Beryl Sin

PeopleSoft ID: 4522433

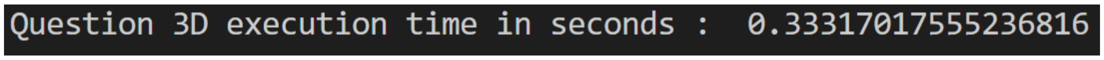
01/11/2024

Problem Set 1

1. Conceptual question: propose one new regression problem (one that we have not discussed in class) that you can solve with machine learning, and describe how you would go about solving it.
   Problem: Predict the number of passengers that take the subway in NYC
   Please give a short answer (text) to the following;
    a. What features (x) would you use?
        i. Fare price
        ii. Date
        iii. Time
        iv. Station name
    b. What would the labels (y) be?
        i. Number of passengers
    c. How would you collect data?
        i. Track the number of swipes/taps on the card readers from a station each day.
    d. Why might the problem turn out to be challenging?
        i. There may be other important factors outside of the features listed that may influence the number of passengers that take the subway
        ii. Not enough data may affect results

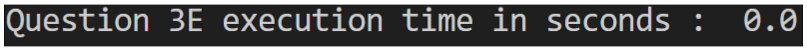
2. Repeat question 1 for a classification problem.
   Problem: Is this an apple?
   a. What features (x) would you use?
      i. Color
      ii. Size
      iii. Shape
      iv. Texture
   b. What would the labels (y) be?
      i. Apple
      ii. Not apple
   c. How would you collect data?
      i. Use an open-source fruit image dataset (here is a link to one: https://www.vicos.si/resources/fids30/)
   d. Why might the problem turn out to be challenging?
      i. Not enough important factors were considered
      ii. Outliers can impact results

3. Basic operations
    a. See ps1.py for code.
    b. See ps1.py for code.
    c. See ps1.py for code. See ps1-3-c-1.png and ps1-3-c-2.png for output images.
       The histogram for vector x does look like a Gaussian distribution.
       The histogram for vector z does look like a uniform distribution.
    d. See ps1.py for code.



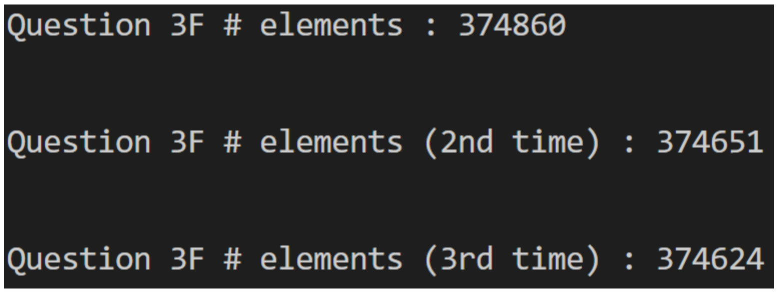
```
Question 3D execution time in seconds :  0.33317017555236816
```

    e. See ps1.py for code.



```
Question 3E execution time in seconds :  0.0
```

       Adding a constant to a long vector without using a loop is more efficient.
    f. See ps1.py for code.

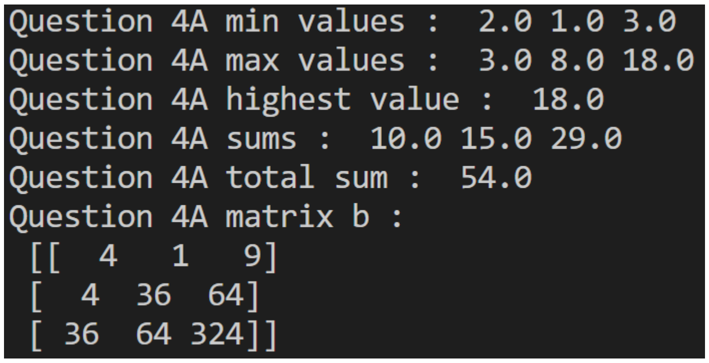


```
Question 3F # elements : 374860


Question 3F # elements (2nd time) : 374651


Question 3F # elements (3rd time) : 374624
```

       The number of retrieved elements are different each time the same lines of code is ran because the function (in the code, np.random.uniform) used to generate vector z uses random numbers from a normal distribution. These random numbers change every time the function is called, which affects vector y, which is determined by how many positive elements are less than 1.5 in vector z.

