CS274b Homework #4

Learning in Graphical Models: Spring 2016

Due: Friday, June 3, 2016

Write neatly (or type) and show all your work!

Please remember to turn in at most two documents, one with any handwritten solutions, and one PDF file with any electronic solutions.

In this problem, we will build a structured support vector machine (SSVM) model for part-of-speech tagging in text data. The data can be found in the `data` subdirectory of the zip file, with one file per sentence; the data and feature preprocessing are thanks to David Sontag of NYU. Each line of the file contains seven comma-separated values:

$$t_i^{(s)}\ ,\ y_i^{(s)}\ ,\ x_{i;0}^{(s)}\ ,\ \ldots\ ,\ x_{i;4}^{(s)}$$

where $t_i^{(s)}$ is the actual observed text for word i in sentence s (a string), $y_i^{(s)}$ is the POS tag for word i, and $x_{i;k}^{(s)}$ are four processed features of $t_i^{(s)}$ that we will use in the model:

$x_{i;0}^{(s)} = 1$	:	bias term; always 1
$x_{i;1}^{(s)} \in \{0,1\}$	:	$t_i^{(s)}$ begins with a capital letter
$x_{i;2}^{(s)} \in \{0,1\}$	:	$t_i^{(s)}$ is all captial letters
$x_{i;3}^{(s)} \in \{1,\ldots,201\}$	:	prefix token
$x_{i;4}^{(s)} \in \{1,\ldots,201\}$	:	suffix token

The prefix and suffix tokens $x_{i;3}$ and $x_{i;4}$ take one of 201 values corresponding to the most likely two-character prefixes and suffixes. The tag labels, $y_i \in \{1,\ldots 10\}$, correspond to the tags {verb, noun, adjective, adverb, preposition, pronoun, determiner, number, punctuation, other}.

To load a data point in Python, you can do e.g.,

```
datapath = 'data/';
files = os.listdir(datapath)

s = 50;   # to load file number 50:

fh = open(datapath+files[s],'r');
rawlines = fh.readlines();
lines = [line.strip('\n').split(',') for line in rawlines];
fh.close();
ys = [int(l[1])-1 for l in lines];
xs = [[int(l[2])-1,int(l[3]),int(l[4]),int(l[5])-1,int(l[6])-1] for l in lines];
```

We will build a structured support vector machine as described in class, using a regularized hinge loss. Specifically, we will use a structured linear predictor:

$$\hat{y}^{(s)} = \arg\max_y \theta \cdot u(y, x^{(s)}) = \arg\max_y \sum_{i=1}^{n_s} \sum_f \theta_f(y_i, x_{i;f}^{(s)}) + \sum_{i=1}^{n_s-1} \theta_{pair}(y_i, y_{i+1})$$

To be explicit, this model contains 10 + (10*2) + (10*2) + (10*201) + (10*201) + (10*10) parameters (for the five θ_f and θ_{pair} respectively).

We will train the model to optimize the hinge loss using stochastic gradient descent; the loss on one data point s is:

$$J(\theta) = \max_y \Delta(y, y^{(s)}) + \theta \cdot u(y, x^{(s)}) - \theta \cdot u(y^{(s)}, x^{(s)}) + \lambda \|\theta\|^2 \tag{1}$$

where $\Delta(y, y^{(s)}) = \sum_i \mathbb{1}[y_i \neq y_i^{(s)}]$ is the Hamming loss of the prediction.

Perform stochastic gradient descent on the hinge loss, e.g., modify & fill in the key parts of the code:

```
import pyGM as gm
feature_sizes = [1,2,2,201,201]
ThetaF = [.001*np.random.rand(10,feature_sizes[f]) for f in range(len(feature_sizes))];
ThetaP = .001*np.random.rand(10,10);
Loss = 1.0 - np.eye(10);  # hamming loss

# step size, etc.

for iter in range(num_iter):
  for s in np.random.permutation(len(files)):
    # Load data ys,xs
    ns = len(ys)

    # Define random variables for the inference process:
    Y = [gm.Var(i,10) for i in range(n)];

    # Build "prediction model" using your parameters
    factors = [ ...
    # don't forget pyGM expects models to be products of factors,
    #  so exponentiate the factors before making a model...
    model_pred = gm.GraphModel(factors);

    # Copy factors and add extra Hamming factors for loss-augmented model
    factors_aug = [ f for f in factors ]
    factors_aug.extend( [gm.Factor([Y[i]], Loss[:,ys[i]]).exp() for i in range(n)] );
    model_aug = gm.GraphModel(factors_aug);

    order = range(n);  # eliminate in sequence (Markov chain on y)
    wt = 1e-4;         # for max elimination in JTree implementation

    # Now, the most likely configuration of the prediction model (for prediction) is:
    yhat_pred = gm.wmb.JTree(model_pred,order,wt).argmax();
    # and the maximizing argument of the loss (for computing the gradient) is
    yhat_aug  = gm.wmb.JTree(model_aug,order,wt).argmax();

    # use yhat_pred & ys to keep a running estimate of your prediction accuracy & print it
    ...    # how often etc is up to you

    # use yhat_aug & ys to update your parameters theta in the negative gradient direction
    ...
```

Verify that your code is decreasing the loss (stochastically) and getting better at prediction. You may want to test using only a few data points to verify your gradient updates, and may need to experiment a bit with step sizes.

Try training your model with regularization parameters $\lambda = 0.01$, and give its performance in terms of the average per-symbol Hamming error and average per-symbol regularized hinge loss (e.g., divide (1) by the sequence length). Show both losses (hinge and Hamming) as a function of time or iteration.

Not required

Some additional ideas to explore if you like:

(1) Try initializing your model to the closed-form generative parameters corresponding to $\log p(x_{i;f}|y_i)$ and $\log p(y_{i+1}|y_i)$.

(2) Modify the code to do mini-batch updates.

(3) Optimize your regularization λ using a hold-out set.

(4) Try replacing your SSVM model with a CRF model, e.g., train on the conditional likelihood of the sequences. This mainly involves replacing the prediction steps (argmax) with marginalization inference, and updating the gradient accordingly.

(5) Modify your code to train a discriminative version of the Markov model in Homework 1, using the same parameters (but not constrained to be conditional distributions). Compare its performance to the results from Homework 1.