Machine Learning Assignment № 6

Martin Huber, Roman Remme, Lucas-Raphael Müller June 2, 2017

Implementation

Listing 1: Implementation.

```
# some python stuff
from scipy import optimize
import math
import numpy as np
import copy
import heapq
import random

class Edge:
    def __init__(self, pointer, value):
        self.PointedNode = pointer
        self.Phi = value

class Node:
    def __init__(self, identifier, unary, beta = 0.01, startPoint = False, endPoint = False):
        self.Identifier = identifier
        self.Unary = unary
        self.StartPoint = startPoint
        self.EndPoint = endPoint
        self.beta = beta

        self.Joint = [ ]

    def constructEdges(self, Neighbour):
        def phiP(v1, v2, alpha = 0, beta = self.beta):
            if v1 == v2:
                return alpha
            else:
                return beta
        if Neighbour != None:
            if self.StartPoint == False and Neighbour.EndPoint == False:
                potential = phiP(self.Identifier[1], Neighbour.Identifier[1]) + Neighbour.Unary
                self.Joint.append(Edge(Neighbour.Identifier, potential))
```

```python


def dijkstra(graph, start, end):
    queue, seen = [(0, start, [])], set()
    while True:
        (cost, v, path) = heapq.heappop(queue)
        if v not in seen:
            path = path + [v]
            seen.add(v)
            if v == end:
                return cost, path
            for (next, c) in graph[v].items():
                heapq.heappush(queue, (cost + c, next, path))


noNodes = 20
labels = [0, 1]
noLabels = np.size(labels)
graph = np.ndarray((noNodes + 2, noLabels),dtype=np.object)
# get same random numbers every time


#betas
beta = np.array((0.01, 0.1, 0.2, 0.5, 1))
for bI, bEl in enumerate(beta):
    random.seed(0)

    # count number of already assigned nodes
    counter = 0
    # construct real nodes (not source nor target)
    for i in range(noNodes):
        for l in range(noLabels):
            # shuffle for first label
            if l == 0:
                rN = random.random()
            else:
                rN = 1 - graph[i,0].Unary

            graph[i,l] = Node([i,l], rN, bEl)
            counter += 1

    # reserve source and target
    i += 1
    graph[i, 0] = Node([i,0], math.nan, True, False)
    indexOfStart = i
    potential = graph[0,0].Unary
    graph[i, 0].Joint.append(Edge([0,0], potential))
```

```python
    potential = graph[0,1].Unary
    graph[i, 0].Joint.append(Edge([0,1], potential))


    i += 1
    counter += 1

    graph[i, 0] = Node([i,0], math.nan, False, True)
    indexOfEnd = i
    counter += 1

    # create linearlized reference object
    graphLin = np.ndarray((noNodes * noLabels + 2),dtype=np.object)

    counter = 0
    for i in range(graph.shape[0]):
        for j in range(graph.shape[1]):
            if graph[i,j] != None:
                graphLin[counter] = graph[i,j]
                graphLin[counter].LinIdentifier = counter
                counter += 1

    for i in range(graph.shape[0]):
        if graph[i, 0] != None and i < noNodes - 1:
            graph[i, 0].constructEdges(graph[i + 1, 0])
            graph[i, 0].constructEdges(graph[i + 1, 1])
        if graph[i, 1] != None and i < noNodes - 1:
            graph[i, 1].constructEdges(graph[i + 1, 0])
            graph[i, 1].constructEdges(graph[i + 1, 1])

    # hardcode for last one
    graph[noNodes - 1, 0].Joint.append(Edge([indexOfEnd,0], 0))
    graph[noNodes - 1, 1].Joint.append(Edge([indexOfEnd,0], 0))

    #graph[i for i in range(noNodes * noLabels + 2) if graph[i,0].StartPoint == True,0]

    graphDict = {}
    for i in range(graph.shape[0]):
        for j in range(graph.shape[1]):
            currNode = graph[i,j]
            nodeDict = {}
            if currNode != None:
                for k in range(np.size(currNode.Joint)):
                    currJoint = currNode.Joint[k]
                    nodeDict[(currJoint.PointedNode[0], currJoint.PointedNode[1])] = currJoint.Phi
```

```
                graphDict[(i, j)] = nodeDict

    cost, path = dijkstra(graphDict, (indexOfStart,0), (indexOfEnd,0))
    print('beta is ' + str(bEl))
    print(cost, path)
#print(graphDict)
#print('done')
```

Output

Should we assume

$$\phi_i(1) = 1 - exp(\phi_i(0)) \text{ ?}$$

Listing 2: Console output for unaries in 0...1 and beta as below

```
beta is 0.01
9.29924563302 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.1
9.92603071832 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.2
10.4938933736 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.5
11.2801892371 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 1.0
11.547815311895228 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 1), (21, 0)]

Process finished with exit code 0
```

Listing 3: Console output for unaries in -1...1 and beta as below

```
beta is -1.0
-13.4090386599 [(20, 0), (0, 1), (1, 0), (2, 1), (3, 0), (4, 1), (5, 0), (6, 1), (7, 0), (8, 1), (9, 0), (10, 1), (11, 0), (12, 1), (13, 0), (14, 1), (15, 0), (16, 0), (17, 1), (18, 0), (19, 1), (21, 0)]
beta is -0.1
```

```
-2.36150873396 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is -0.01
-1.64150873396 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.01
-1.48150873396 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.1
-0.761508733962 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 0), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.2
-0.147938563357 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 0), (7, 1), (8, 0), (9, 0), (10, 0), (11, 1), (12, 1), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 0.5
1.56037847414 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]
beta is 1.0
2.56037847414 [(20, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 1), (18, 1), (19, 1), (21, 0)]

Process finished with exit code 0
```