NullColsRowSpace

February 6, 2020

1 Left Null Space and Column Space

Author : Satrya Budi Pratama

Equation of Null Space :

$\boldsymbol{A}x = 0$

Notation :

- Column Space : $C(\boldsymbol{A})$
- Null Space : $N(\boldsymbol{A}) = N(rref(\boldsymbol{A}))$

we can calculate the null space of a **A** with that equation $\begin{bmatrix} 1 & 2 & 1 & 1 & 5 \\ -2 & 4 & 0 & 4 & -2 \\ 1 & 2 & 2 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

1.1 Pseudocode

1. Generate an A(rows,cols), in this case we would solve 3x5 matrices $\begin{bmatrix} 1 & 2 & 1 & 1 & 5 \\ -2 & 4 & 0 & 4 & -2 \\ 1 & 2 & 2 & 4 & 9 \end{bmatrix}$
2. Do reduced row echelon form (rref)
   - Iterate over cols
   - Find the pivot by get the absolute maximum element on same cols
   - If pivot is zero, skip the cols and set to zero
   - If pivot index is not same as current rows then swap the rows of the pivot
   - Normalize current rows with the current pivot element
   - Eliminate above and below
   - if r same as number of rows then finish
3. Get basic variables, the variables corresponding to the pivots in row reduced echelon form are called **basic variables**.
4. Get Free variables, other variables are called **free variables**.
5. Get column space
6. Get rows space
7. Get null space by calculate the linear combination.

8. Testing the result of linear combination as x to the $y = \boldsymbol{A}x$, if $y = 0$ it means the algorithm is working

Reference : https://www.youtube.com/watch?v=Qy4KzVGpzkM

[1]:
```
import numpy as np
A = np.array([[1. , 2. , 1. , 1. , 5.],
            [-2. , 4. , 0. , 4. , -2.],
            [1. , 2. , 2. , 4. , 9.]])
```

[2]:
```
# get rref
def get_reduced_row_echelon_form(B):
    A = np.copy(B)
    zero_tol=1e-8
    rows, cols = A.shape
    r = 0
    # iterate over cols
    for c in range(cols):
        print('\n Step : {}'.format(c+1))
        print("----------------- \nNow at row {} and col {} with \n A: {}".
↪format(r,c,A))
        ## Find the pivot row by get the maximum element on same rows
        pivot = np.argmax (np.abs (A[r:rows,c])) + r
        #print(np.argmax (np.abs (A[r:rows,c])))
        #print(np.abs (A[r:rows,c]))
        #print(pivot)
        m = np.abs(A[pivot, c])
        print("Found pivot {} in row {}".format(m,pivot))
        if m <= zero_tol:
            ## Skip column c, making sure the approximately zero terms are
            ## actually zero.
            A[r:rows, c] = np.zeros(rows-r)
            print("All elements at and below ( {} , {}) : 0.. moving on..".
↪format(r,c))
        else:
            if pivot != r:
                ## Swap current row and pivot row
                A[[pivot, r], c:cols] = A[[r, pivot], c:cols]
                print("Swap row {} with row {},  \n A:{} : ".format(r,pivot,A))


            ## Normalize pivot row
            print('Normalize {} / {} '.format(A[r, c:cols],   A[r, c] ))
            A[r, c:cols] = A[r, c:cols] / A[r, c]; # dividing
            print('Currents Rows : {} '.format(A[r,c:cols]))

            print('---- elimination ----')
            ## Eliminate the current rows
```

```
            v = A[r, c:cols]
            ## Above (before row r):
            if r > 0:
                ridx_above = np.arange(r)
                out_product = np.outer(v, A[ridx_above, c]).T
                rows_above = A[ridx_above, c:cols]
                A[ridx_above, c:cols] = rows_above - out_product
                print('Eliminate Above : {} -  {}'.
↪format(rows_above,out_product))
                print('Cols above : {} '.format(A[ridx_above, c].T))
                print("Elimination above performed: \n A:{}".format(A))
            ## Below (after row r):
            if r < rows-1:
                ridx_below = np.arange(r+1,rows)
                out_product = np.outer(v,A[ridx_below, c]).T
                rows_below = A[ridx_below,c:cols]
                A[ridx_below, c:cols] = A[ridx_below, c:cols] - np.outer(v,␣
↪A[ridx_below, c]).T
                print('Eliminate Below: {} -  {}'.
↪format(rows_below,out_product))
                print('Cols Below : {} '.format(A[ridx_below, c].T))
                print("Elimination above performed: \n A:{}".format(A))

        r += 1 # increment rows

        ## Check if done
        if r == rows:
            print("Finished reduced row echo form..")
            break

    return A
```

