Python: Assignment 3

Monday, June 2 2014

Andrew Fox, Kyle Hundman, Sari Nahmad

Einsum

File myeinsum.py

Description

This function mirrors the original Einsum functionality in order to evaluate Einstein summations of numpy arrays.

Usage

```
>>> import myeinsum
>>> import numpy as np
```

Input data tables: for each create a numpy array

```
>>> dataTable = np.array(data table)
```

Call einsum function

```
>>> myeinsum.einsum(subscripts, operands)
```

Subscripts: Specifies the subscripts for summation. This string is a comma-separated list of subscript labels, where each label refers to a dimension of the corresponding operand.

Operands: Arrays for the operation.

Bayesian Network Examples

```
# Input data with numpy arrays
>>> a = np.array([ 0.9, 0.1])
>>> c = np.array([ 0.1,  0.2,  0.3,  0.4])
>>> cab = np.array([[[ 0.2 ,  0.4 ,  0.4 ],[ 0.33,  0.33,  0.34]],[[ 0.1 ,  0.5 ,  0.4 ],[ 0.3 ,  0.1 ,  0.6 ]],[[ 0.01,  0.01,  0.98],[ 0.2 ,  0.7 ,  0.1 ]],[[ 0.2 ,  0.1 ,  0.7 ],[ 0.9 ,  0.05,  0.05]]])
>>> b = np.array([0,0,1])

>>> myeinsum.einsum('A,C,CAB,B->ABC',a,c,cab,b)

[[[ 0.      0.      0.      0.    ]
  [ 0.      0.      0.      0.    ]
  [ 0.036   0.072   0.2646  0.252 ]]

 [[ 0.      0.      0.      0.    ]
  [ 0.      0.      0.      0.    ]
  [ 0.0034  0.012   0.003   0.002 ]]]
```

```
>>> myeinsum.einsum('A,C,CAB,B->A',a,c,cab,b)

[ 0.6246  0.0204]

>>> myeinsum.einsum('A,C,CAB,B->B',a,c,cab,b)

[ 0.     0.     0.645]
```

Numpy Documentation Examples

```
# Input data
>>> a = np.arange(60.).reshape(3,4,5)
>>> b = np.arange(24.).reshape(4,3,2)
>>> c = np.arange(25).reshape(5,5)

>>> myeinsum.einsum('IJK,JIL->KL', a, b)

[[ 4400.  4730.]
 [ 4532.  4874.]
 [ 4664.  5018.]
 [ 4796.  5162.]
 [ 4928.  5306.]]

>>> myeinsum.einsum('II', c)
60

>>> myeinsum.einsum('II->I', c)
 [0, 6, 12, 18, 24]
```

Assignment Examples

```
# input data
>>> i = np.array([[.08, .02], [.6,.4]])
>>> j = np.array([[.3, .7], [.23,.77]])

>>> myeinsum.einsum('12,23->13', i, j)

[[ 0.0286  0.0714]
 [ 0.272   0.728 ]]

# input data
>>> q = np.array([[[ 0.2 ,  0.4 ,  0.4 ],[ 0.33,  0.33,  0.34]],[[ 0.1 ,  0.5 ,  0.4 ],[ 0.3 ,  0.1 ,  0.6 ]],[[
0.01,  0.01,  0.98],[ 0.2 ,  0.7 ,  0.1 ]],[[ 0.2 ,  0.1 ,  0.7 ],[ 0.9 ,  0.05,  0.05]]])

>>> myeinsum.einsum('XYZ->',q)

8.0
```

Minimum Spanning Tree (MST)

File minspantree.py

Description

This code implements the Kruskal algorithm to find a minimum spanning tree from the user provided network.

Usage

```
>>> import minspantree
>>> import numpy as np
```

Create data by providing X and Y coordinates for each point

>>> inputName = np.array([[X_o,Y_o], [X_1, Y_1] … [X_n, Y_n]])

Print result

```
>>> minspantree.mst(inputName)
```

MST Example

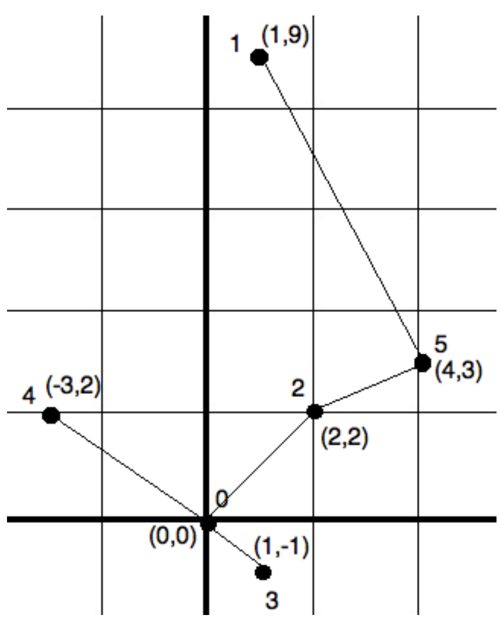
The node coordinates are input as (x, y) coordinates (see figure below). The order of the nodes as written in the array equation represents the *node numbers* used in the output. For example the node at point [0,0] becomes node 0, while [1,9] becomes node 1.

```
>>> input1 = np.array([[0,0],[1,9],[2,2],[1,-1],[-3,2],[4,3]])
>>> minspantree.mst(input1)
```

The output represents the *minimum distance* and *edges* of the MST in the order that they are added to the network. Therefore [0,3] represents that node 0 [0,0] is connected to node 3 [1,-1]. This was the first edge added to the MST followed by the edge between node 2 and node 5.

