Assignment 1

SENG 474 / CSC 578D
Data Mining - Spring 2018
Total: SEng 474: 65 points; CSc 578D: 75 points
Worth 5% of your final grade

Due: February 14th (Happy Valentine's day! ♡)

Submit a pdf through ConneX before 11:55pm. I will accept submissions up to 24 hours late for a 10% reduction in your mark. After that point, no submissions will be accepted.

Show your work for all questions.

Different marking schemes will be used for undergrad (SEng 474) and grad (CSc 578D) students. Undergrad students do not have to answer the grad questions.

All code questions use the python/numpy/pytorch/matplotlib. You may install these libraries on your own computer, or you can work in the lab.

For each question you should **hand the attached code skeleton, including your implementations in it. Include graphs and written answers in your pdf.**

1: Neural Networks

(SEng 474; CSc 578D: 20 points)

Use the attached code skeleton `circle.py` to complete this question.

(a) Using PyTorch, build a network that can properly classify a point in 2D space as being inside or outside of the unit circle. See skeleton code provided. Plot the percent correct on training data against iteration number. [**10 pts**]

(b) Generate a new test batch and run your final model on it. Plot the points in 2D and color them based on their predicted label. We suggest using the provided `plot_decision_boundary` function. [**5 pts**]

(c) The simple neural network architecture provided in the skeleton code cannot properly predict for this task. Why? Plotting the predicted labels (as in b) for the simple architecture may help you to answer this question. [**5 pts**]

2. Gradient Descent: Linear Regression

(SENG 474: 30 points, CSC578D: 40 points. SEng 474; CSc 587D: 5 Bonus points)

In this part of the assignment we will implement a linear regression model, using the Boston houses dataset. We will implement two techniques for minimizing the cost function: batch gradient descent, and stochastic gradient descent.

You are provided with a skeleton python program `linreg.py` and you have to implement the missing parts.

First let's recall some theory and mathematical notation[1]. We express our dataset as a matrix $X \in \mathbb{R}^{n \times m}$ where each row is a training sample and each column represents a feature, and a vector $\boldsymbol{y} \in \mathbb{R}^n$ of target values. In order to make the following notation easier, we defined a training sample as vector as $\boldsymbol{x} = [1, x_1, x_2, \ldots, x_m]$. Basically we add a column of ones to X. The purpose of linear regression is to estimate the vector of parameters $\boldsymbol{w} = [w_0, w_1, \ldots, w_m]$ such that the hyper-plane $\boldsymbol{x} \cdot \boldsymbol{w}^T$ fits, in an optimal way, our training samples. Once we have obtained $\boldsymbol{w}$ we can predict the value of testing sample, $\boldsymbol{x}_{test}$, simply by plugging it into the equation $\hat{y} = \boldsymbol{x}_{test} \cdot \boldsymbol{w}^T$. We estimate $\boldsymbol{w}$ by minimizing a cost function over the entire dataset, which is defined as follows

$$E(\boldsymbol{w}) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \boldsymbol{x}_i \cdot \boldsymbol{w}^T \right)^2$$

we apply gradient descent techniques to find the parameters vector $\boldsymbol{w}$ that minimizes the cost function $E(\boldsymbol{w})$. This is achieved by an iterative algorithm that starts with an initial guess for $\boldsymbol{w}$ and repeatedly performs the update

$$w_j := w_j - \kappa \frac{\partial}{\partial w_j} E(\boldsymbol{w}) \quad j = 0, 1, \ldots, m$$

where κ is the learning rate parameter.

In this question you are asked to implement some gradient descent techniques. We are going to set a maximum number of iteration, `max_iter` in the code, and analyze what happens to the cost function at each iteration by plotting the error function at each iteration.

SEng 474; CSc 587D: 30 pts

(a) Implement the *batch gradient descent* algorithm. Plot the cost function for each iteration. Test it with a few different learning rates. E.g. $\kappa = 0.001$, 0.01, 0.1. Explain what is happening. [**15 pts**]

(b) Implement the *stochastic gradient descent* algorithm. Plot the cost function with different learning rate. Compare it with the batch gradient descent. [**15 pts**]

[1] Matrices are represented by a capital letter, vectors with a lower case bold letter.

CSc 587D only, 10 pts

(c) For the code above, we used a fixed learning rate. For this question, you should research alternative methods for setting the learning rate. Describe in detail one method that uses a learning rate schedule, and one method that is adaptive. Explain not only the update rule, but the intuition behind why the approach works. [**5 pts**]

SEng 474; CSc 587D: Bonus Question, 5 pts

(d) Now we introduce L_2 *regularization*. Explain why it is used. Implement it and repeat (b) with different values of λ i.e 10, 100, 1000. Remember that with regularization the cost function becomes

$$E(\boldsymbol{w}) = \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \boldsymbol{x}^{(i)} \cdot \boldsymbol{w}^T\right)^2 + \lambda \sum_{j=1}^{m} w_j^2\right)$$

Note that by convention we don't regularize w_0. [**5 pts**]

3. Gradient Descent: Logistic Regression

(SENG 474; CSC578D: 15 points)

For this question, we will implement logistic regression, and test it using the breast cancer dataset and the code skeleton in `logreg.py`. For this question, you only need to implement batch gradient descent. This time, plot the *log likelihood* function for each iteration:

$$\mathcal{L}(\mathbf{w}) = \sum_i y_i(\mathbf{w}'\mathbf{x_i}) - \sum_i \ln(1 + e^{\mathbf{w}'\mathbf{x_i}}) \tag{1}$$

Test it with a few different learning rates. E.g. $\kappa = 0.001, 0.01, 0.1$. Explain what is happening. [**15 pts**]