Linear and Quadratic Knapsack Problem

Large Scale Optimization for Data Science

Kale-ab Tessera, 1973752

In the Knapsack problem, our goal is to find the optimal combination of objects which are bound by a total weight W and which achieve the highest profit / value (v). The problem can be applied to various real life applications such as the resource allocation problem.

Two variants of the Knapsack problem are considered. The **Linear Knapsack Problem**, where items have individual weights and values, and the **Quadratic Knapsack Problem** where there is an additional term in the objective functions, which represents the extra profit gained from choosing a particular combination of items (in our case pairs).

Problem 1 - Linear Knapsack Problem

Linear Implementation of Knapsack Problem.

$$(v_i, w_i) = \{(2,7),(6,3),(8,3),(7,5),(3,4),(4,7),(6,5),(5,4),(10,15),(9,10),(8,17),(11,3),(12,6),(15,11),(6,6),(8,14),(13,4),(14,8),$$

$$(15,9),(16,10),(13,14),(14,17),(15,9),(26,24),(13,11),(9,17),(25,12),(26,14)\}$$

with **total capacity W = 30**

1.1 Greedy Implementation

In [1]:

```
import numpy as np

# Calculate Score - v_i/w_i
def calculateScore(v,w,n):
    score = np.array([])
    for i in range(n):
        score = np.append(score,round((v[i]/w[i]),3))
    scoreIndexs = np.argsort(-score)

    return scoreIndexs

def linearKnapSack(v,w,n,W,shouldPrint=False):
    sumItemsInKnapsack = 0

    #Sort items according to their score
    scoreIndexs =  calculateScore(v,w,n)
    indexofElementsInKnapsack = np.zeros(n).astype(int)

    # Add item with highest score, that is below capacity W.
    for i in range(n):
       indexOfLargestElement = scoreIndexs[i]
       if((sumItemsInKnapsack+w[indexOfLargestElement]) <= W ):
           indexofElementsInKnapsack[indexOfLargestElement] = 1
           sumItemsInKnapsack += w[indexOfLargestElement]
           if(shouldPrint):
               print("Adding (",v[indexOfLargestElement],",",w[indexOfLargestElement],
                     ") to the knapsack")

    if(shouldPrint):
        print("***************************")
        print("Arrays: ")
        print("v_i = {0}".format(v))
        print("w_i = {0}".format(w))
        print("x_i = {0}".format(indexofElementsInKnapsack))

    #Return index of set that yields maximum profit
    return indexofElementsInKnapsack
```

In [2]:

```
n = 28
v = np.array([2,6,8,7,3,4,6,5,10,9,8,11,12,15,6,8,13,14,15,16,13,14,15,26,13,9,25,26])
w = np.array([7,3,3,5,4,7,5,4,15,10,17,3,6,11,6,14,4,8,9,10,14,17,9,24,11,17,12,14])
W = 30

print("Problem 1 - Greedy Algorithm ")
indexofElementsInKnapsack = linearKnapSack(v,w,n,W)
print("Selected w:",w[indexofElementsInKnapsack==1])
print("Selected v:",v[indexofElementsInKnapsack==1])
print("Profit - linear knapsack:" , np.sum(v[indexofElementsInKnapsack==1]))
print("Weight - linear knapsack:" , np.sum(w[indexofElementsInKnapsack==1]))
```

```
Problem 1 - Greedy Algorithm
Selected w: [ 3  3  5  3  4 12]
Selected v: [ 6  8  7 11 13 25]
Profit - linear knapsack: 70
Weight - linear knapsack: 30
```

