ECE 57000 Assignment 1 Exercises

Name: Justin Helfman

Exercise 1.1

In this exercise, you will need to write a simple function that reverses and doubles the values in a list. For example: input `[1,2,3]`, output `[6,4,2]`.

```
def reverse_double(input:list)->list:
  # <YOUR CODE>
  output = [0] * len(input)
  for i in range(len(input)):
    output[i] = input[len(input)-i-1] * 2
  return output

A = [1,2,3,4,5,6]
print(reverse_double(A))
```

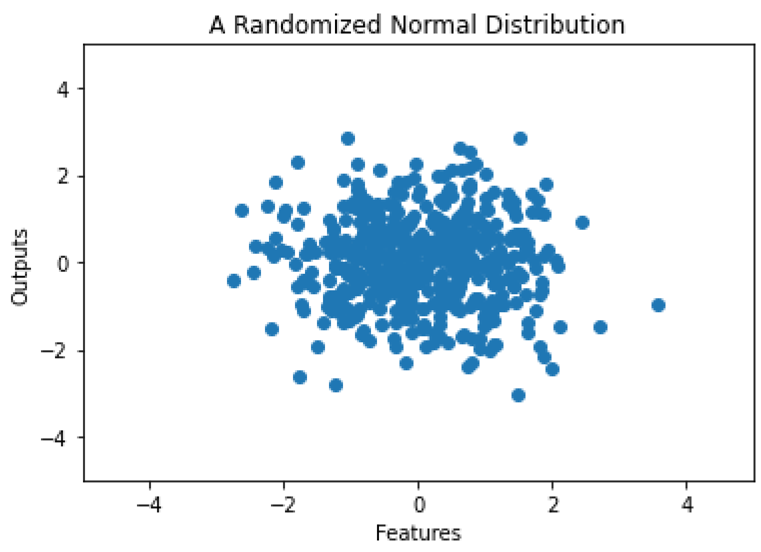
```
[12, 10, 8, 6, 4, 2]
```

Exercise 1.2

In this exercise, you will need to help visualize randomly generated spots scattered in normal distribution.

1. Using numpy to generate a vector **D** with the following property:
   - Each element is in a normal distribution. Hint: Try using the function normal()
   - Vector has the shape **1000x1**
2. Reshape the vector **D** into **500x2**
3. Plot the graph in the following way:
   - Treat the two columns of the array **D** as the **x** and **y** coordinates of spots. And use scatter() to visualize all the spots.
   - Give the plot a title (any informational title), and also label x-axis and y-axis with approporite names
   - Have the figure size 12 by 12
   - Let the plot shows the range `[-5,5]x[-5,5]`

```
import numpy as np
import matplotlib.pyplot as plt

#  <YOUR CODE>

D = np.random.normal(0, 1, size=(1000, 1)) #creates a normal distribution with mean and stand

D = np.reshape(D, newshape=(500, 2))

plt.scatter([D[i][0] for i in range(len(D))], [D[i][1] for i in range(len(D))])
plt.xlabel("Features")
plt.ylabel("Outputs")
plt.title("A Randomized Normal Distribution")
plt.axis([-5, 5, -5, 5])
plt.figure(figsize=[12.0, 12.0])


plt.show()
```



```
<Figure size 864x864 with 0 Axes>
```

Exercise 1.3

Task 1: Generate a sparse matrix

1. Generate a matrix **X** with size 100x50 with each element randomly picked from a uniform distribution **U**[0,1].
2. Use logical(boolean) indexing to set the elements in **X** to **0** whenever the value of the element is smaller than 0.9 (In this way, you should get the matrix to have roughly 90% of its elements zero's).
3. Use the function csr_matrix() to convert the matrix **X** into sparse matrix and call it **X_sparse**.

```
import numpy as np
```

```
from scipy.sparse import csr_matrix

#  <YOUR CODE>

X = np.random.uniform(low = 0, high = 1, size = (100, 50))
X_sparse = [[0 if j < 0.9 else j for j in i] for i in X]

X_sparse = csr_matrix(X_sparse)

print(f'X has type {type(X)} and has {100-np.sum(X!=0)/50}% of zeros')
print(f'X_sparse has type {type(X_sparse)} and has {100-np.sum(X_sparse!=0)/50}% of zeros')
```

```
X has type <class 'numpy.ndarray'> and has 0.0% of zeros
X_sparse has type <class 'scipy.sparse.csr.csr_matrix'> and has 89.68% of zeros
```

