EECS 553 HW 4

Due Thursday, September 26, by 11:59 PM via Gradescope and Canvas

When you upload your solutions to Gradescope, please indicate which pages of your solution are associated with each problem. Also please note that your code should be uploaded to Gradescope (as a .pdf) and Canvas (as .py) as discussed below.

1. **Subgradient methods for the optimal soft margin hyperplane** (3 points each)

   In this problem you will implement the subgradient and stochastic subgradient methods for minimizing the convex but nondifferentiable function

$$J(\boldsymbol{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \left(L(y_i, \boldsymbol{w}^T \boldsymbol{x}_i + b) + \frac{\lambda}{2} \|\boldsymbol{w}\|^2 \right)$$

   where $L(y, t) = \max\{0, 1 - yt\}$ is the hinge loss. As we saw in class, this corresponds to the optimal soft margin hyperplane classifier.

   (a) Determine $J_i(\boldsymbol{w}, b)$ such that

$$J(\boldsymbol{w}, b) = \sum_{i=1}^{n} J_i(\boldsymbol{w}, b).$$

   Determine a subgradient $\boldsymbol{u}_i$ of each J_i with respect to the variable $\boldsymbol{\theta} = [b\ \boldsymbol{w}^T]^T$. A subgradient of J is then $\sum_i \boldsymbol{u}_i$.

   Note: If $f(\boldsymbol{z}) = g(h(\boldsymbol{z}))$ where $g : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}$, and both g and h are differentiable, then

$$\nabla f(\boldsymbol{z}) = \nabla h(\boldsymbol{z}) \cdot g'(h(\boldsymbol{z})).$$

   If g is convex and h is differentiable, the same formula gives a subgradient of f at $\boldsymbol{z}$, where $g'(h(\boldsymbol{z}))$ is replaced by a subgradient of g at $h(\boldsymbol{z})$.

   Download the file `hw4_pulsars.zip` from canvas. The data for this binary classification problem consists of features which are statistics of radio signals from stars, and the binary classes denote whether the signal came from a pulsar or not. For more details on the dataset, see: `archive.ics.uci.edu/ml/datasets/HTRU2`.

   `pulsar_features.npy` contains a $d \times n$ matrix of features, where $d = 2$ features and $n = 3278$ samples (note we are only using the 1st and 7th features from the original HTRU2 dataset). `pulsar_labels.npy` contains an n vector of labels where -1 denotes not a pulsar and 1 denotes a pulsar.

   (b) Implement the subgradient method for minimizing J and apply it to the pulsar data. Submit two figures: One showing the data and the learned line, the other showing J as a function of iteration number. Also report the estimated hyperplane parameters, the margin, and the minimum achieved value of the objective function.

   Use the starter code in `hw4_p1.py` to visualize the data and seed the random number generator.

   Comments:

- Use $\lambda = 0.001$.
- Use a step-size of $\eta_j = 100/j$, where j is the iteration number.
- To compute the subgradient of J, write a subroutine to find the subgradient of J_i, and then sum those results.
- Perform 10 iterations of gradient descent.
- Initialize the hyperplane parameters to zeros before training.
- Debugging goes faster if you just look at a subsample of the data. You can also use the Python debugger pdb: `https://realpython.com/python-debugging-pdb/`.

(c) Now implement the *stochastic subgradient* (SGD) method, which is like the subgradient method, except that your step direction is a subgradient of a randomly selected J_i, not J. Be sure to cycle through all data points before starting a new loop through the data. Report/hand in the same items as for part **(b)**. In addition, comment on the (empirical) rate of convergence of the stochastic subgradient method relative to the subgradient method.

More comments:

- Use the same λ, η_j, and initialization as in part **(b)**. Here j indexes the number of times you have cycled (randomly) through the data.
- Cycle through the data 10 times (i.e. perform $10n$ steps of stochastic subgradient descent).
- To save time, you do not need to compute J after every update, as that would result in too many computations. You should compute J after every iteration through the data points.
- To generate a random permutation use

  `np.random.permutation`
- Submit all code to Canvas (.py) and Gradescope (.pdf).