1.2 Polynomial Time Approximation Algorithm

In [3]:

```
#This function retrieves all subsets in a set recursively.
def findSubsets(allSubsets,subset,currentIndex,k):
    if(len(subset) == currentIndex):
        return allSubsets

    newSet = []
    for i in range(0,len(allSubsets)):
      if(len(allSubsets[i]) < k):
        newSet = allSubsets[i].copy()
        newSet.append(subset[currentIndex])
        allSubsets.append(newSet)

    findSubsets(allSubsets, subset, currentIndex+1,k)

def isLargerThanCapicity(chosenW,W):
      sumWeights = np.sum(chosenW)
      if(sumWeights > W):
          return True
      else:
          return False

# Find all subsets of a specific size
def findAllSubsetsSpecificSize(array, subsetSize,k):
    allSubsets = [[]]
    findSubsets(allSubsets,array,0,k)
    returnSet = []
    for sets in allSubsets:
        if(len(sets) == subsetSize):
            returnSet.append(sets)
    return returnSet

#Find all subsets larger than minSubsetSize
def findAllSubsetsWithinRange(array, minSubsetSize,k):
    allSubsets = [[]]
    findSubsets(allSubsets,array,0,k)
    returnSet = []
    for sets in allSubsets:
        if(len(sets) >= minSubsetSize):
            returnSet.append(sets)
    return returnSet

def PTAS(v,w,n,W,minSizeSubset,k,shouldPrint=False):
    currentW = 0
    sendW = 0

    allSubsets = [[]]
    indexOfSet = []
    for i in range(0,28):
        indexOfSet.append(i)

    #Step 1 - consider all subsets up to at most k items
    allSubsets = findAllSubsetsWithinRange(indexOfSet,minSizeSubset,k)
    allSubsetsNumpy = np.array(allSubsets)
    highestProfit = 0
    bestSetIndex = []
```

```
    bestLinearIndex = []


    #Step 2  - Pack F into Knapsack
    for sets in allSubsetsNumpy:
        chosenV = np.take(v,sets)
        chosenW = np.take(w,sets)
        if(isLargerThanCapicity(chosenW,W) == False):
            otherV = np.take(v,list(set(indexOfSet) - set(sets)))
            otherW = np.take(w,list(set(indexOfSet) - set(sets)))
            sumProfitPTAS = np.sum(chosenV)
            sumWeights = np.sum(chosenW)
            remainingW = W - sumWeights
            indexofElementsInKnapsack = linearKnapSack(otherV,otherW,len(otherV) ,remainingW)
            totalProfit = sumProfitPTAS + np.sum(otherV[indexofElementsInKnapsack==1])
            if(shouldPrint):
                print("Weight - PTAS: ",sumWeights)
                print("Profit - PTAS: ",sumProfitPTAS)
                print("Total Profit: ",totalProfit)
                print("**************************")

            if(totalProfit >highestProfit ):
                highestProfit = totalProfit
                bestSetIndex = sets
                bestLinearIndex = indexofElementsInKnapsack
                bestOtherV = otherV
                bestOtherW = otherW

    #Step 2 - Greedily fill the remaining capacity
    bestLinearIndexValues = np.where(bestLinearIndex==1)

    print("Best Indexs for PTAS:",bestSetIndex)
    print("Selected v - PTAS:",np.take(v,bestSetIndex))
    print("Selected v - Linear:",np.take(bestOtherV,bestLinearIndexValues)[0])

    bestValuedV = np.concatenate([np.take(v,bestSetIndex),
                np.take(bestOtherV,bestLinearIndexValues)[0]])
    bestValuedW = np.concatenate([np.take(w,bestSetIndex),
                np.take(bestOtherW,bestLinearIndexValues)[0]])

    #Step 3 - Return Highest valued combination set
    return bestValuedV, bestValuedW
```

In [4]:

```
print("Problem 1 - Polynomial Time Approximation Algorithm (PTAS) ")
minSizeSubset = 3
k = 10
bestValuedV, bestValuedW = PTAS(v,w,n,W,minSizeSubset,k)

print("Selected v - all:",bestValuedV)
print("Selected w - all",bestValuedW)
print("Total Weight - all:",np.sum(bestValuedW))
print("Best Profit - all:",np.sum(bestValuedV))
```

```
Problem 1 - Polynomial Time Approximation Algorithm (PTAS)
Best Indexs for PTAS: [2, 11, 17]
Selected v - PTAS: [ 8 11 14]
Selected v - Linear: [13 25]
Selected v - all: [ 8 11 14 13 25]
Selected w - all [ 3  3  8  4 12]
Total Weight - all: 30
Best Profit - all: 71
```

