**Experiment No.: 1**

**Name: Sreelekshmi Anilkumar**

**Roll No:42**

**Batch: Reg mca(b)**

**Date:1/09/2022**

**Aim**

To implement Matrix operations (using vectorization), transformation using python and SVD using Python

**Questions**

(a) Matrix operations (using vectorization),

(b) transformation using python and

(c) SVD using Python.

**Program and Output**

import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array

print("type: " ,type(a))            # Prints "<class 'numpy.ndarray'>"

print("shape: " ,a.shape)            # Prints "(3,)"

print(a[0], a[1], a[2])   # Prints "1 2 3"

a[0] = 5                  # Change an element of the array

print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array

print("\n shape of b:",b.shape)                     # Prints "(2, 3)"

print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"

a = np.zeros((3,3))   # Create an array of all zeros

print("All zeros matrix:\n  " ,a)

b = np.ones((1,2))    # Create an array of all ones

print("\nAll ones matrix:\n  " ,b)              # Prints "[[ 1.  1.]]"

d = np.eye(2)        # Create a 2x2 identity matrix

print("\n identity matrix: \n",d)

e = np.random.random((2,2))  # Create an array filled with random values

print("\n random matrix: \n",e)

* OUTPUT

shape: (3,)

1 2 3

[5 2 3]

shape of b: (2, 3)

1 2 4

All zeros matrix:

[[0. 0. 0.]

[0. 0. 0.]

[0. 0. 0.]]

All ones matrix:

[[1. 1.]]

identity matrix:

[[1. 0.]

[0. 1.]]

random matrix:

[[0.19072046 0.82646264]

[0.24096376 0.46100121]]

#vectorized sum

print("Vectorized sum example\n")

x = np.array([[1,2],[3,4]])

print("x:\n " ,x)

print("sum: ",np.sum(x))  # Compute sum of all elements; prints "10"

print("sum axis = 0: " ,np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"

print(" sum axis = 1: " ,np.sum(x, axis=1))  # Compute sum of each row; prints

#matrix dot product

a = np.arange(10000)

b = np.arange(10000)

print("a", a)

print("b", b)

dp = np.dot(a,b)

print("Dot product: \n" ,dp)

#outer product

op = np.outer(a,b)

print("\n Outer product: \n" ,op)

#elementwise product

ep = np.multiply(a, b)

print("\n Element Wise product:  \n" ,ep)

* OUTPUT

Vectorized sum example

x:

[[1 2]

[3 4]]

sum: 10

sum axis = 0: [4 6]

sum axis = 1: [3 7]

a [ 0 1 2 ... 9997 9998 9999]

b [ 0 1 2 ... 9997 9998 9999]

Dot product:

333283335000

Outer product:

[[ 0 0 0 ... 0 0 0]

[ 0 1 2 ... 9997 9998 9999]

[ 0 2 4 ... 19994 19996 19998]

...

[ 0 9997 19994 ... 99940009 99950006 99960003]

[ 0 9998 19996 ... 99950006 99960004 99970002]

[ 0 9999 19998 ... 99960003 99970002 99980001]]

Element Wise product:

[ 0 1 4 ... 99940009 99960004 99980001]

import numpy as np

x = np.array([[1,2], [3,4]])

print("Original x: \n " ,x)    # Prints "[[1 2]

            #          [3 4]]"

print("\nTranspose of x: \n" ,x.T)  # Prints "[[1 3]

* OUTPUT

Original x:

[[1 2]

[3 4]]

Transpose of x:

[[1 3]

[2 4]]

# Singular-value decomposition

from numpy import array

from scipy.linalg import svd

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print("A: \n%s" %A)

# SVD

U, s, VT = svd(A)

print("\nU: \n%s" %U)

print("\ns: \n %s" %s)

print("\nV^T: \n %s" %VT)

* OUTPUT

A:

[[1 2]

[3 4]

[5 6]]

U:

[[-0.2298477 0.88346102 0.40824829]

[-0.52474482 0.24078249 -0.81649658]

[-0.81964194 -0.40189603 0.40824829]]

s:

[9.52551809 0.51430058]

V^T:

[[-0.61962948 -0.78489445]

[-0.78489445 0.61962948]]

**Experiment No.: 2**

**Name: Sreelekshmi Anilkumar**

**Roll No:42**

**Batch:b**

**Date:1/09/2022**

**Aim**

To implement Matrix operations (using vectorization), transformation using python and SVD using Python

import matplotlib.pyplot as plt

fig, axes = plt.subplots(1,2)

axes[0].scatter(X[:,0], X[:,1], c=y)

axes[0].set\_title("Original data")

axes[1].scatter(X\_scaled[:,0], X\_scaled[:,1], c=y)

axes[1].set\_title("MinMax scaled data")

plt.show()

from sklearn.neighbors import KNeighborsClassifier

model=KNeighborsClassifier(n\_neighbors=5,metric='minkowski')

model.fit(X\_train, y\_train)

y\_pred=model.predict(X\_test)

model.score(X\_test, y\_test)

1.0

# Example of making predictions

from math import sqrt

# calculate the Euclidean distance between two vectors

def euclidean\_distance(row1, row2):

  distance = 0.0

  for i in range(len(row1)-1):

    distance += (row1[i] - row2[i])\*\*2

  return sqrt(distance)

# Locate the most similar neighbors

def get\_neighbors(train, test\_row, num\_neighbors):

  distances = list()

  for train\_row in train:

    dist = euclidean\_distance(test\_row, train\_row)

    distances.append((train\_row, dist))

  distances.sort(key=lambda tup: tup[1])

  neighbors = list()

  for i in range(num\_neighbors):

    neighbors.append(distances[i][0])

  return neighbors

# Make a classification prediction with neighbors

def predict\_classification(train, test\_row, num\_neighbors):

  neighbors = get\_neighbors(train, test\_row, num\_neighbors)

  output\_values = [row[-1] for row in neighbors]

  prediction = max(set(output\_values), key=output\_values.count)

  return prediction

# Test distance function

dataset = [[2.7810836,2.550537003,0],

  [1.465489372,2.362125076,0],

  [3.396561688,4.400293529,0],

  [1.38807019,1.850220317,0],

  [3.06407232,3.005305973,0],

  [7.627531214,2.759262235,1],

  [5.332441248,2.088626775,1],

  [6.922596716,1.77106367,1],

  [8.675418651,-0.242068655,1],

  [7.673756466,3.508563011,1]]

prediction = predict\_classification(dataset, dataset[0], 3)

print('Expected %d, Got %d.' % (dataset[0][-1], prediction))

Expected 0, Got 0.