CS446: Machine Learning **Spring 2018**

Machine Problem 4

Handed Out: Jan. 16, 2018 *Due: Feb. 15, 2018 (11:59AM Central Time)*

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

Part 1: Setup

- Remote connect to an 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 mp4 into your svn directory, and change directory to mp4.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp4 .
cd mp4
```

- Install the requirements through pip.

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

- Create data directory and download the data into the data directory.

```
mkdir data
wget --user (netid) --ask-password \
https://courses.engr.illinois.edu/cs446/sp2018/\
secure/assignment4_data.zip  -O data/assignment4_data.zip
```

- Unzip assignment4_data.zip

```
unzip data/assignment4_data.zip -d data/
```

- Prevent svn from checking in the data directory.

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

Part 2: Exercise

When collecting datasets, it is very often that we will have to clean the dataset. For example, we may want to remove gray-scale images as its image statistics is very different than color-images. We can easily identify them, as can be seen in Fig. 1, however, we would like to automate process.

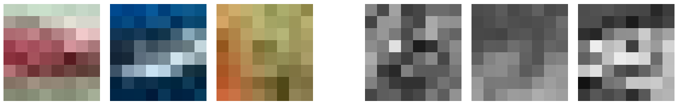

Figure 1: The left images are color images, the right images are gray-scale image.

One of the main difficulties is that gray-scale images may be saved in three channels and JPEG format. Due to the JPEG compression, we can no longer just check if the RGB channels are identical for all pixels. Therefore, in this exercise we will build a system to classify between gray-scale and color images using a SVM!

We illustrate the overall pipeline of the system in Fig. 2. We will implement each of the blocks.

In `main.py`, the overall program structure is provided for you.

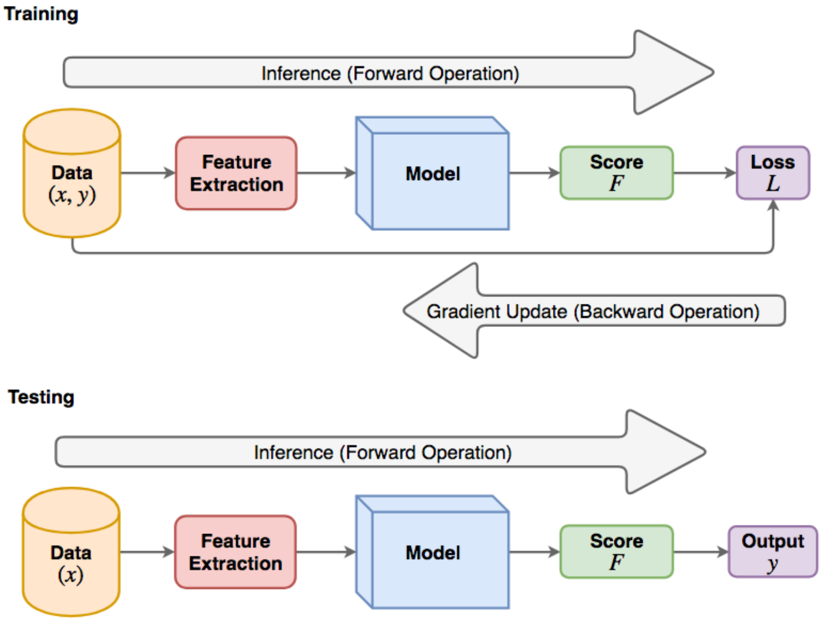

Figure 2: High-level pipeline

Part 2.1 Numpy Implementation

- **Reading in data.**

  In `utils/io_tools.py` we will fill in one function for reading in the dataset. The dataset consists of the directory of images stored in jpg format, and txt files indicating the filename of the image and label.

  There are three txt files, `train_lab.txt`, `val_lab.txt`, `test_lab.txt`, each indicate the images and labels for the corresponding dataset split.

  The txt format is in comma separated format. The columns are file_name, label.

  For example:

  ```
  0000,1
  ```

  The image_file "0000.jpg" has the label "1", meaning the image is a color image.

  Hint: skimage provides helpful functions to read in images.

- **Data processing.** In `utils/data_tools.py` we will fill in 3 functions for extracting feature. The feature extraction process involves simple image manipulation, such as mean removal, and element-wise operations. More details are provided in the function docstring.

  Hint: skimage provides helpful functions to extract image features.

- **Linear model implementation.** In `models/linear_model.py`, we will implement an abstract base class for linear models, then we will extend it to linear regression. The models will support the following operations:

  - **Forward operation.** Forward operation is the function which takes an input and outputs a score. In this case, for linear models, it is $F = \mathbf{w}^\intercal \mathbf{x} + b$. For simplicity, we will redefine $\mathbf{x} = [\mathbf{x}, 1]$ and $\mathbf{w} = [\mathbf{w}, b]$, then $F = \mathbf{w}^\intercal \mathbf{x}$.
  - **Loss function.** Loss function takes in a score, and ground-truth label and outputs a scalar. The loss function indicates how good the models predicted score fits to the ground-truth. We will use $\mathcal{L}$ to denote the loss. Note: Here we implement a modified SVM. (*i.e.* we include the bias-term in the margin; In practice, no observable difference in performance and easier to implement).
  - **Backward operation.** Backward operation is for computing the gradient of the loss function with respect to the model parameters. This is computed after the forward operation to update the model.
  - **Predict operation** The prediction operation is a function which takes a score as input and outputs a prediction in $\{-1, 1\}$.

- **Optimization**

  - Gradient descent: In `models/train_eval_model.py`, we will implement gradient descent. Gradient descent is an optimization algorithm, where the model adjusts the parameters in direction of the negative gradient of $\mathcal{L}$.

Repeat until convergence:

$$\mathbf{w}^{(t)} = \mathbf{w}^{t-1} - \eta \nabla \mathcal{L}^{(t-1)}$$

The above equation is referred as an update step, which consists of one pass of the forward and backward operation.

- Quadratic Programming: We can also optimize a SVM by formulating it as a quadratic program. We will use a generic QPsolver, `cvxopt`, for solving this quadratic program. In the written assignment 4, we have formulated a hard-svm as a quadratic program. In this assignment, we will reformulate a soft-svm. Recall that a soft-svm

$$\min_{w} \frac{C}{2} \|\mathbf{w}\|^2 + \cdot \sum_{i=1}^{|D|} \xi^{(i)} \quad \text{s.t. } y^{(i)}(\mathbf{w}^\intercal \mathbf{x}^{(i)} \geq 1 - \xi^{(i)}), \xi^{(i)} \geq 0 \ \ , \forall (x^{(i)}, y^{(i)}) \in \mathcal{D} \tag{1}$$

Again, we will use the same quadratic programming form.

$$\begin{aligned} \underset{\mathbf{z}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{z}^\intercal P \mathbf{z} + \mathbf{q}^\intercal \mathbf{z} \\ \text{subject to} \quad & G\mathbf{z} \leq \mathbf{h} \end{aligned}$$

In `train_eval_mode.py`, we will code up the matrix for $P, \mathbf{q}, G, \mathbf{h}$.
Hint: Write it out on paper!

- **Running Experiments.** In `main.py`, experiment with different features, weights initialization and learning rate. We will not grade the `main.py` file, feel free to modify it.

  To run `main.py`

```
python main.py
```

- **Things to think about.** Here is list of things to think about, you do not have to hand in anything here.
  - Which optimization method is better, gradient descent or QP solver?
  - What kind of features will be helpful?
  - What happens if we set $C = 0$? What happens to the train, validation and testing accuracy?

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)/mp4/
```