(c). Next, we will see what happens when we vary λ. For 200 evenly spaced samples of λ ranging between 0 and 10^5, compute the d ridge regression coefficients and offset as we did in part (b). Then plot the trajectories of the d ridge coefficients $\mathbf{w}$ on a single plot as a function of λ (do not plot the offset w_0).

On a separate figure, use the training data to plot $MSE = \frac{1}{n}\sum_{j=1}^{n}(\mathbf{w}^T\mathbf{x}_j + w_0 - y_j)^2$ as λ varies over this range. Submit both the weight trajectories and training MSE plots.

(d). Finally, we will compare error on the training and test sets. First, obtain the ridge regression coefficients and offsets for 200 evenly spaced values of λ between 1 and 100. For each λ, evaluate MSE on the training and test data.

Submit a plot which shows both the train and test MSE as a function of λ. Compare the trends for train and test, and briefly comment on similarities/differences. Report the λ which minimizes the train MSE, and the λ which minimizes test MSE.

Note that, prior to applying the ridge predictor, for each test sample feature i you will need to subtract the value $\hat{\mu}_i$ and divide by the value $\hat{\sigma}_i$, determined in part (a).

2. Optimal soft-margin hyperplane

Let $(\boldsymbol{w}^*, b^*, \boldsymbol{\xi}^*)$ denote the solution to the soft-margin hyperplane quadratic program. (Note that the parameter b is denoted w_0, and that the vectors $\boldsymbol{w}$ and $\boldsymbol{\xi}$ are a different font in Sec. 001 lectures.)

a. Show that if $\boldsymbol{x}_i$ is misclassified by the optimal soft-margin hyperplane classifier, then $\xi_i^* \geq 1$. Conclude that $\frac{1}{n}\sum_i \xi_i^*$ upper bounds the training error. Hence, the OSM objective is balancing margin maximization with minimizing a bound on the training error.

b. Show that if $\xi_i^* > 0$, then ξ_i^* is proportional to the distance from $\boldsymbol{x}_i$ to the margin hyperplane associated with class y_i (that is, the set $\{\boldsymbol{x} : (\boldsymbol{w}^*)^T\boldsymbol{x} + b^* = y_i\}$), and give the constant of proportionality.

3. Subgradient methods for the optimal soft margin hyperplane

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

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

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 $J_i(\mathbf{w}, b)$ such that

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

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

Note: Recall the chain rule for subdifferentials discussed in class: If $f(\mathbf{z}) = g(h(\mathbf{z}))$ where $g : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}$, and both g and h are differentiable, then

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

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

Download the file `hw3_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 nuclear 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 `hw3_p3.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. Explain your findings.

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).