Problem 2 - Quadratic Knapsack Problem

Quadratic Implementation of Knapsack Probelm. Extra profit when choosing certain pairs are considered.

$$v_i = \{7, 6, 13, 16, 5, 10, 9, 23, 18, 12, 9, 22, 17, 32, 8\}$$

$$w_i = \{13, 14, 14, 15, 15, 9, 26, 24, 13, 11, 9, 12, 25, 12, 26\}, W = 50$$

and

$$p_{ij} = \{(12, 7, 6, 13, 8, 11, 7, 15, 23, 14, 15, 17, 9, 15, 15, 13, 10, 15, 9, 10, 8, 17, 11, 13, 12, 16, 15, 11, 16, 6, 8, 14, 13, 4, 14, 8, 15, 9$$
$$16, 10, 13,$$
$$14, 14, 17, 15, 14, 6, 24, 13, 4, 9, 7, 25, 12, 6, 6, 16, 10, 15, 14, 2, 13, 12, 16, 9, 11, 23, 10, 21, 818, 4, 13, 14, 14, 17, 15, 9, 16, 12, 3,$$
$$14, 27, 15,$$
$$16, 13, 14, 7, 17, 28, 5, 19, 6, 18, 13, 4, 13, 16, 11, 19, 13, 15, 12, 16)\}$$

Interpretation of the following steps:

- 4 . Add into knapsack the pair of items with the highest score, ensuring that the accumulated weight does not exceed the maximum capacity.
- 5 . Repeat steps 1 through 4 until pairs can no longer be added.

How they are interpreted:

- Once a sample has been selected and pairs have been ordered according to an efficiency function, the pair with the highest efficiency function is selected and it's weight is checked and compared with remaining capacity on the Knapsack, before attempting to add it to the knapsack.
- If the pair with highest efficiency cannot be added to the Knapsack, the data is resampled and the previous step is repeated.
- This resampling process happens for 1000 iterations, until attempting to use the linear greedy method.