[3]:
```
# invoke rref method
Aref = get_reduced_row_echelon_form(A)
```

```
 Step : 1
-----------------
Now at row 0 and col 0 with
 A: [[ 1.  2.  1.  1.  5.]
 [-2.  4.  0.  4. -2.]
 [ 1.  2.  2.  4.  9.]]
Found pivot 2.0 in row 1
Swap row 0 with row 1,
 A:[[-2.  4.  0.  4. -2.]
 [ 1.  2.  1.  1.  5.]
 [ 1.  2.  2.  4.  9.]] :
```

```
Normalize [-2.  4.  0.  4. -2.] / -2.0
Currents Rows : [ 1. -2. -0. -2.  1.]
---- elimination ----
Eliminate Below: [[1. 2. 1. 1. 5.]
 [1. 2. 2. 4. 9.]] -  [[ 1. -2. -0. -2.  1.]
 [ 1. -2. -0. -2.  1.]]
Cols Below : [0. 0.]
Elimination above performed:
 A:[[ 1. -2. -0. -2.  1.]
 [ 0.  4.  1.  3.  4.]
 [ 0.  4.  2.  6.  8.]]

 Step : 2
-----------------
Now at row 1 and col 1 with
 A: [[ 1. -2. -0. -2.  1.]
 [ 0.  4.  1.  3.  4.]
 [ 0.  4.  2.  6.  8.]]
Found pivot 4.0 in row 1
Normalize [4. 1. 3. 4.] / 4.0
Currents Rows : [1.   0.25 0.75 1.  ]
---- elimination ----
Eliminate Above : [[-2. -0. -2.  1.]] -  [[-2.  -0.5 -1.5 -2. ]]
Cols above : [0.]
Elimination above performed:
 A:[[ 1.    0.    0.5  -0.5   3.  ]
 [ 0.    1.    0.25  0.75  1.  ]
 [ 0.    4.    2.    6.    8.  ]]
Eliminate Below: [[4. 2. 6. 8.]] -  [[4. 1. 3. 4.]]
Cols Below : [0.]
Elimination above performed:
 A:[[ 1.    0.    0.5  -0.5   3.  ]
 [ 0.    1.    0.25  0.75  1.  ]
 [ 0.    0.    1.    3.    4.  ]]

 Step : 3
-----------------
Now at row 2 and col 2 with
 A: [[ 1.    0.    0.5  -0.5   3.  ]
 [ 0.    1.    0.25  0.75  1.  ]
 [ 0.    0.    1.    3.    4.  ]]
Found pivot 1.0 in row 2
Normalize [1. 3. 4.] / 1.0
Currents Rows : [1. 3. 4.]
---- elimination ----
Eliminate Above : [[ 0.5  -0.5   3.  ]
 [ 0.25  0.75  1.  ]] -  [[0.5  1.5  2.  ]
 [0.25 0.75 1.  ]]
```

```
Cols above : [0. 0.]
Elimination above performed:
 A:[[ 1.  0.  0. -2.  1.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  3.  4.]]
Finished reduced row echo form..
```

1.2 Get Basic Variable

[4]:
```python
# get basic variable
def get_basic(rref):
    list_var = []
    rows,cols = rref.shape
    r = 0
    # iterate over cols
    for c in range(cols):
        print('Element rows {} , cols {} : {}'.format(c,r,rref[r][c]))
        print(rref[r][c+1:cols])
        cols_left = rref[r][c+1:cols]
        #print(~cols_left.any(axis=1))
        #all_zero = np.all(cols_left==0)

        # check pivot contain value not zero
        if(rref[r][c] == 1):
            str_free_var = 'x_{}'.format(c+1)
            list_var.append([str_free_var, r , c]) # append name, rows , cols
            print('Found')

        r += 1

        # check r until rows
        if(r == rows):
            break
    return list_var
```