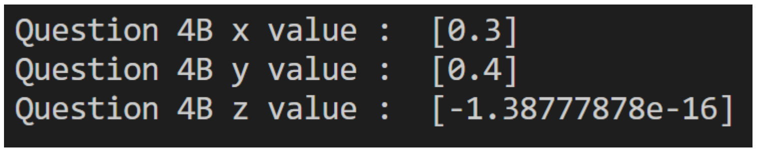
4. Linear Algebra

a. See ps1.py for code, results:



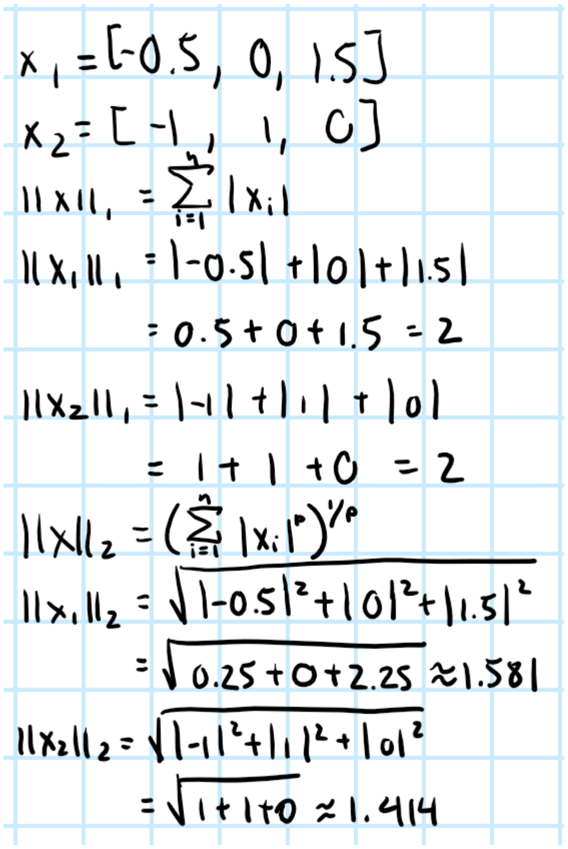
```
Question 4A min values :  2.0 1.0 3.0
Question 4A max values :  3.0 8.0 18.0
Question 4A highest value :  18.0
Question 4A sums :  10.0 15.0 29.0
Question 4A total sum :  54.0
Question 4A matrix b :
 [[  4   1   9]
 [  4  36  64]
 [ 36  64 324]]
```

b. See ps1.py for code, results:



```
Question 4B x value :  [0.3]
Question 4B y value :  [0.4]
Question 4B z value :  [-1.38777878e-16]
```

c. By hand:



$x_1 = [-0.5,\ 0,\ 1.5]$

$x_2 = [-1,\ 1,\ 0]$

$\|x\|_1 = \sum_{i=1}^{n} |x_i|$

$\|x_1\|_1 = |-0.5| + |0| + |1.5|$

$= 0.5 + 0 + 1.5 = 2$

$\|x_2\|_1 = |-1| + |1| + |0|$

$= 1 + 1 + 0 = 2$

$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$

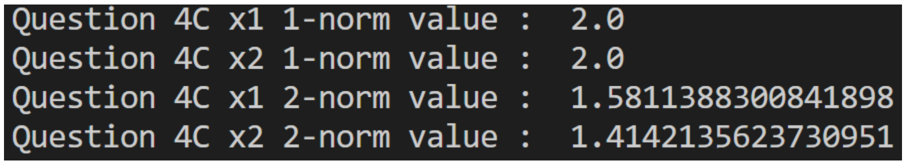
$\|x_1\|_2 = \sqrt{|-0.5|^2 + |0|^2 + |1.5|^2}$

$= \sqrt{0.25 + 0 + 2.25} \approx 1.581$

$\|x_2\|_2 = \sqrt{|-1|^2 + |1|^2 + |0|^2}$

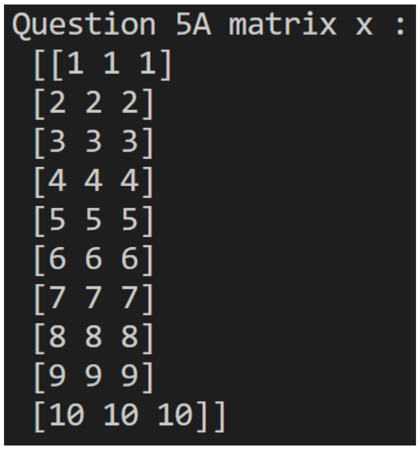
$= \sqrt{1 + 1 + 0} \approx 1.414$

See ps1.py for code, results using the norm function in python:



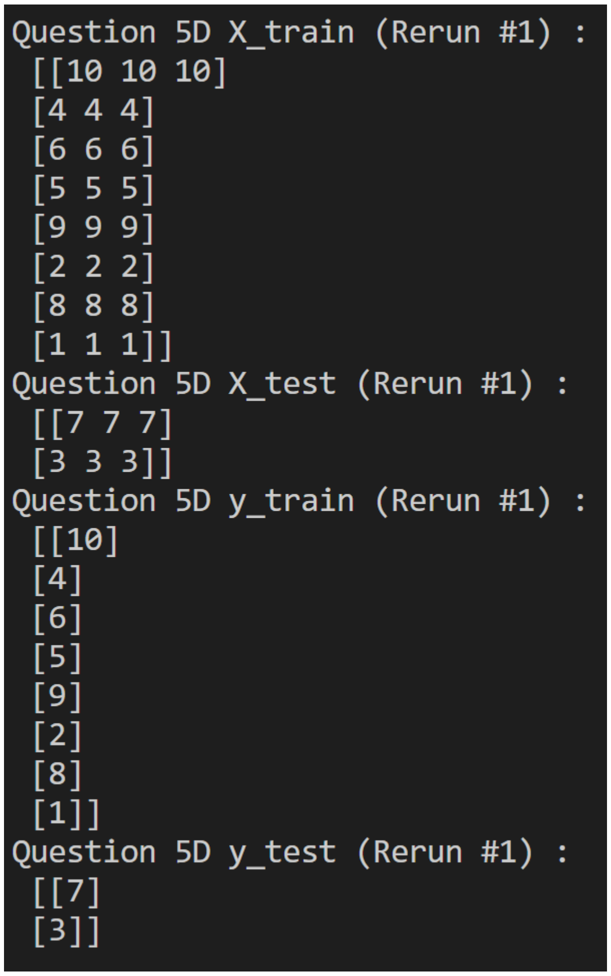
```
Question 4C x1 1-norm value :  2.0
Question 4C x2 1-norm value :  2.0
Question 4C x1 2-norm value :  1.5811388300841898
Question 4C x2 2-norm value :  1.4142135623730951
```

5. Splitting Data
   a. See ps1.py for code, results:

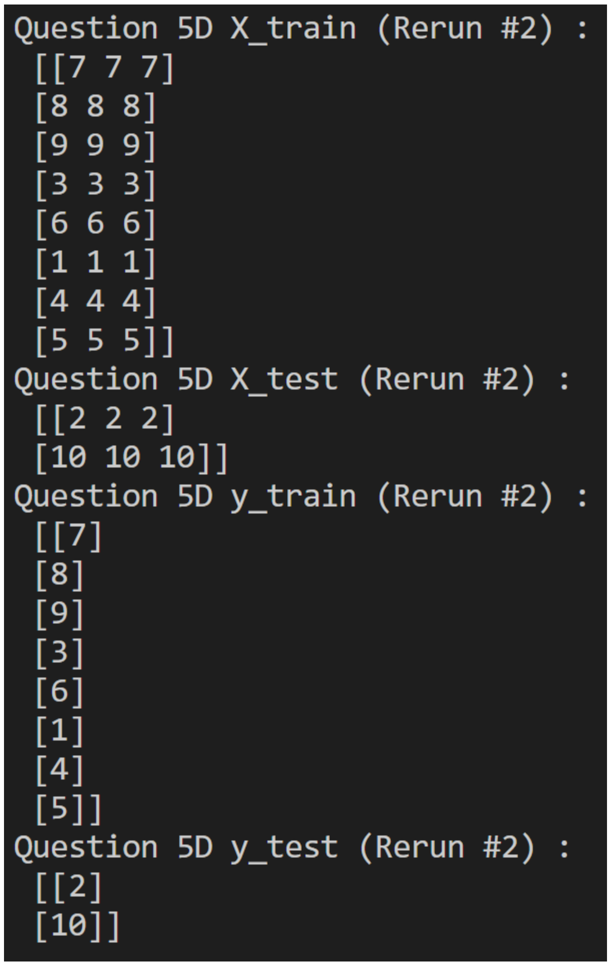


```
Question 5A matrix x :
 [[1 1 1]
 [2 2 2]
 [3 3 3]
 [4 4 4]
 [5 5 5]
 [6 6 6]
 [7 7 7]
 [8 8 8]
 [9 9 9]
 [10 10 10]]
```

