Homework 2

Deadline: Thursday, Oct. 10, at 11:59pm.

Submission: You need to submit three files through MarkUs[1]:

- Your answers to Questions 1, 2, and 3, as a PDF file titled `hw2_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.
- Your code for Question 1, as the Python file `q1.py`.
- Your completed code for Question 2, as the Python file `q2.py`.

If you wish to write the code in a Jupyter notebook instead, then please submit a PDF printout of the notebook, rather than the notebook itself.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Computing: To install Python and required libraries, see the instructions on the course web page.

Collaboration: Homeworks are individual work. See the course web page for detailed policies.

1. **[3pts] Robust Regression.** One problem with linear regression using squared error loss is that it can be sensitive to outliers. Another loss function we could use is the *Huber loss*, parameterized by a hyperparameter δ:

$$L_\delta(y,t) = H_\delta(y-t)$$
$$H_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{if } |a| > \delta \end{cases}$$

   (a) **[1pt]** Sketch the Huber loss $L_\delta(y,t)$ and squared error loss $L_{\mathrm{SE}}(y,t) = \frac{1}{2}(y-t)^2$ for $t = 0$, either by hand or using a plotting library. Based on your sketch, why would you expect the Huber loss to be more robust to outliers?

   (b) **[1pt]** Just as with linear regression, assume a linear model:

$$y = \mathbf{w}^\top \mathbf{x} + b.$$

   Give formulas for the partial derivatives $\partial L_\delta / \partial \mathbf{w}$ and $\partial L_\delta / \partial b$. (We recommend you find a formula for the derivative $H'_\delta(a)$, and then give your answers in terms of $H'_\delta(y-t)$.)

   (c) **[1pt]** Write Python code to perform (full batch mode) gradient descent on this model. Assume the training dataset is given as a design matrix `X` and target vector `y`. Initialize $\mathbf{w}$ and b to all zeros. Your code should be vectorized, i.e. you should not have a `for` loop over training examples or input dimensions. You may find the function `np.where` helpful.

   Submit your code as `q1.py`.

[1] `https://markus.teach.cs.toronto.edu/csc2515-2019-09`

2. **[5pts] Locally Weighted Regression.**

   (a) **[2pts]** Given $\{(\mathbf{x}^{(1)}, y^{(1)}), .., (\mathbf{x}^{(N)}, y^{(N)})\}$ and positive weights $a^{(1)}, ..., a^{(N)}$ show that the solution to the *weighted* least squares problem

   $$\mathbf{w}^* = \arg\min \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2 + \frac{\lambda}{2} ||\mathbf{w}||^2 \tag{1}$$

   is given by the formula

   $$\mathbf{w}^* = \left(\mathbf{X}^T \mathbf{A} \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^T \mathbf{A} \mathbf{y} \tag{2}$$

   where $\mathbf{X}$ is the design matrix (defined in class) and $\mathbf{A}$ is a diagonal matrix where $\mathbf{A}_{ii} = a^{(i)}$

   It may help you to review Section 3.1 of the csc321 notes[2].

   (b) **[2pts]** Locally reweighted least squares combines ideas from k-NN and linear regression. For each new test example $\mathbf{x}$ we compute distance-based weights for each training example $a^{(i)} = \frac{\exp(-||\mathbf{x}-\mathbf{x}^{(i)}||^2/2\tau^2)}{\sum_j \exp(-||\mathbf{x}-\mathbf{x}^{(j)}||^2/2\tau^2)}$, computes $\mathbf{w}^* = \arg\min \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2 + \frac{\lambda}{2} ||\mathbf{w}||^2$ and predicts $\hat{y} = \mathbf{x}^T \mathbf{w}^*$. Complete the implementation of locally reweighted least squares by providing the missing parts for `q2.py`.

   Important things to notice while implementing: First, do not invert any matrix, use a linear solver (numpy.linalg.solve is one example). Second, notice that $\frac{\exp(A_i)}{\sum_j \exp(A_j)} = \frac{\exp(A_i - B)}{\sum_j \exp(A_j - B)}$ but if we use $B = \max_j A_j$ it is much more numerically stable as $\frac{\exp(A_i)}{\sum_j \exp(A_j)}$ overflows/underflows easily. *This is handled automatically in the scipy package with the* `scipy.misc.logsumexp` *function*[3].

   (c) **[1pt]** Based on our understanding of overfitting and underfitting, how would you expect the training error and the validation error to vary as a function of τ? (I.e., what do you expect the curves to look like?)

   Now run the experiment. Randomly hold out 30% of the dataset as a validation set. Compute the average loss for different values of τ in the range [10,1000] on both the training set and the validation set. Plot the training and validation losses as a function of τ (using a log scale for τ). Was your guess correct?

3. **[2pts] AdaBoost.** The goal of this question is to show that the AdaBoost algorithm changes the weights in order to force the weak learner to focus on difficult data points. Here we consider the case that the target labels are from the set $\{-1, +1\}$ and the weak learner also returns a classifier whose outputs belongs to $\{-1, +1\}$ (instead of $\{0, 1\}$). Consider the t-th iteration of AdaBoost, where the weak learner is

   $$h_t \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \sum_{i=1}^{N} w_i \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\},$$

   the w-weighted classification error is

   $$\text{err}_t = \frac{\sum_{i=1}^{N} w_i \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i},$$

[2] `http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L02%20Linear%20Regression.pdf`

[3] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.misc.logsumexp.html`

and the classifier coefficient is $\alpha_t = \frac{1}{2} \log \frac{1-\mathrm{err}_t}{\mathrm{err}_t}$. (Here, log denotes the natural logarithm.) AdaBoost changes the weights of each sample depending on whether the weak learner h_t classifies it correctly or incorrectly. The updated weights for sample i is denoted by w_i' and is

$$w_i' \leftarrow w_i \exp\left(-\alpha_t t^{(i)} h_t(\mathbf{x}^{(i)})\right).$$

Show that the error w.r.t. $(w_1', \ldots, w_N')$ is exactly $\frac{1}{2}$. That is, show that

$$\mathrm{err}_t' = \frac{\sum_{i=1}^N w_i' \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i'} = \frac{1}{2}.$$

Note that here we use the weak learner of iteration t and evaluate it according to the new weights, which will be used to learn the $t+1$-st weak learner. What is the interpretation of this result?

Tips:

- Start from err_t' and divide the summation to two sets of $E = \{i : h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}$ and its complement $E^c = \{i : h_t(\mathbf{x}^{(i)}) = t^{(i)}\}$.
- Note that
$$\frac{\sum_{i \in E} w_i}{\sum_{i=1}^N w_i} = \mathrm{err}_t.$$