EE 559 | **Homework 5 (Week 7)** | Posted: Sat, 2/26/2022
Jenkins | | Due: Sat., 3/5/2022, 11:59 PM

Please note:

1. *Reminder:* python only for coding.
2. *Coding quality.* On Homeworks 4 and 5, you will be given feedback (comments) on your coding quality, but you coding quality will not affect your homework score. You are encouraged on Homework 5 to try to apply the guidelines for coding quality as you code; it might help you write code that functions correctly and is easier to debug. After Homework 5, your coding quality will be graded and will count towards your homework score. Information on coding quality can be found in Discussion 6 (Week 7) and related documents posted on D2L in Week 7.
3. *Files to upload.* This homework has 2 problems that require coding. **Please submit 3 files:** 1 pdf file of your solutions, 1 pdf file of all your code for Problem 1, and 1 pdf file of all your code for Problem 3.

1. In this problem, you will code up a linear regressor based on the MSE criterion function. You will also investigate different learning rate parameter schedules and their effect on convergence in iterative gradient descent (GD) optimization.

   Coding yourself. As in previous homework assignments, in this problem you are required to write the code yourself; you may use only Python built-in functions, NumPy, and matplotlib; you may use the "random" module because it is part of the Python standard library; and you may use pandas only for reading and/or writing csv files.

   Dataset. Throughout this problem, you will use a real dataset based on the "Combined Cycle Power Plant Data Set"
   [https://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant]
   from the UCI Machine Learning Repository .

   We have subsampled the data (to save on computation time) and preprocessed it, so be sure to **use the dataset files *h5w7_pr1_power_train.csv* and *h5w7_pr1_power_test.csv* provided with this homework assignment**. There are a total of 500 data points, with a split of 75% of points in the training set, and 25% of points in the test set.

   The dataset has 4 real-valued features: temperature (T), ambient pressure (AP), relative humidity (RH), and exhaust vacuum (V). The output value y_i is the net hourly electrical energy output (EP). Note that the 5th column in the dataset files is the output value. We have normalized the input features already, so that each input feature value is in the range $0 \leq x_{ij} \leq 1$.

   (a) This part does not require any coding. Do the convergence criteria given in Lecture 10, page 12 for MSE classification, also apply to MSE regression? Justify your answer.

   Hint: compare the weight update formulas.

Answer the parts below as instructed, regardless of your answer to (a).

Please code the regressor as follows:

(i) Code a linear MSE regressor that uses iterative GD for optimization (LMS algorithm); use basic sequential GD. The schedule to use for $\eta(i)$ will be given below. Hints on how to perform the random shuffle are given in Homework 4.

(ii) For the initial weight vector, let each w_j = a random number drawn independent and identically distributed (i.i.d.) according to the uniform density function $U[-0.1, +0.1]$. **Hint:** use numpy.random.uniform().

(iii) Before the first epoch (at $i = 0$, call it epoch $m = 0$), and at the end of each epoch, compute and store the RMS error for epoch m as:

$$E_{RMS}^{(m)} = \left(J(\underline{w})\right)^{\frac{1}{2}}$$

(iv) For halting condition, halt when either one of these 2 conditions are met, and have the code output which halting condition was reached:

iv.1 $E_{RMS}^{(m)} < 0.001 E_{RMS}^{(0)}$

iv.2 100 epochs have been completed.

(v) When the iterations halt, store the final value of $\underline{w}$ as $\hat{\underline{w}}$ for each value of (A, B) (as described below), in a table.

(vi) You will also need a function that gives the output prediction $\hat{y}(\underline{x})$ for any given input $\underline{x}$, given the optimal $\hat{\underline{w}}$ from the learning algorithm.

(b) In this part you try a learning rate parameter of the form:

$$\eta(i) = \frac{A}{B + i}$$

in which i is the iteration index (increases by 1 with each successive data point). Try a grid search over the following values: $A = 0.01, 0.1, 1, 10, 100$, and $B = 1, 10, 100, 1000$ (i.e., use two nested loops to try all combinations of A and B). For each pair (A, B), plot a learning curve as $E_{RMS}^{(m)}$ *vs.* m.

Tip: do 5 plots total, one plot for each value of A. Have each plot show 4 curves (one curve for each value of B).

(c) Comment on the dependence of the learning curves on A and B.

(d) Pick the resulting best pair (A, B) from (b) above (based on final $E_{RMS}^{(m)}$ for each pair), and use its value of $\hat{\underline{w}}$ to calculate the E_{RMS} error on the test set.

(e) As a comparison, consider a trivial regressor that always outputs $\hat{y}(\underline{x}) = \bar{y}$ = {mean value of the outputs over the training data}. Calculate the E_{RMS} of this trivial regressor on the test set. Is your regressor's error from (d) substantially lower than the error of this trivial regressor?

2. In lecture we derived the solution weight vector $\underline{\hat{w}}^{(+)}$ for ridge regression, in which the entire augmented weight vector is regularized. Often the non-augmented weight vector is regularized, and the bias term w_0 is used to help minimize the MSE but is not regularized. For this case, the criterion function becomes:

$$J(\underline{w}) = \left\| \underline{\underline{X}}^{(+)} \underline{w}^{(+)} - \underline{y} \right\|_2^2 + \lambda \left\| \underline{w}^{(0)} \right\|_2^2$$

Derive the optimal weight vector solution $\underline{\hat{w}}^{(+)}$ for this case.

Hint: use augmented notation throughout, and make use of the diagonal matrix $\underline{\underline{I}}' = diag\{0,1,1,\cdots,1\}$.

3. In this problem, you will implement a nonlinear mapping from a 2D feature space to an expanded feature space, perform learning in the expanded space, then map the resulting decision regions into a more optimal 2D space, and separately into the original 2D feature space. Most of this you will do on computer; you will have help on the coding in the form of given functions and tips. You may use non-built-in functions from libraries like sklearn only where specifically stated as allowed or where you are given sklearn classes or functions to use. If you are not yet familiar with OOP in Python, this problem gives you a good opportunity to try it.

The file *h5w7_pr3_helper_functions.py* contains functions that will help you complete the problem and the file *h5w7_pr3_data.csv* contains the synthetic dataset. Each data point is represented by two features, x_1 and x_2, and it may belong to class 1 or class 2. You will use all the data points in every one of the following parts.

(a) Plot the data points in (non-augmented) feature space. Use different colors or different markers to indicate the class to which each point belongs. Without running any learning algorithm, state whether you think this data set is linearly separable in this feature space.

(b) Use sklearn's implementation of the perceptron to train a perceptron on this dataset. Obtain and report the classification accuracy.

Hints:

(i) Sklearn's perceptron documentation at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html contains an example of how to use this class.

(ii) You can either use the `score` method to directly obtain the mean accuracy or the `predict` method, which will return the predicted labels for each sample.

(iii) This dataset is already centered, that is, the mean of each feature is zero. Therefore, you should set the parameter for finding $\hat{w}_0$, `fit_intercept`, to `False`. Use the default value for all other parameters.

(c) Plot the learned decision boundaries from the perceptron in item (b).

We provide two functions to help with this task, but you can also code your own.

Hints:

(i) The function linear_decision_function implements the perceptron linear decision function. You do not need to call it in your code. It is called automatically by the plotting function.

(ii) The function plot_perceptron_boundaries takes as parameters, in addition to the training data, the learned weights and a decision function. Here's an example of how to call it with the linear_decision_function:

```
weights = classifier.coef_[0]  # get the perceptron weights
plot_perceptron_boundaries(X_train, y_train, weights,
                                 linear_decision_function)
```

in which classifier is a trained perceptron object.

Also note that the provided function returns a figure object, which can be used to programmatically save the plot.

(d) Let's try to use a quadratic feature space expansion to improve the result above, i.e.,

$$(x_1, x_2) \rightarrow (x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

Code a function to apply the feature expansion procedure to the entire dataset. Then train a perceptron on the expanded dataset and report its classification accuracy. Is the dataset linearly separable in the expanded feature space?

Hint: remember that you do not need to add the bias term.

(e) **[Extra credit]** We want to plot the decision boundary and regions as a function of the 2 most relevant features of the expanded feature space. For this, follow these steps:

(i) Provide the learned weight vector from item (d)

(ii) Which two features received the highest weights in absolute value? Note: your answer must be two elements from $(x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Write the decision function only in terms of these two features and their weights.

(iii) Create a new feature matrix that contains only the two most relevant features found above. Create a new weight vector that contains only the two highest weights in absolute value.

(iv) Call the plotting function with the new feature matrix and new weight vector. Example:

```
plot_perceptron_boundaries(phi_X_best_2,y_train,
weights_best_2, linear_decision_function)
```

Is the dataset linearly separable in this feature space?

(f) **[Extra credit]** Next, we want to plot the decision boundary and regions in the original feature space. To do this, you need to code the decision function in the original feature space. The skeleton of the nonlinear_decision_function is already provided. You can call the plotting function as:

```
plot_perceptron_boundaries(X_train, y_train, relevant_weights
                           nonlinear_decision_function)
```