CS446: Machine Learning **Spring 2018**

Machine Problem 5

Handed Out: Jan. 16, 2018 *Due: Feb. 22, 2018 (11:59 AM Central Time)*

Part 1: Setup

- Remote 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 previous MPs.

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

- Copy mp5 into your svn directory, and change directory to mp5.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp5 .
cd mp5
```

- Install the requirements through pip.

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

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

```
mkdir data
wget --user (netid) --ask-password \
https://courses.engr.illinois.edu/cs446/sp2018/\
secure/assignment5_data.zip -O data/assignment5_data.zip
unzip data/assignment5_data.zip -d data/
```

- Prevent svn from checking in the data directory.

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

Part 2: Exercises

In this exercise we will use SVM to recognize handwritten digits. The dataset we use is MNIST handwritten digit database, where every handwritten digit image is represented as 28×28 pixels, each with value 0 to 255. We want to classify each image as one of 0 to 9.

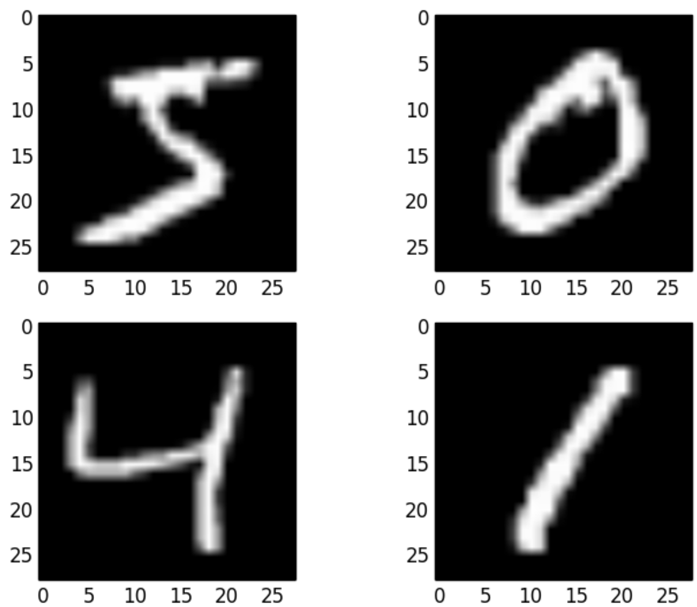

Figure 1: MNIST examples.

In the provided version of MNIST dataset, features are flattened into $784 = 28 \times 28$ dimensional vectors. To save running time, we only use the first 50%(5000) of the original MNIST test data as our training dataset, and the next 50%(5000) of the original test data as our test dataset.

In this exercise, we will first use Scikit Learn's built-in multiclass classification functions to train one-vs-rest(one-vs-all) and one-vs-one multiclass linear SVM models. Then we will implement one-vs-rest and one-vs-one multiclass classifiers ourselves, using binary SVM models.

Throughout the exercise, we will use `sklearn.svm.LinearSVC` as our binary SVM classifier, with default parameters unless specifically mentioned, and `random_state=12345` for reproductivity. If you are not familiar with Scikit Learn, please make sure you understand how to use `fit` and `predict` methods in the above link.

Part 2.1 Using Built-in Multiclass Functions

In this exercise, we will use Scikit Learn's `sklearn.multiclass.OneVsRestClassifier` and `sklearn.multiclass.OneVsOneClassifier` to perform multiclass classification. We will also use Crammer-Singer multiclass SVM (the multiclass SVM formulation in the lecture), which is supported in `sklearn.svm.LinearSVC`.

Task 1:
Implement `sklearn_multiclass_prediction` in `model/sklearn_multiclass.py`, which takes training and test features and labels, as well as a string `mode` being one of "ovr", "ovo", or "crammer". The function should pick the correct classifier to train and predict labels for both training and test data.

Thinking Questions: How would you compare OVR, OVO and Crammer-Singer, in terms of classification accuracy and time efficiency?

Part 2.2 Implementing One-vs-Rest and One-vs-One Classification

In this exercise, we will use `sklearn.svm.LinearSVC` only with binary labels, 0 and 1. (You can use any other two labels, but 0 and 1 are recommended.) We will implement class `MulticlassSVM` in `model/self_multiclass.py`, which is constructed given `mode` being one of "ovr" and "ovo". When `fit` and `predict` methods are called, it will call the correct version of multiclass classification given `mode`.

Task 2:
Implement `bsvm_ovr_student` `bsvm_ovo_student` in `model/self_multiclass.py`, which takes training data `X` and `y`, and returns a python `dict` with keys being labels for OVR, and pairs of labels for OVO, and with values being trained OVR or OVO binary `sklearn.svm.LinearSVC` classifiers.

Task 3:
Implement `scores_ovr_student` `scores_ovo_student`
in `model/self_multiclass.py`, which takes features `X`, and returns a numpy ndarray $Score$ with shape $(\#Samples, \#Labels)$, where $Score(i, j)$ is the number of votes of the j-th label received for the i-th sample in OVO, and is the confidence score of the j-th label for the i-th sample in OVR (use `decision_function` as confidence score).

Thinking Questions: Why do we need use confidence scores for OVR? Why do we not use confidence scores for OVO?

After the scoring functions are implemented, `predict_ovr` and `predict_ovo` will return the labels with maximum votes.

Part 2.3 Implementing Multiclass SVM

In this part, we will implement our own loss function of (Crammer-Singer) multiclass linear

SVM as:

$$\min_{w_1,\dots,w_K} \frac{1}{2}\sum_{j=1}^{K}\|w_j\|_2^2 + C\sum_{i=1}^{N}\max_{j=1\dots K}\left(1-\delta_{j,y_i}+w_j^\top x_i\right) - w_{y_i}^\top x_i$$

where $\delta_{j,y_i} = 1$ if $j = y_i$ and 0 if $j \neq y_i$, and optimize it via gradient descent. Your task is to compute the loss function and the gradients of the loss function w.r.t. $W = [w_1, ..., w_K]^\top \in \mathbb{R}^{K\times d}$, given W, $X = [x_1, ..., x_N]^\top \in \mathbb{R}^{N\times d}$, $Y = [y_1, ..., y_N] \in \{0, ..., 9\}^N$ and $C = 1$.

Task 4:
Implement `loss_student` `grad_student` in `model/self_multiclass.py`, which takes `W`, `X` and `y`, and returns the loss function and its gradient w.r.t. `W` respectively.

Hint:

1. Think some concrete cases, for example the one in the written assignment.
2. Though not required, try using matrix operations of Numpy when possible for faster performance. Some useful functions: `np.sum`, `np.max`, `np.argmax`.

Part 2.4 Comparing Built-in and Self-Implemented Functions

Compare the classification accuracy of self-implemented and built-in OVR and OVO. They should be almost the same. It is normal to be a little bit off for OVO, due to tie-breaking; but you should not get more than 0.5% difference. For OVR, the accuracy should be exactly the same. For Crammer-Singer multiclass SVM, it is normal that your accuracy is different from that of the built-in function, due to implementation details.

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)/mp5/
```

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