Result format: (Minimum distance, [edges between node A and B])

(16.792463872582434, [(0, 3), (2, 5), (0, 2), (0, 4), (1, 5)])



Appendix A: Einsum_Final.py

```
import re
import numpy as np

def combo(n):
   #create a list that is the size of the tuple
   r = [0] * len(n)
   #good for general loops with conditions appearing elsewhere in the loop
   while True:
      #need to return a tuple so it cannot be modified
      yield(tuple(r))
      #number of digits
      p = len(r) - 1
      #add 1 digit
      r[p] += 1
      #n[p] is the maximum, each value of the tuple
      while r[p] == n[p]:
         #return this digit to 0
         r[p] = 0
         #move on to the next tuple element to the left
         p -= 1
         #if hit the last element in the tuple, end
         if p < 0:
            return
         #increment the next digit by 1
         r[p] += 1

def einsum(procedure, *pots):

   #Use regular expression to check for repeats
   regexp = re.compile(r"(.)\1")
   match = re.search(regexp, procedure)

   if '->' in procedure:
      if match:
         #CONDITION 1: '->' and repeated letter (return diagonal)
         l = len(pots[0])
         diag = []
         for i in range(0,l):
            diag.append(pots[0][i][i])
         return(diag)
      else:
         #CONDITION 8: Tensor dot multiplication and dimension specific broadcasting
         halves = procedure.split("->")
         tables = halves[0]
         broadcastDims = []
         combinations = []
         uniqueTables = []
         originalTables = []
         flatTables = []
         flatDims = []
         uniqueDict = {}
         broadcastList = []
         nameAndDims = {}
         tables = tables.split(",")
         broadcast = str(halves[1])
         if not broadcast:
```

```
            sumOut = 0
            for pot in pots:
                for row in range(len(pot)):
                    for col in range(len(pot[0])):
                        sumOut = sumOut + pot[row][col]
            answer = sum(sumOut)
            return(answer)
        else:
            for letter in broadcast:
                broadcastList.append(letter)
            dims = []
            for x in range(0,len(pots)):
                nameAndDims[tables[x]] = pots[x].shape
                dims.append(pots[x].shape)

            for i in range(0,len(dims)):
                for h in tables[i]:
                    flatTables.append(h)
                    if h not in originalTables:
                        originalTables.append(h)
                for dim in dims[i]:
                    flatDims.append(dim)
            uniqueTables = sorted(originalTables)
            for pos in range(0,len(flatTables)):
                uniqueDict[flatTables[pos]] = flatDims[pos]
            for z in uniqueTables:
                if z in uniqueDict.keys():
                    combinations.append(uniqueDict.get(z))
            keepGoing = True
            while keepGoing == True:
                for letter in broadcast:
                    if letter in broadcastList:
                        broadcastDims.append(uniqueDict.get(letter))
                        keepGoing = False
                    else:
                        print('projection char not found')

            #COMPUTE
            combos = combo(combinations) #Generate all combos needed
            broadcastCombos = combo(broadcastDims) #Generate all combos (dimensions) needed in the broadcast
            out = np.zeros(broadcastDims)
            for bcomb in broadcastCombos:
                combos = combo(combinations) # generator gets exhausted on first loop through, need to reset it
                plug = 0
                for comb in combos:
                    skipCombo = False
                    if any(comb[uniqueTables.index(letter)] != bcomb[broadcastList.index(letter)] for letter in
broadcastList):
                        skipCombo = True
                    if skipCombo == False:
                        forMultiplying = []
                        for v in range(0,len(tables)):
                            indices = []
                            for char in tables[v]:
                                indices.append([comb[uniqueTables.index(char)]])
                            forMultiplying.append(float(pots[v][indices]))
                        value = 1
```

```
                for num in forMultiplying:
                    value *= num
                plug += value
            else:
                pass
        out[bcomb] = plug
    return(out)

#CONDITION 2: REPEATED LETTER BUT NO  '->' (sum diagonal)
else:
    l = len(pots[0])
    diag = []
    for i in range(0,l):
        diag.append(pots[0][i][i])
    return(sum(diag))
```

Appendix B: MST.py

```
import numpy as np

def mst(t):
    #initialize the dictionary
    #key = edge; value = weight/distance
    edges = {}
    totalSpan = 0
    tree = []
    vertices = {}
    #iterate over all points
    for index in range(0,len(t)):
        #for each point, iterate over other points
        #fill the vertex dictionary with key = vertex #, value = # ("color"); these start out equal but the values will change
        vertices["vertex " + str(index)] = index
        for dest in range(0,len(t)):
            #check that it's not the same point
            if index != dest:
                #create the edge, lowest index first
                if index > dest:
                    edge = (dest, index)
                else:
                    edge = (index, dest)
                #check if the edge has already been defined
                if edge not in edges:
                    #calculate distance, create the dictionary item
                    edges[edge] = np.linalg.norm(t[index]-t[dest])

    #sort by weight/distance
    for key, value in sorted(edges.items(), key=lambda x: x[1]):
        #call them origin and destination, even though they are undirected. o/d is arbitrary.
        origin = key[0]
        dest = key[1]
        #when edges connect previously disconnected sets of edges, we will change the vertices to all have the same 'color'
        vert_to_change = []
        #must be different 'colors' to prevent a cycle
        if vertices['vertex ' + str(origin)] != vertices['vertex ' + str(dest)]:
            #find all the vertices with the color you need to change
            vert_to_change = [vert_key for vert_key, vert_value in vertices.items() if vert_value == vertices['vertex ' + str(dest)]]
            #change the color to the origin color
            for vert_key in vert_to_change:
                vertices[vert_key] = vertices['vertex ' + str(origin)]
            #add the sedge to the tree
            tree.append(key)
            #calculate the total weight
            totalSpan += value
        #if all the colors are the same, the tree is finished
        if len(set(vertices.values())) == 1:
            return(totalSpan, tree)
```