Homework 3

Deadline: Friday, Feb. 8, at 11:59pm.

Submission: You need to submit three files through MarkUs `https://markus.teach.cs.toronto.edu/csc411-2019-01`:

- Your answers to Questions 1 and 2 as a PDF file titled `hw3_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.
- Your completed code files `q1.py` and `q2.py`

Neatness Point: One of the 10 points will be given for neatness. You will receive this point as long as we don't have a hard time reading your solutions or understanding the structure of your code.

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.

Collaboration. Weekly homeworks are individual work. See the Course Information handout `http://www.cs.toronto.edu/~mren/teach/csc411_19s/syllabus.pdf` for detailed policies.

Data. In this assignment we will be working with the Boston Housing dataset[1]. This dataset contains 506 entries. Each entry consists of a house price and 13 features for houses within the Boston area. We suggest working in python and using the `scikit-learn` package[2] to load the data.

Starter Code. Starter code written in Python is provided for Question 2.

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 a Python function `gradient_descent(X, y, lr, num_iter, delta)` which takes as input:

[1] `http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`
[2] `http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html`

- a training set `X`: given as a design matrix, an ndarray of shape (N, D)
- a target vector `y`: an ndarray of shape $(N,)$
- scalar `lr`: learning rate value
- `num_iter`: integer value specifying number of iterations to run gradient descent
- `delta`: hyperparameter for the Huber loss

and returns the value of the parameters $\mathbf{w}$ and b after performing (full batch mode) gradient descent to minimize this model's cost function for `num_iter` iterations. Initialize $\mathbf{w}$ and b to all zeros inside the function. 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`.

2. **[6pts] 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 $\mathbf{A}_{ii} = a^{(i)}$.

(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]** 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).

(d) **[1pt]** How would you expect this algorithm to behave as $\tau \to \infty$? When $\tau \to 0$? Is this what actually happened?

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