see our NIPS 2016 paper.

NOTE: Make sure you've written some of your own LFs in the previous notebook to get a decent score!!!

1. Training the Model

When training the generative model, we'll tune our hyperparamters using a simple grid search.

Parameter Definitions

hyper-paras tuning

```
epochs     A single pass through all the data in your training set
step_size  The factor by which we update model weights after computing the gradient
decay      The rate our update factor dimishes (decay) over time.
```

In [ ]:
```
from snorkel.learning import GenerativeModel
from snorkel.learning import RandomSearch

# use random search to optimize the generative model
param_ranges = {
    'step_size' : [1e-3, 1e-4, 1e-5, 1e-6],
    'decay'     : [0.9, 0.95],
    'epochs'    : [50, 100],
    'reg_param' : [1e-3],
}

model_class_params = {'lf_propensity' : False}

searcher = RandomSearch(GenerativeModel, param_ranges, L_train, n=5, model_class_params=model_class_params)
%time gen_model, run_stats = searcher.fit(L_dev, L_gold_dev)

run_stats
```

deps= deps

set

	step_size	decay	epochs	Prec.	Rec.	F1
0	0.000004	0.95	50	0.348624	0.4	0.372549
1	0.000010	0.90	50	0.348624	0.4	0.372549
2	0.000004	0.95	50	0.348624	0.4	0.372549
3	0.000004	0.95	50	0.348624	0.4	0.372549

2. Model Accuracies

These are the weights learned for each LF

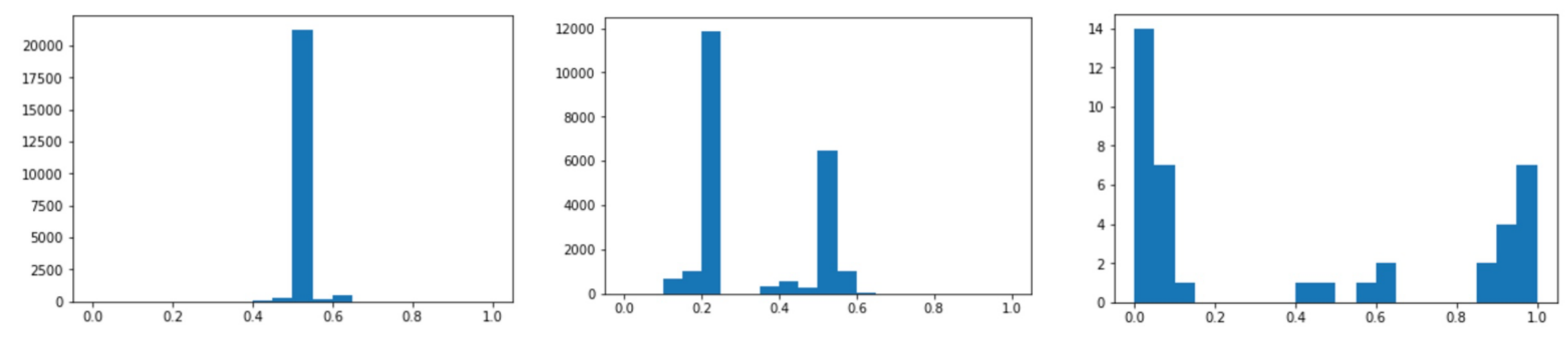
In [ ]:
```
x = L_dev.lf_stats(session, L_gold_dev)
```

In [ ]:
```
train_marginals = gen_model.marginals(L_train)
```

	j	Coverage	Overlaps	Conflicts	TP	FP	FN	TN	Empirical Acc.	Learnec Acc.
RMS_marriage_[between|words]_TRUE	0	0.071861	0.006403	0.006403	64	129	0	0	0.331606	0.55227C
ST_SUPERVISION_dbpedia_TRUE	1	0.009249	0.009249	0.009249	22	4	0	0	0.846154	0.547CC3
RMS_almost_married_[between|words]_FALSE	2	0.012451	0.002846	0.002846	0	0	7	24	0.774194	0.550307
_and_marriage	3	0.003557	0.003557	0.003557	10	0	0	0	1.000000	0.553265
ntrarian	4	0.009249	0.009249	0.009249	0	0	22	4	0.153846	0.546729

3. Plotting Marginal Probabilities

One immediate santity check you can peform using the generative model is to visually examine the distribution of predicted training marginals. Ideally, there should get a bimodal distribution with large seperation between each peaks, as shown below by the far right image. The corresponds to good signal for true and positive class labels. For your first Snorkel application, you'll probably see marginals closer to the far left or middle images. With all mass centered around p=0.5, you probably need to write more LFs got get more overall *coverage*. In the middle image, you have good negative coverage, but not enough positive LFs



In [ ]:
```
import matplotlib.pyplot as plt
plt.hist(train_marginals, bins=20, range=(0.0, 1.0))
plt.show()
```

4. Generative Model Metrics

In [ ]:
```
dev_marginals = gen_model.marginals(L_dev)
_, _, _, _ = gen_model.error_analysis(session, L_dev, L_gold_dev)
```

5. Saving our training labels

Finally, we'll save the `training_marginals`, which are our **"noise-aware training labels"**, so that we can use them in the next tutorial to train our end extraction model:

In [ ]:
```
from snorkel.annotations import save_marginals
%time save_marginals(session, L_train, train_marginals)
```

III. Advanced Generative Model Features

A. Structure Learning

We may also want to include the dependencies between our LFs when training the generative model. Snorkel makes it easy to do this! `DependencySelector` runs a fast structure learning algorithm over the matrix of LF outputs to identify a set of likely dependencies.

In [ ]:
```
from snorkel.learning.structure import DependencySelector

MAX_DEPS = 5

ds = DependencySelector()
deps = ds.select(L_train, threshold=0.1)
deps = set(list(deps)[0:min(len(deps), MAX_DEPS)])

print("Using {} dependencies".format(len(deps)))
```

tunable

← set of dependency

Now train the generative model with dependencies, we just pass in the above set as the `deps` argument to our model train function.

```
searcher = RandomSearch(GenerativeModel, param_grid, L_train, n=4, lf_propensity=False)
gen_model, run_stats = searcher.fit(L_dev, L_gold_dev, deps=deps)
run_stats
```

In [ ]: