EE 559 **Homework 4 (Week 6)** Posted: Fri., 2/26/2021
Jenkins Due: Fri., 3/5/2021, 11:59 PM PST

Notes:

1. This is our first python-only homework: you are required to use only python for the coding problem (Problem 3).
2. The quality of your python code will be checked and you will be given comments on it where there is room for improvement; hopefully you will find this helpful. For this homework assignment only, there will be no points awarded or deducted based on the quality of your code; it's a trial run.

1. In Lecture 7, we defined the criterion function for a 2-class Perceptron Learning problem (in augmented space) as:

$$J(\underline{w}) = -\sum_{n=1}^{N} [\![\underline{w}^T z_n\, \underline{x}_n \leq 0]\!]\, \underline{w}^T z_n\, \underline{x}_n$$

   (a) Rewrite the criterion function using $max\{\cdot\}$ function.

   Hint: the second plot on Page 8 of Discussion 4.

   (b) Based on your solution in (a), show that the Perceptron criterion function is convex using the definition of convex functions.

2. For 2-class perceptron *with margin* algorithm, using basic sequential GD, fixed increment, prove convergence for linearly separable training data by modifying the perceptron convergence proof covered in class. You may write out the proof, or you may take the 3-page proof from the attached handout (or from lecture), and mark up the proof to show all changes as needed. If you mark up the existing proof, be sure to mark everything that needs changing (*e.g.*, if a change propagates through the proof, be sure to make all changes for a complete answer), and please make your mark-ups stand out for readability.

3. In this problem, you will code and run a multiclass perceptron learning algorithm.

   Coding. You are required to code the algorithm yourself, using python. You can use built-in python functions, numpy, and matplotlib. Use of other libraries or packages (e.g., pandas or scikit-learn) are not in the spirit of this homework problem and will result in points deducted (one exception: you can use pandas to read and write csv data files).

   Follow the multiclass perceptron algorithm given in Lecture 9. You will use this on a 3-class problem, but it is recommended that you code it for a C-class problem, with $C \geq 3$. Similarly, it is recommended to code it for D features, although the data will have $D = 13$ features.

You will use the **wine data from Homework 1 (Week 2)** for this problem. Use all 13 features, and use the (raw) unnormalized data.

You will use stochastic gradient descent variant 1 (SGD 1), and stochastic gradient descent variant 2 (SGD 2), as instructed below.

Use two halting conditions as follows (the algorithm should halt when at least one halting condition is satisfied). Note that an "iteration" refers to using one data point.

(h1) For SGD 1: The usual perceptron halting condition (halts when there are no weight updates in 1 full epoch)

For SGD 2: Halts when there have been no weight updates in the previous 100 iterations.

(h2) For either SGD: Halts after $i_{\max}$ iterations. It then considers the error rate from the triplet of weight vectors $\underline{w}_1(i), \underline{w}_2(i), \underline{w}_3(i)$ at each of the last 100 iterations; the final weight vectors are those from the iteration that gave the lowest error rate on the training data. Thus $\underline{\hat{w}}_k = \underline{w}_k(i_0),\ k = 1,2,3$, in which i_0 is the iteration out of the last 100 that gave the best error rate on the training data.

The error rate is defined by:

$$\text{error rate} = \frac{\text{number of data points that are not classified correctly}}{\text{total number of data points}}$$

Tips: You may find the following functions useful for the randomization steps. Please read the document pages for more examples. Just keep in mind that the ground truth labels should be aligned with your data points all the time no matter how you shuffle them.

1) For SGD 1: *np.random.permutation* can be used to generate shuffled data point indices, which can be further used to shuffle your data by indexing.
2) For SGD 2: *np.random.choice* with *size* = 1 can be used to generate one index with replacement, and then draw your data point by indexing.

(a) Run your perceptron learning algorithm code to learn from the training data, using SGD 1. Use the following parameters:

$$i_{\max} = 10{,}000, \quad \underline{w}(0) = \underline{1} .$$

(i) Report on the classification error rate on the training set and on the test set.

(ii) Which halting condition was met?

(iii) From your results of (i) and (ii), what (if anything) can you say about the linear separability of the training dataset?

(b) Using the same system as in the previous part, compute the error rate on the training set at every 10th iteration from $i = 0$ until it halts. Run this experiment 10 times.

Tip: you may find it easier to generate the plots below by storing your error rates in a 10x1000 matrix.

Show 3 plots:

(i) Error rate vs. iteration for your first run

(ii) Mean and standard deviation of the error rate over the 10 runs, vs. iteration

(iii) Maximum and minimum error rate, over the 10 runs, vs. iteration

If these plots show the error approaching an asymptote long before the algorithm halts, then include another plot that "magnifies" the iteration axis (and doesn't go all the way until it halts) so that it is easier to see the dependence on *i*.

(c) From your plots in (ii)-(iii) of the previous part, approximately how many iterations does it take to get to a good halting point? What is the error rate at this halting point? (You can give the mean result, and the worst-case (maximum plot) result, for both questions.)

(d)-(f) Repeat (a)-(c) using SGD 2.

(g) Answer the following:

(i) Do you see any significant difference in the number of iterations to get to a good halting point, between SGD 1 and SGD 2? Please describe and try to explain why.

(ii) Do you see any significant difference in the error rate at your choice of good halting point, between SGD 1 and SGD 2? Please describe and try to explain why.

(iii) Do you see any other significant difference in the plots of (b), as compared with the plots of (e)? Please describe and try to explain why.

4. During the Lectures, we introduced the mean-squared criterion function for a linear regression model $\hat{y}_n = \underline{w}^T \underline{x}_n$ (in augmented space) to be:

$$J(\underline{w}) = \frac{1}{N}\sum_{n=1}^{N}[\hat{y}_n - y_n]^2$$

(a) Based on the second-order derivative, prove that the mean-squared criterion function is a convex function.

(b) Based on the *Linear Regression* models, if we add a penalty term which is equal to the square of the coefficients in the weight vector (L-2 regularization), it becomes *Ridge Regression*. Similarly, if we use L-1 regularization where the penalty is the absolute sum of the coefficients, we have *Lasso Regression* with the following criterion function:

$$J(\underline{w}) = \frac{1}{N}\sum_{n=1}^{N}[\hat{y}_n - y_n]^2 + \lambda\sum_{j=1}^{D+1}|w_j|, \qquad \lambda \geq 0$$

(i) Express Lasso Regression criterion function in matrix/vector form. (**Hint:** your final solution should have no summation operation.)

(ii) Is the regularization term of Lasso Regression convex? Is it strictly convex? Prove based on the definition of convex functions.

(iii) Is the Lasso Regression criterion function convex? Justify your answer.