Snorkel Workshop: Extracting Spouse Relations from the News

Advanced Part 6: Hyperparameter Tuning via Grid Search

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 *
```

We repeat our definition of the `Spouse` `Candidate` subclass, and load the test set:

In [ ]:

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

I. Training a `SparseLogisticRegression` Discriminative Model

We use the training marginals to train a discriminative model that classifies each `Candidate` as a true or false mention. We'll use a random hyperparameter search, evaluated on the development set labels, to find the best hyperparameters for our model. To run a hyperparameter search, we need labels for a development set. If they aren't already available, we can manually create labels using the Viewer.

Feature Extraction

Instead of using a deep learning approach to start, let's look at a standard sparse logistic regression model. First, we need to extract out features. This can take a while, but we only have to do it once!

In [ ]:

```
from lib.features import hybrid_span_mention_ftrs
from snorkel.annotations import FeatureAnnotator

featurizer = FeatureAnnotator(f=hybrid_span_mention_ftrs)
```

In [ ]:

```
F_train = featurizer.load_matrix(session, split=0)
F_dev   = featurizer.load_matrix(session, split=1)
F_test  = featurizer.load_matrix(session, split=2)

if F_train.size == 0:
    %time F_train = featurizer.apply(split=0, parallelism=1)
if F_dev.size == 0:
    %time F_dev  = featurizer.apply_existing(split=1, parallelism=1)
if F_test.size == 0:
    %time F_test = featurizer.apply_existing(split=2, parallelism=1)

print(F_train.shape)
print(F_dev.shape)
print(F_test.shape)
```

First, reload the training marginals:

In [ ]:

```
from snorkel.annotations import load_marginals
train_marginals = load_marginals(session, split=0)
```

In [ ]:

```
import matplotlib.pyplot as plt
plt.hist(train_marginals, bins=20)
plt.show()
```

Load our development data for tuning

In [ ]:

```
from snorkel.annotations import load_gold_labels

L_gold_dev = load_gold_labels(session, annotator_name='gold', split=1)
L_gold_dev.shape
```

The following code performs model selection by tuning our learning algorithm's hyperparamters. ***Note: This requires installing tensorflow:*** ***`conda install tensorflow`***.

In [ ]:

```
from snorkel.learning import RandomSearch
from snorkel.learning.tensorflow import SparseLogisticRegression

seed = 1234
num_model_search = 5

# search over this parameter grid
param_grid = {}
param_grid['batch_size'] = [64, 128]
param_grid['lr']         = [1e-4, 1e-3, 1e-2]
param_grid['l1_penalty'] = [1e-6, 1e-4, 1e-2]
param_grid['l2_penalty'] = [1e-6, 1e-4, 1e-2]
param_grid['rebalance']  = [0.0, 0.5]

model_class_params = {
    'n_threads':1
}

model_hyperparams = {
    'n_epochs': 30,
    'print_freq': 10,
    'dev_ckpt_delay': 0.5,
    'X_dev': F_dev,
    'Y_dev': L_gold_dev
}

searcher = RandomSearch(SparseLogisticRegression, param_grid, F_train, train_marginals,
                        n=num_model_search, seed=seed,
                        model_class_params=model_class_params,
                        model_hyperparams=model_hyperparams)

print("Discriminitive Model Parameter Space (seed={}):".format(seed))
for i, params in enumerate(searcher.search_space()):
    print("{} {}".format(i, params))

disc_model, run_stats = searcher.fit(X_valid=F_dev, Y_valid=L_gold_dev, n_threads=1)
run_stats
```

Examining Features

Extracting features allows us to inspect and interperet our learned weights

In [ ]:

```
from lib.scoring import *
print_top_k_features(session, disc_model, F_train, top_k=25)
```