Assignment-3
Python

Q1. (Base Conversion)

You might be familiar with converting a given number in binary (base, $b = 2$) into its decimal equivalent. This question requires you to generalise it to any base b, and the program should be written in Python.

Given a string of a fractional number N_b in base b, convert it into its decimal equivalent N_D. You need to check if the input is a valid number and get rid of leading zeros from the input.

$-999999999 <= N_D <= 999999999, \; N_D \in \mathbb{R}$
$2 <= b <= 36$

Example-

For N_b = HELLO.PY	&	$b = 35$;	output, $N_D = 26137359.742041$.
For N_b = 00101.101	&	$b = 2$;	output, $N_D = 5.625$.
For N_b = GJDGXR	&	$b = 36$;	output, $N_D = 999999999$.
For N_b = -4G	&	$b = 17$;	output, $N_D = -84$.
For N_b = HELLO.PY	&	$b = 10$;	output, “Invalid Input”.

Important Note: You are **NOT** allowed to use built in functions like int() etc.

Q2. (Intro to ML)

In this question, we will code a 1-D linear regression problem. We will provide pseudo-code here, the students are required to transform it into a python code. Use numpy library for array/vector/matrices etc.
The data files are train.csv and test.csv.
File structure:
The structure of both files are similar. Train.csv has n_train = 10^4 rows and test.csv has n_test = 10^3 rows, each row corresponding to one data point. Each row has two values separated by comma. The first value is feature and second value is label of the data point.

Example-
If one row of train.txt is :
4,7
Then feature, x = 4 and label, y = 7.

Pseudo-code

Step-1:

- Read files train.csv
- Create vector - X_train (dim - n_train x 1) and vector - y_train (dim - n_train x 1)
- Add a column to X_train so that its dimension becomes n_train x 2. First column of X_train should be all 1 and 2nd column is the same as before adding extra column.

Example -

```
X_train = [
            2
            3
            4
           ]
 New X_train = [
                 1   2
                 1   3
                 1   4
                ]
```

Step-2:

Generate a 2-D vector w (dim: 2 x 1) initialised randomly with floating point numbers.

Step-3:

Plot y vs x using matplotlib where x is the feature and y is the label read from the file train.csv.

Consider x' = [1 x] (prepending 1 to x to generate 2-dimensional x-vector)

On the same figure plot the line w^T*x' vs x.

Your figure should have a dot corresponding to each datapoint (x,y) and a straight line on the plot corresponding to w^T*x'.

Step-4:

Set w_direct = (X_train^T * X_train)^(-1)* X_train^T*y_train

X_train is the n_train x 2 matrix defined earlier and y_train is the corresponding label vector.

Plot y vs x using matplotlib where x is the feature and y is the label read from the file train.csv.

Consider x' = [1 x] (prepending 1 to x to generate 2-dimensional x-vector)

On the same figure plot the line w_direct^T*x' vs x.

Your figure should have a dot corresponding to each datapoint (x,y) and a straight line on the plot corresponding to w_direct^T*x'.

Step-5: **(Training)**
w - 2-dim vector initialised earlier (step-2)
Loop: for nepoch = 1 to N (N is the number of pass through the data (~10), play with it to find
best fit)
Loop : for j = 1 to n_train
w ← w – eta*(w^T*x' - y)*x' (eta = 0.0001 students can change this value)
If j%100 == 0
Then plot y vs x as earlier and use current value of w to plot w^T*x' vs x

Step-6:
Finally redraw the plot as earlier with latest value of w.

Note:- Don't use test.csv for training

Step-7: **(Evaluation)**
- Read files test.csv
- Create vector - X_test (dim - n_test x 1) and vector - y_test (dim - n_test x 1)
- Add a column to X_test so that its dimension becomes n_test x 2. First column of X_test should be all 1 and 2nd column is the same as before adding extra column.

- Let y_pred1 = X_test*w (w is the final value after doing step 5)
- Calculate root mean squared error between y_pred1 and y_test.
- Let y_pred2 = X_test*w_direct
- Calculate root mean squared error between y_pred2 and y_test.

Derivation of Updates (optional reading)
In the loop of step 5 we did the following
w ← w – eta*(w^T*x' - y)*x'

The objective of the above step is to reduce the least squared error.
Let error = 0.5(w^T*x'-y)^2
So, derivative w.r.t. w gives: (w^T*x'-y)x'

We need to descend down the gradient to reach the minima, so we subtract eta times the above derivative from w to gradually reach minima. We set eta to small value because otherwise, w may overshoot the minima. w_direct can also be computed on the same line by setting derivative to zero (Google it!).