[5]:
```python
basic = get_basic(Aref)
basic
```

```
Element rows 0 , cols 0 : 1.0
[ 0.  0. -2.  1.]
Found
Element rows 1 , cols 1 : 1.0
[0. 0. 0.]
Found
Element rows 2 , cols 2 : 1.0
[3. 4.]
Found
```

[5]: [['x_1', 0, 0], ['x_2', 1, 1], ['x_3', 2, 2]]

1.3 Get Column Space

[6]:
```
# column space
def get_cols_space(A,basic):
    list_cols = []
    for i in range(len(basic)):
        cols_space = A[:,basic[i][2]] # get cols of free_var from A
        list_cols.append(cols_space)

    return list_cols
```

[7]:
```
get_cols_space(A,basic)
```

[7]: [array([1., -2., 1.]), array([2., 4., 2.]), array([1., 0., 2.])]

1.4 Get Rows Space

[8]:
```
# row space
def get_rows_space(rref,basic):
    list_rows = []
    for i in range(len(basic)):
        rows_space = rref[basic[i][1]]
        list_rows.append(rows_space)

    return list_rows
```

[9]:
```
get_rows_space(Aref,basic)
```

[9]:
```
[array([ 1.,  0.,  0., -2.,  1.]),
 array([0., 1., 0., 0., 0.]),
 array([0., 0., 1., 3., 4.])]
```

1.5 Get Free Variable

[10]:
```
def get_free_variable(A,basic):
    list_free = []
    subs = len(A[0,:]) - len(basic)
    print(subs)
    return subs
```

```
free = get_free_variable(A,basic)
```

2

1.6 Get Null Space

[11]:
```
def get_null_space(rref,free):
    rows, cols = rref.shape
    list_equation = []
    r = 0
    # iterate over cols to make list equation
    for c in range(cols):
        # make equation:
        cols_left = rref[r][c+1:cols] # get the rows after pivot
        #print(cols_left)
        #list_combination_linear.append(equation)
        nested_equation = []
        for j in range(len(cols_left)):
            if cols_left[j] != 0:
                nested_equation.append(-1 * cols_left[j]) # change positive to␣
↪minus because change of position
                #print(nested_equation)

        r += 1

        # check rows filled with zeros or not
        if(len(nested_equation) != 0):
            list_equation.append(nested_equation)
        else:
            list_equation.append([0.,0.])

        # stop
        if (rows == r):
            break

    # add free variable to list equation
    # [1 ,0],[0,1]
    free_var = np.eye(free)
    for val in free_var:
        list_equation.append(val)

    # [[2.0, -1.0], [0,0], [-3.0, -4.0],[1,0],[0,1]]
    # assume x4 = s , x5 = t
    # this means x1 = 2s - t, x2 = 0 , x3 = -3s - 4t
    # the null space is : (2,0,3,1,0), (-1,0,-4,0,1)
```

```python
    list_combination_linear = np.transpose(list_equation) #transpose

    return list_combination_linear
```

[12]:
```python
null_space = get_null_space(Aref,free)
null_space
```

[12]:
```
array([[ 2.,  0., -3.,  1.,  0.],
       [-1.,  0., -4.,  0.,  1.]])
```

1.7 Testing Null Space

[13]:
```python
# A*x = 0
A.dot(null_space[0])
```

[13]:
```
array([0., 0., 0.])
```

[14]:
```python
# A*x = 0
A.dot(null_space[1])
```

[14]:
```
array([0., 0., 0.])
```

[15]:
```python
for null in null_space:
    result = A.dot(null) # Ax=0
    print('\n A: {} \n x: {} \n Result : {}'.format(A, null,result))

    if(np.all(result) == 0):
        print("-- Proved --")
    else:
        print("-- Failed -- ")
```

```
 A: [[ 1.  2.  1.  1.  5.]
 [-2.  4.  0.  4. -2.]
 [ 1.  2.  2.  4.  9.]]
 x: [ 2.  0. -3.  1.  0.]
 Result : [0. 0. 0.]
-- Proved --

 A: [[ 1.  2.  1.  1.  5.]
 [-2.  4.  0.  4. -2.]
 [ 1.  2.  2.  4.  9.]]
 x: [-1.  0. -4.  0.  1.]
 Result : [0. 0. 0.]
-- Proved --
```