CS446: Machine Learning **Spring 2018**

Machine Problem 3

Handed Out: Jan. 16, 2018 *Due: Feb. 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. Also, you should include your best-trained 'trained_weights.np' under `codefromscratch` when you make submission to SVN.

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 mp0.

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

- Copy mp3 into your svn directory, and change directory to mp3.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp3 .
cd mp3
```

- 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 \
http://courses.engr.illinois.edu/cs446/secure/assignment3_data.zip \
-O data/assignment3_data.zip
```

- Unzip assignment3_data.zip

```
unzip data/assignment3_data.zip -d data/
```

- Prevent svn from checking in the data directory.

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

Part 2: Exercise

In this exercise, we will build a logistic model based binary classifier that can separate two-class audio datasets. To be specific, the datasets consist of features extracted from audio waveforms of letters of the alphabet. The two classes represent the ee-set (b,c,d,e,g,p,t,v,z) and the eh-set (f,l,m,n,s,x), respectively. The binary classifier will be based on the logistic regression model from our lecture as well as the sigmoid function from the written homework.

The overall flow of the binary classifier is illustrated in Fig. 1 and we will implement each of the blocks in both "code-from-scratch" and "TensorFlow" parts. In `main.py`, the overall program structure is provided for you.

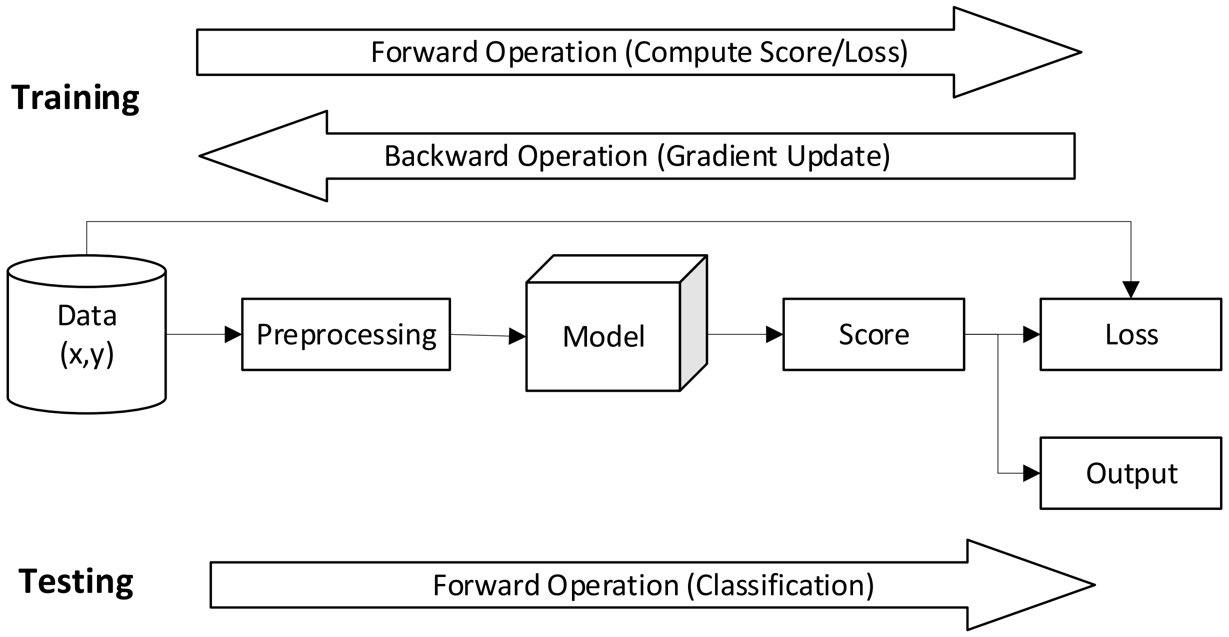
Training
Forward Operation (Compute Score/Loss)
Backward Operation (Gradient Update)
Data (x,y)
Preprocessing
Model
Score
Loss
Output
Testing
Forward Operation (Classification)

