Machine Learning
Homework 3 : Regression - I
(due Midnight Feb 7)

Instructions

1. In case you are unfamiliar with the Python data ecosystem (NumPy, Pandas), you are recommended to study the first four chapters of the Python data science handbook. A doubt clearing session would be organised in case you have any difficulties in the data science ecosystem.
2. The deadline for full score is Midnight Feb 7. You can get 50% credit for late submission (Midnight Feb 9).
3. Total marks = 22
4. You have to type the assignment using a word processing engine, create a pdf and upload on the form. Please note that only pdf files will be accepted.
5. All code/Jupyter notebooks must be put up as **secret gists** and linked in the created pdf submission. Again, only secret gists. Not public ones.
6. Any instances of cheating/plagiarism will not be tolerated at all.
7. Cite all the pertinent references in IEEE format.
8. The least count of grading would be 0.5 marks.

1. Learn $y = \theta_0 + \theta_1 \times x$ on the following small dataset on pen and paper. You may scan or click picture of your answers and attach to the pdf.

$$X = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix}$$

and

$$Y = \begin{bmatrix} 6 \\ 10 \\ 16 \end{bmatrix}$$

using:

   (a) Normal equations or matrix method **[1 mark]**

   (b) Gradient desent where initial values of $\theta_0, \theta_1)$ is (0, 0) and step size (or learning rate) is $\alpha = 0.1$. Show the calculations for initial 5 iterations. **[1 mark]**
   EDIT: Does the cost/objective diverge? Are you using residual sum of squares (sum of squared errors)? Maybe you want to try using mean of squared errors as the objective? Or, use a small learning rate α.

   (c) Using the formula in terms of covariance and variance **[1 mark]**

2. (a) For the following X and y, use scikit-learn to learn a linear model. **[1 mark]**

   (b) Solve the problem using normal equations. You may find that one of the matrix in the normal equation is non-invertible. Why does the matrix turn out to be non-invertible? Why can scikit-learn implementation still correctly solve this regression problem? **[1 mark]**

$$X = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \\ 4 & 8 \end{bmatrix}$$

and

$$Y = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

3. (a) Show the usage of scikit learn's linear regression module for the real estate price prediction regression problem. What is the RMS error on the test set? **[1 mark]**

   (b) Based on the regression co-efficients what can you comment about the importance of different features? Is it correct to assume that larger co-efficients means more important feature? **[1 mark]**

   (c) Now, standardize the dataset to have all features on a scale of 0 to 1. Re-learn the regression co-efficients and now comment on the importance of different features. **[1 mark]**

   (d) What is the distribution of the residuals? **[1 mark]**
   EDIT: Hint - is it a normal distribution? or, lognormal, or ...?

   (e) Use cross-validation to find the optimal set of features to use for regression.

      i. Using all possible feature sets of length 1, 2, 3, or 4, what is the optimal feature set as per the validation set and how does this set of features perform on the test set wrt the model learnt on the entire feature set? **[1 mark]**

      ii. Use Sequential Forward Selection (or Stepwise Forward Selection) which is a greedy procedure to find the optimal set of features. How does this how does it perform on the test set wrt the model learnt on the entire feature set? **[1 mark]**

4. In this question, you will be writing your custom linear regression implementation.

   (a) Write a function `normalEquationRegression(X, y)` where X is our feature matrix containing N samples (rows) and d features (columns) and y is our output vector containing N samples. This function returns a vector θ containing $d + 1$ rows. You are free to use numpy's matrix inverse, determinant and multiplication routines. **[1 mark]**

   (b) Write a function `gradientDescentRegression(X, y, alpha = 0.1)` to learn the regression coefficients using gradient descent. You have to write the formulae for gradient wrt the different $\theta_j \forall j \in (1, ..d)$ **[1 mark]**

   (c) Write a function `gradientDescentAutogradRegression(X, y, alpha = 0.1)` to learn the regression coefficients using gradient descent. Instead of writing the formulae for computing gradients by yourself, you will use Autograd to automatically do that for you. **[1 mark]**
   EDIT: In the above function you solved earlier: `gradientDescentRegression(X, y, alpha = 0.1)`, you manually computed the formulae for gradients and plugged that into gradient descent. In this version, you will use autograd to compute the gradient of the cost or objective function (which is sum of squared errors or mean of squared errors) wrt $\theta's$.

   (d) Write a function `gradientDescentPyTorchRegression(X, y, alpha = 0.1)` to learn the regression coefficients using gradient descent. Instead of numpy, you would now be using PyTorch. This question will set the ground for your project and future assignment. Similar to Autograd linked in the previous question, PyTorch also has an automatic gradient computation routine that you should make use of. Please note that do not use `nn.Linear()` for this question. **[1 mark]**

   (e) Illustrate the usage of all of your above four versions on the real estate price dataset in Q3. Report the time taken and accuracy for the four implementations compared with scikit-learn inbuilt function. Use all features from the dataset for all the methods. **[2 marks]**

5. In this question, we will be implementing polynomial regression as a special case of linear regression. First, we will be generating some data.

```
import numpy as np
x = np.arange(0, 20.1, 0.1)
np.random.seed(0)
y = 1*x**5 + 3*x**4 - 100*x**3 + 8*x**2 -300*x - 1e5 + np.random.randn(len(x))*1e5
```

Now, we want to learn a polynomial function of degree p on this dataset, i.e. $y = \theta_0 + \theta_1 \times x^1 + \theta_2 \times x^2 + ...\theta_p \times x^p$. We can use our developed linear regression implementations for doing so, by transforming our dataset and creating the matrix X containing columns corresponding to x^0, x^1, x^2, ..., x^p. Using any of your implementations learn the regression coefficients for $p = 5$ and $p = 4$. How close are your coefficients for $p = 5$ to the ones used to generate the data? **[2 marks]**

6. The following question is to aid our understanding of gradient descent. We will be reusing the data from Q1.

   (a) Create a contour plot in the θ_0 and θ_1 space of the residual sum of squares **[1 mark]**

   (b) Create a Matplotlib animation where the plot contains two columns: the first one being the contour plot and the second one being the linear regression fit on the data. The different frames in the animation correspond to different iterations of gradient descent applied on the dataset to learn θ_0 and θ_1. For each iteration, draw the current value of θ_0 and θ_1 on the contour plot and also an arrow to the next θ_0 and θ_1 as learnt by gradient update rule. Correspondingly draw the $y = \theta_0 + \theta_1 \times x$ line on the other subplot showing the scatter plot. The overall title of the plot show be the iteration number and the residual sum of squares. You are free to use any gradient desent implementation. **[2 marks]**