In [20]:

```
import random

#Sampling k random items, from a set of Size N - 15 in our case.
#alreadySelectedNumbers ensures we do select elements already in the Knsapsack
def generateRandomNumbers(k,N,alreadySelectedNumbers ):
    #Retrieve unselected numbers/indexs from range(N)
    unselectedNumbers = np.delete(range(N),alreadySelectedNumbers)

    #Ensuring that sample size will not be larger than the amount of unselected numbers.
    if(len(unselectedNumbers) < k):
        numRandomNumbers = len(unselectedNumbers)
    else:
        numRandomNumbers = k

    #Sampling without replacement - ensuring numbers can't be reselected while sampling.
    listRandomIndexs = np.random.choice(unselectedNumbers, numRandomNumbers, replace=False)
    return listRandomIndexs

def sortByEfficiencyFunction(arrayOfPairs,p,w):
    allEfficiencies = np.array([])
    for pair in arrayOfPairs:
        P_ij = p[pair[0]][pair[1]]
        w_i = w[pair[0]]
        w_j = w[pair[1]]
        efficiency = P_ij/(w_i+w_j)
        allEfficiencies = np.append(allEfficiencies,efficiency)

    efficiencyIndex = np.argsort(-allEfficiencies)
    return efficiencyIndex

def isAnElementFromPairAlreadySelected(pairToAdd,pairsAlreadyChosen):
    indexFirstElementInPair = pairToAdd[0]
```

```
    indexSecondElementInPair = pairToAdd[1]


    if (indexFirstElementInPair in pairsAlreadyChosen
        or indexSecondElementInPair in pairsAlreadyChosen):
        return True
    else:
        return False

def QuadraticKnapsackAlgorithm(k,N,randomSeed,v,w,W,p):
    np.random.seed(randomSeed)
    indexChosenElements = []
    sizeOfArrayOfPairs = 2
    sumWeightElementsKnapsack = 0
    sumProfit = 0
    indexOfSet = []
    for i in range(0,15):
        indexOfSet.append(i)

    maxIterations = 1000
    iterationNumber = 0
    #Step 5 - Attempting the Quadratic Knapsack algorithm, while resampling for 1000 iterations
    while(iterationNumber< maxIterations):
        iterationNumber += 1
        # Step 1 - Sampling k items
        randomNumberIndex = generateRandomNumbers(k,N,indexChosenElements)

        # Step 2 - Obtaining a set of all pairs
        arrayOfPairs = findAllSubsetsSpecificSize(randomNumberIndex,sizeOfArrayOfPairs,k)

        # Step 3-  Sorting according to efficiency function p_ij / (w_i+ w_j)
        efficiencyIndexs = sortByEfficiencyFunction(arrayOfPairs,p,w)
        scoreIndex = np.argsort(efficiencyIndexs)

        # Adding elements to the Knapsack
        currentLargestPair = arrayOfPairs[efficiencyIndexs[0]]
        indexFirstElementInPair = currentLargestPair[0]
        indexSecondElementInPair = currentLargestPair[1]
        w_i = w[indexFirstElementInPair]
        w_j = w[indexSecondElementInPair]

        if((sumWeightElementsKnapsack+w_i+w_j) <= W):
            if(isAnElementFromPairAlreadySelected(currentLargestPair,indexChosenElements) == False):
                indexChosenElements.append(indexFirstElementInPair)
                indexChosenElements.append(indexSecondElementInPair)
                otherV = np.copy(v)
                otherW = np.copy(w)
                np.put(otherV,currentLargestPair,0)
                sumWeightElementsKnapsack += (w_i+w_j)
                sumProfit += (v[indexFirstElementInPair] + v[indexSecondElementInPair])
                print("Adding Pair - Index (",currentLargestPair,"),Value (",v[indexFirstElementInPair]
                      ,v[indexSecondElementInPair],") Weights(",w_i,w_j, ") to the knapsack")
                W = W - sumWeightElementsKnapsack
        else:
            continue

    selectedVQKP = np.take(v,indexChosenElements)
    selectedWQKP = np.take(w,indexChosenElements)
    print("Weight - QKP:" , np.sum(selectedWQKP))
    print("Profit - QKP (Not considering P_ij):" , np.sum(selectedVQKP))

    #Step 6 - Fill the remaining capacity using the linearGreedyApproach
    indexofElementsInKnapsack = linearKnapSack(otherV,otherW,len(otherV) ,W)

    linearIndex = np.where(indexofElementsInKnapsack == 1)[0]
    allIndexs = np.concatenate((linearIndex,indexChosenElements))

    allPairs = findAllSubsetsSpecificSize(allIndexs,sizeOfArrayOfPairs,N)

    singleItemProfit = np.sum(np.concatenate([np.take(v,indexChosenElements),
    otherV[indexofElementsInKnapsack==1]]))
    totalProfit = singleItemProfit

    for i in range(len(allPairs)):
      currentPair = allPairs[i]
      totalProfit += p[currentPair[0]][currentPair[1]]

    print("Weight - linear knapsack:" , np.sum(otherW[indexofElementsInKnapsack==1]))
    print("Profit - linear knapsack:" , np.sum(otherV[indexofElementsInKnapsack==1]))

    print("Selected v - QKP:",selectedVQKP)
```

```
    print("Selected v - Linear:",otherV[indexofElementsInKnapsack==1])

    print("Selected w - QKP:",selectedWQKP)
    print("Selected w - Linear:",otherW[indexofElementsInKnapsack==1])

    print("Selected v - all:",np.concatenate([np.take(v,indexChosenElements),otherV[indexofElementsInKnapsac
k==1]]))

    print("Selected w - all: ",np.concatenate([np.take(w,indexChosenElements),
    otherW[indexofElementsInKnapsack ==1]]))
    print("Total Weight - all: ",np.sum(np.concatenate([np.take(w,indexChosenElements),
    otherW[indexofElementsInKnapsack ==1]])))
    print("Total Profit - Single Values:",singleItemProfit)
    print("Total Profit - (Including P_ij):",totalProfit)
```