Figure 1: High-level Flow

Part 2.1 Code-From-Scratch Implementation

- **Reading and Preprocessing Data.**

  In `io_tools.py`, we will fill in one function for reading in the dataset as well as preprocessing it. The downloaded dataset has only "trainset" because the real "testset" belongs to autograder, which is used to evaluate the actual accuracy of your classifier. No matter "trainset" or "testset", they share the same format. Specifically, the "trainset" has an "indexing.txt" and a folder "samples", where the former file lists all samples line-by-line in the form of "class_label sample_file.txt" and the latter folder contains all sample files in ASCII text format. For more details, check the "README.txt". After the dataset is read, it should be preprocessed to match the specified form in return arguments of the function. More details are provided in the function docstring.

- **Logistic Model**

  In `logistic_model.py`, we will implement a class for logistic models and use it for binary classification. The class will support the following operations:

  - **Forward operation.** Forward operation is the function which takes input features and generate a probability score. In this case, for logistic models, it is $score = \mathrm{g}(\mathbf{w}^\intercal \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\intercal \mathbf{x}}}$, where the sigmoid function is used from the written homework. Note that, we redefine $\mathbf{x} = [1, \mathbf{x}]$ and $\mathbf{w} = [b, \mathbf{w}]$.
  - **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. **Note that, in this part, loss function should use the $\sum_i \log\left(1 + \exp(-y^{(i)}\mathbf{w}^T\phi(x^{(i)}))\right)$ as in the lecture slides**.
  - **Classify operation.** Classify operation performs the binary classification on input dataset. Each of input samples is computed based on current update-to-date model and then is predicted as one of the two classes $\{+1/-1\}$. Since all samples are taken in as input, the final output will be a vector of predicted class labels.
  - **Fit operation.** It is the training function which utilizes the above methods as necessary to fit the logistic model to the input dataset through the gradient descent approach. As in the lecture, gradient descent is an optimization algorithm, where the model adjusts its parameters in direction of the negative gradient of the loss value. The gradient descent should be repeated iteratively until convergence.
  - **Save/Load operation.** After training with hundreds or thousands of iterations, the final model should be saved into a binary file using the methods – `np.tofile('trained_weights.np')` and `np.fromfile('trained_weights.np')`, so that, in future, the well-trained model can be restored immediately and perform classification tasks. **Note that, you should also include your best-trained 'trained_weights.np' under `codefromscratch` when submitting your code to SVN.**

- **Running Experiments.** The `main.py` shows the overall flow containing all previous methods. We will not grade the `main.py` file, feel free to modify it.

Relevant Files: everything in the folder `codefromscratch`

Part 2.2 TensorFlow Implementation

In this part, we will implement the model in the previous section using TensorFlow (TF). The TF part focuses more on training method using the quadratic loss function $\sum_i \left(y_i - \mathrm{g}(\mathbf{w}^\top \mathbf{x}_i)\right)^2$

where $y \in \{0, 1\}$, instead of the previous loss function $\sum_i \log\left(1 + \exp(-y^{(i)}\mathbf{w}^T\phi(x^{(i)}))\right)$ where $y \in \{-1, 1\}$. Also, for simplicity, less operations are required to implement in the TF part.

- **Reading and Preprocessing Data.** In general, everything is the same except the class labels.
- **Logistic Model** The class in TF part only needs to support these operations:
  - **Build Graph operation.** This operation builds the entire training and testing TF graph for our logistic classifier, which includes model creation, forward score/loss computation, optimizer definition, as well as training accuracy computation. It is better to put the entire TF graph in a single method, according to our experience.
  - **Fit operation.** The training operation in TF utilizes the quadratic loss as mentioned earlier, which also includes iteratively gradient descent update on weights as well as computation of forward score and training accuracy during each iteration. After the model is trained, this operation also returns the score vector of last iteration.
- **Running Experiments.** The `main.py` shows the overall flow containing all previous methods. We will not grade the `main.py` file, feel free to modify it.

Relevant Files: everything in the folder `codefromtf`

Part 3: Writing Tests

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

```
python test_*.py -v
or
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)/mp3/
```