CS446: Machine Learning	**Spring 2018**

Machine Problem 2

Handed Out: Jan. 16, 2018 — *Due: Feb. 1, 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 mp2 into your svn directory, and change directory to mp2.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp2 .
cd mp2
```

- 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/assignment2_data.zip -O data/assignment2_data.zip
```

- Unzip assignment2_data.zip

```
unzip data/assignment2_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 system to predict housing prices. We illustrate the overall pipeline of the system in Fig. 1. We will implement each of the blocks.

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

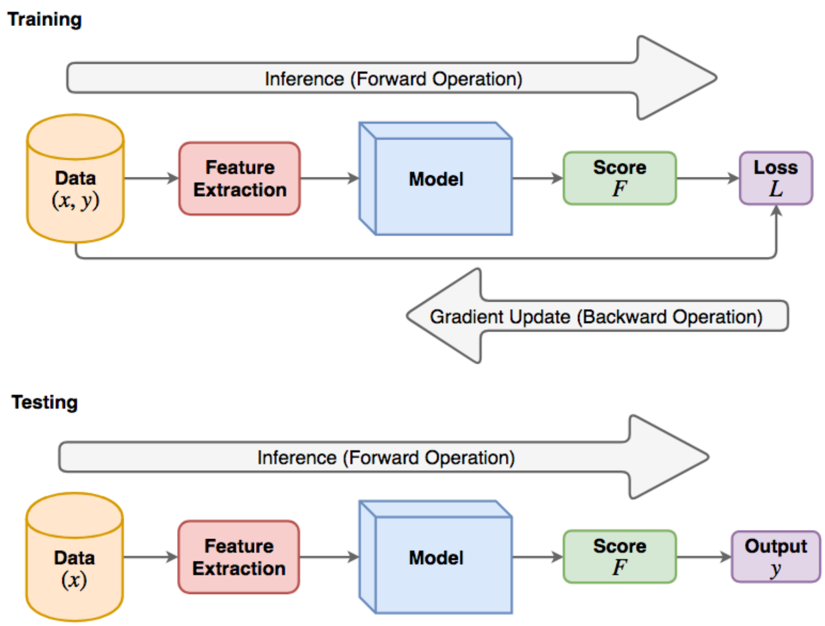

Figure 1: High-level pipeline

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 housing features (*e.g.* the size of the house, location, ..., etc.) and the price of the house.

  There are three csv files, `train.csv`, `val.csv`, and `test.csv`, each contains examples in each of the dataset splits.

  The format is comma separated, and the first line containing the header of each column.

  ```
  Id,BldgType,OverallQual,GrLivArea,GarageArea,SalePrice
  1,1Fam,7,1710,548,208500
  ```

  Everything before the SalePrice may be the input to our system, and SalePrice is the quantity we hope to predict.

- **Data processing.** In `utils/data_tools.py`, we will implement functions to transform the data into vector forms. For example, converting the location column into one-hot encoding. There is a total of five types of buildings, 1Fam, 2FmCon, Duplx, TwnhsE, TwnhsI. In order to represent this, we construct a vector of length five, one for each type, where each element is a Boolean variable indicating the existence of the building type. For example,

```
1Fam   = [1, 0, 0, 0, 0]
2FmCon = [0, 1, 0, 0, 0]
...etc.
```

  More details are provided in the function docstring.

- **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.
  - **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.

- **Optimization**
  - **Gradient descent.** In `models/train_eval_model.py`, we will implement gradient descent. Gradient descent is a 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.
  - Linear regression also has an analytic solution, which we will also implement.

- **Model selection.** For the optimization above, it is about learning the model parameters $\mathbf{w}$. In this case, we use the training split of the dataset to "train" these parameters. Additionally, there are several hyper-parameters in this model, (*e.g.* learning rate, weight decay factor, the column features). These hyper-parameters should be chosen based on the validation split (*i.e.* for each hyper-parameter setting, we find the optimal $\mathbf{w}$ using the training set then compute the loss on the validation set; We will choose the hyper-parameters with the lowest validation error as the final model.

- **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 a list of things to think about, you do not have to hand in anything here.

  - How does learning effect convergence?
  - Which optimization is better, analytic solution or gradient descent?
  - Are squared features better? Why?
  - Which of the column features are important?

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 following 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)/mp2/
```