Greedy Approach for 0-1 Multi-Constraint Knapsack Problem

Omer Bolukbasi

Table of Contents

1- Greedy Solution for 0-1 Multi-Constraint Knapsack Problem

In the greedy solution for 1 constraint knapsack problem, v/w is calculated for each item. And the items are sorted in knapsacks based on this v/w values. The single knapsack is filled with the items that have highest v/w values. Higher v/w values are considered as high valued but low cost items (1). So, initailly the items that have highest values but lowest weights are consumed with this method.

In this homework we reduce the multi constraint knapsack problem to a single constraint knapsack problem by calculating a single v/w value which is produced from multiple weights.

In order to reduce the multiple constraints into 1 constraint, first we normalize the weights with the following formula:

Mean Normalization: normalized_df = mean(series))/std(series)

We also test the Min-Max Normalization and Log Normalization but the test results are not as good as mean normalization. So, mean normalization is used in this implementation.

2- Algorithm Implementation:

First, all inputs are taken into a Pandas DataFrame as seperate columns. As stated above, mean normalization is calculated for each constraints. Then, for each item the normalized constraints are added, and the sum is divided with the values of items.

df["v/w(normalized)"] = df["values"] / df["sum_of_normalized_constraint"]

	0	1	values	normalized_0	normalized_1	sum_of_normalized_constraint	v/w(normalized)
6	30	35	4650	0.189467	0.015058	0.204525	22735.619959
7	0	73	30800	0.747895	1.053072	1.800967	17101.927642
11	25	40	4225	0.282538	0.125485	0.408023	10354.803186
22	85	50	11748	0.834318	0.406572	1.240890	9467.397147
4	65	80	14148	0.462033	1.249833	1.711865	8264.668448
13	0	10	11880	0.747895	0.717775	1.465670	8105.507540
20	165	60	24355	2.323460	0.687659	3.011118	8088.356591
0	45	30	1898	0.089747	0.155602	0.245349	7735.918317
2	85	125	22507	0.834318	2.514724	3.349042	6720.429965
23	0	36	4550	0.747895	0.013050	0.760945	5979.405911
25	0	40	3720	0.747895	0.125485	0.873380	4259.314831

Next, the items in the table are sorted based on normalized v/w. And then we fill in the knapsacks from the sorted items. The highst v/w valued item gets the highest rate. And initially, knapsacks are filled with the highest valued items up to knapsack's capacity. So, accumulated knapsacks which filtered up to their capacities are as follows:

	0	1	values	normalized_0	normalized_1	sum_of_normalized_constraint	v/w(normalized)	knapsack_accumulation_0	knapsack_accumulation_1
7	0	73	30800	0.747895	1.053072	1.800967	17101.927642	30.0	108.0
11	25	40	4225	0.282538	0.125485	0.408023	10354.803186	55.0	148.0
22	85	50	11748	0.834318	0.406572	1.240890	9467.397147	140.0	198.0
4	65	80	14148	0.462033	1.249833	1.711865	8264.668448	205.0	278.0
13	0	10	11880	0.747895	0.717775	1.465670	8105.507540	205.0	288.0
20	165	60	24355	2.323460	0.687659	3.011118	8088.356591	370.0	348.0
0	45	30	1898	0.089747	0.155602	0.245349	7735.918317	415.0	378.0
2	85	125	22507	0.834318	2.514724	3.349042	6720.429965	500.0	503.0
23	0	36	4550	0.747895	0.013050	0.760945	5979.405911	500.0	539.0
25	0	40	3720	0.747895	0.125485	0.873380	4259.314831	500.0	579.0
9	0	15	4975	0.747895	0.577232	1.325127	3754.358091	500.0	594.0

But still some empty places are remained in the knapsacks, which can not fill with the next highest rated (v/w value) item. Because if this next item is put in the knapsack the capacity will be exceeded. So, the remaining capacities are being filled with the items which does not overflow the knapsack and also again have the highest rate if there exist any in the remained item list.

Finally, the items put in knapsacks are signed as ‘1’ and the rest is signed as ‘0’ with the final table.

	item_id	0	1	values	normalized_0	normalized_1	sum_of_normalized_constraint	v/w(normalized)	items_in_knapsack
0	0	45	30	1898	0.089747	0.155602	0.245349	7735.918317	1
1	1	0	20	440	0.747895	0.436689	1.184583	371.438655	0
2	2	85	125	22507	0.834318	2.514724	3.349042	6720.429965	1
3	3	150	5	270	2.044246	0.858319	2.902564	93.021194	0
4	4	65	80	14148	0.462033	1.249833	1.711865	8264.668448	1
5	5	95	25	3100	1.020461	0.296145	1.316606	2354.539115	0

The list of items is stored as an output file.

3- References:

1- A. Levitin, *Introduction to the Design & Analysis 3rd Edition* (Greedy Algorithms for the Knapsack Problem, Page 454, 2012)