In [21]:

```
print("Problem 2 - Quadratic Knapsack Problem ")
v = np.array([7, 6, 13,16, 5, 10, 9, 23, 18, 12, 9, 22, 17, 32, 8])
w = np.array([13, 14, 14, 15, 15, 9, 26, 24, 13, 11, 9, 12, 25, 12, 26])
W = 50
p = np.array([
7       ,12     ,7      ,6      ,13     ,8      ,11     ,7      ,15     ,23     ,14     ,15     ,17     ,9
,15,
12      ,6      ,15     ,13     ,10     ,15     ,9      ,10     ,8      ,17     ,11     ,13     ,12     ,16
,15,
7       ,15     ,13     ,11     ,16     ,6      ,8      ,14     ,13     ,4      ,14     ,8      ,15     ,9
,16,
6       ,13     ,11     ,16     ,10     ,13     ,14     ,14     ,17     ,15     ,14     ,6      ,24     ,13
, 4,
13      ,10     ,16     ,10     ,5      ,9      ,7      ,25     ,12     ,6      ,6      ,16     ,10     ,15
,14,
8       ,15     ,6      ,13     ,9      ,10     ,2      ,13     ,12     ,16     ,9      ,11     ,23     ,10
,21,
11      ,9      ,8      ,14     ,7      ,2      ,9      ,8      ,18     ,4      ,13     ,14     ,14     ,17
,15,
7       ,10     ,14     ,14     ,25     ,13     ,8      ,23     ,9      ,16     ,12     ,3      ,14     ,14
,27,
15      ,8      ,13     ,17     ,12     ,12     ,18     ,9      ,18     ,15     ,16     ,13     ,14     ,7
,17,
23      ,17     ,4      ,15     ,6      ,16     ,4      ,16     ,15     ,12     ,28     ,5      ,19     ,6
,18,
14      ,11     ,14     ,14     ,6      ,9      ,13     ,12     ,16     ,28     ,9      ,13     ,4      ,13
,16,
15      ,13     ,8      ,6      ,16     ,11     ,14     ,3      ,13     ,5      ,13     ,22     ,11     ,19
,13,
17      ,12     ,15     ,24     ,10     ,23     ,14     ,14     ,14     ,19     ,4      ,11     ,17     ,15
,12,
9       ,16     ,9      ,13     ,15     ,10     ,17     ,14     ,7      ,6      ,13     ,19     ,15     ,32
,16,
15      ,15     ,16     ,4      ,14     ,21     ,15     ,27     ,17     ,18     ,16     ,13     ,12     ,16
, 8
    ])
p = p.reshape(15,15)
k = 7
N = 15
randomSeed = 7

QuadraticKnapsackAlgorithm(k,N,randomSeed,v,w,W,p)
```

```
Problem 2 - Quadratic Knapsack Problem
Adding Pair - Index ( [8, 10] ),Value ( 18 9 ) Weights( 13 9 ) to the knapsack
Weight - QKP: 22
Profit - QKP (Not considering P_ij): 27
Weight - linear knapsack: 24
Profit - linear knapsack: 54
Selected v - QKP: [18  9]
Selected v - Linear: [22 32]
Selected w - QKP: [13  9]
Selected w - Linear: [12 12]
Selected v - all: [18  9 22 32]
Selected w - all:  [13  9 12 12]
Total Weight - all:  46
Total Profit - Single Values: 81
Total Profit - (Including P_ij): 162
```