CS446: Machine Learning	Spring 2018
Machine Problem 7	
Handed Out: Jan. 16, 2018	*Due: Mar. 8, 2018 (11:59 AM Central Time)*

Note: The assignment will be autograded. It is important that you do not use additional libraries, or change the provided functions input and output.

Part 1: Setup

- Remove connect to a EWS machine.

```
ssh (netid)@remlnx.ews.illinois.edu
```

- Load python module, this will also load pip and virtualenv

```
module load python/3.4.3
```

- Reuse the virtual environment from mp1.

```
source ~/cs446sp_2018/bin/activate
```

- Copy mp7 into your svn directory, and change directory to mp7.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp7 .
cd mp7
```

- Install the requirements through pip.

```
pip install -r requirements.txt
```

- Prevent svn from checking in the data directory.

```
svn propset svn:ignore data .
```

Part 2: Exercise

In this exercise we will build a structured prediction model that, given several noisy samples $\hat{I}$ of some image I, learns a model that attempts to recover the image I. To simplify the problem, we will be representing each pixel of the images as binary values $\{0, 1\}$ (you can think of these as representing black/white).

The model for this function will be a linear markov random field represented by a grid graph whose nodes represent the pixels in an image and whose edges connect pairs of adjacent pixels. For a graph with n nodes, the specific scoring function used will be the following:

$$F(\mathbf{w}, x_1, \dots, x_n, y_1, \dots, y_n) = \sum_{i=1}^{n} \phi_u(\mathbf{w}, x_i, y_i) + \sum_{(i,j)\in\text{pairs}} \phi_{pw}(\mathbf{w}, y_i, y_j)$$

where $\mathbf{w} = [w_u, w_{pw}]$ are the linear weights for the model, x_i represents the observation at pixel i, y_i represents the hidden "true" value of pixel i, Φ_u is the unary potential function, and Φ_{pw} is the pairwise potential function. The potential functions are:

$$\begin{aligned}\phi_u(\mathbf{w}, x_i, y_i) &= w_u f_u(x_i, y_i)\\ \phi_{pw}(\mathbf{w}, y_i, y_j) &= w_{pw} f_{pw}(y_i, y_j)\end{aligned}$$

where f_u and f_{pw} are feature functions of the form

$$\begin{aligned}f_u(x_i, y_i) &= \mathbf{1}[x_i = y_i]\\ f_{pw}(y_i, y_j) &= \mathbf{1}[y_i = y_j]\end{aligned}$$

where $\mathbf{1}[\cdot]$ is the indicator function returning 1 if the argument is true and 0 if the argument is false. These feature functions attempt to enforce the predictions for the true values to be similar to the observations (in the case of f_u) and for neighboring nodes to have similar values (in the case of f_{pw}).

Inference for this model consists of finding the variable assignments $(y_1, \dots, y_n)$ that maximizes the scoring function. Since exact inference is slow, we will use an approximate greedy inference algorithm that computes a set of beliefs $b_r(y_r) \in \{0, 1\}$, where $b_r(y_r) = 1$ indicates an assignment of region to the value y_r. More specific details for this algorithm can be found in the code file `linear_mrf.py`.

We will use the following learning objective, which is derived from the general framework for learning presented in the lecture by taking the limit as ϵ approaches 0, replacing exact inference with our approximation, setting $C = 0$, and ignoring the task-loss term:

$$\begin{aligned}\min_{\mathbf{w}} \sum_{i\in\mathcal{D}} \Bigg(&\sum_{i=1}^{n} \sum_{y_i\in\{0,1\}} b_i(y_i)\phi_u(\mathbf{w}, x_i, y_i)\\ &+ \sum_{(i,j)\in\text{pairs}} \sum_{y_i\in\{0,1\}} \sum_{y_j\in\{0,1\}} b_{(i,j)}(y_i, y_j)\phi_{pw}(\mathbf{w}, y_i, y_j)\\ &- F(\mathbf{w}, x^{(i)}, y^{(i)})\Bigg)\end{aligned}$$

The model code template is found in `linear_mrf.py`. The overall structure is provided (including the implementation of the train/test loops) along with the function interfaces with detailed required functionality, which you will be required to implement. Specifically, you will be required to implement the following functions:

- `get_unary_features`
- `get_pairwise_features`
- `calculate_unary_potentials`
- `calculate_pairwise_potentials`
- `build_training_obj`
- `inference_itr`
- `calculate_local_score`
- `check_convergence`
- `get_pairwise_beliefs`

The file `main.py` has also been provided, which allows you to run training/testing on a few provided sample images.

Part 3: Writing Tests

In `test.py` we have provided basic test-cases. Feel free to write more. To test the code, run

```
nose2
```

Part 4: Submit

Submitting the code is equivalent to committing the code. This can be done with the follow command:

```
svn commit -m "Some meaningful comment here."
```

Lastly, double check on your browser that you can see your code at

```
https://subversion.ews.illinois.edu/svn/sp18-cs446/(netid)/mp7/
```