CS 334: Homework #3

Submission Instructions: Your submission should consist of two steps (detailed in Homeworks #1, #2). If *either of the two steps are late, then your assignment is late.*

Dataset Description: For this homework, you will be predicting the energy usage of appliances in a low energy house. The dataset (`energydata.zip`) contains measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station, and the recorded energy use of lighting fixtures to predict the energy consumption of appliances in a low energy house. There are 27 attributes for each 10-minute interval[1]. Details of the column attributes are described in detail on the University of California, Irvine Machine Learning repository (`https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction`).

The dataset has been split into training data from measurements up to 5/7/16 4:30, and test data from measurements after 5/7/16 4:30.

1. **Feature Extraction + Feature Selection** (2+6+3+2+4+3+2+3=25 points): For this problem, you will consider performing some feature selection and extraction of the data. The template code for this problem is found in `selFeat.py`.

   (a) (Written) The first column of the energy data has a date and timestamp (e.g., 3/20/16 5:30). What are features that you can extract from this data that might be useful for predicting energy usage? Choosing to only drop this column is not an acceptable answer.

   (b) (Code) Implement your feature extraction in `extract_features` to add the new features to the pandas dataframe and to remove the date column. Do not name any of the new features as 'date'.

   (c) (Code) Calculate the Pearson correlation in `cal_corr` between the features themselves and the features with the target as a correlation matrix ($(d+1) \times (d+1)$ matrix), where each value at (i, j) and (j, i) represents the correlation between the ith feature and the jth feature. Note that a value close to 0 implies almost no correlation while a value close to 1 and -1 implies a strong positive and negative correlation, respectively.

   (d) (Written) Plot the correlation matrix using a heatmap from the previous part, 1c (Hint: use `seaborn.heatmap`).

   (e) (Written) Using the correlation plot, identify the features that you think should be used to train a linear regression model and your reasoning for using them. You must justify your feature selection to get full points.

   (f) (Code) Implement your feature selection in `select_features`.

   (g) (Written) What other preprocessing steps do you need to do to the data?

   (h) (Code) Implement your preprocessing steps in the `preprocess_data` function in the `selFeat.py`. Now use `selFeat.py` to preprocess both the training and test data and save it to new files. The main program will (1) extract the features from the date and timestamp (a); (2) select the features (c); and (3) preprocess the remaining columns (d). You can do this from the command line by doing the following:

   ```
   $python selFeat.py new_xTrain.csv new_xTest.csv
   ```

   You'll be using these two new files (new_xTrain.csv and new_xTest.csv) to train the linear regression models in the subsequent questions.

[1] We have removed the last two random variables and log transformed the energy usage.

2. **(3+7=10 pts) Linear Regression: Single Unique Solution** For this problem, you will implement the closed form solution for linear regression without using `scikit-learn`. Since you will be implementing two algorithms for Linear Regression, we've created an Abstract class, `LinearRegression`, to standardize the functions and to share code[2]. The two classes that will implement the `LinearRegression` class are `StandardLR` and `SgdLR`. The template code for this problem is found in `lr.py,standardLR.py`.

   (a) (Code) Implement the `predict` function in the `LinearRegression` class object (file `lr.py`). Given the coefficients (stored in the class variable `beta`) and a feature matrix `xFeat`, predict the response for each sample.

   (b) (Code) Implement the closed form solution of linear regression in the `train_predict` function of the `StandardLR` object. The idea is to train the model on the training data and also simultaneously evaluate it on the test data. The parameter specifications are:

   - `xTrain` is a numpy $n \times d$ array where each row represents a sample and each column a feature,
   - `yTrain` is a numpy 1-D array of size $n \times 1$ that contains which class (0 or 1)
   - `xTest` is a numpy $m \times d$ array where each row represents a sample and each column a feature,
   - `yTest` is a numpy 1-D array of size $m \times 1$ that contains which class (0 or 1)

   The output of the function is a single dictionary with the statistics on the data. The key is the iteration number and the value is a dictionary with the time elapsed, the training mean squared error, and the testing mean squared error. For example in the closed-form solution, we might get something like:

```
{0: {'time': 1,
     'train-mse': 0.85,
     'test-mse': 0.95 }}
```

   Run the closed form regression on the energy dataset. You can run it from the command line using the new files from 1(d):

```
$python standardLR.py new_xTrain.csv eng_yTrain.csv new_xTest.csv eng_yTest.csv
```

3. **(10+15+10+5=40 pts) Linear Regression using SGD**

   For this problem, you will implement the solution to linear regression using permutation-based stochastic gradient descent (see Algorithm 1), a variant of stochastic gradient descent. Instead of randomly selecting the samples, you will randomly shuffle the dataset to create a random permutation. The template code is found in `sgdLR.py`. Similar to the previous problem, you *ARE NOT* allowed to use any existing toolbox / implementation (i.e., you can use numpy and scipy but not others).

   (a) (Code) Implement the `grad_pt` helper function to calculate the gradient for a mini-batch sample.

[2] The abstract class in Python is very similar to Java except that Python objects can implement multiple abstract classes if needed. The annotation `@abstractmethod` and the inheritance of the abstract class from `ABC` lets the interpreter known that this is the intention of the class.

Algorithm 1 Stochastic Gradient Descent

for epoch = 1 : MAX EPOCH **do**
  Randomly shuffle training data and break into $B = N/\text{BatchSize}$ batches
  for $b = 1 : B$ **do**
    Compute the gradients associated with samples in batch b, ∇f
    Take the average of the gradients in the batch
    Update the parameters based on the gradient
  end for
end for

(b) (Code) Implement the `train_predict` function in the linear regression using stochastic gradient descent. Your implementation should use a user-specified fixed learning rate (η), the mini-batch size ($bs \in [1, n]$), and the maximum number of epochs. For the output dictionary, the key is the iteration number or batch number that also reflects the epoch. For example, if the total number of batches is 40, and the algorithm just completed the 2nd epoch, then it would start at iteration number 80:

```
{0:{'time': 0.05,
     'train-mse': 3.58,
     'test-mse': 5.88 },
...
80: {'time': 5.9,
     'train-mse': 0.95,
     'test-mse': 1.25 }}
```

(c) (Written) For a batch size of 1 (i.e., $bs = 1$) and a random subset of 40% of the training data, try various learning rates (such as 1, 0.1, 0.01, 0.001, 0.0001, 0.00001) and plot the mean squared error on the training data as a function of the epoch (i.e., one epoch = one pass through the training data). What seems to be an optimal learning rate? Justify your answer based on the plot.

(d) (Written) Using the optimal learning rate from above, train the model on the entire dataset and plot the mean squared error on the training data and the test data as a function of the epoch.

4. **(20+5=25 pts) Comparison of Linear Regression Algorithms using SGD and Closed Form solutions**

For this problem, you will compare the results of the linear regression algorithm using SGD (problem 3) and the closed form solution (problem 2).

(a) (Written) Explore the impact of batch sizes on the computation time and convergence of the SGD algorithm. Choose a variety of batch sizes (it should include 1, and n, and a few more, where the training size is divisible by the number). For each batch size, find a reasonable learning rate and then run SGD using that batch size. Plot the mean squared error of the training data and the test data as a function of the total time for different batch sizes. To make it easier, you may want to split training and test into two different plots. The plots should also include the point reflecting the closed form solution.

(b) (Written) What are your observations based on the plot in (a)? What are the trade-offs between different batch sizes as well as the closed form solution?