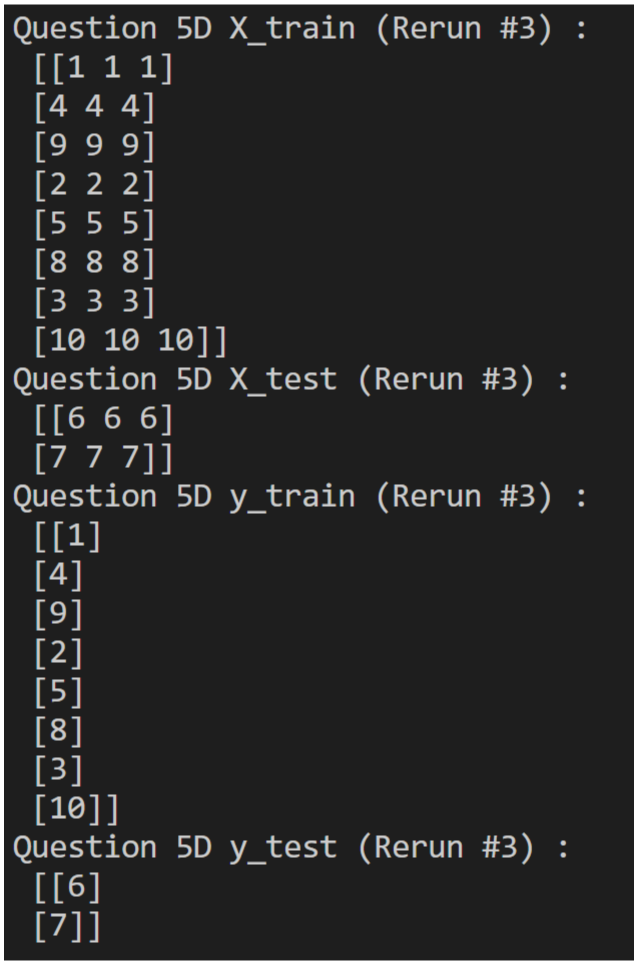
   b. See ps1.py for code.
   c. See ps1.py for code.
   d. See ps1.py for code, results:



```
Question 5D X_train (Rerun #1) :
 [[10 10 10]
 [4 4 4]
 [6 6 6]
 [5 5 5]
 [9 9 9]
 [2 2 2]
 [8 8 8]
 [1 1 1]]
Question 5D X_test (Rerun #1) :
 [[7 7 7]
 [3 3 3]]
Question 5D y_train (Rerun #1) :
 [[10]
 [4]
 [6]
 [5]
 [9]
 [2]
 [8]
 [1]]
Question 5D y_test (Rerun #1) :
 [[7]
 [3]]
```



```
Question 5D X_train (Rerun #2) :
 [[7 7 7]
 [8 8 8]
 [9 9 9]
 [3 3 3]
 [6 6 6]
 [1 1 1]
 [4 4 4]
 [5 5 5]]
Question 5D X_test (Rerun #2) :
 [[2 2 2]
 [10 10 10]]
Question 5D y_train (Rerun #2) :
 [[7]
 [8]
 [9]
 [3]
 [6]
 [1]
 [4]
 [5]]
Question 5D y_test (Rerun #2) :
 [[2]
 [10]]
```



```
Question 5D X_train (Rerun #3) :
 [[1 1 1]
 [4 4 4]
 [9 9 9]
 [2 2 2]
 [5 5 5]
 [8 8 8]
 [3 3 3]
 [10 10 10]]
Question 5D X_test (Rerun #3) :
 [[6 6 6]
 [7 7 7]]
Question 5D y_train (Rerun #3) :
 [[1]
 [4]
 [9]
 [2]
 [5]
 [8]
 [3]
 [10]]
Question 5D y_test (Rerun #3) :
 [[6]
 [7]]
```

I did not get the same submatrices each time I split X because the shuffle function used to partition the matrices mixes the row indices randomly so that the rows used for the training set/testing set differs every time the function is called.