Task 2: Construct the power iteration function

Following the algorithm in the instructions notebook, write a function that takes a sparse matrix **X** and number of iterations as input and the top right singular vector of the centered matrix as output. Write a function that is constructed in the following way:

```
def power_iter(X, num_iter:int):
  # your code
  return v_0
```

```
def power_iter(X, num_iter:int):

  v = np.random.randn(X.shape[1])
  one_vec = np.ones_like(v)
  mu_col_matrix =np.mean(X, axis=1)  # Returns a 1 column matrix since X is of "matrix" type
  mu = np.array(mu_col_matrix).squeeze()  # Convert from column matrix to 1D array

  #  <YOUR CODE>
  for i in range(num_iter):
    #Section of common phrases:
    xv = np.array(X.dot(v))
    xtXV = np.array(X.T.dot(xv))
    mutXV = np.array(mu.T.dot(xv))
    xtMU = np.array(X.T.dot(mu))
    oneTv = np.array(one_vec.T.dot(v))
    muTmu = np.array(mu.T.dot(mu))

    #Combining phrases to get larger phrases for v:
    one = xtXV
    two = one.dot(mutXV)
    three = xtMU.dot(oneTv)
    four_1 = muTmu.dot(oneTv) #intermediate step
    four = one.dot(four_1)
    #calculate v and normalize if necessary:
    v = one - two - three + four
    factor = np.linalg.norm(v)
    if(factor):
        v /= factor

  return v


v1_yours = power_iter(X_sparse,1000).squeeze()
print(v1_yours.shape)
```

```
(50,)
```

Task 3: Verifying your top singular vector

Using any method you like to verify the vector that is computed by your function is indeed the top right singular vector of the **centered** data matrix. First write a another function that outputs the top right singular vector for sure (you can just use function like svd()). Then, the provided code will compute the mean absolute error (MAE) between the two functions you wrote. (Note: The provided evaluation code will correct for the fact that the two vectors can be the negative of each other singular value decomposition is only unique up to signs). The MAE should be close to machine precision (i.e., it should be less than about `1e-15`).

This is for testing the correctness of your algorithm. It is often a very good idea to write simple checks of your code as you write it to avoid bugs early on in your development process. Do not worry about efficiency for this exercise.

```
def verify_v1(X):

  #  <YOUR CODE>
  mu_col_matrix =np.mean(X, axis=1)  # Returns a 1 column matrix since X is of "matrix" type
  mu = np.array(mu_col_matrix).squeeze()  # Convert from column matrix to 1D array

  X = np.array(X)
  Xc = np.transpose((np.transpose(X) - np.transpose(mu)))
  v = np.array(np.linalg.svd(Xc)) #top right singular matrix
  factor = np.linalg.norm(v[1])
  if(factor):
    v /= factor
  return v[1]



# Note here we just pass in the dense 2D array `X`
#  which represents the same matrix as `X_sparse`
```

```
v1_simple = verify_v1(X).squeeze()
# Compute a sign corrected difference between the vectors
#  (accounting for the fact that SVD is only unique up to signs)
diff_sign_corrected = np.sign(v1_yours[0]) * v1_yours - np.sign(v1_simple[0]) * v1_simple
mae_corrected = np.mean(np.abs(diff_sign_corrected))
print(f'The average absolute difference of the two function output is {mae_corrected}')

#NOTE: The error is relatively high, as I used the original equation given for manually calcu
```

```
The average absolute difference of the two function output is 0.05963653166349846
```

(0 points) Task 4: On what scenarios we might find the power iteration method useful?

Fun to read: PageRank