2. Soft Thresholding Derivation (3 points each)

Consider the lasso as discussed in class, and in particular, the update for w_j at iteration t of a coordinate descent algorithm. Introduce the following notation:

$$\boldsymbol{w}_{-j}^{(t)} = \begin{bmatrix} w_1^{(t)} \\ \vdots \\ w_{j-1}^{(t)} \\ w_{j+1}^{(t-1)} \\ \vdots \\ w_d^{(t-1)} \end{bmatrix} \qquad \boldsymbol{x}_{i,-j} = \begin{bmatrix} x_{i,1} \\ \vdots \\ x_{i,j-1} \\ x_{i,j+1} \\ \vdots \\ x_{i,d} \end{bmatrix}.$$

(a) Show that the subdifferential of $g(w_j)$ is

$$\partial g(w_j) = \begin{cases} a_j^{(t)} w_j - c_j^{(t)} - \lambda, & w_j < 0 \\ [a_j^{(t)} w_j - c_j^{(t)} - \lambda, a_j^{(t)} w_j - c_j^{(t)} + \lambda], & w_j = 0 \\ a_j^{(t)} w_j - c_j^{(t)} + \lambda, & w_j > 0 \end{cases}.$$

where

$$a_j^{(t)} = \frac{2}{n} \sum_i x_{ij}^2, \qquad c_j^{(t)} = \frac{2}{n} \sum_i x_{ij}(y_i - (\boldsymbol{w}_{-j}^{(t)})^T \boldsymbol{x}_{i,-j} - b^{(t)}).$$

(b) The value of w_j should be selected such that $0 \in \partial g(w_j)$. Determine the value of w_j that makes this true by considering the following three cases separately: $c_j^{(t)} < -\lambda, c_j^{(t)} \in [-\lambda, \lambda], c_j^{(t)} > \lambda$. Deduce that the optimal value of w_j is given by the soft-thresholding formula as stated in the lecture slides.

3. Sparse Linear Regression (3 points each)

In this problem, you will implement the lasso via coordinate descent on the Ames, Iowa Housing dataset (located on Canvas `hw4_iowa.zip`. The data is composed of 2000 train instances and 925 test instances. There are 58 features, and the target/response is home price in \$1000. More details about the dataset can be found at `jse.amstat.org/v19n3/decock/DataDocumentation.txt`. We are using 55 of these features, as well as 3 additional features (`Total.SF, Total.Bath, Total.Porch.SF`). The file `housing_feature_names.npy` gives the feature names for corresponding indices in the data.

We will estimate a linear function to predict house prices given the other features. Our objective function is

$$J(\boldsymbol{w}, b) := \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{w}^T\boldsymbol{x}_i + b - y_j)^2 + \lambda\|\boldsymbol{w}\|_1$$

where $\lambda > 0$, $\boldsymbol{w} = [w_1, \ldots, w_d]^T$ is the vector of weights, not including the offset term, and $\boldsymbol{x}_i$ is the i-th feature vector of dimension d. The data is split into training and test samples.

(a) Sphere the feature variables in the training data by transforming the $d \times n$ feature matrix $\mathbb{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ so that each of its rows have zero sample mean and have sample variance equal to one. Specifically, for each feature j, compute the sample mean $\hat{\mu}_j = n^{-1}\sum_{i=1}^{n} x_{ij}$ and the sample variance $\hat{\sigma}_j^2 = n^{-1}\sum_{i=1}^{n}(x_{ij} - \hat{\mu}_j)^2$, and then replace x_{ij} by $(x_{ij} - \hat{\mu}_j)/\hat{\sigma}_j$, $j = 1, \ldots, d$, $i = 1, \ldots, n$. Verify that the rows of the thus sphered $\mathbb{X}$ training data matrix have zero sample mean and unit variance.

Submit plots of the vector $\hat{\boldsymbol{\mu}}$ of sample means and the vector $\hat{\boldsymbol{\sigma}}$ of sample standard deviations.

Note: Above, x_{ij} denotes the j-th entry of $\boldsymbol{x}_i$. An equivalent notation would be $\boldsymbol{X}_{ji}$, denoting the (j, i) entry of the data matrix $\boldsymbol{X}$.

(b) Implement CD Lasso regression in python. Run your CD Lasso regression code on the **sphered** data you created in part (a) for 2900 updates with $\lambda = 100/n$ and with $\boldsymbol{w}$ initialized to be a vector of all 1's. Report the test MSE. Also comment on how many, if any, w_i are zero.

4. Kernels (3 points each)

(a) Verify that

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v})^3$$

is an inner product kernel by determining an associated feature map.

(b) Prove one direction of an equivalence stated in the notes, namely, that if k is an inner product kernel, then k is a symmetric, positive definite kernel.

(c) In kernel ridge regression with offset (see lecture notes), give a formula for the offset b that can be evaluated using the kernel. In your formula, the training instances $\boldsymbol{x}_i$ should only occur as an argument of the kernel. In other words, the formula for the offset should be kernelized.